ПРИНЦИПЫ ПРОЕКТИРОВАНИЯ REST API

Гайлитис В.С., студент,

Брянский государственный инженерно-технологический университет, г. Брянск, Россия

Аннотация. В статье дан обзор технологии REST API — это основа всех взаимодействий между различными приложениями. Сегодня многие компании и организации предлагают API для взаимодействия со своими приложениями. REST API позволяют сторонним разработчикам приложений выполнять такие операции, как передача данных или доступ к ним от одного приложения к другому.

Ключевые слова: разработка, принципы, информационные технологии, REST API

REST (от representational state transfer) — это тип архитектуры API, который предусматривает несколько стандартов и соглашений, которые должны соблюдаться для облегчения взаимодействия между приложениями. API, в которых соблюдается стандарт REST, называются REST API или RESTful API[1].

Основные принципы работы REST API:

REST API (Representational State Transfer APIs) — это тип архитектуры API, который предусматривает несколько стандартов и соглашений, которые должны соблюдаться для облегчения взаимодействия между приложениями.

REST API разработаны так, чтобы быть без статических данных, масштабируемыми, гибкими и простыми в использовании.

API REST не зависят от платформы и могут использоваться с любым языком программирования.

REST API поддерживают множество форматов данных, таких как JSON, XML и обычный текст.

API REST могут использовать преимущества механизмов HTTP-кэширования, снижая нагрузку на сервер и улучшая время отклика при повторных запросах. [1]

Стандарт REST накладывает шесть архитектурных ограничений, которые должны соблюдаться системой, чтобы она могла считаться RESTful-системой. Строгое следование этим шести ограничениям обеспечивает оптимальную надежность, масштабируемость и расширяемость. Рассматриваются шесть принципов архитектуры

REST.

Разделение клиента и сервера. Обязанности на стороне сервера и клиента разделены таким образом, что каждая сторона может быть реализована независимо от другой. Код на стороне сервера (API) и код на стороне клиента могут быть изменены без ущерба для друг друга, пока оба продолжают взаимодействовать в одном формате. В архитектуре REST разные клиенты отправляют запросы к одним и тем же конечным точкам, выполняют одни и те же действия и получают одни и те же ответы.

Отсутствие необходимости отслеживания состояния сеансов. При взаимодействии между клиентом и сервером состояние сеансов не отслеживается от одного запроса к другому. Состояние сессии включается в каждый запрос, поэтому ни клиенту, ни серверу не нужно знать состояние другого для взаимодействия. Каждый запрос является полным и самодостаточным. Нет необходимости поддерживать непрерывное соединение между клиентом и сервером, что подразумевает большую устойчивость к сбоям. Кроме того, это позволяет REST API отвечать на запросы нескольких клиентов, не перегружая порты сервера. Исключением из этого правила является аутентификация, так что клиенту не нужно указывать свою аутентификационную информацию при каждом запросе.

Единообразие интерфейса. Различные доступные действия и/или ресурсы с их специфическими конечными точками и параметрами должны быть определены и соблюдаться религиозно, единообразно клиентом и сервером. Каждый ответ должен содержать достаточно информации для интерпретации, чтобы клиент не нуждался в какой-либо другой информации. Ответы не должны быть слишком длинными и должны

содержать ссылки на другие конечные точки.

Кэширование. Чтобы не перегружать сервер без необходимости, ответы можно кэшировать. Кэширование должно быть хорошо управляемым: В REST API должно быть указано, можно ли кэшировать ответ и на какой срок, чтобы клиент не получал устаревшую информацию.

Многослойная архитектура. Клиент, подключенный к REST API, обычно не может отличить, общается ли он с конечным или промежуточным сервером. Архитектура REST позволяет API, например, получать запросы на сервере A, хранить свои данные на сервере B, а управлять аутентификацией на сервере C.

Код по требованию. Это ограничение является необязательным. Оно означает, что API может возвращать исполняемый код вместо ответа, например, в формате JSON или XML. Это означает, что RESTful API может расширять код клиента и одновременно упрощать его, предоставляя исполняемый код, такой как сценарий JavaScript или апплет Java[2].

Использование REST API (Representational State Transfer APIs) дает множество преимуществ при разработке современного программного обеспечения и веб-сервисов:

Простота и удобство использования: REST API придерживаются прямолинейного архитектурного стиля, используя стандартные методы HTTP (GET, POST, PUT, DELETE), что делает их простыми для понимания, реализации и использования.

Независимость от платформы: REST API являются платформонезависимыми, что означает, что они могут использоваться с любым языком программирования и могут быть доступны с различных клиентских устройств, включая веб-браузеры, мобильные устройства и IoT-устройства.

Масштабируемость: API REST разработаны как статические, что обеспечивает горизонтальную масштабируемость, когда несколько экземпляров API могут работать параллельно, чтобы справиться с возросшей нагрузкой.

Гибкость: REST API поддерживают множество форматов данных, таких как JSON, XML и обычный текст, что позволяет клиентам выбирать формат, который соответствует их потребностям.

Кэширование: API REST могут использовать преимущества механизмов кэширования HTTP, снижая нагрузку на сервер и улучшая время отклика при повторных запросах.

Разделение забот: Дизайн REST API предполагает четкое разделение между клиентом и сервером, что повышает модульность и ремонтопригодность системы.

Безопасность: REST API могут использовать стандартные механизмы безопасности, такие как OAuth для аутентификации и SSL/TLS для шифрования, обеспечивая безопасную передачу данных и контроль доступа.

Независимость от языка и технологии: API REST можно использовать с различными языками программирования, фреймворками и технологиями, что обеспечивает взаимодействие и интеграцию различных систем.

Интеграция: REST API облегчают интеграцию различных сервисов и систем, позволяя им беспрепятственно взаимодействовать и обмениваться данными.

Эволюционность: REST API поддерживают версионность, что позволяет разработчикам вносить изменения и обновления в API, не ломая существующие клиентские приложения. [2]

Литература

1. Гаврилюк, В. И. RESTful системы: основные принципы и применение / В. И. Гаврилюк, В. И. Гаврилюк. — Текст: непосредственный // Молодой ученый. — 2020. — № 13 (303). — С. 4-6. — URL: https://moluch.ru/archive/303/68397/ (дата обращения: 25.03.2024).

2. Ивонин, О. А. Создание клиент-серверного приложения на основе restful api архитектуры / О. А. Ивонин, А. Г. Гречихин, И. Р. Гильматдинов. — Текст: непосредственный // Молодой ученый. — 2021. — № 29 (371). — С. 18-20. — URL: https://moluch.ru/archive/371/83265/ (дата обращения: 25.03.2024).