СТАНДАРТ ST.90

РЕКОМЕНДАЦИИ ПО ОБРАБОТКЕ И ОБМЕНУ ДАННЫМИ ОБ ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ С ИСПОЛЬЗОВАНИЕМ ВЕБ-ИНТЕРФЕЙСОВ API (ИНТЕРФЕЙСОВ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ)

Версия 1.1

Пересмотр одобрен Комитетом по Стандартам ВОИС (КСВ) на своей десятой сессии 25 ноября 2022

ОГЛАВЛЕНИЕ

СТАНДАРТ ST.90

РЕКОМЕНДАЦИИ ПО ОБРАБОТКЕ И ОБМЕНУ ДАННЫМИ ОБ ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ С ИСПОЛЬЗОВАНИЕМ ВЕБ-ИНТЕРФЕЙСОВ API (ИНТЕРФЕЙСОВ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ)

Версия 1.1

Пересмотр одобрен Комитетом по Стандартам ВОИС (КСВ)
на своей десятой сессии 25 ноября 2022

ВВЕДЕНИЕ

1. Настоящий стандарт содержит рекомендации по интерфейсам прикладного программирования (API) для облегчения обработки и обмена данными об интеллектуальной собственности (ИС) согласованным способом через Интернет.

2. Настоящий стандарт предназначен чтобы:

- обеспечить согласованность путем установления единых принципов проектирования веб-сервисов;
- улучшить взаимодействие данных между партнерами по веб-сервисам;
- поощрять возможность повторного использования благодаря унифицированному дизайну;
- способствовать гибкости именования данных в бизнес-подразделениях посредством четко определенной политики пространства имен в связанных XML-ресурсах;
- способствовать безопасному обмену информацией;
- предлагать соответствующие внутренние бизнес-процессы в качестве услуг с добавленной стоимостью, которые могут быть использованы другими организациями; и
- интегрировать свои внутренние бизнес-процессы и динамически связывать их с бизнес-партнерами.

ОПРЕДЕЛЕНИЯ И ТЕРМИНОЛОГИЯ

3. Для целей настоящего стандарта выражения:

- "Протокол передачи гипертекста (HTTP)" означает прикладной протокол для распределенных, совместных и гипермедийных информационных систем. HTTP является основой передачи данных для Всемирной паутины (World Wide Web). HTTP функционирует как протокол "запрос-ответ" в сервис-ориентированной модели вычислений;
- "Интерфейсы прикладного программирования" (API) означают программные компоненты, которые обеспечивают многократно используемый интерфейс между различными приложениями, которые могут легко взаимодействовать для обмена данными;
- "Representational State Transfer (REST)" описывает набор архитектурных принципов, с помощью которых данные могут передаваться через стандартизированный интерфейс, т.е. HTTP. REST не содержит дополнительного уровня обмена сообщениями и

фокусируется на правилах проектирования для создания сервисов без статических данных;

- "Простой протокол доступа к объектам (SOAP)" означает протокол для отправки и получения сообщений между приложениями, без столкновений с проблемами совместимости. SOAP определяет стандартный протокол связи (набор правил) спецификации для обмена сообщениями на основе XML. SOAP использует различные транспортные протоколы, такие как HTTP и SMTP. Стандартный протокол HTTP облегчает туннелирование модели SOAP через брандмауэры и прокси-серверы без каких-либо изменений в протоколе SOAP;
- "Веб-сервис" означает метод связи между двумя приложениями или электронными машинами через Всемирную паутину (WWW), веб-сервисы бывают двух видов: REST и SOAP;
- "RESTful Web API" означает набор веб-сервисов, основанных на архитектурной парадигме REST и обычно использующих JSON или XML для передачи данных;
- "SOAP Web API" означает набор веб-сервисов SOAP, основанных на SOAP и предусматривающих использование XML в качестве формата полезных данных;
- "Язык описания веб-служб (Web Services Description Language - WSDL)" означает стандарт W3C, который используется с протоколом SOAP для предоставления описания веб-службы. В него входят методы, которые использует веб-служба, параметры, которые она принимает, средства поиска веб-служб и т.д.;
- RESTful API Modelling Language (RAML) относится к языку, который позволяет разработчикам предоставлять спецификацию своего API;
- Open API Specification (OAS) относится к языку, который позволяет разработчикам предоставлять спецификацию своего API;
- "Сервисный контракт (Service contract)" (или контракт веб-сервиса) означает документ, который показывает, как сервис раскрывает другим программам свои возможности в виде функций и ресурсов, предлагаемых в качестве опубликованного API сервиса; термин "документация REST API" взаимозаменяемо используется для обозначения сервисного контракта для RESTful Web API;
- "Поставщик услуг (Service provider)" означает исполняемое программное обеспечение веб-службы;
- "Потребитель услуг (Service consumer) " означает роль среды выполнения, которую берет на себя программа, когда она обращается к службе и вызывает ее. Точнее, когда программа посылает сообщение в сервисную функцию, прописанную в сервисном контракте. После получения запроса служба начинает обработку и может вернуть или не вернуть потребителю услуги соответствующее ответное сообщение;
- "Camelcase" - это либо lowerCamelCase (например, applicantName), либо UpperCamelCase (например, ApplicantName) соглашение об именовании;
- Kebab-case - это одна из конвенций именования, в которой все слова пишутся в нижнем регистре с дефисом "-", разделяющим слова, например, a-b-c;
- «Открытые стандарты» означают общедоступные стандарты, которые разрабатываются (или утверждаются) и поддерживаются в процессе сотрудничества и достижения консенсуса. «Открытые стандарты» облегчают взаимодействие и обмен данными между различными сервисными продуктами и предназначены для широкого внедрения;
- Унифицированный идентификатор ресурса (URI) идентифицирует ресурс, а универсальный указатель ресурса (URL) - это подмножество URI, включающее сетевое местоположение;

– "Метка сущности (ETag)" означает непрозрачный идентификатор, присваиваемый веб-сервером конкретной версии ресурса, найденного по URL. Если представление ресурса на этом URL когда-либо меняется, присваивается новый и другой ETag. ETag можно быстро сравнить, чтобы определить, являются ли два представления ресурса одинаковыми;
– "Реестр услуг" означает сетевой каталог, содержащий доступные услуги;
– "RMM" означает Модель зрелости Ричардсона - показатель зрелости REST API по шкале от 0 до 3; и
– "Семантическое версионирование" означает схему версионирования, в которой версия идентифицируется номером версии MAJOR.MINOR.PATCH, где:

    • MAJOR версия — это когда вносятся несовместимые изменения в API,
    • MINOR версия — это когда добавляется функциональность обратно совместимым образом и
    • Версия PATCH - это когда вносятся обратно совместимые исправления.

4. С точки зрения соответствия, в правилах оформления, следующие ключевые слова следует интерпретировать так же, как это определено в п. 8 документа ВОИС ST.96[1], а именно:

– ОБЯЗАН (MUST): эквивалент "ТРЕБУЕТСЯ (REQUIRED)" или "ДОЛЖЕН (SHALL)", означает, что определение является абсолютным требованием спецификации;

– ОБЯЗАН НЕ/НЕТ (MUST NOT): эквивалентно "НЕ ДОЛЖЕН (SHALL NOT)", означает, что определение устанавливает абсолютный запрет спецификацией;

– СЛЕДУЕТ (SHOULD): эквивалентно "РЕКОМЕНДУЕТСЯ (RECOMMENDED)", означает, что могут существовать веские причины для игнорирования данного пункта, но последствия этого должны быть полностью рассмотрены;

– НЕ СЛЕДУЕТ: эквивалентно "НЕ РЕКОМЕНДУЕТСЯ", означает, что могут существовать веские причины, по которым такое поведение может быть приемлемым или даже полезным, но последствия такого поведения должны быть тщательно рассмотрены; и

– МОЖЕТ (MAY): эквивалентно "ОПЦИОНАЛЬНО", означает, что данный элемент действительно является необязательным и предоставляется только как один вариант из многих.

ОБОЗНАЧЕНИЯ

Общие обозначения

5. В данном документе используются следующие обозначения:

– <>: Указывает на условное обозначение описательного термина, который при реализации будет заменен конкретным значением;

[1] Пожалуйста, обратитесь к главе "Ссылки»

– " ": Указывает, что текст, заключенный в кавычки, должен быть использован дословно в реализации;
– { }: Указывает, что элементы являются необязательными в реализации; и
– `Шрифт Courier New`: Указывает на ключевые слова или исходный код.

6. URL-адреса, приведенные в настоящем Стандарте, предназначены только для примера и не являются реальными.

Идентификаторы правил

7. Все правила проектирования являются нормативными. Правила проектирования обозначаются с помощью префикса [XX-nn] или [XXY-nn].

(a) Значение "XX" является префиксом для классификации типа правила следующим образом:

– WS для правил проектирования SOAP Web API;
– RS для правил проектирования RESTful Web API; и
– CS для правил проектирования как SOAP, так и RESTful WEB API.

(b) Значение "Y" используется только для правил проектирования RESTful и обеспечивает дополнительную детализацию типа ответа, к которому относится правило:

– "G" означает, что это общее правило для ответов в формате JSON и XML;
– "J" указывает на то, что это ответ в формате JSON; и
– "X" означает, что это ответ в формате XML.

(c) Значение "nn" указывает на следующий доступный номер в последовательности конкретного типа правила. Номер не отражает позицию правила, в частности, для нового правила. Новое правило будет помещено в соответствующий контекст. Например, идентификатор правила [WS-4] идентифицирует четвертое правило проектирования SOAP Web API. Правило [WS-4] может быть помещено между правилами [WS-10] и [WS-11], а не после [WS-3], если это наиболее подходящее место для данного правила.

(d) В случае удаления правила его идентификатор будет сохранен, а текст правила заменен на "Удалено".

ОБЛАСТЬ ПРИМЕНЕНИЯ

8. Настоящий стандарт предназначен для Ведомств Интеллектуальной Собственности (ВИС) и других организаций, которым необходимо управлять данными ИС, хранить их, обрабатывать, обмениваться и распространять с использованием веб-API. Предполагается, что использование настоящего Стандарта позволит упростить и ускорить разработку веб-интерфейсов API согласованным образом и улучшить совместимость между веб-API.

9. Настоящий стандарт предназначен для связи между ВИС и заявителями или пользователями данных, а также между ВИС через взаимодействия вида устройство - устройство и устройство - программные приложения.

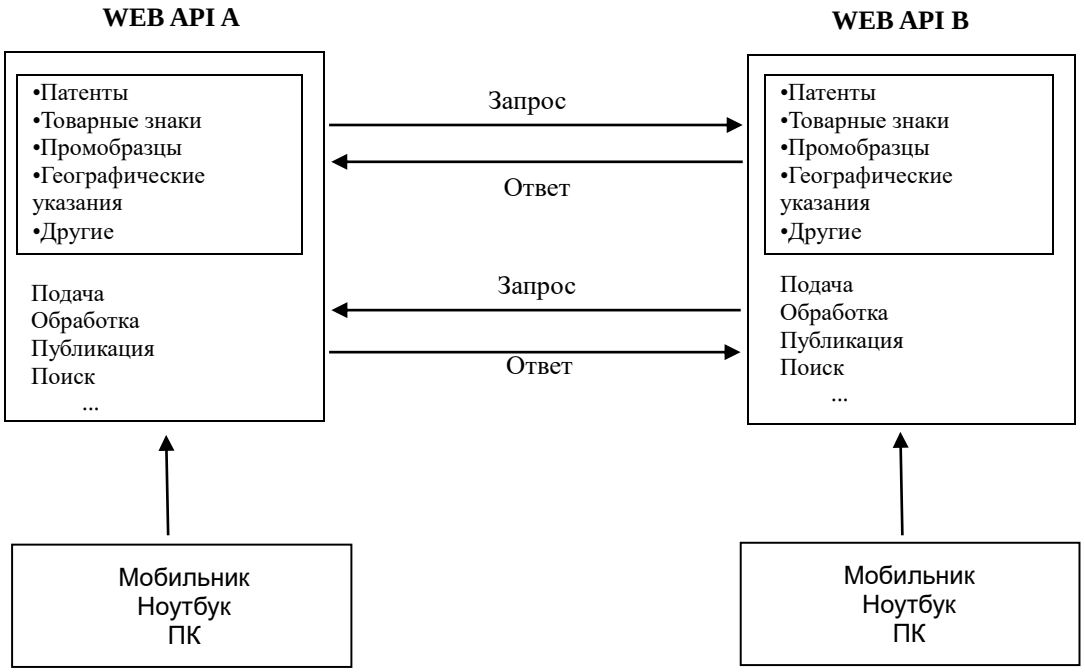


Рис. 1 Цель стандарта.

10. Настоящий Стандарт предоставляет набор правил проектирования и соглашений для RESTful и SOAP Web API; перечень ресурсов данных ИС, которыми будут обмениваться или которые будут раскрыты; и типовую документацию API или контракт на обслуживание, который может быть использован для настройки, с описанием формата сообщений, структуры данных и словаря данных в формате JSON на основе стандарта ВОИС ST.97 и/или XML на основе стандарта ВОИС ST.96.

11. Настоящий стандарт предоставляет типовые сервисные контракты для SOAP Web API, использующие WSDL, и для RESTful Web API, использующие REST API Modeling Language (RAML) и Open API Specification (OAS). Сервисный контракт также определяет или ссылается на типы данных для интерфейсов (см. раздел "Конвенция о типах данных" ниже). Настоящий стандарт рекомендует три типа интерфейсов: REST-XML (XSD), REST-JSON и SOAP-XML (XSD).

12. Настоящий стандарт исключает следующее:

(a) Привязку к конкретным стекам технологий реализации и коммерческим готовым продуктам (COTS);

(b) Привязку к конкретным архитектурным проектам (например, сервис-ориентированная архитектура (SOA) или микросервисная архитектура (MOA));

(c) Привязку к определенным алгоритмам, таким как алгоритмам вычисления ETag, т.е. вычисления уникального идентификатора для конкретной версии ресурса (например, используемого для кэширования).

ПРИНЦИПЫ ПРОЕКТИРОВАНИЯ WEB API

13. Как RESTful Web API, так и SOAP Web API доказали свою способность удовлетворять требованиям крупных организаций, а также обслуживать небольшие встроенные приложения в производстве. При выборе между RESTful и SOAP можно рассмотреть следующие аспекты:

- Безопасность, например, SOAP имеет WS-Security, в то время как REST не определяет никаких моделей безопасности;
- ACID транзакция, например, SOAP имеет спецификацию WS-AT, в то время как REST не имеет соответствующей спецификации;
- Архитектурный стиль, например, микросервисы и бессерверная архитектура используют REST, в то время как SOA использует веб-сервисы SOAP;
- Гибкость;
- Ограничения пропускной способности; и
- Гарантированная доставка, например, SOAP предлагает WS-RM, в то время как REST не имеет соответствующей спецификации.

14. При разработке Web API должны соблюдаться следующие принципы сервис - ориентированного проектирования:

(a) Стандартизированный сервисный контракт: Стандартизация сервисных контрактов является наиболее важным принципом проектирования, поскольку контракты обеспечивают управление и последовательное проектирование сервиса. Сервисный контракт должен быть простым для реализации и понимания. Контракт состоит из метаданных, которые описывают, как будут взаимодействовать поставщик и потребитель. Метаданные также описывают условия их взаимодействия. Рекомендуется, чтобы сервисные контракты включали:

- Функциональные требования: какую функциональность обеспечивает сервис и какие данные он будет возвращать, или, как правило, комбинация этих двух требований;
- Нефункциональные требования: информация об ответственности поставщиков за предоставление своих функциональных возможностей и/или данных, а также об ожидаемых обязанностях потребителей этой информации и о том, что они должны будут предоставить взамен. Например, доступность, безопасность и другие соображения по качеству обслуживания для потребителя.

(b) Свободное соединение сервисов: Клиенты и сервисы должны развиваться независимо друг от друга. Применение этого принципа проектирования требует:

- Версионности услуг - потребители, привязанные к версии Web API, не должны рисковать неожиданными сбоями в работе из-за несовместимых изменений API; и
- Сервисный контракт должен быть независимым от деталей технологии.

(c) Абстракция сервиса - Детали реализации сервиса должны быть скрыты. Дизайн API должен быть независимым от стратегий, поддерживаемых сервером. Например, для веб-сервиса REST модель ресурсов API должна быть отделена от модели сущностей в слое хранения данных;

(d) Нестационарность сервиса - Сервисы должны быть масштабируемыми;

(e) Возможность повторного использования сервиса - Хорошо спроектированный API должен предоставлять многократно используемые сервисы с типовыми контрактами. В связи с этим настоящий стандарт предоставляет типовой контракт на оказание услуг;

(f) Автономность сервиса - функциональные границы службы должны быть четко определены;

(g) Открытость сервиса - Сервисы должны быть эффективно раскрываемы и интерпретированы;

(h) Компонуемость сервиса - Сервисы могут использоваться для компоновки других сервисов;

(i) Использование Стандартов в качестве основы - API предпочтительно следовать отраслевым стандартам (таким как IETF, ISO и OASIS) везде, где это возможно, естественно отдавая им предпочтение перед локально оптимизированными решениями; и

(j) Принцип выбора - Не обязательно реализовывать все правила проектирования API. Правила проектирования должны быть выбраны на основе реализации каждого конкретного случая.

15. Кроме того, необходимо соблюдать следующие принципы, особенно в отношении RESTful Web API:

(a) Кэшируемость: Возвращаемые ответы явно указывают на возможность их кэширования;

(b) Идентификация ресурсов в запросах: отдельные ресурсы идентифицируются в запросах; например, с помощью URI в веб-системах REST. Сами ресурсы концептуально отделены от их представлений, которые возвращаются клиенту;

(c) Гипермедиа как механизм состояния приложения (HATEOAS) - получив доступ к начальному URI для приложения REST - аналогично тому, как человек получает доступ к домашней странице веб-сайта - клиент REST должен иметь возможность динамически использовать предоставленные сервером ссылки для обнаружения всех доступных действий и ресурсов, которые ему необходимы;

(d) Манипулирование ресурсами через их представления - когда клиент имеет отображение ресурса, включая любые прикрепленные метаданные, он обладает достаточной информацией для изменения или удаления ресурса;

(e) Сообщения, не требующие дополнительного описания - каждое сообщение содержит достаточно метаданных, чтобы описать, как обрабатывать содержимое сообщения;

(f) Web API должен следовать семантике HTTP, такой как методы, ошибки и т.д.;

(g) Доступность для общественности - разработка с целью, что API в конечном итоге будет публично доступен в интернете, даже если на данный момент это не планируется;

(h) Общая аутентификация - использование общей схемы аутентификации и авторизации, предпочтительно на основе существующих компонентов безопасности, чтобы избежать создания индивидуального решения для каждого API;

(i) Наименьшая привилегия — права доступа и авторизация должны назначаться потребителям API из соображений минимального объема прав, необходимого для выполнения требуемых функций;

(j) Максимизация энтропии - случайность учетных данных должна быть максимизирована путем использования ключей API, а не имени пользователя и паролей для авторизации API, поскольку ключи API обеспечивают более сложную для потенциальных злоумышленников поверхность атаки; и

(k) Производительность против безопасности - баланс между производительностью и безопасностью с учетом времени жизни ключей и накладных расходов на шифрование/дешифрование.

RESTFUL WEB API

16. RESTful Web API позволяет запрашивающим системам получать доступ к текстовым представлениям веб-ресурсов и манипулировать ими, используя единообразный и заранее определенный набор операций без статических параметров.

Компоненты URI

17. RESTful Web API используют URI для адресации ресурсов. Согласно RFC 3986, синтаксис URI должен быть определен следующим образом:

URI = <scheme> "://" <authority> "/" <path> {"?" query}

URI = <схема> "://" <полномочия> "/" <путь> {"?" запрос}

authority = {userinfo@}host{:port}

Например, https://wipo.int/api/v1/patents?sort=id&offset=10

```
        _____/_____/__________/_________________/
          |     |          |      |
       scheme authority path queri  параметры
```

18. Символ прямой косой черты "/" используется в пути URI для указания иерархических отношений между ресурсами, но путь не должен заканчиваться прямой косой чертой, так как она не несет никакой смысловой нагрузки и может вызвать путаницу.

[RSG-01] Символ прямой косой черты "/" ДОЛЖЕН использоваться в пути URI для указания иерархических отношений между ресурсами, но путь НЕ ДОЛЖЕН заканчиваться прямой косой чертой.

19. URI чувствительны к регистру, за исключением частей схемы и хоста. Например, хотя https://wipo.int/api/my-resources/uniqueId и https://wipo.INT/api/my-resources/uniqueId - это одно и то же, https://wipo.int/api/my-resources/uniqueid - нет. Для имен ресурсов соглашения kebab-case и lowerCamelCase обеспечивают хорошую читабельность и сопоставляют имена ресурсов с сущностями в языках программирования с помощью простого преобразования. Для параметров запроса следует использовать нижний регистр (lowerCamelCase). Например, https://wipo.int/api/v1/inventors?firstName=John. Имена ресурсов и параметры запроса чувствительны к регистру. Обратите внимание, что имена ресурсов и параметры запроса могут быть сокращены.

20. RESTful Web API может иметь аргументы:

- В параметре запроса; например, /inventors?id=1;
- В параметре сегмента пути URI, например, /inventors/1; и
- В полезных данных запроса, например, в теле JSON.

21. За исключением вышеупомянутых типов аргументов, которые являются частью URI, аргумент также может быть частью полезной нагрузки запроса.

[RSG-02] Имена ресурсов ДОЛЖНЫ быть последовательными в своей схеме именования.

[RSG-03] Для имен ресурсов в запросе СЛЕДУЕТ использовать соглашения об именовании в kebab-case стиле, и они МОГУТ быть сокращенными.

[RSG-04] Параметры запроса ДОЛЖНЫ быть последовательными в своем именовании

[RSG-05] Параметры запроса ДОЛЖНЫ использовать соглашение о нижнем регистре (lowerCamelCase), и они МОГУТ быть сокращенными.

22. Конечная точка Web API должна соответствовать стандарту `IETF RFC 3986` и избегать возможных коллизий с URL страниц сайта, размещенного на корневом домене. Web API должен иметь одну четкую точку входа для консолидации всех запросов. В целом, существует два варианта определения конечных точек:

- Как первый сегмент пути в URI, например: https://wipo.int/api/v1/; и
- В качестве поддомена, например: https://api.wipo.int/v1/

[RSG-06] Шаблон URL для Web API ДОЛЖЕН содержать слово "`api`" в URI.

23. Матричные параметры являются признаком того, что API является сложным с несколькими уровнями ресурсов и подресурсов. Это противоречит принципам сервис-ориентированного проектирования, определенным ранее. Кроме того, матричные параметры не являются стандартными, поскольку они применяются к конкретному элементу пути, в то время как параметры запроса применяются к запросу в целом. Примером матричных параметров является следующий: `https://api.wipo.int/v1/path;param1=value1;param2=value2` .

[RSG-07] Матричные параметры НЕ ДОЛЖНЫ использоваться.

Коды состояния

24. Web API должен последовательно применять коды состояния HTTP, как описано в RFC IETF. Коды состояния HTTP должны использоваться из числа тех, которые перечислены в стандартных кодах состояния HTTP (RFC 7807), приведенных в Приложении V.

[RSG-08] Web API ДОЛЖЕН совместимым образом применять коды состояния HTTP, как описано в RFC IETF.

[RSG-09] Рекомендуемые коды в Приложении V СЛЕДУЕТ использовать в Web API для классификации ошибки.

Принцип выбора

25. Сервисный контракт должен быть терпимым к неожиданным параметрам (в запросе, в параметрах запроса), но выдавать ошибку в случае неправильных значений ожидаемых параметров.

[RSG-10] Если API обнаруживает недопустимые значения ввода, он ДОЛЖЕН вернуть код состояния HTTP "`400 Bad Request`". Данные ошибки ДОЛЖНЫ указывать на ошибочное значение.

[RSG-11] Если API обнаруживает синтаксически правильные имена аргументов (в параметрах запроса или запроса), которые не ожидаются, он ДОЛЖЕН игнорировать их.

[RSG-12] Если API обнаруживает допустимые значения, которые требуют нереализованных функций, он ДОЛЖЕН вернуть код состояния HTTP "`501 Not Implemented`". Данные ошибки ДОЛЖНЫ указывать на не обработанное значение.

Ресурсная модель

26. Модель данных ИС должна быть разделена на ограниченные контексты в соответствии с подходом проектирования, ориентированным на предметную область. Каждый ограниченный контекст должен быть сопоставлен с ресурсом. В соответствии с принципами проектирования модель ресурсов Web API должна быть отделена от модели данных. Web API должен быть смоделирован как иерархия ресурсов, чтобы использовать иерархическую природу URI для обозначения структуры (ассоциации, композиции или агрегирования), где каждый узел является либо простым (единичным) ресурсом, либо коллекцией ресурсов.

27. В этой иерархической модели ресурсов узлы в корне называются "узлами верхнего уровня", а все вложенные ресурсы - "субресурсами". Подресурсы должны использоваться только для обозначения композиций, т.е. ресурсов, которые не могут быть ресурсами верхнего уровня, иначе существовало бы несколько способов получения одних и тех же сущностей. Такие субресурсы, подразумевающие объединение, называются субколлекциями. Других иерархических структур, т.е. ассоциации и агрегирования, следует избегать, чтобы избежать сложных API и дублирования функциональности.

28. Конечная точка всегда определяет тип ответа. Например, конечная точка `https://wipo.int/api/v1/patents` всегда возвращает ответы, касающиеся патентных данных. Конечная точка `https://wipo.int/api/v1/patents/1/inventor` всегда возвращает ответы, касающиеся данных изобретателя. Однако конечная точка `https://wipo.int/api/v1/inventors` не допустима, поскольку данные изобретателей не могут быть автономными.

29. Следует использовать только ресурсы верхнего уровня, т.е. не более одного уровня, иначе эти API будут очень сложными для реализации. Например, вместо `https://wipo.int/api/v1/inventors/12345/patents` следует использовать `https://wipo.int/api/v1/patents?inventorId=12345` .

> [RSG-13] Web API ДОЛЖЕН использовать только ресурсы верхнего уровня. Если есть подресурсы, они должны быть коллекциями и подразумевать ассоциацию. Объект должен быть доступен либо как ресурс верхнего уровня, либо как подресурс, но не использовать оба способа.
>
> [RSG-14] Если ресурс может быть самостоятельным, он ДОЛЖЕН быть ресурсом верхнего уровня, в противном случае - подресурсом.
>
> [RSG-15] Параметры запроса ДОЛЖНЫ использоваться вместо путей URL для получения вложенных ресурсов.

30. Существует несколько типов[2] веб-интерфейсов API: CRUD (Create, Read, Update и Delete) и Intent Web API. CRUD Web API моделируют изменения ресурса, т.е. операции создания/чтения/обновления/удаления. Intent Web API, напротив, моделируют бизнес-операции, например, обновление/регистрация/публикация. CRUD-операции должны

[2] В качестве альтернативы мы можем классифицировать API в соответствии с их архетипом. См. например: "REST API Design Rulebook: Проектирование последовательных RESTful Web Service интерфейсов".

использовать существительные, а Intent Web API - глаголы для имен ресурсов. CRUD Web API являются наиболее распространенными, но оба варианта можно комбинировать, например, потребитель услуг может использовать Intent Web API, моделирующий бизнес-операции, который будет дирижировать выполнением одной или нескольких сервисных операций CRUD Web API. При использовании CRUD Web API вызывающий сервис должен управлять бизнес-логикой, а при использовании Intent Web API управление бизнес-логикой осуществляет поставщик услуг. CRUD Web API не являются атомарными по сравнению с Intent Web API[3].

- Например, владелец товарных знаков хочет продлить срок действия тех, которые истекают в ближайшее время (например, на yyyy-mm-dd). Это комбинация следующих бизнес-операций:
    - Извлечь метки, срок действия которых истекает на yyyy-mm-dd; и
    - Продлить действие найденных знаков по их международному регистрационному номеру.

Используя CRUD Web API, предыдущие бизнес-операции были бы смоделированы с помощью неатомарного процесса, требующего двух действий, таких как:

Шаг 1: Получить все товарные знаки в формате XML [4], которые принадлежат владельцу с именем John Smith и срок действия которых истекает, например, 2018-12-31:

```
GET /api/v1/trademarks? holderFullName=John%20Smith&expiryDate=2018-12-31.HTTP/1.1
Host: wipo.int
Accept: application/xml
```

[3] API Intent также позволяет применять паттерн Command Query Responsibility Segregation (CQRS). CQRS - это паттерн, в котором для обновления информации можно использовать другую модель, а не ту, которую вы используете для чтения информации. Обоснование заключается в том, что для многих проблем, особенно в более сложных областях, использование одной и той же концептуальной модели для команд и запросов приводит к созданию более сложной модели, которая не приносит пользы.

[4] Пример JSON пропущен, поскольку он не добавляет никакой ценности в данном случае.

Возвращается следующий пример HTTP-ответа:

```
HTTP/1.1 200 OK

Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>

<tmk:TrademarkBag xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:tmk="http://www.wipo.int/standards/XMLSchema/ST96/Trademark"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Trademark
TrademarkBag.xsd">

       <tmk:Trademark xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:tmk="http://www.wipo.int/standards/XMLSchema/ST96/Trademark"
com:operationCategory="Delete"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Trademark
Trademark.xsd">

              ...

              <com:RegistrationNumber>

                     <com:IPOfficeCode>IT</com:IPOfficeCode>

       <com:ST13ApplicationNumber>000000000000001</com:ST13ApplicationNumber>

              </com:RegistrationNumber>

              ...

              <com:ExpiryDate>2018-12-31</com:ExpiryDate>

              ...

       </tmk:Trademark>

       ...

</tmk:TrademarkBag>
```

Шаг 2: Подать запрос на продление срока действия товарного знака для каждого товарного знака, полученного на предыдущем этапе (здесь изображен только первый запрос на продление срока действия):

```
POST /api/v1/trademarks/renewalRequests HTTP/1.1

Host: wipo.int

Accept: application/xml

Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>

<tmk:MadridRenewal xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:tmk="http://www.wipo.int/standards/XMLSchema/ST96/Trademark"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Trademark
MadridRenewal.xsd">

       ...

       <com:InternationalRegistrationNumber>000000000000001</com:International
RegistrationNumber>

       ...

</tmk:MadridRenewal>
```

- Предыдущий пример также может быть смоделирован с помощью атомарного вызова службы с использованием Intent Web API, например[5]:

```
POST
/api/v1/trademarks/findAndRenew?holderFullName=john%20smith&expiryDate=2018-12-
31

Host: wipo.int
```

31. Тип Web API должен накладывать ограничения на то, как именуются ресурсы, чтобы дать представление о том, какой из них используется. Обратите внимание, что имена ресурсов, локализованные в связи с требованиями бизнеса, могут быть на других языках.

[RSG-16] Именам ресурсов СЛЕДУЕТ быть существительными для CRUD Web API и глаголами для Intent Web API.

[RSG-17] Если имя ресурса является существительным, его СЛЕДУЕТ всегда использовать в форме множественного числа. Неправильные формы существительных НЕ СЛЕДУЕТ использовать. Например, вместо /people следует использовать /persons.

[RSG-18] Имена ресурсов, сегментов и параметры запросов ДОЛЖНЫ состоять из слов на английском языке с использованием основных английских написаний, представленных в Оксфордском словаре английского языка. Имена ресурсов, которые локализованы в связи с требованиями бизнеса, МОГУТ быть на других языках.

[5] Элемент `InternationalRegistrationNumber` был удален из полезной нагрузки для обозначения всех IRN. ST.96 не следует использовать или применять не строго, поскольку приведенный здесь пример расширяет допустимые случаи использования ST.96.

Поддержка нескольких форматов

32. Различные потребители услуг могут иметь различные требования к формату данных в ответах услуг. Тип носителя данных должен быть отделен от самих данных, что позволяет сервису поддерживать различные типы носителей. Поэтому Web API должен поддерживать согласование типа содержимого с помощью HTTP-заголовка `Accept` запроса и HTTP-заголовка `Content-Type` ответа, как того требует IETF RFC 7231. Например, для запроса данных в формате JSON заголовок `Accept` должен быть `Accept: application/json`, а для данных в формате XML - `Accept: application/xml`. Аналогично для заголовка `Content-Type`. Кроме того, Web API может поддерживать другие способы согласования типа содержимого, такие как параметр запроса (например, `?format`) или суффикс URL (например, `.json`).

[RSG-19] Для Web API СЛЕДУЕТ использовать для согласования типа содержимого HTTP-заголовок запроса `Accept` и HTTP-заголовок ответа `Content-Type`.

33. API должны поддерживать запросы и ответы в форматах XML и JSON. В случае XML ответы должны соответствовать Стандарту ВОИС по использованию XML, такому как ST.96, а для JSON ответы должны соответствовать Стандарту ВОИС ST.97. Необходимо использовать последовательное отображение между этими двумя форматами.

[RSG-20] Web API ДОЛЖЕН поддерживать согласование типов содержимого в соответствии с IETF RFC 7231.

[RSG-21] Формат JSON ДОЛЖЕН предполагаться, если не запрашивается определенный тип содержимого.

[RSG-22] Web API СЛЕДУЕТ возвращать код состояния "`406 Not Acceptable`", если запрошенный формат не поддерживается.

[RSG-23] Web API СЛЕДУЕТ отклонять запросы, содержащие неожиданные или отсутствующие заголовки типа содержимого, выдавая кодом состояния HTTP "`406 Not Acceptable`" или "`415 Unsupported Media Type`".

[RSG-24] Запросам и ответам (соглашение об именовании, формат сообщения, структура данных и словарь данных) СЛЕДУЕТ ссылаться на Стандарт ВОИС ST.96 для XML или на Стандарт ВОИС ST.97 для JSON.

[RSJ-25] Имена свойств объектов JSON СЛЕДУЕТ предоставлять в нижнем регистре (lowerCamelCase), например, applicantName.

[RSX-26] Имена компонентов XML СЛЕДУЕТ предоставлять в UpperCamelCase.

[RSG-27] Web API ДОЛЖЕН поддерживать как минимум XML или JSON.

Методы HTTP

34. Методы HTTP (или глаголы HTTP) - это тип функции, предоставляемой единым контрактом для обработки идентификаторов ресурсов и данных. Методы HTTP должны использоваться по назначению в соответствии со стандартной семантикой, как указано в IETF RFC 7231 и 5789, а именно:

- `GET` - получение данных
- `HEAD` - подобно GET, но без полезного содержимого ответа
- `POST` - отправка новых данных
- `PUT` - обновление
- `PATCH` - частичное обновление
- `DELETE` - удалить данные
- `TRACE` - эхо
- `OPTIONS` - глаголы запроса, которые сервер поддерживает для данного URL-адреса

35. Единый контракт устанавливает набор методов, которые должны использоваться службами в рамках данной коллекции или инвентаризации. Туннелирование HTTP-методов может быть полезно, когда HTTP-заголовки отвергаются некоторыми файрволами.

36. Методы HTTP могут следовать принципу "тщательного выбора", который гласит, что должна быть реализована только та функциональность, которая необходима для целевого сценария использования. Некоторые прокси-серверы поддерживают только методы `POST` и `GET`. Чтобы преодолеть эти ограничения, веб-интерфейс может использовать метод POST с пользовательским заголовком HTTP, "туннелирующим" настоящий метод HTTP.

[RSG-28] Методы HTTP ДОЛЖНЫ быть ограничены стандартными методами HTTP `POST`, `GET`, `PUT`, `DELETE`, `OPTIONS`, `PATCH`, `TRACE` и `HEAD`, как указано в IETF RFC 7231 и 5789.

[RSG-29] Методы HTTP МОГУТ следовать принципу выбора, который гласит, что должна быть реализована только та функциональность, которая необходима для целевого сценария использования.

[RSG-30] Некоторые прокси-серверы поддерживают только методы `POST` и `GET`. Чтобы преодолеть эти ограничения, Web API МОЖЕТ использовать метод `POST` с пользовательским заголовком HTTP, "туннелирующим" настоящий метод HTTP. СЛЕДУЕТ использовать пользовательский HTTP-заголовок `X-HTTP-Method`.

[RSG-31] Если метод HTTP не поддерживается, СЛЕДУЕТ возвратить код состояния HTTP `"405 Method Not Allowed"`.

37. В некоторых случаях необходимо поддерживать сразу несколько операций.

[RSG-32] Web API СЛЕДУЕТ поддерживать пакетные операции (они же массовые операции) вместо нескольких отдельных запросов для снижения задержки. Такая же семантика должна использоваться для методов HTTP и кодов состояния HTTP. В содержательной части ответа СЛЕДУЕТ иметь информацию обо всех пакетных

операциях. Если возникает несколько ошибок, содержательной части ошибки СЛЕДУЕТ иметь информацию обо всех ошибках (в атрибуте details). Все пакетные операции СЛЕДУЕТ выполнять атомарно.

GET (запрос GET протокола HTTP)

38. Согласно IETF RFC 2616, протокол HTTP не устанавливает никаких предварительных ограничений на длину URI. С другой стороны, серверы должны быть осторожны, если длина URI превышает 255 байт, поскольку некоторые старые реализации клиентов или прокси-серверов могут не поддерживать такую длину. В случае превышения этого ограничения рекомендуется использовать именованные запросы. В качестве альтернативы необходимо указать набор правил, определяющих, как преобразовывать GET в POST. Согласно IETF RFC 2616, запрос GET должен быть идемпотентным, то есть ответ будет одинаковым независимо от того, сколько раз выполняется запрос.

[RSG-33] Для конечной точки, которая извлекает один ресурс, если ресурс не найден, метод `GET` ДОЛЖЕН возвращать код состояния `"404 Not Found".` Конечные точки, которые возвращают списки ресурсов, просто возвращают пустой список.

[RSG-34] Если ресурс получен успешно, `метод GET` ДОЛЖЕН вернуть `200 OK.`

[RSG-35] Запрос `GET` ДОЛЖЕН быть идемпотентным (независимым от количества выполнений запроса).

[RSG-36] Если длина URI превышает 255 байт, РЕКОМЕНДУЕТСЯ использовать метод `POST` вместо `GET` из-за ограничений `GET`, или если возможно, создавать именованные запросы.

HEAD

39. Когда клиенту нужно узнать информацию об операции, он может использовать `HEAD`. `HEAD` получает HTTP-заголовок, который вы получили бы, если бы сделали `GET`-запрос, но без тела. Это позволяет клиенту определить информацию о кэшировании, какой тип содержимого будет возвращен, какой код состояния будет возвращен. Запрос `HEAD` ДОЛЖЕН быть идемпотентным в соответствии с IETF RFC 2616.

[RSG-37] Запрос `HEAD` ДОЛЖЕН быть идемпотентным.

[RSG-38] Некоторые прокси-серверы поддерживают только методы `POST` и `GET`. Web API СЛЕДУЕТ поддерживать пользовательский заголовок запроса HTTP для переопределения метода HTTP, чтобы преодолеть эти ограничения.

POST

40. Когда клиенту необходимо создать ресурс, он может использовать `POST`. Например, следующий HTTP-запрос отправляет запрос на патентную заявку.

- Например, заявка на патент подается следующим образом.

Пример с содержательной частью в XML на основе ST.96

Клиенты подают заявку на патент в формате XML:

```
POST /v1/patents/applications HTTP/1.1
Host: wipo.int
Accept: application/xml
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:ApplicationBody xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:languageCode="pl" com:receivingOffice="ST" com:st96Version="V5_0"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
ApplicationBody_V5_0.xsd">
       ...
</pat:ApplicationBody>
```

Следующий ответ HTTP возвращается для обозначения успешной подачи патентной заявки:

```
HTTP/1.1 201 Created
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:ApplicationBody xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:languageCode="pl" com:receivingOffice="ST" com:st96Version="V5_0"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
ApplicationBody_V5_0.xsd" applicationBodyStatus=”pending”>
       ...
</pat:ApplicationBody>
```

Пример с содержательной частью в JSON на основании ST.97

Клиенты отправляют запрос на получение патента в формате JSON:

```
POST /v1/patents/applications HTTP/1.1
Host: wipo.int
Accept: application/json
Content-Type: application/json
{
       " applicationBody ": {
              ...
       }
}
```

Следующий ответ HTTP возвращается для обозначения успешной подачи патентной заявки:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
       " applicationBody ": {
              "applicationBodyStatus" : "pending",
              ...
       }
}
```

[RSG-39] POST-запрос НЕ ДОЛЖЕН быть идемпотентным в соответствии с IETF RFC 2616.

[RSG-40] Если создание ресурса было успешным, в HTTP-заголовоке `Location` СЛЕДУЕТ содержать URI (абсолютный или относительный), указывающий на созданный ресурс.

[RSG-41] Если создание ресурса прошло успешно, в ответе СЛЕДУЕТ содержать код состояния "`201 Created`".

[RSG-42] Если создание ресурса прошло успешно, в полезной части ответа СЛЕДУЕТ по умолчанию содержаться телу созданного ресурса, чтобы клиент мог использовать его без дополнительного вызова HTTP.

PUT

41. Когда клиенту необходимо полностью заменить существующий ресурс, он может использовать `PUT`. Следует учитывать идемпотентные характеристики `PUT`. Запрос `PUT` имеет семантику обновления (как указано в IETF RFC 7231) и семантику вставки.

[RSG-43] Запрос `PUT` ДОЛЖЕН быть идемпотентным.

[RSG-44] Если ресурс не найден, `PUT` ДОЛЖЕН возвращать код состояния "`404 Not Found`".

[RSG-45] Если ресурс успешно обновлен, `PUT` ДОЛЖЕН вернуть код состояния "200 `OK`", если обновленный ресурс возвращен, или "`204 No Content`", если он не возвращен.

PATCH

42. Когда клиенту требуется частичное обновление, он может использовать `PATCH`. Следует учитывать идемпотентные характеристики `PATCH`.

- Например, следующий запрос обновляет только язык патента с указанием его номера:

```
PATCH /api/v1/patents/publications/100000000000001 HTTP/1.1
Host: wipo.int
If-Match:456
Content-Type: application/merge-patch+json
{ "languageCode": "en" }
```

43. PATCH не должен быть идемпотентным в соответствии с IETF RFC 2616. Чтобы сделать его идемпотентным, API может следовать предложению IETF RFC 5789 об использовании оптимистической блокировки.

[RSG-46] Запрос `PATCH` НЕ ДОЛЖЕН быть идемпотентным.

[RSG-47] Если Web API реализует частичные обновления, то идемпотентные характеристики `PATCH` РЕКОМЕНДУЕТСЯ принимать во внимание. Для того чтобы сделать его идемпотентным, API МОЖЕТ следовать предложению IETF RFC 5789 об использовании оптимистической блокировки.

[RSG-48] Если ресурс не найден, `PATCH` ДОЛЖЕН вернуть код состояния "`404 Not Found`".

[RSJ-49] Если Web API реализует частичные обновления с помощью `PATCH`, он ДОЛЖЕН использовать формат JSON Merge Patch для описания частичного набора изменений, как описано в IETF RFC 7386, используя тип содержимого `application/merge-patch+json`.

DELETE

44. Когда клиенту нужно удалить ресурс, он может использовать `DELETE`. Запрос `DELETE` не должен быть идемпотентным в соответствии с IETF RFC 2616

[RSG-50] Запрос `DELETE` НЕ ДОЛЖЕН быть идемпотентным.

[RSG-51] Если ресурс не найден, DELETE ДОЛЖЕН вернуть код состояния "`404 Not Found`".

[RSG-52] Если ресурс успешно удален, `DELETE` ДОЛЖЕН вернуть статус "`200 OK`", если удаленный ресурс возвращен, или "`204 No Content`", если он не возвращен.

TRACE

45. Метод `TRACE` не несет семантики API и используется для тестирования и получения диагностической информации в соответствии с IETF RFC 2616, например, для тестирования цепочки прокси-серверов. TRACE позволяет клиенту увидеть, что получено на другом конце цепочки запросов, и использовать эти данные. Запрос `TRACE` НЕ ДОЛЖЕН быть идемпотентным в соответствии с IETF RFC 2616.

[RSG-53] Конечным получателем является либо исходный сервер, либо первый прокси или шлюз, получивший в запросе значение `Max-Forwards` равное нулю. Запрос `TRACE` НЕ ДОЛЖЕН включать тело (запроса).

[RSG-54] Запрос `TRACE` НЕ ДОЛЖЕН быть идемпотентным.

[RSG-55] Значение поля заголовка `Via` HTTP ДОЛЖНО служить для отслеживания цепочки запросов.

[RSG-56] Поле заголовка HTTP `Max-Forwards` ДОЛЖНО использоваться для того, чтобы клиент мог ограничить длину цепочки запросов.

[RSG-57] Если запрос корректный, в ответ СЛЕДУЕТ поместить все сообщение запроса в теле ответа, с `Content-Type "message/http".`

[RSG-58] Ответы на `TRACE` НЕ ДОЛЖНЫ кэшироваться.

[RSG-59] Код состояния "`200 OK`" СЛЕДУЕТ вернуть в `TRACE`.

OPTIONS `(опции в запросе HTTP)`

46. Когда клиенту необходимо получить информацию о Web API, он может использовать `OPTIONS`. `OPTIONS` не несет семантики API. Запрос `OPTIONS` ДОЛЖЕН быть идемпотентным в соответствии с IETF RFC 2616, Custom HTTP Headers.

[RSG-60] Запрос `OPTIONS` ДОЛЖЕН быть идемпотентным.

47. Это обычная практика для Web API, использующих пользовательские HTTP-заголовки, использовать "`X-`" в качестве общего префикса, который RFC 6648 делает устаревшим и не рекомендует использовать.

[RSG-61] Пользовательские HTTP-заголовки, начинающиеся с префикса "`X-`", НЕ РЕКОМЕНДУЕТСЯ использовать.

[RSG-62] Пользовательские HTTP-заголовки НЕ СЛЕДУЕТ использовать для изменения поведения HTTP-методов, за исключением случаев, когда это необходимо для устранения существующих технических ограничений (например, см. [RSG-39]).

[RSG-63] Соглашение об именовании пользовательских HTTP-заголовков - `<организация>-<имя заголовка>` (`<organization>-<header name>`), где полям `<организация>` и `<заголовок>` рекомендуется следовать соглашению о kebab-case стиле именования.

48. Согласно принципам сервис-ориентированного проектирования, клиенты и сервисы должны развиваться независимо друг от друга. Версионность сервисов позволяет это сделать. Общими реализациями версионирования сервисов являются: версионирование заголовка (с помощью пользовательского заголовка), версионирование строки запроса (например, `?v=v1`), версионирование типа носителя (например, `Accept:`

`application/vnd.v1+json`) и версионирование URI (например, `/api/v1/inventors`).

[RSG-64] Web API СЛЕДУЕТ поддерживать единственный метод версионирования сервиса с использованием версионирования URI, например `/api/v1/inventors` или версионирования заголовка, например `Accept-version: v1` или версионирования типа данных, например `Accept: application/vnd.v1+json`. Версионирование строк запросов НЕ РЕКОМЕНДУЕТСЯ использовать.

49. Согласно принципам сервис-ориентированного проектирования, поставщики и потребители услуг также должны развиваться независимо друг от друга. На потребителя услуг не должны влиять незначительные (обратно совместимые) изменения поставщика услуг. Поэтому при версионировании услуг следует использовать только основные версии. Для внутренних неопубликованных API (например, для разработки и тестирования) могут также использоваться минорные версии, например, Semantic Versioning.

[RSG-65] Схему нумерации версий СЛЕДУЕТ применять с учетом только основного номера версии (например, `/v1`).

50. Идентификаторы конечных точек обслуживания включают информацию, которая может меняться со временем. Может оказаться невозможным заменить все ссылки на устаревшую конечную точку, что может привести к тому, что потребитель услуг не сможет в дальнейшем взаимодействовать с конечной точкой обслуживания. Поэтому поставщик услуг может вернуть ответ о перенаправлении. Перенаправление может быть временным или постоянным. Доступны следующие коды состояния HTTP:

	Постоянный	**Временный**
Позволяет изменить метод запроса с POST на GET	301	302
Не позволяет изменить метод запроса с POST на GET	308	307

Поскольку `301` и `302` являются более общими, они предпочтительнее для повышения гибкости и преодоления излишней сложности.

[RSG-66] Сервисные контракты API МОГУТ включать функцию перенаправления конечных точек. Когда потребитель услуг пытается вызвать услугу, может быть возвращен ответ о перенаправлении, чтобы сообщить потребителю услуг о необходимости повторно отправить запрос на новую конечную точку. Перенаправления МОГУТ быть временными или постоянными:

- Временное перенаправление - с использованием заголовка HTTP ответа `Location` и кода статуса HTTP "`302 Found`" в соответствии с IETF RFC 7231; или
- Постоянное перенаправление - с использованием заголовка HTTP-ответа `Location` и кода статуса HTTP "`301 Moved Permanently`" в соответствии с IETF RFC 7238.

51. По мере развития API будет проходить через ряд основных фаз: планирование и проектирование, разработка, тестирование, развертывание и вывод из эксплуатации. Вместо того чтобы давать рекомендации по срокам, в течение которых API предпочтительно должен оставаться в определенной фазе, желательно, чтобы организации или поставщики услуг публиковали свою стратегию жизненного цикла API. Шаблон, содержащий основные компоненты, определяющие стратегию жизненного цикла, приведен в Приложении VII.

[RSG-67] Стратегии жизненного цикла API СЛЕДУЕТ публиковать разработчикам, чтобы помочь пользователям понять, как долго будет поддерживаться та или иная версия.

Шаблоны запросов данных

Параметры пагинации

52. Пагинация - это механизм, позволяющий клиенту получать данные постранично. Используя пагинацию, мы, в соответствии с принципами проектирования, предотвращаем перегрузку поставщика услуг ресурсоемкими запросами. Сервер должен установить размер страницы по умолчанию, если потребитель услуг не указал его. Страничные запросы не могут быть идемпотентными, т.е. страничный запрос не создает моментальный снимок данных.

[RSG-68] Web API СЛЕДУЕТ поддерживать пагинацию (разбивки на страницы).

[RSG-69] Пагинированные запросы НЕ МОГУТ быть идемпотентными.

[RSG-70] Web API ДОЛЖЕН использовать параметры запроса для реализации пагинации (разбивки на страницы).

[RSG-71] Веб-интерфейс НЕ ДОЛЖЕН использовать HTTP-заголовки для реализации пагинации.

[RSG-72] СЛЕДУЕТ использовать параметры `запроса limit=<количество элементов для доставки>` и `offset=<количество элементов для пропуска>` (`limit=<number of items to deliver>` and `offset=<number of items to skip>`), `где limit` - количество возвращаемых элементов (размер страницы), а `пропуск (skip)` - количество элементов, которые должны быть пропущены (смещение). Если ограничение на размер страницы не указано, СЛЕДУЕТ определить значение по умолчанию - глобальное или для каждой коллекции; смещение по умолчанию ДОЛЖНО быть равно нулю "0":

- Например, следующий URL является допустимым:

```
https://wipo.int/api/v1/patents?limit=10&offset=20
```

[RSG-73] Значения параметров `limit` и `offset` СЛЕДУЕТ включить в ответ.

Сортировка

53. Для полученных данных может потребоваться их сортировка по возрастанию или убыванию. Также может использоваться многоключевой критерий сортировки. Сортировка определяется с помощью параметра строки запроса `sort`. Значение этого параметра представляет собой разделенный запятыми список ключей сортировки и направлений сортировки, которые по желанию могут быть добавлены к каждому ключу сортировки и разделены символом двоеточия ':'. Поддерживаются следующие направления сортировки: `'asc'` - по возрастанию или `'desc'` - по убыванию. Клиент может указать направление сортировки для каждого ключа. Если направление сортировки для ключа не указано, то направление по умолчанию задается сервером.

Например:

(a) Только указанные ключи сортировки:

```
sort=key1,key2
```

'`key1`' - первый ключ, `'key2'` - второй ключ, а направления сортировки задаются сервером по умолчанию.

(b) Указаны некоторые направления:

```
sort=key1:asc,key2
```

где `'key1'` - первый ключ (порядок возрастания), а '`key2`' - второй ключ (направление по умолчанию задается сервером, т.е. любой ключ сортировки без соответствующего направления задается по умолчанию).

(c) каждый ключ с заданным направлением:

```
sort=key1:asc,key2:desc
```

где `'key1'` - первый ключ (в порядке возрастания), а `'key2'` - второй ключ (в порядке убывания).

54. Чтобы задать сортировку по многоатрибутным критериям, значением параметра запроса может быть список ключей сортировки и направлений сортировки, разделенных запятыми, с символами `'asc'` для восходящего или '`desc`' для нисходящего, которые могут быть добавлены к каждому ключу сортировки, разделенные символом двоеточия ':'.

[RSG-74] Web API СЛЕДУЕТ поддерживать сортировку.

[RSG-75] Для указания критерия сортировки по нескольким атрибутам ДОЛЖЕН использоваться параметр запроса. Значение этого параметра представляет собой список ключей сортировки, разделенных запятыми, а направления сортировки либо `'asc'` для восходящего, либо `'desc'` для нисходящего МОГУТ быть добавлены к каждому ключу сортировки, разделенные символом двоеточия ':'. Направление по умолчанию ДОЛЖНО быть определено сервером в случае, если для ключа не указано направление сортировки.

[RSG-76] Web API СЛЕДУЕТ возвращать критерии сортировки в ответе.

Расширение

55. Потребитель услуг может контролировать объем получаемых им данных, расширяя одно поле в более крупные объекты. Обычно это сочетается с поддержкой гипермедиа. Вместо того чтобы просто запрашивать включение идентификатора связанной сущности, пользователь сервиса может запросить полное представление сущности в результатах. Вызовы служб могут использовать расширения для получения всех необходимых данных в одном запросе API:

- Например, если поддерживается гипермедиа, то следующий HTTP-запрос получает патент и расширяет его заявителя.

Пример с полезной информацией JSON на основе ST.97

Получение патента по его номеру[6]:

```
GET /api/v1/patents/publications/100000000000001 HTTP/1.1
Host: wipo.int
Accept: application/json
```

Ответ HTTP выглядит следующим образом:

[6] Patent/PatentNumber.xsd

```
HTTP/1.1 200 OK
Content-Type: application/json
200 OK
{
       "patentPublication":{
              "languageCode": "en",
               ...
              "bibliographicData": {
                     "st96Version": "V5_0",
                     "applicationIdentification": {
                           "ipOfficeCode": "XX",
                           "applicationNumber": {
                                 "applicationNumberText": "13797521"
                           },
                           "inventionSubjectMatterCategory": "Utility",
                           "filingDate": "2013-03-12"
                     },
                     "patentGrantIdentification": {
                           "ipOfficeCode": "XX",
                            "patentNumber": "100000000000001"
                     },
                     ...
                     "partyBag": {
                           "applicantBag": {
                                 "applicant": {
                                  "href":
"https://wipo.int/api/v1/link/to/applicants"
                                 },
                            ...
                            }
              },
              ...
       }
}
```

Вместо предыдущего запроса с помощью следующего HTTP-запроса можно получить полную информацию о заявителе патента с номером 100000000000001:

```
GET /api/v1/patents/publications?id=100000000000001&expand=applicant HTTP/1.1
Host: wipo.int
Accept: application/json
```

Ответ HTTP выглядит следующим образом:

```
HTTP/1.1 200 OK
Content-Type: application/json
200 OK
{
       "patentPublication":{
              "languageCode": "en",
              ...
             "bibliographicData": {
                    "st96Version": "V5_0",
                    "applicationIdentification": {
                          "ipOfficeCode": "XX",
                          "applicationNumber": {
                                "applicationNumberText": "13797521"
                          },
                          "inventionSubjectMatterCategory": "Utility",
                          "filingDate": "2013-03-12"
                    },
                    "patentGrantIdentification": {
                          "ipOfficeCode": "XX",
                          "patentNumber": "100000000000001"
                    },
                   ...
                   "partyBag": {
                         "applicantBag": {
                               "applicant": {
                                          {
                                             "sequenceNumber": "001",
                                             "publicationContact": [
                                                  {
                                                       "name": {
                                                            "personName": …
                                             "applicantCategory": "Applicant",
                                           },
                                           {
                                             "sequenceNumber": "002",
                                             "publicationContact": [
                                                   {
                                                         "name": {
                                                             "personName": …
                                                         }
                                                   }
                                            },
                                            "applicantCategory": "Applicant",
                                          },
                                          {
                                            "sequenceNumber": "003",
                                            "publicationContact": [
                                                   [
                                                         "name": {
                                                             "personName": …
                                                          }
                                                   }
                                             },
                                             "applicantCategory": "Applicant",
                                          },
                                    ...
                                     }
                               }

                          },

                                    ... }}
```

56. Web API может поддерживать расширение тела возвращаемого содержимого.

> [RSG-77] Web API МОЖЕТ поддерживать расширение тела возвращаемого содержимого. СЛЕДУЕТ использовать параметр запроса `expand=<разделенный запятыми список имен атрибутов>`.

Проекция

57. Web API следует поддерживать проекцию полей, которая контролирует, какая часть данных сущности возвращается в ответ на запрос API. Проекция полей может уменьшить время ответа и размер полезной нагрузки. Если требуются только определенные атрибуты из полученных данных, вместо путей URL следует использовать параметр запроса проекции. Параметр запроса должен быть сформирован следующим образом: `"fields="<разделенный запятыми список имен атрибутов>`. Параметр запроса проекции проще в реализации и позволяет получить несколько атрибутов. Если поддерживается проекция, схема XSD/JSON не должна применяться в ответе, поскольку ответ не будет соответствовать оригинальной схеме XSD/JSON.

- Например, следующий запрос возвращает только полное имя запрашиваемого изобретателя патента:

В случае с содержательной частью в виде XML на основании ST.96

Получить полное имя изобретателя патента с id, равным id12345:

```
GET /api/v1/patents/inventors/id12345?fields=fullName
Host: wipo.int
Accept: application/xml
```

Показан пример ответного сообщения HTTP:

```
HTTP/1.1 200 OK
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:Inventor xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:sequenceNumber="String" com:id="ID1"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
PatentPublication_V5_0.xsd">
       <Contact>
              <Name>
                     <PersonName>
                            <PersonFullName>John Smith</PersonFullName>
                     </PersonName>
              </Name>
       </Contact>
</pat:Inventor>
```

В случае с содержимым в виде JSON на основании ST.97

Получить полное имя изобретателя патента с id[7], равным id12345:

```
GET /api/v1/patents/inventors/id12345?fields=fullName
Host: wipo.int
Accept: application/json
```

Показан пример ответного сообщения HTTP:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      "inventor": {
            "sequenceNumber": "001",
            "Contact": [
                  {
                        "name": {
                              "personName": [
                                    {
                                          "personFullName": "John Smith"
                                    }
                              ]
                        }
                  }
            ]
}}
```

[RSG-78] Параметр запроса СЛЕДУЕТ использовать вместо путей URL в случае, если веб-интерфейс поддерживает проекцию в формате: "fields="<разделенный запятыми список имен атрибутов>.

Количество элементов

58. В некоторых случаях потребителя API может интересовать количество элементов в коллекции. Это очень часто встречается в сочетании с пагинацией, чтобы узнать общее количество элементов в коллекции.

- Например, следующий HTTP-запрос извлекает максимум 3 патентные публикации, пропуская первые 4 результата, и должен также содержать в ответе общее количество доступных результатов:

Пример с содержательной частью в XML на основе ST.96

```
GET /api/v1/patents/publications?count=true&limit=3&offset=4 HTTP/1.1
Host: wipo.int
Accept: application/xml
```

Возвращается следующий пример HTTP-ответа:

[7] Common/id.xsd

```
HTTP/1.1 200 OK
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:PatentPublication
xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:languageCode="de" com:st96Version="V5_0"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
PatentPublication_V5_0.xsd">
   ...
</pat:PatentPublication>
<pat:PatentPublication>
   ...
</pat:PatentPublication>
   ...
<pat:PatentPublication>
   ...
</pat:PatentPublication>
<count>100</count>
```

Пример с содержимым в виде JSON на основании ST.97

```
GET /api/v1/patents/publications?count=true&limit=3&offset=4 HTTP/1.1
Host: wipo.int
Accept: application/json
```

Возвращается следующий пример HTTP-ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
       "patentPublication": [
              {
              ...
              },
              {
              ...
              },
              {
              ...
              }
       ],
       "count": 100
}
```

59. В качестве альтернативы, веб-интерфейс может поддерживать возврат количества элементов в коллекции внутри, т.е. в той части ответа, которая содержит саму коллекцию. В качестве альтернативы, это может быть часть набора метаданных, вне основного тела ответа.

[RSG-79] Web API ДОЛЖЕН поддерживать возврат количества элементов в коллекции.

[RSG-80] Параметр запроса ДОЛЖЕН использоваться для поддержки возврата количества элементов в коллекции.

[RSG-81] СЛЕДУЕТ использовать параметр запроса `count` для возврата количества элементов в коллекции.

[RSG-82] Web API МОЖЕТ поддерживать возврат количества элементов в коллекции внутри, т.е. как часть ответа, которая содержит саму коллекцию. При этом ДОЛЖЕН использоваться параметр запроса.

[RSG-83] СЛЕДУЕТ использовать параметр запроса `count=true`. Если он не указан, то по умолчанию `count` следует установить в `false`.

[RSG-84] Если Web API поддерживает пагинацию, то СЛЕДУЕТ поддерживать возврат внутри ответа размера коллекции (т.е. общего количества элементов коллекции).

Сложные поисковые выражения

60. Для получения данных с несколькими критериями поиска достаточно параметров запроса. Если есть сценарий использования, где мы должны искать данные с помощью сложных поисковых выражений (с несколькими критериями, булевыми выражениями и операторами поиска), то API должен быть разработан с использованием более сложного языка запросов. Язык запросов должен поддерживаться грамматикой поиска.

61. Язык контекстных запросов (CQL - Contextual Query Language) - это формальный язык для представления запросов к информационно-поисковым системам, таким как поисковые системы, библиографические каталоги и информация о музейных коллекциях. Основанный на семантике стандарта Z39.50[8], его цель проектирования заключается в том, что запросы должны быть читаемыми и записываемыми, а язык должен быть интуитивно понятным и сохранять выразительность более сложных языков запросов. Это лишь один из вариантов, рекомендованных к использованию, поскольку он широко применяется в промышленности.

[RSG-85] Если Web API поддерживает сложные поисковые выражения, СЛЕДУЕТ указывать язык запросов, например, CQL.

[RSG-86] Сервисный (веб-сервисный) контракт ДОЛЖЕН определять поддерживаемую грамматику (например, поля, функции, ключевые слова и операторы).

[RSG-87] ДОЛЖЕН использоваться параметр запроса "q".

Обработка ошибок

62. В ответах на ошибки всегда следует использовать соответствующий код состояния HTTP, выбранный из стандартного списка кодов состояния HTTP (RFC 7807), приведенного в Приложении V. Если запрашивающая сторона ожидает JSON, возвращайте информацию об ошибке в общей структуре данных. Если проект не требует иного, нет необходимости определять специфические для приложения коды ошибок. Трассировка стека и другая

[8] Пожалуйста, обратитесь к главе "Ссылки

информация, связанная с отладкой, не должна присутствовать в теле ответа на ошибку в производственных средах.

Содержательная часть ошибки (ответа с указанием на ошибку)

63. Обработка ошибок осуществляется на двух уровнях: на уровне протокола (HTTP) и на уровне приложения (возвращаемая полезная нагрузка). На уровне протокола Web API возвращает соответствующий код состояния HTTP, а на уровне приложения Web API возвращает полезную нагрузку, сообщающую об ошибке с достаточной детализацией (обязательные и необязательные атрибуты).

64. Что касается обязательных и необязательных атрибутов для обработки ошибок на уровне приложения,

(a) следующие атрибуты `кода` и `сообщения` являются обязательными, и хотя `сообщение` (`message`) может измениться в будущем, `код` не изменится; он фиксирован и всегда будет относиться к данной конкретной проблеме:

- `код` (целое число) - технический код ситуации ошибки, который будет использоваться в целях поддержки; и
- `message` (string) - Сообщение для пользователя (локализуемое), описывающее ошибочный запрос в соответствии с требованиями HTTP-заголовка `Accept-Language` (см. RSG-114).

(b) Следующие атрибуты являются условно обязательными:

- `details` - Если обработка ошибок требует вложенности ответов на ошибки, она должна использовать поле details для этой цели. Поле details должно содержать массив объектов JSON, в котором отображаются свойства кода и сообщения с такой же семантикой, как описано выше.

(c) Следующие атрибуты являются необязательными:

- `target` - структура сообщения об ошибке может содержать атрибут target, описывающий элемент данных (например, путь к ресурсу);
- `status` - Дубликат кода состояния HTTP для распространения его по цепочке вызовов или для записи в журнал поддержки без необходимости каждый раз явно добавлять код состояния HTTP;
- `moreInfo` - Массив ссылок, содержащих дополнительную информацию о ситуации с ошибкой, например, дающих подсказки конечному пользователю; и
- `internalMessage` - Техническое сообщение, например, для ведения журнала.

65. В обработке ошибок следует соответствовать стандартам HTTP (RFC 2616). Рекомендуется минимальная полезная часть сообщения об ошибке:

- Например, следующие ответы HTTP возвращаются, когда товарный знак не найден для указанного номера международной регистрации:

Пример с содержательной частью в XML на основе ST.96

```
GET /api/v1/trademarks?irn=000000000000001John%20Smith&expiryDate=2018-12-31.
HTTP/1.1
Host: wipo.int
Accept: application/xml
```

Возвращается следующий пример HTTP-ответа:

```
HTTP/1.1 404
Content-Type: application/xml
 <?xml version="1.0" encoding="UTF-8"?>
<com:TransactionError xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Common
TransactionError.xsd">
       <com:TransactionErrorCode>TRADEMARK_NOT_FOUND</com:TransactionErrorCode>
       <com:TransactionErrorText>The trademark with the provided International
Registration Number was not found</com:TransactionErrorCode>
</com:TransactionError>
```

Пример с содержательной частью в виде JSON на основе ST.97

```
HTTP/1.1 404
Content-Type: application/json
{
       "transactionError": [
             {
                   "transactionErrorCode": " TRADEMARK_NOT_FOUND",
             },
             {
                   "transactionErrorText": " The trademark with the provided
International Registration Number was not found"
             },
      ]
 }
```

[RSG-88] На уровне протокола Web API ДОЛЖЕН возвращать соответствующий код состояния HTTP, выбранный из списка стандартных кодов состояния HTTP.

[RSJ-89] На уровне приложения Web API ДОЛЖЕН возвращать часть, сообщающую об ошибке с достаточной степенью детализации. Атрибуты `code` и `message` являются обязательными, атрибут `details` является условно обязательным, атрибуты `target`, `status`, `moreInfo` и `internalMessage` являются необязательными.

[RSG-90] Ошибки НЕ ДОЛЖНЫ раскрывать критически важные для безопасности данные или внутренние технические детали, такие как стеки вызовов, в сообщениях об ошибках.

[RSG-91] `Заголовок` HTTP: `Reason-Phrase` (описанный в RFC 2616) НЕ ДОЛЖЕН использоваться для передачи сообщений об ошибках.

Идентификатор корреляции (Correlation ID)

66. Как правило, потребление услуги приводит к запуску множества других услуг. Должен существовать механизм для корреляции всех активаций сервисов в одном и том же контексте выполнения. Например, включение идентификатора корреляции в сообщения журнала, так как это однозначно идентифицирует зарегистрированную ошибку. Следует использовать имя заголовка. Например, обычно используются Request-ID или Correlation-ID, так как учет этого фактора на этапе разработки API будет способствовать совместимости между различными API и новыми реализациями.

[RSG-92] Каждой зарегистрированной ошибке СЛЕДУЕТ иметь уникальный идентификатор корреляции. СЛЕДУЕТ использовать пользовательский HTTP-заголовок, которому СЛЕДУЕТ иметь имя Correlation-ID.

Договор на оказание услуг (сервисный контракт)

67. REST - это не протокол и не архитектура, а архитектурный стиль с архитектурными свойствами и архитектурными ограничениями. Официальных стандартов для контрактов REST API не существует. В настоящем стандарте документация API называется Контрактом услуг REST. Сервисный контракт основан на следующих трех фундаментальных элементах:

(a) Синтаксис идентификатора ресурса - как мы можем выразить, куда или откуда передаются данные?

(b) Методы - какие протокольные механизмы используются для передачи данных?

(c) Типы носителей - какой тип данных передается? Отдельные сервисы REST используют эти элементы в различных комбинациях для раскрытия своих возможностей. Определение основного набора этих элементов для использования коллекцией (или перечнем) сервисов делает этот тип сервисного контракта "унифицированным".

[RSG-93] В формат сервисного контракта СЛЕДУЕТ включать следующее:

- Версия API;
- Информация о семантике элементов API;
- Ресурсы;
- Атрибуты ресурса;
- Параметры запроса;
- Методы;
- Типы носителей;
- Грамматика поиска (если она поддерживается);
- Коды состояния HTTP;
- Методы HTTP;
- Ограничения и отличительные особенности; и
- Безопасность (например, частные схемы).

[RSG-94] В формат сервисного контракта СЛЕДУЕТ включать запросы и ответы в XML-схеме или JSON-схеме и примеры использования API в поддерживаемых форматах, т.е. XML или JSON.

[RSG-95] REST API ДОЛЖЕН предоставлять документацию API в виде сервисного контракта.

[RSG-96] Реализация Web API, отклоняющаяся от настоящего стандарта, ДОЛЖНА быть явно документирована в сервисном контракте. Если правило отклонения не указано в сервисном контракте, ДОЛЖНО подразумеваться, что соблюдается настоящий стандарт.

[RSG-97] Сервисный контракт ДОЛЖЕН позволять генерировать скелетный код клиента API.

[RSG-98] Сервисный контракт РЕКОМЕНДУЕТСЯ, что может позволять генерировать скелетный код сервера.

68. Документация Web API может быть написана, например, на RESTful API Modeling Language (RAML — язык моделирования API RESTful), Open API Specification (OAS) и WSDL. Поскольку только RAML полностью поддерживает проверку на корректность запросов/ответов в форматах XML и JSON (с помощью схем XSD и схем JSON), настоящий стандарт рекомендует использовать RAML[9].

[RSG-99] Документацию Web API СЛЕДУЕТ писать на RAML или OAS. Пользовательские форматы документации НЕ СЛЕДУЕТ использовать.

Тайм-аут

69. В соответствии с принципами сервис-ориентированного проектирования, использование сервера должно быть ограничено.

[RSG-100] СЛЕДУЕТ дать возможность потребителю Web API указать тайм-аут сервера для каждого запроса; следует использовать пользовательский HTTP-заголовок. Максимальный тайм-аут сервера СЛЕДУЕТ также применять для защиты ресурсов сервера от чрезмерного использования.

Управление состоянием

70. Если разработка осуществляется в соответствии с принципами REST, управление состоянием должно осуществляться на стороне клиента, а не на сервере, поскольку REST API не имеет состояния. Например, если несколько серверов реализуют сеанс, не следует поощрять репликацию.

Версионирование ответов

71. Многократное получение одного и того же набора данных может привести к потреблению полосы пропускания, если набор данных не был изменен между запросами. Данные следует извлекать условно, если только они не были модифицированы. Это можно сделать с помощью проверки ресурса на основе содержимого или проверки ресурса на основе времени. При использовании версионности ответа потребитель услуг может реализовать оптимистичную блокировку.

[9] OAS - это спецификация. Она также поддерживает Markdown, а RAML - нет. С другой стороны, хотя и OAS, и RAML поддерживают проверку JSON Schema для запросов и ответов, OAS не поддерживает XSD. Поэтому в будущем, когда OAS будет полностью функциональна, ее можно будет рекомендовать.

[RSG-101] Web API СЛЕДУЕТ поддерживать условное получение данных, чтобы гарантировать, что будут получены только измененные данные. СЛЕДУЕТ применять проверку ресурсов на основе содержания, поскольку она точнее.

[RSG-102] Для реализации Content-based Resource Validation (проверка корректности ресурса по содержимому) HTTP-заголовок `ETag` СЛЕДУЕТ использовать в ответе для кодирования состояния данных. Затем это значение СЛЕДУЕТ использовать в последующих запросах в условных HTTP-заголовках (таких как If-Match или If-None-Match — Если соответствует или Если не соответствует). Если данные не были изменены с тех пор, как запрос вернул `ETag`, серверу СЛЕДУЕТ вернуть код состояния "`304 Not Modified`" (если данные не изменены). Этот механизм описан в IETF RFC 7231 и 7232.

[RSG-103] Для реализации проверки ресурсов по времени СЛЕДУЕТ использовать HTTP-заголовок `Last-Modified`. Этот механизм описан в IETF RFC 7231 и 7232.

[RSG-104] Используя версионность ответа, потребитель услуг МОЖЕТ реализовать оптимистическую блокировку.

Кэширование

72. В реализации Web API следует поддерживать обработку кэша для экономии пропускной способности, в соответствии с IETF RFC 7234.

[RSG-105] Web API ДОЛЖЕН поддерживать кэширование результатов `GET`; Web API МОЖЕТ поддерживать кэширование результатов и других методов HTTP.

[RSG-106] СЛЕДУЕТ использовать HTTP-заголовки ответа `Cache-Control` и `Expires`. Последний МОЖЕТ использоваться для поддержки устаревших клиентов.

Управляемая передача файлов

73. Передача (т.е. загрузка или выгрузка) больших файлов имеет высокую вероятность сбоя из-за проблем сети или каких-либо других. Она также потребляет большое количество памяти как у поставщика услуг, так и у потребителя услуг. Поэтому рекомендуется передавать большие файлы несколькими кусками с несколькими запросами. В этом случае также обеспечивается индикация общего прогресса загрузки или выгрузки. Частичная передача больших файлов должна поддерживать возобновление. Поставщик услуг должен объявить, поддерживает ли он частичную передачу больших файлов.[10]

74. Существует два подхода для реализации этого типа передачи: первый - использование заголовка `Transfer-Encoding: chunked` и второй - использование заголовка `Content-Length`. Эти заголовки не следует использовать вместе. `Content-Length` указывает на полный размер передаваемого файла, поэтому получатель будет знать длину тела и сможет оценить время завершения загрузки. Заголовок `Transfer-Encoding: chunked` полезен для неограниченной потоковой передачи ограниченных данных, таких как аудио или видео, но не файлов. Для скачивания рекомендуется использовать заголовок

[10] Поставщик услуг может вернуть местоположение файла, а затем потребитель услуг может вызвать службу каталогов для загрузки файла. В конце требуется частичная загрузка файла. Этот пункт не учитывает не-REST протоколы, такие как FTP или sFTP или rsync.

`Content-Length`, так как загрузка сервера низкая по сравнению с `Transfer-Encoding: chunked`. Для заливки данных рекомендуется использовать заголовок `Transfer-Encoding: chunked`.

Web API следует сообщать, поддерживает ли он частичную загрузку файлов, отвечая на запросы HEAD и передавая в ответ заголовки HTTP-ответов: `Accept-Ranges` и `Content-Length`. Первый должен указывать единицу измерения, которая может быть использована для определения диапазона, и никогда не должен быть определен как 'none'. Последний указывает полный размер загружаемого файла.

[RSG-107] Web API СЛЕДУЕТ сообщать, поддерживается ли частичная загрузка файлов, отвечая на запросы HEAD и передавая в ответ HTTP-заголовки `Accept-Ranges` и `Content-Length`.

75. В Web API, поддерживающем загрузку больших файлов, следует поддерживать частичные запросы в соответствии с IETF RFC 7232, то есть:

- Потребитель услуг, запрашивающий диапазон, должен использовать HTTP-заголовок `Range;`
- Ответ поставщика услуг должен содержать HTTP-заголовки `Content-Range` и `Content-Length`; и
- В случае успешного запроса диапазона ответ поставщика услуг должен иметь HTTP-статус `206 Partial Content`. В случае, если запрос диапазона выходит за рамки (значения диапазона перекрывают объем ресурса), сервер отвечает статусом "`416 Requested Range Not Satisfiable`". В случае если запрос диапазона не поддерживается, сервер посылает в ответ статус "`200 OK`".

[RSG-108] Web API СЛЕДУЕТ поддерживать частичную загрузку файлов. Следует поддерживать диапазоны из нескольких частей.

76. Составные диапазоны также могут быть запрошены, если используется HTTP-заголовок `Content-Type: multipart/byteranges; border=XXXXX`. Запрос диапазона может быть условным, если он сочетается с HTTP-заголовками `ETag` или `If-Range`.

77. Не существует RFC IETF для заливки больших файлов. Поэтому в данном стандарте мы не даем никаких рекомендаций по реализации заливки больших файлов.

[RSG-109] Web API СЛЕДУЕТ сообщать, поддерживает ли он частичную заливку файлов.

[RSG-110] Web API СЛЕДУЕТ поддерживать частичную заливку файлов. СЛЕДУЕТ поддерживать составные диапазоны.

78. IETF RFC 2616 не устанавливает никаких конкретных ограничений на размер запросов. API сервисного контракта должен определять максимальный предел для запросов. Более того, во время выполнения поставщик услуг должен указать потребителю услуг, если допустимый максимальный предел был превышен.

[RSG-111] Поставщику услуг СЛЕДУЕТ возвращать в заголовках ответа HTTP заголовок "`413 Request Entity Too Large`" в случае, если запрос превысил максимально допустимый предел. Пользовательский HTTP-заголовок МОЖЕТ использоваться для указания максимального размера запроса.

Обработка предпочтений (настроек)

79. Поставщик услуг может позволить потребителю услуг настраивать значения и влиять на то, как первый обрабатывает запросы второго. Стандартные средства для реализации обработки предпочтений описаны в IETF RFC 7240.

[RSG-112] Если Web API поддерживает обработку настроек, ее СЛЕДУЕТ реализовать в соответствии с IETF RFC 7240, т.е. следует использовать HTTP-заголовок запроса `Prefer`, в HTTP-заголовке ответа СЛЕДУЕТ вернуть `Preference-Applied` (с повторением исходного запроса).

[RSG-113] Если Web API поддерживает обработку настроек, номенклатура настроек, которые МОГУТ быть установлены с помощью заголовка `Prefer`, ДОЛЖНА быть записана в сервисном контракте.

Перевод

80. Потребитель услуги может запросить ответы на определенном языке, если поставщик услуги поддерживает его. Стандартная спецификация для обработки набора естественных языков изложена в IETF TFC 7231.

[RSG-114] Если Web API поддерживает локализованные данные, HTTP-заголовок запроса `Accept-Language` ДОЛЖЕН поддерживаться для указания набора естественных языков, которые предпочтительны в ответе, как указано в IETF RFC 7231.

Продолжительные операции

81. Бывают случаи, когда Web API может включать в себя длительные операции. Например, создание PDF-файла поставщиком услуг может занять несколько минут. В этом параграфе рекомендуется типичный шаблон обмена сообщениями для реализации таких случаев, например:

```
// (a)
GET https://wipo.int/api/v1/patents
Accept: application/pdf
…
// (b)
HTTP/1.1 202 Accepted
Location: https://wipo.int/api/v1/queues/12345
…
// (c1)
GET https://wipo.int/api/v1/queues/12345
…
HTTP/1.1 200 OK
…
// (c2)
GET https://wipo.int/api/v1/queues/12345
HTTP/1.1 303 See Other
Location: https://wipo.int/api/v1/path/to/pdf
…
// (c3)
GET https://wipo.int/api/v1/path/to/pdf
…
```

82. Если API поддерживает длительные операции, им следует выполняться асинхронно, чтобы пользователю не приходилось ждать ответа. В приведенном ниже правиле изложен рекомендуемый подход к реализации.

[RSG-115] Если API поддерживает длительные операции, им СЛЕДУЕТ быть асинхронными. Должен применяться следующий подход:

a) Потребитель услуг активирует сервисную операцию;

b) Операция сервиса возвращает код состояния "`202 Accepted`" в соответствии с IETF RFC 7231 (раздел 6.3.3), т.е. запрос был принят к обработке, но обработка не была завершена. Местоположение созданной задачи в очереди также возвращается с HTTP-заголовком `Location`; и

c) Потребитель услуги обращается к возвращаемому местоположению, чтобы узнать, доступен ли ресурс. Если ресурс недоступен, в ответе СЛЕДУЕТ иметь код состояния "`200 OK`", содержать статус задачи (например, ожидание) и МОЖЕТ содержаться другая информация (например, индикатор выполнения и/или ссылка для отмены или удаления задачи с помощью HTTP-метода `DELETE`). Если ресурс доступен, в ответе СЛЕДУЕТ привести код состояния "`303 See Other`", а в заголовоке HTTP `Location` СЛЕДУЕТ привести URL для получения результатов задачи.

Модель безопасности

Общие правила

83. В рамках настоящего стандарта безопасность API связана с основными атрибутами безопасности, которые обеспечивают безопасность информации, доступной через API, и самих API на протяжении всего их жизненного цикла. Этими атрибутами являются

конфиденциальность, целостность, доступность, доверие, неотказуемость, разделение, аутентификация, авторизация и аудит.

[RSG-116] Конфиденциальность**:** Различные API и информация в API ДОЛЖНЫ быть идентифицированы, классифицированы и защищены от несанкционированного доступа, раскрытия и перехвата в любое время. Должны соблюдаться [11]принципы наименьших привилегий, нулевого доверия, необходимости знать и необходимости делиться.

[RSG-117] Обеспечение целостности: Различные API и информация в API ДОЛЖНЫ быть защищены от несанкционированной модификации, дублирования, повреждения и уничтожения. Информация ДОЛЖНА модифицироваться через подтвержденные транзакции и интерфейсы. Системы ДОЛЖНЫ обновляться с использованием утвержденных процессов управления конфигурацией, управления изменениями и управления исправлениями.

[RSG-118] Доступность: Различные API и информация в API ДОЛЖНЫ быть доступны авторизованным пользователям в установленное время в соответствии с соглашениями об уровне обслуживания (SLA), политиками контроля доступа и определенными бизнес-процессами.

[RSG-119] Неотказуемость (безотзывность): Каждая обрабатываемая транзакция или действие, выполняемое API, ДОЛЖНЫ обеспечивать неотказуемость посредством реализации надлежащего аудита, авторизации, аутентификации, а также реализации безопасных путей, а также и сервисов и механизмов неотказуемости.

[RSG-120] Аутентификация, авторизация, аудит: Пользователи, системы, API или устройства, вовлеченные в критические транзакции или действия, ДОЛЖНЫ быть аутентифицированы, авторизованы с помощью служб контроля доступа на основе ролей или атрибутов и поддерживать разделение обязанностей. Кроме того, все действия ДОЛЖНЫ регистрироваться, а надежность аутентификации должна увеличиваться в соответствии с соответствующим информационным риском.

Руководство по безопасному и устойчивому к угрозам управлению API

84. API следует проектировать, создавать, тестировать и внедрять с учетом требований безопасности и рисков. Соответствующие меры противодействия и средства контроля должны быть встроены непосредственно в проект, а не рассматриваться после его завершения. Рекомендуется использовать лучшие практики и стандарты, такие как OWASP.

[RSG-121] При разработке API ТРЕБУЕТСЯ тщательно учитывать угрозы, случаи вредоносного использования, методы безопасного кодирования, безопасность транспортного уровня и тестирование безопасности, особенно:

– `PUTs` и `POSTs` – т.е.: какие изменения внутренних данных потенциально могут быть использованы для атаки или ложной информации;

[11] https://www.owasp.org/index.php/Security_by_Design_Principles

– DELETES - т.е.: может использоваться для удаления содержимого внутреннего хранилища ресурсов;

– Белый список допустимых методов - для обеспечения того, чтобы допустимые методы HTTP были должным образом ограничены, в то время как другие будут возвращать правильный код ответа; и

– Хорошо известные атаки следует учитывать на этапе моделирования угроз в процессе проектирования, чтобы не допустить увеличения риска угроз. Угрозы и меры по их снижению, определенные в OWASP Top Ten Cheat Sheet[12], ДОЛЖНЫ быть приняты во внимание.

[RSG-122] При разработке API следует придерживаться стандартов и лучших практик, перечисленных ниже:

– Лучшие практики безопасного кодирования: Принципы безопасного кодирования OWASP;

– Безопасность Rest API: Памятка по безопасности REST;

– Экранирование ввода и защита от межсайтового скриптинга: OWASP XSS Cheat Sheet;

– Предотвращение SQL-инъекций: Памятка OWASP по SQL-иньекциям, памятка OWASP по Parameterization; и

– Безопасность транспортного уровня: Памятка OWASP по защите транспортного уровня.

[RSG-123] Тестирование безопасности и оценка уязвимостей ДОЛЖНЫ проводиться для обеспечения безопасности и устойчивости API к угрозам. Это требование МОЖЕТ быть выполнено путем использования статического и динамического тестирования безопасности приложений (SAST/DAST), автоматизированных инструментов управления уязвимостями и тестирования на проникновение.

Шифрование, целостность и безотказность

85. Защищенные сервисы должны быть обезопасены для защиты учетных данных аутентификации при передаче: например, паролей, API-ключей или JSON Web Tokens. Также должна быть гарантирована целостность передаваемых данных и неотрицание совершенных действий. Безопасные криптографические механизмы могут обеспечить конфиденциальность, шифрование, гарантию целостности и неотказуемость. Совершенная прямая секретность (perfect forward security — один из алгоритмов безопасности) является одним из средств обеспечения невозможности компрометации сеансовых ключей.

[12] https://www.owasp.org/index.php/Top_10-2017_Top_10

[RSG-124] Защищенные службы ДОЛЖНЫ предоставлять только конечные точки HTTPS, использующие TLS 1.2 или выше, с набором шифров, включающим ECDHE для обмена ключами.

[RSG-125] При рассмотрении протоколов аутентификации для обеспечения транспортной безопасности СЛЕДУЕТ использовать совершенную прямую секретность. Не СЛЕДУЕТ допускать использование небезопасных криптографических алгоритмов и обратную совместимость с SSL 3 и TLS 1.0/1.1.

[RSG-126] Для обеспечения максимальной безопасности и доверия следует устанавливать соединение через VPN IPSEC между сайтами для дополнительной защиты информации, передаваемой через незащищенные сети.

[RSG-127] Пользовательскому приложению СЛЕДУЕТ проверять цепочку сертификатов TLS при выполнении запросов к защищенным ресурсам, включая проверку списка отзыва сертификатов.

[RSG-128] В защищенных сервисах СЛЕДУЕТ использовать только действительные сертификаты, выданные доверенным центром сертификации (ЦС).

[RSG-129] Токены СЛЕДУЕТ подписывать с использованием безопасных алгоритмов подписи, соответствующих стандарту цифровой подписи (DSS) FIPS -186-4. Следует рассматривать алгоритм цифровой подписи RSA или алгоритм ECDSA.

Аутентификация и авторизация

86. Авторизация - это действие по осуществлению контроля доступа к ресурсу. Авторизация охватывает не только обеспечение контроля доступа, но и определение этого контроля. Сюда входят правила и политики доступа, которые должны определять необходимый уровень доступа, приемлемый как для провайдера, так и для потребляющего приложения. Основой управления доступом является предоставление или отказ провайдером потребляющему приложению и/или потребителю доступа к ресурсу с определенным уровнем детализации. Крупнозернистый доступ должен рассматриваться в точке запроса API или API-шлюза, а мелкозернистый контроль должен рассматриваться в бэкенд-сервисе, если это возможно. Можно рассмотреть модель управления доступом на основе ролей (RBAC) или модель управления доступом на основе атрибутов (ABAC).

87. Если сервис защищен, то Open ID Connect (метод аутентификации) следует предпочесть OAuth 2.0, поскольку он заполняет многие пробелы последнего и обеспечивает стандартизированный способ получения данных профиля владельца ресурса, стандартизированный формат токенов JSON Web Token (JWT) и криптографию. Не следует использовать другие схемы безопасности, такие как HTTP Basic Authorization, которая требует, чтобы клиент хранил пароль где-то открытым текстом для отправки вместе с каждым запросом. Кроме того, проверка этого пароля будет происходить медленнее, поскольку придется обращаться к хранилищу мандатов. OAuth 2.0 не определяет маркер безопасности. Поэтому следует использовать маркер JWT по сравнению, например, с SAML 2.0, который более подробен.

[RSG-130] Анонимная аутентификация ДОЛЖНА использоваться только тогда, когда клиенты и используемое ими приложение получают доступ к информации или функциям с низким уровнем чувствительности, которые не должны требовать аутентификации, например, к публичной информации.

[RSG-131] Аутентификация с использованием имени пользователя и пароля или хэша пароля НЕ ДОЛЖНА разрешаться.

[RSG-132] Если сервис защищен, СЛЕДУЕТ использоваться Open ID Connect.

[RSG-133] При использовании JSON Web Token (JWT) секрету JWT СЛЕДУЕТ обладать высокой энтропией, чтобы увеличить трудоемкость атаки грубой силой; токенам TTL и RTTL СЛЕДУЕТ быть как можно короче; и НЕ СЛЕДУЕТ хранить конфиденциальную информацию в полезной нагрузке JWT.

88. Общим выбором при проектировании системы безопасности является централизация аутентификации пользователей. Она должна храниться в Identity Provider (провайдерах идентификации - IdP) или локально в конечных точках REST.

89. Сервисы должны тщательно следить за тем, чтобы предотвратить утечку учетных данных. Пароли, маркеры безопасности и ключи API не должны появляться в URL, поскольку это может быть зафиксировано в журналах веб-сервера, что делает их ценными по своей сути. Пример как неправильно (API-ключ в URL): https://wipo.int/api/patents?apiKey=a53f435643de32.

[RSG-134] В запросах `POST/PUT` конфиденциальные данные СЛЕДУЕТ передавать в теле или в заголовках запроса.

[RSG-135] В запросах `GET` конфиденциальные данные СЛЕДУЕТ передавать в заголовке HTTP.

[RSG-136] Для того чтобы минимизировать задержки и уменьшить связь между защищенными сервисами, решение об управлении доступом СЛЕДУЕТ принимать на локальном уровне в конечных точках REST.

90. Аутентификация по API-ключам: API-ключи следует использовать там, где требуется аутентификация между системами, и они должны генерироваться автоматически и случайным образом. Риск, присущий этому режиму аутентификации, заключается в том, что любой, имеющий копию ключа API, может использовать его так, как будто он является легитимным приложением-потребителем. Следовательно, все коммуникации должны соответствовать RSG-124, чтобы защитить ключ при передаче. На разработчике приложения лежит ответственность за надлежащую защиту своей копии ключа API. Если ключ API встроен в потребляющее приложение, его можно декомпилировать и извлечь. Если он хранится в обычных текстовых файлах, его можно украсть и повторно использовать в злонамеренных целях. Поэтому API-ключ должен быть защищен хранилищем учетных данных или механизмом управления секретами. API-ключи могут использоваться для контроля использования сервисов даже для публичных сервисов.

[RSG-137] API-ключи СЛЕДУЕТ использовать и для защищенных и для публичных сервисов, чтобы не перегружать поставщика услуг многочисленными запросами (атаки типа "отказ в обслуживании"). Для защищенных сервисов API-ключи МОГУТ использоваться для монетизации (приобретенные тарифные планы), применения политики использования (QoS) и мониторинга.

[RSG-138] Ключи API МОГУТ быть объединены с заголовком HTTP запроса user-agent для различения пользователя-человека и программного агента, как указано в IETF RFC 7231.

[RSG-139] Сервис-провайдеру СЛЕДУЕТ возвращать вместе с заголовками ответа HTTP текущий статус использования. Следующие данные ответа МОГУТ быть возвращены:

- предел скорости (rate limit) - предел скорости (в минуту), установленный в системе;
- rate limit remaining - оставшееся количество запросов, разрешенных в течение текущего тайм-слота (-1 означает, что лимит превышен); и
- сброс лимита скорости (rate limit reset) - время (в секундах), оставшееся до сброса счетчика запросов.

[RSG-140] Сервис-провайдеру СЛЕДУЕТ возвращать код состояния "429 Too Many Requests", если запросы поступают слишком быстро.

[RSG-141] Ключи API ДОЛЖНЫ быть отозваны, если клиент нарушает соглашение об использовании, как указано в ОИС (Организации интеллектульной собственности).

[RSG-142] Ключи API СЛЕДУЕТ передавать с помощью пользовательских HTTP-заголовков. Их НЕ СЛЕДУЕТ передавать с помощью параметров запроса.

[RSG-143] Ключи API СЛЕДУЕТ генерировать случайным образом.

91. Хотя использование криптографии с открытым ключом и сертификатов сопряжено с накладными расходами, взаимную аутентификацию на основе сертификатов следует использовать в тех случаях, когда для обеспечения дополнительной безопасности Web API требуется более надежная аутентификация, чем та, которая обеспечивается ключами API. Безопасные и надежные сертификаты должны быть выданы взаимно доверенным центром сертификации (ЦС) в процессе установления доверия или перекрестной сертификации. Для снижения рисков безопасности идентификации, характерных для чувствительных систем и привилегированных действий, можно использовать строгую аутентификацию. Следует использовать общие для клиента и сервера сертификаты, например, X.509.

[RSG-144] Безопасные и доверенные сертификаты ДОЛЖНЫ выдаваться взаимно доверенным центром сертификации (ЦС) через процесс установления доверия или перекрестной сертификации.

[RSG-145] Сертификаты, совместно используемые клиентом и сервером, СЛЕДУЕТ использовать для снижения рисков безопасности идентификации, характерных для чувствительных систем и привилегированных действий, например, X.509.

[RSG-146] Для высокопривилегированных служб в двусторонней взаимной аутентификации между клиентом и сервером СЛЕДУЕТ использовать сертификаты для обеспечения дополнительной защиты.

[RSG-147] Для высокорисковых приложений, систем, обрабатывающих очень чувствительную информацию или выполняющих привилегированные действия СЛЕДУЕТ реализовать многофакторную аутентификацию для снижения рисков идентификации.

Доступность и защита от угроз

92. Доступность в этом контексте включает в себя защиту от угроз для минимизации времени простоя API, рассматривая, как можно уменьшить угрозы для открытых API, используя базовые принципы проектирования. Доступность также включает масштабирование в соответствии со спросом, обеспечение стабильности среды размещения и т. д. Эти уровни доступности обеспечиваются для всех аппаратных и программных стеков, поддерживающих доставку API. Доступность обычно рассматривается в соответствии со стандартами непрерывности бизнеса и аварийного восстановления, которые рекомендуют подход к оценке рисков для определения требований к доступности.

Междоменные запросы

93. Некоторые «междоменные» запросы, в частности Ajax-запросы, по умолчанию запрещены политикой безопасности одного и того же источника. В соответствии с политикой единого источника веб-браузер разрешает сценариям, содержащимся на первой веб-странице, доступ к данным на второй веб-странице, только если обе веб-страницы имеют одинаковое происхождение (т. е. сочетание схемы URI, имени хоста и номера порта).

94. Междоменный обмен ресурсами (CORS - Cross-Origin Resource Sharing) - это стандарт W3C, позволяющий гибко определять, какие междоменные запросы разрешены. Передавая соответствующие HTTP-заголовки междоменного обмена, ваш REST API сигнализирует браузеру, каким доменам или источникам разрешено выполнять JavaScript-запросы к REST-сервису.

95. JSON с дополнением (JSONP - JSON with padding) - это метод отправки данных JSON, не беспокоясь о проблемах междоменных запросов. Он вводит функции обратного вызова для загрузки данных JSON из разных доменов. Идея метода основана на том, что тег HTML `<script>` не подвержен влиянию политики происхождения. Все, что импортируется через этот тег, немедленно выполняется в глобальном контексте. Вместо того чтобы передавать файл JavaScript, можно передать URL-адрес службы, которая возвращает код JavaScript.

96. Для обхода этого ограничения обычно используются следующие подходы:

– JSONP - это обходной путь для междоменных запросов. Он не предлагает никакого механизма обнаружения ошибок, т.е. если возникла проблема, и сервис не сработал или ответил HTTP-ошибкой, нет способа определить, в чем была проблема на стороне клиента. В результате AJAX-приложение просто "зависнет". Более того, сайт, использующий JSONP, будет безоговорочно доверять JSON, предоставленному из другого домена;

– Iframe - это альтернативное обходное решение для междоменных запросов. Используя метод JavaScript `window.postMessage (message, targetOrigin)` на объекте iframe, можно передать запрос сайту другого домена. Подход iframe имеет хорошую совместимость даже в старых браузерах. Более того, он поддерживает только GET. Источник страницы Iframe всегда должен быть проверен из-за вопросов безопасности; и

– CORS - это стандартизированный подход для выполнения вызова к внешнему домену. Он может использовать `XMLHttpRequest` для отправки и получения данных и имеет лучший механизм обработки ошибок, чем JSONP. Он поддерживает множество типов авторизации по сравнению с JSONP, который поддерживает только куки. Он также поддерживает HTTP-методы по сравнению с JSONP, который поддерживает только `GET`. С другой стороны, не всегда возможно реализовать CORS, поскольку браузеры должны его поддерживать, а потребители API должны быть внесены в белый список CORS.

[RSG-148] Если REST API является публичным, HTTP-заголовок `Access-Control-Allow-Origin` ДОЛЖЕН быть установлен на '*'.

[RSG-149] Если REST API защищенный, СЛЕДУЕТ использовать CORS, если это возможно. В противном случае, JSONP МОЖЕТ использоваться в качестве запасного варианта, но только для GET-запросов, например, когда пользователь получает доступ с помощью старого браузера. НЕ СЛЕДУЕТ использовать Iframe.

Модель зрелости API

97. Обычно REST API классифицируют с помощью модели зрелости. Хотя существуют различные модели, настоящий стандарт ссылается на модель зрелости Ричардсона (RMM). RMM определяет три уровня, и настоящий стандарт рекомендует уровень 2 для REST API, поскольку уровень 3 сложен для реализации и требует значительных концептуальных и связанных с разработкой инвестиций от поставщиков и потребителей сервисов. В то же время, он не приносит немедленной выгоды потребителям сервисов.

98. Если Web API реализует уровень 3 RMM, необходимо внедрить гипермедийный формат. Hypertext Application Language (HAL)[13] прост и совместим с ответами JSON и XML. Однако он является лишь проектом рекомендации, наряду с другими гипермедийными форматами, такими как JSON-LD[14]. Следует использовать JSON-Schema[15], поскольку, хотя в настоящее время не существует спецификации для уровня 3 RMM, он считается наиболее

[13] https://tools.ietf.org/html/draft-kelly-json-hal-08t
[14] https://www.w3.org/TR/json-ld/
[15] https://json-schema.org/specification.html#specification-documents

зрелым. Не следует рассматривать следующие форматы гипермедиа: IETF RFC 5988 и `Collection+JSON`.

99. Рекомендуется, чтобы экземпляры, описанные схемой, предоставляли ссылку на загружаемую JSON-схему с использованием отношения ссылки "`describedby`", как определено в Linked Data Protocol 1.0, раздел 8.1 [W3C.REC-ldp-20150226].[16]

В HTTP такие ссылки могут быть прикреплены к любому ответу с помощью заголовка `Link` [**RFC8288**]. Примером такого заголовка может быть:

```
Link: <http://example.com/my-hyper-schema#>; rel="describedby"
```

[RSJ-150] При использовании экземпляров, описанных схемой, СЛЕДУЕТ использовать заголовок `Link` для предоставления ссылки на загружаемую JSON-схему В СООТВЕТСТВИИ С RFC8288.

[RSJ-151] Web API СЛЕДУЕТ реализовать, по крайней мере уровень 2 (свойства, присущие транспорту) RMM. Уровень 3 (Гипермедиа) МОЖЕТ быть реализован, чтобы сделать API полностью информативным.

100. Может быть разработан собственный формат гипермедиа. В этом случае рекомендуется набор атрибутов. Например:

```
{
   "link": {
      "href": "/patents",
      "rel": "self"
   },
   ...
 }
```

[RSJ-152] Для разработки пользовательского формата гипермедиа СЛЕДУЕТ использовать следующий набор атрибутов, заключенных в атрибут-ссылку:

- `href` - целевой URI;
- `rel` - значение целевого URI;
- `self` - URI ссылается на сам ресурс;
- `next` – URI ссылки на предыдущую страницу (если используется при пагинации);
- `previous` - URI ссылки на следующую страницу (если используется при постраничной разбивке); и

[16] http://json-schema.org/latest/json-schema-core.html#hypermedia

– Произвольное имя v обозначает пользовательское значение отношения.

WEB API SOAP

101. Настоящий стандарт рекомендует архитектурный стиль REST как предпочтительный подход к проектированию API. Архитектуры RESTful обычно проще в разработке, расширении и интеграции, чем SOAP. SOAP включен сюда для полноты; примеры и случаи использования не приводятся.

102. SOAP Web API - это программное приложение, идентифицируемое URI, интерфейсы и привязка которого могут быть определены, описаны и обнаружены с помощью XML-сущностей. Оно также поддерживает прямое взаимодействие с другими программными приложениями, используя сообщения на основе XML, через интернет-протоколы, такие как SOAP и HTTP.

103. Контракт на основе SOAP описывается в языке определения веб-службы (WSDL), стандартном документе W3C. В данном документе "документ WSDL контракта веб-службы" будет называться просто "WSDL".

104. При создании веб-сервисов существует два стиля разработки: контракт в последнюю очередь и контракт в первую очередь. При использовании подхода "контракт-последний" вы начинаете с программного кода и позволяете сгенерировать контракт веб-сервиса на его основе. При использовании подхода контракт-первый вы начинаете с контракта WSDL и используете код для реализации этого контракта.

Общие правила

105. Профиль взаимодействия веб-служб (WS-I) является одним из наиболее важных стандартов в отношении API-интерфейсов на основе SOAP., и он обеспечивает минимальную основу для написания веб-сервисов, которые могут работать вместе. WS-I предоставляет руководство по тому, как сервисы "открываются" друг другу и как они передают информацию (это называется "обмен сообщениями"). Это профиль для реализации конкретных версий некоторых из наиболее важных стандартов веб-служб, таких как WSDL, SOAP, XML и т.д. Присоединение к определенным профилям неявно указывает на присоединение к конкретным версиям этих стандартов Web-сервисов. WS-I Basic Profile v1.1 предоставляет руководство по использованию XML 1.0, HTTP 1.1, UDDI, SOAP 1.1, WSDL 1.1 и UDDI 2.0. WS-I Basic Profile 2.0 предоставляет руководство по использованию SOAP 1.2, WSDL 1.1, UDDI 2.0, WS-Addressing и MTOM. SOAP 1.2 обеспечивает четкую модель обработки и ведет к лучшей функциональной совместимости. WSDL 2.0 был разработан для решения проблем совместимости, обнаруженных в WSDL 1.1, путем использования улучшенных привязок SOAP 1.2.

[WS-01] Все WSDL ДОЛЖНЫ соответствовать WS-I Basic Profile 2.0 (базовый профиль WS-I). WSDL 1.2 МОЖЕТ быть использован.

106. Связка WSDL SOAP может быть либо связкой в стиле удаленного вызова процедур (RPC), либо связкой в стиле документа. Привязка SOAP также может иметь кодированное использование или буквальное использование. Это дает вам пять моделей стиля/использования: RPC/encoded, RPC/literal, document/encoded, document/literal,

document/literal wrapped (RPC/закодированный, RPC/литеральный, документ/закодированный, документ/литеральный, документ/литеральный обернутый).

[WS-02] Сервисы ДОЛЖНЫ использовать привязку в стиле документа и литеральные модели использования (либо document/literal либо document/literal wrapped). При наличии графов ДОЛЖЕН использоваться стиль RPC/encoded.

[WS-03] Когда существуют исключительные случаи использования, например, когда в WSDL есть перегруженные операции, СЛЕДУЕТ применять все остальные стили.

107. Конкретный WSDL должен быть отделен от абстрактного WSDL, чтобы обеспечить более модульный и гибкий интерфейс. Абстрактный WSDL определяет типы данных, сообщения, операции и тип порта. Конкретный WSDL определяет привязку, порт и сервис.

[WS-04] WSDL СЛЕДУЕТ разделить на абстрактную и конкретную части.

[WS-05] Все типы данных СЛЕДУЕТ определить в XSD-файле и импортировать в абстрактную WSDL.

[WS-06] Конкретный WSDL ДОЛЖЕН определять только один сервис на одном порте.

Схемы (XML)

108. Схемы, используемые в WSDL, должны соответствовать стандарту ВОИС ST.96. Для целей повторного использования и модульности схема должна быть отдельным документом, который либо включается, либо импортируется в WSDL, вместо того, чтобы определять ее непосредственно в WSDL. Это позволит вносить изменения в структуру XML без изменения WSDL.

[WS-07] Схема, определенная в элементе `wsdl:types`, ДОЛЖНА быть импортирована из самостоятельного файла схемы, чтобы обеспечить модульность и повторное использование.

[WS-08] Импорт внешней схемы ДОЛЖЕН быть реализован с помощью техники `xsd:import`, а не `xsd:include`.

[WS-09] Элемент `xsd:any` НЕ ДОЛЖЕН использоваться для указания корневого элемента в теле сообщения.

[WS-10] Целевое пространство имен для WSDL (атрибут `targetNamespace` на `wsdl:definitions`) ДОЛЖНО отличаться от целевого пространства имен схемы (атрибут `targetNamespace` на `xsd:schema`).

[WS-11] Запросы и ответы (соглашение об именовании, формат сообщений, структура данных и словарь данных) СЛЕДУЕТ брать из стандарта ВОИС ST.96.

Именование и версионирование

109. При именовании служб и элементов WSDL также следует применять соответствующие соглашения об именовании. Соглашениям об именовании следует соответствовать тем, которые реализованы в стандарте ВОИС ST.96.

[WS-12] Сервисы ДОЛЖНЫ называться в верхнем регистре (UpperCamelCase) и иметь суффикс 'Service', например, `https://wipo.int/PatentsService`.

[WS-13] Элементы WSDL message, part, portType, operation, input, output и binding СЛЕДУЮТ наименонованиям в верхнем регистре (UpperCamelCase).

[WS-14] Именам сообщений запроса СЛЕДУЕТ давать суффикс 'Request'.

[WS-15] Именам сообщений ответа СЛЕДУЕТ давать суффикс 'Response'.

[WS-16] Именам операций СЛЕДУЕТ соответствовать формату `<Verb><Object>{<Qualifier>}`, где `<Verb>` указывает операцию (предпочтительно Get, Create, Update или Delete, где это применимо) над `<Object>` операции, необязательная вариантивная часть в конце следует за `<Qualifier>` у `<Object>`.

110. Все названия операций состоят как минимум из двух частей. Необязательная третья часть может быть включена для дальнейшего прояснения и/или уточнения деловой цели операции. Три части: `<Verb> <Object> <Qualifier - необязательно>`. Каждая часть будет подробно описана ниже.

Глагол (Verb) - Имя каждой операции начинается с глагола. Примеры глаголов, которые обычно используются, описаны ниже:

Глагол	**Описание**	**Пример**
Get	Получить отдельный объект	GetBibData
Create	Получить новый объект	CreateBibData
Update	Обновление объекта	UpdateBibData
Delete	Удалить объект	DeleteCustomer

Объект (object)- существительное, следующее за глаголом, представляет собой краткое и однозначное описание бизнес-функции, которую обеспечивает операция. Цель состоит в том, чтобы предоставить потребителям лучшее понимание того, что делает операция, без двусмысленности. Учитывая, что определение некоторых

сущностей не является общим для различных подразделений, объект может быть составным полем, где первым узлом является подразделение, а вторым - сущность, например, `PatentCustomer`.

Квалификатор (qualifier) - Цель атрибута классификатора объекта (необязательного) заключается в том, чтобы уточнить деловую или предметную область, например, `GetCustomerList`. `Get` обозначает операцию, которая должна быть выполнена над клиентом, а `List` дополнительно описывает тот факт, что намерение состоит в получении списка клиентов, а не только одного клиента, как в `GetCustomer`.

111. В соответствии с принципами сервис-ориентированного проектирования, поставщики и потребители сервисов должны развиваться независимо друг от друга. Потребитель не должен быть затронут незначительными (обратно совместимыми) изменениями поставщика сервисов. Поэтому при версионировании сервисов следует использовать только основные номера версий. Для внутренних API (например, для разработки и тестирования) могут также использоваться минорные версии, например, Semantic Versioning (семантическое версионирование).

[WS-17] Имени файла WSDL СЛЕДУЕТ соответствовать следующему образцу: `<имя службы>_V<основной номер версии>. (<service name>_V<major version number> )`

[WS-18] Пространству имен WSDL-файла СЛЕДУЕТ содержать версию сервиса; например, `https://wipo.int/PatentsService/V1".`

112. Описание сервиса и его операций предоставляется в виде документации WSDL.

[WS-19] Элемент `wsdl:documentation` СЛЕДУЕТ использовать в WSDL с описанием сервиса (как первый дочерний элемент `wsdl:definitions` в WSDL) и его операций.

Разработка контрактов веб-сервисов

113. В контракт веб-сервиса следует включать технический интерфейс, состоящий из языка определения веб-сервиса (WSDL), определений XML-схемы, описания WS-политики, а также нетехнический интерфейс, состоящий из одного или нескольких документов описания сервиса.

114. WSDL, часть "Контракта сервиса", должна быть спроектирована до разработки кода. Никакой WSDL никогда не следует автоматически генерировать из кода. Девиз: "Сначала контракт", а НЕ "Сначала код". Все контракты веб-сервисов должны соответствовать базовому профилю взаимодействия веб-сервисов (WS-I BP). Любой проект, который автогенерируется из кода, будет обязан внести поправки для обеспечения соответствия этим стандартам.

Прикрепление политик к определениям WSDL

115. Контракты веб-сервисов могут быть расширены политиками безопасности, которые выражают дополнительные ограничения, требования и качества, обычно относящиеся к

поведению сервисов. Политики безопасности могут быть человекочитаемыми и становиться частью дополнительного соглашения об уровне сервиса, или могут быть машиночитаемыми, обрабатываемыми во время выполнения. Машиночитаемые политики определяются с помощью языка WS-Policy и соответствующих спецификаций WS-Policy.

[WS-20] Выражения политики ДОЛЖНЫ быть выделены в отдельный документ определения WS-Policy, на который затем делается ссылка в документе WSDL через элемент `wsp:PolicyReference`.

[WS-21] Глобальные или специфические для домена политики СЛЕДУЕТ изолировать и применять к нескольким службам.

[WS-22] РЕКОМЕНДУЕТСЯ, что точки вложения политики должны соответствовать WSDL 1.1 или более поздней версии, предпочтительно версии 2.0, элементам точки вложения и соответствующим субъектам политики (сервис, конечная точка, операция и сообщение).

SOAP - Безопасность веб-служб

116. Безопасность веб-сервисов (WSS - Web Services Security): безопасность сообщений SOAP (SOAP Message Security) - это набор усовершенствований для обмена сообщениями SOAP, обеспечивающих целостность и конфиденциальность сообщений. Является расширяемым и может использовать различные модели безопасности и технологии шифрования. Предоставляет три основных механизма, которые могут использоваться независимо или вместе:

- Возможность отправки токенов безопасности как части сообщения и ассоциирования токенов безопасности с содержимым сообщения;
- Способность защитить содержимое сообщения от несанкционированного и необнаруженного изменения (целостность сообщения); и
- Способность защитить содержимое сообщения от несанкционированного раскрытия (конфиденциальность сообщения).

WSS: SOAP Message Security может использоваться в сочетании с другими расширениями веб-служб и протоколами, специфичными для приложений, для удовлетворения различных требований безопасности.

[WS-23] Веб-сервисы, использующие сообщение SOAP, СЛЕДУЕТ защищать в соответствии с рекомендациями стандарта WSS:SOAP.

ФОРМАТЫ ТИПОВ ДАННЫХ

117. Настоящий стандарт рекомендует форматы примитивных типов данных, таких как время, дата и язык, в соответствии с рекомендациями стандартов ВОИС ST.96 и ST.97, которые используются для запросов и ответов XML и JSON соответственно и для параметров запроса.

[CS-01] Запись времени ДОЛЖНА быть отформатирована так, как указано в IETF RFC 3339 (это профиль ISO 8601).

[CS-02] Информацию о часовом поясе СЛЕДУЕТ использовать как указано в IETF RFC 3339. Например: `20:54:21+00:00`

[CS-03] Дата ДОЛЖНА быть отформатирована, как указано в IETF RFC 3339 (это профиль ISO 8601). Например: `2018-10-19`

[CS-04] Объекты Datetime (т.е. штампы времени) ДОЛЖНЫ быть отформатированы так, как указано в IETF RFC 3339 (это профиль ISO 8601).

[CS-05] Соответствующий часовой пояс СЛЕДУЕТ использовать, как указано в IETF RFC 3339. Например: `2017-02-14T20:54:21+00:00`

[CS-06] Для кодов валют ДОЛЖЕН использоваться ISO 4217-Alpha (3-буквенные коды валют). Точность значения (т.е. количество цифр после десятичной точки) МОЖЕТ варьироваться в зависимости от деловых требований.

[CS-07] Двухбуквенные коды в стандарте ВОИС ST.3 должны использоваться для представления ОИС (Организации интеллектуальной собственности), государств, других образований, организаций и для приоритетных и назначенных стран/организаций.

[CS-08] Элементы кода ISO 3166-1-альфа-2 (2-буквенные коды стран) ДОЛЖНЫ использоваться для представления названий стран, зависимых территорий и других областей, представляющих особый геополитический интерес, на основе списков названий стран, полученных от Организации Объединенных Наций.

[CS-09] ISO 639-1 (2-буквенные коды языков) ДОЛЖНЫ использоваться для кодов языков.

[CS-10] Для единиц измерения СЛЕДУЕТ использовать единицы измерения, как описано в "Едином кодексе единиц измерения" (основанном на определениях ISO 80000). Например, `для измерения веса с использованием килограммов (кг)`

[CSJ-11] Символы, используемые в значениях перечислений, ДОЛЖНЫ быть ограничены следующим набором: {a-z, A-Z, 0-9, точка (.), запятая (,), пробелы ( ), тире (-) и знак подчеркивания (_).

[CSJ-12] Термины представления в Приложении VI ДОЛЖНЫ использоваться для атомарных имен свойств.

[CSJ-13] Аббревиатуры и сокращения, появляющиеся в начале имени свойства, ДОЛЖНЫ быть в нижнем регистре. В остальных случаях все значения перечисления, акронимы и аббревиатуры ДОЛЖНЫ быть в верхнем регистре.

СООТВЕТСТВИЕ (CONFORMANCE)

118. Настоящий стандарт разработан как набор правил проектирования и соглашений, которые могут быть наложены поверх существующих или новых API веб-сервисов для обеспечения общей функциональности. Не все сервисы будут поддерживать все соглашения,

определенные в стандарте, из-за деловых (например, QoS может не требоваться) или технических ограничений (например, OAuth 2.0 может уже использоваться).

119. Настоящий стандарт определяет два уровня соответствия: A и AA уровни соответствия. Обратите внимание, что правила, указанные МОЖНО (MAY), не считаются важными при определении соответствия.

120. API веб-сервисам рекомендуется поддерживать столько дополнительных функций, выходящих за рамки их уровня соответствия, сколько необходимо для предполагаемого сценария.

121. Определены два уровня соответствия:

- **Уровень A**: Для соответствия уровню A, API показывает, что соблюдаются требуемые общие правила проектирования (RSG), которые в настоящем стандарте обозначены как "ДОЛЖЕН (MUST)". Кроме того, должны быть соблюдены правила, специфичные для типа возвращаемого ответа. Другими словами, указаны следующие подуровни соответствия:
    - Уровень AJ: возврат ответа в формате ST.97 JSON должен соответствовать всем правилам общего уровня (RSG), определенным как ДОЛЖЕН, а также всем правилам, специфичным для JSON (RSJ), определенным как ДОЛЖЕН;
    - Уровень AX: при возврате экземпляра ST.96 XML необходимо соответствовать всем правилам общего уровня (RSG), определенным как ДОЛЖЕН, а также всем правилам, специфичным для XML (RSX), определенным как ДОЛЖЕН; и
    - Уровень A: возвращая ответ в формате JSON или XML, необходимо соответствовать всем правилам общего уровня (RSG), обозначенным как ДОЛЖЕН, а также всем правилам для JSON (RSJ), обозначенным как ДОЛЖЕН, и всем правилам для XML (RSX), обозначенным как ДОЛЖЕН.
- **Уровень AA:** Для соответствия уровню AA API показывает, что он соответствует уровню A, а также соблюдены все рекомендуемые правила проектирования, которые в настоящем стандарте обозначены как "СЛЕДУЕТ (РЕКОМЕНДОВАНО)". Как и в случае с уровнем A, существуют подуровни, зависящие от типа ответа:
    - Уровень AAJ: соответствие уровню AJ, а также рекомендуемым правилам СЛЕДУЕТ, применимым к ответу JSON;

- Уровень AAX: соответствие уровню AX, а также рекомендуемым правилам СЛЕДУЕТ, применимым к XML-ответу.

122. Матрица прослеживаемости между правилами проектирования и уровнями соответствия приведена в Приложении I.

ССЫЛКИ

Стандарты ВОИС

Стандарт ВОИС ST.3 Рекомендуемый стандарт на двубуквенные коды для представления стран, административных единиц и межправительственных организаций

Стандарт ВОИС ST.96 Обработка информации о промышленной собственности с использованием XML

Стандарт ВОИС ST.97 Обработка информации о промышленной собственности с использованием JSON

Стандарты и конвенции

IEFT RFC 2119 Ключевые слова для использования в RFC для обозначения уровней требований - www.ietf.org/rfc/rfc2119.txt

IEFT RFC 3339 Дата и время в Интернете: штампы времени - www.ietf.org/rfc/rfc3339.txt

IEFT RFC 3986 Унифицированный идентификатор ресурса (URI): Общий синтаксис - www.ietf.org/rfc/rfc3986.txt

IEFT RFC 5789 Метод PATCH для HTTP - https://tools.ietf.org/rfc/rfc5789.txt

IEFT RFC 5988 Web Linking - https://tools.ietf.org/rfc/rfc5988.txt

IEFT RFC 6648 Избавление от префикса "X-" Префикс и подобные конструкции в прикладных протоколах - https://tools.ietf.org/rfc/rfc6648.txt

IEFT RFC 6750 Механизм авторизации OAuth 2.0: Использование токена носителя - https://tools.ietf.org/rfc/rfc6750.txt

IEFT RFC 7231 Протокол передачи гипертекста (HTTP/1.1): Семантика и содержание - www.ietf.org/rfc/rfc7231.txt

IEFT RFC 7232 Протокол передачи гипертекста (HTTP/1.1) - Условные запросы www.ietf.org/rfc/rfc7232.txt

IEFT RFC 7234 Протокол передачи гипертекста (HTTP/1.1) - Кэширование www.ietf.org/rfc/rfc7234.txt

IEFT RFC 7386	JSON Merge Patch - www.ietf.org/rfc/rfc7386.txt.
IEFT RFC 7240	Предпочтительный заголовок для HTTP - https://tools.ietf.org/rfc/rfc7240.txt
IEFT RFC 7519	JSON Web Token - www.ietf.org/rfc/rfc7519.txt
IEFT RFC 7540	Протокол передачи гипертекста версии 2 (HTTP/2) - https://tools.ietf.org/html/rfc7540
IEFT BCP-47	Теги для идентификации языков - https://tools.ietf.org/rfc/bcp/bcp47.txt
ISO 639-1	Языковые коды - https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes
ISO 3166-1 alpha-2	Двухбуквенные аббревиатуры для кодов стран - https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2
ISO 3166-1 alpha-3	Трехбуквенные аббревиатуры для кодов стран - https://en.wikipedia.org/wiki/ISO_3166-1_alpha-3
ISO 4217	Коды валют - www.iso.org/iso/home/standards/currency_codes.htm
ISO 8601	Форматы даты и времени - https://en.wikipedia.org/wiki/ISO_8601
OData	https://www.odata.org/

OASIS OData Metadata Service Entity Model - http://docs.oasis-open.org/odata/odata/v4.0/os/models/MetadataService.edmx

OASIS OData JSON Format Version 4.0. Под редакцией Ральфа Хандля, Майкла Пиццо и Марка Биамонте. Последняя версия - http://docs.oasis-open.org/odata/odata-json-format/v4.0/odata-json-format-v4.0.html

OASIS OData Atom Format Version 4.0. Под редакцией Мартина Цурмуэля, Майкла Пиццо и Ральфа Хандля. Последняя версия - http://docs.oasis-open.org/odata/odata-atom-format/v4.0/odata-atom-format-v4.0.html

OASIS OData "OData Version 4.0

- Part 1: Протокол - http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.html
- Part 2: URL Conventions - http://docs.oasis-open.org/odata/odata/v4.0/os/part2-url-conventions/odata-v4.0-os-part2-url-conventions.html

– Part 3: Common Schema Definition Language (CSDL) - http://docs.oasis-open.org/odata/odata/v4.0/os/part3-csdl/odata-v4.0-os-part3-csdl.html

Компоненты OASIS ABNF: Правила построения OData ABNF версии 4.0 и тестовые примеры OData ABNF - http://docs.oasis-open.org/odata/odata/v4.0/os/abnf/.

Компоненты OASIS Vocabulary: OData Core Vocabulary, OData Measures Vocabulary и OData Capabilities Vocabulary - http://docs.oasis-open.org/odata/odata/v4.0/os/vocabularies/.

XML-схемы OASIS:

OData EDMX XML Schema и OData EDM XML Schema- http://docs.oasis-open.org/odata/odata/v4.0/os/schemas/.

OASIS SAML 2.0 http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html

RAML (ReSTful API Modeling Language) - http://raml.org

Инициатива OpenAPI www.openapis.org

Модель зрелости REST API Ричардсона (RMM) - https://martinfowler.com/articles/richardsonMaturityModel.html.

HAL http://stateless.co/hal_specification.html

JSON-LD https://json-ld.org

Коллекция+JSON - Формат документов http://amundsen.com/media-types/collection/format/

BadgerFish http://badgerfish.ning.com/

Семантическое версионирование (Semantic Versioning) https://semver.org/

REST https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

CQL https://en.wikipedia.org/wiki/Contextual_Query_Language

Z39.50 https://www.loc.gov/z3950/agency/Z39-50-2003.pdf

WS-I Basic Profile 2.0 http://ws-i.org/profiles/basicprofile-2.0-2010-11-09.html

W3C SOAP 1.2 Часть 1 Структура обмена сообщениями - https://www.w3.org/TR/soap12-part1/

W3C SOAP 1.2 Часть 2 Адъюнкты - https://www.w3.org/TR/soap12-part2/

W3C WSDL версии 2.0 Часть 1 Основной язык - https://www.w3.org/TR/wsdl20/

W3C CORS https://www.w3.org/TR/cors/

Матричные параметры W3C	https://www.w3.org/DesignIssues/MatrixURIs.html

REST API ведомств ИС

ЕПВ Open Patent Services	OPS v 3.2 https://developers.epo.org
USPTO PatentsView	http://www.patentsview.org/api/doc.html
ВОИС - ePCTv1.1	https://pct.wipo.int/
EUIPO TMview, Designview, TMclass	http://www.tm-xml.org/TM-XML/TM-XML_xml/TM-XML_TM-Search.xml

Отраслевые REST API и рекомендации по проектированию

Facebook	https://developers.facebook.com/docs/graph-api/reference
GitHub	https://developer.github.com/v3
Руководство по проектированию API Google	https://cloud.google.com/apis/design/
Azure	https://docs.microsoft.com/en-us/rest/api/
OpenAPI	https://swagger.io/docs/specification/about/
Odata	http://www.odata.org/documentation/
JSON API	http://jsonapi.org/format/
Microsoft API Design	https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
JIRA REST API	https://developer.atlassian.com/server/jira/platform/jira-rest-api-examples
Confluece REST API	https://developer.atlassian.com/server/confluence/
API Ebay	https://developer.ebay.com/api-docs/static/ebay-rest-landing.html
Oracle REST Data Services	http://www.oracle.com/technetwork/developer-tools/rest-data-services/overview/index.html
PayPal REST API	https://developer.paypal.com/docs/api/overview/
Данные в Интернете Лучшие практики	https://www.w3.org/TR/dwbp/#intro

Руководство SAP по будущей гармонизации REST API
https://d.dam.sap.com/m/xAUymP/54014_GB_54014_enUS.pdf

API GitHub https://developer.github.com/v3/

Zalando https://github.com/zalando/ReSTful-api-guidelines

Dropbox https://www.dropbox.com/developers

Twitter https://developer.twitter.com/en/docs

Другие

CQRS https://martinfowler.com/bliki/CQRS.html

МСЭ (ITU) https://www.itu.int/en/ITU-T/ipr/Pages/open.aspx

Памятка OWASP по безопасности REST - https://www.owasp.org/index.php/REST_Security_Cheat_Sheet

DDD https://martinfowler.com/bliki/BoundedContext.html

Принципы REST https://en.wikipedia.org/wiki/Representational_state_transfer

Принцип открытости/закрытости https://en.wikipedia.org/wiki/Open/closed_principle

Какой стиль WSDL следует использовать? https://www.ibm.com/developerworks/library/ws-whichwsdl/

Правительство Новой Зеландии

Стандарт API и Руководство https://www.ict.govt.nz/guidance-and-resources/standards-compliance/api-standard-and-guidelines/

Памятка по предотвращению межсайтового скриптинга https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

Серия памяток OWASP https://cheatsheetseries.owasp.org/

Стандарт цифровой подписи (DSS) https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.186-4.pdf

SOAP Message Security 1.0, OASIS Standard 200401 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soapmessage-security-1.0.pdf

SOA Принципы проектирования услуг, Томас Эрл (2008)

[Приложение I к ST.90 следует].

ПРИЛОЖЕНИЕ I

СПИСОК ПРАВИЛ И СОГЛАШЕНИЙ ПО РАЗРАБОТКЕ RESTFUL WEB-СЕРВИСОВ

Версия 1.1

Пересмотр одобрен Комитетом по Стандартам ВОИС (КСВ)
на своей десятой сессии 25 ноября 2022

В следующих таблицах обобщены правила и соглашения по проектированию сервисов, а также определены основные требования соответствия с точки зрения того, какой уровень соответствия поддерживает реализация Web Services API. Ниже приводится руководство к приведенным таблицам:

- В таблице 1 приведены правила, которые должны быть соблюдены для достижения уровня соответствия AJ (для ответа в формате JSON);

- Таблица 2 **Ошибка! Источник ссылки не найден.** содержит свод правил проектирования, которые должны быть соблюдены для достижения соответствия уровню AX (для ответа XML) ;

- Таблица 3 содержит сводку правил проектирования, которые должны быть соблюдены для достижения соответствия уровню AAJ (для ответа JSON); и

- Таблица 4 содержит правила проектирования, которые должны быть соблюдены для достижения соответствия уровню AAX (для ответа XML).

[Примечание редакции: Для достижения соответствия уровню A достаточно соблюдать правила, приведенные в таблицах 1 и 2. Для достижения соответствия уровню AA необходимо следовать правилам, приведенным в таблицах 3 и 4. Третья буква указывает на тип предоставленного ответа.]

Таблица 1: Таблицы соответствия для JSON содержимого в ответе

Идентификатор правила	Описание правил	**Перекрестные ссылки и примечания**
[RSG-01]	Символ прямой косой черты "/" ДОЛЖЕН использоваться в пути URI для указания иерархических отношений между ресурсами, но путь НЕ ДОЛЖЕН заканчиваться прямой косой чертой, так как это не дает никакого семантического значения и может привести к путанице.	AJ, AX, AAJ, AAX
[RSG-02]	Имена ресурсов ДОЛЖНЫ придерживаться своего шаблона именования.	AJ, AX, AAJ, AAX
[RSG-04	Параметры запроса ДОЛЖНЫ соответствовать своему шаблону именования	AJ, AX

[RSG-06]	Шаблон URL для Web API ДОЛЖЕН содержать слово "api" в URI.	AJ, AX, AAJ, AAX
[RSG-07]	Матричные параметры НЕ ДОЛЖНЫ использоваться.	AJ, AX, AAJ, AAX
[RSG-08]	Web API ДОЛЖЕН совместимым образом применять коды состояния HTTP, как описано в RFC IETF.	AJ, AX, AAJ, AAX
[RSG-10]	Если API обнаруживает недопустимые значения ввода, он ДОЛЖЕН вернуть код состояния HTTP "`400 Bad Request`". Содержимое ошибки ДОЛЖНО указывать на ошибочное значение.	AJ, AX, AAJ, AAX
[RSG-12]	Если API обнаруживает допустимые значения, которые требуют нереализованных функций, он ДОЛЖЕН вернуть код состояния HTTP "`501 Not Implemented`". Содержимое ошибки ДОЛЖНО указывать на не обработанное значение.	AJ, AX, AAJ, AAX
[RSG-14]	Если ресурс может быть самостоятельным, он ДОЛЖЕН быть ресурсом верхнего уровня, в противном случае - подресурсом.	AJ, AX, AAJ, AAX
[RSG-15]	Параметры запроса ДОЛЖНЫ использоваться вместо путей URL для получения вложенных ресурсов.	AJ, AX, AAJ, AAX
[RSG-18]	Имена ресурсов, сегментов и параметры запросов ДОЛЖНЫ состоять из слов на английском языке с использованием основных английских написаний, представленных в Оксфордском словаре английского языка. Имена ресурсов, которые локализованы в связи с требованиями бизнеса, МОГУТ быть на других языках.	AJ, AX, AAJ, AAX
[RSG-20]	Web API ДОЛЖЕН поддерживать согласование типов содержимого в соответствии с IETF RFC 7231.	AJ, AX, AAJ, AAX
[RSG-21]	Формат JSON ДОЛЖЕН предполагаться, если не запрашивается определенный тип содержимого.	AJ, AX, AAJ, AAX
[RSG-27]	Web API ДОЛЖЕН поддерживать как минимум XML или JSON.	AJ, AX, AAJ, AAX
[RSG-28]	Методы HTTP ДОЛЖНЫ быть ограничены стандартными методами HTTP `POST, GET, PUT, DELETE, OPTIONS, PATCH, TRACE и HEAD`, как указано в IETF RFC 7231 и 5789.	AJ, AX, AAJ, AAX
[RSG-33]	Для конечной точки, которая извлекает один ресурс, если ресурс не найден, метод `GET` ДОЛЖЕН возвращать код состояния "`404 Not Found`". Конечные точки, которые возвращают списки ресурсов, просто возвращают пустой список.	AJ, AX, AAJ, AAX
[RSG-34]	Если ресурс получен успешно, метод `GET` ДОЛЖЕН вернуть `200 OK`.	AJ, AX, AAJ, AAX
[RSG-35]	Запрос `GET` ДОЛЖЕН быть идемпотентным (независимым от количества выполнений запроса).	AJ, AX, AAJ, AAX
[RSG-37]	Запрос `HEAD` ДОЛЖЕН быть идемпотентным.	AJ, AX, AAJ, AAX
[RSG-39]	`POST`-запрос НЕ ДОЛЖЕН быть идемпотентным в соответствии с IETF RFC 2616.	AJ, AX, AAJ, AAX

[RSG-43]	Запрос `PUT` ДОЛЖЕН быть идемпотентным.	AJ, AX, AAJ, AAX

[RSG-44]	Если ресурс не найден, `PUT` ДОЛЖЕН вернуть код состояния "`404 Not Found`".	AJ, AX, AAJ, AAX
[RSG-45]	Если ресурс успешно обновлен, `PUT` ДОЛЖЕН вернуть код состояния "`200 OK`", если обновленный ресурс возвращен, или "`204 No Content`", если он не возвращен.	AJ, AX, AAJ, AAX
[RSG-46]	Запрос `PATCH` НЕ ДОЛЖЕН быть идемпотентным.	AJ, AX, AAJ, AAX
[RSG-48]	Если ресурс не найден, `PATCH` ДОЛЖЕН вернуть код состояния "`404 Not Found`".	AJ, AX, AAJ, AAX
[RSJ-49]	Если Web API реализует частичные обновления с помощью `PATCH`, он ДОЛЖЕН использовать формат JSON Merge Patch для описания частичного набора изменений, как описано в IETF RFC 7386, используя тип содержимого `application/merge-patch+json`.	AJ, AAJ
[RSG-50]	Запрос `DELETE` НЕ ДОЛЖЕН быть идемпотентным.	AJ, AX, AAJ, AAX
[RSG-51]	Если ресурс не найден, `DELETE` ДОЛЖЕН вернуть код состояния "`404 Not Found`".	AJ, AX, AAJ, AAX
[RSG-52]	Если ресурс удален успешно, `DELETE` ДОЛЖЕН вернуть статус "`200 OK`", если удаленный ресурс возвращен, или "`204 No Content`", если он не возвращен.	AJ, AX, AAJ, AAX
[RSG-53].	Конечным получателем является либо исходный сервер, либо первый прокси или шлюз, получивший в запросе значение Max-Forwards равное нулю. Запрос `TRACE` НЕ ДОЛЖЕН включать тело (запроса).	AJ, AX, AAJ, AAX
[RSG-54]	Запрос `TRACE` НЕ ДОЛЖЕН быть идемпотентным.	AJ, AX, AAJ, AAX
[RSG-55].	Значение поля заголовка `Via` HTTP ДОЛЖНО служить для отслеживания цепочки запросов.	AJ, AX, AAJ, AAX
[RSG-56]	Поле заголовка HTTP Max-Forwards ДОЛЖНО использоваться для того, чтобы клиент мог ограничить длину цепочки запросов.	AJ, AX, AAJ, AAX
[RSG-58].	Ответы на `TRACE` НЕ ДОЛЖНЫ кэшироваться.	AJ, AX, AAJ, AAX
[RSG-60]	Запрос `OPTIONS` ДОЛЖЕН быть идемпотентным.	AJ, AX, AAJ, AAX
[RSG-70]	Web API ДОЛЖЕН использовать параметры запроса для реализации пагинации (разбивки на страницы).	AJ, AX, AAJ, AAX
[RSG-71]	Веб-интерфейс НЕ ДОЛЖЕН использовать HTTP-заголовки для реализации пагинации.	AJ, AX, AAJ, AAX
[RSG-75]	Для указания критерия сортировки по нескольким атрибутам ДОЛЖЕН использоваться параметр запроса. Значение этого параметра представляет собой список ключей сортировки, разделенных запятыми, а направления сортировки либо 'asc' для восходящего, либо 'desc' для нисходящего МОГУТ быть добавлены к каждому ключу сортировки, разделенные символом двоеточия ':'. Направление по умолчанию ДОЛЖНО быть определено сервером в случае, если для ключа не указано направление сортировки.	AJ, AX, AAJ, AAX
[RSG-76]	Web API СЛЕДУЕТ возвращать критерии сортировки в	AJ, AX, AAJ, AAX

	ответе.	

[RSG-79]	Web API ДОЛЖЕН поддерживать возврат количества элементов в коллекции.	AJ, AX, AAJ, AAX
[RSG-80]	Параметр запроса ДОЛЖЕН использоваться для поддержки возврата количества элементов в коллекции.	AJ, AX, AAJ, AAX
[RSG-82]	Web API МОЖЕТ поддерживать возврат количества элементов в коллекции в режиме inline, т.е. как часть ответа, которая содержит саму коллекцию. При этом ДОЛЖЕН использоваться параметр запроса.	AJ, AX, AAJ, AAX
[RSG-86]	Сервисный (веб-сервисный) контракт ДОЛЖЕН определять поддерживаемую грамматику (например, поля, функции, ключевые слова и операторы).	AJ, AX, AAJ, AAX
[RSG-87]	ДОЛЖЕН использоваться параметр запроса "q".	AJ, AX, AAJ, AAX
[RSG-88]	На уровне протокола Web API ДОЛЖЕН возвращать соответствующий код состояния HTTP, выбранный из списка стандартных кодов состояния HTTP.	AJ, AX, AAJ, AAX
[RSJ-89]	На уровне приложения Web API ДОЛЖЕН возвращать часть, сообщающую об ошибке с достаточной степенью детализации. Атрибуты code и message являются обязательными, атрибут details является условно обязательным, атрибуты target, status, `moreInfo` и `internalMessage` являются необязательными.	AJ, AX, AAJ, AAX
[RSG-90]	Ошибки НЕ ДОЛЖНЫ раскрывать критически важные для безопасности данные или внутренние технические детали, такие как стеки вызовов в сообщениях об ошибках.	AJ, AX, AAJ, AAX
[RSG-91].	`Заголовок` HTTP: `Reason-Phrase` (описанный в RFC 2616) НЕ ДОЛЖЕН использоваться для передачи сообщений об ошибках.	AJ, AX, AAJ, AAX
[RSG-93].	В формат сервисного контракта СЛЕДУЕТ включать следующее: - Версия API; - Информация о семантике элементов API; - Ресурсы; - Атрибуты ресурса; - Параметры запроса; - Методы; - Типы носителей; - Грамматика поиска (если она поддерживается); - Коды состояния HTTP; - Методы HTTP; - Ограничения и отличительные особенности; и - Безопасность (если есть).	AJ, AX, AAJ, AAX
[RSG-95]	REST API ДОЛЖЕН предоставлять документацию API в виде сервисного контракта.	AJ, AX, AAJ, AAX

[RSG-96]	Реализация Web API, отклоняющаяся от настоящего стандарта, ДОЛЖНА быть явно документирована в сервисном контракте. Если правило отклонения не указано в сервисном контракте, ДОЛЖНО подразумеваться, что соблюдается настоящий стандарт.	AJ, AX, AAJ, AAX
[RSG-97]	Сервисный контракт ДОЛЖЕН позволять генерировать скелетный код клиента API.	AJ, AX, AAJ, AAX
[RSG-105]	Web API ДОЛЖЕН поддерживать кэширование результатов GET; Web API МОЖЕТ поддерживать кэширование результатов и других методов HTTP.	AJ, AX, AAJ
[RSG-113]	Если Web API поддерживает обработку настроек, номенклатура настроек, которые МОГУТ быть установлены с помощью заголовка `Prefer`, ДОЛЖНА быть записана в сервисном контракте.	AAJ, AAX, AJ, AX
[RSG-114]	Если Web API поддерживает локализованные данные, HTTP-заголовок запроса `Accept-Language` ДОЛЖЕН поддерживаться для указания набора естественных языков, которые предпочтительны в ответе, как указано в IETF RFC 7231.	AJ, AX, AAJ, AAX
[RSG-116]	Конфиденциальность: Различные API и информация в API ДОЛЖНЫ быть идентифицированы, классифицированы и защищены от несанкционированного доступа, раскрытия и перехвата в любое время. Должны соблюдаться принципы наименьших привилегий, нулевого доверия, необходимости знать и необходимости делиться.	AJ, AX, AAJ, AAX
[RSG-117]	Обеспечение целостности: Различные API и информация в API ДОЛЖНЫ быть защищены от несанкционированной модификации, дублирования, повреждения и уничтожения. Информация ДОЛЖНА модифицироваться через подтвержденные транзакции и интерфейсы. Системы ДОЛЖНЫ обновляться с использованием утвержденных процессов управления конфигурацией, управления изменениями и управления исправлениями.	AJ, AX, AAJ, AAX
[RSG-118]	Доступность: Различные API и информация в API ДОЛЖНЫ быть доступны авторизованным пользователям в установленное время в соответствии с соглашениями об уровне обслуживания (SLA), политиками контроля доступа и определенными бизнес-процессами.	AJ, AX, AAJ, AAX
[RSG-119]	Неотказуемость (безотзывность): Каждая обрабатываемая транзакция или действие, выполняемое API, ДОЛЖНЫ обеспечивать неотказуемость посредством реализации надлежащего аудита, авторизации, аутентификации, а также реализации безопасных путей, а также и сервисов и механизмов неотказуемости.	AJ, AX, AAJ, AAX

[RSG-120]	Аутентификация, авторизация, аудит: Пользователи, системы, API или устройства, вовлеченные в критические транзакции или действия, ДОЛЖНЫ быть аутентифицированы, авторизованы с помощью служб контроля доступа на основе ролей или атрибутов и	AJ, AX, AAJ, AAX
	поддерживать разделение обязанностей. Кроме того, все действия ДОЛЖНЫ регистрироваться, а надежность аутентификации должна увеличиваться в соответствии с соответствующим информационным риском.	
[RSG-121]	При разработке API ТРЕБУЕТСЯ тщательно учитывать угрозы, случаи вредоносного использования, методы безопасного кодирования, безопасность транспортного уровня и тестирование безопасности, особенно: - `PUTs и POSTs` - т.е.: какие изменения внутренних данных потенциально могут быть использованы для атаки или ложной информации; - `DELETES` - т.е.: может использоваться для удаления содержимого внутреннего хранилища ресурсов; - Белый список допустимых методов - для обеспечения того, чтобы допустимые методы HTTP были должным образом ограничены, в то время как другие будут возвращать правильный код ответа; и - Хорошо известные атаки следует учитывать на этапе моделирования угроз в процессе проектирования, чтобы не допустить увеличения риска угроз. Угрозы и меры по их снижению, определенные в OWASP Top Ten Cheat Sheet ДОЛЖНЫ быть приняты во внимание.	AJ, AX, AAJ, AAX
[RSG-122]	При разработке API следует придерживаться стандартов и лучших практик, перечисленных ниже: - Лучшие практики безопасного кодирования: Принципы безопасного кодирования OWASP; - Безопасность Rest API: Памятка по безопасности REST; - Экранирование ввода и защита от межсайтового скриптинга: OWASP XSS Cheat Sheet; - Предотвращение SQL-инъекций: Памятка OWASP по SQL-иньекциям, памятка OWASP по Parameterization; и - Безопасность транспортного уровня: Памятка OWASP по защите транспортного уровня.	AJ, AX, AAX, AAJ
[RSG-123]	Тестирование безопасности и оценка уязвимостей ДОЛЖНЫ проводиться для обеспечения безопасности и устойчивости API к угрозам. Это требование МОЖЕТ быть выполнено путем использования статического и динамического тестирования безопасности приложений (SAST/DAST), автоматизированных инструментов управления уязвимостями и тестирования на проникновение.	AJ, AX, AAX, AAJ

[RSG-124]	Защищенные службы ДОЛЖНЫ предоставлять конечные точки HTTPS только с использованием TLS 1.2 или выше с набором шифров, включающим ECDHE для обмена ключами.	AJ, AX, AAJ, AAX
[RSG-130]	Анонимная аутентификация ДОЛЖНА использоваться только тогда, когда клиенты и используемое ими приложение получают доступ к информации или функциям с низким уровнем чувствительности, которые не должны требовать аутентификации, например, к публичной информации.	AJ, AX, AAJ, AAX
[RSG-131]	Аутентификация с использованием имени пользователя и пароля или хэш-пароля НЕ ДОЛЖНА разрешаться.	AJ, AX, AAJ, AAX
[RSG-141]	Ключи API ДОЛЖНЫ быть отозваны, если клиент нарушает соглашение об использовании, как указано в ОИС (Организации интеллектульной собственности).	AJ, AX, AAJ, AAX
[RSG-144]	Безопасные и надежные сертификаты ДОЛЖНЫ выдаваться взаимно доверенным центром сертификации (ЦС) через процесс установления доверия или перекрестной сертификации.	AJ, AX, AAJ, AAX
[RSG-145]	Сертификаты, совместно используемые клиентом и сервером, СЛЕДУЕТ использовать для снижения рисков безопасности идентификации, характерных для чувствительных систем и привилегированных действий, например, X.509.	AJ, AX, AAJ, AAX
[RSG-148]	Если REST API является публичным, HTTP-заголовок Access-Control-Allow-Origin ДОЛЖЕН иметь значение '*'.	AJ, AX, AAJ, AAX

Таблица 2: Таблица соответствия для XML содержимого в ответе

Идентификатор правила	Описание правил	Перекрестные ссылки и примечания
[RSG-01]	Символ прямой косой черты "/" ДОЛЖЕН использоваться в пути URI для указания иерархических отношений между ресурсами, но путь НЕ ДОЛЖЕН заканчиваться прямой косой чертой, так как это не дает никакого семантического значения и может привести к путанице.	AJ, AX, AAJ, AAX
[RSG-02]	Имена ресурсов ДОЛЖНЫ придерживаться своего шаблона именования.	AJ, AX, AAJ, AAX
[RSG-04]	Параметры запроса ДОЛЖНЫ соответствовать своему шаблону именования	AJ, AX
[RSG-06]	Шаблон URL для Web API ДОЛЖЕН содержать слово "api" в URI.	AJ, AX, AAJ, AAX
[RSG-07]	Матричные параметры НЕ ДОЛЖНЫ использоваться.	AJ, AX, AAJ, AAX
[RSG-08]	Web API ДОЛЖЕН совместимым образом применять коды состояния HTTP, как описано в RFC IETF.	AJ, AX, AAJ, AAX
[RSG-10]	Если API обнаруживает недопустимые значения ввода, он ДОЛЖЕН вернуть код состояния HTTP "`400 Bad Request`". Содержимое ошибки ДОЛЖНО указывать на ошибочное значение.	AJ, AX, AAJ, AAX
[RSG-12]	Если API обнаруживает допустимые значения, которые требуют нереализованных функций, он ДОЛЖЕН вернуть код состояния HTTP "`501 Not Implemented`". Содержимое ошибки ДОЛЖНО указывать на не обработанное значение.	AJ, AX, AAJ, AAX
[RSG-14]	Если ресурс может быть самостоятельным, он ДОЛЖЕН быть ресурсом верхнего уровня, в противном случае - подресурсом.	AJ, AX, AAJ, AAX
[RSG-15]	Параметры запроса ДОЛЖНЫ использоваться вместо путей URL для получения вложенных ресурсов.	AJ, AX, AAJ, AAX
[RSG-18]	Имена ресурсов, сегментов и параметры запросов ДОЛЖНЫ состоять из слов на английском языке с использованием основных английских написаний, представленных в Оксфордском словаре английского языка. Имена ресурсов, которые локализованы в связи с требованиями бизнеса, МОГУТ быть на других языках.	AJ, AX, AAJ, AAX
[RSG-20]	Web API ДОЛЖЕН поддерживать согласование типов содержимого в соответствии с IETF RFC 7231.	AJ, AX, AAJ, AAX
[RSG-21]	Формат JSON ДОЛЖЕН предполагаться, если не запрашивается определенный тип содержимого.	AJ, AX, AAJ, AAX
[RSG-27]	Web API ДОЛЖЕН поддерживать как минимум XML или JSON.	AJ, AX, AAJ, AAX
[RSG-28]	Методы HTTP ДОЛЖНЫ быть ограничены стандартными методами HTTP `POST, GET, PUT, DELETE, OPTIONS, PATCH, TRACE и HEAD`, как указано в IETF RFC 7231 и 5789.	AJ, AX, AAJ, AAX

[RSG-33]	Для конечной точки, которая извлекает один ресурс, если ресурс не найден, метод `GET` ДОЛЖЕН вернуть код состояния "`404 Not Found`". Конечные точки, которые возвращают списки ресурсов, просто возвращают пустой список.	AJ, AX, AAJ, AAX
[RSG-34]	Если ресурс получен успешно, метод `GET` ДОЛЖЕН вернуть `200 OK`.	AJ, AX, AAJ, AAX
[RSG-35]	Запрос `GET` ДОЛЖЕН быть идемпотентным (независимым от количества выполнений запроса).	AJ, AX, AAJ, AAX
[RSG-37]	Запрос `HEAD` ДОЛЖЕН быть идемпотентным.	AJ, AX, AAJ, AAX
[RSG-39]	`POST`-запрос НЕ ДОЛЖЕН быть идемпотентным в соответствии с IETF RFC 2616.	AJ, AX, AAJ, AAX
[RSG-43]	Запрос `PUT` ДОЛЖЕН быть идемпотентным.	AJ, AX, AAJ, AAX
[RSG-44]	Если ресурс не найден, `PUT` ДОЛЖЕН вернуть код состояния "`404 Not Found`".	AJ, AX, AAJ, AAX
[RSG-45]	Если ресурс успешно обновлен, `PUT` ДОЛЖЕН вернуть код состояния "`200 OK`", если обновленный ресурс возвращен, или "`204 No Content`", если он не возвращен.	AJ, AX, AAJ, AAX
[RSG-46]	Запрос `PATCH` НЕ ДОЛЖЕН быть идемпотентным.	AJ, AX, AAJ, AAX
[RSG-48]	Если ресурс не найден, `PATCH` ДОЛЖЕН вернуть код состояния "`404 Not Found`".	AJ, AX, AAJ, AAX
[RSG-50]	Запрос `DELETE` НЕ ДОЛЖЕН быть идемпотентным.	AJ, AX, AAJ, AAX
[RSG-51].	Если ресурс не найден, `DELETE` ДОЛЖЕН вернуть код состояния "`404 Not Found`".	AJ, AX, AAJ, AAX
[RSG-52]	Если ресурс удален успешно, `DELETE` ДОЛЖЕН вернуть статус "`200 OK`", если удаленный ресурс возвращен, или "`204 No Content`", если он не возвращен.	AJ, AX, AAJ, AAX
[RSG-53].	Конечным получателем является либо исходный сервер, либо первый прокси или шлюз, получивший в запросе значение Max-Forwards равное нулю. Запрос `TRACE` НЕ ДОЛЖЕН включать тело (запроса).	AJ, AX, AAJ, AAX
[RSG-54]	Запрос `TRACE` НЕ ДОЛЖЕН быть идемпотентным.	AJ, AX, AAJ, AAX
[RSG-55].	Значение поля заголовка `Via` HTTP ДОЛЖНО служить для отслеживания цепочки запросов.	AJ, AX, AAJ, AAX
[RSG-56]	Поле заголовка HTTP Max-Forwards ДОЛЖНО использоваться для того, чтобы клиент мог ограничить длину цепочки запросов.	AJ, AX, AAJ, AAX
[RSG-58]	Ответы на `TRACE` НЕ ДОЛЖНЫ кэшироваться.	AJ, AX, AAJ, AAX
[RSG-60]	Запрос `OPTIONS` ДОЛЖЕН быть идемпотентным.	AJ, AX, AAJ, AAX
[RSG-70]	Web API ДОЛЖЕН использовать параметры запроса для реализации пагинации (разбивки на страницы).	AJ, AX, AAJ, AAX
[RSG-71]	Веб-интерфейс НЕ ДОЛЖЕН использовать HTTP-заголовки для реализации пагинации.	AJ, AX, AAJ, AAX

[RSG-75]	Для указания критерия сортировки по нескольким атрибутам ДОЛЖЕН использоваться параметр запроса. Значение этого параметра представляет собой список ключей сортировки, разделенных запятыми, а направления сортировки либо 'asc' для восходящего, либо 'desc' для нисходящего МОГУТ быть добавлены к каждому ключу сортировки, разделенные символом двоеточия ':'. Направление по умолчанию ДОЛЖНО быть определено сервером в случае, если для ключа не указано направление сортировки.	AJ, AX, AAJ, AAX
[RSG-76]	Web API СЛЕДУЕТ возвращать критерии сортировки в ответе.	AJ, AX, AAJ, AAX
[RSG-79]	Web API ДОЛЖЕН поддерживать возврат количества элементов в коллекции.	AJ, AX, AAJ, AAX
[RSG-80]	Параметр запроса ДОЛЖЕН использоваться для поддержки возврата количества элементов в коллекции.	AJ, AX, AAJ, AAX
[RSG-82]	Web API МОЖЕТ поддерживать возврат количества элементов в коллекции в режиме inline, т.е. как часть ответа, которая содержит саму коллекцию. При этом ДОЛЖЕН использоваться параметр запроса.	AJ, AX, AAJ, AAX
[RSG-86]	Сервисный (веб-сервисный) контракт ДОЛЖЕН определять поддерживаемую грамматику (например, поля, функции, ключевые слова и операторы).	AJ, AX, AAJ, AAX
[RSG-87]	ДОЛЖЕН использоваться параметр запроса "q".	AJ, AX, AAJ, AAX
[RSG-88]	На уровне протокола Web API ДОЛЖЕН возвращать соответствующий код состояния HTTP, выбранный из списка стандартных кодов состояния HTTP.	AJ, AX, AAJ, AAX
[RSG-89]	На уровне приложения Web API ДОЛЖЕН возвращать часть, сообщающую об ошибке с достаточной степенью детализации. Атрибуты code и message являются обязательными, атрибут details является условно обязательным, атрибуты target, status, `moreInfo` и `internalMessage` являются необязательными.	AJ, AX, AAJ, AAX
[RSG-90]	Ошибки НЕ ДОЛЖНЫ раскрывать критически важные для безопасности данные или внутренние технические детали, такие как стеки вызовов в сообщениях об ошибках.	AJ, AX, AAJ, AAX
[RSG-91].	`Заголовок` HTTP: `Reason-Phrase` (описанный в RFC 2616) НЕ ДОЛЖЕН использоваться для передачи сообщений об ошибках.	AJ, AX, AAJ, AAX

[RSG-93].	В формат сервисного контракта СЛЕДУЕТ включать следующее: - Версия API; - Информация о семантике элементов API; - Ресурсы; - Атрибуты ресурса; - Параметры запроса; - Методы; - Типы носителей; - Грамматика поиска (если она поддерживается); - Коды состояния HTTP; - Методы HTTP; - Ограничения и отличительные особенности; и - Безопасность (если есть).	AJ, AX, AAJ, AAX
[RSG-95]	REST API ДОЛЖЕН предоставлять документацию API в виде сервисного контракта.	AJ, AX, AAJ, AAX
[RSG-96]	Реализация Web API, отклоняющаяся от настоящего стандарта, ДОЛЖНА быть явно документирована в сервисном контракте. Если правило отклонения не указано в сервисном контракте, ДОЛЖНО подразумеваться, что соблюдается настоящий стандарт.	AJ, AX, AAJ, AAX
[RSG-97]	Сервисный контракт ДОЛЖЕН позволять генерировать скелетный код клиента API.	AJ, AX, AAJ, AAX
[RSG-105]	Web API ДОЛЖЕН поддерживать кэширование результатов GET; Web API МОЖЕТ поддерживать кэширование результатов других методов HTTP.	AJ, AX, AAJ
[RSG-113]	Если Web API поддерживает обработку настроек, номенклатура настроек, которые МОГУТ быть установлены с помощью заголовка `Prefer`, ДОЛЖНА быть записана в сервисном контракте.	AAJ, AAX, AJ, AX
[RSG-114]	Если Web API поддерживает локализованные данные, HTTP-заголовок запроса `Accept-Language` ДОЛЖЕН поддерживаться для указания набора естественных языков, которые предпочтительны в ответе, как указано в IETF RFC 7231.	AAJ, AAX, AJ, AX
[RSG-116]	Конфиденциальность: Различные API и информация в API ДОЛЖНЫ быть идентифицированы, классифицированы и защищены от несанкционированного доступа, раскрытия и перехвата в любое время. Должны соблюдаться принципы наименьших привилегий, нулевого доверия, необходимости знать и необходимости делиться.	AAJ, AAX, AJ, AX

[RSG-117]	Обеспечение целостности: Различные API и информация в API ДОЛЖНЫ быть защищены от несанкционированной модификации, дублирования, повреждения и уничтожения. Информация ДОЛЖНА модифицироваться через подтвержденные транзакции и интерфейсы. Системы ДОЛЖНЫ обновляться с использованием утвержденных процессов управления конфигурацией, управления изменениями и управления исправлениями.	AAJ, AAX, AJ, AX
[RSG-118]	Доступность: Различные API и информация в API ДОЛЖНЫ быть доступны авторизованным пользователям в установленное время в соответствии с соглашениями об уровне обслуживания (SLA), политиками контроля доступа и определенными бизнес-процессами.Неотказуемость (безотзывность): Каждая обрабатываемая транзакция или действие, выполняемое API, ДОЛЖНЫ обеспечивать неотказуемость посредством реализации надлежащего аудита, авторизации, аутентификации, а также реализации безопасных путей, а также и сервисов и механизмов неотказуемости.	AAJ, AAX, AJ, AX
[RSG-119]	Неотказуемость (безотзывность): Каждая обрабатываемая транзакция или действие, выполняемое API, ДОЛЖНЫ обеспечивать неотказуемость посредством реализации надлежащего аудита, авторизации, аутентификации, а также реализации безопасных путей, а также и сервисов и механизмов неотказуемости.	AAJ, AAX, AJ, AX
[RSG-120]	Аутентификация, авторизация, аудит: Пользователи, системы, API или устройства, вовлеченные в критические транзакции или действия, ДОЛЖНЫ быть аутентифицированы, авторизованы с помощью служб контроля доступа на основе ролей или атрибутов и поддерживать разделение обязанностей. Кроме того, все действия ДОЛЖНЫ регистрироваться, а надежность аутентификации должна увеличиваться в соответствии с соответствующим информационным риском.	AAJ, AAX, AJ, AX

[RSG-121]	При разработке API ТРЕБУЕТСЯ тщательно учитывать угрозы, случаи вредоносного использования, методы безопасного кодирования, безопасность транспортного уровня и тестирование безопасности, особенно: `PUTs и POSTs` - т.е.: какие изменения внутренних данных потенциально могут быть использованы для атаки или ложной информации; `DELETES` - т.е.: может использоваться для удаления содержимого внутреннего хранилища ресурсов; Белый список допустимых методов - для обеспечения того, чтобы допустимые методы HTTP были должным образом ограничены, в то время как другие будут возвращать правильный код ответа; и Хорошо известные атаки следует учитывать на этапе моделирования угроз в процессе проектирования, чтобы не допустить увеличения риска угроз. Угрозы и меры по их снижению, определенные в OWASP Top Ten Cheat Sheet, ДОЛЖНЫ быть приняты во внимание.	AAJ, AAX, AJ, AX
[RSG-122]	При разработке API следует придерживаться стандартов и лучших практик, перечисленных ниже: - Лучшие практики безопасного кодирования: Принципы безопасного кодирования OWASP; - Безопасность Rest API: Памятка по безопасности REST; - Экранирование ввода и защита от межсайтового скриптинга: OWASP XSS Cheat Sheet; - Предотвращение SQL-инъекций: Памятка OWASP по SQL-иньекциям, памятка OWASP по Parameterization и - Безопасность транспортного уровня: Памятка OWASP по защите транспортного уровня.	AJ, AX, AAX, AAJ
[RSG-123]	Тестирование безопасности и оценка уязвимостей ДОЛЖНЫ проводиться для обеспечения безопасности и устойчивости API к угрозам. Это требование МОЖЕТ быть выполнено путем использования статического и динамического тестирования безопасности приложений (SAST/DAST), автоматизированных инструментов управления уязвимостями и тестирования на проникновение.	AJ, AX, AAJ, AAX

[RSG-124]	Защищенные службы ДОЛЖНЫ предоставлять конечные точки HTTPS только с использованием TLS 1.2 или выше с набором шифров, включающим ECDHE для обмена ключами.	AJ, AX, AAJ, AAX
[RSG-130]	Анонимная аутентификация ДОЛЖНА использоваться только тогда, когда клиенты и используемое ими приложение получают доступ к информации или функциям с низким уровнем чувствительности, которые не должны требовать аутентификации, например, к публичной информации.	AJ, AX, AAJ, AAX
[RSG-131]	Аутентификация с использованием имени пользователя и пароля или хэш-пароля НЕ ДОЛЖНА разрешаться.	AJ, AX, AAJ, AAX
[RSG-141]	Ключи API ДОЛЖНЫ быть отозваны, если клиент нарушает соглашение об использовании, как указано в ОИС (Организации интеллектульной собственности).	AJ, AX, AAJ, AAX
[RSG-144]	Безопасные и надежные сертификаты ДОЛЖНЫ выдаваться взаимно доверенным центром сертификации (ЦС) через процесс установления доверия или перекрестной сертификации.	AJ, AX, AAJ, AAX
[RSG-145]	Сертификаты, совместно используемые клиентом и сервером, ДОЛЖНЫ использоваться для снижения рисков безопасности идентификации, характерных для чувствительных систем и привилегированных действий, например, X.509.	AJ, AX, AAJ, AAX
[RSG-148]	Если REST API является публичным, HTTP-заголовок Access-Control-Allow-Origin ДОЛЖЕН иметь значение '*'.	AJ, AX, AAJ, AAX

Таблица 3: Таблица соответствия Уровень AAJ

Идентификатор правила	Правило	Перекрестные ссылки и примечания
[RSG-01]	Символ прямой косой черты "/" ДОЛЖЕН использоваться в пути URI для указания иерархических отношений между ресурсами, но путь НЕ ДОЛЖЕН заканчиваться прямой косой чертой, так как это не дает никакого семантического значения и может привести к путанице.	AAJ, AAX, AX, AJ
[RSG-02]	Имена ресурсов ДОЛЖНЫ придерживаться своего шаблона именования.	AAJ, AAX, AX, AJ
[RSG-03]	Для имен ресурсов в запросе СЛЕДУЕТ использовать соглашения об именовании в kebab-case стиле, и они МОГУТ быть сокращенными.	AAJ, AAX
[RSG-05]	Параметры запроса ДОЛЖНЫ использовать соглашение о нижнем регистре (lowerCamelCase), и они МОГУТ быть сокращенными.	AAJ, AAX
[RSG-06]	Шаблон URL для Web API ДОЛЖЕН содержать слово "api" в URI.	AAJ, AAX, AX, AJ
[RSG-07]	Матричные параметры НЕ ДОЛЖНЫ использоваться.	AAJ, AAX, AX, AJ
[RSG-08]	Web API ДОЛЖЕН совместимым образом применять коды состояния HTTP, как описано в RFC IETF.	AAJ, AAX, AX, AJ
[RSG-09]	Рекомендуемые коды в Приложении V СЛЕДУЕТ использовать в Web API для классификации ошибки.	AAX, AAJ
[RSG-10]	Если API обнаруживает недопустимые значения ввода, он ДОЛЖЕН вернуть код состояния HTTP "`400 Bad Request`". Содержимое ошибки ДОЛЖНО указывать на ошибочное значение.	AAJ, AAX, AX, AJ
[RSG-11]	Если API обнаруживает синтаксически правильные имена аргументов (в параметрах запроса или запроса), которые не ожидаются, ему СЛЕДУЕТ игнорировать их.	AAJ, AAX
[RSG-12]	Если API обнаруживает допустимые значения, которые требуют нереализованных функций, он ДОЛЖЕН вернуть код состояния HTTP "501 Not Implemented". Содержимое ошибки ДОЛЖНО указывать на не обработанное значение.	AAJ, AAX, AX, AJ
[RSG-13]	Web API СЛЕДУЕТ использовать только ресурсы верхнего уровня. Если есть подресурсы, они должны быть коллекциями и подразумевать ассоциацию. Объект должен быть доступен либо как ресурс верхнего уровня, либо как подресурс, но не использовать оба способа.	AAJ, AAX
[RSG-14]	Если ресурс может быть самостоятельным, он ДОЛЖЕН быть ресурсом верхнего уровня, в противном случае - подресурсом.	AAJ, AAX, AX, AJ
[RSG-15]	Параметры запроса ДОЛЖНЫ использоваться вместо путей URL для получения вложенных ресурсов.	AAJ, AAX, AX, AJ
[RSG-16]	Именам ресурсов СЛЕДУЕТ быть существительными для CRUD Web API и глаголами для Intent Web API.	AAJ, AAX

[RSG-17]	Если имя ресурса является существительным, его СЛЕДУЕТ всегда использовать в форме множественного числа. Неправильные формы существительных НЕ СЛЕДУЕТ использовать. Например, вместо `/people` следует использовать `/persons`.	AAJ, AAX
[RSG-18]	Имена ресурсов, сегментов и параметры запросов ДОЛЖНЫ состоять из слов на английском языке с использованием основных английских написаний, представленных в Оксфордском словаре английского языка. Имена ресурсов, которые локализованы в связи с требованиями бизнеса, МОГУТ быть на других языках.	AAJ, AAX, AX, AJ
[RSG-19]	Для Web API СЛЕДУЕТ использовать для согласования типа содержимого HTTP-заголовок запроса `Accept` и HTTP-заголовок ответа `Content-Type`.	AAJ, AAX
[RSG-20]	Web API ДОЛЖЕН поддерживать согласование типов содержимого в соответствии с IETF RFC 7231.	AAJ, AAX, AX, AJ
[RSG-21]	Формат JSON ДОЛЖЕН предполагаться, если не запрашивается определенный тип содержимого.	AAJ, AAX, AX, AJ
[RSG-22]	Web API СЛЕДУЕТ возвращать код состояния "`406 Not Acceptable`", если запрошенный формат не поддерживается.	AAJ, AAX
[RSG-23]	Web API СЛЕДУЕТ отклонять запросы, содержащие неожиданные или отсутствующие заголовки типа содержимого, выдавая кодом состояния HTTP "`406 Not Acceptable`" или "`415 Unsupported Media Type`".	AAJ, AAX
[RSG-24]	Запросы и ответы (соглашение об именах, формат сообщения, данные структуру и словарь данных) СЛЕДУЕТ ссылаться на стандарт ВОИС ST.96 для XML или стандарт ВОИС ST.97 для JSON.	AAX, AAJ
[RSJ-25]	Имена свойств объектов JSON СЛЕДУЕТ предоставлять в нижнем регистре (lowerCamelCase), например, `applicantName`.	AAJ
[RSG-27]	Web API ДОЛЖЕН поддерживать как минимум XML или JSON.	AAJ, AAX, AX, AJ
[RSG-28]	Методы HTTP ДОЛЖНЫ быть ограничены стандартными методами HTTP `POST, GET, PUT, DELETE, OPTIONS, PATCH, TRACE и HEAD`, как указано в IETF RFC 7231 и 5789.	AAJ, AAX, AX, AJ
[RSG-29]	Методы HTTP МОГУТ следовать принципу выбора, который гласит, что должна быть реализована только та функциональность, которая необходима для целевого сценария использования.	AAJ, AAX

[RSG-30]	Некоторые прокси-серверы поддерживают только методы `POST и GET`. Чтобы преодолеть эти ограничения, Web API МОЖЕТ использовать метод `POST` с пользовательским заголовком HTTP, "туннелирующим" настоящий метод HTTP. СЛЕДУЕТ использовать пользовательский HTTP-заголовок `X-HTTP-Method`.	AAJ, AAX
[RSG-31]	Если метод HTTP не поддерживается, СЛЕДУЕТ возвратить код состояния HTTP "`405 Method Not Allowed`".	AAJ, AAX
[RSG-32]	Web API СЛЕДУЕТ поддерживать пакетные операции (они же массовые операции) вместо нескольких отдельных запросов для снижения задержки. Такая же семантика должна использоваться для методов HTTP и кодов состояния HTTP. В содержательной части ответа СЛЕДУЕТ иметь информацию обо всех пакетных операциях. Если возникает несколько ошибок, содержательной части ошибки СЛЕДУЕТ иметь информацию обо всех ошибках (в атрибуте details). Все пакетные операции СЛЕДУЕТ выполнять атомарно.	AAJ, AAX
[RSG-33]	Для конечной точки, которая извлекает один ресурс, если ресурс не найден, метод `GET` ДОЛЖЕН вернуть код состояния "`404 Not Found`". Конечные точки, которые возвращают списки ресурсов, просто возвращают пустой список.	AAJ, AAX, AX, AJ
[RSG-34]	Если ресурс получен успешно, метод `GET` ДОЛЖЕН вернуть `200 OK`.	AAJ, AAX, AX, AJ
[RSG-35]	Запрос `GET` ДОЛЖЕН быть идемпотентным (независимым от количества выполнений запроса).	AAJ, AAX, AX, AJ
[RSG-36]	Если длина URI превышает 255 байт, РЕКОМЕНДУЕТСЯ использовать метод `POST` вместо `GET` из-за ограничений `GET`, или если возможно, создавать именованные запросы.	AAJ, AAX
[RSG-37]	Запрос `HEAD` ДОЛЖЕН быть идемпотентным.	AAJ, AAX, AX, AJ
[RSG-38]	Некоторые прокси-серверы поддерживают только методы `POST и GET`. Web API СЛЕДУЕТ поддерживать пользовательский заголовок запроса HTTP для переопределения метода HTTP, чтобы преодолеть эти ограничения.	AAJ, AAX
[RSG-39]	`POST`-запрос НЕ ДОЛЖЕН быть идемпотентным в соответствии с IETF RFC 2616.	AAJ, AAX, AX, AJ
[RSG-40]	Если создание ресурса было успешным, в HTTP-заголовоке Location СЛЕДУЕТ содержать URI (абсолютный или относительный), указывающий на созданный ресурс.	AAJ, AAX
[RSG-41]	Если создание ресурса прошло успешно, в ответе СЛЕДУЕТ содержать код состояния "201 Created".	AAJ, AAX

[RSG-42]	Если создание ресурса прошло успешно, в полезной части ответа СЛЕДУЕТ по умолчанию содержаться телу созданного ресурса, чтобы клиент мог использовать его без дополнительного вызова HTTP.	AAJ, AAX
[RSG-43]	Запрос `PUT` ДОЛЖЕН быть идемпотентным.	AAJ, AAX, AX, AJ
[RSG-44]	Если ресурс не найден, `PUT` ДОЛЖЕН вернуть код состояния "`404 Not Found`".	AAJ, AAX, AX, AJ
[RSG-45]	Если ресурс успешно обновлен, `PUT` ДОЛЖЕН вернуть код состояния "`200 OK`", если обновленный ресурс возвращен, или "`204 No Content`", если он не возвращен.	AAJ, AAX, AX, AJ
[RSG-46]	Запрос `PATCH` НЕ ДОЛЖЕН быть идемпотентным.	AAJ, AAX, AX, AJ
[RSG-47]	Если Web API реализует частичные обновления, то идемпотентные характеристики `PATCH` РЕКОМЕНДУЕТСЯ принимать во внимание. Для того чтобы сделать его идемпотентным, API МОЖЕТ следовать предложению IETF RFC 5789 об использовании оптимистической блокировки.	AAJ, AAX
[RSG-48]	Если ресурс не найден, `PATCH` ДОЛЖЕН вернуть код состояния "`404 Not Found`".	AAJ, AAX, AX, AJ
[RSJ-49]	Если Web API реализует частичные обновления с помощью `PATCH`, он ДОЛЖЕН использовать формат JSON Merge Patch для описания частичного набора изменений, как описано в IETF RFC 7386, используя тип содержимого `application/merge-patch+json`.	AAJ, AJ
[RSG-50]	Запрос `DELETE` НЕ ДОЛЖЕН быть идемпотентным.	AAJ, AAX, AX, AJ
[RSG-51]	Если ресурс не найден, `DELETE` ДОЛЖЕН вернуть код состояния "`404 Not Found`".	AAJ, AAX, AX, AJ
[RSG-52]	Если ресурс удален успешно, `DELETE` ДОЛЖЕН вернуть статус "`200 OK`", если удаленный ресурс возвращен, или "`204 No Content`", если он не возвращен.	AAJ, AAX, AX, AJ
[RSG-53].	Конечным получателем является либо исходный сервер, либо первый прокси или шлюз, получивший в запросе значение Max-Forwards равное нулю. Запрос `TRACE` НЕ ДОЛЖЕН включать тело (запроса).	AAJ, AAX, AX, AJ
[RSG-54]	Запрос `TRACE` НЕ ДОЛЖЕН быть идемпотентным.	AAJ, AAX, AX, AJ
[RSG-55].	Значение поля заголовка `Via` HTTP ДОЛЖНО служить для отслеживания цепочки запросов.	AAJ, AAX, AX, AJ
[RSG-56]	Поле заголовка HTTP Max-Forwards ДОЛЖНО использоваться для того, чтобы клиент мог ограничить длину цепочки запросов.	AAJ, AAX, AX, AJ
[RSG-57]	Если запрос корректный, в ответ СЛЕДУЕТ поместить все сообщение запроса в теле ответа, с Content-Type "message/http".	AAJ, AAX
[RSG-58].	Ответы на `TRACE` НЕ ДОЛЖНЫ кэшироваться.	AAJ, AAX, AX, AJ
[RSG-59]	Код состояния "`200 OK`" СЛЕДУЕТ вернуть в `TRACE`.	AAJ, AAX

[RSG-60]	Запрос `OPTIONS` ДОЛЖЕН быть идемпотентным.	AAJ, AAX, AX, AJ
[RSG-61]	Пользовательские HTTP-заголовки, начинающиеся с префикса "X-", НЕ РЕКОМЕНДУЕТСЯ использовать.	AAJ, AAX
[RSG-62]	Пользовательские HTTP-заголовки НЕ СЛЕДУЕТ использовать для изменения поведения HTTP-методов, за исключением случаев, когда это необходимо для устранения существующих технических ограничений (например, см. [RSG-39]).	AAJ, AAX
[RSG-63]	Соглашение об именовании пользовательских HTTP-заголовков - <организация>-<имя заголовка> (`<organization>-<header name>`) , где полям <организация> и <заголовок> рекомендуется следовать соглашению о kebab-case стиле именования.	AAJ, AAX
[RSG-64]	Web API СЛЕДУЕТ поддерживать единственный метод версионирования сервиса с использованием версионирования URI, например `/api/v1/inventors` или версионирования заголовка, например `Accept-version: v1` или версионирования типа данных, например `Accept: application/vnd.v1+json`. Версионирование строк запросов НЕ РЕКОМЕНДУЕТСЯ использовать.	AAJ, AAX
[RSG-65]	Схему нумерации версий СЛЕДУЕТ применять с учетом только основного номера версии (например, `/v1`).	AAJ, AAX
[RSG-66]	Сервисные контракты API МОГУТ включать функцию перенаправления конечных точек. Когда потребитель услуг пытается вызвать услугу, может быть возвращен ответ о перенаправлении, чтобы сообщить потребителю	AAJ, AAX
	услуг о необходимости повторно отправить запрос на новую конечную точку. Перенаправления МОГУТ быть временными или постоянными: Временное перенаправление - с использованием заголовка HTTP ответа Location и кода статуса HTTP "302 Found" в соответствии с IETF RFC 7231; или Постоянное перенаправление - с использованием заголовка HTTP-ответа Location и кода статуса HTTP "301 Moved Permanently" в соответствии с IETF RFC 7238.	
[RSG-67]	Стратегии жизненного цикла API СЛЕДУЕТ публиковать разработчикам, чтобы помочь пользователям понять, как долго будет поддерживаться та или иная версия.	AAJ, AAX
[RSG-68]	Web API СЛЕДУЕТ поддерживать пагинацию (разбивки на страницы).	AAJ, AAX
[RSG-69]	Пагинированные запросы НЕ МОГУТ быть идемпотентными.	AAJ, AAX

[RSG-70]	Web API ДОЛЖЕН использовать параметры запроса для реализации пагинации.	AAJ, AAX, AX, AJ
[RSG-71]	Веб-интерфейс НЕ ДОЛЖЕН использовать HTTP-заголовки для реализации пагинации.	AAJ, AAX, AX, AJ
[RSG-72]	СЛЕДУЕТ использовать параметры запроса `limit=<количество элементов для доставки>` и `offset=<количество элементов для пропуска>` (`limit=<number of items to deliver> and offset=<number of items to skip>`), где limit - количество возвращаемых элементов (размер страницы), а пропуск (skip) - количество элементов, которые должны быть пропущены (смещение). Если ограничение на размер страницы не указано, СЛЕДУЕТ определить значение по умолчанию - глобальное или для каждой коллекции; смещение по умолчанию ДОЛЖНО быть равно нулю "0": Например, следующий URL является допустимым: https://wipo.int/api/v1/patents?limit=10&offset=20	AAJ, AAX
[RSG-73].	Значения параметров limit и skip СЛЕДУЕТ включить в ответ.	AAJ, AAX
[RSG-74].	Web API СЛЕДУЕТ поддерживать сортировку.	AAJ, AAX
[RSG-75]	Для указания критерия сортировки по нескольким атрибутам ДОЛЖЕН использоваться параметр запроса. Значение этого параметра представляет собой список ключей сортировки, разделенных запятыми, а	AAJ, AAX, AX, AJ
	направления сортировки либо 'asc' для восходящего, либо 'desc' для нисходящего МОГУТ быть добавлены к каждому ключу сортировки, разделенные символом двоеточия ':'. Направление по умолчанию ДОЛЖНО быть определено сервером в случае, если для ключа не указано направление сортировки.	
[RSG-76]	Web API СЛЕДУЕТ возвращать критерии сортировки в ответе.	AAJ, AAX, AX, AJ
[RSG-77]	Web API МОЖЕТ поддерживать расширение тела возвращаемого содержимого. СЛЕДУЕТ использовать параметр запроса `expand=<разделенный запятыми список имен атрибутов>`.	AAJ, AAX
[RSG-78]	Параметр запроса СЛЕДУЕТ использовать вместо путей URL в случае, если веб-интерфейс поддерживает проекцию в формате: `"fields="<разделенный запятыми список имен атрибутов>.`	AAJ, AAX
[RSG-79]	Web API ДОЛЖЕН поддерживать возврат количества элементов в коллекции.	AAJ, AAX, AX, AJ
[RSG-80]	Параметр запроса ДОЛЖЕН использоваться для поддержки возврата количества элементов в коллекции.	AAJ, AAX, AX, AJ

[RSG-81]	СЛЕДУЕТ использовать параметр запроса count для возврата количества элементов в коллекции.	AAJ, AAX
[RSG-82]	Web API МОЖЕТ поддерживать возврат количества элементов в коллекции внутри, т.е. как часть ответа, которая содержит саму коллекцию. При этом ДОЛЖЕН использоваться параметр запроса.	AAJ, AAX, AX, AJ
[RSG-83]	СЛЕДУЕТ использовать параметр запроса `count=true`. Если он не указан, то по умолчанию count следует установить в false.	AAJ, AAX
[RSG-84]	Если Web API поддерживает пагинацию, то СЛЕДУЕТ поддерживать возврат внутри ответа размера коллекции (т.е. общего количества элементов коллекции).	AAJ, AAX
[RSG-85]	Если Web API поддерживает сложные поисковые выражения, СЛЕДУЕТ указывать язык запросов, например, CQL.	AAJ, AAX
[RSG-86]	Сервисный (веб-сервисный) контракт ДОЛЖЕН определять поддерживаемую грамматику (например, поля, функции, ключевые слова и операторы).	AAJ, AAX, AX, AJ
[RSG-87]	ДОЛЖЕН использоваться параметр запроса "q".	AAJ, AAX, AX, AJ
[RSG-88]	На уровне протокола Web API ДОЛЖЕН возвращать соответствующий код состояния HTTP, выбранный из списка стандартных кодов состояния HTTP.	AAJ, AAX, AX, AJ
[RSJ-89]	На уровне приложения Web API ДОЛЖЕН возвращать часть, сообщающую об ошибке с достаточной степенью детализации. Атрибуты `code` и `message` являются обязательными, атрибут `details` является условно обязательным, атрибуты `target, status, moreInfo` и `internalMessage` являются необязательными.	AAJ, AAX, AX, AJ
[RSG-90]	Ошибки НЕ ДОЛЖНЫ раскрывать критически важные для безопасности данные или внутренние технические детали, такие как стеки вызовов в сообщениях об ошибках.	AAJ, AAX, AX, AJ
[RSG-91].	`Заголовок` HTTP: `Reason-Phrase` (описанный в RFC 2616) НЕ ДОЛЖЕН использоваться для передачи сообщений об ошибках.	AAJ, AAX, AX, AJ
[RSG-92].	Каждой зарегистрированной ошибке СЛЕДУЕТ иметь уникальный идентификатор корреляции. СЛЕДУЕТ использовать пользовательский HTTP-заголовок, которому СЛЕДУЕТ иметь имя Correlation-ID.	AAJ, AAX
[RSG-93].	В формат сервисного контракта СЛЕДУЕТ включать следующее: Версия API; Информация о семантике элементов API; Ресурсы; Атрибуты ресурса; Параметры запроса; Методы; Типы носителей;	AAJ, AAX, AX, AJ

	Грамматика поиска (если она поддерживается); Коды состояния HTTP; Методы HTTP; Ограничения и отличительные особенности; и Безопасность (если есть).	
[RSG-94]	В формат сервисного контракта СЛЕДУЕТ включать запросы и ответы в XML-схеме или JSON-схеме и примеры использования API в поддерживаемых форматах, т.е. XML или JSON.	AAJ, AAX
[RSG-95]	REST API ДОЛЖЕН предоставлять документацию API в виде сервисного контракта.	AAJ, AAX, AX, AJ
[RSG-96]	Реализация Web API, отклоняющаяся от настоящего стандарта, ДОЛЖНА быть явно документирована в сервисном контракте. Если правило отклонения не указано в сервисном контракте, ДОЛЖНО подразумеваться, что соблюдается настоящий стандарт.	AAJ, AAX, AX, AJ
[RSG-97]	Сервисный контракт ДОЛЖЕН позволять генерировать скелетный код клиента API.	AAJ, AAX, AX, AJ
[RSG-98]	Сервисный контракт РЕКОМЕНДУЕТСЯ, что может позволять генерировать скелетный код сервера.	AAJ, AAX
[RSG-99]	Документацию Web API СЛЕДУЕТ писать на RAML или OAS. Пользовательские форматы документации НЕ СЛЕДУЕТ использовать.	AAJ, AAX
[RSG-100]	СЛЕДУЕТ дать возможность потребителю Web API указать тайм-аут сервера для каждого запроса; следует использовать пользовательский HTTP-заголовок. Максимальный тайм-аут сервера СЛЕДУЕТ также применять для защиты ресурсов сервера от чрезмерного использования.	AAJ, AAX
[RSG-101]	Web API СЛЕДУЕТ поддерживать условное получение данных, чтобы гарантировать, что будут получены только измененные данные. СЛЕДУЕТ применять проверку ресурсов на основе содержания, поскольку она точнее.	AAJ, AAX
[RSG-102]	Для реализации Content-based Resource Validation (проверка корректности ресурса по содержимому) HTTP-заголовок `ETag` СЛЕДУЕТ использовать в ответе для кодирования состояния данных. Затем это значение СЛЕДУЕТ использовать в последующих запросах в условных HTTP-заголовках (таких как If-Match или If-None-Match — Если соответствует или Если не соответствует). Если данные не были изменены с тех пор, как запрос вернул `ETag`, серверу СЛЕДУЕТ вернуть код состояния "`304 Not Modified`" (если данные не изменены). Этот механизм описан в IETF RFC 7231 и 7232.	AAJ, AAX

[RSG-103]	Для реализации проверки ресурсов по времени СЛЕДУЕТ использовать HTTP-заголовок `Last-Modified`. Этот механизм описан в IETF RFC 7231 и 7232.	AAJ, AAX
[RSG-104]	Используя версионность ответа, потребитель услуг МОЖЕТ реализовать оптимистическую блокировку.	AAJ, AAX
[RSG-105]	Web API ДОЛЖЕН поддерживать кэширование результатов `GET`; Web API МОЖЕТ поддерживать кэширование результатов других методов HTTP.	AAJ, AJ, AX
[RSG-106]	СЛЕДУЕТ использовать HTTP-заголовки ответа `Cache-Control` и `Expires`. Последний МОЖЕТ использоваться для поддержки устаревших клиентов.	AAJ, AAX
[RSG-107]	Web API СЛЕДУЕТ сообщать, поддерживается ли частичная загрузка файлов, отвечая на запросы `HEAD` и передавая в ответ HTTP-заголовки `Accept-Ranges` и `Content-Length`.	AAJ, AAX
[RSG-108]	Web API СЛЕДУЕТ поддерживать частичную загрузку файлов. Следует поддерживать диапазоны из нескольких частей.	AAJ, AAX
[RSG-109]	Web API СЛЕДУЕТ сообщать, поддерживает ли он частичную заливку файлов.	AAJ, AAX
[RSG-110]	Web API СЛЕДУЕТ поддерживать частичную заливку файлов. СЛЕДУЕТ поддерживать составные диапазоны.	AAJ, AAX
[RSG-111]	Поставщику услуг СЛЕДУЕТ возвращать в заголовках ответа HTTP заголовок "`413 Request Entity Too Large`" в случае, если запрос превысил максимально допустимый предел. Пользовательский HTTP-заголовок МОЖЕТ использоваться для указания максимального размера запроса.	AAJ, AAX
[RSG-112]	Если Web API поддерживает обработку настроек, ее СЛЕДУЕТ реализовать в соответствии с IETF RFC 7240, т.е. следует использовать HTTP-заголовок запроса `Prefer`, в HTTP-заголовке ответа СЛЕДУЕТ вернуть `Preference-Applied` (с повторением исходного запроса).	AAJ, AAX
[RSG-113]	Если Web API поддерживает обработку настроек, номенклатура настроек, которые МОГУТ быть установлены с помощью заголовка `Prefer`, ДОЛЖНА быть записана в сервисном контракте.	AAJ, AAX, AJ, AX
[RSG-114]	Если Web API поддерживает локализованные данные, HTTP-заголовок запроса `Accept-Language` ДОЛЖЕН поддерживаться для указания набора естественных языков, которые предпочтительны в ответе, как указано в IETF RFC 7231.	AAJ, AAX, AJ, AX

[RSG-115]	Если API поддерживает длительные операции, им СЛЕДУЕТ быть асинхронными. Должен применяться следующий подход: a) Потребитель услуг активирует сервисную операцию; b) Операция сервиса возвращает код состояния "`202 Accepted`" в соответствии с IETF RFC 7231 (раздел 6.3.3), т.е. запрос был принят к обработке, но обработка не была завершена. Местоположение созданной задачи в очереди также возвращается с HTTP-заголовком `Location`; и c) Потребитель услуги обращается к возвращаемому `местоположению`, чтобы узнать, доступен ли ресурс. Если ресурс недоступен, в ответе СЛЕДУЕТ иметь код состояния "`200 OK`", содержать статус задачи (например, ожидание) и МОЖЕТ содержаться другая информация (например, индикатор выполнения и/или ссылка для отмены или удаления задачи с помощью HTTP-метода DELETE). Если ресурс доступен, в ответе СЛЕДУЕТ привести код состояния "`303 See Other`", а в заголовоке HTTP `Location` СЛЕДУЕТ привести URL для получения результатов задачи.	AAJ, AAX
[RSG-116]	Конфиденциальность: Различные API и информация в API ДОЛЖНЫ быть идентифицированы, классифицированы и защищены от несанкционированного доступа, раскрытия и перехвата в любое время. Должны соблюдаться принципы наименьших привилегий, нулевого доверия, необходимости знать и необходимости делиться.	AAJ, AAX, AJ, AX
[RSG-117]	Обеспечение целостности: Различные API и информация в API ДОЛЖНЫ быть защищены от несанкционированной модификации, дублирования, повреждения и уничтожения. Информация ДОЛЖНА модифицироваться через подтвержденные транзакции и интерфейсы. Системы ДОЛЖНЫ обновляться с использованием утвержденных процессов управления конфигурацией, управления изменениями и управления исправлениями.	AAJ, AAX, AJ, AX
[RSG-118]	Доступность: Различные API и информация в API ДОЛЖНЫ быть доступны авторизованным пользователям в установленное время в соответствии с соглашениями об уровне обслуживания (SLA), политиками контроля доступа и определенными бизнес-процессами.	AAJ, AAX, AJ, AX

[RSG-119]	Неотказуемость (безотзывность): Каждая обрабатываемая транзакция или действие, выполняемое API, ДОЛЖНЫ обеспечивать неотказуемость посредством реализации надлежащего аудита, авторизации, аутентификации, а также реализации безопасных путей, а также и сервисов и механизмов неотказуемости.	AAJ, AAX, AJ, AX
[RSG-120]	Аутентификация, авторизация, аудит: Пользователи, системы, API или устройства, вовлеченные в критические транзакции или действия, ДОЛЖНЫ быть аутентифицированы, авторизованы с помощью служб контроля доступа на основе ролей или атрибутов и поддерживать разделение обязанностей. Кроме того, все действия ДОЛЖНЫ регистрироваться, а надежность аутентификации должна увеличиваться в соответствии с соответствующим информационным риском.	AAJ, AAX, AJ, AX
[RSG-121]	При разработке API ТРЕБУЕТСЯ тщательно учитывать угрозы, случаи вредоносного использования, методы безопасного кодирования, безопасность транспортного уровня и тестирование безопасности, особенно: – PUTs и POSTs - т.е.: какие изменения внутренних данных потенциально могут быть использованы для атаки или ложной информации; – DELETES - т.е.: может использоваться для удаления содержимого внутреннего хранилища ресурсов; - Белый список допустимых методов - для обеспечения того, чтобы допустимые методы HTTP были должным образом ограничены, в то время как другие будут возвращать правильный код ответа; и - Хорошо известные атаки следует учитывать на этапе моделирования угроз в процессе проектирования, чтобы не допустить увеличения риска угроз. Угрозы и меры по их снижению, определенные в OWASP Top Ten Cheat Sheet, ДОЛЖНЫ быть приняты во внимание.	AAJ, AAX, AJ, AX

[RSG-122]	При разработке API следует придерживаться стандартов и лучших практик, перечисленных ниже: - Лучшие практики безопасного кодирования: Принципы безопасного кодирования OWASP; - Безопасность Rest API: Памятка по безопасности REST; - Экранирование ввода и защита от межсайтового скриптинга: OWASP XSS Cheat Sheet; - Предотвращение SQL-инъекций: Памятка OWASP по SQL-иньекциям, памятка OWASP по Parameterization; и - Безопасность транспортного уровня: Памятка OWASP по защите транспортного уровня.	AAJ, AAX, AJ, AX
[RSG-123]	Тестирование безопасности и оценка уязвимостей ДОЛЖНЫ проводиться для обеспечения безопасности и устойчивости API к угрозам. Это требование МОЖЕТ быть выполнено путем использования статического и динамического тестирования безопасности приложений (SAST/DAST), автоматизированных инструментов управления уязвимостями и тестирования на проникновение.	AAJ, AAX, AJ, AX
[RSG-124]	Защищенные службы ДОЛЖНЫ предоставлять конечные точки HTTPS только с использованием TLS 1.2 или выше с набором шифров, включающим ECDHE для обмена ключами.	AAJ, AAX, AJ, AX
[RSG-125]	При рассмотрении протоколов аутентификации для обеспечения транспортной безопасности СЛЕДУЕТ использовать совершенную прямую секретность. Не СЛЕДУЕТ допускать использование небезопасных криптографических алгоритмов и обратную совместимость с SSL 3 и TLS 1.0/1.1.	AAX, AAJ
[RSG-126]	Для обеспечения максимальной безопасности и доверия следует устанавливать соединение через VPN IPSEC между сайтами для дополнительной защиты информации, передаваемой через незащищенные сети.	AAX, AAJ
[RSG-127]	Пользовательскому приложению СЛЕДУЕТ проверять цепочку сертификатов TLS при выполнении запросов к защищенным ресурсам, включая проверку списка отзыва сертификатов.	AAX, AAJ
[RSG-128]	Защищенные службы ДОЛЖНЫ использовать только действительные сертификаты, выданные доверенным центром сертификации (ЦС).	AAX, AAJ
[RSG-129]	Токены СЛЕДУЕТ подписывать с использованием безопасных алгоритмов подписи, соответствующих стандарту цифровой подписи (DSS) FIPS -186-4. Следует рассматривать алгоритм цифровой подписи RSA или алгоритм ECDSA.	AAX, AAJ

[RSG-130]	Анонимная аутентификация ДОЛЖНА использоваться только тогда, когда клиенты и используемое ими приложение получают доступ к информации или функциям с низким уровнем чувствительности, которые не должны требовать аутентификации, например, к публичной информации.	AAJ, AAX, AJ, AX
[RSG-131]	Аутентификация с использованием имени пользователя и пароля или хэш-пароля НЕ ДОЛЖНА разрешаться.	AAJ, AAX, AJ, AX
[RSG-132]	Если сервис защищен, СЛЕДУЕТ использоваться Open ID Connect.	AAX, AAJ
[RSG-133]	При использовании JSON Web Token (JWT) секрету JWT СЛЕДУЕТ обладать высокой энтропией, чтобы увеличить трудоемкость атаки грубой силой; токенам TTL и RTTL СЛЕДУЕТ быть как можно короче; и НЕ СЛЕДУЕТ хранить конфиденциальную информацию в полезной нагрузке JWT.	AAX, AAJ
[RSG-134]	В запросах POST/PUT конфиденциальные данные СЛЕДУЕТ передавать в теле или в заголовках запроса.	AAX, AAJ
[RSG-135]	В запросах GET конфиденциальные данные СЛЕДУЕТ передавать в заголовке HTTP.	AAX, AAJ
[RSG-136]	Для того чтобы минимизировать задержки и уменьшить связь между защищенными сервисами, решение об управлении доступом СЛЕДУЕТ принимать на локальном уровне в конечных точках REST.	AAX, AAJ
[RSG-137]	API-ключи СЛЕДУЕТ использовать и для защищенных и для публичных сервисов, чтобы не перегружать поставщика услуг многочисленными запросами (атаки типа "отказ в обслуживании"). Для защищенных сервисов API-ключи МОГУТ использоваться для монетизации (приобретенные тарифные планы), применения политики использования (QoS) и мониторинга.	AAX, AAJ
[RSG-138]	Ключи API МОГУТ быть объединены с заголовком HTTP запроса user-agent для различения пользователя-человека и программного агента, как указано в IETF RFC 7231.	AAX, AAJ
[RSG-139]	Сервис-провайдеру СЛЕДУЕТ возвращать вместе с заголовками ответа HTTP текущий статус использования. Следующие данные ответа МОГУТ быть возвращены: - предел скорости (rate limit) - предел скорости (в минуту), установленный в системе; - rate limit remaining - оставшееся количество запросов, разрешенных в течение текущего тайм-слота (-1 означает, что лимит превышен); и	AAX, AAJ

	- сброс лимита скорости (rate limit reset) - время (в секундах), оставшееся до сброса счетчика запросов.	
[RSG-140]	Сервис-провайдеру СЛЕДУЕТ возвращать код состояния "`429 Too Many Requests`", если запросы поступают слишком быстро.	AAX, AAJ
[RSG-141]	Ключи API ДОЛЖНЫ быть отозваны, если клиент нарушает соглашение об использовании, как указано в ОИС (Организации интеллектульной собственности).	AAJ, AAX, AJ, AX
[RSG-142]	Ключи API СЛЕДУЕТ передавать с помощью пользовательских HTTP-заголовков. Их НЕ СЛЕДУЕТ передавать с помощью параметров запроса.	AAX, AAJ
[RSG-143]	Ключи API СЛЕДУЕТ генерировать случайным образом.	AAX, AAJ
[RSG-144]	Безопасные и надежные сертификаты ДОЛЖНЫ выдаваться взаимно доверенным центром сертификации (ЦС) через процесс установления доверия или перекрестной сертификации.	AAJ, AAX, AJ, AX
[RSG-145]	Сертификаты, совместно используемые клиентом и сервером, СЛЕДУЕТ использовать для снижения рисков безопасности идентификации, характерных для чувствительных систем и привилегированных действий, например, X.509.	AAJ, AAX, AJ, AX
[RSG-146]	Для высокопривилегированных служб в двусторонней взаимной аутентификации между клиентом и сервером СЛЕДУЕТ использовать сертификаты для обеспечения дополнительной защиты.	AAX, AAJ
[RSG-147]	Для высокорисковых приложений, систем, обрабатывающих очень чувствительную информацию или выполняющих привилегированные действия СЛЕДУЕТ реализовать многофакторную аутентификация для снижения рисков идентификации.	AAX, AAJ
[RSG-148]	Если REST API является публичным, HTTP-заголовок Access-Control-Allow-Origin ДОЛЖЕН иметь значение '*'.	AAJ, AAX, AJ, AX
[RSG-149]	Если REST API защищенный, СЛЕДУЕТ использовать CORS, если это возможно. В противном случае, JSONP МОЖЕТ использоваться в качестве запасного варианта, но только для GET-запросов, например, когда пользователь получает доступ с помощью старого браузера. НЕ СЛЕДУЕТ использовать Iframe.	AAX, AAJ
[RSJ-150]	При использовании экземпляров, описанных схемой, СЛЕДУЕТ использовать заголовок Link для предоставления ссылки на загружаемую JSON-схему В СООТВЕТСТВИИ С RFC8288.	AAJ

[RSJ-151]	Web API СЛЕДУЕТ реализовать, по крайней мере уровень 2 (свойства, присущие транспорту) RMM. Уровень 3 (Гипермедиа) МОЖЕТ быть реализован, чтобы сделать API полностью информативным.	AAJ
[RSJ-152]	Для разработки пользовательского формата гипермедиа СЛЕДУЕТ использовать следующий набор атрибутов, заключенных в атрибут-ссылку: – `href` - целевой URI; – `rel` - значение целевого URI; – `self` – URI ссылается на сам ресурс; – `next` - URI ссылки на предыдущую страницу (если используется при пагинации); – previous - URI ссылки на следующую страницу (если используется при постраничной разбивке); и – Произвольное имя `v` обозначает пользовательское значение отношения.	AAJ

Таблица 4: Уровень соответствия AAX

Идентификатор правила	Правило	Перекрестные ссылки и примечания
[RSG-01]	Символ прямой косой черты "/" ДОЛЖЕН использоваться в пути URI для указания иерархических отношений между ресурсами, но путь НЕ ДОЛЖЕН заканчиваться прямой косой чертой, так как это не дает никакого семантического значения и может привести к путанице.	AAJ, AAX, AX, AJ
[RSG-02]	Имена ресурсов ДОЛЖНЫ придерживаться своего шаблона именования.	AAJ, AAX, AX, AJ
[RSG-03]	Для имен ресурсов в запросе СЛЕДУЕТ использовать соглашения об именовании в kebab-case стиле, и они МОГУТ быть сокращенными.	AAJ, AAX
[RSG-05]	Параметры запроса ДОЛЖНЫ использовать соглашение о нижнем регистре (lowerCamelCase), и они МОГУТ быть сокращенными.	AAJ, AAX
[RSG-06]	Шаблон URL для Web API ДОЛЖЕН содержать слово "api" в URI.	AAJ, AAX, AX, AJ
[RSG-07]	Матричные параметры НЕ ДОЛЖНЫ использоваться.	AAJ, AAX, AX, AJ
[RSG-08]	Web API ДОЛЖЕН совместимым образом применять коды состояния HTTP, как описано в RFC IETF.	AAJ, AAX, AX, AJ
[RSG-09]	Рекомендуемые коды в Приложении V СЛЕДУЕТ использовать в Web API для классификации ошибки.	AAX, AAJ
[RSG-10]	Если API обнаруживает недопустимые значения ввода, он ДОЛЖЕН вернуть код состояния HTTP "`400 Bad Request`". Содержимое ошибки ДОЛЖНО указывать на ошибочное значение.	AAJ, AAX, AX, AJ
[RSG-11]	Если API обнаруживает синтаксически правильные имена аргументов (в параметрах запроса или запроса), которые не ожидаются, ему СЛЕДУЕТ игнорировать их.	AAJ, AAX
[RSG-12]	Если API обнаруживает допустимые значения, которые требуют нереализованных функций, он ДОЛЖЕН вернуть код состояния HTTP "`501 Not Implemented`". Содержимое ошибки ДОЛЖНО указывать на не обработанное значение.	AAJ, AAX, AX, AJ
[RSG-13]	Web API СЛЕДУЕТ использовать только ресурсы верхнего уровня. Если есть подресурсы, они должны быть коллекциями и подразумевать ассоциацию. Объект должен быть доступен либо как ресурс верхнего уровня, либо как подресурс, но не использовать оба способа.	AAJ, AAX
[RSG-14]	Если ресурс может быть самостоятельным, он ДОЛЖЕН быть ресурсом верхнего уровня, в противном случае - подресурсом.	AAJ, AAX, AX, AJ
[RSG-15]	Параметры запроса ДОЛЖНЫ использоваться вместо путей URL для получения вложенных ресурсов.	AAJ, AAX, AX, AJ

[RSG-16]	Именам ресурсов СЛЕДУЕТ быть существительными для CRUD Web API и глаголами для Intent Web API.	AAJ, AAX
[RSG-17]	Если имя ресурса является существительным, его СЛЕДУЕТ всегда использовать в форме множественного числа. Неправильные формы существительных НЕ СЛЕДУЕТ использовать. Например, вместо `/people` следует использовать `/persons`.	AAJ, AAX
[RSG-18]	Имена ресурсов, сегментов и параметры запросов ДОЛЖНЫ состоять из слов на английском языке с использованием основных английских написаний, представленных в Оксфордском словаре английского языка. Имена ресурсов, которые локализованы в связи с требованиями бизнеса, МОГУТ быть на других языках.	AAJ, AAX, AX, AJ
[RSG-19]	Для Web API СЛЕДУЕТ использовать для согласования типа содержимого HTTP-заголовок запроса `Accept` и HTTP-заголовок ответа `Content-Type`.	AAJ, AAX
[RSG-20]	Web API ДОЛЖЕН поддерживать согласование типов содержимого в соответствии с IETF RFC 7231.	AAJ, AAX, AX, AJ
[RSG-21]	Формат JSON ДОЛЖЕН предполагаться, если не запрашивается определенный тип содержимого.	AAJ, AAX, AX, AJ
[RSG-22]	Web API СЛЕДУЕТ возвращать код состояния "`406 Not Acceptable`", если запрошенный формат не поддерживается.	AAJ, AAX
[RSG-23]	Web API СЛЕДУЕТ отклонять запросы, содержащие неожиданные или отсутствующие заголовки типа содержимого, выдавая кодом состояния HTTP "`406 Not Acceptable`" или "`415 Unsupported Media Type`".	AAJ, AAX
[RSG -24]	Запросам и ответам (соглашение об именовании, формат сообщения, структура данных и словарь данных) СЛЕДУЕТ ссылаться на стандарт ВОИС ST.96 для XML или стандарт ВОИС ST.97 для JSON.	AAX
[RSX-26]	Имена компонентов XML СЛЕДУЕТ предоставлять в UpperCamelCase.	AAX
[RSG-27]	Web API ДОЛЖЕН поддерживать как минимум XML или JSON.	AAJ, AAX, AX, AJ
[RSG-28]	Методы HTTP ДОЛЖНЫ быть ограничены стандартными методами HTTP `POST, GET, PUT, DELETE, OPTIONS, PATCH, TRACE и HEAD`, как указано в IETF RFC 7231 и 5789.	AAJ, AAX, AX, AJ
[RSG-29]	Методы HTTP МОГУТ следовать принципу выбора, который гласит, что должна быть реализована только та функциональность, которая необходима для целевого сценария использования.	AAJ, AAX

[RSG-30]	Некоторые прокси-серверы поддерживают только методы `POST` и `GET`. Чтобы преодолеть эти ограничения, Web API МОЖЕТ использовать метод `POST` с пользовательским заголовком HTTP, "туннелирующим" настоящий метод HTTP. СЛЕДУЕТ использовать пользовательский HTTP-`заголовок X-HTTP-Method`.	AAJ, AAX
[RSG-31]	Если метод HTTP не поддерживается, СЛЕДУЕТ возвратить код состояния HTTP "`405 Method Not Allowed`".	AAJ, AAX
[RSG-32]	Web API СЛЕДУЕТ поддерживать пакетные операции (они же массовые операции) вместо нескольких отдельных запросов для снижения задержки. Такая же семантика должна использоваться для методов HTTP и кодов состояния HTTP. В содержательной части ответа СЛЕДУЕТ иметь информацию обо всех пакетных операциях. Если возникает несколько ошибок, содержательной части ошибки СЛЕДУЕТ иметь информацию обо всех ошибках (в атрибуте details). Все пакетные операции СЛЕДУЕТ выполнять атомарно.	AAJ, AAX
[RSG-33]	Для конечной точки, которая извлекает один ресурс, если ресурс не найден, метод `GET` ДОЛЖЕН вернуть код состояния "`404 Not Found`". Конечные точки, которые возвращают списки ресурсов, просто возвращают пустой список.	AAJ, AAX, AX, AJ
[RSG-34]	Если ресурс получен успешно, метод `GET` ДОЛЖЕН вернуть `200 OK`.	AAJ, AAX, AX, AJ
[RSG-35]	Запрос `GET` ДОЛЖЕН быть идемпотентным (независимым от количества выполнений запроса).	AAJ, AAX, AX, AJ
[RSG-36]	Если длина URI превышает 255 байт, РЕКОМЕНДУЕТСЯ использовать метод `POST` вместо `GET` из-за ограничений `GET`, или если возможно, создавать именованные запросы.	AAJ, AAX
[RSG-37]	Запрос `HEAD` ДОЛЖЕН быть идемпотентным.	AAJ, AAX, AX, AJ
[RSG-38]	Некоторые прокси-серверы поддерживают только методы `POST` и `GET`. Web API СЛЕДУЕТ поддерживать пользовательский заголовок запроса HTTP для переопределения метода HTTP, чтобы преодолеть эти ограничения.	AAJ, AAX
[RSG-39]	`POST`-запрос НЕ ДОЛЖЕН быть идемпотентным в соответствии с IETF RFC 2616.	AAJ, AAX, AX, AJ
[RSG-40]	Если создание ресурса было успешным, в HTTP-заголовоке Location СЛЕДУЕТ содержать URI (абсолютный или относительный), указывающий на созданный ресурс.	AAJ, AAX
[RSG-41]	Если создание ресурса прошло успешно, в ответе СЛЕДУЕТ содержать код состояния "`201 Created`".	AAJ, AAX

[RSG-42]	Если создание ресурса прошло успешно, в полезной части ответа СЛЕДУЕТ по умолчанию содержаться телу созданного ресурса, чтобы клиент мог использовать его без дополнительного вызова HTTP.	AAJ, AAX
[RSG-43]	Запрос `PUT` ДОЛЖЕН быть идемпотентным.	AAJ, AAX, AX, AJ
[RSG-44]	Если ресурс не найден, `PUT` ДОЛЖЕН вернуть код состояния "`404 Not Found`".	AAJ, AAX, AX, AJ
[RSG-45]	Если ресурс успешно обновлен, `PUT` ДОЛЖЕН вернуть код состояния "`200 OK`", если обновленный ресурс возвращен, или "`204 No Content`", если он не возвращен.	AAJ, AAX, AX, AJ
[RSG-46]	Запрос `PATCH` НЕ ДОЛЖЕН быть идемпотентным.	AAJ, AAX, AX, AJ
[RSG-47]	Если Web API реализует частичные обновления, то идемпотентные характеристики `PATCH` РЕКОМЕНДУЕТСЯ принимать во внимание. Для того чтобы сделать его идемпотентным, API МОЖЕТ следовать предложению IETF RFC 5789 об использовании оптимистической блокировки.	AAJ, AAX
[RSG-48]	Если ресурс не найден, PATCH ДОЛЖЕН вернуть код состояния "`404 Not Found`".	AAJ, AAX, AX, AJ
[RSG-50]	Запрос `DELETE` НЕ ДОЛЖЕН быть идемпотентным.	AAJ, AAX, AX, AJ
[RSG-51]	Если ресурс не найден, `DELETE` ДОЛЖЕН вернуть код состояния "`404 Not Found`".	AAJ, AAX, AX, AJ
[RSG-52]	Если ресурс удален успешно, `DELETE` ДОЛЖЕН вернуть статус "`200 OK`", если удаленный ресурс возвращен, или "`204 No Content`", если он не возвращен.	AAJ, AAX, AX, AJ
[RSG-53].	Конечным получателем является либо исходный сервер, либо первый прокси или шлюз, получивший в запросе значение Max-Forwards равное нулю. Запрос `TRACE` НЕ ДОЛЖЕН включать тело (запроса).	AAJ, AAX, AX, AJ
[RSG-54]	Запрос `TRACE` НЕ ДОЛЖЕН быть идемпотентным.	AAJ, AAX, AX, AJ
[RSG-55].	Значение поля заголовка `Via` HTTP ДОЛЖНО служить для отслеживания цепочки запросов.	AAJ, AAX, AX, AJ
[RSG-56]	Поле заголовка HTTP Max-Forwards ДОЛЖНО использоваться для того, чтобы клиент мог ограничить длину цепочки запросов.	AAJ, AAX, AX, AJ
[RSG-57]	Если запрос корректный, в ответ СЛЕДУЕТ поместить все сообщение запроса в теле ответа, с Content-Type "`message/http`".	AAJ, AAX
[RSG-58].	Ответы на `TRACE` НЕ ДОЛЖНЫ кэшироваться.	AAJ, AAX, AX, AJ
[RSG-59]	Код состояния "`200 OK`" СЛЕДУЕТ вернуть в `TRACE`.	AAJ, AAX
[RSG-60]	Запрос `OPTIONS` ДОЛЖЕН быть идемпотентным.	AAJ, AAX, AX, AJ
[RSG-61]	Пользовательские HTTP-заголовки, начинающиеся с префикса "X-", НЕ РЕКОМЕНДУЕТСЯ использовать.	AAJ, AAX

[RSG-62]	Пользовательские HTTP-заголовки НЕ СЛЕДУЕТ использовать для изменения поведения HTTP-методов, за исключением случаев, когда это необходимо для устранения существующих технических ограничений (например, см. [RSG-39]).	AAJ, AAX
[RSG-63]	Соглашение об именовании пользовательских HTTP-заголовков - <организация>-<имя заголовка> (`<organization>-<header name>`) , где полям <организация> и <заголовок> рекомендуется следовать соглашению о kebab-case стиле именования.	AAJ, AAX
[RSG-64]	Web API СЛЕДУЕТ поддерживать единственный метод версионирования сервиса с использованием версионирования URI, например /api/v1/inventors или версионирования заголовка, например Accept-version: v1 или версионирования типа данных, например Accept: application/vnd.v1+json. Версионирование строк запросов НЕ РЕКОМЕНДУЕТСЯ использовать.	AAJ, AAX
[RSG-65]	Схему нумерации версий СЛЕДУЕТ применять с учетом только основного номера версии (например, `/v1`).	AAJ, AAX
[RSG-66]	Сервисные контракты API МОГУТ включать функцию перенаправления конечных точек. Когда потребитель услуг пытается вызвать услугу, может быть возвращен ответ о перенаправлении, чтобы сообщить потребителю услуг о необходимости повторно отправить запрос на новую конечную точку. Перенаправления МОГУТ быть временными или постоянными: - Временное перенаправление - с использованием заголовка HTTP ответа Location и кода статуса HTTP "302 Found" в соответствии с IETF RFC 7231; или - Постоянное перенаправление - с использованием заголовка HTTP-ответа Location и кода статуса HTTP "301 Moved Permanently" в соответствии с IETF RFC 7238.	AAJ, AAX
[RSG-67]	Стратегии жизненного цикла API СЛЕДУЕТ публиковать разработчикам, чтобы помочь пользователям понять, как долго будет поддерживаться та или иная версия.	AAJ, AAX
[RSG-68]	Web API СЛЕДУЕТ поддерживать пагинацию (разбивки на страницы).	AAJ, AAX
[RSG-69]	Пагинированные запросы НЕ МОГУТ быть идемпотентными.	AAJ, AAX
[RSG-70]	Web API ДОЛЖЕН использовать параметры запроса для реализации пагинации.	AAJ, AAX, AX, AJ
[RSG-71]	Веб-интерфейс НЕ ДОЛЖЕН использовать HTTP-заголовки для реализации пагинации.	AAJ, AAX, AX, AJ

[RSG-72]	СЛЕДУЕТ использовать параметры запроса `limit=<количество элементов для доставки> и offset=<количество элементов для пропуска> (limit=<number of items to deliver> and offset=<number of items to skip>)`, где limit - количество возвращаемых элементов (размер страницы), а пропуск (skip) - количество элементов, которые должны быть пропущены (смещение). Если ограничение на размер страницы не указано, СЛЕДУЕТ определить значение по умолчанию - глобальное или для каждой коллекции; смещение по умолчанию ДОЛЖНО быть равно нулю "0": Например, следующий URL является допустимым: https://wipo.int/api/v1/patents?limit=10&offset=20	AAJ, AAX
[RSG-73].	Значения параметров limit и skip СЛЕДУЕТ включить в ответ.	AAJ, AAX
[RSG-74].	Web API СЛЕДУЕТ поддерживать сортировку.	AAJ, AAX
[RSG-75]	Для указания критерия сортировки по нескольким атрибутам ДОЛЖЕН использоваться параметр запроса. Значение этого параметра представляет собой список ключей сортировки, разделенных запятыми, а направления сортировки либо 'asc' для восходящего, либо 'desc' для нисходящего МОГУТ быть добавлены к каждому ключу сортировки, разделенные символом двоеточия ':'. Направление по умолчанию ДОЛЖНО быть определено сервером в случае, если для ключа не указано направление сортировки.	AAJ, AAX, AX, AJ
[RSG-76]	Web API СЛЕДУЕТ возвращать критерии сортировки в ответе.	AAJ, AAX, AX, AJ
[RSG-77]	Web API МОЖЕТ поддерживать расширение тела возвращаемого содержимого. СЛЕДУЕТ использовать параметр запроса `expand=<разделенный запятыми список имен атрибутов>`.	AAJ, AAX
[RSG-78]	Параметр запроса СЛЕДУЕТ использовать вместо путей URL в случае, если веб-интерфейс поддерживает проекцию в формате: `"fields="<разделенный запятыми список имен атрибутов>.`	AAJ, AAX
[RSG-79]	Web API ДОЛЖЕН поддерживать возврат количества элементов в коллекции.	AAJ, AAX, AX, AJ
[RSG-80]	Параметр запроса ДОЛЖЕН использоваться для поддержки возврата количества элементов в коллекции.	AAJ, AAX, AX, AJ
[RSG-81]	СЛЕДУЕТ использовать параметр запроса count для возврата количества элементов в коллекции.	AAJ, AAX

[RSG-82]	Web API МОЖЕТ поддерживать возврат количества элементов в коллекции внутри, т.е. как часть ответа, которая содержит саму коллекцию. При этом ДОЛЖЕН использоваться параметр запроса.	AAJ, AAX, AX, AJ
[RSG-83]	СЛЕДУЕТ использовать параметр запроса `count=true`. Если он не указан, то по умолчанию count следует установить в false.	AAJ, AAX
[RSG-84]	Если Web API поддерживает пагинацию, то СЛЕДУЕТ поддерживать возврат внутри ответа размера коллекции (т.е. общего количества элементов коллекции).	AAJ, AAX
[RSG-85]	Если Web API поддерживает сложные поисковые выражения, СЛЕДУЕТ указывать язык запросов, например, CQL.	AAJ, AAX
[RSG-86]	Сервисный (веб-сервисный) контракт ДОЛЖЕН определять поддерживаемую грамматику (например, поля, функции, ключевые слова и операторы).	AAJ, AAX, AX, AJ
[RSG-87]	ДОЛЖЕН использоваться параметр запроса "q".	AAJ, AAX, AX, AJ
[RSG-88]	На уровне протокола Web API ДОЛЖЕН возвращать соответствующий код состояния HTTP, выбранный из списка стандартных кодов состояния HTTP.	AAJ, AAX, AX, AJ
[RSJ-89]	На уровне приложения Web API ДОЛЖЕН возвращать часть, сообщающую об ошибке с достаточной степенью детализации. Атрибуты `code и message` являются обязательными, атрибут `details` является условно обязательным, атрибуты `target, status, moreInfo` и `internalMessage` являются необязательными.	AAJ, AAX, AX, AJ
[RSG-90]	Ошибки НЕ ДОЛЖНЫ раскрывать критически важные для безопасности данные или внутренние технические детали, такие как стеки вызовов в сообщениях об ошибках.	AAJ, AAX, AX, AJ
[RSG-91].	`Заголовок` HTTP: `Reason-Phrase` (описанный в RFC 2616) НЕ ДОЛЖЕН использоваться для передачи сообщений об ошибках.	AAJ, AAX, AX, AJ
[RSG-92].	Каждой зарегистрированной ошибке СЛЕДУЕТ иметь уникальный идентификатор корреляции. СЛЕДУЕТ использовать пользовательский HTTP-заголовок, которому СЛЕДУЕТ иметь имя Correlation-ID.	AAJ, AAX
[RSG-93].	В формат сервисного контракта СЛЕДУЕТ включать следующее: - Версия API; - Информация о семантике элементов API; - Ресурсы; - Атрибуты ресурса; - Параметры запроса; - Методы; - Типы носителей; - Грамматика поиска (если она поддерживается); - Коды состояния HTTP;	AAJ, AAX, AX, AJ

	- Методы HTTP; - Ограничения и отличительные особенности; и - Безопасность (если есть).	
[RSG-94]	В формат сервисного контракта СЛЕДУЕТ включать запросы и ответы в XML-схеме или JSON-схеме и примеры использования API в поддерживаемых форматах, т.е. XML или JSON.	AAJ, AAX
[RSG-95]	REST API ДОЛЖЕН предоставлять документацию API в виде сервисного контракта.	AAJ, AAX, AX, AJ
[RSG-96]	Реализация Web API, отклоняющаяся от настоящего стандарта, ДОЛЖНА быть явно документирована в сервисном контракте. Если правило отклонения не указано в сервисном контракте, ДОЛЖНО подразумеваться, что соблюдается настоящий стандарт.	AAJ, AAX, AX, AJ
[RSG-97]	Сервисный контракт ДОЛЖЕН позволять генерировать скелетный код клиента API.	AAJ, AAX, AX, AJ
[RSG-98]	Сервисный контракт РЕКОМЕНДУЕТСЯ, что может позволять генерировать скелетный код сервера.	AAJ, AAX
[RSG-99]	Документацию Web API СЛЕДУЕТ писать на RAML или OAS. Пользовательские форматы документации НЕ СЛЕДУЕТ использовать.	AAJ, AAX
[RSG-100]	СЛЕДУЕТ дать возможность потребителю Web API указать тайм-аут сервера для каждого запроса; следует использовать пользовательский HTTP-заголовок. Максимальный тайм-аут сервера СЛЕДУЕТ также применять для защиты ресурсов сервера от чрезмерного использования.	AAJ, AAX
[RSG-101]	Web API СЛЕДУЕТ поддерживать условное получение данных, чтобы гарантировать, что будут получены только измененные данные. СЛЕДУЕТ применять проверку ресурсов на основе содержания, поскольку она точнее.	AAJ, AAX
[RSG-102]	Для реализации Content-based Resource Validation (проверка корректности ресурса по содержимому) HTTP-заголовок `ETag` СЛЕДУЕТ использовать в ответе для кодирования состояния данных. Затем это значение СЛЕДУЕТ использовать в последующих запросах в	AAJ, AAX
	условных HTTP-заголовках (таких как If-Match или If-None-Match — Если соответствует или Если не соответствует). Если данные не были изменены с тех пор, как запрос вернул `ETag`, серверу СЛЕДУЕТ вернуть код состояния "`304 Not Modified`" (если данные не изменены). Этот механизм описан в IETF RFC 7231 и 7232.	
[RSG-103]	Для реализации проверки ресурсов по времени СЛЕДУЕТ использовать HTTP-заголовок `Last-Modified`. Этот механизм описан в IETF RFC 7231 и 7232.	AAJ, AAX

[RSG-104]	Используя версионность ответа, потребитель услуг МОЖЕТ реализовать оптимистическую блокировку.	AAJ, AAX
[RSG-106]	СЛЕДУЕТ использовать HTTP-заголовки ответа `Cache-Control` и `Expires`. Последний МОЖЕТ использоваться для поддержки устаревших клиентов.	AAJ, AAX
[RSG-107]	Web API СЛЕДУЕТ сообщать, поддерживается ли частичная загрузка файлов, отвечая на запросы `HEAD` и передавая в ответ HTTP-заголовки `Accept-Ranges` и `Content-Length`.	AAJ, AAX
[RSG-108]	Web API СЛЕДУЕТ поддерживать частичную загрузку файлов. Следует поддерживать диапазоны из нескольких частей.	AAJ, AAX
[RSG-109]	Web API СЛЕДУЕТ сообщать, поддерживает ли он частичную заливку файлов.	AAJ, AAX
[RSG-110]	Web API СЛЕДУЕТ поддерживать частичную заливку файлов. СЛЕДУЕТ поддерживать составные диапазоны.	AAJ, AAX
[RSG-111]	Поставщику услуг СЛЕДУЕТ возвращать в заголовках ответа HTTP заголовок "`413 Request Entity Too Large`" в случае, если запрос превысил максимально допустимый предел. Пользовательский HTTP-заголовок МОЖЕТ использоваться для указания максимального размера запроса.	AAJ, AAX
[RSG-112]	Если Web API поддерживает обработку настроек, ее СЛЕДУЕТ реализовать в соответствии с IETF RFC 7240, т.е. следует использовать HTTP-заголовок запроса `Prefer`, в HTTP-заголовке ответа СЛЕДУЕТ вернуть `Preference-Applied` (с повторением исходного запроса).	AAJ, AAX
[RSG-113]	Если Web API поддерживает обработку настроек, номенклатура настроек, которые МОГУТ быть установлены с помощью заголовка `Prefer`, ДОЛЖНА быть записана в сервисном контракте.	AAJ, AAX, AJ, AX
[RSG-114]	Если Web API поддерживает локализованные данные, HTTP-заголовок запроса `Accept-Language` ДОЛЖЕН поддерживаться для указания набора естественных языков, которые предпочтительны в ответе, как указано в IETF RFC 7231.	AAJ, AAX, AJ, AX
[RSG-115]	Если API поддерживает длительные операции, им СЛЕДУЕТ быть асинхронными. Должен применяться следующий подход: a) Потребитель услуг активирует сервисную операцию; b) Операция сервиса возвращает код состояния "`202 Accepted`" в соответствии с IETF RFC 7231 (раздел 6.3.3), т.е. запрос был принят к обработке, но обработка не была завершена.	AAJ, AAX

	c) Местоположение созданной задачи в очереди также возвращается с HTTP-заголовком `Location`; и Потребитель услуги обращается к возвращаемому местоположению, чтобы узнать, доступен ли ресурс. Если ресурс недоступен, в ответе СЛЕДУЕТ иметь код состояния "`200 OK`", содержать статус задачи (например, ожидание) и МОЖЕТ содержаться другая информация (например, индикатор выполнения и/или ссылка для отмены или удаления задачи с помощью HTTP-метода `DELETE`). Если ресурс доступен, в ответе СЛЕДУЕТ привести код состояния "`303 See Other`", а в заголовоке HTTP `Location` СЛЕДУЕТ привести URL для получения результатов задачи.	
[RSG-116]	Конфиденциальность: Различные API и информация в API ДОЛЖНЫ быть идентифицированы, классифицированы и защищены от несанкционированного доступа, раскрытия и перехвата в любое время. Должны соблюдаться принципы наименьших привилегий, нулевого доверия, необходимости знать и необходимости делиться.	AAJ, AAX, AJ, AX
[RSG-117]	Обеспечение целостности: Различные API и информация в API ДОЛЖНЫ быть защищены от несанкционированной модификации, дублирования, повреждения и уничтожения. Информация ДОЛЖНА модифицироваться через подтвержденные транзакции и интерфейсы. Системы ДОЛЖНЫ обновляться с использованием утвержденных процессов управления конфигурацией, управления изменениями и управления исправлениями.	AAJ, AAX, AJ, AX
[RSG-118]	Доступность: Различные API и информация в API ДОЛЖНЫ быть доступны авторизованным пользователям в установленное время в соответствии с соглашениями об уровне обслуживания (SLA), политиками контроля доступа и определенными бизнес-процессами.	AAJ, AAX, AJ, AX
[RSG-119]	Неотказуемость (безотзывность): Каждая обрабатываемая транзакция или действие, выполняемое API, ДОЛЖНЫ обеспечивать неотказуемость посредством реализации надлежащего аудита, авторизации, аутентификации, а также реализации безопасных путей, а также и сервисов и механизмов неотказуемости.	AAJ, AAX, AJ, AX

[RSG-120]	Аутентификация, авторизация, аудит: Пользователи, системы, API или устройства, вовлеченные в критические транзакции или действия, ДОЛЖНЫ быть аутентифицированы, авторизованы с помощью служб контроля доступа на основе ролей или атрибутов и поддерживать разделение обязанностей. Кроме того, все действия ДОЛЖНЫ регистрироваться, а надежность аутентификации должна увеличиваться в соответствии с соответствующим информационным риском.	AAJ, AAX, AJ, AX
[RSG-121]	При разработке API ТРЕБУЕТСЯ тщательно учитывать угрозы, случаи вредоносного использования, методы безопасного кодирования, безопасность транспортного уровня и тестирование безопасности, особенно: - PUTs и POSTs - т.е.: какие изменения внутренних данных потенциально могут быть использованы для атаки или ложной информации; - DELETES - т.е.: может использоваться для удаления содержимого внутреннего хранилища ресурсов; - Белый список допустимых методов - для обеспечения того, чтобы допустимые методы HTTP были должным образом ограничены, в то время как другие будут возвращать правильный код ответа; и - Хорошо известные атаки следует учитывать на этапе моделирования угроз в процессе проектирования, чтобы не допустить увеличения риска угроз. Угрозы и меры по их снижению, определенные в OWASP Top Ten Cheat Sheet, ДОЛЖНЫ быть приняты во внимание.	AAJ, AAX, AJ, AX
[RSG-122]	При разработке API следует придерживаться стандартов и лучших практик, перечисленных ниже: - Лучшие практики безопасного кодирования: Принципы безопасного кодирования OWASP; - Безопасность Rest API: Памятка по безопасности REST;	AAJ, AAX, AJ, AX
	- Экранирование ввода и защита от межсайтового скриптинга: OWASP XSS Cheat Sheet - Предотвращение SQL-инъекций: Памятка OWASP по SQL-иньекциям, памятка OWASP по Parameterization; и - Безопасность транспортного уровня: Памятка OWASP по защите транспортного уровня.	

[RSG-123]	Тестирование безопасности и оценка уязвимостей ДОЛЖНЫ проводиться для обеспечения безопасности и устойчивости API к угрозам. Это требование МОЖЕТ быть выполнено путем использования статического и динамического тестирования безопасности приложений (SAST/DAST), автоматизированных инструментов управления уязвимостями и тестирования на проникновение.	AAJ, AAX, AJ, AX
[RSG-124]	Защищенные службы ДОЛЖНЫ предоставлять конечные точки HTTPS только с использованием TLS 1.2 или выше с набором шифров, включающим ECDHE для обмена ключами.	AAJ, AAX, AJ, AX
[RSG-125]	При рассмотрении протоколов аутентификации для обеспечения транспортной безопасности СЛЕДУЕТ использовать совершенную прямую секретность. Не СЛЕДУЕТ допускать использование небезопасных криптографических алгоритмов и обратную совместимость с SSL 3 и TLS 1.0/1.1.	AAX, AAJ
[RSG-126]	Для обеспечения максимальной безопасности и доверия следует устанавливать соединение через VPN IPSEC между сайтами для дополнительной защиты информации, передаваемой через незащищенные сети.	AAX, AAJ
[RSG-127]	Пользовательскому приложению СЛЕДУЕТ проверять цепочку сертификатов TLS при выполнении запросов к защищенным ресурсам, включая проверку списка отзыва сертификатов.	AAX, AAJ
[RSG-128]	Защищенные службы ДОЛЖНЫ использовать только действительные сертификаты, выданные доверенным центром сертификации (ЦС).	AAX, AAJ
[RSG-129]	Токены СЛЕДУЕТ подписывать с использованием безопасных алгоритмов подписи, соответствующих стандарту цифровой подписи (DSS) FIPS -186-4. Следует рассматривать алгоритм цифровой подписи RSA или алгоритм ECDSA.	AAX, AAJ
[RSG-130]	Анонимная аутентификация ДОЛЖНА использоваться только тогда, когда клиенты и используемое ими приложение получают доступ к информации или функциям с низким уровнем чувствительности, которые не должны требовать аутентификации, например, к публичной информации.	AAJ, AAX, AJ, AX
[RSG-131]	Аутентификация с использованием имени пользователя и пароля или хэш-пароля НЕ ДОЛЖНА разрешаться.	AAJ, AAX, AJ, AX
[RSG-132]	Если сервис защищен, СЛЕДУЕТ использоваться Open ID Connect.	AAX, AAJ

[RSG-133]	При использовании JSON Web Token (JWT) секрету JWT СЛЕДУЕТ обладать высокой энтропией, чтобы увеличить трудоемкость атаки грубой силой; токенам TTL и RTTL СЛЕДУЕТ быть как можно короче; и НЕ СЛЕДУЕТ хранить конфиденциальную информацию в полезной нагрузке JWT.	AAX, AAJ
[RSG-134]	В запросах `POST/PUT` конфиденциальные данные СЛЕДУЕТ передавать в теле или в заголовках запроса.	AAX, AAJ
[RSG-135]	В запросах `GET` конфиденциальные данные СЛЕДУЕТ передавать в заголовке HTTP.	AAX, AAJ
[RSG-136]	Для того чтобы минимизировать задержки и уменьшить связь между защищенными сервисами, решение об управлении доступом СЛЕДУЕТ принимать на локальном уровне в конечных точках REST.	AAX, AAJ
[RSG-137]	API-ключи СЛЕДУЕТ использовать и для защищенных и для публичных сервисов, чтобы не перегружать поставщика услуг многочисленными запросами (атаки типа "отказ в обслуживании"). Для защищенных сервисов API-ключи МОГУТ использоваться для монетизации (приобретенные тарифные планы), применения политики использования (QoS) и мониторинга.	AAX, AAJ
[RSG-138]	Ключи API МОГУТ быть объединены с заголовком HTTP запроса user-agent для различения пользователя-человека и программного агента, как указано в IETF RFC 7231.	AAX, AAJ
[RSG-139]	Сервис-провайдеру СЛЕДУЕТ возвращать вместе с заголовками ответа HTTP текущий статус использования. Следующие данные ответа МОГУТ быть возвращены: предел скорости (rate limit) - предел скорости (в минуту), установленный в системе; rate limit remaining - оставшееся количество запросов, разрешенных в течение текущего тайм-слота (-1 означает, что лимит превышен); и сброс лимита скорости (rate limit reset) - время (в секундах), оставшееся до сброса счетчика запросов.	AAX, AAJ
[RSG-140]	Сервис-провайдеру СЛЕДУЕТ возвращать код состояния `"429 Too Many Requests"`, если запросы поступают слишком быстро.	AAX, AAJ
[RSG-141]	Ключи API ДОЛЖНЫ быть отозваны, если клиент нарушает соглашение об использовании, как указано в ОИС (Организации интеллектульной собственности).	AAJ, AAX, AJ, AX
[RSG-142]	Ключи API СЛЕДУЕТ передавать с помощью пользовательских HTTP-заголовков. Их НЕ СЛЕДУЕТ передавать с помощью параметров запроса.	AAX, AAJ
[RSG-143]	Ключи API СЛЕДУЕТ `генерировать` случайным образом.	AAX, AAJ

[RSG-144]	Безопасные и надежные сертификаты ДОЛЖНЫ выдаваться взаимно доверенным центром сертификации (ЦС) через процесс установления доверия или перекрестной сертификации.	AAJ, AAX, AJ, AX
[RSG-145]	Сертификаты, совместно используемые клиентом и сервером, СЛЕДУЕТ использовать для снижения рисков безопасности идентификации, характерных для чувствительных систем и привилегированных действий, например, X.509.	AAJ, AAX, AJ, AX
[RSG-146]	Для высокопривилегированных служб в двусторонней взаимной аутентификации между клиентом и сервером СЛЕДУЕТ использовать сертификаты для обеспечения дополнительной защиты.	AAX, AAJ
[RSG-147]	Для высокорисковых приложений, систем, обрабатывающих очень чувствительную информацию или выполняющих привилегированные действия СЛЕДУЕТ реализовать многофакторную аутентификация для снижения рисков идентификации.	AAX, AAJ
[RSG-148]	Если REST API является публичным, HTTP-заголовок Access-Control-Allow-Origin ДОЛЖЕН иметь значение '*'.	AAJ, AAX, AJ, AX
[RSG-149]	Если REST API защищенный, СЛЕДУЕТ использовать CORS, если это возможно. В противном случае, JSONP МОЖЕТ использоваться в качестве запасного варианта, но только для GET-запросов, например, когда пользователь получает доступ с помощью старого браузера. НЕ СЛЕДУЕТ использовать Iframe.	AAX, AAJ

[Приложение II следует].

ПРИЛОЖЕНИЕ II

Словарь REST для ИС

Версия 1.1

Пересмотр одобрен Комитетом по Стандартам ВОИС (КСВ)
на своей десятой сессии 25 ноября 2022

1.Следующий словарь по интеллектуальной собственности (далее ИС) приводится в Таблице 5 в качестве примеров базовых (/basic) параметров запросов сервисов в RESTful модели. Ведомства ИС, вероятно, столкнутся с необходимостью разработки более сложных запросов и разнообразной содержательной части ответа в соответствии со своими деловыми потребностями. Параметры в этой таблице являются примерами элементов ST.96 в нижнем регистре (lowerCamelCase), используемых для ответа в формате JSON. Полный словарь данных ST.96 ИС и XML-схемы ИС можно получить по адресу: https://www.wipo.int/standards/en/st96/v5-0/.

[Примечание редакции: в будущем планируется предоставить ссылку на более полный список словаря REST ИС XML и JSON, который будет динамически поддерживаться на постоянной основе по мере развития элементов ИС и словаря.]

Таблица 5: Пример бизнес-лексики API в нижнем регистре (lowerCamelCase) в соответствии с ST.96 XSDs

Сфера применения (бизнес-домен(ы))	Название ресурса (ов)	Имя параметра	Описание
ВСЕ	/trademarks /patents /designs (товарные знаки, патенты, промобразцы)	`st13ApplicationNumber`	Номер заявки на поданную ИС, с использованием формата WIPO ST.13, который представляет собой строку из нескольких значений, включая номер национальной заявки, тип ИС и страну/организацию.
ВСЕ	/trademarks /patents /designs	`applicationNumber`	Номер заявки на поданную ИС в формате национального ведомства.

МНОЖЕСТВО	/trademarks /designs	internationalRegistrationNumber	Международный регистрационный номер права ИС. Для товарных знаков это относится к Мадридской системе В отношении промышленных образцов это относится к Гаагской системе.
ВСЕ	/trademarks /patents /designs	availableDocument	Единичная входная запись документа для критериев поиска API DocList
ВСЕ	/trademarks /patents /designs	sortingCriteria	Критерий сортировки, используемый API DocList
ВСЕ	/trademarks /patents /designs	receivingOfficeCode	Код ОИС (организации интеллектуальной собственности), в формате WIPO ST.2.
ВСЕ	/trademarks /patents /designs	receivingOfficeDate	Дата получения в ОИС
Товарные знаки	/trademarks	registrationDate	Дата, зарегистрированная в ОИС
		applicationDate	Дата подачи заявления
		markCurrentStatusCode	Код текущего правового статуса заявки
		markCurrentStatusDate	Дата текущего юридического статуса заявки
Патенты	/patents	filingDate	Дата подачи заявления
		grantPublicationDate	Дата публикации выдачи патента
		fileReferenceIdentifier	Справочный номер заявителя
		applicationBodyStatus	Статус органа заявки
		statusEventData	Данные, связанные с событием правового статуса в отношении конкретной патентной заявки
		keyEventCode	Код, обозначающий широкое, высокоуровневое событие, которое охватывает наиболее общие и важные ситуации в категории
Промышленные образцы	/designs	applicationDate	Дата подачи заявления
		designApplicationCurrentStatus	Категория текущего правового статуса заявки на промышленный образец
		designApplicationCurrentStatusDate	Дата текущего юридического статуса заявки на промышленный образец

2. Следующие `технические` параметры запроса, определенные в таблице 6 следует применять во всех сервисах REST API:

Таблица 6: Техническая лексика API

Запрос/Параметр в пути (запроса)	Значение параметра Тип данных	Ограничение	Формат	Описание	Правило проектирования
format	string		`type/subtype; parameter=value` в соответствии с RFC7231, 3.1.1.1. Тип носителя	Используется для согласования типа содержимого (предпочтительнее заголовок запроса HTTP).	[RSG-19]
v	string		`v%`, где % - целое положительное число	Используется для определения версии сервиса (предпочтительно указывать версию как сегмент пути в URL)	[RSG-64]
limit	integer	положительное число	`limit=10`	Размер страницы, используемый для пагинации (разбивки на страницы)	[RSG-73].
offset	integer	положительноее число; по умолчанию 0	`offset=5`	Смещение, используемое для пагинации	[RSG-73].
sort	список строк (string), разделенных запятыми	Возможные значения: - asc - desc (восходящий, нисходящий)	`sort=key1:asc, key2:desc`	Многоатрибутный критерий сортировки	[RSG-74] – [RSG-76]
expand	список строк (string), разделенных запятыми		`expand=key1,key2`	Используется для расширения тела возвращаемого содержимого	[RSG-77]

count	boolean	По умолчанию false	`count=true`	Возвращает количество элементов в коллекции (может быть встроенно в тело ответа)	[RSG-81]
apiKey	string		`apiKey=abcdef12345`	Используется для указания ключа Web API (следует предпочитать использовать HTTP-заголовок).	[RSG-137] - [RSG-138]

[Приложение III следует].

ПРИЛОЖЕНИЕ III

РУКОВОДСТВО ПО RESTFUL WEB API И ТИПОВОЙ СЕРВИСНЫЙ КОНТРАКТ

Версия 1.1

Пересмотр одобрен Комитетом по Стандартам ВОИС (КСВ)
на своей десятой сессии 25 ноября 2022

1. В Приложении III приведены два примера моделей спецификаций API, соответствующих стандарту, которые призваны служить руководством для ведомств интеллектуальной собственности (ВИС), желающих разработать веб-сервисы в соответствии с настоящим стандартом. Подробная информация о двух примерах моделей приведена ниже и в Дополнениях A и B.

2. Следует отметить, что примеры моделей были созданы с использованием гибридного подхода, состоящего из подходов "контракт - первый" и "код - первый".

Пример модели DocList

3. Первая из примеров моделей была создана на основе набора веб-сервисов IP5[17] Office Open Portal Dossier (OPD — досье открытого портала). API DocList (список документов) предоставляет список соответствующих патентных документов, связанных, по крайней мере, с номером заявки или публикации.

Пример модели правового статуса патентов

4. Второй из примеров моделей является API патентного правового статуса, который предоставляет либо историю событий правового статуса для конкретного номера заявки, либо детали конкретного события правового статуса.

[Дополнения A и B к Приложению III следуют].

[17] Ведомства IP5 состоят из Китайской национальной администрации интеллектуальной собственности (CNIPA), Европейского патентного ведомства (EPO), Японского патентного ведомства (JPO), Корейского ведомства интеллектуальной собственности (KIPO) и Ведомства по патентам и товарным знакам США (USPTO).

ДОПОЛНЕНИЕ A

ПРИМЕР МОДЕЛИ DOCLIST

1. Дополнение A содержит ссылку на zip-файл, включающий документ с требованиями, в котором описаны форматы запросов и ответов, спецификация YAML и компоненты XSD.

2. Дополнение A доступно по адресу:
https://www.wipo.int/standards/en/st90/annex-iii_appendix_a_V1_0.zip

ДОПОЛНЕНИЕ B

ПРИМЕР МОДЕЛИ ПАТЕНТНО-ПРАВОВОГО СТАТУСА

1. В Дополнении B приведена ссылка на zip-файл, содержащий спецификацию API на языке RAML, примеры данных и списки перечислений стандарта ВОИС ST.96.

2. Приложение B доступно по адресу:
https://www.wipo.int/standards/en/st90/annex-iii_appendix_b_V1_0.zip

[Приложение IV следует].

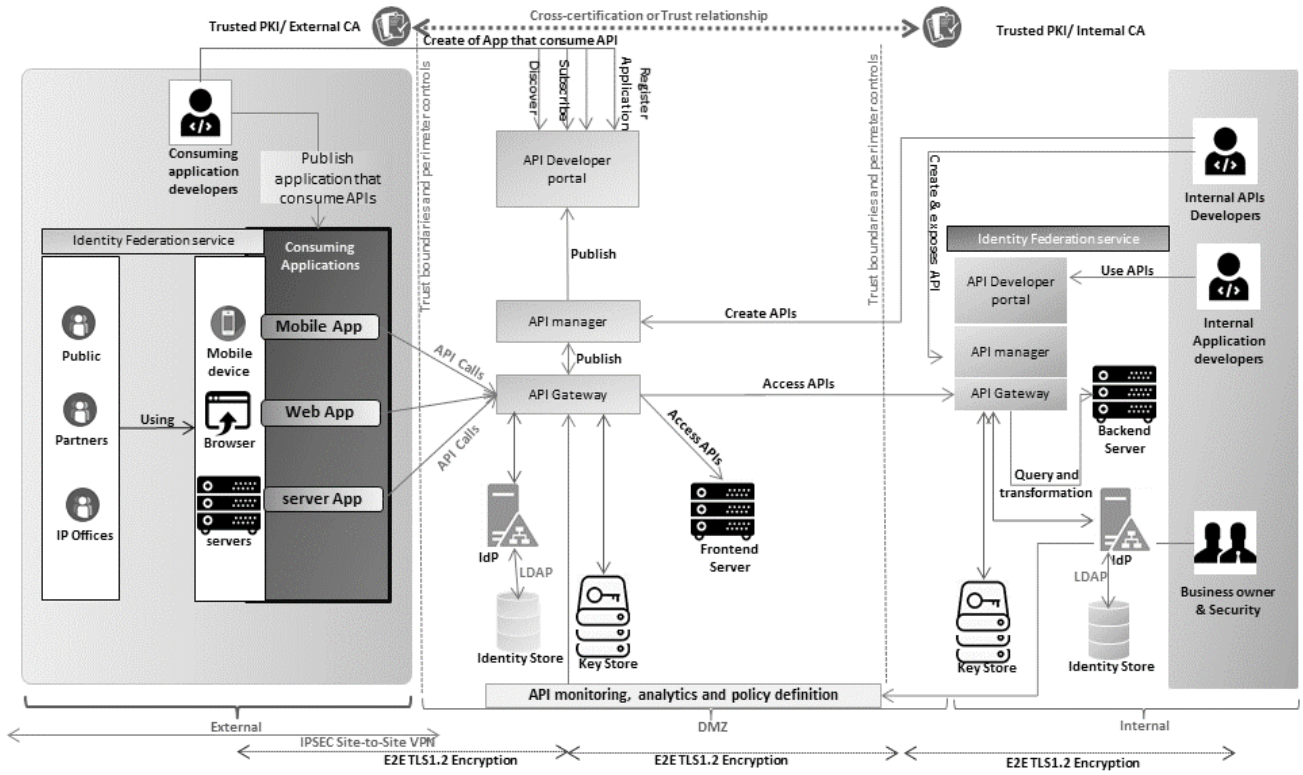
ПРИЛОЖЕНИЕ IV

ЛУЧШИЕ ПРАКТИКИ АРХИТЕКТУРЫ БЕЗОПАСНОСТИ ВЫСОКОГО УРОВНЯ

Версия 1.1

Пересмотр одобрен Комитетом по Стандартам ВОИС (КСВ)
на своей десятой сессии 25 ноября 2022

1. Архитектура безопасности определяет сервисы и механизмы, которые должны быть реализованы для обеспечения соблюдения установленных политик и правил, а также обеспечивает основу для дальнейшей стандартизации и автоматизации безопасности. Основные сервисы и механизмы этого фреймворка API безопасности (портал разработки, менеджер API и шлюз API) обеспечивают группировку функциональных возможностей. Эти функции могут быть предоставлены отдельными приложениями, разработкой кода на заказ, с помощью COTS-продуктов или путем использования существующих технологий, которые могут быть настроены для предоставления этих функций / услуг. Некоторые из функций могут пересекаться или объединяться в один или несколько продуктов в зависимости от используемого поставщика.



2. В рекомендуемой архитектуре безопасности СЛЕДУЕТ иметь следующие сервисы и механизмы безопасности API:

- Портал Web API (API portal) для обеспечения таких функций, как API доступа (API Discover), API (API analytics) для аналитики, доступ к спецификациям и описанию, включая SLA, социальная сеть и FAQ (часто задаваемые вопросы);

- Web API управления (API manager) для обеспечения централизованного администрирования API и управления каталогами API, управления регистрацией и включением различных сообществ разработчиков API, управления жизненным циклом API, применения предварительно определенных профилей безопасности и управления жизненным циклом политик безопасности;

- Шлюзовое Web API (API Gateway) для обеспечения возможностей автоматизации безопасности, включая, но не ограничиваясь, централизованной защитой от угроз, API централизованной аутентификации, авторизацией, ведением журнала, применением политик безопасности, шифрованием сообщений, мониторингом и аналитикой;

- Сервис мониторинга и аналитики Web API для обеспечения таких функций, как расширенный мониторинг сервисов API, аналитика, использование профилей для базовых уровней безопасности, изменения использования и запросов;

- Хранилище учетных данных для обеспечения возможности безопасного хранения ключей API, секретов, сертификатов и т.д.;

- Доверенный центр сертификации (ЦС) для выпуска безопасных сертификатов и установления доверия между различными ведомствами;

- Система управления информацией и событиями безопасности (SIEM) для корреляции журналов безопасности и расширенной аналитики и мониторинга безопасности;

- Провайдер идентификации для управления идентификационными данными, хранящимися в каталогах LDAP, и обеспечения аутентификации; и

- Веб-приложение, которое выполняет регулярное сканирование безопасности и проводит анализ на основе доверенной базы безопасности, такой как OWASP Top 10.

[Приложение V следует].

ПРИЛОЖЕНИЕ V

КОДЫ СОСТОЯНИЯ HTTP

Версия 1.1

Пересмотр одобрен Комитетом по Стандартам ВОИС (КСВ)
на своей десятой сессии 25 ноября 2022

1. Важно согласовывать ответы с соответствующим кодом состояния HTTP и следовать стандартным кодам HTTP. Помимо соответствующего кода состояния, в теле ответа HTTP рекомендуется иметь полезное и краткое описание ошибки. Ответы должны быть конкретными и четкими, чтобы потребители могли быстро сделать вывод при использовании API.

2. Набор кодов состояния HTTP определен на основе в RFC7231. Перечисленные ниже коды состояния следует использовать в API, где это применимо.

3. Определены следующие категории кодов состояния ответа:

- 1xx: Informational - передает информацию на уровне протокола передачи;
- 2xx: Success - Указывает на то, что запрос клиента был успешно принят;
- 3xx: Redirection - Указывает на то, что клиент должен предпринять некоторые дополнительные действия, чтобы завершить свой запрос;
- 4xx: Client Error - эта категория кодов состояния ошибки указывает на клиентов; и
- 5xx: Server Error — коды ошибок, за которые ответствен сервер.

4.В следующей таблице объединены коды состояния HTTP и приведены ссылки на соответствующие RFC IETF.

Значение	**Описание**	**Ссылка**
100	Continue - Продолжить	[RFC7231, раздел 6.2.1].
101	Switching Protocols -Протоколы коммутации	[RFC7231, раздел 6.2.2].
102	Processing - Обработка	[RFC2518]
103	Early Hints - Ранние подсказки (перед финальным ответом)	[RFC8297]
104-199	Не описаны	
200	OK - Успешно	[RFC7231, раздел 6.3.1].
201	Created - Создано	[RFC7231, раздел 6.3.2].
202	Accepted - Принято	[RFC7231, раздел 6.3.3].
203	Non-Authoritative Information — Неофициальная информация	[RFC7231, раздел 6.3.4].
204	No Content - Нет содержимого	[RFC7231, раздел 6.3.5].
205	Reset Content - Сброс содержимого	[RFC7231, раздел 6.3.6].

206	Partial Content - Часть содержимого	[RFC7233, раздел 4.1].
207	Multi-Status - Мульти-статус	[RFC4918]
208	Already Reported - Уже сообщено	[RFC5842].
209-225	Не описаны	
226	IM Used - Используемый для управления экземпляром	[RFC3229].
227-299	Не описаны	
300	Multiple Choices - Множественный выбор	[RFC7231, раздел 6.4.1].
301	Moved Permanently -Перемещено постоянно	[RFC7231, раздел 6.4.2].
302	Found - Найдено	[RFC7231, раздел 6.4.3].
303	See Other - См. другое	[RFC7231, раздел 6.4.4].
304	Not Modified - Не изменен	[RFC7232, раздел 4.1].
305	Use Proxy - Использовать прокси-сервер	[RFC7231, раздел 6.4.5].
306	(Unused) - (Не используется)	[RFC7231, раздел 6.4.6].
307	Temporary Redirect -Временное перенаправление	[RFC7231, раздел 6.4.7].
308	Permanent Redirect -Постоянное перенаправление	[RFC7538].
309-399	Не описаны	
400	Bad Request - Плохой запрос	[RFC7231, Раздел 6.5.1]
401	Unauthorized -Неавторизованный (требует авторизации)	[RFC7235, раздел 3.1].
402	Payment Required -Требуется оплата	[RFC7231, раздел 6.5.2].
403	Forbidden - Запрещено	[RFC7231, раздел 6.5.3].
404	Not Found - Не найдено	[RFC7231, раздел 6.5.4].
405	Method Not Allowed - Метод не разрешен	[RFC7231, Раздел 6.5.5].
406	Not Acceptable -Неприемлемо	[RFC7231, раздел 6.5.6].
407	Proxy Authentication Required - Требуется аутентификация прокси-сервера	[RFC7235, раздел 3.2].
408	Request Timeout - Таймаут запроса	[RFC7231, раздел 6.5.7].
409	Conflict - Конфликт	[RFC7231, раздел 6.5.8].
410	Gone — Исчезло (доступ более невозможен)	[RFC7231, раздел 6.5.9].
411	Length Required — требуется длина	[RFC7231, раздел 6.5.10].
412	Precondition Failed -Условие не выполнено	[RFC7232, раздел 4.2][RFC8144, раздел 3.2]
413	Payload Too Large - Слишком большое полезное содержимое	[RFC7231, раздел 6.5.11].
414	URI Too Long - URI Слишком длинный URI	[RFC7231, раздел 6.5.12].
415	Unsupported Media Type - Неподдерживаемый тип носителя (данных)	[RFC7231, раздел 6.5.13][RFC7694, раздел 3]
416	Range Not Satisfiable -Диапазон не удовлетворительный	[RFC7233, раздел 4.4].
417	Expectation Failed — неудовлетворительные ожидаемые данные	[RFC7231, Раздел 6.5.14].
418-420	Не описаны	

421	Misdirected Request - Неправильно направленный запрос	[RFC7540, раздел 9.1.2].
422	Unprocessable Entity -Необрабатываемая сущность	[RFC4918]
423	Locked — Заблокировано (закрыто)	[RFC4918]
424	Failed Dependency -Неудачная зависимость	[RFC4918]
425	Не описан	
426	Upgrade Required -Требуется модернизация	[RFC7231, Раздел 6.5.15].
427	Не описан	
428	Precondition Required -Требуется предварительное условие	[RFC6585]
429	Too Many Requests -Слишком много запросов	[RFC6585]
430	Не описан	
431	Request Header Fields Too Large - Слишком большие поля заголовка запроса	[RFC6585]
432-450	Не описаны	
451	Unavailable For Legal Reasons - Недоступно по юридическим причинам	[RFC7725]
452-499	Не описаны	
500	nternal Server Error -Внутренняя ошибка сервера	[RFC7231, раздел 6.6.1].
501	Not Implemented - Не реализовано	[RFC7231, раздел 6.6.2].
502	Bad Gateway - Ошибка шлюза	[RFC7231, раздел 6.6.3].
503	Service Unavailable - Сервис недоступен	[RFC7231, раздел 6.6.4].
504	Gateway Timeout - Таймаут шлюза	[RFC7231, раздел 6.6.5].
505	HTTP Version Not Supported - Версия HTTP не поддерживается	[RFC7231, раздел 6.6.6].
506	Variant Also Negotiates -Вариант также взаимодействует	[RFC2295]
507	Insufficient Storage -Недостато места для хранения	[RFC4918]
508	Loop Detected - Обнаружена петля	[RFC5842].
509	Не описан	
510	Not Extended - Не расширено (не удался расширенный запрос)	[RFC2774]
511	Network Authentication Required - Требуется сетевая аутентификация	[RFC6585]
512-599	Не описано	

[Приложение VI следует].

ПРИЛОЖЕНИЕ VI

ТЕРМИНЫ ПРЕДСТАВЛЕНИЯ

Версия 1.1

Пересмотр одобрен Комитетом по Стандартам ВОИС (КСВ)
на своей десятой сессии 25 ноября 2022

Срок	Определение	Тип данных
Amount	Денежное выражение.	Number - число
Category	Специально определенное подразделение или подмножество в системе классификации, в которой все элементы имеют одну и ту же систематическую концепцию.	String - Строка
Code	Комбинация из одной или нескольких цифр, букв или специальных символов, которая подставляется для конкретного значения. Представляет собой конечные, заранее определенные значения или свободный формат.	String - Строка
Date	Представление конкретного момента времени, выраженное годом, месяцем и днем.	String - Строка
Directory	Всегда предшествует PATH	String - Строка
Document	CLOB расшифровывается как "Character Large Object" (большой символьный объект), что является специфическим типом данных почти для всех баз данных. Проще говоря, CLOB - это указатель на текст, хранящийся вне таблицы в специальном блоке. Используется для документов XML. Состоит из текстовой информации об обмениваемой международной регистрации товарного знака. Теги XML идентифицируют элементы данных, относящиеся к такой информации. Команда разработчиков TIS - Madrid может определить атрибут XML_DOC как CLOB, указатель на тегированные данные, хранящиеся вне таблицы в выделенном блоке.	String - Строка
Identifier	Комбинация из одного или нескольких целых чисел, букв, специальных символов, которая однозначно идентифицирует конкретный экземпляр бизнес-объекта, но может не иметь легко формулируемого смысла.	String - Строка
Indicator	Сигнал о наличии, отсутствии или необходимости чего-либо. Рекомендуемые значения: “Y”, “N” и, при необходимости, "?".	Boolean — Булевый (логический)

Measure	Мера - это числовое значение, определяемое путем измерения объекта вместе с указанной единицей измерения. MeasureType используется для представления физического измерения, такого как температура, длина, скорость, ширина, вес, объем, широта объекта. Более точно, MeasureType следует использовать для измерения внутренних или физических свойств объекта, рассматриваемого как единое целое.	`Number - Номер`
Name	Обозначение объекта, выраженное в слове или фразе.	`String - Строка`
Number	Строка цифровых или буквенно-цифровых символов, выражающая метку, значение, количество или идентификацию.	`Number, String - Число, Строка`
Percent	Число, представляющее собой часть целого, которое делится на 100.	`Number - Номер`
Quantity	Количество - это подсчитанное число неденежных единиц, возможно, включая дроби. Количество (Quantity) используется для представления подсчитанного числа вещей. Количество должно использоваться для простых свойств объекта, рассматриваемого как составная часть, коллекция или контейнер, для количественной оценки или подсчета его компонентов. Количество всегда должно выражать подсчитанное количество вещей, а свойство быть такое как общее, отгруженое, загруженое, сохраненое. `QuantityType` следует использовать для компонентов, которые требуют информации о единицах измерения; а `xsd:nonNegativeInteger` следует использовать для счетных компонентов, которые не требуют информации о единицах измерения.	`Номер`
Rate	Количество или сумма, измеренная по отношению к другому количеству или сумме.	`Number - Номер`
Text	Неформатированная строка символов, обычно в виде слов. (Включает: Аббревиатура, Комментарии).	`String - Строка`
Time	Обозначение определенного хронологического момента в рамках периода.	`Date - Дата`
DateTime	Зафиксированные дата и время события в момент его наступления.	`Date - Дата`
URI	Единый идентификатор ресурса, определяющий, где находится файл.	`String - Строка`

[Приложение VII следует].

ПРИЛОЖЕНИЕ VII

Публикация управления жизненным циклом API

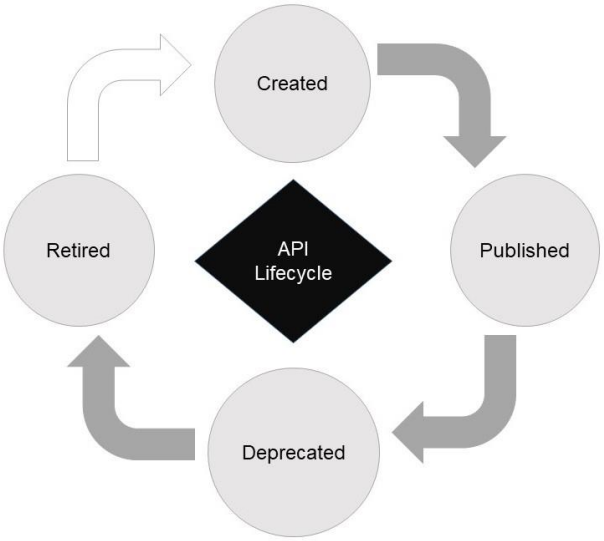
Версия 1.1

Пересмотр одобрен Комитетом по Стандартам ВОИС (КСВ)
на своей десятой сессии 25 ноября 2022

1. В данном приложении представлен краткий обзор управления жизненным циклом API и предложены ключевые фрагменты информации, которые должны быть опубликованы в программном документе ведомства ИС, чтобы помочь потребителям API понять, как лучше использовать эти API.

2. Управление жизненным циклом API является важнейшим аспектом стратегии API, поскольку оно обеспечивает основу для жизни API от создания до выхода из эксплуатации. Оно полезно как для внутренних разработчиков и операционных команд, так и для внешних потребителей API. Для внутренних разработчиков она помогает создать структуру и определить ожидания от разработки API, а для операционных команд - понять требования к поддержке. Для потребителей API, как внутренних, так и внешних, оно дает неформальный договор чего можно ожидать при использовании конкретного API. Это станет ясно, когда каждый этап жизненного цикла будет представлен ниже.

3. Опубликованные жизненные циклы API могут состоять из простых 4-шаговых процессов или более сложных, включающих до 10 и более шагов. Однако в большинстве случаев жизненные циклы с большим количеством этапов выглядят более подробными версиями жизненных циклов с меньшим количеством этапов. Поэтому в данном документе мы сосредоточимся на базовом 4-этапном процессе, необходимом для регистрации жизненного цикла API: Created -> Published -> Deprecated -> Retired. (Создано -> Опубликовано -> Устарело -> Выведено). Любой опубликованный документ о жизненном цикле API должен включать, по крайней мере, описание этих четырех этапов, которыми управляет ведомство по ИС.



Создано (Created)

4. При создании API внимание сосредоточено на проектировании, реализации и документировании API. Важнейшим моментом на этапе создания является рассмотрение цели API и общей структуры, необходимой для "защиты от будущего" API, насколько это возможно. В идеале API должен соответствовать набору внутренних и внешних стандартов, таких как рекомендации, включенные в действующий Стандарт. Если API будет монетизироваться, то на этом этапе следует рассмотреть возможность определения стратегии монетизации.

Опубликовано (Published)

5. После создания API его необходимо опубликовать. Он должен быть версионирован с использованием стандартной стратегии версионирования, и должна быть предоставлена документация, включающая спецификацию API и примеры запросов и ответов (см. [RSG-64]-[RSG-65]). Опубликованное API используется приложениями. Заметьте, что исправления и улучшения могут быть включены на этапе публикации.

Устарело (Deprecated)

6. В какой-то момент API теряет полезность. Он либо заменяется более новой версией API, либо утрачивает актуальность в силу каких-то внешних или внутренних факторов. Необходимо связаться с потребителями API и подготовиться к удалению API из каталога. На данном этапе, скорее всего, будут исправляться только основные ошибки API.

Выведено (Retired)

7. На этом этапе API выводится из эксплуатации. Процесс должен состоять из отключения доступа к API и удаления его с платформы API. Следует рассмотреть вопрос о том, будет ли предложена "расширенная поддержка" или есть случаи, когда вывод из эксплуатации будет отложен.

8. Последние два этапа являются наиболее важными для документирования с точки зрения управления жизненным циклом - этапы устаревания и выхода из эксплуатации. Для потребителей API очень важно понимать ожидания, возлагаемые на них, когда они начинают использовать API, чтобы избежать разочарования или проблем при попытке удалить API из каталога. Документирование должно включать, например, управление основными и второстепенными версиями и любые сроки уведомления об изменениях. На высоком уровне, как правило, существует два подхода к устареванию/отмене API: либо сохранение ранее указанного количества версий, либо сохранение старых версий в течение определенного периода времени. Можно также использовать комбинацию этих подходов, но количество поддерживаемых старых версий или продолжительность сохранения старых версий должны быть четко указаны в опубликованном документе о жизненном цикле.

[Конец Приложения VII и Стандарта].