Столыпинский вестник

Научная статья

Original article

УДК 004

ИССЛЕДОВАНИЕ МИКРОФРОНТЕНДНОЙ АРХИТЕКТУРЫ – ИНСТРУМЕНТЫ И РЕКОМЕНДОВАННЫЕ ПРАКТИКИ ИСПОЛЬЗОВАНИЯ

STUDY ON MICROFRONT ARCHITECTURE - TOOLS AND RECOMMENDED USE PRACTICES

Берьянов Максим Сергеевич, Магистрант, 2 курс, Факультет ПИиКТ, Университет ИТМО Россия, г. Санкт-Петербург

Монченко Артем Сергеевич, Магистрант, 2 курс, Факультет ПИиКТ, Университет ИТМО, Россия, г. Санкт-Петербург

Дерябин Андрей, Магистрант, 2 курс, Факультет ПИиКТ, Университет ИТМО Россия, г. Санкт-Петербург

Buryanov Maxim Sergeevich, berjnov.ru@mail.ru

Monchenko Artem Sergeevich, berjnov.ru@mail.ru

Deryabin Andrey, berjnov.ru@mail.ru

Аннотация. С развитием микросервисов большие приложения получили несколько преимуществ – данный подход помогает эффективно разрабатывать, развертывать и масштабировать отдельные части серверной части приложения. Тем не менее, многие осознают, что аналогичные проблемы существуют и для внешнего интерфейса. Именно здесь мы обычно начинаем

разбивать монолит внешнего интерфейса на микрофронтенды. Идея микрофронтенда базируется на том же принципе, что и микросервисы в бэкэнде. Каждое приложение существует независимо и имеет четко определенную цель. В данной статье предлагается познакомиться с данной концепцией, выделить ключевые особенности, а также подходы к проектированию приложений с такой архитектурой.

Abstract. With the development of microservices, large applications have received several advantages - this approach helps to efficiently develop, deploy and scale individual parts of the application's backend part. However, many people are aware that similar issues exist for the frontend part as well. This is where we usually start to break the frontend monolith into microfrontends. The idea of a microfrontend is based on the same principle as microservices in the backend world. Each application exists independently and has a well-defined purpose. This article proposes to get acquainted with this concept, highlight key features, as well as approaches to designing applications with such architecture.

Ключевые слова: веб-разработка, React.JS, микросервис, микрофронтенд.

Keywords: web development, React.JS, microservice, microfrontend.

Понятие микрофронтендной архитектуры

Термин «микрофронтенды» распространяет концепции микросервисов из мира серверных технологий на мир графических интерфейсов. Ниже на рисунке 1 показана эволюция от монолитной архитектуры до связки микрофронтенды + микросервисы, за исключением того, что будущее на самом деле должно настать уже сейчас.

В течение многих лет происходила тенденция к уделению большого внимания микросервисам, включающая в себя разделение на компоненты серверной части, оставляя внешний интерфейс по-прежнему гигантским

монолитом. Микрофронтенды же были созданы, чтобы декомпозировать монолитный интерфейс, превращая всю систему в разрозненные компоненты.

Ключевые преимущества микрофронтендов:

- лучшая масштабируемость;
- более быстрая разработка, так как команды могут работать независимо – это преимущество применимо только в том случае, если проект является большим и в нем участвуют более одной команды фронтенда;
- независимость CI/CD – сборка и последующее разворачивание нашего микрофронтенд-приложения не повлияет на все приложение в целом – изменения коснутся именно той части бизнес-процесса, функции которой данный микрофронтенд выполняет;
- возможность обновлять части внешнего интерфейса проще, чем это было доступно раньше;

- присутствие большей вероятности, что другая часть приложения останется стабильной при разработке функционала, поскольку она независима, так как независимы микрофронтенды – с разделенными интерфейсами больше не нужно отслеживать все приложение;

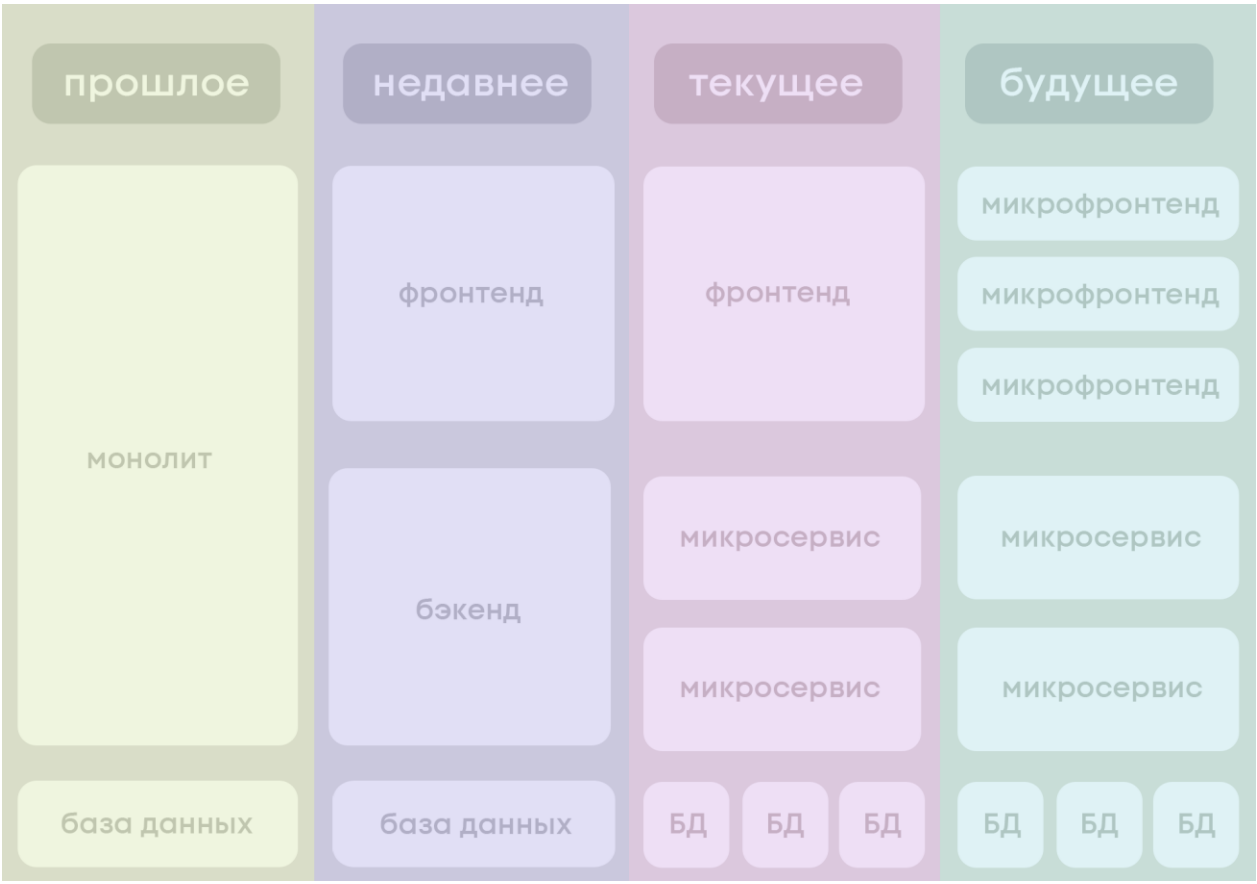

Рисунок 1 – Эволюция архитектуры

- кодовые базы меньше по размеру и более управляемы;
- более простое тестирование, так как происходит тестирование только отдельных функций.

Инструменты разработки

И Angular, и React.js — наиболее распространенные и прекрасные инструменты для фронтенд-разработки. У них есть масса преимуществ, но из приведенного ниже сравнения (таблица 1) и статистики становится ясно, что React.js работает лучше, используется больше, развивается и остается в тренде.

Кроме того, данная библиотека получает огромную поддержку от сообщества разработчиков.

Критерий	**React.js**	**Angular**
Производительность	Виртуальный DOM позволяет нам обновлять изменения без переписывания всего HTML-документа. Это значительно ускоряет рендеринг обновлений и обеспечивает высокую производительность независимо от размера приложения.	Реальный DOM вместо того, чтобы изменять только часть требуемых данных, обновляет всю древовидную структуру HTML-таблиц, пока не достигнет необходимых данных.
Компонентная архитектура	С React мы создаем компоненты, которые управляют своим собственным состоянием, и структурируем их вместе в более сложные пользовательские интерфейсы. React легче понять, но для полного использования его потенциала требуется несколько интеграций, таких как, к примеру, Redux.	Angular — это чистый полноценный фреймворк, который поставляется со многими готовыми функциями, такими как RxJS, angular CLI, Angular Universal.
Размер приложения	React использует webpack (разбивает код на более мелкие части), встряска дерева и динамический импорт, которые уменьшают размер пакета. Встряска дерева (или устранение неиспользуемого кода) означает, что неиспользуемые модули	В процессе сборки Angular избавляется от утилит разработки и неиспользуемых модулей, затем выполняется минификация и сжатие, что позволяет фреймворку выполнять дополнительные оптимизации.

	не будут включены в комплект поставки приложения в процессе сборки. При динамическом импорте приложение загружает код, необходимый изначально, а остальное загружает по запросу.	
Обратная совместимость	React — это библиотека с полной обратной совместимостью – мы можем добавлять в приложение разные версии библиотек и обновлять старые версии.	Невозможно перейти с Angular 2.0 на 7.0 – нужно устанавливать обновления между версиями, одну за другой. Если мы планируем постепенно улучшать проект, добавляя новые функции, лучшим выбором будет React, поскольку он обладает абсолютной обратной совместимостью.
Масштабирование	React сильно зависит от инструментов сторонних разработчиков (библиотек).	Angular поставляется со всеми основными функциями, которые могут понадобиться разработчикам для масштабирования существующего приложения путем добавления новых функций.

Таблица 1 – Сравнение React и Angular

React.js выгоднее использовать, чем Angular, благодаря реализации виртуальной DOM и оптимизации рендеринга. Миграция между версиями React.js довольно проста – не нужно устанавливать обновления по одному последовательно, как в случае с Angular. Наконец, вместе с React.js у разработчиков появляется множество уже существующих решений, которые они могут использовать – это ускоряет время разработки и сокращает

количество ошибок. Стоит отметить, что, если из-за ограниченности ресурсов разработчиков на React.js в команде совершенно нормально использовать Angular или смешивать рассматриваемые библиотеки, разработка приложений-полиглотов (спроектированных на разных языках) является одним из преимуществ микрофронтендной архитектуры.

Управление кодовой базой

Монорепозиторий — это репозиторий, в котором хранятся несколько проектов, а мультирепозиторий — это несколько репозиториев, в которых также хранятся несколько проектов, причем в каждом репозитории хранится свой собственный.

Монорепозиторий может решить некоторые распространенные проблемы микрофронтендной архитектуры:

- совместное использование общей функциональности – совместное использование библиотек между несколькими репозиториями затруднено. В монорепозитории создать общую библиотеку так же просто, как создать новый файл и импортировать его в другие проекты;
- несоответствие между микроинтерфейсами – работа над микроинтерфейсами в отдельных репозиториях может привести к рассогласованности между проектами. Наличие микрофронтендов в одном репозитории помогает поддерживать согласованность, особенно когда мы можем применить автоматизацию, например один и тот же инструмент форматирования или настроенную ранее конфигурацию технологии, чтобы гарантировать, что стиль кода и, собственно, технологии остаются согласованными для разных микрофронтендов в одном монорепозитории;
- масштабирование приложения – имеется удобная возможность вносить изменения в единственном коммите в монорепозитории, в то время как если микрофронтенды разделены на множество репозиториев, то для

внесения таких изменений потребуется изменить несколько репозиториев, и нескольким командам может потребоваться синхронизировать свои наработки.

Итого, можно выделить преимущества использования монорепопозитория:

- общий код и видимость – повторное использование кода для тестирования, компонентов пользовательского интерфейса внутри одной кодовой базы;
- согласованные изменения – при изменении доступа к серверному API также изменяются микрофронтенды, использующие этот API в том же коммите. Также, можно модифицировать компонент какой-либо кнопки, и приложения, использующие этот компонент, автоматически увидят изменения – монорепозиторий избавляет от необходимости координировать и синхронизировать коммиты в нескольких репозиториях;
- мобильность разработчиков – согласованный способ создания и тестирования приложений, написанных с использованием различных инструментов и технологий;
- единый набор зависимостей – единая версия всех сторонних зависимостей, уменьшающая несоответствия и возможные технологические расхождения между приложениями.

По вышеуказанным причинам монорепозитории и микрофронтенды идеально подходят друг другу.

Кастомизация графического интерфейса

Одними из наиболее часто используемых графических фреймворков являются Bootstrap и Material UI. Оба варианта имеют плюсы и минусы (таблица 2). Для микрофронтендов на основе React.js, поскольку Material UI

основан на данной библиотеке, естественнее всего использовать последний графический фреймворк.

Критерий	**Bootstrap**	**Material UI**
Кастомизация	Обеспечивает согласованность пользовательского опыта и удобного интерфейса, имеет средние возможности настройки.	Низкая согласованность пользовательского опыта, поскольку дизайнеры создают большое множество различных пользовательских интерфейсов. Обеспечивает уникальный легко настраиваемый дизайн.
Зависимости	Приложения на основе Bootstrap довольно тяжелые и могут работать медленно, если не уделять время избавлению от ненужных компонентов и JS-скриптов.	Технология основана только на React, с использованием чистого CSS без каких-либо сторонних библиотек.
Разметка	Система сетки предназначена как для мобильных устройств, так и для настольных компьютеров (в первую очередь для мобильных устройств) с понятным и удобочитаемым пользовательским интерфейсом для всех платформ.	Предназначенная в основном для мобильных устройств и очень удобная для пользователя, но пользовательский интерфейс может быть перегружен анимированными взаимодействиями (для настольных компьютеров).
Концепция	Фреймворк над CSS, HTML, JS для разработки адаптивных веб-сайтов.	Фреймворк на React, который следует принципам Material Design от Google.

Таблица 2 – Сравнение Bootstrap и Material UI

Подходы к интеграции

Module Federation – интеграционный runtime-подход для микрофронтендных приложений, превосходит интеграционный build-time подход, предоставляемый такими фреймворками, как Bit. Однако можно рассмотреть возможность совместного использования Bit и Module Federation,

поскольку Bit хорош для публикации и управления настроенными компонентами, если решено использовать Bit для этой цели.

Представленный в Webpack 5 в качестве плагина, Module Federation набирает большую популярность с декабря 2020 года. Zack Jackson, создатель Module Federation, описывает его следующим образом: «Module Federation позволяет приложению JavaScript динамически загружать код из другого приложения — в процессе обмена зависимостями, если приложение, использующее федеративный модуль, не имеет зависимости, необходимой для федеративного кода — webpack загрузит отсутствующую зависимость из этой федеративной сборки». Иными словами, Module Federation позволяет комбинировать отдельные сборки для формирования единого приложения. Это, в свою очередь, позволяет разрабатывать и развертывать модули независимо друг от друга и комбинировать их во время выполнения. Поскольку модули объединяются в одно приложение, можно совместно использовать пакеты сторонних библиотек, обмениваться событиями, иметь доступ к глобальным объектам браузера из любой части приложения.

Минусы микрофронтендной архитектуры

Микрофронтенды, так же, как и микросервисы, не без недостатков. Перечислим самые существенные из них:

- в большинстве случаев потребуется разработать способ, позволяющий разным частям приложения взаимодействовать друг с другом, что сопряжено с дополнительными затратами ресурсов;
- также скорее всего будет необходимым добавить в приложения некоторые ограничения, чтобы обеспечить совместимость различных микрофронтендов, замещающих части приложения;
- процесс развертывания и настройка интеграции определенно запросят дополнительных усилий, так как каждый микрофронтенд необходимо развертывать и администрировать отдельно.

Вывод

Итак, насколько целесообразно внедрять микрофронтенды? Если происходит разработка небольшого сайта компании, то, скорее всего, ответ — нет. Если же работа ведется над крупным приложением, включающим большое число отдельных элементов UI или представлений, охватывающих множество доменов (а может и то, и другое сразу), то, вероятно, стоит к ним прибегнуть.

В любом случае, микрофронтендная архитектура выглядит достаточно правильным решением, способным изменить мир frontend-разработки – так же, как и микросервисы в мире backend-технологий.

Библиографический список

1. Беспарольная аутентификация [Электронный ресурс] URL: https://habr.com/ru/company/vk/blog/343288/ (дата обращения: 08.11.2022).
2. Отправляем магические ссылки с помощью Node.js [Электронный ресурс] URL: https://habr.com/ru/company/vk/blog/343288/ (дата обращения: 08.11.2022).
3. TOTP (Алгоритм Time-based One-Time Password) [Электронный ресурс] URL: https://habr.com/ru/post/534064/ (дата обращения: 09.11.2022).
4. React.js [Электронный ресурс] URL: https://reactjs.com (дата обращения: 09.11.2022).
5. Mojo.Auth [Электронный ресурс] URL: https://mojoauth.com/docs/ (дата обращения: 09.11.2022).
6. Semantic UI [Электронный ресурс] URL: https://semantic-ui.com/ (дата обращения: 09.11.2022).
7. React Router [Электронный ресурс] URL: https://reactrouter.com/en/main (дата обращения: 09.11.2022).

Bibliographic list

1. Password-free authentication [Electronic resource] URL: https://habr.com/ru/company/vk/blog/343288 / (accessed: 08.11.2022).

2. We send magic links using Node.js [Electronic resource] URL: https://habr.com/ru/company/vk/blog/343288 / (accessed: 08.11.2022).
3. TOTP (Time-based One-Time Password Algorithm) [Electronic resource] URL: https://habr.com/ru/post/534064 / (accessed: 09.11.2022).
4. React.js [Electronic resource] URL: https://reactjs.com (accessed: 09.11.2022).
5. Mojo.Auth [Electronic resource] URL: https://mojoauth.com/docs / (accessed: 09.11.2022).
6. Semantic UI [Electronic resource] URL: https://semantic-ui.com / (accessed: 09.11.2022).
7. React Router [Electronic resource] URL: https://reactrouter.com/en/main (accessed: 09.11.2022).

Для цитирования: Берьянов М.С., Монченко А.С., Дерябин А. ИССЛЕДОВАНИЕ МИКРОФРОНТЕНДНОЙ АРХИТЕКТУРЫ – ИНСТРУМЕНТЫ И РЕКОМЕНДОВАННЫЕ ПРАКТИКИ ИСПОЛЬЗОВАНИЯ//Научный сетевой журнал «Столыпинский вестник» №10/2022.