1.Обоснование выбранной архитектуры веб-приложения

Архитектура Веб-приложения

Основные критерии построения надежной архитектуры веб-приложения:

• Эффективность;

• Гибкость;

• Легкость в тестировании;

• Последовательность и успешность в решении задач;

• Хорошо структурированный и понятный код;

• Масштабируемость в процессе разработки;

• Быстрое время отклика;

• Простота;

• Опора на опробованные стандарты безопасности.

Модель компонентов веб-приложения: один веб-сервер, одна база данных.

Веб-приложение является монолитным.

Монолитная архитектура — это традиционная модель программного обеспечения, которая представляет собой единый модуль, работающий автономно и независимо от других приложений.

Обычно монолитное приложение состоит из базы данных, клиентского пользовательского интерфейса и серверного приложения. Все части программного обеспечения унифицированы, и все его функции управляются в одном месте.

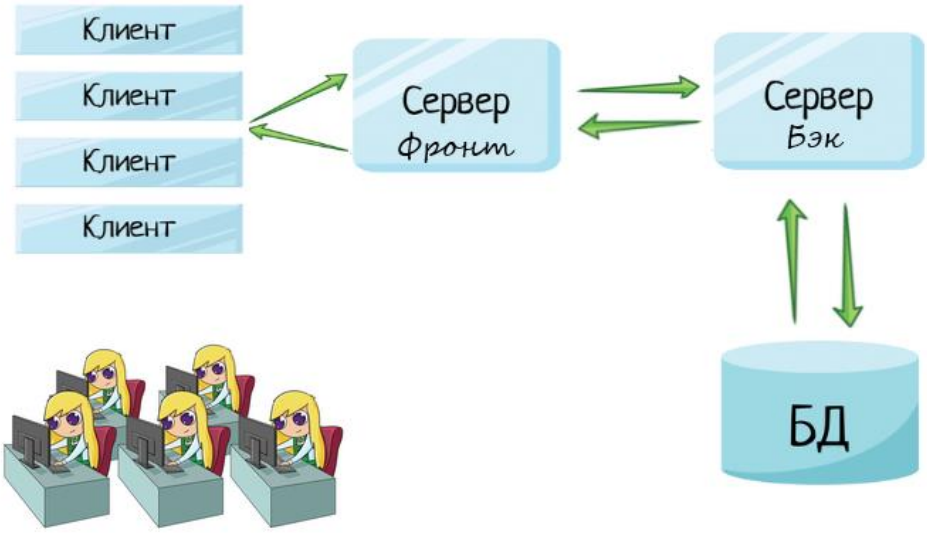
При использовании монолитной архитектуры удобно создавать приложения на основе одной базы кода, поэтому ее основное преимущество заключается в быстроте разработки.

Монолитная архитектура удобна для работы небольших групп, поэтому наша команда выбрала этот подход при создании приложения.

Компоненты монолитного программного обеспечения взаимосвязаны и взаимозависимы, что помогает программному обеспечению быть самодостаточным.

Эта архитектура является традиционным решением для создания приложений, но некоторые разработчики считают ее устаревшей. Тем не менее, мы считаем, что монолитная архитектура является идеальным решением при некоторых обстоятельствах.

Веб-приложение построена на следующей цепочке взаимодействия:



Основные преимущества выбранной архитектуры:

1. Упрощенная разработка и развертывание

Есть много инструментов, которые можно интегрировать для облегчения разработки. Кроме того, все действия выполняются с одним каталогом, что упрощает развертывание. Благодаря монолитному ядру разработчикам не нужно развертывать изменения или обновления по отдельности, поскольку они могут сделать это сразу и сэкономить много времени.

Вывод: Использование одного исполняемого файла или каталога упрощает развертывание.

2. Меньше сквозных проблем

Большинство приложений зависят от множества межкомпонентных задач, таких как контрольные журналы, ведение логов, ограничение скорости и т. д. Монолитные приложения гораздо легче учитывают эти вопросы благодаря своей единой кодовой базе. К этим задачам проще подключать компоненты, когда все работает в одном приложении.

Вывод: Приложение легче разрабатывать, когда оно создано с использованием одной базы кода.

3. Лучшая производительность

При правильной сборке монолитные приложения обычно более производительны, чем приложения на основе микросервисов. Например, приложению с микросервисной архитектурой может потребоваться выполнить 40 вызовов API для 40 различных

микросервисов чтобы загрузить каждый экран, что, очевидно, приводит к снижению производительности. Монолитные приложения, в свою очередь, обеспечивают более быструю связь между программными компонентами благодаря общему коду и памяти.

Вывод: В централизованной базе кода и репозитории один интерфейс API часто может выполнять ту функцию, которую при работе с микросервисами выполняют многочисленные API.

4. Упрощенное тестирование**.**

Монолитное приложение представляет собой единый централизованный модуль, поэтому сквозное тестирование можно проводить быстрее, чем при использовании распределенного приложения.

5. Удобная отладка**.**

Весь код находится в одном месте, благодаря чему становится легче выполнять запросы и находить проблемы.

Конечно, использование монолитной архитектуры имеет и определенное количество минусов:

Минусы монолитной архитектуры

1. Кодовая база со временем становится громоздкой

С течением времени большинство продуктов продолжают разрабатываться и увеличиваются в объеме, а их структура становится размытой. Кодовая база начинает выглядеть действительно громоздко и становится трудной для понимания и изменения, особенно для новых разработчиков. Также становится все труднее находить побочные эффекты и зависимости. С ростом кодовой базы ухудшается качество и перегружается IDE.

2. Сложно внедрять новые технологии

Если в наше приложение необходимо добавить какую-то новую технологию, разработчики могут столкнуться с препятствиями для на пути внедрения. Добавление новой технологии означает переписывание всего приложения, что является дорогостоящим и требует много времени.

3. Ограниченная гибкость

В монолитных приложениях каждое небольшое обновление требует полного повторного развертывания. Таким образом, все разработчики должны ждать, пока это не будет сделано. Когда несколько команд работают над одним проектом, гибкость может быть значительно снижена.

В итоге

Монолитная модель не устарела, и в некоторых случаях она по-прежнему прекрасно работает. Некоторые гигантские компании, такие как Etsy, остаются монолитными, несмотря на сегодняшнюю популярность микросервисов.

Архитектура монолитного программного обеспечения может быть полезной, если команда находится на начальной стадии, особенно, когда создаем непроверенный продукт и не имеем опыта работы с микросервисами.

Монолит идеально подходит для стартапов, которым необходимо как можно быстрее запустить продукт в эксплуатацию. Однако некоторые проблемы, упомянутые выше, идут рука об руку с монолитной архитектурой.

Так же нами была использована архитектура одностраничных приложений.

Тип архитектуры веб-приложения: Архитектура одностраничных приложений.

SPA - это веб-приложение, которое загружает всю необходимую информацию при входе на страницу.

Благодаря SPA (одностраничное приложение) достигается плавная работа приложения, интуитивно понятное и интерактивное взаимодействие с пользователем.

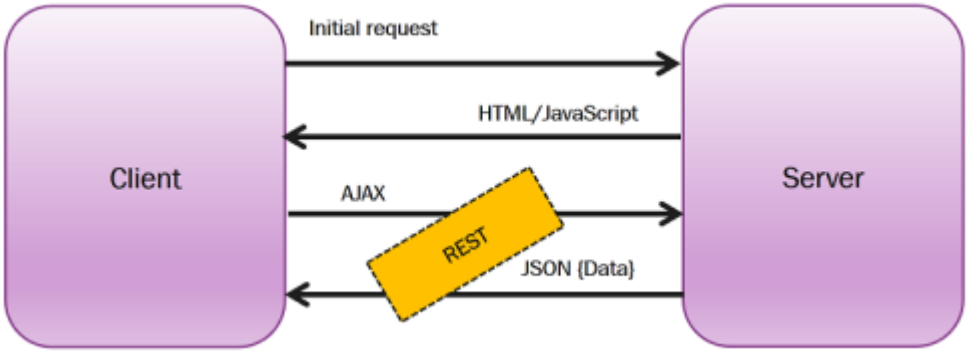
Одностраничные приложения имеют одно существенное преимущество - они обеспечивают потрясающий пользовательский интерфейс, поскольку пользователи не испытывают перезагрузки веб-страниц.

Вместо загрузки новой страницы SPA загружают одну веб-страницу и перезагружают запрошенные данные на той же странице с динамически обновляемым содержимым. Остальная часть веб-страницы остается нетронутой.

Одностраничные приложения хороши тем, что они требуют меньше ресурсов для веб-сервера.

Разработка одностраничного приложения - это лучшая идея, чтобы привлечь пользователей и привести их в единое веб-пространство с помощью простой архитектуры.

SPA разрабатываются на стороне клиента с использованием фреймворков JavaScript, так как вся логика всегда переносится на фронтенд.



Процесс рендеринга на стороне клиента выглядит следующим образом:

• Веб-браузер получает запрос от клиента и отправляет его на сервер в формате HTML.

• Сервер получает запрос клиента и отвечает HTML-файлом, который включает текст, ссылки для изображений, CSS и JavaScript.

• Когда сервер запрашивает HTML-файл, пользователь видит загружающиеся изображения или пустую страницу при выполнении JavaScript.

• Тем временем одностраничное приложение получает данные, создает представления и помещает их в Data Object Model(DOM).

• Наконец, веб-приложение готово к использованию.

Преимущества выбранной архитектуры:

- Обеспечение лучшего пользовательского опыта

Обеспечение лучшего пользовательского опыта - это главное преимущество одностраничного приложения, которое делает его популярным среди пользователей.

Успех веб-приложений зависит от лучшего пользовательского опыта. Чем проще веб-приложение, тем больше трафика будет на нем.

Исследования утверждают, что пользователи остаются на веб-страницах благодаря простоте навигации по контенту.

В результате одностраничные приложения прошли долгий путь, поскольку они не перезагружают контент. Эти веб-приложения обеспечивают лучший пользовательский опыт за счет обновления нескольких частей контента вместо повторной загрузки всего содержимого.

В результате SPA помогают компаниям повысить конверсию благодаря более быстрой загрузке страниц.

- Оптимизация скорости

Веб-приложение, которое загружается медленнее, увеличивает показатель отказов и вредит поисковой оптимизации (SEO). П

Пользователи ожидают, что веб-страница будет загружаться быстрее, и не хотят тратить свое время в этом быстро развивающемся цифровом мире. Поэтому веб-приложения должны оптимизировать скорость загрузки страниц для повышения вовлеченности пользователей.

В этом отношении одностраничные приложения предлагают оптимизацию скорости, поскольку они не загружают весь контент заново. Вместо этого эти настольные и

мобильные приложения обновляют несколько фрагментов контента для оптимизации скорости.

- Простой процесс разработки

Разработка одностраничного приложения требует меньше ресурсов и снижает накладные расходы команд разработчиков.

Так, процесс разработки SPA прост, поскольку командам разработчиков не нужно одновременно писать код и серверный рендеринг. Вместо этого разработчики, участвующие в процессе разработки, могут разделить бэк-энд для создания фронт-энда. Таким образом, front-end и back-end разработчики могут работать независимо друг от друга, не путаясь.

Таким образом, back-end-разработчики будут уделять внимание подкрепленным технологиям, таким как back-end API. В то время как front-end разработчики будут создавать и внедрять front-end, не беспокоясь об API back-end.

Таким образом, разработка одностраничных мобильных или настольных приложений снижает накладные расходы команд разработчиков и упрощает процесс разработки по сравнению с многостраничными приложениями.

- Эффективный процесс кэширования

Помимо скорости и лучшего пользовательского опыта, одностраничное приложение также предлагает эффективный процесс кэширования, чем многостраничное приложение.

Причина в том, что одностраничное мобильное приложение отправляет запрос только один раз и сохраняет эти данные для дальнейшего использования.

Таким образом, преимущество эффективного кэширования заключается в том, что пользователи могут получить доступ к странице, даже если у них низкая пропускная способность интернета. Пользователи по-прежнему могут получить доступ к настольным приложениям, поскольку они синхронизируются с сервером при наличии стабильного интернет-соединения.

- Простая отладка

Обнаружение и устранение ошибок играет важную роль в использовании оптимальных возможностей приложения. Поскольку одностраничные приложения используют популярные фреймворки SPA, такие как React, Angular, или JavaScript-фреймворки, отлаживать такие приложения проще, чем традиционные веб-страницы. Благодаря использованию популярных фреймворков, вы можете легко отслеживать и обнаруживать ошибки в данных и элементах страницы.

Более того, SPA обеспечивают более легкую отладку, чем многостраничные приложения, поскольку имеют инструменты разработчика для поисковых систем, таких как

Google Chrome. Поэтому разработчики могут отлаживать ошибки, просматривая код JavaScript в браузере, вместо того, чтобы просматривать множество строк кода.

- Снижение нагрузки на сервер

Одностраничные приложения снижают нагрузку на сервер, поскольку не заставляют его выполнять многократную визуализацию.

Таким образом, SPA могут полагаться на несколько серверов для управления тем же трафиком, чем приобретение дополнительных серверов.

Можете ли вы представить себе работу с несколькими серверами в многостраничном приложении? Нет. Причина в том, что для управления традиционным многостраничным приложением нужны дополнительные средства на ресурсы.

2. Обоснование выбранных технологий для разработки

- Серверная часть - бэкэнд, управляет бизнес-логикой и отвечает на HTTP-запросы.

Серверный код написан на Java (spring).

Java — один из наиболее используемых языков программирования. Он активно применяется в разработке корпоративных, веб- и мобильных приложений.

Одна из основных причин использовать Java для веба состоит в том, что это высокопроизводительный язык программирования. Он гарантирует, что приложения смогут работать быстро и без каких-либо ошибок обработки. Существуют также и другие причины:

1. Открытый исходный код

Являясь языком программирования с открытым исходным кодом, Java имеет ряд преимуществ. Во-первых, он снижает затраты на процесс разработки приложений. Во-вторых, разработчики легко изменяют язык и часто обновляют его, поскольку он имеет открытый код. Кроме того, Java обладает простым для чтения синтаксисом, который упрощает разработку приложений для интернета и мобильных устройств. И наконец, разработчики могут использовать имеющуюся кодовую базу языка и улучшать ее.

2. Кроссплатформенность

Еще одним преимуществом программирования на Java является то, что это кроссплатформенный язык. Разработчики могут писать код в Windows и запускать его в macOS и Linux. Здесь действует принцип “написал один раз, запускай везде”. Это упрощает работу для разработчиков, работающих на разных системах, и упрощает процесс тестирования на разных машинах. Например, разработчики могут проверить, будет ли программа правильно работать на разных размерах экрана и операционных системах.

3. Инструменты и библиотеки

Одним из самых значительных преимуществ Java является его совместимость с различными инструментами. Такие фреймворки, как Spring, Hibernate, Struts, Spark и другие значительно упрощают процесс разработки. Все эти инструменты предлагают различные функции для создания интерактивных и динамических приложений. Такие библиотеки, как Apache Commons, стандартные библиотеки Java, Maven, Jackson и т. д. позволяют разработчикам добавлять функциональные возможности без написания кода с нуля.

Это отличный и мощный язык программирования для использования на предприятиях, которым сейчас нужна облачная архитектура.

В сочетании с JavaScript Java позволяет создавать высокопроизводительные веб-приложения, которые можно запускать на любой платформе.

- Сторона клиента - интерфейс, именно здесь происходит взаимодействие с пользователем.

  Код написан на JavaScript (react) и хранится в браузере.

Facebook представил фреймворк React в 2013 году. Это JavaScript-фреймворк, широко используемый в известных одностраничных приложениях, таких как Facebook, Instagram, Uber и WhatsApp. Среди всех других фреймворков, react имеет тысячи вкладов на GitHub, которые могут помочь разработчикам узнать последние тенденции и решить проблемы, с которыми они сталкиваются в процессе разработки. Это легкий и простой в тестировании фреймворк, наиболее широко используемый разработчиками.

React - лучший выбор для разработчиков, только начинающих фронт-енд архитектуру одностраничных приложений. Более того, этот фреймворк легко интегрируется с другими фреймворками и технологиями, что может помочь вам при работе над крупномасштабным проектом.

- Сервер базы данных, который отправляет запрошенные данные на сторону сервера.

  Используется PostgreSQL.

  PostgreSQL — бесплатная система управления базами данных.

  1. Свободный доступ

Любой специалист может бесплатно скачать, установить СУБД и сразу начать работу с базами данных.

2. Можно установить на любую платформу

PostgreSQL подходит для работы в любой операционной системе: Linux, macOS, Windows. Пользователь получает систему «из коробки» — чтобы установить и использовать программу, не нужны дополнительные инструменты.

3. Поддерживает разные форматы данных

PostgreSQL поддерживает много разных типов и структур данных, в том числе сетевые адреса, данные в текстовом формате JSON и геометрические данные для координат геопозиций. Все эти форматы можно хранить и обрабатывать в СУБД.

В PostgreSQL можно создавать собственные типы данных, их называют пользовательскими. Пользовательские типы данных нужны, чтобы упростить работу с базой или установить ограничения.

4. Позволяет работать с большими размерами данных

Размер базы данных в PostgreSQL не ограничен и зависит от того, сколько свободной памяти есть в месте хранения: на сервере, локальном компьютере или в облаке.

Максимальный размер таблицы — 32 терабайта. Этого более чем достаточно для хранения данных компаний типа Amazon. Одна строка в базе данных не может превышать 1,6 терабайт, а максимальный размер одной ячейки — 1 гигабайт. В такую ячейку можно добавить даже видео.

PostgreSQL позволяет работать с базами данных и элементами в них таких размеров, которые на практике в большинстве случаев не нужны. Поэтому эти ограничения можно назвать условными.

5. Соответствует требованиям ACID

Аббревиатура ACID расшифровывается так:

- атомарность (от англ. atomicity),
- согласованность (от англ. consistency),
- изолированность (от англ. isolation),
- устойчивость (от англ. durability).

Это четыре требования для надёжной работы систем, которые обрабатывают данные в режиме реального времени. Если все требования выполняются, данные не будут теряться из-за технических ошибок или сбоев в работе оборудования.

6. Поддерживает все функции, которые есть в современных базах

Например, в PostgreSQL есть оконные функции, вложенные транзакции и триггеры.

Оконные функции позволяют выбрать определённые записи в таблице и делать вычисления с ними в отдельном столбце. Например, можно добавить в таблицу с данными интернет-магазина столбец с датой первого посещения пользователем сайта. Этот столбец пригодится, если понадобится рассчитать LTV (от англ. customer lifetime value).

Вложенными называют транзакции внутри других. Например, выполнение серии переводов траншами в рамках одного договора. Допустим, были выполнены пять транзакций, а на шестой возникли проблемы. Откатиться должны все предыдущие транзакции, которые были внутри одной большой.

В PostgreSQL можно создавать триггеры — функции, которые автоматически запускаются при определённых условиях. Например, можно создать триггер, который запускается при удалении данных о закрытой компании из базы. Созданный триггер автоматически добавит в нужном поле другой таблицы запись: «данные о компании удалены, компания закрыта».

7. Можно настроить синхронное дублирование данных

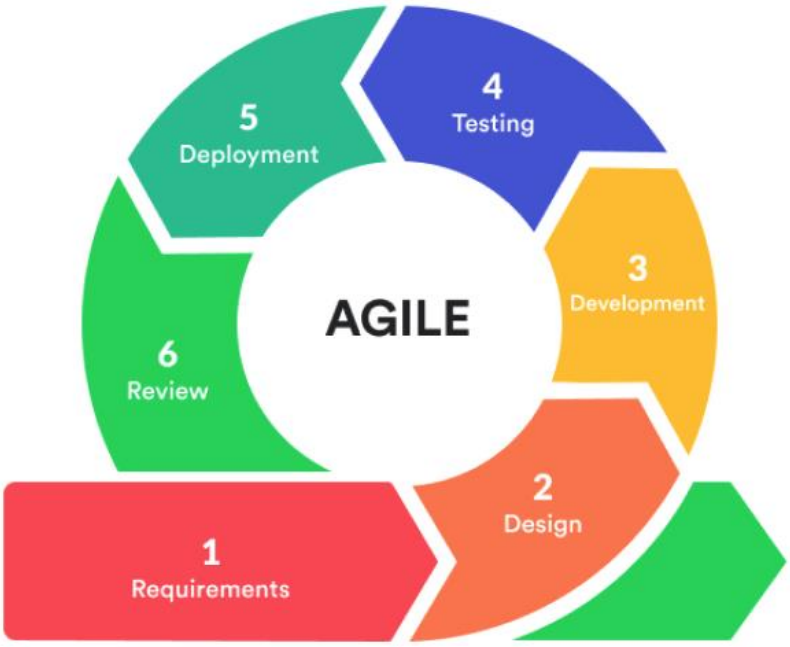
PostgreSQL поддерживает логическую репликацию. Репликация — это сохранение копии базы данных. Копия может находиться на другом сервере. При логической репликации любые изменения синхронизируются во всех копиях базы данных вне зависимости от места их хранения. Это значит, что везде будет храниться одна версия базы данных.

8. Можно без потерь перенести данные из другой СУБД

Объём данных крупных компаний может быть размером 10 терабайт. Их перенос займёт время и приостановит работу. Небольшие компании или стартапы смогут «переехать» в PostgreSQL из другой СУБД быстро, не потеряв ничего в процессе. Перенести все данные можно с помощью специальных инструментов.

3. Обоснование методологии разработки

Наша команда использовала гибкую методику разработки Agile.



Agile — это методология разработки программного обеспечения, призванная помочь командам быстро и эффективно создавать продукты.

Наша команда придерживается следующей философией в работе:

- Чтобы люди работали эффективнее, процессы и инструменты не должны их ограничивать.

В Agile ни процесс, ни тем более программный инструмент не диктует, что людям делать. Более того, они сами решают, как менять процессы/инструменты своей работы.

- Чтобы ускорить процесс разработки, люди также должны взаимодействовать напрямую (без посредников в виде документов или других людей), активно общаться между собой лично, а не письменно.

Наша команда проводила регулярные видеовстречи не менее 3 раз в неделю.

- Работающий продукт важнее исчерпывающей документации

Чтобы клиенты были довольны, им нужен именно работающий продукт. Поэтому разработчики команды фокусировались именно на том, чтобы продуктом можно было как можно скорее воспользоваться, а не на составлении списков, диаграмм, требований, отчетов перед заказчиком.

Чтобы укладываться в сжатые сроки с минимумом затрат, зачастую не стоит связывать себя документацией. Поддержка документации в адекватном продукту состоянии нередко замедляет разработку и требует неоправданно больших затрат.

- Готовность к изменениям важнее, чем следование плану

Чтобы не откладывать риски проектов на последние стадии разработки (когда будет уже поздно уменьшать содержание работы, сдвигать срок или усиливать команду), Agile предлагает не только итеративность работы, но и готовность к изменениям на всех стадиях.

Чтобы в первую очередь делалось самое ценное, текущее видение бизнес-ценности и позиционирования продукта должно быть прозрачно для разработчиков, а процесс их работы должен позволять вносить существенные изменения в прежние планы. В том числе, разработчики должны быть готовы добавлять в продукт незапланированные новые возможности, если они стали ценными в изменившейся ситуации.

Для нашей работы именно готовность к изменениям внесла большой вклад в проект.