ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ИХ БЕЗОПАСНОСТЬ

ОЦЕНКА ВОЗМОЖНОСТИ ИСПОЛЬЗОВАНИЯ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ ПРИ РАЗРАБОТКЕ ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ КЛИЕНТ-СЕРВЕРНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

***М.Г. Городничев**, к.т.н., доцент, Московский технический университет связи и информатики, 8А, gorodnichev89@yandex.ru*;
***Р.В. Полонский**, Московский технический университет связи и информатики, rvpolonskiy@gmail.com.*

УДК 004

Аннотация. Цель проведенного авторами исследования – оценить возможность использования микросервисной архитектуры в разработке современных веб-интерфейсов. Научная новизна исследования заключается в исправлении недостатков монолитного подхода при разработке веб-интерфейсов за счет использования микросервисной архитектуры. В статье сделан обзор типов современных веб-приложений и архитектуры, используемой для их разработки, дано сравнение микросервисной и монолитной архитектуры. В результате выделены и описаны ключевые аспекты, требующие проработки перед успешным применением микросервисного подхода, а также рассмотрены достоинства и недостатки такого подхода в приложении к разработке современных веб-интерфейсов.

Ключевые слова: микросервисная архитектура; монолитная архитектура; пользовательский интерфейс; веб-приложение; клиент-серверное программное обеспечение; микро-фронтенд.

ASSESSMENT OF POSSIBILITIES TO APPLY MICROSERVICE ARCHITECTURE APPROACH FOR CLIENT-SIDE DEVELOPMENT IN CLIENT-SERVER SOFTWARE

***Mikhail Gorodnichev,** Ph.D., associate professor, Moscow technical university of communications and informatics;*
***Roman Polonsky,** Moscow technical university of communications and informatics.*

Annotation. The purpose of the study is to evaluate the possibility of using microservice architecture in the development of modern web interfaces. The scientific novelty of the research is to correct the shortcomings of the monolithic approach in the development of web interfaces through the use of microservice architecture. The article reviews the types of modern web applications and the architecture used for their development, and compares microservice and monolithic architecture. As a result, the key aspects that need to be worked out before the successful application of the micro-service approach are identified and described, as well as the advantages and disadvantages of this approach in the application to the development of modern web interfaces.

Keywords: micro service architecture; monolithic architecture; user interface; web application; client-server software; micro frontend.

Введение и ключевые понятия

В наше время программное обеспечение используется повсеместно во всех сферах деятельности современного человека. Мы пользуемся программным обеспечением явно либо неявно ежедневно, как для выполнения рабочих задач, так и в целях досуга. Формально программное обеспечение предназначается для выполнения требуемых пользовательских задач и непосредственного взаимодействия с человеком-пользователем.

В последние десятилетия приобрело популярность программное обеспечение, основанное на клиент-серверной архитектуре – *клиент-серверное программное обеспечение*. Данный тип программного обеспечения состоит из двух основных частей – *клиентской* и *серверной*.

В серверной части (сервер) реализована логика обработки запросов от клиентской части (клиент), выполнения требуемых вычислений, подготовки данных, формирования ответа для клиента в соответствии с согласованным протоколом обмена информацией. Иными словами, серверная часть ответственна за корректное получение, хранение, обработку и возвращение информации для клиентской части, а в конечном итоге, для пользователя данного программного обеспечения.

В клиентской части имплементирован *пользовательский интерфейс*, логика формирования и исполнения запросов к серверу, обработки результатов запросов, представления обработанных результатов в удобном для пользователя виде.

В современной терминологии корректно называть клиентскую часть пользовательским интерфейсом. Также широкое распространение и общее признание среди современных специалистов по разработке программного обеспечения получили англицизмы *фронтенд* и *бэкенд*:

- фронтенд (англ. *front-end*) – клиентская часть, пользовательский интерфейс;
- бэкенд (англ. *back-end*) – серверная часть.

Среди современного клиент-серверного программного обеспечения особо распространены так называемые *веб-приложения*. Веб-приложение – это прикладное клиент-серверное программное обеспечение, в котором взаимодействие фронтенда и бэкенда происходит с помощью специального веб-обозревателя – браузера.

Пользовательский интерфейс веб-приложений – *веб-интерфейс* – с каждым годом становится сложней как по объемам доступной функциональности, так и по количеству ресурсов, требуемых для разработки, доставки и поддержки требуемой функциональности. Иначе говоря, трудоемкость разработки веб-интерфейсов растет в связи с тенденцией к усложнению пользовательских интерфейсов, увеличению функциональности программного обеспечения, развитию стандартов качества и удобства использования веб-приложений. По различным оценкам время, требуемое для разработки веб-интерфейсов, составляет не менее половины общего времени разработки веб-приложения.

С развитием и усложнением веб-интерфейсов развиваются и подходы к их разработке. Далее рассмотрим основные подходы к разработке пользовательских интерфейсов и современные типы веб-приложений.

Типы современных веб-приложений

В настоящее время можно выделить две классификации типов веб-приложений. По одной классификации веб-приложения делятся на следующие типы:

- статические сайты (англ. *static-page website*) – приложения, возвращающие одно и то же жестко закодированное содержимое с сервера каждый раз, когда запрашивается конкретный ресурс;
- динамические сайты (англ. *dynamic website*) – приложения, где часть содержимого ответа генерируется динамически по необходимости, пользовательский интерфейс создается путем вставки данных в заполнители в шаблонах.

По другой классификации веб-приложения можно разделить на следующие типы:

- одностраничное приложение (англ. *single-page application*) – приложение, использующее один *html* документ как контейнер для всех экранов веб-интерфейса, генерируемых динамически на клиентской стороне;
- изоморфное приложение (англ. *isomorphic application*) или универсальное приложение – приложение, в котором генерация экранов пользовательского интерфейса может исполняться как на клиентской стороне, так и на серверной, в зависимости от контекста исполнения.

Архитектура современных веб-интерфейсов

Несмотря на разнообразие типов веб-приложений, архитектура пользовательского интерфейса этих приложений всегда представляет собой единое и неделимое ядро, иначе говоря монолит. В этом случае клиентская часть полностью замкнута в контексте поведения, разрабатывается и разворачивается в эксплуатацию как один неделимый элемент (рис. 1).

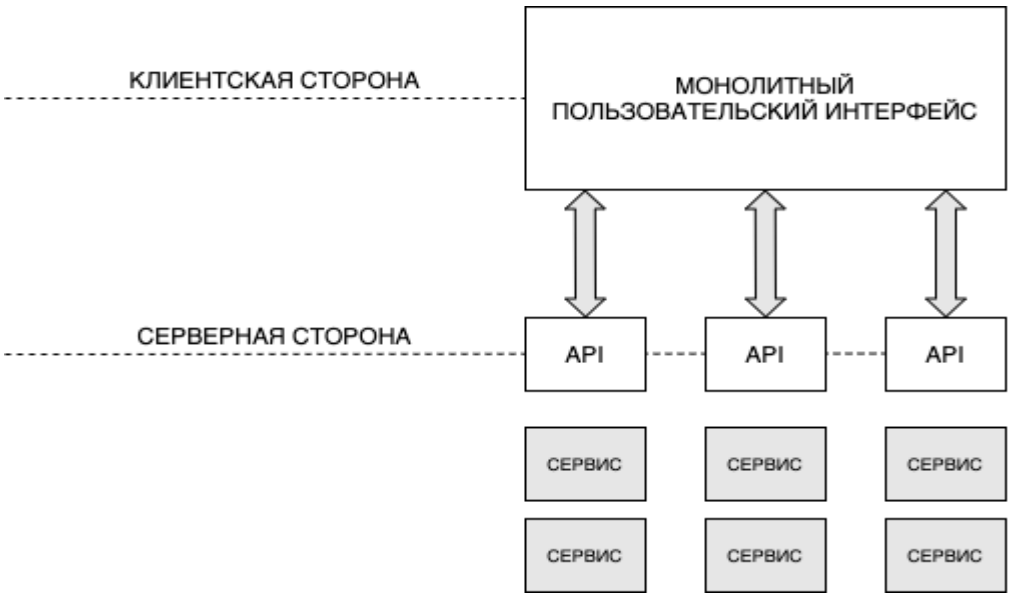

Рисунок 1

У такого подхода есть ряд недостатков, который с ростом сложности пользовательского интерфейса становится все более и более ощутимым:

- сложность или невозможность изменения технологического стека клиентской части во время разработки;
- необходимость полного обновления системы при изменении незначительных деталей клиентской части;
- в случае аварийного завершения работы весь функционал становится недоступным для пользователей;
- сложность и дороговизна масштабирования.

В связи с этим, в последнее время появляется интерес к *микросервисному подходу* к построению архитектуры пользовательского интерфейса. Тем не менее, в текущее время в сообществе нет общего понимания, сформировавшихся принципов и богатого опыта в этом направлении.

Использование микросервисной архитектуры в пользовательских интерфейсах

Развитие в сторону использования микросервисов для разработки пользовательских интерфейсов является очень перспективным и исследование этого направления в данный момент весьма актуально. Микросервисный подход призван решить ряд проблем и недостатков, описанных выше.

Тем не менее стоит здраво оценивать возможность применения рассматриваемого подхода в пользовательских интерфейсах, чтобы четко понимать не только его плюсы, но и минусы.

Главное отличие микросервисной архитектуры от монолитной – использование микросервисов, специализированных модулей (программ), в совокупности описывающих всю бизнес-логику веб-приложения. Микросервисы могут разрабатываться и развертываться независимо друг от друга. Каждый микросервис инкапсулирует определенную часть функционала веб-приложения и создается, исходя из особенностей предметной области.

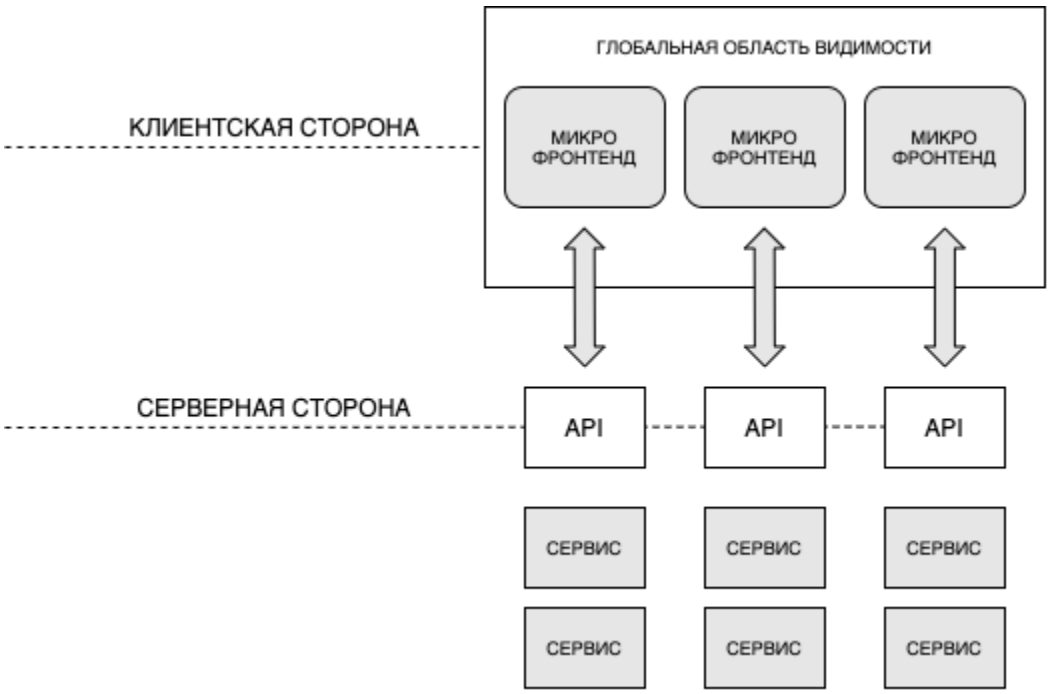

Рисунок 2

В приложении к пользовательскому интерфейсу определение микросервиса можно сформулировать следующим образом:

- микросервисом можно считать полностью изолированную, независимую часть интерфейса, так называемый фрагмент интерфейса, предоставляющий доступ к единой по типу информации или единой по бизнес-логике и предметной области функциональности;
- отдельно взятый микросервис пользовательского интерфейса отвечает за определенный набор функционала от начала до конца, то есть является полнофункциональным и самодостаточным; может быть разработан, введен в эксплуатацию и выведен из нее независимо от других частей пользовательского интерфейса.

Для удобства введем и будем использовать в дальнейшем понятие микро-*фронтенда* – как отдельного микросервиса клиентской части веб-приложения или, иначе говоря, фрагмента пользовательского интерфейса клиент-серверного программного обеспечения. Схематично микросервисная архитектура пользовательского интерфейса представлена на рис. 2.

Принципы построения микросервисов пользовательского интерфейса

Сформулируем следующие принципы, необходимые для успешного применения подхода микросервисной архитектуры и эффективной разработки микро-фронтендов:

1. Проектирование вокруг предметной области.

Моделирование микро-фронтендов в соответствии с правилами предметно-ориентированного проектирования (англ. *Domain-driven design, DDD*) и возможно, и очень значимо. Сами правила *DDD* помогают создавать программные абстракции, иначе говоря, модели предметных областей, и позволяют значительно ускорить процесс проектирования программного обеспечения.

Для соответствия этому принципу следует инвестировать ресурсы на описание всех моделей предметной области разрабатываемого программного обеспечения в самом начале его разработки.

Затем не составит большого труда разделить клиентскую часть на микро-фронтенды, каждый из которых отражает определенную модель предметной области.

2. Скрытие деталей реализации.

Для достижения автономности каждого микро-фронтенда необходимо скрывать детали реализации и не допускать возможности использования частей одного микро-фронтенда в другом. В противном случае есть риск получить хрупкую архитектуру всего веб-интерфейса, где при изменении или отказе одного микро-фронтенда, выходят из строя или работают некорректно зависимые от него части системы, причем зависимые зачастую неявным образом.

3. Культура автоматизации.

Так как микро-фронтенды это множество автономных программных единиц, особо остро встает вопрос автоматизации тестирования, доставки в эксплуатацию и обновления каждого отдельно взятого микро-фронтенда. Необходима хорошая культура автоматизации микросервисов. Лучшие практики должны применяться не только при автоматизации серверных микросервисов, но и клиентских микро-фронтендов. Это позволит командам разработки двигаться быстрей и создавать надежные решения.

4. Децентрализация.

Децентрализация разработки при использовании микросервисного подхода дает возможность каждому разработчику принимать решения независимо и эффективней решать поставленные задачи, в отличие от централизованной разработки монолитов, когда большая часть ключевых решений должна быть принята или одобрена наиболее опытными и компетентными специалистами организации.

Децентрализация также делает жизненный цикл программного обеспечения более быстрым, так как избавляет от множества этапов централизованного контроля за целостностью и работоспособностью всего веб-приложения при изменении каждого из его компонентов.

5. Высокая прозрачность.

С увеличением сложности архитектуры фронтенда становится необходимым использование инструментария, позволяющего четко и ясно понимать, где программное обеспечение дает сбой и почему. В современной разработке возможность быстро понять и устранить причину неполадки становится гораздо более важной, чем превентивная стратегия недопущения такой неполадки. Скорость разработки и выпуска программного обеспечения в эксплуатацию зачастую является более критичной, чем инвестирование значительных ресурсов в создание полностью безотказного продукта. Таким образом, прозрачность и контролируемость микро-фронтенда является необходимым принципом для успешного внедрения микросервисной архитектуры разработки пользовательских интерфейсов.

Подходы к компоновке микросервисов пользовательского интерфейса

Выделим следующие подходы к компоновке микро-фронтендов:

1. Компоновка на клиентской стороне.

При данном подходе точка входа пользовательского интерфейса подгружает все требуемые микро-фронтенды напрямую из сети доставки содержимого (англ. *Content Delivery Network, CDN*). Далее подгруженные части интерфейса динамически встраиваются в его структуру, в результате чего мы получаем единое представление, собранное из различных распределенных источников незаметно для конечного пользователя (рис. 3).

Существуют различные способы встраивания частей интерфейса в общую структуру, например, использование *iframe* тэгов или «ленивая» подгрузка компонентов с заменой соответствующих тэгов-заполнителей.

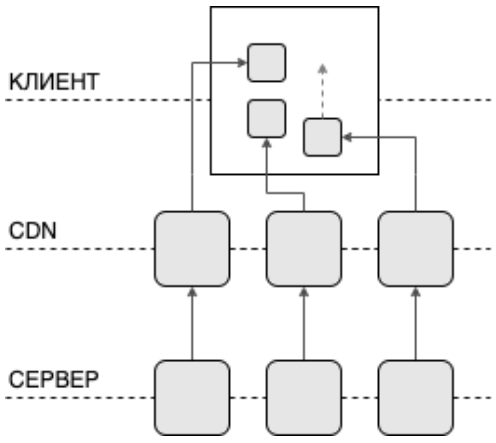

Рисунок 3

2. Компоновка на стыке клиентской и серверной стороны.

При данном подходе итоговое представление интерфейса собирается на уровне *CDN* (рис. 4). Большое количество поставщиков *CDN* услуг предоставляет возможность использования языка разметки *ESI* (англ. *Edge Side Include*), основанного на языке разметки *XML*. Использование языка *ESI* дает возможность для масштабирования инфраструктуры пользовательского интерфейса, но с другой стороны может усложнить процесс смены поставщика *CDN* услуг, а также потенциально может увеличить количество программного кода, требуемого для работоспособности пользовательского интерфейса.

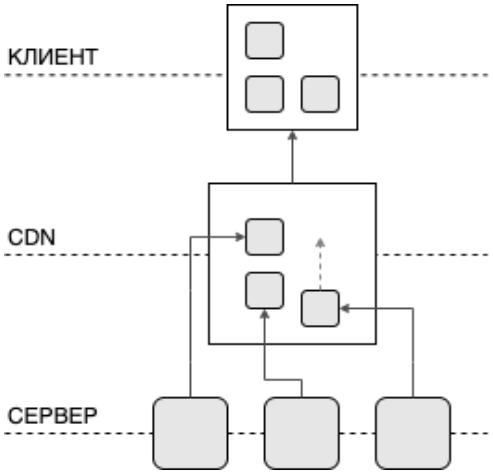

Рисунок 4

3. Компоновка на серверной стороне.

Построение на серверной стороне может происходить как во время исполнения кода, так и во время компиляции. В данном случае сервер выстраивает итоговое представление, собирая все необходимые микро-фронтенды воедино (рис. 5). При необходимости кэширование результата может производиться на уровне *CDN*.

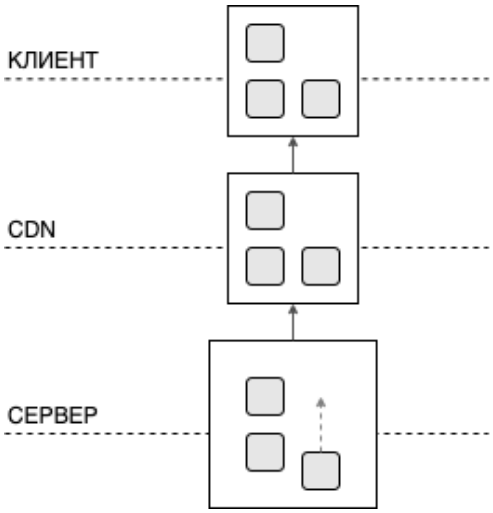

Рисунок 5

Если выбор пал на данный подход к построению интерфейса во время исполнения кода, стоит глубоко оценить возможности и выбрать стратегии масштабирования сервера во избежание длительных простоев и периодов недоступности веб-приложения для пользователей.

Определившись с принципами и подходами к построению архитектуры микро-фронтендов, далее стоит рассмотреть возможности по маршрутизации и коммуникации между-фронтендами, так как без этого пользовательский интерфейс не заработает как единое решение, целостное для конечного пользователя.

Маршрутизация в микросервисной архитектуре пользовательского интерфейса

Решение о том, какую маршрутизацию микро-фронтендов использовать, тесно связано с выбранным подходом к построению архитектуры, описанным выше. Важно принять это решение на самых первых этапах до начала активной фазы разработки.

Мы можем маршрутизировать запросы либо на стороне клиента, либо на уровне *CDN*, либо на сервере.

Выбор в пользу маршрутизации на уровне *CDN* в сочетании с построением представления на стыке клиента и сервера может быть оправдан, если есть необходимость уменьшить риск возникновения проблем нагрузки и масштабирования, которые могут возникнуть на стороне сервера, так как нагрузка может быть сбалансирована посредством *CDN*. С другой стороны, стоит понимать и возможные недостатки: скорость исполнения кода на уровне *CDN* ниже, чем на стороне сервера; внешние конфигурации потенциально более сложные, чем на стороне сервера; итоговое время получения ответа может быть выше за счет дополнительной задержки на уровне *CDN*.

В случае выбора в пользу маршрутизации на стороне клиента ответственность за корректную логику лежит на точке входа. Точка входа должна получить актуальную конфигурацию маршрутов до микро-фронтендов и затем в соответствии с запросом скомпоновать требуемые микро-фронтенды в единый интерфейс. Это самый подходящий вариант для веб-приложений с высокой степенью персонализации представления для каждого пользователя, основанных на аутентификации, активно использующих возможности гео-позиционирования, гибко настраивающихся под нужды пользователя.

Выбор в пользу маршрутизации на стороне сервера оправдан в случае, если интерфейс не требует или не может быть кэширован, если скорость исполнения кода и время отклика важней устойчивости к нагрузке и возможности быстрого масштабирования. А также, если количество или качество ресурсов, доступных для разработки серверной стороны веб-приложения преобладают над ресурсами, доступными для клиентской стороны.

Также стоит отметить, что возможно сочетание вариантов, например, использование для маршрутизации и уровня *CDN* и стороны сервера, либо *CDN* и клиентской стороны одновременно.

Коммуникация между микросервисами пользовательского интерфейса

В самом идеальном случае, в соответствии с изложенными принципами, коммуникации между микро-фронтендами быть не должно. В реальности есть множество ситуаций, когда необходимо уведомить изолированные друг от друга микро-фронтенды о произошедшем событии. Отсюда следует, что необходима

продуманная система коммуникации и уведомления о событиях при соблюдении принципа автономности и изолированности.

Для этих целей возможно использовать в каждом микро-фронтенде общую шину событий. С помощью такой шины микро-фронтенд может отправлять событие в очередь, а подписанные на это событие микро-фронтенды могут считывать события из очереди и реагировать на них должным образом.

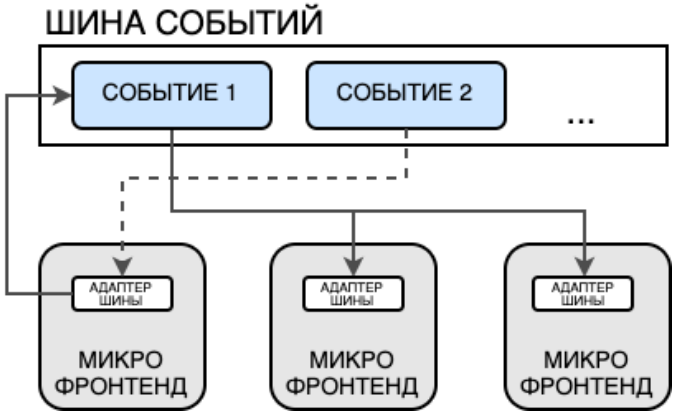

Рисунок 6

В данном случае появляется инфраструктурный компонент веб-приложения, являющийся зависимостью для каждого микро-фронтенда (рис. 6). И важной задачей является обеспечить работоспособность всех зависимых микро-фронтендов даже в случае отказа этого инфраструктурного компонента.

Другой подход для налаживания коммуникации между микро-фронтендами – использовать глобальную область видимости скомпонованного представления для отправки и получения событий. В этом случае пропадает необходимость в дополнительных инфраструктурных компонентах, но снижается прозрачность и контролируемость происходящего в интерфейсе, что несет за собой риски недоступности веб-приложения для пользователей и увеличивает сложность и длительность решения возникающих сбоев (рис. 7).

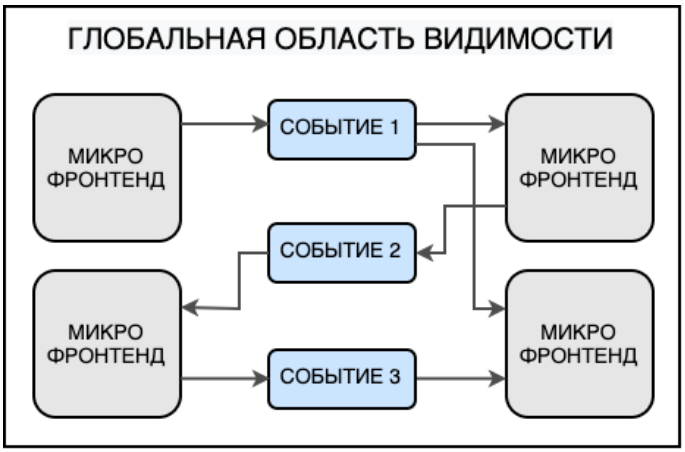

Рисунок 7

Достоинства микросервисной архитектуры пользовательского интерфейса

Одно из явных достоинств использования микро-фронтендов – это возможность выбора технологий каждой командой разработки в отдельности и прозрачное управление зависимостями. Кроме этого, к достоинствам можно причислить:

- очень четко разделенные зоны ответственности как микро-фронтенда, так и команды разработки;
- независимые выдачи: каждый микро-фронтенд может иметь свой жизненный цикл как независимая программа;
- повышение общей стабильности программного обеспечения, так как работоспособность отдельных микро-фронтендов не влияет на другие;
- возможность легко выводить в эксплуатацию обновления, откатывать ошибочные изменения, исправлять найденные ошибки и уязвимости, тем самым повышая гибкость разработки и безопасность программного обеспечения;
- микро-фронтенды как фрагменты интерфейса, инкапсулирующие функциональность одной предметной области, упрощают работу команды разработки, так как нивелируют необходимость всеобъемлющей компетенции в работе всего веб-приложения, что приводит к взаимозаменяемости членов команды и более глубокому пониманию работы и особенностей разрабатываемого микросервиса.

Недостатки микросервисной архитектуры пользовательского интерфейса

Стоит также упомянуть и о возможных недостатках такого подхода, так как при принятии решения о целесообразности использования микросервисов пользовательского интерфейса, данные недостатки могут оказаться решающими и склонить в сторону более классических решений:

- взаимодействие между микро-фронтендами невозможно обеспечить стандартными методами (например, внедрением зависимостей);
- размер клиентской части может быть значительно больше размеров клиентской части, разработанной как монолит в случае, если общие зависимости не вынесены из микро-фронтендов;
- за маршрутизацию в конечном итоге все равно должен отвечать главный микро-фронтенд, такой как точка входа;
- сложность инфраструктуры, обеспечивающей работоспособность и взаимодействие микросервисов на порядки выше, поскольку возникает острая необходимость автоматизации этапов жизненного цикла каждого микро-фронтенда;
- возникает необходимость в квалифицированных специалистах, отвечающих за инфраструктуру; компетенция команд разработки в данной области необходима, а это сужает круг потенциальных кандидатов и удорожает их стоимость.

Заключение

Таким образом, для применения подхода микросервисной архитектуры к разработке пользовательского интерфейса необходимо глубоко проработать

четыре ключевых аспекта: четко смоделировать предметную область для декомпозиции на микро-фронтенды; проработать стратегию компоновки микро-фронтендов в единое представление пользовательского интерфейса; продумать системы маршрутизации и системы коммуникации между микро-фронтендами, наиболее подходящие для данного программного обеспечения.

Несмотря на актуальность и потенциал данного подхода, стоит здраво оценивать и достоинства, и его недостатки, так как от этого зависит итоговая ценность программного обеспечения для конечного пользователя и экономическая оправданность его разработки для организации.

Литература

1. Фаулер М. Архитектура корпоративных программных приложений. – М.: Издательство Вильямс, 2007. – 544 с.
2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – П.: Издательство Питер, 2015. – 368 с.
3. Зяблов Д.В., Кот А.А. Применение микросервисной архитектуры при разработке корпоративных веб-приложений // Студенческий: электрон. научн. Журн, 2017. – № 18 (18).
4. Erl T. Service-Oriented Architecture. Analysis and Design for Services and Microservices.: Издательство Prentice Hall, 2016. – 416 с.
5. Wolff E. Microservices: Flexible Software Architecture.: Издательство Addison-Wesley Professional, – 430 с.
6. Ньюмен С. Создание микросервисов. – П.: Издательство Питер, 2016. – 304 с.
7. Fowler M. Microservices (https://martinfowler.com/articles/microservices.html) - дата обращения - июнь 2020 г.
8. Mezzalira L. Micro-frontends in context (https://increment.com/frontend/micro-frontends-in-context) - дата обращения - июнь 2020 г.
9. Общие архитектуры веб-приложений (https://docs.microsoft.com/ru-ru/dotnet/standard/modern-web-apps-azure-architecture/common-web-application-architectures) - дата обращения - июнь 2020 г.
10. Введение в серверную часть (https://developer.mozilla.org/ru/docs/Learn/Server-side/First_steps/Introduction) - дата обращения - июнь 2020 г.

ИСПОЛЬЗОВАНИЕ ТЕСТОВЫХ ИНФОРМАЦИОННО-ТЕХНИЧЕСКИХ ВОЗДЕЙСТВИЙ ДЛЯ ПРЕВЕНТИВНОГО АУДИТА ЗАЩИЩЕННОСТИ ИНФОРМАЦИОННО-ТЕЛЕКОММУНИКАЦИОННЫХ СЕТЕЙ

Г.Е. Смирнов, *ООО «Корпорация «Интел групп», science.cybersec@yandex.ru;*
С.И. Макаренко, *д.т.н., доцент, Санкт-Петербургский федеральный исследовательский центр РАН, Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» имени В.И. Ульянова (Ленина), mak-serg@yandex.ru.*

УДК 004.7.056.53

Аннотация. В статье рассмотрены информационно-телекоммуникационные сети (ИТКС), в частности ИТКС специального назначения.