DOI 10.37882/2223-2966.2023.8-2.07

ИЗМЕНЕНИЕ ПОДХОДОВ К РАЗРАБОТКЕ ДИНАМИЧЕСКИХ ВЕБ-САЙТОВ

CHANGING APPROACHES TO DYNAMIC WEBSITE DEVELOPMENT

R. Bogdanov
V. Samokhina

Summary. In the modern world, web technologies penetrate almost all aspects of human life. With their help, you can create, debug and synchronize accounts on different devices, manage business processes. At the same time, there is the problem of increasing the level of efficiency in the design of dynamic websites that will provide access to a variety of information from any corner of the world where there is Internet access. More and more dynamic websites appear every day, the power of computer technology is growing, and at the same time their level of complexity is increasing, which in turn leads to an increase in efforts and time to maintain them. This causes the need to justify the use of various approaches to their development based on a monolithic, microservice microfrontend architecture. That is why the paper highlights the advantages and disadvantages of developing dynamic websites using different approaches. The main approaches to building a microfrontend of dynamic websites are substantiated: composition of templates on the server side, integration during assembly, integration during execution using the iframe HTML element, integration during execution using the JavaScript programming language, integration during execution using web-components technology. Based on the analysis of these approaches, the advantages of the microfrontend architecture are identified and characterized: gradual (incremental) updates, simple, unrelated code bases, independent deployment, autonomous teams.

Keywords: development, dynamic websites, microservices, monolithic, microservice, microfrontend, architecture.

Богданов Роман Александрович
Технический институт (филиал) Северо-Восточный федеральный университет имени М.К. Аммосова, г. Нерюнгри
noone214@mail.ru
Самохина Виктория Михайловна
Кандидат педагогических наук, доцент, Технический институт (филиал) Северо-Восточный федеральный университет имени М.К. Аммосова, г. Нерюнгри
vsamokhina@bk.ru

Аннотация. В современном мире веб-технологии проникают почти во все аспекты жизни человека. С их помощью можно создавать, отлаживать и синхронизировать учетные записи на разных устройствах, управлять процессами бизнеса. В то же время возникает проблема повышения уровня эффективности проектирования динамических веб-сайтов, которые будут предоставлять доступ к разнообразной информации из любого уголка мира, где есть доступ в Интернет. Все больше динамических веб-сайтов появляется с каждым днем, мощность компьютерной техники растет, а вместе с этим повышается уровень их сложности, что в свою очередь обуславливает увеличение усилий и времени на их поддержку. Таким образом, создается потребность в обосновании использования различных подходов к их разработке на основе монолитной, микросервисной, микрофронтендной архитектуре. Именно поэтому в работе отмечены преимущества и недостатки разработки динамических веб сайтов с использованием различных подходов. Обоснованы основные подходы к построению микрофронтенда динамических веб сайтов: композиция шаблонов на стороне сервера, интеграция при сборке, интеграция при выполнении с использованием HTML-элемента iframe, интеграция при выполнении с помощью средств языка программирования JavaScript, интеграция при выполнении с использованием технологии web-components. На основе анализа указанных подходов выделены и охарактеризованы преимущества микрофронтендной архитектуры: постепенные (инкрементальные) обновления, простые, не связанные между собой кодовые базы, независимое развертывание, автономные команды.

Ключевые слова: разработка, динамические веб сайты, микросервисы, монолитная, микросервисная, микрофронтендная, архитектура.

Введение

С каждым днем появляется все больше интернет веб-сайтов, мощность компьютерной техники растет, а вместе с тем повышается уровень их сложности, что, в свою очередь, обуславливает увеличение усилий и времени на их поддержку, в частности: обновлять содержательное наполнение веб-сайтов и технологий, которые используются при разработке для увеличения быстродействия приложений, уменьшать время на расчеты, улучшать удобство работы с приложениями и т.д.

С ростом размера динамических веб-сайтов увеличивается количество времени, необходимого для того, чтобы новый сотрудник разобрался в их архитектуре и начал разрабатывать новый функционал системы. В то же время растет вероятность внесения дефектов в разных частях системы при разработке нового функционала динамических веб-сайтов [1]. Поэтому важен внимательный и тщательный подход к выбору архитектуры динамических веб-сайтов, что позволит определить дальнейшее развитие всего проекта. Построение больших и сложных динамических веб-сайтов является непростой задачей, для решения которой применяется

как монолитный, так и распределенный подходы в зависимости от контекста и требований [2].

Анализ последних исследований и публикаций говорит про значительное количество работ в направлении тематики различных подходов к разработке динамических веб-сайтов. В частности, М. Фаулер, С. Ньюмен, И. Надерешвили, И. Ивентс, М. Бауман, Л. Цзин, С. Невман, А. Кумар, Р. Купер и др. в своих работах освещают разные варианты построения динамических веб-сайтов, базирующихся на разных типах архитектуре их создания.

Цель работы — проанализировать изменение подходов к разработке динамических веб-сайтов.

Основная часть

Проектирование и разработка качественных динамических веб-сайтов достаточно важны в современном информационном мире. В ходе развития отрасли информационных технологий обоснован ряд подходов и концепций построения сложных программных систем в виде динамических веб-сайтов [3].

Долгое время «монолитная архитектура» занимала ключевое место при построении как серверных, так и клиентских динамических веб-сайтов. Система, использующая указанный подход, представляет собой монолит, размещенный на одном сервере, запущенный в одном процессе и выполняющий при этом всю бизнес-логику системы. Монолит на стороне сервера подвергается только горизонтальному масштабированию, достигаемому за счет запуска большого количества отдельных серверов с каждым отдельным монолитом. Со временем появлялись другие подходы к разработке серверных динамических веб-сайтов, среди которых сервис-ориентированная архитектура (SOA), которая, в отличие от монолита, является распределенной системой, которая обменивается сообщениями, используя определенные протоколы [4].

С развитием индустрии информационных технологий разработан новый подход к организации динамических веб-сайтов — микросервисная архитектура (MSA). Ее можно считать частью SOA, которая имеет определенные отличия от классической SOA. Основное отличие — это незначительное количество кода, в то время как в SOA объем кодовой базы не имеет значения.

Также важной особенностью MSA является то, что каждый микросервис динамических веб-сайтов ограничен своим контекстом — бизнес-задачей. Ограничений по количеству сервисов нет; важно, чтобы каждый из них работал только над своей бизнес-задачей. Для межсервисной коммуникации используют стандартные протоколы передачи данных (например, HTTP) и в то же время сервис может иметь свой протокол для общения с другими.

Среди преимуществ микросервисной архитектуры динамических веб-сайтов можно выделить [5]:

— низкую связность между компонентами системы;
— относительно простое развертывание: каждый сервис разворачивается независимо от других, в то время как при использовании монолитной архитектуры вся система разворачивается вместе в одном процессе и невозможны внесения динамических изменений;
— простое масштабирование системы, как в горизонтальном, так и в вертикальном направлении;
— относительная отказоустойчивость: при недееспособности отдельных сервисов система остается работоспособной;
— применение различных технологий, фреймворков и языков программирования, а также разных типов хранилищ данных;
— при создании сервисов возможна организация небольших групп программистов, ответственных за разработку сервиса.

Среди недостатков микросервисного подхода создания динамических веб-сайтов можно выделить следующие [5]:

— сложность разработки и замены членов команды, прямо пропорционально зависит от количества языков программирования, фреймворков и типов баз данных, использованных при разработке;
— потеря ресурсов во время межсервисной коммуникации, что приводит к увеличению времени на обработку, сериализацию и десериализацию сообщений;
— проблемы с версионированием и поддержкой обратной совместимости между сервисами;
— значительно комплекснее и сложнее по сравнению с монолитом интеграционное тестирование.

Чаще всего разработку динамических веб-сайтов, в которых был успешно использован микросервисный подход, начинали с монолита, который со временем увеличивался в объеме. В то же время, бывает и такое, что проект начинается с микросервисного подхода и не достигает цели. Поэтому лучше начать с разработки монолита, даже если есть уверенность, что проект будет иметь значительный объем. Микросервисы являются достаточно эффективным архитектурным приемом, но их преимущества раскрываются только при использовании в больших и сложных системах.

Для разработки динамических веб-сайтов необходимо, прежде всего, убедиться в их целесообразности, полезности и спросе среди пользователей. В начале разработки максимальную ценность имеет скорость

разработки, а значит и скорость получения отзывов от пользователей. Следующая проблема разработки микросервисных динамических веб-сайтов состоит в том, что нужно получить набор ограниченных контекстов, а любые переработки и рефакторинг значительно сложнее при использовании микросервисов по сравнению с монолитом. Построение монолита позволяет легко разделить его на логические части, выявить слабые места и недостатки системы, выделить часто используемые фрагменты кода в собственные библиотеки и т.д.

Сейчас существует значительное количество путей реализации стратегии «сначала монолит». Наиболее логичным является проектирование монолита с четкой модульной структурой. При правильной реализации переход к микросервисам будет достаточно обычным.

Более общим считается написание монолита с постепенным отделением от него сервисов. В таком случае значительная часть монолита останется как первоначальное «ядро» системы, а большая часть новой разработки будет проходить в сервисах. Также в основном используют полную замену монолита на микросервисы. Среди недостатков этого подхода является необходимость остановки разработки нового функционала, что чаще всего неприемлемо для бизнеса.

За последние годы популярность микросервисов стремительно растет, и многие организации использовали этот архитектурный стиль во избежание ограничений крупных монолитов.

При разработке микросервисных динамических веб-сайтов чаще всего возникают проблемы с такими аспектами [6]:

— безопасной интеграцией нового кода в существующее приложение;
— использованием новых функций языка JavaScript или другого из множества языков, которые можно компилировать в JavaScript и сопутствующими проблемами со сборником приложения;
— масштабированием разработки в целях распределения частей приложения между командами.

Эти проблемы могут негативно отразиться на способности эффективно доставлять высококачественный продукт своим клиентам. Именно поэтому в последнее время уделяется все больше внимания всеобщей архитектуре и организации кодовой базы, необходимой для сложной, современной веб-разработки динамических веб-сайтов с помощью микрофонтенда. Микрофронтенды — это архитектурный стиль динамических веб-сайтов, когда независимые приложения для интерфейса складываются в единое целое, разрезание крупных монолитов на более мелкие, более управляемые куски, а затем — явное понимание зависимостей между ними. Выбор технологий, кодовые базы, команды и процессы выпуска должны иметь возможность работать и развиваться независимо друг от друга без лишней координации.

Существует множество подходов, которые можно назвать микрофронтендовыми. Выделяют пять основных подходов к построению микрофронтенда динамических веб-сайтов [6]:

— композиция шаблонов на стороне сервера (server side composition);
— интеграция при сборке;
— интеграция при выполнении (runtime integration) с использованием HTML-элемента iframe;
— интеграция при выполнении (runtime integration) с помощью средств языка программирования JavaScript;
— интеграция при выполнении (runtime integration) с использованием технологии web components.

При этом используется index.html, содержащий любые общие элементы страницы, а затем сервер составляет содержимое страницы из фрагментов HTML-файлов:

Это достаточно стандартная композиция на стороне сервера. Такой подход относится к микрофронтендам, так как разделив код таким образом, каждый фрагмент является самостоятельной частью бизнес-логики, которую может поставить независимая команда. Здесь не показано, как эти разные HTML-файлы попадают на вебсервер, но предположение состоит в том, что каждый из них имеет свой конвейер развертывания, позволяющий размещать изменения на одной странице, без использования какой-либо другой страницы. Для еще большей независимости можно использовать отдельный сервер, отвечающий за предоставление и обслуживание каждого микрофронтенда.

При интеграции при сборке каждый микрофронтенд публикуется как отдельный пакет, а контейнер учитывает пакеты как отдельные зависимости. На первый взгляд, такой подход имеет смысл. Создается единый пакет JavaScript, позволяющий избежать дублирования кодовой базы и посторонних зависимостей. Однако такой подход означает, что придется перекомпилировать и выпустить каждый микрофронтенд, чтобы внести изменения в любую отдельную часть продукта, влияющую на время, затрачиваемое на сборку.

Интеграция с использованием iFrame — один из самых простых подходов к составлению программ в браузере. По своей природе iFrame облегчают создание страницы с независимых подстраниц. Они также предлагают хороший уровень изоляции с точки зрения стилизации и глобальных переменных за счет того, что не мешают друг другу. Этот подход достаточно непопулярен. Упо-

мянутая выше простая изоляция, как правило, делает их менее гибкими, чем другие варианты. Построить интеграцию между разными частями программы может быть трудно, они делают значительно сложнее маршрутизацию, взаимодействие с историей браузера и глубокую маршрутизацию.

Runtime integration с использованием web components состоит в том, что каждый микрофронтенд является кастомным HTML-элементом в соответствии со стандартом web components, а не глобальной функцией в window.

Конечный результат здесь похож на предыдущий пример, главное отличие состоит в том, что решили делать вебкомпонент. Стоит отметить, что технология web components имеет достаточно низкую поддержку среди популярных браузеров, что является значительным недостатком.

Сегодня появляются схемы разложения монолитов фронтенда на более мелкие, простые куски — микрофронтенды, которые можно разработать, протестировать и развернуть самостоятельно и которые при этом отражаются для клиентов как единственный продукт.

Некоторые реализации микрофронтендов могут привести к дублированию зависимостей, увеличивая количество пользовательского трафика. Кроме того, резкое повышение автономности команды может вызвать раздробленность в работе ваших команд. Однако считаем, что этими рисками можно управлять и преимущества микрофронтендов часто превышают затраты.

Выделим преимущества архитектуры микрофронтендов для динамических веб-сайтов [1–3]:

1. Постепенные (инкрементальные) обновления. Основным преимуществом является то, что появляется больше пространства для принятия конкретных решений по отдельным частям продукта, совершенствованию архитектуры и зависимостей, экспериментов с новыми технологиями и способами взаимодействия.
2. Простые, не связанные между собой кодовые базы. Исходный код для каждого отдельного микрофронтенда по определению будет значительно меньше исходного кода сплошного монолита. Эти меньшие кодовые базы, как правило, более просты для работы разработчиков. В частности, можно избежать проблем, возникающих вследствие непреднамеренной и ненадлежащей связи между компонентами, не имеющими влияния друг на друга. Микрофронтенды побуждают разработчиков быть четкими и обдуманными по поводу того, как разные части программы взаимодействуют с данными и реагируют на события.
3. Независимое развертывание. Как и в случае с микросервисами, независимое развертывание является ключевым преимуществом. Оно ускоряет время развертывания, что, в свою очередь, уменьшает связанные с этим риски. Каждый микрофронтенд должен развертываться вне зависимости от состояния кодовых баз других компонентов. Если отдельный микрофронтенд готов быть развернутым на live серверах, он должен это сделать, и это решение зависит только от команды, которая его проектирует и поддерживает.

Отметим недостатки архитектуры микрофронтендов.

1. Значительный объем загружаемого кода. Существует прямая взаимосвязь между производительностью страницы и привлечением/конверсией пользователей, и значительная часть мира работает с интернет-подключением гораздо медленнее, чем привыкли в высокоразвитых городах, поэтому ряд причин заботиться о размерах загрузок. Этот вопрос решить непросто. Достаточно сложно найти баланс между желанием разрешить командам самостоятельно разрабатывать свои части приложения, чтобы они могли работать автономно, и желанием строить программы таким образом, чтобы они могли делиться общими зависимостями. Один из подходов заключается во внешнем применении общих зависимостей от составных пакетов. Недостатком такого подхода является то, что все части приложения должны использовать точные версии этих зависимостей. При необходимости изменить версию зависимости потребуются большие скоординированные усилия по обновлению и блокировке разработки нового функционала. Это все, что нужно пытаться избегать, в первую очередь, благодаря микрофронтендам. Однако даже если будет решено не делать ничего по поводу дублирования зависимостей, возможно, каждая отдельная страница все равно будет загружаться быстрее, чем при использовании монолитного приложения. В классических монолитах, когда загружается любая страница программы, загружается исходный код и зависимости каждой страницы сразу, а используя архитектуру микрофронтендов, будет загружаться только код, который необходим для работы этой страницы. Это может привести к более быстрой начальной загрузке страниц, но более медленной следующей навигации, поскольку пользователи вынуждены повторно загружать одинаковые зависимости на каждой странице. Если не перекрывать микрофронтенды ненужными зависимостями, или если пользователи будут соблюдать только одну или две страницы в программе, можно добиться высокого уровня производительности, даже при дублировании зависимостей.

2. Различия среды. Каждый микрофронтенд должен разрабатываться без учета других. Возможно, даже возникновение необходимости запустить его в автономном режиме на отдельной странице, а не внутри программы-контейнера. Это может значительно упростить разработку, особенно, когда программа-контейнер является сложной и устаревшей, что является довольно распространенным при миграции из монолита. Однако существуют риски, связанные с разработкой в среде, содержащей значительные различия по сравнению с live-средой. Если контейнер для разработки ведет себя иначе, чем часть программы, в которую будет интегрироваться разрабатываемый код, то возможно выявление нарушений или изменения в функционировании при развертывании на live-серверах. Особенно опасными могут быть глобальные стили, вызываемые контейнером или другими микрофронтендами. Решением этой проблемы сродни ситуации, когда особенности среды имеют значение. Необходимо обеспечить регулярную интеграцию и развертывание микрофронтендов в тестовые среды и внедрять на этих средах процесс ручного и автоматизированного тестирования с целью скорейшего выявления интеграционных проблем.
3. Сложность управления. Использование более распределенной архитектуры микрофронтендов неизбежно приводит к увеличению количества элементов, которыми необходимо управлять — больше репозитариев и, как следствие, — больше инструментов, доменов и серверов, сборки и развертывания конвейеров.

Был проведен анализ статистики по применению различных подходов в процессах разработки динамических веб-сайтов, результаты которого представлены на рис. 1.

При подходе к решению архитектурных задач в разработке динамических веб-сайтов существует большое количество эффективных приемов, поэтому важен внимательный и тщательный подход к выбору архитектуры системы, поскольку это позволяет определить дальнейшее развитие всего проекта. Построение больших и сложных систем динамических веб-сайтов представляет собой непростую задачу, для решения которой применяется как монолитный, так и распределенный подходы в зависимости от контекста и бизнес-требований.

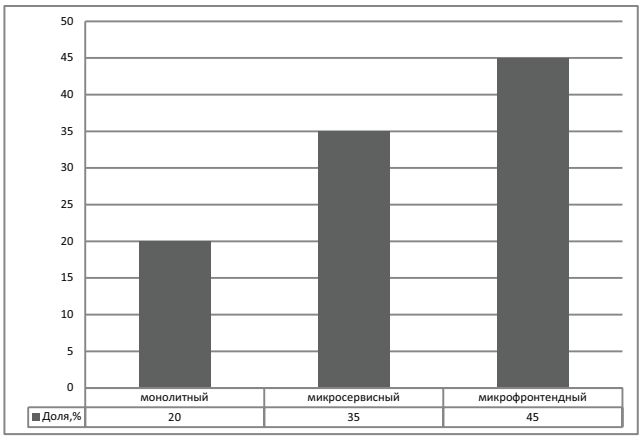

Рис. 1. Результаты анализа статистики по применению различных подходов в процессах разработки динамических веб-сайтов

Выводы

Существует несколько вариантов построения приложений, основанных на архитектуре микрофронтендов. Анализ преимуществ и недостатков каждого из них позволяет выбрать вариант, максимально отвечающий поставленным задачам. Перспективы дальнейших исследований видим в моделировании архитектуры модулей монолитного приложения в микросервисный. Выделенные преимущества и недостатки каждого варианта построения динамических веб-сайтов позволяет выбрать вариант, который будет лучше всего отвечать поставленным задачам, выделить ключевые компоненты системы и сформировать требования к ней.

ЛИТЕРАТУРА

1. Kuepper R. Hands-On Swift 5 Microservices Development: Build microservices for mobile and web applications using Swift 5 and Vapor 4 / R. Kuepper. — UK: Pack Publishing Ltd, 2020. — 362 p.
2. Kumar A. Micro Frontends Architecture: Introduction, Design, Techniques & Technology / A. Kumar. — USA: Kdp Print Us, 2019. — 124 p.
3. Microservice Architecture: Aligning Principles, Practices, and Culture / I. Nadareishvili, R. Mitra, M. McLarty, M. Amundsen. — O'Reilly Media, 2016 — 146 p.
4. Володченко В.С., Ланцова Д.С., Миронова Т.А., Сапунова Е.В., Бышок К.А. Разработка web-сайта // Достижения науки и образования. 2019. №12 (53). URL: https://cyberleninka.ru/article/n/razrabotka-web-sayta (дата обращения: 03.05.2023).
5. Гусев С.С., Макаров В.В. Система управления сайтом с модульной структурой и поддержкой мультиязычности и кеширования на технологиях XML и AJAX // Инженерно-строительный вестник Прикаспия. 2022. №1 (39). URL: https://cyberleninka.ru/article/n/sistema-upravleniya-saytom-s-modulnoy-strukturoy-i-podderzhkoy-multiyazychnosti-i-keshirovaniya-na-tehnologiyah-xml-i-ajax (дата обращения: 03.05.2023).
6. Ахмеджанова Заррина, Гафурова Парвина Применение html и css для создания интерактивных Веб сайтов // Евразийский Союз Ученых. 2019. №4-3 (61). URL: https://cyberleninka.ru/article/n/primenenie-html-i-css-dlya-sozdaniya-interaktivnyh-veb-saytov (дата обращения: 03.05.2023).

Журнал «Современная наука: актуальные проблемы теории и практики»