Столыпинский
вестник

Научная статья
Original article
УДК 004

МИКРОСЕРВИСНАЯ АРХИТЕКТУРА ПРИ РАЗРАБОТКЕ ФРОНТЕНД ПРИЛОЖЕНИЙ

MICROSERVICE ARCHITECTURE FOR FRONTEND APPLICATION DEVELOPMENT

Яровая Екатерина Владимировна, Магистрант Гродненского государственного университета им Янки Купалы, г. Гродно

Katsiaryna V. Yaravaya, Master's student Yanka Kupala State University of Grodno, Grodno

Аннотация

В статье рассмотрим основные проблемы, возникающие при архитектуре крупных энтерпрайз приложений для клиентской части приложения. Обсудим как эти проблемы можно решить и какие готовые решения можно использовать для фрагментирования манолитных проектов. Обсудим основные принципы построения микросервисной архитектуры. Рассмотрим одно из готовых решений под названием Module Federation, упомянем терминологию, используемую в данном подходе. Рассмотрим компоненты Module Federation, также плюсы и минусы.

Разберём зачем нужен webpack и его применение.

Annotation

In this article we will consider the main problems arising when architecting large-scale enterprise applications for the client side of the application. We will discuss how these problems can be solved and what ready-made solutions can be used for fragmenting manolithic projects. Let's discuss the basic principles of building a microservice architecture. Let's look at one of the ready-made solutions called Module Federation and mention the terminology used in this approach. Let's look at the Module Federation components and the pros and cons.

Let's look at why we need a webpack and how to use it.

Ключевые слова: JavaScript, фронтенд технологии, микросервисная архитектура, module federation.

Keywords: JavaScript, frontend technologies, microservice architecture, module federation.

С 1990 году, когда англичанин Тим Бернерс-Ли создал первый в мире сайт, который назывался info.cern.ch, прошло не один десяток лет и веб страницы сделали огромный скачок в своем развитии, от чего сложность веб проектов усложнилось во много раз. Веб приложение как правило состоит из нескольких компонентов HTML (HyperText Markup Language) верстки, CSS (Cascading Style Sheets) стилей и JavaScript кода, и все эти элементы во взаимодействии друг с другом, которые придают приложению максимальный уровень интерактивности и оптимизированности. Чем сложнее структура приложения, тем сложнее интерфейс, предоставляемый для пользователей. По этой причине фронтенд-разработка из дополнения, как пользовательский интерфейс в сложную экосистему с огромным количеством всевозможных интеграций. С ростом и количеством новой функциональности, логика приложения становится все сложнее и запутаннее, затрудняют разработку и тестирования.

Концепция микросервисной архитектуры была заимствована у разработчиков серверной части, где принято разбивать одно монолитное

приложение на несколько независимых или малозависимых друг от друга сервисов, которые выполняют строго отведенные для них задачи, и это широко распространенный архитектурный паттерн для разработки.

Причины появления микросервисной архитектуры призваны решить все те же проблемы, что и у разработки серверной части приложений. При разработке архитектуры больших приложений с множеством функциональности на клиентской части, все чаще и чаще разбиваются на микросервисы, давая определение, что такое микросервис, мы должны понимать, что это полностью изолированная часть, и никак не зависимая от других таких сервисов, этот сервис должен быть разработан как отдельное приложение, также сама зона ответственности должна отвечать только одной бизнес функции, также может быть написана на любой технологии, но это не самое лучшее решение хотя оно и будет полноценно работать.

Пришло время понять, что такое микросервис, это фрагмент приложения, который состоит из JavaScript кода, CSS и неких правил развертывания. Фрагмент является независимой частью, которая должна следовать определенным правилам, чтобы его можно было использовать в родительском приложении. Прямого воздействия на другие фрагменты быть не должно. А благодаря правилам развертывания, мы можем получить информацию о других зарегистрированных фрагментах и при надобности иметь к ним доступ по уникальному идентификатору. Данный подход позволит динамически подгружать нужный сервис на страницу пользователя. Для лучшего понимания давайте представим страницу, в которой есть часть навигации, и окно отображения контента от разных фрагментов, которые встраиваются и управляются родительским приложением, которое и отвечает за подгрузку того или иного фрагмента.

Одним из основных плюсов такого подхода – мы получаем не монолитное приложение, а фрагментированное. И тут мы можем подчеркнуть ряд плюсов: разделение зоны ответственности, каждая команда выполняет строго

поставленная перед ними задачи, которые будет реализовывать выпуск новой функциональности согласно их календарю релизов. Повышение стабильности работы, так как сервисы не зависят друг от друга, облегчают тестирование и разработку в несколько раз, потому что небольшая команда легко может отслеживать и анализировать каждую добавленную строку кода, а при необходимости, откатить на более старую версию. Если возникает потребность, то приложение с такой архитектурой легко масштабируется.

Такой способ разработки выглядит очень привлекательным, но и в нем есть ряд минусов, которые надо обсудить. Нет возможности взаимодействовать между фрагментами стандартными методами. Размер приложения будет слишком велик, если не придерживаться строгим правилам общих зависимостей. За навигацию также должна отвечать родительская часть.

Теперь немного о технологиях, которые помогли бы нам достичь нашей цели. Что мы могли бы использовать, очень старая технология под названием Server-Side Fragment Composition – основная идея которого заключалась в том, что веб-сервер собирает из разных блоков сервисы в единую html страницу. Iframe – это Transclusion, который работает на стороне клиента и дает возможность вставлять блоки по URL, широко используют в баннерной рекламе, особенно ощутимый минус такого подхода, это высокая нагрузка на устройства пользователей и большой вес приложений, так как по сути это отдельные веб страницы, которым прилагаются все используемые ими библиотеки. Web Components является стандартом 2011 - 2013 для браузеров, предоставляют возможность определять и настраивать динамические элементы с инкапсулированной логикой. Еще один подход — это Linked Pages & SPAs, идея заключалась в том, что при помощи load-балансировки мы можем получать то или иное SPA (single page application) приложение. Следующий подход single-spa достаточно популярный, являются тонкой прослойкой, которая по URL делает запуск того или иного фрагмента.

В качестве более подробного примера, хотелось бы упомянуть об одном из механизмов, которые помогают реализовать микросервисную архитектуру, это решение было предложено и зародилось в голове Зака Джексона в середине 2017 году. Первое публичное обсуждение в github было в декабре 2018, в октябре 2019 году появился первый анонс в виде статьи от Зака, и в октябре 2020 вышел в релиз как core-плагин для Webpack 5 под названием Module Federation. Также активное участие принимали в разработке Мариус Россоу и ассистировал им Тобиас Копперс, который, собственно, и подстраивал архитектуру пятого webpack для успешного внедрения Module Federation. Эта технология позволяет подключать из других webpack сборок, которые располагаются на других хостах, адресах. Основные цели, преследуемые при разработке Module Federation, были не вызывать перезагрузку веб страницы при смене модуля, не загружать так называемый vendor code, который можно использовать из другого Webpack сборки тем самым уменьшить “бандел”, не пересобирать родительское приложение если меняется родительский фрагмент. Управление модулями должно происходить на стороне пользователя.

Основные компоненты архитектуры Module Federation являются: Host (consumers) – это наше родительское приложение, которое первым инициализируется, первым призагружает страницы и является каркасом для всех остальных сервисов. Remote (consumable) – дочерний компонент встраиваемый в Host. Omnidirectional host – это такая часть приложения, которая может быть как host так и remote, например раздавать компоненту кнопку. Exposed modules, модуль, к которому имеют доступ другие компоненты приложения, например картинки или стили. Shared modules может выступать, к примеру – библиотека React.

Из минусов можно отметить возможность vendor lock, проблемы во время разработки, нужно пересматривать подход к деплоям. Есть вероятность появления проблем с асинхронным запуском.

Данный подход к написанию клиентской части приложения весьма перспективен и вероятно всего будет успешно применяться к большим многофункциональным проектам, которые должны включать в себя множество команд. Этот подход существенно облегчит разработку.

Литература

1. Крис Ричардсон, Микросервисы. Паттерны разработки и рефакторинга (2019)
2. Парминдер Сингх Кочер, Микросервисы и контейнеры Docker (2022)
3. Сэм Ньюман, Building Microservices: Designing Fine-Grained Systems(2014)
4. Zack Jackson, Practical Module Federation, (2021)
5. Д. Флэнаган JavaScript. Полное руководство | Флэнаган Дэвид 7-е издание, (2021)

Literature

1. Chris Richardson, Microservices. Patterns of Development and Refactoring (2019)
2. Parminder Singh Kocher, Docker Microservices and Containers (2022)
3. Sam Newman, Building Microservices: Designing Fine-Grained Systems (2014)
4. Zack Jackson, Practical Module Federation, (2021)
5. The Complete Guide | Flanagan David 7th Edition, (2021)

Для цитирования: *Яровая Е.В.* МИКРОСЕРВИСНАЯ АРХИТЕКТУРА ПРИ РАЗРАБОТКЕ ФРОНТЕНД ПРИЛОЖЕНИЙ// Научный сетевой журнал «Столыпинский вестник» №5/2022.