УДК 004.051

ОБЗОР ПРИМЕНЕНИЯ ПОДХОДА МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ ПРИ ПРОЕКТИРОВАНИИ КЛИЕНТСКОЙ ЧАСТИ ВЕБ-ПРИЛОЖЕНИЯ

Черников А.С.

Декан Факультета Международных Образовательных Программ, Директор Центра Международных Стратегических Партнерств, Руководитель учебно-научной лаборатории «Компьютерные информационные системы и технологии»

Московский государственный технический университет им. Н. Э. Баумана

Россия, г. Москва

Никитин И.В.

студент 2 курса магистратуры

Московский государственный технический университет им. Н. Э. Баумана

Россия, г. Москва

Гриценко Т.Ю.

студент 2 курса магистратуры

Московский государственный технический университет им. Н. Э. Баумана

Россия, г. Москва

Бородаенко Д.В.

студент 2 курса магистратуры

Московский государственный технический университет им. Н. Э. Баумана

Россия, г. Москва

Аннотация

В статье рассматриваются подход к использованию микросервисной архитектуры при проектировании клиентской части веб-приложения. Представлены особенности такой архитектуры, а также проанализированы практические варианты применения такого подхода.

Ключевые слова: веб-приложение, микросервисная архитектура, клиентская часть веб-приложения

APPROACH OVERVIEW OF MICROSERVICE ARCHITECTURE IN DESIGNING A CLIENT PART OF WEB APPLICATION

Chernikov A.S.

The Dean of The Faculty of International Educational Programmes, The Head of The Centre of International Strategic Partnerships, The Head of Educational &Research Laboratory "Computer Information Systems and Technologies"

Bauman Moscow State Technical University

Russian Federation, Moscow

Nikitin I.V.

master student

Bauman Moscow State Technical University

Russian Federation, Moscow

Gritsenko T.Y.

master student

Bauman Moscow State Technical University

Russian Federation, Moscow

Borodaenko D.V.

master student

Bauman Moscow State Technical University

Russian Federation, Moscow

Annotation

In this article, the idea of using microservice on client-side part of web-application is presented. Some features of this architecture covered; also, there are analysis of some practical solutions.

Keywords: web-application, microservice architecture, client-side part of web-application

Введение

Веб-приложения стали неотъемлемой частью нашей жизни: каждый день мы используем десятки различных веб-сайтов для решения своих задач. И от того, насколько хорошим был опыт взаимодействия пользователя с приложением, зависит его желание вернуться к этому приложению в будущем.

Любое приложение состоит из трех связанных между собой частей: клиентская часть, серверная часть и база данных. Пользователь так или иначе сталкивается с клиентской частью веб-приложения, по которой он и будет делать вывод об удобстве использования приложения в целом. Оно, в свою очередь, напрямую влияет на его желание в будущем вновь воспользоваться этим приложением. Так, например, если приложение после каждого действия перезагружает страницу или очень долго отображает контент, то желание пользоваться им у пользователя пропадает. Поэтому очень важная задача при создании веб-приложения – обеспечить удобство использования и быстрый вывод необходимой информации пользователю.

Для того, чтобы обеспечить максимальное удобство использования веб-приложением, существует три отличающихся между собой подхода к реализации клиентской части приложения: многостраничное приложение (Multi Page Application, MPA), одностраничное приложение (Single Page Application, SPA) и микросервисы. В данной статье дается краткий обзор MPA и SPA подходов, рассмотрено отличие этих подходов от микросервисов, особенности реализации микросервисов на клиентской части в отличие от серверной части. В конце статьи представлен анализ нескольких решений по реализации микросервисов на клиентской части.

Многостраничное приложение, MPA

Многостраничное приложение является «классическим» подходом к реализации веб-приложения, который используется достаточно долго. Основная идея такого подхода – приложение состоит из нескольких отдельных страниц, каждая из которых загружается по мере необходимости. В результате, когда пользователь заканчивает взаимодействие со страницей (заполнил форму и нажал кнопку отправки данных или перешел по ссылке на другую страницу), на сервер совершается запрос за новой страницей, которая отображается пользователю. Схема работы многостраничного приложения отображена на рис. 1.

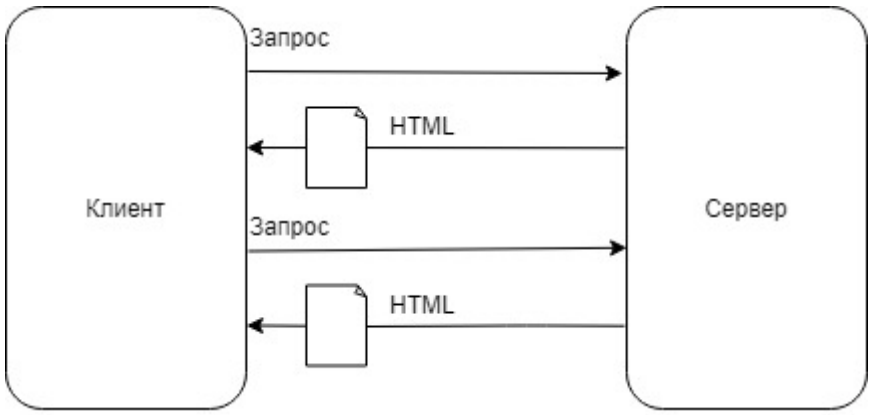

Рис. 1 – Схема работы MPA приложения

Главное преимущество такого подхода – простота реализации со стороны клиента. Для работы такого приложения необходима HTML-разметка будущей страницы с возможностью шаблонизации для того, чтобы добавить данные с сервера в страницу (такую возможность предоставляет большинство современных фреймворков). Однако использование фреймворка может вызвать проблемы, связанные с тем, что их становится два на сервере: один для обработки запросов и один для шаблонизации. При развитии приложения этот вопрос может стать важным, так как из-за наличия большого количества фреймворков много времени может уходить на их поддержку. Также, важным достоинством многостраничного приложения является возможность его

индексирования поисковыми роботами, в результате в поисковых системах приложение будет появляться чаще.

К недостаткам такого подхода можно отнести то, что при переходе на каждую новую страницу ее необходимо загрузить. Однако, помимо разметки страницы, необходимо загрузить дополнительные данные: стили приложения, картинки, шрифты, js-файлы. Загрузка последнего как раз и может вызвать проблемы. Приложения, обладающие какой-либо интерактивностью, достигают ее за счет использования JavaScript языка, исполняемого в браузере. И в большинстве случаев этот код требует подключения дополнительных библиотек. В результате, при каждой загрузке страницы каждый раз происходит браузер выполняет запрос за файлами этой библиотеки (который может быть оптимизирован за счет кэширования), а также инициализация и, соответственно, блокирование на это время потока выполнения JavaScript-скрипта. В результате увеличивается время до возможности первого интерактивного взаимодействия с приложением. Также стоит отметить, что использование фреймворка шаблонизации сильно завязано на данных, приходящих с сервера, а это значит, что разделить клиентскую часть и серверную сложно. Отсутствие границы между этими частями может усложнить работу и поддержку кода.

Так как каждый раз запрашивать страницу с измененными данными может быть достаточно накладно, одной из оптимизаций многостраничных приложений является реализация возможности выполнения запроса за данными без перезагрузки страницы. В случае, если необходимо отобразить какие-либо новые данные с сервера, браузером выполняется запрос за этими данными на сервер и полученные данные выводятся пользователю. В результате нет необходимости перезапрашивать и ожидать загрузки всей страницы каждый раз.

Одностраничное приложение, SPA

Учитывая выше описанный недостаток многостраничных приложений, связанный с тем, что необходимо каждый раз перезагружать страницу, одним из вариантов оптимизации – загрузить все необходимые данные шаблонов страниц один раз, а переключение страниц осуществлять уже в браузере специальным скриптом. Таким образом пользователь видит некую общую часть приложения, которая не изменяется при перезагрузках страниц, при этом остальное содержимое страницы изменяется, но на самом деле никаких дополнительных запросов непосредственно за шаблоном страницы не происходит. Вместо этого браузером выполняется запрос только за данными, которые возвращаются в специальном формате, например, JSON. Схема работы одностраничного приложения показана на рис. 2.

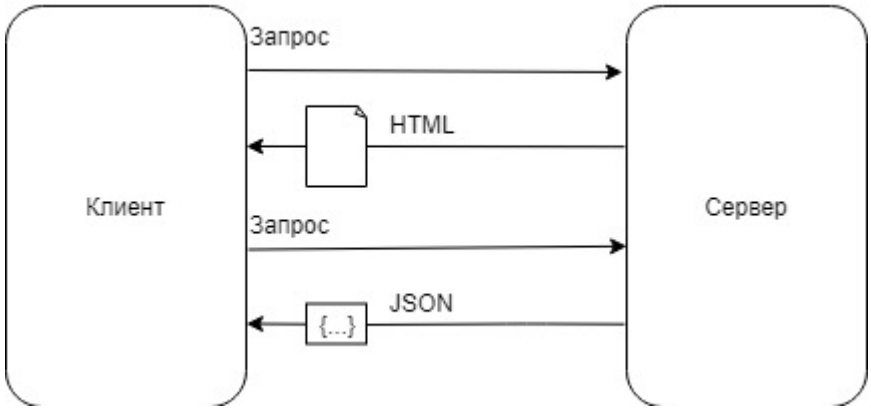

Рис. 2 – Схема работы SPA приложения

Подход одностраничного приложения имеет ряд особенностей, которые необходимо упомянуть. Так как большая часть работы в таком подходе выполняется на стороне клиента (браузера), то важным становится организация клиентского кода, написанного на JavaScript. В отличие от предыдущего подхода, когда такой код в основном служил для того, чтобы обеспечить интерактивность, в подходе одностраничного приложения этот код также отвечает за навигацию на странице и отображение необходимых данных для каждой страницы, которую просматривает пользователь. На данный момент

уже существует большое количество специальных библиотек, которые позволяют упростить разработку одностраничных приложений: AngularJs [5], Ember.js [7], Meteor.js [10], Knockout.js [9]. Все эти библиотеки бесплатны для использования и поддерживаются крупными компаниями-создателями и сообществом.

К преимуществам одностраничного приложения кроме того, что нет необходимости каждый раз загружать новую страницу, можно также отнести то, что разработка и отладка таких приложений проще, чем в случае многостраничного приложения. Так как вся основная логика находится на стороне клиента, то для разработки достаточно открыть такое приложение в браузере и с помощью инструментов разработки отлаживать приложение. Также важным преимуществом такого подхода является то, что серверную часть, написанную под одностраничное приложение, можно переиспользовать при разработке мобильных приложений.

Однако у такого подхода также есть и свои недостатки. Основной недостаток – одностраничные приложения плохо оптимизированы для поисковых роботов. Если в случае многостраничного приложения все данные уже отдаются вместе со страницей и могут быть прочитаны роботом, то в случае одностраничного приложения данные запрашиваются уже после того, как пользователь получил страницу. А это значит, что на полученной странице нет никаких данных, которые могут быть необходимы поисковому боту. Однако на данный момент проблема с поисковыми роботами уже не является проблемой. Существуют специальные библиотеки [13], которые позволяют подготавливать данные в одностраничных приложениях и первую страницу, которую запросил пользователь, и загружать их с уже подготовленными данными. Такая технология называется серверным рендерингом и похожа на принцип работы многостраничного приложения.

Помимо этого, стоит отметить то, что хоть весь необходимый статический контент будет запрошен один раз при загрузке страницы, часто этот контент содержит много логики, а значит файлы большие по размеру. Также необходимо помнить, что необходимо загрузить библиотеки, которые позволяют работать с одностраничными приложениями. Все это приводит к тому, что пользователю необходимо, во-первых, загрузить все необходимые файлы, а, во-вторых, проинициализировать их на стороне пользователя. Все эти операции могут занимать много времени, что негативно скажется на желании пользователя воспользоваться приложением.

К еще одному важному недостатку можно отнести то, что пользователь в браузере может отключить выполнение JavaScript-кода, и тогда приложение либо будет работать неправильно для пользователя, либо не будет работать вообще.

Многостраничные и одностраничные приложения являются общеизвестными и принятыми подходами к реализации веб-приложения [14]. Каждый из подходов имеет свои достоинства и недостатки, которые необходимо учитывать при выборе определенного подхода [15]. Если важно, чтобы приложение было правильно прочитано поисковыми роботами или количество страниц в приложении много, то лучше выбрать многостраничное приложение. Такой подход может хорошо себя показать, например, в сайтах электронной коммерции, где важно, чтобы поисковыми роботами был прочитан товар и этот товар пользователь увидел при поиске. Если в планах есть разработка мобильного приложения или количество страниц небольшой или необходимо совершать плавные переходы между страницами, то предпочтение лучше отдать одностраничному приложению. Такой подход может подойти для небольших приложений, которые обладают большим функционал на нескольких страницах. Например, сайты для прослушивания музыки или просмотра видео могут использовать одностраничный подход.

Микросервисы на клиенте, Микрофронтенды

По мере роста веб-приложения в нем появляется много разных частей, требующих разных библиотек и фреймворков для своей работы. Так, например, главная страница может быть написана с использованием одного набора библиотек, а личный кабинет пользователя с другим. Также на определенном уровне жизни приложения такие части как главная страница и личный кабинет могут быть слабо связаны с клиентской точки зрения. В результате возникает необходимость разработки и доставки изменений функционала отдельных частей приложения без привязки к общему коду. В описанных ранее подходах такой возможности нет: в случае многостраничного приложения есть сильная привязка к серверному коду и такое разделение без каких-либо изменений как со стороны клиентского кода, так и со стороны серверного почти невозможно; в случае одностраничного подхода клиентский код наоборот завязан друг на друге, так как должен обеспечивать бесшовный переход между страницами, поэтому такой подход тоже плохо подходит для решения описанных проблем. В результате была выдвинута новая концепция, которая позволяет разрабатывать части приложения и доставлять изменения до пользователей независимо друг от друга – микросервисы на клиенте или микрофронтенды [4].

Концепция микрофронтендов достаточно новая и активно развивается. Как было сказано ранее, основное преимущество такого подхода – независимая разработка и доставка изменений функционала различных частей приложения. При этом сами эти части могут быть написаны с использование разных технологий, никак не пересекающихся между собой. На данный момент нет четкого представления о том, какой именно подход является лучшим для того, чтобы учесть все проблемы, которые могут возникнуть у пользователей (к основным из таких проблем можно отнести смену адресов страниц на стороне браузера и конфликтующие стили разных частей приложения), однако любая реализация будет состоять как минимум из двух частей: это сами

микрофронтенды, а также специальный «сшивающий» слой. Этот слой необходим для того, чтобы страница для пользователя, которая может состоять из данных от нескольких микрофронтендов, загружалась пользователю не отдельными частями, а сразу одной собранной, готовой к использованию страницей. В результате схему работы можно описать следующим образом: браузер осуществляет запрос для получения данных страницы; этот запрос попадает на сшивающий слой, который посылает запросы в необходимые микрофронтенды; микрофронтенд получает запрос, если необходимо делает запрос для получения данных от сервера и возвращает свою часть страницы сшивающему слою; получив данные от микрофронтендов, сшивающий слой «собирает» итоговую страницу и отправляет ее пользователю. Схема взаимодействия приложения с микрофронтендами представлена на рис. 3.

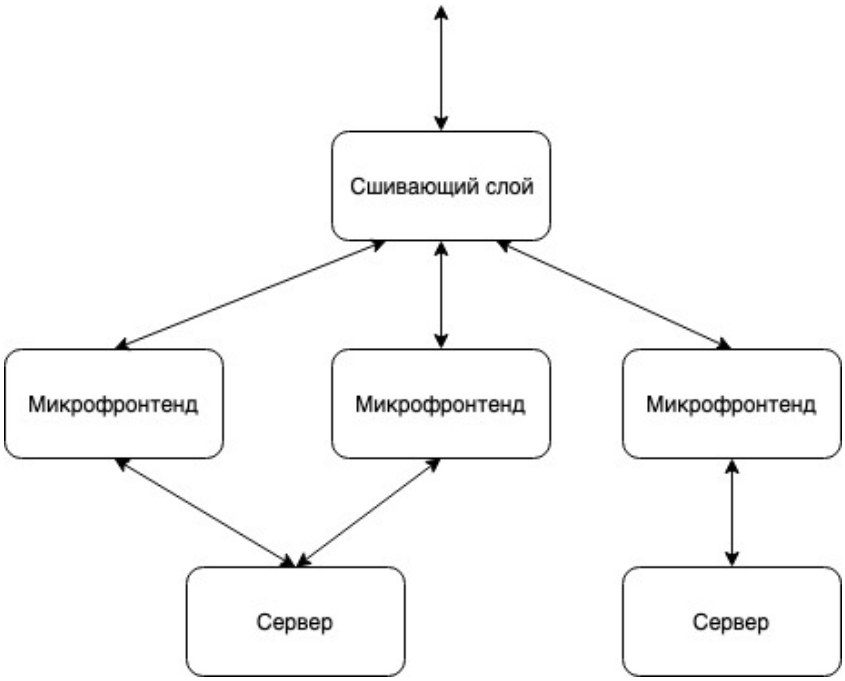

Рис. 3 – Схема взаимодействия элементов в приложении с микрофронтендами

Хоть такая схема и кажется хорошей, и удобной, она обладает рядом недостатков. Самый главный ее недостаток – сложность создания. Как

отмечалось ранее при разработке микрофронтендов необходимо решить такие проблемы, как конфликты стилей между разными частями (так, например, стили одного приложения могут переопределять стили другого, в результате пользователь видит не то, что предполагалось) или проблема смены адресов страниц (кто принимает решение какую часть приложения загружать пользователю; какая часть адреса является общей, а какая относится уже к конкретному микрофронтенду). Помимо этого, как отмечалось ранее, одно из преимуществ одностраничного приложения – это бесшовный переход между страницами для пользователя. Однако в микрофронтендах все не так просто: приложение – это набор отдельных независимых компонентов. Поэтому для того, чтобы обеспечить бесшовный переход между страницами приложениями необходима дополнительная логика. Также необходимо решить проблему обмена данными между микрофронтендами. Из-за того, что микрофронтенды независимы друг от друга, они ничего не знают о соседних микрофронтендах. В результате необходимо предусмотреть какой-нибудь механизм пересылки сообщений (например, браузерные события).

Все выше обозначенные проблемы можно отнести к непосредственно клиентским, которые могут быть решены на самых ранних стадиях жизни приложения. Однако помимо этого стоит отметить, что в подходе микрофронтендов также стоит обратить внимание на инфраструктурные проблемы (независимость доставки изменений пользователю) и серверные (часть логики сшивающего слоя может находится на сервере, а также может присутствовать серверный рендеринг). О микрофронтендах написано много статей описывающие плюсы и минусы данного подхода ([11] и [16]). Кроме этого уже сейчас есть несколько готовых проектов, которые предоставляют инструменты для решения обозначенных ранее проблем.

Проект Мозаик (Project Mosaic)

Проект мозаик [12] – это набор библиотек для построения микроархитектуры на фронтенде. Проект был представлен в 2016 году и до сих пор активно развивается, благодаря открытому исходному коду. Он состоит из следующих компонентов:

- Роутер (Router) – используется для определения адресов запросов
- Файл-конфигурации для роутера (Router config Api) – Содержит описание всех возможных адресов, используемых в приложении
- Сервис шаблонов (Layout service) – Сервис, отвечающий за получение итогового ответа
- Хранилище шаблонов (Layout storage) – Хранилище шаблонов, используемых в приложении и его частях
- Фрагменты (Fragments) – составные части шаблонов

Общая схема взаимодействия этих компонентов выглядит представлена на рисунке 4.

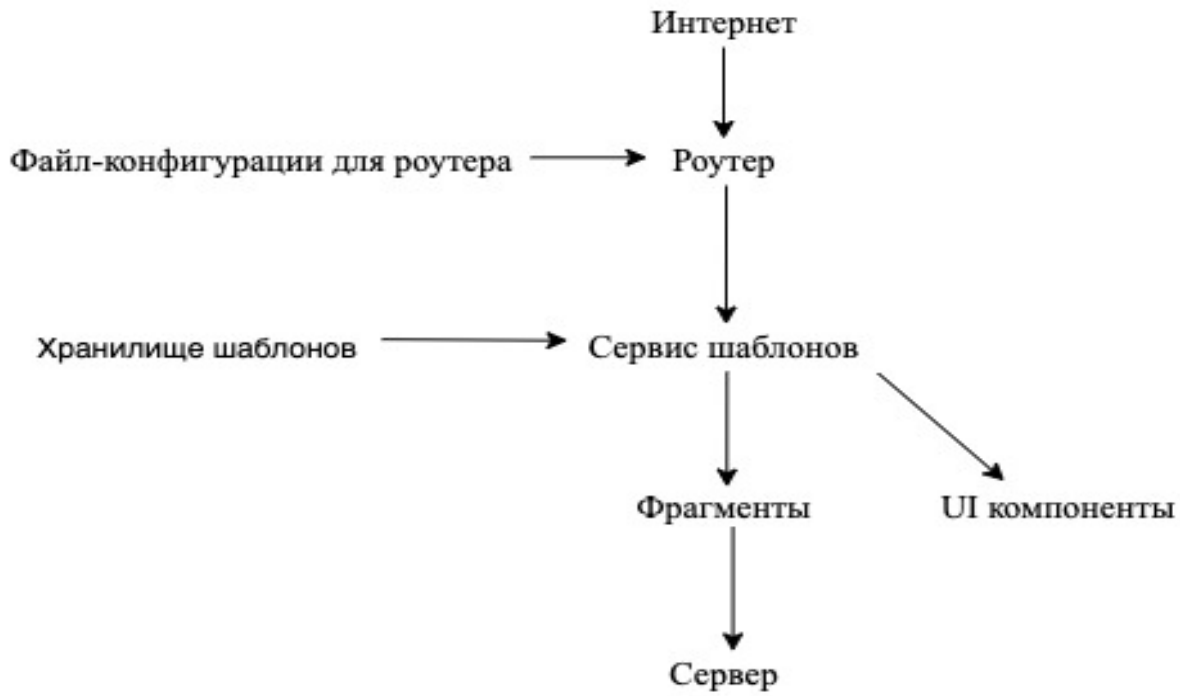

Рис. 4 – Схема взаимодействия элементов в проекте Мозаик

Работает система следующим образом: файл-конфигурации для роутера содержит в себе описание всех возможных адресов, которые поддерживаются данным приложением. Этот файл используется роутером для определения адреса, по которому будет запрошен шаблон. Сам роутер принимает HTTP запрос и перенаправляет его на сервис шаблонов для того, чтобы получить необходимую страницу. Сервис шаблонов использует хранилище шаблонов для определения частей, из которых будет состоят итоговая страница. После того, как сервис шаблонов будет знать из каких фрагментов будет состоят страница, он параллельно запросит их и, по мере получения фрагментов страницы, будет собирать итоговую страницу. После того, как все необходимые страницы получены, результирующая страница будет возвращена пользователю.

Все компоненты проекта – отдельные небольшие приложения с открытым исходным кодом. В результате любой желающий может как подробно изучить работу каждого компонента и понять, как именно были решены проблемы микрофронтендов, а также понять какие проблемы все еще актуальны и помочь решить их.

Библиотека Apa (Ara Framework)

Другим примером реализации микрофронтендов является библиотека Apa [6]. Проект был представлен в 2019 году и постепенно набирает популярность, благодаря открытому исходному коду. В его основе, как и в проекте Мозаик, лежит несколько логических элементов:

- Привязки (Nova Bindings) – позволяет использовать различные клиентские библиотеки на одной странице
- Директивы (Nova Directive) – встраивает в итоговую страницу заглушки компонентов, а также возвращает информацию, необходимую для запроса за компонентами, которые заменят заглушки

- Посредник (Nova proxy and middleware) – Сервис, который содержит в себе описание всех доступных в приложении адресов, а также этот же сервис совершает запрос за компонентом, который заменит заглушку
- Кластер (Nova cluster) – агрегирует информацию со всех сервисов микрофронтедов

Схема взаимодействия компонентов представлена на рис. 5.

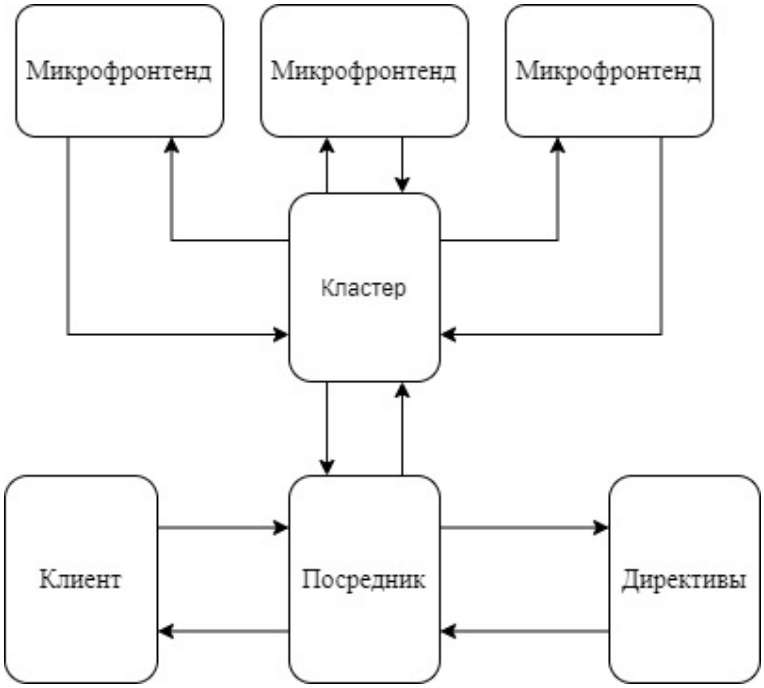

Рис. 5 - Схема взаимодействия элементов библиотеки Apa

Библиотека работает следующим образом: клиент отправляет запрос, который попадает на Посредник-а. Определив какую страницу запрашивает пользователь, Посредник совершает запрос к сервису Директив. Сервис Директив возвращает Посреднику страницу, которая содержит заглушки элементов и данные, необходимые для получения самих элементов вместо страниц. После этого Посредник отправляет запрос в кластер, который уже отправляет запросы непосредственно в микрофронтенды и возвращает результат посреднику, который изменяет страницу в соответствии с новыми данными и отправляет итоговую страницу клиенту.

В отличие от проекта Мозаик, библиотека Ара представляет собой единый набор компонентов, которые не могут существовать в отдельности. Однако исходный код проекта открыт, поэтому проект также развивается, благодаря сообществу.

Применение подхода микросервисной архитектуры

Как отмечалось ранее, подход микросервисной архитектуры является относительно новым, однако уже сейчас можно найти множество применений этого подхода в разных крупных компаниях.

Так, к примеру, компания Netcracker столкнулась с рядом проблем при разработке своего приложения. К проблемам можно отнести монолитность приложения, из-за которой разработчики перестают быть универсальными (то есть решать проблемы разного рода), также возникли проблемы с доставкой изменения функционала приложения и управлением зависимостями. В результате перехода на микросервисную архитектуру получилось добиться независимости фрагментов приложения и повысить стабильность в целом; также отдельные фрагменты просты в усвоении одним разработчиком. Однако появился ряд новых проблем: взаимодействие фрагментов между собой, за роутинг приложения отвечает один человек, ошибки при обращении к фрагментам [1].

Другой пример компании, использующей микрофронтенды – Почта Mail.Ru [3]. В результате перехода на микросервисы получилось добиться ускорения разработки и тестирования, уменьшился порог входа для нового разработчика. Однако переход на такой тип архитектуры потребовал много ресурсов: было использовано два различных подхода к сборке проекта и интеграции микросервисов – с помощью iframe и bundle-сборки.

Подход микросервисов на клиентской части попробовали применить и в банке Точка [8]. Огромная кодовая база (порядка 30 мегабайт) и жесткий релизный цикл на эту кодовую базу вызывал большие проблемы. Основной

посыл перехода на микросервисы – уменьшить связанность кода и уменьшить релизный цикл. В результате приложение было разбито на отдельные модули, при этом каждый из них использует разные библиотеки и фреймворки, а релиз каждой части происходит отдельно.

ЦИАН также для разработки своего клиентского кода решил использовать микросервисный подход [2]. Применение подхода микросервисной архитектуры позволило уменьшить порог входа в проект, а также упростило доставку изменения функционала приложения: теперь какое-либо изменение может быть доставлено отдельно для каждой части приложения. Однако возникли и проблемы: релизы должны быть синхронизированы, а также приложения вынуждены нарушать изоляцию друг друга, в результате микросервисы перестают быть независимыми.

Анализируя приведенные выше примеры можно сделать вывод, что для большинства компаний главное преимущество микрофронтендов – короткий и независимый релизный цикл. В результате изменения до пользователей доставляются быстро, а в случае каких-либо проблем исправления могут быть внесены в отдельную часть приложения, которая необходимо поправить.

Однако при этом микрофронтенды подходят не всем компаниям. Можно обратить внимание, что переходы к микрофронтендам требовали от компаний больших вложений сил, при этом так и не удалось решить всех проблем. Если изучить исходные коды проекта Мозаик и библиотеки Ара, то можно увидеть, что обе библиотеки пытаются решить инфраструктурные проблемы, так как основная сложность данного подхода – это настройка инфраструктуры. Если проект небольшой или нет необходимости в разделении отдельных частей приложения на маленькие независимые сервисы, то не следует использовать микросервисный подход, так он может принести больше вреда, чем пользы. В случае маленьких проектов гораздо выгоднее использовать подходы

одностраничного приложения или многостраничного, так как они проще в реализации.

Заключение

Клиентская часть приложения необходима для того, чтобы у пользователя был приятный опыт использования приложения. Этот опыт напрямую влияет на то, будет ли пользователь в дальнейшем пользоваться этим приложением или нет. На данный момент существует три основных архитектурных подхода к разработке клиентской части приложения: многостраничные приложения, одностраничные приложения и микросервисы. Многостраничные приложения состоят из большого числа отдельных веб-страниц, между которыми происходит переход и перезагрузка всего приложения. Такой подход хорошо подходит для приложений, содержащих большое количество разнородных страниц, например, крупные интернет-магазины. Одностраничные приложения представляют собой приложение, состоящие из одной страницы, в которой данные меняются с помощью кода, выполняемого на клиентской части. Такой подход хорошо использовать там, где важно, чтобы пользователь не замечал переходов между экранами (или они были не настолько явными), например, для сайтов-визиток или любых других приложений, где необходимо донести до пользователя информацию. Микросервисный подход является более новым подходом и сочетает в себе преимущества предыдущих двух: приложение состоит из большого количества независимых компонентов/страниц, при этом переход между ними может осуществляться так же, как и в одностраничном. Такой подход необходимо использовать на поздних этапах жизни крупного веб-приложения, так как необходимо учесть и решить определенные трудности, связанные со сложной инфраструктурой подхода. Однако с учетом того, что подход относительно новый, многие проблемы могут быть решены в ближайшем будущем, а ряд крупных компаний уже используют его при разработке своего приложения.

Библиографический список

1. Микросервисный фронтенд – современный подход к разделению фронта [Электронный ресурс]. – Режим доступа – URL: https://habr.com/ru/company/netcracker/blog/420753/ (Дата обращения 05.05.2019)
2. Микросервисный фронтенд [Электронный ресурс]. – Режим доступа – URL: https://www.youtube.com/watch?v=YANolrn4PYc (Дата обращения 29.04.2019)
3. Микросервисы на фронтенде в Почте Mail.Ru [Электронный ресурс]. – Режим доступа – URL: https://www.youtube.com/watch?v=ebly22v8z18 (Дата обращения 05.05.2019)
4. Микрофронтенды: о чем это мы? Электронный ресурс – Режим доступа – URL: https://habr.com/ru/company/raiffeisenbank/blog/459540/ (Дата обращения 20.09.2019)
5. AngularJs [Электронный ресурс] – Режим доступа – URL: https://angularjs.org/ (Дата обращения 04.05.2019)
6. Ara Framework [Электронный ресурс] – Режим доступа – URL: https://ara-framework.github.io/website/ (Дата обращения 10.03.2020)
7. Ember.js [Электронный ресурс] – Режим доступа – URL: https://emberjs.com/ (Дата обращения 04.05.2019)
8. Frontend-приложение как микросервисы [Электронный ресурс]. – Режим доступа – URL: https://www.youtube.com/watch?v=S2iu8nbFpYo (Дата обращения 29.04.2019)
9. Knockout.js [Электронный ресурс] – Режим доступа – URL: https://knockoutjs.com/ (Дата обращения 04.05.2019)
10. Meteor.js [Электронный ресурс] – Режим доступа – URL: https://www.meteor.com/ (Дата обращения 04.05.2019)

11. Microfrontends: An approach to building Scalable Web Apps [Электронный ресурс]. – Режим доступа – URL: https://medium.com/@areai51/microfrontends-an-approach-to-building-scalable-web-apps-e8678e2acdd6 (Дата обращения 29.04.2019)
12. Project Mosaic, Microservices for the Frontend [Электронный ресурс]. – Режим доступа – URL: https://www.mosaic9.org/ (Дата обращения 01.05.2019)
13. ReactDOMServer – [Электронный ресурс] – Режим доступа – URL: https://ru.reactjs.org/docs/react-dom-server.html (Дата обращения 20.06.2019)
14. Single-page application vs. Multiple application [Электронный ресурс]. – Режим доступа – URL: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58 (Дата обращения 05.05.2019)
15. Single Page Application (SPA) и Multi Page Application (MPA): преимущества и недостатки [Электронный ресурс]. – Режим доступа – URL: https://merehead.com/ru/blog/single-page-application-vs-multi-page-application/ (Дата обращения 01.05.2019)
16. Understanding Micro Frontends [Электронный ресурс]. – Режим доступа – URL: https://hackernoon.com/understanding-micro-frontends-b1c11585a297 (Дата обращения 05.05.2019)

Оригинальность 90%