УДК 004.415.2.043
doi:10.18720/SPBPU/2/id23-473

Резединова Евгения Юрьевна [1],
старший преподаватель;
Кыркунов Павел Николаевич [2],
программист;
Сергеев Анатолий Васильевич [3],
доцент, канд. техн. наук

ВЫБОР СЕРВИС-ОРИЕНТИРОВАННОЙ АРХИТЕКТУРЫ ДЛЯ СОЗДАНИЯ СЕРВИСА ПО БЛАГОУСТРОЙСТВУ ГОРОДА

[1, 2, 3] Россия, Санкт-Петербург, Санкт-Петербургский политехнический университет Петра Великого,
[1] Rezedinova_eyu@spbstu.ru, [2] kyrkunov.pn@edu.spbstu.ru,
[3] Sergeev_av@spbstu.ru

Аннотация. Сервис-ориентированная архитектура позволяет создавать веб-приложение, с которым одновременно работает большое количество пользователей. Существует несколько методов реализации сервис-ориентированной архитектуры. В статье рассматриваются разные подходы при реализации сервис-ориентированной архитектуры, сравниваются их достоинства и недостатки. Выбирается метод реализации сервис-ориентированной архитектуры, который в наибольшей степени подходит для создания сервиса по благоустройству города.

Ключевые слова: сервис-ориентированная архитектура, REST, SOAP, gRPC, ACID, благоустройство города.

Evgenia Yu. Rezedinova [1],
Senior Lecturer;
Pavel N. Kyrkunov [2],
Programmer;
Anatoliy V. Sergeev [3],
Associate Professor, Candidate of Technical Sciences

CHOICE OF SERVICE-ORIENTED ARCHITECTURE TO CREATE A CITY IMPROVEMENT SERVICE

[1, 2, 3] Peter the Great St. Petersburg Polytechnic University, St. Petersburg, Russia,
[1] Rezedinova_eyu@spbstu.ru, [2] kyrkunov.pn@edu.spbstu.ru,
[3] Sergeev_av@spbstu.ru

Abstract. Service-oriented architecture allows you to create a web application with which a large number of users work simultaneously. There are several methods of service-oriented architecture implementation. This article discusses different approaches to service-oriented architecture implementation, comparing their advantages and disadvantages. The method of service-oriented architecture implementation is selected, which is most suitable for creating a service for city improvement.

Keywords: service-oriented architecture, REST, SOAP, gRPC, ACID, city improvement.

Введение

Благоустройство является важной и актуальной задачей как для больших, так и для небольших городов. Благоустройство позволяет решать проблемы плохого состояния городской инфраструктуры: повреждённые заборы и детские площадки, ямы на дорогах, газоны ненадлежащего качества отсутствие освещения на улицах и в парках и т. д. Для решения данных проблем требуется не только активное участие муниципальных органов власти, но и удобные механизмы взаимодействия между населением и государственными служащими.

Для решения задачи взаимодействия населения городов с муниципальными органами власти по вопросам благоустройства городов используются специальные веб-сервисы. Известен проект общероссийского народного фонда «Дорожная инспекция ОНФ / Карта убитых дорог» [1]. Сервис предназначен для того, чтобы граждане сообщали информацию об участках дорог с плохим покрытием. Проект существует на всероссийском уровне достаточно давно, используется в Москве, Санкт-Петербурге и других городах. Этот проект вполне удобен, имеет простой и понятный интерфейс. К недостаткам данного проекта относится отсутствие обратной связи с пользователями. Серьёзным недостатком данного проекта является то, что нет возможности сообщить о других проблемах благоустройства города.

Недостатком рассмотренных проектов является невозможность сообщить о других проблемах благоустройства города, не связанных с качеством дорог. Таким образом, население не может в полной мере участвовать в решении задачи благоустройства. Необходимо создать веб-сервис, позволяющий принимать информацию о состоянии городской инфраструктуры, а также различные предложения по благоустройству города. Рассматриваемый сервис должен быть адаптивен, т. е. должна существовать возможность достаточно простого добавления новых функций. Кроме этого, сервис должен легко масштабироваться, чтобы быть доступным для использования в крупных и мелких городах.

За реализацию вышеперечисленных требований, отвечает серверная часть сервиса, поэтому в дальнейшем рассмотрим архитектуру серверной части веб-сервиса.

1. Сервис-ориентированная архитектура

Рассмотрим сервис-ориентированную архитектуру для определения возможности её использования в серверной части. Позволит ли использование сервис-ориентированной архитектуры выполнить требование по адаптивности и масштабируемости приложения?

Сервис-ориентированная архитектура — это модульный подход к разработке программного обеспечения, основанный на использовании распределённых, слабо связанных заменяемых компонентов, взаимодей-

ствующих между собой по стандартизированным транспортным протоколам, используя стандартизированные интерфейсы [2–5]. Данная архитектура облегчает работу программных компонентов в различных сетях друг с другом. Компоненты, связанные сервис-ориентированной архитектурой, обычно называются веб-службами или веб-сервисами. Сервис-ориентированная архитектура базируется на следующих ключевых принципах [6–11]:

– Стандартизированный контракт. Каждый сервис должен иметь какое-то описание, описывающее его суть. При этом каждый сервис обязан соответствовать своему описанию. Это позволяет клиентским приложениям понимать, за что ответственен сервис и, тем самым, облегчает взаимодействие с сервисом.

– Слабая связанность. Одна из самых важных характеристик веб-сервисов, указывающая, что должно быть как можно меньше зависимости между веб-сервисами и клиентом, вызывающим веб-сервис. Если функциональность сервиса изменяется, то это не должно нарушать работу клиентского приложения или останавливать его.

– Абстракция. Веб-сервисы скрывают логику своей работы от внешнего мира. Сервис не должен раскрывать, как он выполняет свою функциональность; он должен просто рассказать клиентскому приложению о том, что он делает, а не о том, как он это делает.

– Повторное использование сервиса. Сервис создаётся как можно более универсальным, чтоб его можно было использовать в других приложениях или в разных частях одного приложения.

– Отсутствие состояния. Данные хранятся отдельно от сервиса, и сервис всегда имеет одно и то же состояние. Таким образом, ответ сервиса зависит только от входных данных.

– Декомпозиция. Один сервис не должен предоставлять весь функционал приложения. Следует разбивать сервис на модули, каждый из которых выполняет свою функцию.

– Совместимость. Сервисы должны использовать стандарты, позволяющие различным клиентам использовать сервис. Например, в веб-сервисах используются такие форматы передачи данных, как XML или JSON и коммуникация с помощью протокола HTTP.

Сервис-ориентированная архитектура позволяет запускать несколько экземпляров одного сервиса. Такой подход позволяет в случае необходимости быстро масштабировать приложение. Отсутствие состояния у сервиса приводит к тому, что пользователь не привязан к одному экземпляру сервиса, а может подключаться к любому свободному экземпляру сервиса. В результате осуществляется гибкое распределение нагрузки на все экземпляры сервиса.

Слабая связанность сервисов позволяет достаточно просто модифицировать сервисы или добавлять новые сервисы, реализующие новый функционал. В соответствии с принципом декомпозиции сервисы должны реализовывать одну функцию или часть функции. Благодаря такому подходу можно относительно просто по сравнению с монолитным приложением изменять функционал, добавляя или удаляя соответствующие сервисы.

Таким образом, исходя из вышеизложенного, можно сделать вывод, что сервис ориентированная архитектура позволяет создать приложение, которое легко масштабировать в зависимости от нагрузки и также относительно просто модифицировать.

Сервис-ориентированную архитектуру можно реализовать разными способами, рассмотрим их и вберем наиболее подходящий для создания сервиса по благоустройству города.

2. SOAP

SOAP — это протокол обмена данными по сети, основанный на формате XML. В настоящее время он в основном используется для проектирования веб-сервисов и передачи данных по HTTP/HTTPs, но не ограничивается только ими. Стандарт SOAP определяет структуру сообщения, правила кодирования и формат запросов и ответов. В результате SOAP даёт возможность передавать данные по сети независимо от платформы или языка программирования.

Стандарт SOAP обеспечивает высокую надёжность передачи сообщения благодаря тому, что удовлетворяет требованиям ACID. ACID расшифровывается как «atomicity, consistency, isolation, durability» (атомарность, согласованность, изолированность, надёжность). ACID является фактическим стандартом для высоконадёжных систем.

Соответствие ACID означает, что транзакции отвечают следующим требованиям:

– Атомарность. Несколько связанных транзакций либо работают как единое целое, либо не работают вообще. Иногда это называется подходом «все или ничего». Этот набор транзакций сравнивается с атомом, который состоит из нескольких тесно связанных элементов.

– Согласованность. Если какая-либо часть транзакции выходит из строя, то система откатывается обратно в исходное состояние.

– Изолированность. Транзакции изолированы и независимы друг от друга.

– Надёжность. Даже если вся система даёт сбой, завершенные транзакции сохраняются.

Для достижения ACID-совместимости, необходимо использовать стандартизированный протокол WS-Atomic Transaction.

SOAP позволяет использовать только XML-формат для обмена данными. Данные в формате XML имеют большой размер и создают большую нагрузку на каналы передачи данных. Обработка XML данных тоже требует значительных вычислений. [12, 13] Таким образом, недостаткам SOAP является то, что этот протокол требует больше вычислительных ресурсов по сравнению, например, с REST. [14–17].

Несмотря на указанные недостатки, SOAP широко используется, особенно в тех случаях, когда существуют повышенные требования к безопасности и устойчивости системы.

3. REST

REST — это архитектурный стиль взаимодействия компонентов распределённого приложения в сети по протоколу HTTP. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой системы. Принципы такого подхода были сформулированы в 2000 году ученым Роем Филдингом.

Ключевыми элементами REST API являются:

– клиент или программное обеспечение, которое работает на компьютере или смартфоне пользователя и инициирует общение;

– сервер, предлагающий API в качестве средства доступа к своим данным или функциям;

– ресурс, который сервер предоставляет клиенту. Ресурсом является, например, текстовый файл, картинка, видео.

Чтобы получить доступ к ресурсу, клиент посылает HTTP-запрос. В ответ сервер генерирует HTTP-ответ с закодированными данными о ресурсе. Оба типа REST-сообщений содержат информацию о том, как их интерпретировать и обрабатывать.

Свойства архитектуры, которые зависят от ограничений, наложенных на REST-системы:

– простота унифицированного интерфейса;

– открытость компонентов к возможным изменениям для удовлетворения изменяющихся потребностей (даже при работающем приложении);

– прозрачность связей между компонентами системы для сервисных служб;

– переносимость компонентов системы путем перемещения программного кода вместе с данными;

– надёжность, выражающаяся в устойчивости к отказам на уровне системы при наличии отказов отдельных компонентов, соединений или данных.

REST не связан с какой-либо определенной технологией или платформой. Он не задаёт правила построения API. REST задаёт ограничения. Работая в рамках этих ограничений, система приобретает желаемые

свойства, такие как производительность, масштабируемость, простота, способность к изменениям, переносимость, надёжность.

Рассмотрим данные ограничения подробнее:

– Клиент-серверная архитектура. При использовании клиент-серверной архитектуры алгоритм работы сервера никак не связан с алгоритмом работы клиента.

– Многоуровневая архитектура. Система, построенная на принципах REST, имеет многоуровневую структуру, в которой каждый уровень работает независимо и взаимодействует только с соседними уровнями.

– Кэширование. REST позволяет клиентам хранить часто используемые данные на своей стороне вместо того, чтобы запрашивать их снова и снова. В результате приложение совершает меньше вызовов, что снижает нагрузку на сервер и задержку ответа. В свою очередь, приложение становится более отзывчивым и надежным.

– Отсутствие сохранения состояния (stateless). Сервер не хранит никакой информации, относящейся к предыдущим сессиям, обрабатывая каждый запрос самостоятельно. Все данные о текущем состоянии клиента содержатся в теле запроса. Таким образом, отпадает потребность синхронизации состояния клиента с сервером. Кроме этого любой сервер из группы может обрабатывать запросы вне зависимости, какой сервер обрабатывал предыдущие запросы. Это улучшает производительность приложения и снижает риск сбоя.

– Код по запросу (Code on demand). Сервер по запросу клиента отправляет исполняемый код клиенту. Такой подход позволяет переложить часть вычислительной нагрузки на клиента. Кроме этого уменьшается трафик между клиентом и сервером, так как вместо большого объёма данных передаётся небольшая программа, которая на стороне клиента создаёт необходимые данные. С другой стороны, использование этой технологии уменьшает безопасность, так как исполняемый код может оказаться вредоносным. Поэтому это ограничение является необязательным.

– Унифицированный интерфейс. Унифицированный интерфейс определяет стандартизованный способ взаимодействия с заданным сервером, независимо от того, какое клиентское приложение или устройство его запускает.

Рассмотрим требования к интерфейсу:

1) Каждому ресурсу присваивается уникальный идентификатор (URI).

2) Сообщения клиента и сервера содержат информацию о том, как их интерпретировать и обрабатывать.

3) Существует возможность использовать данные, хранящиеся в формате JSON или XML, а также использовать гиперссылки на различные мультимедиа ресурсы.

Такой подход облегчает разработчикам понимание логики API и, следовательно, упрощает разработку приложения.

4. gRPC

gRPC — это система удалённого вызова процедур с открытым исходным кодом, разработанная компанией Google на основе RPC. RPC предлагает классический стиль API, разработанный достаточно давно и используемый и в настоящее время. gRPC осуществляет вызов процедур для запроса данных с удаленного сервера. Чаще всего используется для подключения служб в микросервисной архитектуре и для привязки клиентов к серверным службам [18–22].

Рассмотрим особенности технологии gRPC. При обмене данными используется протокол сериализации структурированных данных Protocol Buffers. Этот протокол передаёт данные не в текстовом формате как XML, а в двоичном. Двоичный формат значительно сокращает объём передаваемых данных. В текстовом файле с расширением .proto определяется схема структурирования данных. Используя прото-компилятор, этот файл затем автоматически компилируется в любой из многочисленных поддерживаемых языков, таких как Java, C++, Python, Go, Dart, Objective-C, Ruby и другие.

Для передачи данных gRPC использует протокол HTTP/2. Рассмотрим основные преимущества HTTP/2:

– Бинарное кадрирование. HTTP/2 позволяет передавать данные в двоичном формате.

– Параллельность запросов. В то время как HTTP/1.1 позволяет обрабатывать только один запрос за раз, HTTP/2 поддерживает несколько вызовов по одному каналу. Более того, связь является двунаправленной — одно соединение может отправлять как запросы, так и ответы одновременно.

Недостатком протокола HTTP/2 является его недостаточная распространённость. По статистике в настоящее время только около 50 % сайтов поддерживают данный протокол. Это одна из причин, по которой gRPC в основном используется в серверной части приложений.

Использование gRPC не всегда является обоснованным по сравнению с другими архитектурами, например, REST. Для того, чтобы применение gRPC было максимально эффективно, необходимо, чтоб соблюдались следующие условия:

– наличие веб-сервисов, связанных между собой и общающихся в реальном времени;

– необходимость использования потоковых запросов;

– наличие веб-сервисов, которые написаны на разных языках программирования и обмениваются данными между собой.

Таким образом, получается, что технологию gRPC целесообразно применять при использовании микросервисной архитектуры.

Микросервисная архитектура — вариант сервис-ориентированной архитектуры программного обеспечения, направленный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей — микросервисов.

gRPC целесообразно использовать для связи между внутренними микросервисами, так как gRPC имеет высокую скорость передачи данных и позволяет передавать несколько сообщений одновременно. Кроме этого gRPC позволяет обмениваться данными микросервисам, написанными на разных языках программирования. Именно по этой причине Netflix, Lyft, WePay и другие компании, работающие с микросервисами, перешли на gRPC.

С другой стороны, для микросервисов с внешним интерфейсом рекомендуется использовать REST, так как текстовые сообщения, используемые в данном протоколе, ориентированы на пользователя человека.

Особенности gRPC:

– Использует транспортный протокол HTTP/2. HTTP/2 в отличие от HTTP является бинарным, имеет изменённые способы разбиения данных на фрагменты и передачи данных между компонентами (в том числе потоковая передача данных).
– Используется бинарный формат protobuf в качестве контракта обмена данными. Данный формат уменьшает размер сообщения и увеличивает скорость передачи данных, но при этом требует кодирование на стороне отправителя и декодирование на стороне получателя.

5. Выводы

Основываясь на вышеприведённом анализе различных механизмов организации сервис-ориентированной архитектуры для реализации сервиса по благоустройству города, можно сделать следующие выводы.

1. Архитектура REST поддерживает несколько форматов данных, в том числе JSON, который является более компактным и быстрым по сравнению с XML и более читаемым, чем protobuf.

2. REST использует протокол HTTP. Таким образом, все наработки на базе HTTP, такие как кэширование на серверном уровне и масштабирование, работают в REST архитектуре. SOAP тоже использует HTTP как транспортный протокол и обеспечивает достаточно высокую безопасность. С другой стороны использование SOAP требует больше вычислительных ресурсов. Разрабатываемому сервису по благоустройству города не требуется очень высокий уровень безопасности, т. к. он не будет обрабатывать секретные данные. Сервис предназначен для одновре-

менного обслуживания большого количества пользователей, поэтому целесообразно использовать протоколы, не требующие значительных вычислительных ресурсов. Такому требованию удовлетворяет REST.

3. REST и gRPC являются очень простыми в изучении и применении, в отличие от SOAP.

4. gRPC, в силу своих особенностей, является наиболее эффективным подходом при проектировании микросервисной архитектуры и при наличии сервисов, написанных на разных языках. Применительно к разрабатываемому приложению это достоинство не является существенным, так как для реализации серверной части предполагается использовать только один язык программирования.

Исходя из вышеприведённых соображений, наиболее подходящая технология для создания сервиса по благоустройству города — это технология REST.

Заключение

В работе были рассмотрены существующие веб-сервисы по благоустройству города. Недостатком существующих сервисов является то, что они ориентированы только на повышение качества дорог и не рассматривают другие проблемы благоустройства города.

Было показано, что сервис-ориентированный подход при создании серверной части приложения позволяет удовлетворить требования по масштабируемости и адаптивности приложения.

В серверной части системы было предложено использовать сервис-ориентированную архитектуру. Были рассмотрены следующие реализации сервис-ориентированной архитектуры: SOAP, REST, gRPC. Наиболее подходящим для использования в сервисе по благоустройству города признан метод REST. В дальнейшем серверная часть сервиса по благоустройству города будет реализована с использованием этого метода.

Список литературы

1. Проект общественной организации «Убитые дороги». Дорожная инспекция / карта убитых дорог». [Электронный ресурс] // URL: https://dorogi-onf.ru/city/4940/ (дата обращения 06.11.22).

2. Граничин О.Н., Шеронов И.Л. Сервис-ориентированная архитектура ИС ВШМ СПбГУ и проблемы стохастической оптимизации // Информационные системы. – 2007. – Т. 3. No 1–1. – С. 138–152.

3. Кашалкин Д. Ю., Курчидис В. А. Принципы построения семантической сервис-ориентированной архитектуры // Модел. и анализ информ. систем. – 2007. – Т. 14, номер 1. – С. 48–53.

4. Кожомбаева А.Т., Щетилов А.В., Зотин А.Г. Анализ технологий взаимодействия мобильных приложений с веб-сервисами // Актуальные проблемы авиации и космонавтики. – 2015. – № 11.

5. Исикдаг Умит. Шаблоны проектирования для сервисно-ориентированных архитектур на основе BIM-технологий // Вестник ТГАСУ. – 2013. – № 2 (39).

6. Дубинин В.Н., Дубинин А.В., Ручкин М.А. Программные модели мехатронных систем на основе сервис-ориентированной архитектуры // Модели, системы, сети в экономике, технике, природе и обществе. – 2020. – № 4.

7. Жуковский О.И., Ощепков С.С., Рыбалов Н.Б. Применение стилей сервис-ориентированной архитектуры для разработки геоинформационной системы // Информатика, телекоммуникации и управление. – 2009. – № 1 (72).

8. Людвиченко А.А. Сервис-ориентированный подход к архитектуре интернет-магазина // Вопросы науки и образования. – 2017. – №7 (8).

9. Сатунина А.Е., Сысоев А.С. Подход к проектированию безопасности в сервис-ориентированных архитектурах // История и архивы. – 2010. – № 12 (55).

10. Кузин М.В. Информационные технологии в образовании: сервис-ориентированная архитектура // Вестник Марийского государственного университета. – 2009. – № 3.

11. Гридин В.Н., Дмитревич Г.Д., Анисимов Д.А. Архитектура распределенных сервис-ориентированных систем автоматизированного проектирования // Известия ЮФУ. – Технические науки. – 2014. – № 7 (156).

12. General availability: SOAP and XML request and response validation. [Электронный ресурс]. – URL: https://azure.microsoft.com/ru-ru/updates/general-availability-soap-and-xml-request-and-response-validation/ (дата обращения 06.11.22).

13. Kohlhoff Christopher, Steele Robert. Использование SOAP в высокопроизводительных бизнес-приложениях: торговые системы реального времени [Электронный ресурс]. – URL: http://www.k-press.ru/cs/2004/4/soaptest/soaptest.asp (дата обращения 06.11.22).

14. Halili Festim, Ramadani Erenis. Web Services: A Comparison of Soap and Rest Services // Modern Applied Science. – 2018. – Vol. 12. 175. – DOI: 10.5539/mas.v12n3p175.

15. Демичев А. П., Крюков А. П., Шамардин Л. В. Принципы построения грид с использованием RESTful-веб-сервисов // Программные продукты и системы. – 2009. – № 4.

16. Черныш Б.А., Картамышев А.С. Разработка ядра интегрированной информационной системы // Программные продукты и системы. – 2021. – № 2.

17. Танатканова А.К. Применение микросервисной архитектуры при разработке корпоративных веб-приложений // Вестник науки. – 2019. – № 5 (14).

18. gRPC Technology White Paper. [Электронный ресурс]. – URL: https://downloadcdn.h3c.com/en/202007/24/20200724_5098534_99-book_1316570_294551_0.pdf (дата обращения 06.11.22).

19. Донцов А.А., Суторихин И.А. Разработка геоинформационной системы на базе микросервисной архитектуры // Интерэкспо Гео-Сибирь. – 2021. – № 1.

20. Пацей Н.В., Шитько А.М. Интеграция микросервисов на основе RPC // Эпоха науки. – 2021. – № 27.

21. Гагарин В.Ю., Вагнер А.В., Тропченко А.А. Особенности миграции технических проектов с микросервисной архитектурой и использованием GRPC в качестве протокола межсервисного взаимодействия с платформы .NET CORE 3 на платформу .NET 6 // МНИЖ. – 2022. – №4-1 (118).

22 Чекан М.А., Широков В.В. Реализация сетевой архитектуры для распределённой системы проведения аукционов на виртуальные ресурсы // The Scientific Heritage. – 2021. – № 67-1. – URL: https://cyberleninka.ru/article/n/realizatsiya-setevoy-arhitektury-dlya-raspredelyonnoy-sistemy-provedeniya-auktsionov-na-virtualnye-resursy (дата обращения: 15.11.2022).