ISSN 2658-5014 (Print) СИИТ ISSN 2686-7044 (Online)
2024. Т. 6, № 2 (17). С. 78–85 http://siit.ugatu.su
СИСТЕМНАЯ ИНЖЕНЕРИЯ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ
Научные сообщения
УДК 004.028 DOI 10.54708/2658-5014-SIIT-2024-no2-p78 EDN KCYBPB

Сравнение технологий API для интеграции с корпоративными системами: REST, SOAP, WebSocket, GraphQL

И. Р. Тукаев • В. А. Беспоясова

Аннотация. С развитием компьютерных технологий выросла потребность в интеграции корпоративных приложений и систем, наиболее удобным и простым в разработке является интеграция по API. В свою очередь, данный подход имеет большое количество реализаций, наиболее популярными способами взаимодействия систем являются REST, SOAP, GraphQL и WebSocket. В данной статье представлен краткий обзор каждого из вышеперечисленных способов, выделены основные преимущества и недостатки. Сделан вывод о том, что каждый из подходов имеет свои сильные и слабые стороны и подходит для решения конкретных задач.

Ключевые слова: API; REST; SOAP; WebSocket; GraphQL; веб-сервер; интеграция.

Введение

В ходе развития информационных технологий увеличивались количество и производительность электронных вычислительных машин, в результате чего многие задачи стали автоматизироваться и перекладываться на ЭВМ. Одной из возникших потребностей стало связывание компьютеров в единую сеть, решение данной задачи вело к необходимости совершенствования программного обеспечения, которое должно было обеспечивать одновременный доступ большого числа пользователей с различных устройств. Для этого требовалось решить ряд проблем и вопросов:

- необходимость работы приложения в кроссплатформенном режиме, то есть программный продукт должен быть доступен на любом устройстве с любой операционной системой;
- необходимость разделения приложения на клиентскую и серверную части для доступа к программному продукту более чем с одного устройства;
- необходимость создания универсальных подходов к созданию сетевых приложений, не зависящих от операционной системы, языка программирования реализации серверной части;
- возможность расширяемости и масштабируемости приложений.

На сегодняшний день существуют ряд подходов к взаимодействию нескольких информационных систем и приложений [1], один из наиболее популярных и удобных – интеграция через API (Application Program Interface). Однако, в свою очередь, данный подход разбивается на несколько методов. В настоящее время самыми распространёнными являются: REST, GraphQL, SOAP и WebSocket [2–13]. Поэтому при разработке приложения, требующего взаимодействия по API, возникает вопрос: какой из методов лучше использовать в конкретном случае? Целью данной работы являются краткий сравнительный анализ и обзор наиболее популярных подходов к разработке API веб-приложений.

Подход на основе REST

REST (англ. REpresentational State Transfer – Передача состояния представления) – архитектурный стиль взаимодействия компонентов распределенной системы. Важной особенностью REST является то, что это не единственный и конкретный протокол взаимодействия, а набор рекомендаций того, как разработчику следует реализовывать «общение» разрабатываемого приложения с другими. Существуют 6 принципов REST [3]:

1. Клиент-серверная модель (client-server model): клиент – программа, запрашивающая у сервера данные, сервер – программа, в которой хранятся и обрабатываются данные.

2. Отсутствие состояния (statelessness): сервер не хранит данных о предыдущих запросах и не связывает их, то есть каждый запрос содержит всю информацию о предыдущих.

3. Кэширование (cacheability): сохранение некоторой части часто запрашиваемых данных на стороне клиента или промежуточных серверах для снижения нагрузки на основной сервер.

4. Единообразие интерфейса (uniform interface): единый способ обращения к серверу. Например, формат возвращаемых данных не должен отличаться в старых и новых методах доступа.

5. Многоуровневая система (layered system): между клиентским приложением и сервером должно быть несколько промежуточных узлов для вспомогательных функций – прокси-серверы, которые используются для кэширования или предварительной обработки данных, обеспечения безопасности.

6. Код по требованию (code on demand): возможность передачи исходного кода, который будет выполняться на стороне клиента. Например, JavaScript-код для динамических элементов интерфейса пользователя [3].

Взаимодействие клиентских приложений с сервером, где хранятся данные, в REST API обычно, но не обязательно, происходит по протоколу HTTP(S) с использованием таких методов, как:

• POST – метод, который предназначен для обращения к веб-серверу, который принимает данные, заключенные в тело запроса;

• GET – метод, который предназначен для получения информации с сервера, передавая данные запроса в URL-строке;

• PUT – метод, который предназначен для редактирования данных, которые уже присутствуют на сервере;

• DELETE – метод, который предназначен для удаления данных с сервера.

Отличительной чертой REST является то, что каждый запрос осуществляется по отдельному URL и обрабатывается отдельным методом или функцией. Например, у сервера с адресом /url/ есть некоторые данные о нефтяных скважинах и о подрядчиках, выполняющих работы на скважинах. Если клиентскому приложению необходимо получить информацию о скважине с идентификатором 17, то необходимо отправить GET-запрос по адресу

/url/wells/?well_id=17.

В случае, если клиентскому приложению необходима информация о конкретном подрядчике, то строка для GET-запроса будет отличаться:

/url/contractors/?contractor_id=17.

Другой особенностью взаимодействия приложений по REST API является большой набор доступных форматов: HTML, XML, JSON и любой другой удобный формат.

Таким образом, структура REST запроса следующая:

• конечная точка – URL-строка, к которой отправляется запрос;

• параметры запроса – переменные, передаваемые в URL для получения конкретных данных;

• заголовки – служебная информация о запросе, включающая формат передаваемых данных, спецификацию, версию протокола обмена и так далее;

• тело запроса – данные, которые необходимо обработать, например, в формате XML [4].

Структура ответа содержит в себе следующую информацию:

• код ответа – признак успешности выполнения запроса, представляет собой число, состоящее из трех цифр. Например, ответы вида 1xx – информационные, 2xx – успешная операция (200 – OK), 3xx – необходимость уточнения или перенаправление запроса, 4xx – ошибка на стороне клиента (400 – запрос некорректный, 404 – ресурс не найден), 5xx – ошибка на стороне сервера (504 – таймаут);

• заголовок ответа – служебная информация, аналогичная заголовку запроса;

• тело ответа – данные, передаваемые клиентскому приложению, например, в формате XML [1].

Преимущества REST подхода:

1. Увеличение производительности: RESTful подход позволяет эффективно использовать ресурсы сервера за счет использования кэширования, минимизации передачи данных и распределения нагрузки.
2. Простота в использовании: RESTful API использует простые и понятные HTTP методы для взаимодействия с сервером.
3. Масштабируемость: RESTful подход позволяет легко масштабировать приложения за счет возможности добавления новых ресурсов и функционала без изменения основной структуры.
4. Гибкость: RESTful API позволяет работать с различными типами данных и форматами.
5. Независимость от платформы.

Недостатки REST подхода:

1. Отсутствие спецификации.
2. Неоднозначность методов управления данными.

Подход на основе SOAP

Другим популярным, но менее распространенным форматом взаимодействия приложений является SOAP (англ. Simple Object Access Protocol – протокол доступа к объектам) – протокол обмена структурированными сообщениями [5]. Он может использоваться с любым протоколом прикладного уровня, но наиболее часто встречается реализация с использованием HTTP.

Особенностью данного способа интеграции приложений является использование определенного формата данных – все сообщения передаются в формате XML. С помощью языка XML Schema происходят автоматическая валидация и строгая типизация данных, определяется формат обмена сообщениями.

Другой отличительной чертой данного способа взаимодействия приложений является наличие одной и единственной точки входа на сервере. Пакеты, передаваемые между сервером и клиентским приложением, – SOAP-конверты – содержат в себе информацию о методе, который требуется вызвать, или данных, которые требуется получить. Такой подход называется RPC (Remote Procedure Call) – вызов удаленных процедур [6]. Каждое XML-сообщение включает в себя:

• Envelop (конверт) – корневой элемент, определяющий XML-документ как сообщение SOAP при помощи использования пространств имен;

• Header (заголовок) – элемент, содержащий в себе атрибуты сообщения, метаинформацию. Существуют три атрибута, указывающие на то, как серверу следует обрабатывать сообщение: mustUnderstand – атрибут, который говорит о том, является ли заголовок обязательным к распознаванию; actor – конкретный адрес сообщения; encodingStyle – кодировка сообщения;

• Body (тело) – элемент, содержащий в себе само сообщения – запрос к серверу или ответ от него;

• Fault – опциональный (необязательный) элемент, содержащий в себе информацию об ошибках, которые произошли при обработке сообщений [7].

Пример отправляемого XML-сообщения для запроса данных с сервера о нефтяной скважине с идентификатором 17 (рис. 1):

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd
    ="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org
    /soap/envelope/">
   <soap:Body>
     <getWellInfo xmlns="url">
       <wellID>17</wellID>
     </getWellInfo>
   </soap:Body>
</soap:Envelope>
```

Рис. 1 Пример SOAP-запроса.

Пример ответа сервера по нефтяной скважине (рис. 2):

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <getWellInfoResponse
            xmlns="url">
            <getWellInfoResult>
                <wellID>17</wellID>
                <wellName>433</wellName>
                <affiliateID>1</affiliateID>
                <contractorID>2</contractorID>
                <clusterID>3</clusterID>
                <isActive>true</isActive>
            </getWellInfoResult>
        </getWellInfoResponse>
    </soap:Body>
</soap:Envelope>
```

Рис. 2 Пример SOAP-ответа.

Преимущества:
- отраслевой стандарт, по версии W3C;
- наличие строгой спецификации;
- широкая поддержка в продуктах Microsoft,
- однозначность.

Недостатки:
- сложность реализации;
- сложность/ресурсоемкость парсинга XML-данных.

Подход на основе GraphQL

В настоящее время набирает популярность более простой и лаконичный способ взаимодействия приложений – GraphQL [8]. GraphQL – язык запросов для данных API, разработанный в 2012 г. и имеющий открытый исходный код. Данный подход имеет подробную спецификацию и стандартизирован. Для обмена данными в основном используется HTTP, однако поддержка других транспортных протоколов присутствует – gRPC, веб-сокеты и другие.

Язык создавался в качестве альтернативы REST для уменьшения нагрузки на сеть и упрощения управления конечными точками REST API. Данный подход подразумевает единственную точку входа на сервере и позволяет получить данные в формате JSON в необходимом

количестве и с любой вложенностью. Таким образом, сервер передаёт имеющиеся данные в том формате, который удобен клиентскому приложению, без необходимости создания и поддержания множества различных точек входа и методов их обработки.

Для работы с данными GraphQL имеет 3 основных типа запроса:

• Query (запрос) – тип запроса, который является аналогом GET в REST и служит для получения необходимых данных с сервера;

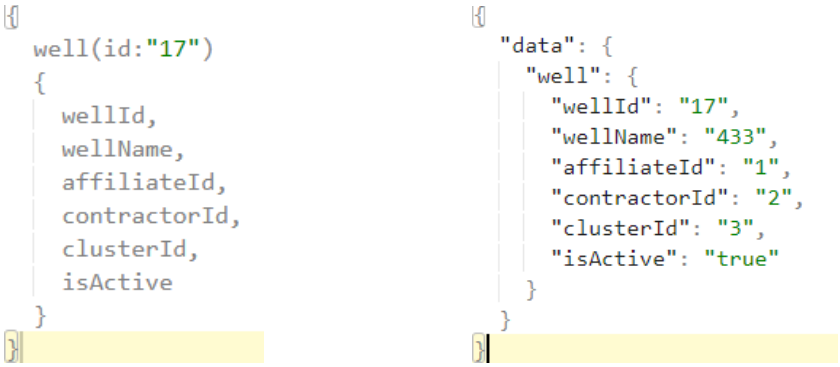
• Mutation (мутации) – тип запроса, который является аналогом POST и PUT в REST и служит для отправки данных на сервер для их изменения или добавления в базу;

• Subscription (подписка) – тип запроса, который подразумевает оповещения клиентского приложения об изменении данных в БД в режиме реального времени.

Использование GraphQL на сервере подразумевает развертку схемы (Scheme), в которой описываются все объекты со строго типизированными полями, с которыми работает сервер, и распознаватели (Resolvers) – функции, с помощью которых можно получить доступ к данным для определенного поля или объекта. Также эти функции позволяют установить ограничения безопасности [1].

Следует отметить, что для работы с GraphQL требуется запуск отдельного GraphQL-сервера, принимающего соответствующие запросы.

Например, для получения данных о нефтяной скважине с идентификатором 17 требуется выполнить следующий запрос (рис. 3, *а*). Пример ответа в формате JSON со следующим содержимым приведен на рис. 3, *б*.



а *б*

Рис. 3 GraphQL-запрос (*а*) и GraphQL-ответ (*б*).

Преимущества:

1. Работает не только поверх протокола HTTP, но и поверх Websockets, SSH и многих других.
2. Вместо нескольких точек входа (endpoints) обращается к одной, через которую проходят все запросы – */graphql*.
3. Строго типизирован – в схеме ответа описываются тип каждого поля и его обязательность в запросе.
4. Предоставляет встроенную площадку для тестирования и отладки запросов – graphiql.
5. Обходится меньшим количеством действий для описания ресурсов – достаточно описать схему зависимостей.
6. Предоставляет возможность передать аргументы в запрос на любом уровне вложенности

Недостатки:

1. При реализации запроса по возврату всех полей необходимо прописывать все поля. *users {name surname age phone address}.*

2. Инструментарий. Стандартная библиотека *graphql-%название_языка%* не предоставляет возможности для работы с файлами *.gql* и *.shema* – все это нужно писать через примитивные структуры языка.

Подход на основе WebSocket

Менее популярным, но всё еще актуальным подходом к взаимодействию приложений является протокол двусторонней связи для непрерывного обмена сообщениями между клиентским приложением и сервером – WebSocket (веб-сокет) [6].

Особенностью данного подхода является то, что вместо периодического соединения в формате «запрос – ответ» поддерживается постоянное соединение с сервером, ввиду чего сервер всегда знает конкретного клиента и может отправить ему информацию без дополнительного запроса.

Данный подход не подразумевает специфического формата обмена данными: WebSocket использует свой собственный бинарный формат, что позволяет отправлять данные любого типа (текстовые сообщения, изображения, видео- и аудиофайлы и так далее) в удобной для клиентского приложения форме. Передача зашифрованных данных производится путём использования надстройки над протоколом WSS [5].

В общем виде механизм WebSocket работает следующим образом:

1. Приложение или веб-страница создает скрипт с функциями обратного вызова: две из них сообщают об установке и закрытии соединения, третья – о получении новых данных от сервера.
2. Клиентское приложение отправляет запрос на подключение на сервер по протоколу TCP.
3. Сервер принимает запрос и отвечает.
4. Клиентское приложение оставляет соединений открытым, что говорит о создании канала. Далее при каждом новом сообщении срабатывает обратный вызов функции из п. 1.

Согласно источнику [5], к преимуществам WebSocket относятся:

1. Двусторонняя связь в режиме реального времени.
2. Низкая задержка и быстрое соединение.
3. Снижение накладных расходов и меньшее количество обращений туда и обратно благодаря одному долгоживущему соединению.
4. Поддержка потоковой передачи высококачественного мультимедиа без задержек.

К недостаткам можно отнести:

1. Поддерживается не всеми браузерами и прокси-серверами.
2. Сложнее масштабировать и управлять по сравнению с традиционным HTTP.
3. Потенциальные сложности при реализации функций безопасности.
4. Отвал соединения без уведомлений. Чтобы понять, отвечает ли клиент, иногда нужно вводить дополнительные механизмы общения между ним и сервером.
5. Смена сети клиентом. Если при переподключении к другой сети клиент не закрыл соединение, сервер не получит информации об изменении адреса.

Сравнение подходов

На основании изложенного, наиболее подходящий протокол во многом будет зависеть от конкретных требований и функций приложения:

1. WebSocket уникален своей способностью обеспечивать двустороннюю связь в реальном времени и наиболее предпочтительным для следующих типов приложений: приложения реального времени, онлайн-игры, финансовые торговые платформы, совместное редактирование, службы потокового вещания в реальном времени.
2. GraphQL рекомендуется использовать в следующих случаях: при использовании архитектуры с большим количеством микросервисов или сущностей с большим количеством полей, которые не всегда нужны, или с большим количеством связей и вложенностей.

3. SOAP API нашел широкое применение в интеграции между системами, особенно в крупных предприятиях, где требуется обмен данными между приложениями и платформами. Также он используется в веб-сервисах для предоставления API для внешних разработчиков.

4. REST API хорошо масштабируется и подходит для построения приложений с микросервисной архитектурой, при разработке которой важно разделение больших частей на маленькие. Кроме того, REST применяется для обращения сервисов к технологиям облачных вычислений.

Заключение

Таким образом, в данной работе был осуществлен обзор четырех наиболее популярных способов взаимодействия клиент-серверных приложений, а также приведены их преимущества и недостатки. Следует отметить, что не существует единой формулы выбора подхода, и каждый случай зависит от решаемой задачи.

Например, для решения задачи, связанной с аналитикой данных автоматизированного тестирования, лучшим вариантом является применение REST API. Это связано с тем, что система состоит из множества микросервисов, каждый из которых выполняет определенную задачу, в будущем она будет масштабироваться, также там применяется кеширование, а отправляемые запросы просты и не требуют дополнительной настройки.

Другим примером решения задачи взаимодействия корпоративных приложений является интеграция информационных систем, которые служат для оптимизации процессов при текущем и капитальном ремонтах скважин и которые предназначены для автоматизации всего цикла проведения внутрискважинных работ. Наиболее подходящим вариантом для обмена данными по операциям и процессам на скважинах между системами является REST API ввиду наличия инкапсуляции данных, повышающей безопасность информации, а также возможности передачи API сторонним системам без дополнительных настроек, финансовых и временных вложений для развертки специального программного и аппаратного обеспечения.

Для создания более гибких запросов к серверу при решении задач, в которых время отклика не является ключевым, хорошим решением станет GraphQL. Для создания чатов мгновенных сообщений или онлайн игр требуется непрерывное подключение клиента к серверу и в данном случае наилучшим решением станет использование WebSocket.

Благодарности

Авторы выражают признательность научному руководителю проф. В. В. Антонову, другим сотрудникам кафедры АСУ за помощь в понимании некоторых аспектов рассматриваемых вопросов [14–18].

СПИСОК ЛИТЕРАТУРЫ / REFERENCES

1. Шор А. М. Сравнительный анализ подходов в разработке api веб-приложений // StudNet. 2020. Вып. 9. С. 533–540. [[Shor A. M. “Comparative analysis of approaches in the development of API web applications” // StudNet. 2020. Issue 9, pp. 533-540. (In Russian).]]
2. Думченков И. А. Обзор методов интеграции информационных систем, их преимуществ и недостатков // Молодой ученый. 2018. № 23 (209). С. 176–177. URL: https://moluch.ru/archive/209/51296/ (дата обращения: 09.04.2024). [[Dumchenkov I. A. “Review of methods for integrating information systems, their advantages and disadvantages” // Young Scientist. 2018. No. 23 (209), pp. 176-177. (In Russian).]]
3. Аникин Д. А. Анализ методов авторизации и аутентификации REST API // Международный журнал информационных технологий и энергоэффективности. 2023. Вып. 5–2. С. 120–124. [[Anikin D. A. “Analysis of authorization and authentication methods of REST API” // International Journal of Information Technologies and Energy Efficiency. 2023. Issue 5-2, pp. 120-124. (In Russian).]]
4. Yandex Cloud. REST API: для чего нужен и как работает. 2024. [Электронный ресурс]. URL: https://yandex.cloud/ru/docs /glossary/rest-api? (дата обращения: 23.03.2024). [[REST API: what is it for and how does it work. 2024. (In Russian).]]
5. Hirsch, Frederick; Kemp, John; Ilkka, Jani (2007-01-11). Mobile Web Services: Architecture and Implementation. John Wiley & Sons (published 2007). ISBN 9780470032596.
6. Ian Fette; Alexey Melnikov (December 2011). "Opening Handshake". RFC 6455 The WebSocket Protocol. IETF. sec. 1.3. doi: 10.17487/RFC6455. RFC 6455.

7. Макджмартин Дж., Сезарини М., Джеини А. Проектирование веб-служб с использованием SOAP и WS-*. М.: ДМК Пресс, 2017. [[McGmartin J., Cesarini M., Geini A. Designing Web Services Using SOAP and WS-*. Moscow: DMK Press, 2017. (In Russian).]]
8. Karlsson, Stefan; Causevic, Adnan; Sundmark, Daniel (May 2021). "Automatic property-based testing of GraphQL APIs" // 2021 IEEE/ACM International Conference on Automation of Software Test (AST). Madrid, Spain: IEEE, pp. 1–10. arXiv:2012.07380. doi:10.1109/AST52587.2021.00009. ISBN 978-1-6654-3567-3. S2CID 229156477.
9. Wilde E., Pautasso C. REST: From Research to Practice. Springer Science & Business Media, 2011. ISBN 978-1-4419-8303-9.
10. Roy Fielding. Architectural Styles and the Design of Network-based Software Architectures. University of California, IRVINE. 2000.
11. Арно Л. Проектирование веб-API. М.: ДМК Пресс, 2020. [[Arno L. Web API Design. Moscow: DMK Press, 2020. (In Russian).]]
12. Порселло Е., Бэнкс А. GraphQL: язык запросов для современных веб-приложений. Питер, 2019. [[Porcello E., Banks A. GraphQL: A Query Language for Modern Web Applications. Peter, 2019. (In Russian).]]
13. Хабаров С. П., Шилкина М. Л. Построение распределенных систем на базе WebSocket. Лань, 2021. [[Khabarov S. P., Shilkina M. L. Construction of Distributed Systems Based on WebSocket. Lan, 2021. (In Russian).]]
14. Родионова Л. Е., Антонов В. В., Баймурзина Л. И., Гидинда Г. М. Модели проектирования программных аналитических комплексов с декартово замкнутой категорией // СИИТ. 2023. Т. 5. № 5(14). С. 3–15. EDN AQLGLE. [[Rodionova L. E., Antonov V. V., Baymurzina L. I., Gidinda G. M. “Design models for software analytical complexes with a Cartesian closed category” // SIIT. 2023. Vol. 5, No. 5(14), pp. 3-15. EDN AQLGLE. (In Russian).]]
15. Антонов В. В., Харисова З. И., Байболдина А. А. Статистический анализ метрик цветовосприятия при зрительной нагрузке цифровыми устройствами // СИИТ. 2024. Т. 6. № 1(16). С. 23–30. EDN JRZEJK. [[Antonov V. V., Kharisova Z. I., Bayboldina A. A. “Statistical analysis of color perception metrics under visual load with digital devices” // SIIT. 2024. Vol. 6, No. 1(16), pp. 23-30. EDN JRZEJK. (In Russian).]]
16. Миронов В. В., Гусаренко А. С., Юсупова Н. И. Ситуационно-ориентированные базы данных: polyglot persistence на основе REST-микросервисов // Прикладная информатика. 2019. Т. 14. № 5(83). С. 86–97. DOI 10.24411/1993-8314-2019-10038. EDN MCKTVY. [[Mironov V. V., Gusarenko A. S., Yusupova N. I. “Situation-oriented databases: polyglot persistence based on REST microservices” // Applied Informatics. 2019. Vol. 14, No. 5(83), pp. 86-97. EDN MCKTVY. (In Russian).]]
17. Миронов В. В., Гусаренко А. С., Юсупова Н. И. Применение веб-сервисов на основе ситуационно-ориентированной базы данных для мониторинга просмотра учебного видеоконтента // Моделирование, оптимизация и информационные технологии. 2019. Т. 7. № 3(26). С. 27. EDN RVUOOD. [[Mironov V. V., Gusarenko A. S., Yusupova N. I. “Application of web services based on a situation-oriented database for monitoring the viewing of educational video content” // Modeling, Optimization and Information Technologies. 2019. Vol. 7, No. 3(26), pp. 27. EDN RVUOOD. (In Russian).]]
18. Гусаренко А. С., Миронов В. В. Использование RESTful-сервисов в ситуационно-ориентированных базах данных // Вестник УГАТУ. 2015. Т. 19. № 1(67). С. 232–239. EDN TPNUOX. [[Gusarenko A. S., Mironov V. V. “Using RESTful services in situation-oriented databases” // Vestnik UGATU. 2015. Vol. 19, No. 1(67), pp. 232-239. EDN TPNUOX. (In Russian).]]

Поступила в редакцию 26 марта 2024 г.

МЕТАДАННЫЕ / METADATA

Title: Comparison of API technologies: REST, SOAP, WebSocket, GraphQL - The choice for integration with corporate systems.

Abstract: With the development of computer technology, the need for integration of corporate applications and systems has grown, and API integration is the most convenient and easy to develop. In turn, this approach has a large number of implementations, the most popular ways of interacting systems are REST, SOAP, GraphQL and WebSocket. This article provides a brief overview of each of the above methods, highlights the main advantages and disadvantages. It is concluded that each of the approaches has its own strengths and weaknesses and is suitable for solving specific tasks.

Key words: API; REST; SOAP; WebSocket; GraphQL; web server; integration.

Язык статьи / Language: русский / Russian.

Об авторах / About the authors:

ТУКАЕВ Ильдар Рашидович
ФГБОУ ВО «Уфимский университет науки и технологий», Россия.
Магистрант каф. автоматизированных систем управления.
E-mail: tukaev_ildar@mail.ru

БЕСПОЯСОВА Виктория Альфридовна
ФГБОУ ВО «Уфимский университет науки и технологий» , Россия.
Магистрант каф. автоматизированных систем управления.
E-mail: v.markutsaa@mail.ru

TUKAEV Ildar Rashidovich
Ufa University of Science and Technologies, Russia.
Undergraduate student of the Automated Control Systems Dept.
E-mail: tukaev_ildar@mail.ru

BESPOYASOVA Viktoria Alfridovna
Ufa University of Science and Technologies, Russia. Student of the Undergraduate student of the Automated Control Systems Dept.
E-mail: v.markutsaa@mail.ru