**Введение**

В современном мире агротехнологический сектор занимает большую часть в национальной экономике. Роль аграрного сектора трудно переоценить. Эта отрасль занимает стратегическое положение в экономике любого государства, поскольку сельское хозяйство выступает в качестве основной производительной системы, обеспечивающей бесперебойное обеспечение граждан продуктами питания и товарами первой необходимости, без которых нельзя жить. Вот почему развитие аграрного сектора можно считать залогом экономической безопасности государства.

Показателями роли аграрного сектора также является процент активного населения, занятого в сельском хозяйстве, а также удельный вес в структуре внутреннего валового продукта. Эти параметры отражают состояние, в котором находится аграрное производство. Они ежегодно меняются в зависимости от ряда факторов, в том числе мировой экономической политики, а также уровня государственной поддержки жителей села.

Поскольку аграрный сектор – это многоотраслевая сфера, в ней задействованы люди сами различных специальностей. Это и люди, занимающиеся созданием уборочной техники для сельского хозяйства, и доярки, и работники свиноферм, а также представители десятков других профессий. Особую роль в аграрном секторе играет деятельность ученых и исследователей, разрабатывающих инновации в сфере сельского хозяйства. Трудоспособное население, занятое в аграрном секторе, представлено не только наемными работниками, но и индивидуальными предпринимателями (организаторами частных фермерских хозяйств).

В наше время роль мобильных приложений в жизни обычных людей достигла своего пика, ибо мобильные телефоны есть у практически каждого человека современно мира и не важно к какой возрастной или социальной группе он относится. За счет этого всевозможными мобильными приложениями пользуются все больше и больше людей. Некоторым людям тяжело представить современную жизнь без приложения карт или например приложения по бронированию отеля в другом городе. Но до сих пор нет современного мобильного приложения из мира агротехнологического сектора, где люди могли бы читать связанные с профессией новости или же купить сельскохозяйственную технику.

Цель выпускной квалификационной работы — разработать мобильное приложение для агротехнологического сектора.

Для достижения цели выпускной квалификационной работы необходимо решить следующие задачи:

1. Научиться работать с необходимыми технологиями для написания мобильного приложения. Научиться работать с кроссплатформенной технологией React Native, языку программирования TypeScript, языку таблиц стилей CSS и системе управления версиями Git;
2. Проанализировать подходы к разработке мобильных приложений, а также функциональные и архитектурные особенности существующих решений;
3. Разработать уникальный дизайн интерфейса мобильного приложения;
4. Разработать нативное мобильное приложение для отслеживания расходов с помощью фреймворка React Native, языка программирования TypeScript и облачного хранилища Firebase;
5. Выполнить тестирование разработанного мобильного приложения и подтвердить его соответствие установленным требованиям.

Объектом исследования в данной работе является разработка мобильного приложения с помощью кроссплатформенного фреймворка React Native.

Предметом исследования являются этапы разработки, тестирования и ввода в эксплуатацию мобильного приложения.

**1. Технологии и подходы, использованные для реализации приложения.**

**1.1 Язык программирования JavaScript**

JavaScript (JS) является высокоуровневым, интерпретируемым языком программирования, который широко используется для разработки веб-приложений и веб-страниц. В основном, он используется для придания интерактивности веб-сайтам, создания динамических элементов пользовательского интерфейса и управления поведением веб-приложений на стороне клиента.

Вот несколько особенностей JavaScript:

1. Интерпретируемый язык: JavaScript интерпретируется непосредственно во время выполнения, а не компилируется перед запуском, как некоторые другие языки программирования. Это позволяет разработчикам быстро прототипировать и тестировать код;
2. Объектно-ориентированный: JavaScript имеет объектно-ориентированный подход к программированию, позволяя создавать объекты с помощью классов и наследования. Он также поддерживает прототипное наследование, что позволяет объектам наследовать свойства и методы от других объектов;
3. Динамическая типизация: В JavaScript переменные не требуется объявлять с явным указанием типа данных. Они автоматически определяют тип во время выполнения, что облегчает и ускоряет процесс разработки;
4. Широкая поддержка: JavaScript поддерживается практически всеми современными веб-браузерами, что делает его одним из самых распространенных языков программирования в веб-разработке. Он также может использоваться на стороне сервера с использованием платформы Node.js;
5. Функциональное программирование: JavaScript поддерживает функциональное программирование, что позволяет разработчикам писать функции, которые могут быть переданы и использованы как аргументы в других функциях;
6. Работа с событиями: JavaScript имеет встроенную поддержку для обработки событий, таких как щелчок мыши, нажатие клавиш и загрузка веб-страницы. Это позволяет создавать интерактивные пользовательские интерфейсы и реагировать на действия пользователя;
7. Доступ к DOM: JavaScript предоставляет возможность взаимодействия с Document Object Model (DOM), который представляет структуру HTML-документа. Это позволяет программно изменять содержимое, стиль и поведение веб-страницы.

Относительно скорости работы JavaScript, она зависит от множества факторов, таких как сложность кода, оптимизация, архитектура приложения и характеристики используемой среды выполнения. Веб-браузеры постоянно улучшают свои JavaScript-движки для повышения производительности и оптимизации выполнения кода.

Сравнивая JavaScript с аналогами, можно упомянуть несколько языков программирования, которые также широко используются в веб-разработке:

1. HTML: HTML является языком разметки, который определяет структуру и содержимое веб-страницы. В отличие от JavaScript, HTML не является языком программирования и предназначен в основном для разметки содержимого.
2. CSS: CSS (Cascading Style Sheets) используется для определения внешнего вида и стиля веб-страницы. Он позволяет разработчикам определить цвета, шрифты, расположение элементов и другие аспекты визуального представления веб-страницы. В отличие от JavaScript, CSS не предоставляет возможностей программирования и фокусируется исключительно на стилизации.
3. Python: Python является универсальным языком программирования, который может использоваться для разработки веб-приложений. Он обладает широким спектром возможностей и хорошо подходит для разработки серверной части веб-приложений. В отличие от JavaScript, который выполняется на стороне клиента, Python чаще используется на стороне сервера.
4. TypeScript является суперсетом JavaScript, который добавляет статическую типизацию и другие возможности, такие как классы и модули. Он позволяет разработчикам обнаруживать ошибки на этапе разработки и улучшает масштабируемость проектов. TypeScript компилируется в JavaScript, поэтому он может быть использован вместо JavaScript или в совместимости с ним.

JavaScript остается одним из наиболее популярных языков программирования, используемых для разработки интерактивных веб-приложений. Он обладает широким набором возможностей, хорошей поддержкой и непрерывно развивающимся экосистемой.

**1.2 Кроссплатформенная мобильная разработка**

Мобильная кроссплатформенная разработка - это процесс создания мобильных приложений, которые могут работать на разных платформах, таких как iOS и Android, с использованием общего кода или основных технологий. Вместо написания отдельного кода для каждой платформы, разработчики могут использовать фреймворки, инструменты и языки программирования, которые позволяют создавать приложения, переносимые на несколько платформ.

Преимущества мобильной кроссплатформенной разработки:

1. Экономия времени и ресурсов. Одним из главных преимуществ кроссплатформенной разработки является возможность написания общего кода, который может быть повторно использован на разных платформах. Это позволяет сэкономить время и усилия при разработке и поддержке приложений.
2. Более широкий охват аудитории. Кроссплатформенные приложения позволяют достичь большей аудитории, так как они могут быть запущены на разных операционных системах. Это позволяет компаниям и разработчикам достичь большего количества пользователей и расширить свою аудиторию.
3. Упрощенная поддержка и обновления. Вместо того, чтобы выпускать обновления для разных версий приложений для разных платформ, кроссплатформенная разработка позволяет выпустить обновления одновременно для всех платформ. Это облегчает процесс поддержки и обновления приложений.
4. Единое пользовательское впечатление. Кроссплатформенные фреймворки обычно предлагают набор компонентов и стилей, которые помогают создавать единое пользовательское впечатление на разных платформах. Это позволяет создавать приложения, которые выглядят и ведут себя схожим образом на разных устройствах.

Некоторые популярные фреймворки для мобильной кроссплатформенной разработки включают:

1. React Native: Он использует JavaScript и библиотеку React для создания переносимых мобильных приложений, работающих на платформах iOS и Android.
2. Flutter. Это фреймворк от Google, основанный на языке программирования Dart. Он позволяет создавать высококачественные кроссплатформенные приложения с использованием собственного движка рендеринга.

Выбор между кроссплатформенной и нативной мобильной разработкой зависит от ряда факторов.

Кроссплатформенная мобильная разработка подходит, когда:

1. Ограниченные ресурсы. Если у вас ограниченные ресурсы в виде времени, бюджета или команды разработчиков, то кроссплатформенная разработка может быть более эффективным решением. Одна кодовая база может быть использована для создания приложений для разных платформ, что сокращает необходимость разработки и поддержки отдельного кода для каждой платформы;
2. Простые приложения. Если вам требуется создать относительно простое приложение с базовыми функциями и стандартным пользовательским интерфейсом, то кроссплатформенная разработка может быть достаточной. Кроссплатформенные фреймворки предлагают набор готовых компонентов и инструментов для создания базовых мобильных приложений;
3. Быстрая разработка и выход на рынок. Если вашей целью является быстрая разработка и выход на рынок, то кроссплатформенная разработка может быть предпочтительной. Одна кодовая база позволяет сократить время разработки и обновления приложений на разных платформах.

Нативная мобильная разработка стоит рассмотреть, когда:

1. Высокая производительность и доступ к функциональности платформы. Если ваше приложение требует максимальной производительности или полного доступа к функциям и возможностям конкретной платформы, то нативная разработка предоставляет большую гибкость и контроль над приложением. Нативные приложения обычно имеют более высокую производительность и могут полностью использовать специфические функции платформы;
2. Сложные и пользовательские интерфейсы. Если ваше приложение имеет сложные пользовательские интерфейсы, специфические анимации или требует максимальной адаптации к каждой платформе, то нативная разработка может быть предпочтительной. Нативные инструменты и фреймворки обычно предлагают более гибкие возможности для создания сложных интерфейсов и пользовательских взаимодействий.
3. Высокая безопасность. Если безопасность данных является высоким приоритетом, то нативная разработка может предоставить более надежные механизмы защиты. Нативные инструменты и API обеспечивают лучшую защиту данных и имеют более широкую поддержку в области безопасности.

**1.3 Фреймворк React Native**

React Native - это фреймворк для разработки мобильных приложений, созданный компанией Facebook. Он позволяет разработчикам использовать язык программирования JavaScript и библиотеку React для создания переносимых мобильных приложений, которые могут работать как на платформе iOS, так и на Android.

Основные особенности и характеристики React Native:

1. Кроссплатформенность. Одной из ключевых особенностей React Native является возможность создания кроссплатформенных приложений. Большая часть кода пишется на JavaScript, а затем переносится на разные платформы, что позволяет экономить время и ресурсы при разработке приложений для iOS и Android;
2. Использование JavaScript и React. React Native основан на библиотеке React, которая позволяет разработчикам создавать компоненты пользовательского интерфейса и эффективно управлять состоянием приложения. Основным языком программирования для разработки React Native является JavaScript, что делает его доступным для большого числа разработчиков;
3. Гибкость и масштабируемость. React Native позволяет создавать сложные и интерактивные пользовательские интерфейсы, используя готовые компоненты и библиотеки. Он также обладает гибкостью, позволяя интегрировать существующий код на Java, Objective-C или Swift в приложение React Native, что делает его идеальным выбором для миграции и усовершенствования существующих проектов;
4. Быстрое развертывание и обновления. React Native позволяет разработчикам быстро развертывать и обновлять приложения. Благодаря горячей перезагрузке (Hot Reload) и возможности обновления приложений на лету (Over-the-Air Updates), изменения в коде можно немедленно видеть и тестировать без необходимости полной перекомпиляции и перезагрузки приложения;
5. Нативные компоненты. React Native предоставляет возможность использовать нативные компоненты платформы, такие как кнопки, полосы прокрутки, текстовые поля и другие, что позволяет создавать приложения с нативным видом и поведением. Это делает приложения, созданные с использованием React Native, более производительными и близкими к нативным приложениям.

Относительно скорости работы React Native, она зависит от разных факторов, таких как сложность приложения, оптимизация кода и производительность устройств. React Native использует нативные компоненты, что может обеспечивать более высокую производительность по сравнению с полностью веб-основанными фреймворками.

**1.4 Язык стилей CSS**

CSS (Cascading Style Sheets) - это язык стилей, который используется для описания внешнего вида и форматирования документа, написанного на языке разметки, таком как HTML или XML. CSS определяет, как элементы HTML должны быть представлены на экране, включая их размеры, цвета, шрифты, отступы, позиционирование и другие свойства.

Основная цель CSS - разделение содержимого (структуры) и представления (стиля) веб-страницы. С помощью CSS можно задать единый стиль и макет для всех страниц сайта, что упрощает его разработку и обслуживание. CSS также позволяет создавать адаптивные и отзывчивые дизайны, анимации и эффекты, что обогащает пользовательский опыт.

CSS не является единственным языком стилей для веб-разработки, и в процессе развития веб-технологий появились альтернативы CSS. Однако CSS остается наиболее широко используемым и предпочтительным языком стилей по нескольким причинам:

1. Стандартная поддержка веб-браузерами. CSS является стандартом веб-разработки и широко поддерживается всеми современными веб-браузерами. Это означает, что CSS-стили будут применяться к веб-страницам на всех платформах и устройствах с использованием общих правил и синтаксиса;
2. Простота и читаемость. CSS имеет простой и понятный синтаксис, который легко понять и использовать. Он основан на выборе элементов и применении стилевых правил к ним. Благодаря этому, разработчики могут быстро и эффективно создавать и изменять стили веб-страниц;
3. Мощные возможности стилизации: CSS обладает широким набором возможностей для стилизации веб-страниц. Он позволяет управлять цветами, шрифтами, размерами, позиционированием, анимациями и другими аспектами визуального представления элементов. Благодаря этим возможностям, разработчики могут создавать привлекательные и функциональные пользовательские интерфейсы;
4. Каскадный подход: Одной из ключевых особенностей CSS является его каскадный подход. Это означает, что стили могут быть применены к элементам веб-страницы с использованием приоритетов и наследования. Это позволяет эффективно управлять стилями на разных уровнях, от общих правил для всей страницы до специфичных правил для отдельных элементов;
5. Расширяемость и модульность: CSS обладает возможностью расширяться и модульности. С появлением новых версий CSS и спецификаций, введение новых свойств и возможностей становится возможным без полной переписывания кода. Благодаря модульности, разработчики могут создавать отдельные стили для разных компонентов и повторно использовать их в разных проектах.

Вместе эти факторы делают CSS предпочтительным выбором для стилизации веб-страниц и приложений. Он предлагает стандартизированный и мощный подход к стилизации, который позволяет разработчикам создавать красивые и современные пользовательские интерфейсы для веб-платформы.

В контексте React Native CSS используется в виде стилизации компонентов. React Native предлагает подход, называемый "стилизация встроенными стилями" (inline styles). Вместо использования CSS-файлов или классов CSS, стили применяются к компонентам непосредственно в коде JavaScript.

В React Native для создания стилей используется JavaScript-объект, который содержит свойства стиля и их значения.

Css в React Native используется следующим образом:

1. Inline стили. Вместо использования отдельных CSS-файлов или классов CSS, стили применяются непосредственно в коде компонента с использованием объектов JavaScript;
2. Свойства стилей. В React Native многие свойства стилей аналогичны тем, что используются в CSS, но есть некоторые отличия и расширения. Например, вы можете использовать flex для управления макетом и распределением элементов, backgroundColor для задания цвета фона, fontSize для определения размера шрифта и т.д. Кроме того, в React Native существуют специфические свойства стилей, такие как fontWeight, textAlign, textTransform, которые применяются к текстовым компонентам;
3. Селекторы и псевдоклассы. В React Native нет полноценной поддержки селекторов и псевдоклассов, как в CSS. Однако вы можете использовать условные операторы JavaScript или условные рендеринги компонентов, чтобы применять стили на основе определенных условий или состояний;
4. Глобальные стили: В React Native вы также можете определить глобальные стили, которые будут применяться ко всем компонентам в приложении. Вы можете создать отдельный файл для глобальных стилей и импортировать его в нужных компонентах.

React Native обеспечивает преобразование стилей в соответствующие нативные стили на каждой платформе (iOS и Android), чтобы обеспечить консистентный внешний вид приложения на разных устройствах.

**1.5 Система контроля версий git, удаленные репозиторий GitLab**

Git - это распределенная система управления версиями, которая используется для отслеживания изменений в файловой системе и совместной работы над проектами. Она была создана Линусом Торвальдсом в 2005 году и с тех пор стала одним из самых популярных инструментов разработки программного обеспечения.

Git используется в различных областях разработки, включая веб-разработку, мобильную разработку, научные исследования и другие проекты с открытым исходным кодом. Он позволяет разработчикам эффективно управлять версиями кода, вносить изменения, создавать ветки, объединять изменения из разных источников и многое другое.

GitLab - это веб-платформа для управления Git-репозиториями и совместной разработки проектов. Он предоставляет функциональность, аналогичную GitHub, но также предлагает расширенные функции и возможность самостоятельно развернуть GitLab на собственном сервере.

GitLab популярен по нескольким причинам:

1. Открытый исходный код. GitLab является проектом с открытым исходным кодом, что позволяет пользователям свободно использовать и изменять его, а также вносить свои вклады в развитие платформы;

2. Полная DevOps-платформа. GitLab предлагает полный набор инструментов для DevOps, включая возможности CI/CD (непрерывная интеграция/непрерывная доставка), управление задачами и проблемами, контроль качества кода и другие функции. Это делает GitLab удобным выбором для команд, которым требуется интеграция и автоматизация процессов разработки;

3. Локальная установка. Одной из главных особенностей GitLab является возможность развертывания на собственном сервере или в облаке. Это позволяет организациям сохранять контроль над своими данными и настраивать платформу в соответствии с их требованиями и политиками безопасности.

Рассмотрим сравнение GitHub и GitLab:

1. Популярность. GitHub является наиболее популярной платформой для хостинга Git-репозиториев и совместной разработки проектов с открытым исходным кодом. Он имеет большое сообщество разработчиков и широкую известность. GitLab, хотя и менее популярен, также обладает значительной базой пользователей и активно развивается.

2. Функциональность. Обе платформы предлагают основные функции системы контроля версий Git, такие как управление репозиториями, ветвление и слияние кода, отслеживание проблем и запросов на слияние. Однако GitLab предлагает расширенные возможности для CI/CD, управления задачами и многое другое. Это делает его предпочтительным для команд, которым требуется интеграция DevOps.

3. Развертывание. GitHub является облачной платформой, что означает, что вам необходимо размещать свои репозитории на серверах GitHub. GitLab, с другой стороны, предлагает возможность самостоятельного развертывания на вашем сервере или в облаке. Это дает больше контроля и гибкости для организаций, которые хотят сохранить данные на собственной инфраструктуре.

4. Стоимость. GitHub предлагает бесплатные публичные репозитории и платные приватные репозитории для коммерческих проектов. GitLab также предлагает бесплатную версию для открытых проектов и коммьюнити, а также платные планы для коммерческого использования. Однако, за счет возможности самостоятельного развертывания, GitLab может быть более экономически выгодным в случае больших коммерческих организаций.

В целом, выбор между GitHub и GitLab зависит от потребностей и предпочтений. GitHub хорошо подходит для проектов с открытым исходным кодом, имеет большое сообщество и простоту использования. GitLab предлагает расширенные возможности DevOps и дает больше контроля организациям, предпочитающим самостоятельное развертывание и интеграцию существующей инфраструктуры.

На проекте будет использоваться Gitlab в основном потому что он предоставляет более широкий спектр функций и возможностей, включая интегрированный CI/CD, управление задачами, контроль качества кода и многое другое. Он также позволяет самостоятельно развернуть платформу на собственном сервере или в облаке, что дает больше контроля и гибкости для организаций. Кроме того, GitLab имеет открытый исходный код и активное сообщество разработчиков, что способствует его непрерывному развитию и поддержке.

**1.6 Серверный фреймворк NestJs для написания Backend приложения**

Бэкенд (backend) в приложенни играет важную роль в обеспечении функциональности и взаимодействия с данными, а также обработке бизнес-логики.

Несколько основных причин, для которых нужен бэкенд в приложении:

1. Обработка запросов и отдача данных. Бэкенд отвечает за обработку входящих запросов от клиентской части сайта и отдачу данных в соответствии с этими запросами. Он может выполнять операции базы данных, обращаться к внешним сервисам, обрабатывать алгоритмы и предоставлять клиенту необходимую информацию;

2. Управление бизнес-логикой. Бэкенд занимается управлением бизнес-логикой сайта. Он может обрабатывать платежи, авторизацию и аутентификацию пользователей, управление правами доступа, управление корзиной покупок и другие важные функции, связанные с бизнес-процессами;

3. Управление базой данных. Бэкенд обеспечивает взаимодействие с базой данных. Он может выполнять операции чтения, записи, обновления и удаления данных в базе данных. Бэкенд также может управлять миграциями базы данных, индексацией и оптимизацией запросов;

4. Обеспечение безопасности. Бэкенд отвечает за обеспечение безопасности сайта. Он может осуществлять проверку подлинности пользователей, применять аутентификацию и авторизацию, защищать данные от несанкционированного доступа и обеспечивать защиту от взломов и атак;

5. Работа с внешними сервисами. Бэкенд может взаимодействовать с различными внешними сервисами, такими как платежные шлюзы, API сторонних сервисов, почтовые сервисы и другие. Он может отправлять запросы к внешним сервисам и обрабатывать полученные ответы.

Это лишь несколько примеров функциональности, которую бэкенд обеспечивает на сайте. Он играет важную роль в обеспечении работы сайта, обработке данных и обеспечении безопасности, а также реализации бизнес-логики и взаимодействия с внешними сервисами.

NestJS - это прогрессивный фреймворк для создания масштабируемых и эффективных серверных приложений на языке программирования TypeScript. Он основан на архитектуре модулей, использует принципы SOLID и обеспечивает эффективную разработку приложений с использованием паттерна Dependency Injection.

Основные причины, почему NestJS может быть предпочтительным выбором:

1. Масштабируемость и эффективность. NestJS построен на основе Node.js и использует его преимущества, такие как неблокирующий ввод/вывод и асинхронность, что делает его масштабируемым и эффективным в обработке больших нагрузок;
2. Модульная архитектура: NestJS предлагает модульную архитектуру, которая позволяет организовать код в отдельные модули с четкими границами ответственности. Это улучшает читаемость кода, повторное использование и тестируемость;
3. TypeScript. NestJS полностью поддерживает TypeScript, что добавляет статическую типизацию и улучшает разработку, позволяя обнаруживать ошибки на этапе компиляции;
4. Dependency Injection (DI). NestJS использует паттерн Dependency Injection для управления зависимостями и обеспечения легкой интеграции различных компонентов. Это способствует созданию модульного и расширяемого кода;
5. Поддержка WebSocket. NestJS предоставляет встроенную поддержку WebSocket, что делает его идеальным для создания приложений в реальном времени, таких как чаты или игры.

Аналогами NestJS являются другие фреймворки для разработки серверных приложений на Node.js, такие как Express.js, Ruby on Rails, Django, ASP.NET.

Cравнение NestJS с некоторыми из этих аналогов:

1. Express.js. Express.js - это минималистичный и гибкий фреймворк для разработки серверных приложений на Node.js. NestJS построен на основе Express.js и предоставляет схожий синтаксис и функциональность, но добавляет дополнительные возможности, такие как модульность, автоматическую генерацию документации и поддержку Dependency Injection;
2. Ruby on Rails. Ruby on Rails (RoR) - это популярный фреймворк для разработки веб-приложений на языке Ruby. NestJS и RoR имеют сходства в том, что они оба предлагают структурированный подход к разработке и обеспечивают множество встроенных функций. Однако NestJS работает на платформе Node.js, что обеспечивает большую гибкость и производительность, а также позволяет использовать JavaScript или TypeScript;
3. Django. Django - это мощный фреймворк для разработки веб-приложений на языке Python. Он предоставляет широкий набор функций и инструментов для разработки бэкенда. NestJS и Django имеют сходства в том, что они оба позволяют разрабатывать масштабируемые приложения с использованием модульной структуры. Однако NestJS работает на платформе Node.js и предоставляет поддержку TypeScript, что может быть привлекательным для разработчиков, предпочитающих JavaScript и его современные возможности;
4. ASP.NET. ASP.NET - это платформа разработки веб-приложений от Microsoft, использующая языки программирования C# или VB.NET. Она предлагает широкие возможности для разработки бэкенда и имеет множество встроенных инструментов и библиотек. NestJS и ASP.NET имеют разные подходы и стек технологий, и выбор между ними зависит от предпочтений языка программирования, платформы и требований проекта.

**1.7 Сборщик модулей и конфигуратор приложения Webpack**

Webpack - это инструмент для сборки (bundling) модулей веб-приложения. Он используется для объединения различных файлов JavaScript, CSS, изображений и других ресурсов в один или несколько оптимизированных пакетов, которые могут быть загружены и использованы веб-приложением.

Некоторые основные причины, почему Webpack популярен и зачем он нужен в кроссплатформенном приложении на React Native:

1. Модульность. Webpack позволяет разбить ваше приложение на модули и импортировать их по необходимости. Это помогает организовать код, улучшает его читаемость и облегчает его поддержку и масштабирование;
2. Управление зависимостями. Webpack позволяет управлять зависимостями в вашем проекте. Он автоматически определяет и подключает необходимые модули и библиотеки, обрабатывает их в соответствии с настройками и формирует итоговые пакеты;
3. Транспиляция и преобразование. Webpack предоставляет возможность использовать различные транспиляторы (например, Babel) и преобразователи (например, PostCSS) для преобразования кода. Это позволяет использовать новейшие возможности JavaScript и других языков, а также оптимизировать и минифицировать код;
4. Оптимизация ресурсов. Webpack позволяет оптимизировать и минимизировать ресурсы, такие как CSS, JavaScript и изображения. Это помогает уменьшить размер файлов, ускорить загрузку страницы и повысить производительность приложения;
5. Расширяемость. Webpack обладает мощной системой плагинов, которые позволяют настроить его поведение и добавить дополнительные возможности. Вы можете использовать готовые плагины из огромного экосистемы Webpack или создать собственные плагины по необходимости.

Популярность Webpack объясняется его гибкостью, мощными возможностями и широким сообществом разработчиков. Он является одним из наиболее распространенных инструментов для сборки веб-приложений и активно поддерживается сообществом разработчиков.

Сравнение Webpack с аналогами может быть сложным, так как существует несколько альтернативных инструментов для сборки, таких как Parcel, Rollup и Vite, каждый из которых имеет свои преимущества и ориентирован на конкретные сценарии использования. Выбор инструмента зависит от ваших потребностей, требований проекта и личных предпочтений.

На данный момент самый популярный аналог это Vite и поэтому основное сравнение будет идти между Webpack и Vite.

Сравнение между двумя технологиями:

1. Скорость сборки. Vite изначально был разработан с упором на быструю скорость разработки. Он использует сборку на основе ES модулей и предоставляет мгновенную перезагрузку (instant reloading) при разработке. Веб-приложение запускается на сервере разработки, и изменения отображаются мгновенно без необходимости полной пересборки. В то время как у Webpack есть мощные возможности и плагины, он обычно требует больше времени на сборку и пересборку при изменениях кода;
2. Конфигурация и настройка. Webpack предлагает широкие возможности настройки и конфигурации. Это может быть полезно для сложных проектов, которым требуется гибкость и множество плагинов. Vite, с другой стороны, стремится к простоте и минимальной конфигурации. Он предоставляет предустановленные настройки для различных фреймворков и библиотек, что делает его быстрее и легче в использовании для маленьких и средних проектов;
3. Экосистема и поддержка. Webpack является проверенным временем инструментом и имеет обширное сообщество разработчиков. У него больше плагинов и решений для различных задач. Он также имеет более длительную историю разработки и обновлений, что может давать большую уверенность в его надежности и стабильности. Vite, с другой стороны, является новым инструментом, но также получил значительную популярность за счет своей скорости и простоты использования.

Выбор между Webpack и Vite зависит от ваших конкретных потребностей и предпочтений. Если у вас большой проект, требующий сложной конфигурации и поддержки различных плагинов, то Webpack может быть лучшим выбором. Однако, если вам нужна быстрая разработка и простота использования для маленьких и средних проектов, то Vite может быть предпочтительным вариантом.

**1.8 Тестирование приложения и технология тестирования Jest**

Тестирование в приложении играет важную роль в обеспечении качества программного продукта.

Основные причины, почему тестирование необходимо:

1. Обнаружение ошибок. Тестирование позволяет выявить ошибки и дефекты в приложении, помогая выявить проблемы до того, как они окажутся в руках пользователей. Это позволяет снизить количество ошибок и повысить стабильность и надежность приложения;

2. Поддержка изменений. Тестирование помогает подтвердить, что изменения и новые функции, внесенные в приложение, не приводят к возникновению нежелательных побочных эффектов или нарушению существующей функциональности. Это обеспечивает сохранение корректности и работоспособности приложения при его развитии;

3. Улучшение проектирования. Тестирование позволяет обнаружить проблемы в проектировании кода и архитектуры приложения. Он выявляет недостатки в структуре и организации кода, а также помогает улучшить его понятность и поддерживаемость;

4. Ускорение разработки. Тестирование позволяет автоматизировать процесс проверки приложения на соответствие требованиям и функциональности. Это упрощает и ускоряет процесс разработки, позволяя разработчикам быстро выявлять и исправлять проблемы.

Перейдем к видам тестирования:

1. Модульное тестирование. Это тестирование отдельных модулей (компонентов, функций, классов) приложения для проверки их работоспособности в изоляции;

2. Интеграционное тестирование. Это тестирование взаимодействия между различными модулями приложения для проверки их взаимодействия и корректности передачи данных;

3. Функциональное тестирование. Это тестирование функциональности приложения в целом для проверки соответствия его требованиям и ожидаемому поведению;

4. UI-тестирование. Это тестирование пользовательского интерфейса приложения для проверки его внешнего вида, взаимодействия и навигации.

Jest - это популярный фреймворк для тестирования JavaScript-приложений, включая приложения на React Native.

Jest обладает следующими преимуществами:

1. Простота использования. Jest имеет простой и понятный синтаксис, что делает его легким в освоении для разработчиков. Он предлагает удобные функции для написания и запуска тестов, а также для создания моков и имитаций;

2. Широкий функционал. Jest поддерживает различные виды тестирования, включая модульное, интеграционное и UI-тестирование. Он также предоставляет функции для измерения покрытия кода тестами и выполнения снимков (snapshots) для контроля визуальных изменений;

3. Высокая производительность. Jest обладает оптимизированной средой выполнения тестов, что позволяет его использование в крупных проектах с большим количеством тестов. Он обеспечивает быстрое выполнение тестов и имеет встроенные механизмы кэширования для сокращения времени сборки;

4. Интеграция с React и React Native. Jest предоставляет удобные средства для тестирования компонентов React и React Native. Он позволяет легко создавать и проверять компоненты, имитировать события и проверять их результаты.

В сравнении с аналогами, Jest имеет широкое сообщество разработчиков и активную поддержку. Он является популярным выбором для тестирования JavaScript-приложений и хорошо интегрируется с экосистемой React и React Native.

**1.9 Управление состоянием приложения и технология Redux**

Для масштабируемости и удобного управления состянием приложения используются state менеджеры, они нужны для того чтобы централизовано хранить и управлять состоянием. На проекте используется самый популярный менеджер Redux.

Redux - это библиотека для управления состоянием (state management) в JavaScript-приложениях. Она предоставляет простую и предсказуемую модель работы с данными, что упрощает разработку сложных приложений.

Основная цель Redux - управление состоянием приложения и обеспечение однонаправленного потока данных. Он использует централизованное хранилище (store), которое содержит все данные приложения. Компоненты приложения могут обращаться к хранилищу и изменять состояние через действия (actions) и редукторы (reducers).

Несколько причин, почему Redux может быть полезен и превосходит некоторые аналоги:

1. Предсказуемость и отслеживаемость. Redux использует однонаправленный поток данных, что делает изменение состояния предсказуемым и отслеживаемым. Это позволяет легко понять, как и когда происходят изменения данных в приложении;

2. Разделение логики и представления. Redux отделяет логику обработки данных от компонентов представления. Это способствует более чистой и модульной структуре кода, упрощает тестирование и повышает переиспользуемость компонентов;

3. Централизованное хранилище. Централизованное хранилище в Redux упрощает управление состоянием приложения. Компоненты могут получать доступ к состоянию и обновлять его, что облегчает синхронизацию данных между различными частями приложения;

4. Расширяемость. Redux предоставляет возможность использовать плагины и middleware для расширения функциональности. Это позволяет легко добавлять дополнительные возможности, такие как логирование, асинхронные операции и временные снимки состояния.

Возможными аналогами Redux являются MobX, Vuex (для Vue.js), и Flux (предшественник Redux). В сравнении с аналогами, Redux обладает широкой популярностью и активным сообществом разработчиков. Он имеет хорошую документацию и обширный набор инструментов и плагинов для разработки. Кроме того, Redux обеспечивает простую и предсказуемую модель управления состоянием, что делает его популярным выбором для масштабных приложений.