Предоставление защищенных данных. Веб-сервис, проверив переданный токен доступа, возвращает в виде ответа на HTTP-запрос затребованный защищенный ресурс в случае, если переданный токен доступа корректен и не просрочен. В противном случае веб-сервис возвратит ответ с соответствующим кодом и сообщением об ошибке.


Рис. 1. Схема потока аутентификации OAuth 2

Таким образом, протокол аутентификации OAuth 2 предоставляет относительно высокий уровень безопасности. Он не требует от пользователя веб-сервиса доверять свои данные авторизации стороннему приложению, не требует хранения пароля в обратимой форме на сервере веб-сервиса. Этот протокол предоставляет возможность осуществлять тонкое разделение разрешений доступа посредством границ доступа (пользователь может предоставлять сторонним приложениям не полный доступ к своим данным в веб-приложении, а доступ лишь к требуемой части данных). Протокол обеспечивает явное согласие пользователя на предоставления доступа к своим данным сторонним приложениям. Также OAuth 2 не требует передачи в открытом виде пароля пользователя в ходе аутентификации своих запросов, т.к. для этих целей используется токен доступа, привязанный к конкретному стороннему приложению.

ЛИТЕРАТУРА

1. Berners-Lee, T. RFC 1945 – Hypertext Transfer Protocol – HTTP/1.0 [Electronic Resource] / Tim Berners-Lee, Roy T. Fielding, Henrik Frystyk Nielsen. – Mode of access: http://tools.ietf.org/pdf/rfc1945.pdf. – Date of access: 30.09.2015.
2. Fielding, R. Architectural Styles and the Design of Network-based Software Architectures / Roy Thomas Fielding. – University of California – Irvine, 2000. – 162 p.
3. Hardt, D. RFC 6749 – The OAuth 2.0 Authorization Framework [Electronic Resource] / Dick Hardt. – Mode of access: http://tools.ietf.org/pdf/rfc6749.pdf. – Date of access: 30.09.2015.

**УДК 004.9**

## ИСПОЛЬЗОВАНИЕ RESTFUL API ПРИ РАЗРАБОТКЕ ВЕБ-СЕРВИСОВ

***Д.А. САВЧЕНКО***
***(Представлено: канд. физ.-мат. наук, доц. О.В. ГОЛУБЕВА)***

*Рассмотрены основные принципы архитектурного стиля REST. Обращено внимание на основные преимущества, достигаемые при проектировании распределенных систем с учетом формальных ограничений REST.*

Развитие веб-сайтов в настоящее время получило широчайшее распространение; они используются с целью предоставления информации, программных услуг, в качестве средства программного доступа к распределенным приложениям. Развитие веб-сайтов построено на базе протокола HTTP.

Протокол гипертекстовой передачи данных (англ. Hypertext Transfer Protocol – HTTP) является протоколом прикладного уровня, обеспечивающим легкость и скорость, требуемую для распределенных взаимодействующих информационных систем [1]. При разработке веб-сайтов посредством этого протокола к веб-браузеру пользователя поставляются документы различных форматов, формирующие пользовательский интерфейс сайта: это обычно документы разметки в формате HTTP, документы описания стилей в формате CSS, документы исполняемых веб-браузером динамических сценариев в формате JS, а также изображения и шрифты.

Помимо предоставления пользовательского интерфейса от веб-сайтов зачастую требуется предоставление программного интерфейса (API) для сторонних приложений. Такой программный интерфейс преимущественно предназначен для приложений, разработанных для мобильных устройств, но может использоваться и сторонними веб-сайтами. Функциональные возможности программного интерфейса в целом должны повторять таковые пользовательского интерфейса. Распространённым и эффективным стилем реализации программного интерфейса, предоставляемого веб-ресурсами, является RESTful API [2] – стиль построения протокола поверх стандартного HTTP с передачей данных в общеизвестном структурированном формате, обычно JSON или XML.

REST определяет стиль построения архитектур программных продуктов для всемирной сети. Он предоставляет набор ограничений, которым должна соответствовать структура компонентов распределенной системы для того, чтобы достигать наиболее производительной и поддерживаемой архитектурой. Системы, соответствующие принципам REST, называются RESTful-системами. Они обычно взаимодействуют поверх протокола HTTP с помощью стандартных команд протокола (GET, POST, PUT, DELETE и пр.). Соответствующие программные интерфейсы обычно оперируют коллекциями ресурсов, которые идентифицируются посредством URL.

Стиль REST определяет ряд формальных ограничений.

*Клиент-серверная архитектура*. Подразумевается четкое разделение клиента и сервера, взаимодействующих посредством унифицированного интерфейса. Таким способом достигается разделение обязанностей: клиент не ответственен за хранение данных и управление ими, а сервер не ответственен за пользовательский интерфейс и его состояние. Таким образом, клиент и сервер могут в широкой степени разрабатываться раздельно, они также могут быть легко заменены на аналогичные при условии сохранения интерфейса взаимодействия.

*Отсутствие состояния*. Взаимодействие клиента и сервера ограничивается отсутствием сохранения на сервере какого-либо клиентского контекста между запросами. Каждый запрос от клиента к серверу должен содержать всю необходимую информацию для выполнения этого запроса. Состояние сессии при этом хранится на стороне клиента.

*Возможность кэширования*. Клиенты должны иметь возможность кэшировать ответы. По этой причине ответы должны быть явным или неявным образом определены как разрешённые для кэширования, либо не разрешённые, для предотвращения использования клиентом устаревших или некорректных данных. Корректно реализованное кэширование способно частично или полностью исключить часть клиент-серверного взаимодействия, тем самым улучшая масштабируемость и производительность.

*Многослойность системы*. Клиент не должен иметь возможность определить, взаимодействует ли он напрямую с конечным сервером, либо взаимодействие происходит через промежуточные серверы. Промежуточные серверы могут улучшать масштабируемость системы, выполняя балансировку нагрузки либо выполняя кэширование разделяемых данных. Они также могут обеспечивать выполнение дополнительных политик безопасности.

*Динамический код по требованию (необязательное ограничение)*. Сервер должен иметь возможность временно расширить функциональность клиента путем передачи в ходе взаимодействия исполняемого кода. Примером такого кода служат сценарии JavaScript либо скомпилированные Java-апплеты. Это ограничение является единственным необязательным в REST-архитектуре.

*Унифицированный интерфейс взаимодействия*. Это ограничение является фундаментальным для построения любой REST-системы. Унифицированный интерфейс упрощает и разделяет архитектуру, что позволяет развивать каждую ее часть независимо. Существует четыре ограничения при построении унифицированного интерфейса.

а) *Идентификация ресурса*. Отдельные ресурсы должны быть идентифицированы в запросах, например посредством URI в случае основанных на всемирной сети системах. Сами по себе ресурсы должны быть отделены от представления, возвращаемого клиенту, например, данные могут пересылаться

в виде HTML, XML, JSON, причем ни одно из этих представлений может не соответствовать внутреннему представлению данных на сервере.

б) *Управление ресурсами через их представление*. Наличия у клиента представления ресурса, в том числе метаданных, должно быть достаточно для возможности клиента изменить либо удалить ресурс.

в) *Самоописываемые сообщения*. Каждое сообщения должно содержать достаточно сведений для описания способа обработки этого сообщения.

В случае, когда стиль REST применяется к проектированию программных интерфейсов веб-ресурсов, указанные ограничения кратко описываются следующими аспектами:

- наличием базового URI, описывающим глобальную точку доступа к программному интерфейсу;
- наличием определённого типа данных в качестве представления передаваемой в сообщениях информации; наиболее распространёнными являются форматы JSON и XML;
- использование стандартных методов протокола HTTP;
- использование гиперссылок для указания на состояния;
- использование гиперссылок для взаимной связи ресурсов.

Наиболее распространенными структурированными форматами передачи данных при построении RESTful-систем являются JSON [3] и XML [4].

JSON – простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Он основан на подмножестве языка программирования JavaScript, определенного в стандарте ECMA-262 [5]. JSON – текстовый формат, полностью независимый от языка реализации, но он использует соглашения, знакомые программистам C-подобных языков, таких как C, C++, C#, Java, JavaScript, Perl, Python и многих других. Эти свойства делают JSON идеальным языком обмена данными. В качестве значений в JSON используются структуры:

- Объект – это неупорядоченное множество пар ключ-значение, заключённое в фигурные скобки. Ключ описывается строкой, между ним и значением стоит символ двоеточия. Пары ключ-значение отделяются друг от друга запятыми.
- Массив – это упорядоченное множество значений. Массив заключается в квадратные скобки. Значения разделяются запятыми.
- Значение может быть строкой в двойных кавычках, числом, объектом, массивом, одним из литералов: true, false или null. Таким образом структуры могут быть вложены друг в друга.
- Строка – это упорядоченное множество из нуля или более символов юникода, заключенное в двойные кавычки.

XML – расширяемый язык разметки. Разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программам и одновременно удобный для чтения и создания документов человеком, с подчеркиванием нацеленности на использование в Интернете. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка. Сочетание простого формального синтаксиса, удобства для человека, расширяемости, а также базирование на кодировках Юникод для представления содержания документов привело к широкому использованию как собственно XML, так и множества производных специализированных языков на базе XML в самых разнообразных программных средствах.

Таким образом, стиль REST позволяет задать широкий спектр требований к архитектуре распределенных систем, позволяющих в дальнейшем улучшить масштабируемость и возможность поддержки и развития этих систем.

ЛИТЕРАТУРА

1. RFC 1945 – Hypertext Transfer Protocol – HTTP/1.0 [Electronic Resource] / Tim Berners-Lee, Roy T. Fielding, Henrik Frystyk Nielsen. – Mode of access: http://tools.ietf.org/pdf/rfc1945.pdf. – Date of access: 30.09.2015.
2. Fielding, Roy. Architectural Styles and the Design of Network-based Software Architectures / Roy Thomas Fielding; University of California. – Irvine, 2000. – 162 p.
3. RFC 7159 – The JavaScript Object Notation (JSON) Data Interchange Format [Electronic Resource] / Tim Bray. Mode of access: https://tools.ietf.org/pdf/rfc7159.pdf. Date of access: 30.09.2015.
4. Extensible Markup Language (XML) 1.0 [Electronic Resource] / World Wide Web Consortium. – Mode of access: http://www.w3.org/TR/REC-xml/. – Date of access: 30.09.2015.
5. ECMA-262. ECMAScript 2015 Language Specification [Electronic Resource] / Ecma International. – Mode of access: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf. – Date of access: 30.09.2015.