Столыпинский вестник

Научная статья

Original article

УДК 004

СПОСОБЫ СВЯЗИ ДВУХ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ

WAYS TO LINK TWO APPLICATIONS

Копылова Яна Антоновна, Студент, МИРЭА-Российский технологический университет (РТУ МИРЭА)

Матвеев Владимир Евгеньевич, Студент, 3 курс, направление «Прикладная информатика», МИРЭА-Российский технологический университет (РТУ МИРЭА), 119454, Россия, г. Москва, проспект Вернадского, 78, Институт информационных технологий, Россия, г. Москва

Kopylova Yana Antonovna, Student, MIREA-Russian Technological University (RTU MIREA)

Matveev Vladimir Evgenievich, Student, 3 course, direction "Applied Informatics", MIREA-Russian Technological University (RTU MIREA), 119454, Russia, Moscow, Vernadsky Avenue, 78, Institute of Information Technology, Russia, Moscow

Аннотация

В наше время технологии развиваются с огромной скоростью. С течением времени потребности бизнеса и обычных пользователей относительно функциональности приложений увеличиваются. Написание программ, которые могли бы выполнять абсолютно все желаемые функции,

может расходовать огромное количество ресурсов, что в результате снижает выгодность такой разработки. Конечно же, данная проблема не могла остаться незамеченной, что привело к появлению интеграции программных приложений. Такой подход был бы выгодным с точки зрения бизнеса, так как пользователи взаимодействуют исключительно с одним программным решением компании-автора, которая получает прибыль в полном объеме.

Abstract

Nowadays, technology is developing at a tremendous speed. Over time, the needs of businesses and ordinary users regarding the functionality of applications increase. Writing programs that can perform absolutely all the desired functions can consume a huge amount of resources, which as a result reduces the profitability of such development. Of course, this problem could not go unnoticed, leading to the emergence of software application integration. This approach would be advantageous from a business point of view, since users interact exclusively with one software solution of the authoring company, which profits in full.

Ключевые слова: интеграция, связь, API, приложения, функционал, ИТ, решения.

Key words: integration, communication, API, applications, functionality, IT, solutions.

На данный момент пара каких-либо приложений может быть связана друг с другом двумя способами: прямое взаимодействие одного приложения с базой данных другого приложения или использование специального программного интерфейса – API (Application Programming Interface, программный интерфейс приложения). Первый подход чаще всего считается ненадежным, так как одно приложение должно предоставлять другому полный доступ к своим данным, из-за чего могут возникать ошибки. Также в таком случае будет нарушена безопасность, так как потенциальный злоумышленник, получив доступ к одному компоненту системы, может взломать и все связанные с ним.

По вышеперечисленным причинам, использование API приветствуется намного больше. API – программный интерфейс приложения, представляющий набор функций, команд, доступных для использования другими приложениями. Например, банковское приложение может предоставить доступ к функции снятия денег со счета клиента для приложений различных магазинов. В таком случае программа, которая захочет списать с банковского аккаунта клиента определенную сумму, не будет иметь доступа ко всем данным покупателя, а отправит запрос в API банка, передав необходимые данные, а уже банковское приложение, получающее данный запрос, обработает его, выполнит необходимые проверки (например, удостоверится, что у покупателя достаточно средств для выполнения операции) и вернет ответное сообщение с информацией об успешности данной операции.

Очевидно, что каждый разработчик, создающий приложение с API, мог бы разрабатывать отдельный формат для каждого интерфейса, получающего данные. Через некоторое время таких потенциальных форматов возникло бы колоссальное количество, что усложнило бы процесс написания кода для всех разработчиков, которые хотели бы использовать возможности других приложений. Поэтому, в современном мире получили широкое распространение два основных формата реализации интерфейсов: SOAP и REST [2].

SOAP (Simple Object Access Protocol, простой протокол для доступа к объектам) использует для передачи данных формат XML (расширенный язык разметки), позволяющий строго типизировать все необходимые действия и передаваемые данные. Преимуществами использования данного протокола являются наличие строгой спецификации, широкая поддержка крупными компаниями и однозначность при дешифровке. Тем не менее, данный метод обладает и недостатками. Самыми существенными из них являются сложность реализации и написания запросов, а также сложность и ресурсоемкость при

декодировании полученной информации. Примеры запроса и ответа в формате XML приведены на Рисунке 1 и на Рисунке 2.

```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="https://www.w3.org/2003/05/soap-envelope/"
               soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding/">
    <soap:Body>
      <m:GetPrice xmlns:m="https://www.w3schools.com/prices">
        <m:Item>Apples</m:Item>
      </m:GetPrice>
    </soap:Body>
</soap:Envelope>
```

Рисунок 1 – Пример запроса в формате XML

```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="https://www.w3.org/2003/05/soap-envelope/"
               soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding/">
    <soap:Body>
      <m:GetPriceResponse xmlns:m="https://www.w3schools.com/prices">
        <m:Price>1.90</m:Price>
      </m:GetPriceResponse>
    </soap:Body>
</soap:Envelope>
```

Рисунок 2 – Пример ответа в формате XML

XML сообщение в протоколе SOAP должно обладать следующими элементами: Envelope – корневой обязательный элемент, означающий начало и окончание сообщения; Header – необязательный элемент заголовка, содержащий дополнительную информацию для обработки сообщения; Body – основной обязательный элемент, содержащий основную передаваемую информацию; наконец, необязательный элемент Fault, содержащий информацию об ошибках [3].

Вторым вариантом стандартизации API является подход REST. REST (Representational State Transfer) является не протоколом, а архитектурным стилем, набором рекомендаций при создании программных интерфейсов.

Требования к такой архитектуре включают в себя следующее: модель клиент-сервер; отсутствие состояния системы, то есть сервер должен получать всю необходимую информацию, а данные о сессии хранятся на стороне клиента; кэширование – возможность клиента сохранять некоторые ответы на запросы для избегания повторения операций; единообразие интерфейса, предполагающее то, что все интерфейсы должны быть выполнены в одном стиле, данное качество также предполагает, например, то, что сервер может отправлять информацию в виде XML или JSON (JavaScript Object Notation – текстовый формат для представления объектов, состоящий из пар «ключ: значение»), но это не означает, что база данных хранит записи в одном из этих форматов; наконец, последнее требование – наличие различных слоев, благодаря которым клиент не может определить взаимодействует ли он напрямую с сервером или с каким-либо промежуточным программным обеспечением, что может быть использовано для повышения безопасности и стабильности всего сервиса.

Любое API, реализующее все принципы REST может называться RESTful API. Чаще всего такая архитектура реализуется на основании HTTP запросов, используя методы GET, UPDATE, POST и DELETE. Примеры запроса и ответа при использовании сервиса, реализованного по принципу REST, приведены на Рисунке 3 и на Рисунке 4.

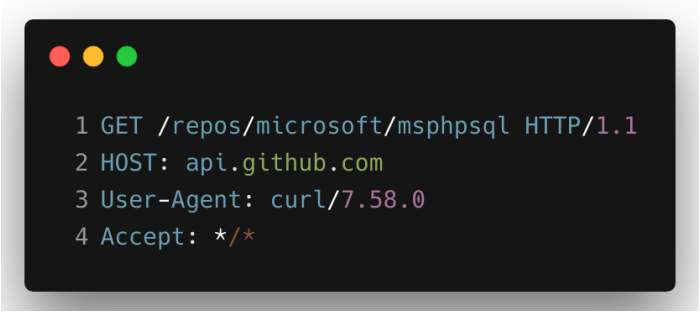

Рисунок 3– Пример HTTP-GET запроса в RESTful API

Рисунок 4 – Пример ответа на запрос, данные передаются в формате JSON

В качестве одного из основных достоинств RESTful API можно выделить их производительность, возникающую за счет отсутствия сложных конвертаций данных. Также весомыми преимуществами данного подхода являются простота его реализации и экономная трата ресурсов.

Тем не менее, данный метод не лишен недостатков. Один из наиболее существенных заключается в том, что у REST отсутствует точная спецификация, что приводит к неоднозначности при управлении данными.

Стоит упомянуть, что два вышеописанных подхода являются реализациями Web API, так как такой вариант связи приложений наиболее распространен на данный момент. В случае с приложениями, обменивающимися информацией не по сети Интернет, интерфейсы реализованы либо на основе SOAP или REST, либо созданы уникальные протоколы в рамках компании, позволяющие осуществлять «общение» между отдельными элементами системы. На момент написания данной работы такой подход устаревает, а компании все чаше прибегают к использованию программных интерфейсов, обменивающихся сообщениями через Интернет.

До сих пор отсутствие связи нескольких приложений в теории считается наилучшем подходом, так как итоговые продукты становятся самодостаточными и не зависят ни от других приложений компании, ни от решений сторонних разработчиков. Тем не менее, если бы в каждом продукте необходимо было бы заново создавать те функции, которые уже реализованы в других проектах, время и стоимость разработки увеличивались бы в

геометрической прогрессии, что непозволительно в современных реалиях, так как одним из ключевых параметров успешности приложения является быстрый выход на рынок. Именно поэтому, на сегодняшний день интеграция программных приложений считается наиболее выгодным решением, так как она позволяет быстро разработать новый функционал, не прибегая к дополнительным затратам ресурсов на повторное создание существующих решений.

Список использованных источников

1. Интеграция программного обеспечения. Описание процесса от бизнес-консультанта. – Текст : электронный // Хабр : [сайт]. – 2014. – URL: https://habr.com/ru/company/trinion/blog/245615/ (дата обращения: 22.09.2022).
2. Интеграция корпоративного приложения с внешними системами. – Текст : электронный // Simpleone : [сайт]. – 2020. – URL:https://simpleone.ru/blog/integracziya-korporativnogo-prilozheniya-s-vneshnimi-sistemami/ (дата обращения: 20.09.2022).
3. Рельсы веб-интеграции. REST и SOAP. – Текст : электронный // Хабр : [сайт]. – 2021. – URL: https://habr.com/ru/post/591573// (дата обращения: 25.09.2022).
4. REST vs SOAP – Текст : электронный // Otus : [сайт]. – 2021. – URL: https://otus.ru/nest/post/2427/ (дата обращения: 10.10.2022).
5. Применение SOAP при интеграции систем – Текст : электронный // systems.education : [сайт]. – 2021. – URL: https://systems.education/soap-integration(дата обращения: 25.10.2022).

List of sources used

1. Software integration. Description of the process from a business consultant. - Text: electronic // Habr: [website]. – 2014. – URL: https://habr.com/ru/company/trinion/blog/245615/ (date of access: 22.09.2022).

2. Integration of a corporate application with external systems. – Text: electronic // Simpleone: [website]. – 2020. – URL: https://simpleone.ru/blog/integracziya-korporativnogo-prilozheniya-s-vneshnimi-sistemami/ (date of access: 20.09.2022).
3. Web Integration Rails. REST and SOAP - Text: electronic // Habr: [website]. - 2021. - URL: https://habr.com/ru/post/591573// (date of access: 09/25/2022).
4. REST vs SOAP - Text: electronic // Otus: [website]. – 2021. – URL: https://otus.ru/nest/post/2427/ (date of access: 10.10.2022).
5. The use of SOAP in systems integration - Text: electronic // systems.education: [website]. - 2021. - URL: https://systems.education/soap-integration (date of access: 10/25/2022).

Для цитирования: Копылова Я.А., Матвеев В.Е. СПОСОБЫ СВЯЗИ ДВУХ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ// Научный сетевой журнал «Столыпинский вестник» №9/2022