МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования

Пензенский государственный университет архитектуры и строительства

Факультет__инженерно-строительный_____
Кафедра_информационно-вычислительных систем___ Направление_09.04.02 «Информационные системы и технологии»_ Программа _____

Оценка _____

«____»_____20__г.
_____ (подпись секретаря ГЭК)

«К защите»
Заведующий кафедрой
_____(_____)
«____»_____20__г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

на тему __Исследование разработки PWA на примере Bullet Journal Application

Научный руководитель
___д.х.н. профессор Кошев А.Н. (должность, степень, фамилия, инициалы)

_____
(подпись)

Консультант по разделу_____
_д.х.н. профессор Кошев А.Н. _____ (должность, степень, фамилия, инициалы, подпись)

Студент гр 18ИСТ 1м

Зайцев Владислав Сергеевич (фамилия, имя, отчество)

_____ (подпись

_____ (дата

Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования Пензенский государственный университет архитектуры и строительства Факультет__ _____ ___инженерно-строительный_____ Кафедра_____ _информационно-вычислительных систем_____ Направление _09.04.02 «Информационные системы и технологии»_ Группа __________

ЗАДАНИЕ
на выпускную квалификационную работу

Студент ____Зайцев Владислав Сергеевич_____ _____
(фамилия, имя, отчество)

Тема выпускной работы:
____ исследование разработки PWA на примере Bullet Journal Application

_____ _____ _____ Время выполнения работы с _25.05.2020 по _25.06.2020 г.

Руководитель выпускной квалификационной работы _____ Кошев А. Н., профессор, д.х.н., кафедра ИВС ПГУАС_____ (фамилия, инициалы, должность, степень, место работы)

_____

Тема выпускной квалификационной работы и руководитель утверждены приказом № 06-09-920__ от «_26_» ___ноября_____ 2020 г.

Консультант по разделу ____Кошев А.Н., д.х.н. профессор, ПГУАС
(фамилия, инициалы, должность, степень, место работы)

Место выполнения работы ___ПГУАС_____

_____

Заведующий кафедрой _Васин Леонид Анатольевич___«___» __-___ 2020г.

Задание принял к исполнению «___» _____ 20__ г.

_____-_____
(подпись студента)

1. Содержание задания

Предпроектные исследования

Выбор архитектуры приложения

Выбор технологий для построения приложения

Разработка кроссплатформенного PWA приложения

Проработка безопасности приложения

2. Задание и исходные данные по разделу

Подпись консультанта

3. Рекомендуемая исходная литература

Bakaus P. What Are Progressive Web AMPs? [Электронный ресурс] // 2016. URL: https://www.smashingmagazine.com/2016/12/progressive-web-amps/

Green I. From AMP to PWA [Электронный ресурс] // 2016. URL: h

Сальников М., Липатцев А., Кардава З., Пугачев С. Progressive Web Apps Day. Онлайн конференция [Электронный ресурс] // 2016. URL: pwaday.

Демьяненко М. Что такое Progressive Web Apps и какие возможности они открывают для вашего бизнеса [Электронный ресурс] // 2016. URL:

Markov D. Progressive Web Apps. Everything You Should Know About Progressive Web Apps [Электронный ресурс] // 2016. URL: http://tutorialzine.com/2016/09/everything-you-should-know-aboutprogressive-web-apps/

_____

Подпись руководителя выпускной работы __

КАЛЕНДАРНЫЙ ГРАФИК РАБОТЫ ПО РАЗДЕЛАМ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ МАГИСТРА

№ п/п	Перечень разделов работы	Срок выполнения	Отметки о выполнении
1	Анализ предметной области	21.05.2020	Выполнено
2	Выбор архитектуры приложения	24.05.2020	Выполнено
3	Выбор технологий	08.06.2020	Выполнено
4	Разработка функционала и интерфейса приложения	19.06.2020	Выполнено
5	Проработка безопасности приложения	24.06.2020	Выполнено

Составлен «___» _____20__ г.

_____ _____

(Подпись руководителя) (Подпись студента)

Реферат

Ключевые слова: PWA, КРОССПЛАТФОРМЕННОЕ ПРИЛОЖЕНИЕ, SERVERLESS, АРХИТЕКТУРА ПРИЛОЖЕНИЙ, РАЗРАБОТКА, REACT, ОБЛАЧНЫЕ СЕРВИСЫ, ЛЯМБДА ФУНКЦИИ, SERVICE WORKER

Тема ВКР: «Исследование разработки PWA на примере Bullet Journal Application».

Результатом работы является готовое кроссплатформенное PWA приложение, способное работать без наличия интернет-соединения.

В разработке приложения задействован фреймворк React, облачный сервис firebase, лямбда функции и service worker. Приложение реализует serverless архитектуру.

Область применения: приложение позволяет пользователям эффективно управлять делами. Опыт, полученный в ходе разработки, позволяет разрабатывать мощные современные приложения.

Эффективность: приложение позволяет пользователям оптимизировать управление временными ресурсами, как своими, так и командными, благодаря дополнительному канбан функционалу.

Актуальность: на данный момент PWA приложения являются новым и достаточно прогрессивным решением, позволяющим достичь кроссплатформенности и более продвинутого функционала по сравнению с аналогами

Оглавлени

					ВКР-09.04.02-181323-20.81	
					Исследование разработки PWA на примере Bullet Journal Application	Лит. / Масса / Масштаб
Разработал	Зайцев В.С.					
Провери	Кошев А.Н.					Лист 5 / Листов 84
Т. Контр.	Кошев А.Н.					
Реценз.	Ремонтов А.П.					ПГУАС,каф.ИВС 18ИСТ1м
Н. Контр.	Кошев А..Н.					
Утв.	Васин Л.А.					

e

Введение.....8
1. Анализ предметной области.....12
1.1. Мобильные приложения в современном мире.....12
1.2. Сравнение Progressive Web Apps (PWA) с сайтом и мобильным приложением.....13
2. Архитектура приложения.....21
2.1. Выбор фреймворка.....21
2.2. Выбор языка программирования.....30
2.3. Использование облачных сервисов.....31
2.5. Лямбда функции.....35
2.6. FireBase для упрощения разработки.....37
2.7. Использование service worker-ов в приложении.....39
2.8. Тестирование.....40
2.9. CI/CD Доставка пользователям.....43
3. Используемые технологии построения приложения.....45
3.1. React.....45
3.2. Redux.....46
3.3. React-Router.....48
3.4. Самописный D&D.....48
3.5. Самописный Virtualized list.....48
3.6. Typescript.....49
3.7. Webpack.....49
3.8. Git.....51
3.9. Git flow.....52
4. Разработка кроссплатформенного PWA приложения.....55
4.1. Вход/регистрация.....55
4.2. Форма регистрации.....56
4.3. Оповещения об ошибках.....57
4.4. Классическое представление списков.....58
4.5. Поиск по спискам.....59

4.6. Список задач.....60
4.7. Архив готовых дел.....62
4.8. Выход из учетной записи.....63
4.9. Работа с большими списками.....64
4.10. Хранение данных локально.....65
4.11. Расширенный функционал приложения.....66
4.12. Работа с канбан листом.....67
4.13. Общий вид приложения.....71
5. Безопасность приложения.....73
ЗАКЛЮЧЕНИЕ.....75
Список литературы.....77
Приложение А.....80
Приложение Б.....81
Приложение В.....82
Приложение Г.....83
Приложение Д.....84

ВВЕДЕНИЕ

Приложение «Bullet Journal Application» служит для облегчения работы с ежедневными задачами при помощи визуализации подхода со схожим названием.

Оно предоставляет возможность пользователям, вместо ручного ввода списка дел и управления им, при составлении нового каждый период (день/неделю/месяц/год), автоматизированный процесс.

В данном подходе управлением делами предполагается распоряжаться делами схоже, как и во всеобще известных подходах, таких как SCRAM и KANBAN: имеется список дел для каждого периода, из которых составляется список дел для более мелких периодов. Подобные Agile методологии очень эффективный подход в управлении небольшими командами.

Из схожего у нас имеется общий пул дел, из которых мы можем выбрать те, что наиболее приоритетны для следующего периода.

В «Bullet Journal Application» подходе у нас имеется список глобальных дел на год, которые мы можем разбить на более мелкие по месяцам, либо привязать каждое дело к определенному месяцу. У каждого месяца соответственно имеется похожая структура с глобальными задачами на месяц, которые могут быть привязаны к неделям и из них уже выбираются ежедневные задачи. Каждая вложенная в период задача не обязательно должна презентовать часть какой-то более глобальной задачи.

Так же к глобальным задачам могут относиться повторяющиеся задачи, как занятия споротом, которые могут

быть глобальной задачей на год и в будущем перейти на следующий год, так и повторяющиеся задачи на небольшой период времени, такие как чтение какой-то определенной книги, обучение на курсах или хобби.

Каждое дело, которое не завершено в текущем периоде переносится в следующий период или отменяется, если оно уже не актуально, как например, пропущенная встреча. При большом списке дел такие переносы на бумаге могут занимать достаточно много времени и со временем начинают надоедать. Здесь то и вступает в действие наше приложение, которое позволит пользователю не беспокоится о ручном переносе дел, а сосредоточиться на их выполнении, более качественном описании и поддержке, а так же в случае просрочки напомнит и предупредит о приближающихся важных встречах. Так же для просроченных дел в последующих периодах будет добавлена пометка, чтобы пользователь видел проблемные задачи.

Так же в ручном оформлении подобных журналов возникают проблемы с оформлением периодических дел, поддержку которых будет осуществлять наше приложение и выводить статистику по повторяющимся делам, чтобы пользователю лучше и понятнее был виден прогресс в них. Это, на мой взгляд, будет повышать мотивацию в их выполнении, если пользователь будет видеть, что у него начались проблемы с ними, либо будет сигнализировать и том, что стоит пересмотреть распорядок дня, чтобы было больше времени на повседневные занятия.

В итоге мы получаем приложение помощник для людей, которые хотят структурировать свой распорядок дня, но у них

нет времени на изучение принципов составления подобных журналов и их поддержку, и заполнение. Сама идея создания подобного приложения у меня появилась в процессе поиска способа структурировать работу с распорядком дня. Мне очень понравилась идея bullet journal-а, но подходящего по функционалу приложения, среди существующих на рынке, я не нашел.

Приложение должно решать следующие задачи:

- Приложение должно быть кроссплатформенное, чтобы пользователь мог пользоваться им с разных устройств без проблем и не было необходимости в разработке похожего функционала для каждого типа устройства отдельно. Это сократит затраты на производство и поддержку, а также позволит сразу иметь возможность монетизации через подключение услуг переноса данных с одного устройства на другое, хранения их в облаке и синхронизации устройств. Модули приложения, ответственные за этот функционал смогут даже не знать где они запущены и все равно правильно осуществлять свою работу, что позволит уменьшить кодовую базу и сделать модули более проработанными, стабильными и протестированными при тех же затратах на производство.
- Приложение должно автоматически переносить дела с одного периода на другой в конце периода. Это позволит создать хороший пользовательский опыт, облегчив пользователям использование нашего приложения и поддержку списка дел в актуальном состоянии.

- Перенесенные с прошлого периода дела должны быть помечены специальными метками, чтобы пользователь видел проблемные места в расписании.
- У пользователя должна быть возможность создавать периодически повторяющиеся дела на определенный период, которые не будут переноситься на следующий период, а отмечаться как не сделанные в случае, если пользователь не отметит из в конце текущего периода.
- У пользователя должна быть возможность просмотра статистики по повторяющимся делам за определенный периода с указанием на графике процента успешного завершения повторяющегося действия.
- Пользователь должен иметь возможность входа в приложения не авторизованным, но при этом у него должен быть ограниченный функционал.
- При авторизации у пользователя должна быть возможность перенести дела из неавторизованного режима в учетную запись.
- Пользователь должен иметь возможность приобретения услуг синхронизации данных на нескольких устройствах с использованием облачного хранилища.

Почему PWA:

Простое написание десктопного, мобильного или веб приложения дало бы опыт, но не позволило бы изучить новые прогрессивные технологии. Написание кроссплатформенного приложения дало бы интересный опыт, но таких приложений много и в этой нише тяжело найти что-то прорывное, с другой стороны новый тип приложений, такой как PWA приложения является достаточно молодой и актуальной технологией, которая работает на разных платформах без особых проблем и подходит нам как ничто иное, так как позволяет работать с приложением без наличия интернета, сохраняя при этом кроссплатформенность.

Предметная область выпускной квалификационной работы – PWA приложение.

Объектом дипломной работы является разработка PWA приложения.

Предметом квалификационной работы создание «Bullet Journal Application»

Целью выпускной квалификационной работы является разработка мобильного приложения.

.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Мобильные приложения в современном мире

Бурное развитие информационных технологий, а также стремительное распространение мобильных устройств привело к созданию и развитию рынка мобильных приложений. В последнее время мобильные приложения являются одним из главных трендов в мире IT- технологий. Мобильные приложения облегчают и разнообразят жизнь пользователям. Наблюдается постоянный рост числа программистов, занимающихся разработкой приложений, связанный с увеличение спроса компаний на разработку собственных приложений, так как в настоящее время эти приложения очень интересны пользователям. Этот перспективный рынок открывает большие возможности для IT-компаний.

Что же такое мобильное приложение? Мобильное приложение – приложение, которое разрабатывают под определённую мобильную платформу (Android, Windows Phone, iOS). Мобильные приложения могут быть установлены на мобильном устройстве или их можно загрузить из онлайн магазинов, например, Google play market, App Store, Яндекс.store, Windows Phone Store.

Разработка мобильных приложений направлена на создание приложений, предсказывающих потребительские предпочтения, решению задач, алгоритм которых неизвестен. Приложения могут проводить анализ информации, полученной из разных источников, пользователь может с помощью приложений контролировать

разные процессы и принимать решения затрачивая меньше времени и ресурсов. Мобильные приложения становятся посредниками между пользователем и потоком разной информации. Приложения бывают разной направленности:

- мультимедийные, которые расширяют возможности мобильных устройств при работе с ауди и видео информацией;
- навигационные, использующие GPS;
- социальные сети, онлайн-сервисы для общения, а также распространения различной информации:
- системные приложения, для использования дополнительных опций телефона и программного обеспечения;
- игры, которые бывают не только для развлечений, но и могут использоваться для обучения и развития;
- интернет-магазины для онлайн покупок;
- промо-приложения, для рекламы различных брендов;
- приложения службы, аналоги сайтов, организаций;
- бизнес-приложения, которые позволяют оптимизировать работу предприятия, обеспечивая мобильность, доступность информации;
- приложения-события, позволяющие транслировать разные события, в том числе и спортивные;
- контентные приложения.

1.2. Сравнение Progressive Web Apps (PWA) с сайтом и мобильным приложением

Чтобы понять, что такое Progressive Web Apps (PWA), достаточно представить, что некий сайт взаимодействует с пользователем как приложение. То есть у пользователя есть возможность установить его на любой гаджет, получать уведомления и работать с ним. Причем работа может продолжаться даже в офлайн. Эта новая модель приложения пытается объединить функции, предлагаемые большинством современных браузеров, с преимуществами мобильной работы

Progressive Web Apps-это гибрид сайта и приложения. Создать его можно как на основе существующего сайта, так и с нуля. PWA-приложение может работать как мобильное приложение, так и как PWA-сайт на рабочем столе компьютера.

- Одно из главных преимуществ PWA- его не нужно создавать для каждой операционной системы. Поэтому разработка обходится дешевле, чем разработка мобильного приложения.
- Прогрессивное приложение может работать без интернета. В офлайне пользователь может читать контент и отправлять формы. При подключении к интернету данные формы поступят на сервер.
- PWA - приложение устанавливается мгновенно. Когда пользователь заходит на сайт, данные сайта сохраняются в кеше. Благодаря этому в будущем сайт загружается быстро. Быстрая загрузка важна, так как по статистики 53% пользователей покидают сайт в течении 3 секунд, если он не загружается.
- Возможность отправлять push-уведомления.

- PWA может получать доступ к геоданным и файлам устройств, взаимодействовать с микрофоном и камерой смартфона.
- Пользователь может переходить на прогрессивное приложение во время просмотра веб-страниц из ссылок в соцсетях.

Рассмотрим, как PWA устанавливается на смартфон

Первый шаг

Пользователь, открыв свой браузер, соглашается сохранить его на главный экран.

Второй шаг

На рабочем столе смартфона появится иконка быстрого доступа. Сайт доступен пользователю.

Третий шаг

Выглядит как мобильное приложение и открывается без интерфейса браузера.

Многие продвинутые сайты, такие как web.telegram.org, avia sales.com, twitter.com. используют PWA – приложения. Прогрессивные приложения не нужно регистрировать в магазинах приложений, следовательно за них не нужно платить в App Store или Google Play, также поддерживать и продвигать их дешевле, чем мобильные приложения. В тоже время с помощью функции Trusted Web Activities (TWA) PWA приложение

можно добавить Google Play, благодаря чему сайт получает дополнительную площадку для распространения.

Рассмотрим преимущество PWA в сравнении с мобильным приложением.

Progressive Web Apps не может полностью заменить мобильные приложения, так как возможностей у нативного мобильного приложения больше. Однако же во многих случаях PWA-сайт становится более дешёвой альтернативой приложению. К преимуществам PWA следует отнести:

- Прогрессивное приложение не требует обновлений.
- PWA- небольшие по размеру по причине того, что они эффективно используют возможности браузера.
- Быстрее в разработке.
- В прогрессивных приложениях легче редактировать контент. Не нужно редактировать контент отдельно в приложении и отдельно на сайте.
- Пользоваться PWA-сайтом проще, что повышает SEO – показатели сайта.

PWA приложения востребованы, многие бренды достигли положительных результатов, воспользовавшись ими. В AliExpress на 104% выросла конверсия сайта, на 74% покупатели стали больше проводить время на сайте, BMW в три раза увеличила скорость загрузки сайта, Lancôme на 84% снизила время загрузки сайта на смартфоне и на 17% увеличила конверсию сайта. За прогрессивными приложениями будущее, они могут принести пользу в разных сферах: интернет-магазинам, салонам красоты, сервисам доставки еды и т.д.

Сравним PWA с мобильным приложение и сайтом с помощью таблиц 1 и 2.

Таблица 1- Сравнительный анализPWA с мобильного приложения

Функции	PWA	Мобильное приложение
Мгновенная установка приложения. Не обязательно использовать магазин приложений.	+	-
Скорость разработки выше	+	-
Меньше занимает места на устройстве	+	-
Стоимость разработки относительно небольшая	+	-
Развивать и поддерживать дешевле	+	-
Взаимодействие с микрофоном и камерой смартфона	+	+
Использование функций смартфона-доступ к датчикам приближения, blutooth и др.	-	+

Таблица 2- Сравнительный анализ PWA и сайта

Функции	PWA	сайт
Скорость загрузки выше	+	-
Работа в офлайн	+	-
Доступ быстрее и проще	+	-
Push- -уведомления	+	+
Более удобный интерфейс	+	-
Поддержка большим количеством браузеров	-	+

Технологии и целевые показатели представлены на рисунке 1.

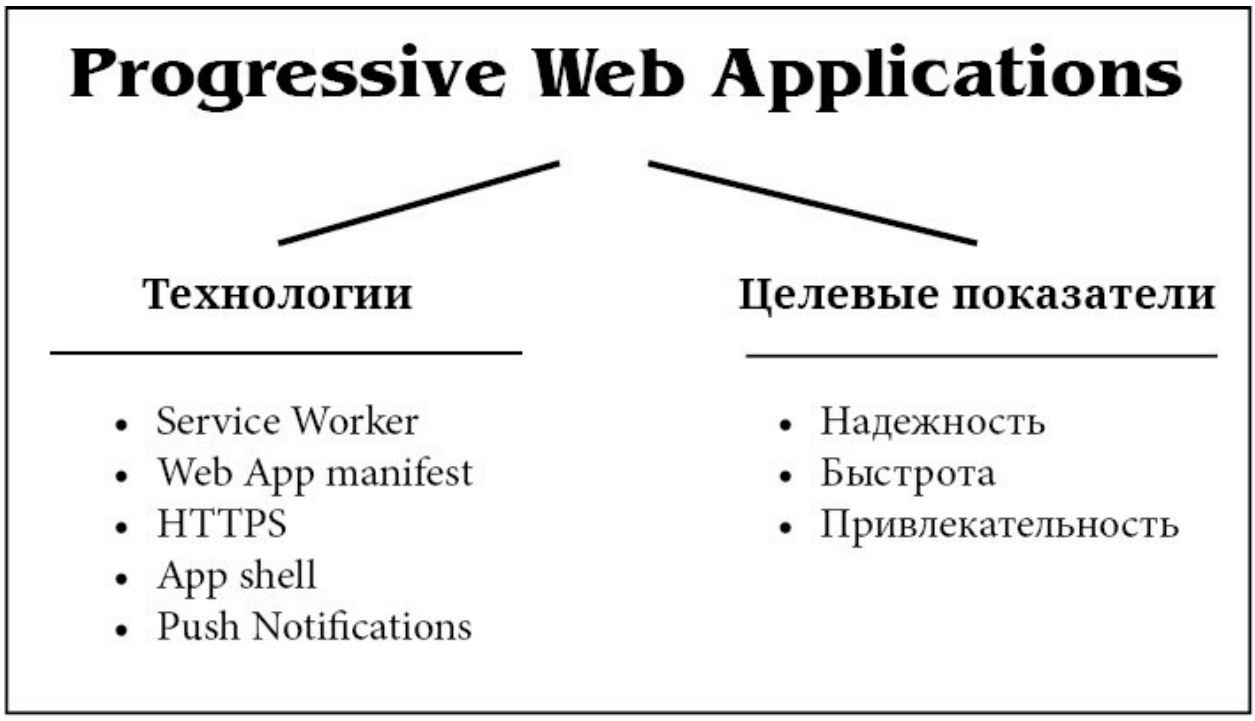

Рисунок 1 - Схема “Технологии и целевые показатели PWA”

Быстрота (Fast)- быстро происходит взаимообмен данными, UI-плавный и отзывчивый.

Надежность (Reliable)- не зависимо от статуса и качества сетевого соединения приложение загружается быстро, «не лагает».

Привлекательность (Engaging) – работать с приложением комфортно.

Service Worker- отвечает за функционирование и техническую часть PWA. Выполняет роль посредника между фронтендом и бекендом.

Расположение service worker-ов в архитектуре frontend-а представлено на рисунке 2.

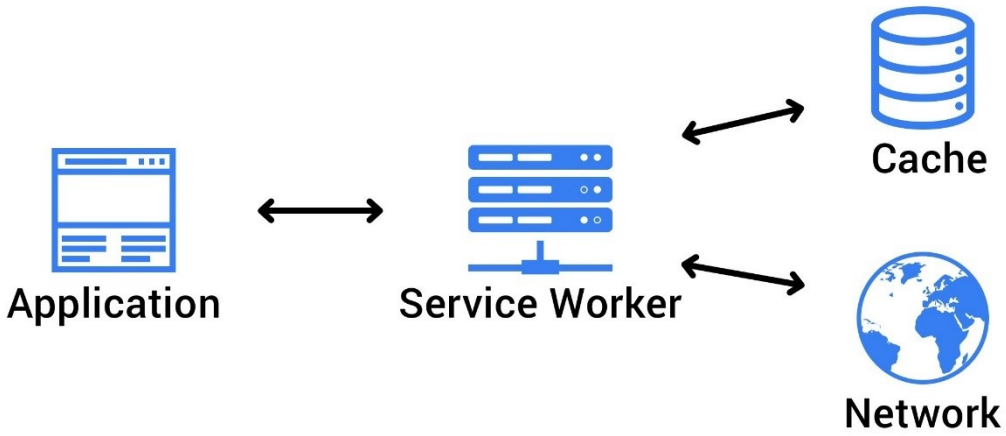

Рисунок 2 - Расположение service worker-ов в архитектуре frontend-а

Это разделение позволило максимально просто переходить из веб сайта в PWA. Через него идут все запросы браузера. У Service Worker есть доступ к Cache Strorage для web ресурсов и IndexDB для данных. Предоставляется свобода для бизнес логики. Два браузерных слоя позволяют писать полноценные приложения. Принципы работы таковы:

- приложение работает в автономном режиме;
- данные синхронизируются в фоновом режиме;

- быстродействие благодаря снижению обращений к сети, предзагрузки ресурсов;
- получение обновлений для использования несколькими частями приложения

HTTPS

Различие между HTTPS и HTTP представлено в виде схемы на
рисунке 3.

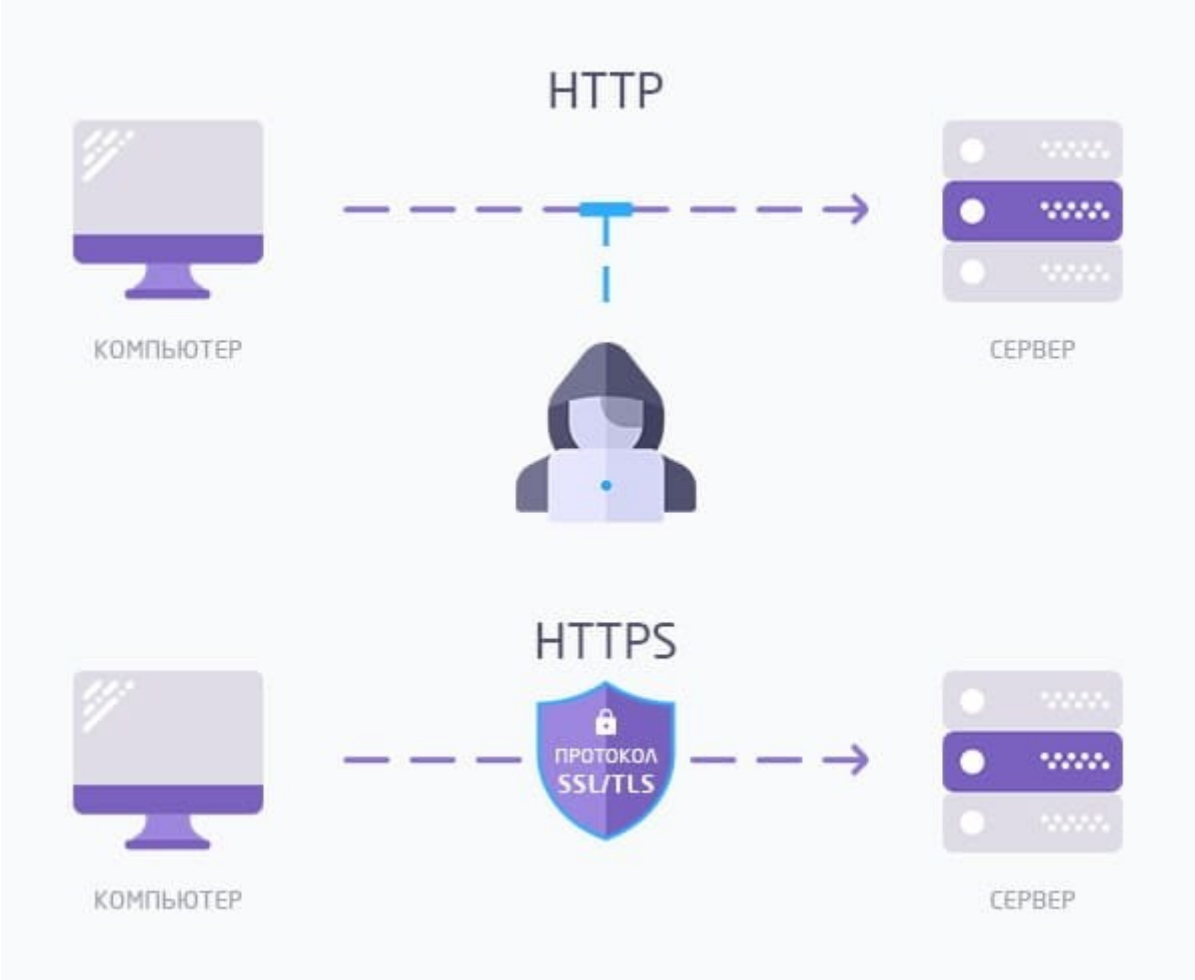

Рисунок 3 - Различие HTTPS и HTTP

Все ресурсы сайта передаются по HTTPS протоколу, что важно для безопасности.

App shell-представляет собой каркас UI и компоненты, необходимые для запуска приложения. Храниться на клиенте и загружается при запуске приложения, далее в него грузится из сети динамическая информация.

Web App manifest- JSON файл, который сообщает браузеру как будет выглядеть PWA (fullsreen, standalone и др.), наименование веб-приложения, иконки, графические объекты. а также другие параметры.

Push- оповещения (нотификации, уведомления) Push-уведомления позволяют отправить с сервера данные клиентскому PWA, даже когда оно может быть закрыто, и пользователь может не работать с ним. Из Push Service Push-уведомления отправляются в Service Worker, который обеспечивает получение уведомлений и в том случае, если приложение закрыто. Так же, как и в обычных приложениях, нотификации происходят на системном уровне. Происходит запрос на разрешение от пользователей на отправку определенных уведомлений. Оповещения должны быть персональные, должны приходить вовремя, быть уместными, краткими. Работа должна производиться вне зависимости от доступа к сети, оповещения не должны содержать рекламы.

2. АРХИТЕКТУРА ПРИЛОЖЕНИЯ

2.1. Выбор фреймворка

Рассмотрим самые популярные фреймворки. Каждый фреймворк имеет есть свои преимущества и свои недостатки, но все они служат одной цели – сделать разработку более быстрым и приятным процессом.

Разберемся чем отличаются фреймворки и библиотеки.

Библиотека – сборник различных классов и функций определенного языка программирования, содержат набор подпрограмм, которые близки по функциональным свойствам. Библиотека – это набор улит или программных модулей. Она предлагает готовое решение, которое можно использовать при разработке своего приложения.

Фреймворк определяет определенную архитектуру, которой нужно следовать. Слово «framework» означает «конструкция», «каркас», «структура». Фреймворк - программный каркас, который упрощает разработку программ и объединение компонентов, так как содержит неменяющуюся от конфигурации к конфигурации часть, заполняемую сменными моделями или точками расширения.

Фреймворк является каркасом, благодаря которому будет строится архитектура приложения, в отличие от библиотеки (DLL), которая содержит набор ограниченных функций. Кроме того, фреймворк может содержать библиотеки по разной тематике.

Фреймворки делятся на виды:

- Фреймворк программной системы;

- Фреймворк приложения;
- Фреймворк концептуальной модели.

Создать сайт можно разными способами:

- написать код с нуля,
- установка готового CMS
- использовать фреймворк

Написание исходного кода с нуля

Такой вариант дает возможность реализовать практически неограниченный функционал, дает свободу действия, но трудоёмкий, занимает много времени, может привести к большому количеству ошибок и недоработок, потребует скрупулёзного тестирования.

Установка готового CMS

CMS – это система управления контентом, набор скриптов для создания и управления сайтом. Например: WordPress, Joomla, OpenCart.

Этот вариант используют люди, плохо знающие веб-разработку и языки программирования. Этот подход позволяет быстро создать сайт, однако при создании можно столкнуться с большим количеством ограничений.

Фреймворк является компромиссом между написанием исходного кода с нуля и CMS. В нем есть каркас, но в то же время есть и функциональная гибкость.

Рассмотрим какие фреймворки существуют их основные особенности.

HTML/CSS-фремворки

Bootstrap - наиболее известный CSS-фреймворк. Основные инструменты: сетки, шаблоны, типографика, медиа, таблицы, формы, навигация, алерты.

Самое главное достоинство - адаптивность, что позволяет создать сайт с отзывчивым дизайном, динамически подстраивающимся под экран, с обеспечением правильного изображения сайта на различных устройствах. Совместим со всеми браузерами, но в старых версиях браузеров возникают проблемы. Он прост в использовании, значительно экономят время шаблоны и стили. Использует Sass и Less языки.

Foundation – один из основных front-end-фреймворков. Foundation- состоит в основном из стилей Sass, имеет модульную структуру. Имеет стартовые шаблоны, позволяющие быстрее создавать веб-проекты, подходит для быстрого прототипирования. Адаптивным макетом является сетка в 940 пикселей. Его используют Mozilla, Amazon, Adobe, eBay и др.

Semantic UI - фреймворк для создания переносимых интерфейсов. В Semantic UI 3000 настраиваемых переменных и 50 компонентов для создания сайта. Интегрирован с React, Meteor, Anqular, Ember и другими фронтенд- инструментами, понятный HTML, имеет много различных иконок, кнопок, изображений, надписей и других элементов. Этот фреймворк достаточно молод. Создатели работают над его развитием.

Uikit – фреймворк, обладает легкой и модульной структурой. Отличается от других синтаксической подсветкой для HTML и markdown (предварительный просмотр в реальном времени. Коллекция компонентов обладает унифицированной системой наименований, что обеспечивает бесконфликтность. Изменения таблицы сетки настраиваются легко, благодаря компоненту Gril, который основан на технологии Flexbox. Имеет хорошо

структурированный и открытый код, поэтому разработчик может модернизировать любую часть кода под себя. Разработан на препроцессорах Sass и Less.

Pure by Yahoo! - в фреймворк содержит небольшие адаптивные CSS-модули, которые можно для использовать в любом проекте. К этому фреймворку обращаются тогда, когда не хотят использовать слишком тяжелый программный каркас, а хотят использовать некоторые возможности фреймворка.

PHP-фреймворки

PHP-фреймворки -ускоряют разработку, в них реализованы базовые функции CRUD, нет необходимости писать запросы в базы данных. В них реализован принцип DRY, что позволяет писать меньше кода. Они удобно и легко работать, приложения легко масштабируются, лучше защищены.

Yii– фреймворк, название которого расшифровывается как “Yes, it is!”, постоянно обновляется. У него широкие возможности: одна из самых высоких производительностей, функция полного или частичного кэширования страниц, гибкие механизмы генерирования кода, обработка ошибок, миграция баз данных, возможность использовать и объединяться с jQuery, возможность оперативного моделирования прототипа проекта для предпродажной демонстрации заказчику.

Laravel – этот фреймворк имеет подробную документацию, код основан на архитектуре MVC, имеет собственную консоль Artisan, шаблонизатор Blade, установка

jQuery b BootStrap, большое количество пакетов (расширений).

Symfony – этот фреймворк часто используется для создания больших порталов, один из самых стабильных PHP-фреймворков. Имеет большой функционал, гибкий и хорошо масштабируемый фреймворк. Имеет собственный обработчик шаблонов Twig, поддерживает дополнительные форматы PHP, YAML, XML. Используется в популярных CMS (Drupal, Magento)

CodeIgniter –имеет подробную документацию по платформе, регулярные релизы новых версий, в которых исправлены баги и добавлены новые возможности, возможно использование как сторонних, так и самописных библиотек, имеет предустановленные библиотеки с большим функционалом. Поддерживает архитектуры MWC PostgreSOL БД MySOL, SOLite,MSSQL,Oracle.

Python-фрейморки

Django – самый популярный фреймворк на языке Python. Имеет большое количество библиотек и плагинов. Отличительной особенностью Django является его принцип DRY, который расшифровывается как “Don’t repeat yourself”. То есть разработчикам не следует повторять те строки кода, которые они уже использовали, поэтому исходный код выглядит лаконично и понятно. К недостаткам можно отнести поведение некоторых компонентов непредсказуемо, не подходит для небольших проектов.

TurboGears – фреймворк с богатым функционалом. Он построен на основе нескольких мощных библиотек, поддерживает множество баз данных и форматов обмена

данными. Он предназначен для разработки веб-проектов и состоит из различных WSGI-компонентов, в том числе Pylons и CherryPy.

Flask – изначально в Flask заложен только необходимый функционал, который затем можно расширять до уровня, необходимого для данного проекта. Обилие расширений может решить много задач. Использует набор инструментов Werkzeug, шаблонизатор Jinia2. Относится к микрофрейморкам.

Tornado – этот фреймворк создан для обеспечения высокой производительности, а именно способностью решить проблему 10 тысяч соединений. Неблокирующая природа сервера, позволяет легко выдерживать тысячи недлительных подключений, которые произведены в одно время. Длительная обработка запроса сводит преимущества Tornado на нет.

Web2spy – этот фреймворк с открытым исходным кодом для веб разработки, позволяет создавать динамические сайты. Он придуман с акцентом на простоту внедрения. Основывается на концепции RAD (rapid application development). Это полнофункциональный фреймворк, который содержит компоненты для всех основных функций.

Ruby-фреймворки

Ruby on Rails (RoR**)-** базируется на следующих принципах разработки:

Максимальное использование механизмов повторного использования кода, позволяющих минимизировать дублирование кода в приложении (принцип DRY формулируется как «Каждая часть знания должна иметь

единственное, непротиворечивое и авторитетное представление в рамках системы»

По умолчанию используется соглашение конфигурации, типичные для большинства приложений (принцип (или концепт) Convention over Configurator проектирования программного обеспечения который заключается в том что рассматриваемые аспекты нуждаются в конфигурации когда аспект не удовлетворяет некоторой спецификации).Фреймворк можно использовать на любой операционной системе, также он поддерживает множество СУБД.

Merb –сокращение от «Mongrel (HTTP сервер) + Erb». Фреймворк реализует архитектуру Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — т.е. модификация каждого компонента может осуществляться независимо.

Hanami–это легковесный фреймворк, который использует на 60% меньше памяти, чем другие Ruby-фреймворки. Программный каркас простой, который позволяет быстро писать гибкий код, который в дальнейшем можно легко отредактировать.

Sinatra – представляет собой небольшое и гибкое приложение, которое не следует Model-View-Controller. Фреймворк с открытым и свободным исходным кодом, который используется для создания веб-приложений. Преимущество этого фреймворка перед Ruby on Rails –

стабильность, так как фреймворк практически не меняется, разработчик полностью контролирует свой код.

Java-фреймворки

Spring– универсальный фреймворк для Java-платформы. Spring благодаря широкой функциональности позволяет решить множество задач, стоящих перед разработчиками. Он обладает расширениями нужными для сложных бизнес-приложений, но и может использоваться при создании небольших проектов. Spring - собрание разных фреймворков, которые можно использовать независимо друг от друга. У фреймворка есть собственная платформа MVC для веб-приложений. Spring позволяет обеспечить проект хорошей масштабируемостью, возможностью простого тестирования и упрощённой интеграцией с другими фреймворками Spring – модель разработки, основанная на лучших стандартах индустрии.

JavaServer Faces (JSF) –фреймворк облегчает процесс построения компонентно-ориентированных клиентских интерфейсов для приложений на Java EE. Классы компонентов интерфейса пользователей содержат функциональный компонент, появляется возможность рендеринга на различных устройствах клиента. JSF дает возможность создавать свои компоненты, но можно воспользоваться и уже имеющимися. При желании также можно использовать MVC-подход. Для удобства JSF представляет библиотеку JSP-тегов для рендеринга на HTML.

Play – этот фреймворк использует паттерн проектирования MVC, написан на двух языках, Scala и Java. Нацелен на повышение производительности, для этого в нем

реализована компилируемость. Имеет строгую статическую типизацию кода в шаблон не могут быть переданы параметры неправильного типа или URL без контроллера.

Vaadin – использует Java в качестве единственного языка программирования при создании веб-контента, который подходит для создания насыщенных интернет-приложений (RIA - rich Internet application). Фреймворк использует событийную модель, виджеты, определенные элементы пользовательского интерфейса, что близко к разработки настольных приложений на Java с использованием HTML и JavaScript. Построение виджитов и модели данных позволяет отображать большие объемы данных в браузере без значительной загрузки оперативной памяти. Vaadin использует Google Web Toolkit для того, чтобы отображать компонентов клиентского интерфейса и взаимодействовать с сервером на стороне пользователя.

Google Web Toolkit (GWT**)** – этот фреймворк позволяет писать и отлаживать AJAX – приложения на языке Java, который затем будет переведен в JavaScript. При создании программного продукта разработчики уделили особое внимание скорости работы веб-приложений. Это достигается путем использования отдельно загружаемых модулей, на которые разделяется веб-приложение. Отладка GWT-приложения разделена на две части, а именно: отладка серверной части приложения осуществляется как отладка обычного Java web-приложения, для отладки клиентской части необходим gwt dev-plugin для браузера.

JavaScript-фреймворки

AngularJS – один из самых востребованных и популярных фреймворков в мире. Основное предназначение данного фреймворка состоит в разработке одностраничных приложений. К преимуществу AngularJS следует выделить декларативный стиль кода, который позволяет сделать код более легковесным. Улучшает тестируемость кода, отделение DOM – манипуляций от логики приложения. Позволяет параллельно вести разработку, разделение клиентской и серверной части. Фреймворк позволяет использовать директивы, которые позволяют уделить особое внимание логике и повысить читаемость кода. Можно использовать готовые модули или создавать приложения из зависимых или автономных модулей.

Ember.js – предназначен для упрощения создания масштабируемых одностраничных приложений. Маршруты, каждому из которых соответствует своя модель, являются основополагающим принципом. В моделях содержится информация о текущем состоянии приложения. Преимущество - отсутствие необходимости писать вспомогательный код, наличие хорошего обработчика путей и модуля для работы с данными.

jQuery – это библиотека, в которой большое внимание уделено взаимодействию HTML и JavaScript Она помогает решать проблемы кроссбраузерности и использования Ajax. Эта библиотека с высокой производительностью, имеющая большое количество плагинов.

Backbone.js – библиотека JavaScript, основана шаблоне Model-View-Presenter. На базе этой библиотеки можно строить свои фреймворки или использовать уже

существующие: Vertebrae, Marionette, Thorax, и другие. В ней понятная документация, малый вес,несложный код.

Polymer – библиотека готовых веб-компонентов, которые свободно компонуются. Она включает в себя стандартные polyfill-сценарии, одинаково ведущие себя во всех современных браузерах.

React-самая продвинутая библиотека JavaScript, которая используется для создания веб-приложений. Особенно ее хорошо применять при создании одностраничных приложений. Использование Virtual DOM позволяет повысить производительность. Изоморфные приложения используют один и тот же код в серверной части и клиентской, отпадает дублирование одного и того же функционала. Код, написанный для создания сайта, можно использовать при создании мобильного приложения.

При выборе фреймворка опираемся на такие факты: архитектура PWA приложений подразумевает, что приложение должно работать без подключения к интернету, как нативное десктопное приложение, плюс мы хотим, чтобы на мобильных устройствах и компьютере использовался один и тот же клиент. Для этого мы будем использовать браузер и PWA приложение. Чтобы в браузере была возможность работы без подключения к интернету нам нужно использовать service worker. Они будут выступать как прослойка между сервером и клиентом, также мы будем использовать локальные хранилища данных. Получается у нас будет толстый клиент и serverless архитектура. Из-за того, что большинство операций должны выполняться на клиенте даже при отсутствии интернета, одним из лучших

вариантов для реализации serverless архитектуры служат облачные сервисы. Они представляют возможность работы с базой данных через уже настроенный API и лямбда функции, которые помогают реализовывать сайд эффекты для базы данных. Провайдером облачного сервиса было решено выбрать Firebase, потому что имеется возможность легкой интеграции с приложением и удобные инструменты разработки и отладки, а также собирает всю необходимую статистику по пользователям и crash аналитику. Так как мы используем serverless архитектуру решения использующие PHP, Ruby, Python и Java нам не подходят и являются излишними. Также нам не подходят CMS системы, потому что нам нужно более продвинутое в функциональном плане приложение. Остается выбор между фреймворками написанными на JS. Среди них наиболее популярны на сегодняшний день React, Angular, Vue. Разделим их на 2 категории. Фреймворк представляет инструменты и навязывает их нам, в то время как библиотека компонентов представляет удобный высокоуровневый интерфейс/обработку под нативным JS/ HTML/ CSS и свободу выбора инструментария. Для нашего приложения не нужна вся мощь Angular-a, так как он самый тяжелый из всех и размер его бандла в среднем в 2-3 раза больше, чем аналогов. Следовательно его можно смело откинуть. В то же время преимущество React, Vue является virtual-DOM. Virtual- DOM - эта технология, которая позволяет значительно оптимизировать работу с браузером DOM деревом за счёт уменьшения количества отрисовок HTML элементов, благодаря механизму реконсиляции. Среди React, Vue было

решено выбрать React, из-за большего сообщества и того, что React разрабатывается огромной командой Facebook-а, Vue- разрабатывается по большей части одним разработчиком.

2.2. Выбор языка программирования

Среди существующих языков программирования для написания PWA больше всего подходит JavaScript, и, в частности, его суперсет TypeScript, так как он нативно поддерживается во всех браузерах, а одной особенностью PWA приложений является возможность работы в любой среде. Из аналогов существуют языки, которые можно запускать в браузерах при помощи web assembly, но это является слишком громоздким решением для данного приложения, так как потребует намного больших сил для настройки и подключения, а также даст излишний функционал, который насм не требуется. К тому же JavaScript просто идеален для прототипирования.

2.3. Использование облачных сервисов

Выбранный тип приложения, а именно PWA, требует работы без интернет-соединения, поэтому нам нужно предусмотреть такие моменты, как полноценная работа без связи с сервером, возможность синхронизации с базой данных при восстановлении интернет-соединения. Для того чтобы приложение могло работать без сервера нам нужно перенести большую часть, если не всю, бизнес логику

приложения на сторону клиента, тем самым сделав клиент толстым, то есть самостоятельным.

В то же время нам необходимо как-то настроить механизм синхронизации данных приложения с базой данных. Для этого нам придётся хранить историю изменений данных на стороне клиента до тех пор, пока отсутствует соединение с сервером, чтобы потом была возможность в синхронизации данных клиента с данными в облаке. Хранить всю историю все же не лучший вариант, поэтому было принято решение о введении дополнительной оптимизации для данной функциональности, которая позволит хранить не всю историю изменений, а лишь разницу данных клиента и базы данных. Это оказалось достаточно сильным усложнением клиентской части приложения, что повлияло на решение отказаться от сервера вовсе, так как в итоге почти все операции пришлось перенести на сторону клиента и дублировать их же на сервере было бы не лучшей идеей.

Такой подход к архитектуре приложения, при котором отсутствует сущность, именуемая сервером, называется serverless. Для реализации данного подхода используются облачные сервисы, одним из которых является firebase. Модель используемой serverless архитектуры представлена в виде схемы на рисунке 4.

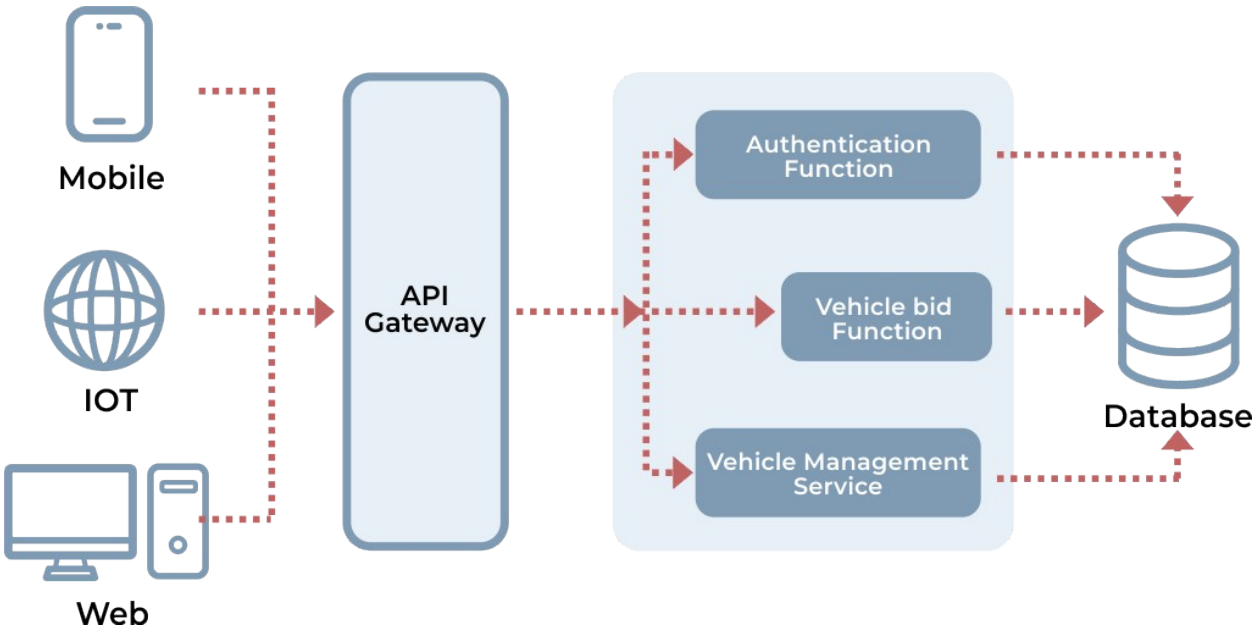

Рисунок 4 - Serverless архитектура приложения

Облачные сервисы помогают хранить данные в облаке и предоставляют удобный API для взаимодействия с базами данных, что сильно помогает при разработке приложений и позволяет отказаться от сущности сервера в архитектуре. Визуализация принципа работы облачного сервиса представлена на рисунке 5.

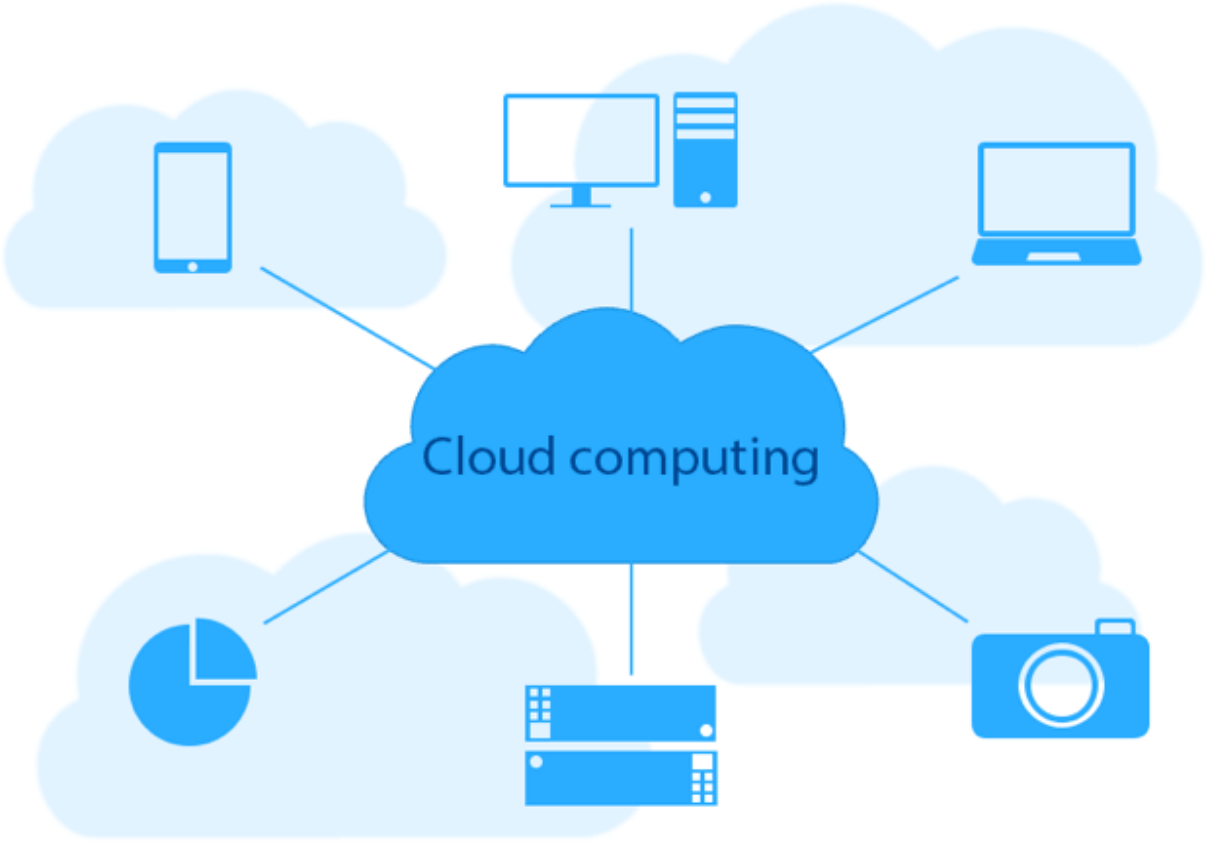

Рисунок 5 - Визуализация работы облачного сервиса

2.4. Serverless архитектура

Для написания приложения была выбрана serverless архитектура, что означает отсутствие сервера в приложении. Вместо этого было принято использовать облачные технологии и лямбда функции. Принятие данного решения позволило значительно ускорить разработку, так как облачные сервисы предоставляют огромный инструментарий для взаимодействия с базами данных со стороны клиента. Мы можем использовать API точко доступа для взаимодействия с базой данных напрямую или более мощные средства, такие как лямбда функции.

Помимо уже готового инструментария для работы с базами данных такой подход позволяет не заботиться о масштабируемости приложения, так как мощный облачный сервис позволяет запустить столько копий наших функций, сколько требуется для обслуживания текущего трафика.

Так же одним из преимуществ является изолированность выполнения кода, который обрабатывает запросы, так как обработка запроса происходит в отдельном экземпляре приложения, изолированном от другого кода. Здесь важно понимать, что запускается не все приложение, а только те функции, которые нужны для выполнения запроса.

Отсюда вытекают следующие проблемы, с которыми приходится считаться:

- во-первых, для запуска отдельного приложения с облачными функциями требуется время, что негативно сказывается на скорости обработки запросов;
- во-вторых, необходимо заботиться об обратной совместимости, так как от любой функции может зависеть другая;
- в-третьих, когда происходят сложные запросы с участием множества функций, то отладка багов занимает кучу времени и сил.

В результате анализа существующих недостатков выяснилось, что на проектируемое приложение они не оказывают практически никакого влияния. Что касается долгих запросов к базе, то это нивелируется тем, что у нас приложение лишь изредка делает запрос к базе данных для синхронизации в фоновом режиме, что никак не сказывается на пользователях. В отношении сложных запросов с

использованием нескольких функций удалось добиться оптимизаций, при которых не вызывается больше 2-х функция на операцию. Так же покрытие тестами облачных функций значительно повысило их надежность. Для того, чтобы добиться хорошей обратной совместимости, пришлось заранее продумать основные функции, которые легли в основу всей бизнесс логики, и в итоге это лишь сыграло на руку, так как до разработки облачных функций уже было четко сформированное представление об их структуре и поведении, что позволило написать намного более качественный и продуманный наперед код.

2.5. Лямбда функции

При использовании serverless архитектуры и облачных функций по началу не встречается практически никаких проблем, так как в нашем распоряжении сразу оказывается удобный API для работы с базой данных со стороны клиента. Но со временем появляются проблемы с тем, что предоставляемый API не слишком гибок, так как он учитывает много факторов для защиты приложения и совершать сложные операции с базой данных, сохраняя их безопасность, становится проблематично.

При наличии сервера таких проблем как правило не возникает, так как сервер может получить полный доступ к базам данных как суперпользователь и распоряжаться ими как угодно. Сервер переносит ответственность за сохранность данных на себя и предоставляет более удобный API для клиента, чем могут предоставить облачные сервисы,

которым нужно заботиться не только о гибкости и удобности их API для массового пользования, но и не забывать о безопасности.

Здесь нам на помощь и приходят лямбда функции, которые выполняются на стороне облачного сервиса в любой момент, когда мы захотим и имеют административные права. Они позволяют производить более сложные и легкие в программировании операции над базой данных на стороне облачного сервиса. По своей сути они близки к тому, что делал бы сервер, но имеют ряд преимуществ.

К преимуществам лямбда функций относится возможность не держать постоянно включенным сервер, так как они запускаются по надобности, выполняют свою работу и потом экземпляр функции уничтожается, то есть мы, как владельцы приложения, платим только за время работы функции, а не за огромный сервер, который бы работал круглосуточно, потребляя огромные ресурсы, даже если он был бы нам не нужен.

Еще одним преимуществом облачных функций является то, что нам не надо заботиться о масштабировании серверных мощностей, так как облачный сервер запускает столько функций, сколько нужно для обработки клиентских запросов. Визуализация масштабирования сервисов, путем создания новых подобных сущностей представлена на рисунке 6.

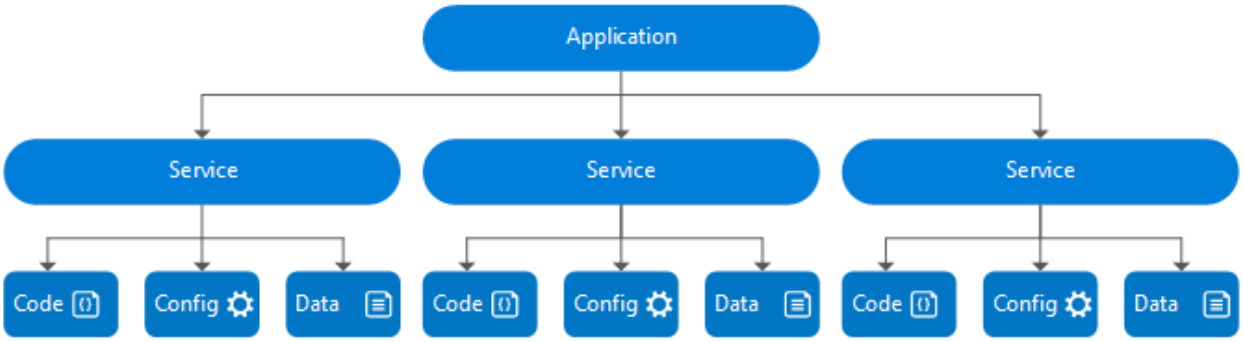

Рисунок 6 - Масштабирование сервиса созданием клона сущности сервиса

Если вдруг резко возрастет трафик приложения, нам не понадобиться сломя голову бежать досказывать новые мощности и настраивать балансировщик, чтобы приложение продолжало стабильно работать. Изменится лишь количество запущенных функций и вырастет суммарное время работы функций, что отразится на ценнике за услуги сервиса и все. В то же время при резком снижении трафика нам не придётся платить за простаивающие сервера.

Так как мы платим за время работы лямбда функций, их оптимизации является важным аспектом в разработке приложения. Есть ряд часто встречающихся ошибок, которые приводят к сильному росту всей работы подобных функций, которые были учтены в данном приложении. Одной из таких проблем, решению которой было уделено пристальное внимание, была работа с промисами. Промысы в JavaScript позволяют управлять асинхронными операциями, так как язык является однопоточным, и, для того, чтобы главный тред не останавливался, существуют асинхронные операции, такие как таймеры или запросы. По своей природе лямбда функции асинхронны и для взаимодействия с базой данных

используют асинхронные запросы. Здесь мы и встречаемся с проблемами. При не правильном использовании промисов в лямбда функциях они подвисают на неопределенное время после окончания выполнения и тем самым растягивают свое время работы, что сильно отражается на их стоимости. Для того чтобы это не происходило нужно следить, чтобы они всегда резолвились при завершении выполнения, тогда сервис будет точно уверен, что пора завершать работу функции и не будет тратиться лишнее время на таймаут, через который облако решит, что функция работает не правильно и принудительно завершит ее работу.

К тому же очень важно настроить функции так, чтобы в период нормальной работы они выполнялись без ошибок и с максимальной скоростью. Для этого были применены следующие шаги: добавлено тестирование на этапе разработки и оптимизированы алгоритмы для совершения минимального числа запросов к базе данных.

2.6. FireBase для упрощения разработки

Одним из наиболее популярных вариантов реализации облачного сервиса является firebase. Среди прочих аналогов, таких как сервисы от Amazon, он подкупил тем, что имеет хорошо написанную документацию и отзывчивое сообщество, а также отличные тарифные планы, которые позволяют вести разработку приложения с использованием данного сервиса совершенно бесплатно. Firebase предоставляет нам полностью готовый API для работы с новейшей базой данных, что немаловажно для быстрого прототипирования

приложения, а также отлично подходит для написания самостоятельного PWA приложения, так как позволяет полностью отказаться от сущности сервера, как прослойки между клиентом и базой данных.

Firebase предоставляет необходимые для текущей архитектуры приложения облачные функции и удобные механизмы для из разработки, поддержания и отладки. Для разработки лямбда функций существует хорошо описанный набор SDK, который предоставляет понятную и тестируемую среду для разработки лямбда функций. К тому же можно пойти еще дальше и настроить среду таким образом, чтобы тестирование шло с полностью функционирующей копией существующей базы данных, и получать результаты выполнения тестируемых функций, как древовидную разницу базы данных.

Помимо удобного механизма тестирования и разработки, предоставляется возможность достаточно простой и интуитивной настройки выгрузки готовых функций в облако, где они незамедлительно начнут свою работу. Это сильно ускорило разработку данного приложения.

Для уже готовых выгруженных функций firebase предоставляет удобные механизмы сбора статистики и аналитики. С их помощью мне удалось оптимизировать работу написанных лямбда функций, снизить время их выполнения и тем самым сократить расходы на использование сервиса.

Так же при необходимости замены или изменения существующих функций есть готовый для этого механизм все в том же SDK. А в самом приложении есть интерфейс для

удобного управления существующими функциями, который позволяет отключать проблемные или менять версию в любой момент, при настроенном механизме версионирования.

Помимо SDK для разработки облачных функций firebase предоставляет удобную библиотеку для интегрирования сервиса в React приложение. Для этого нам нужно добавить пакеты в зависимости для сборщика, в нашем случае webpack, проинициализировать приложение и задать ему публичные ключи для нашего приложения, зарегистрированного в сервисе firebase. Публичные ключи будут храниться на стороне клиента и это совершенно безопасно. После инициализации библиотека firebase предоставляет возможность работы с публичным API нашего приложения и дает возможность настроить сервис регистрации, который будет давать доступ в приватному API приложения.

Настройка сервиса регистрации заключается в подключении дополнительного плагина в клиентском приложении и настройки возможных путей регистрации в WEB интерфейсе сервиса firebase. Во время настройки можно указать параметры, по которым будет производиться валидация пользователей и настроить дополнительные способы регистрации, а также усилить безопасность пользователей, настроив возможность привязать почту или мобильный телефон к аккаунту. После настройки данного сервиса у нас, как администратора приложения появляется возможность управлять существующими пользователями, смотреть их статистику и удобно управлять правовой моделью пользователей.

Firebase аналитика по пользователям поможет при разработке после первой доставки приложения пользователям. В ней будут собираться данные о скорости загрузки приложения для разного вида устройств, возможные неполадки, с которыми столкнулись пользователи и на каких устройствах чаще всего запускается приложение. Эти данные помогут сконцентрировать усилия на узких местах приложения, для увеличения дружелюбности к пользователям в последующих версиях.

2.7. Использование service worker-ов в приложении

Так как PWA приложение по своей сути является одностраничным, то добавить к нему service worker для работы при отсутствии интернет соединения достаточно просто. Интерфейс приложения кэшируется при первом запуске. Файлы и дополнительные данные закачиваются по первому запросу и проблем с этим не возникает. Могут возникать проблемы при неправильной настройке service worker-ов с их обновлениями или удалением, но мне удалось избежать подобных проблем, так как в результате изучения информации по ним я столкнулся с инструкцией по правильной настройке и провел несколько дополнительных этапов тестирования, чтобы убедиться в правильности их настройки.

Из особенностей service worker-ов можно отметить, что у них нет доступа к хранилищу localStorage, поэтому здесь мне пришлось сделать исключения для принципа разделения использования локальных хранилищ, который я соблюдал при

построении остальной архитектуры, при котором локальные данные хранятся в localStorage, а indexedDB заведуется только firebase библиотека.

Так же для настройки возможности добавления пользователем приложения к себе на рабочий стол была произведена настройка manifest.json и Meta data.

2.8. Тестирование

Для того, чтобы быть уверенным в работоспособности текущей версии приложения в процессе разработки было принято решение о добавлении автоматического тестирования, так как с ростом кодовой базы стало трудно отлавливать ошибки, к которым могли привести добавляемые в код изменения. В результате изучения этой проблемы было принято решение использовать пирамиду тестирования, состоящую из unit, интеграционных и e2e тестов. Иерархия тестирования представлена на рисунке 7.

Русунок 7 - Пирамида тестирования

Первым этапом добавления тестов в приложение стал этап добавления unit тестов, так как это один из самых быстрых видов тестирования. Для проведения unit тестов был выбран фреймворк jest, из-за его популярности и можного функционала, позволяющего писать очень выразительные тесты. Главной особенностью unit тестов, является то, что мы тестируем одну определунную сущность и ее поведение, а любые ее взаимодействия с другими сущностями мы мокаем (подставляем вместо других сущностей болванки). Покрытие unit тестами утилитарных сущностей, используемых по всему проекту в коде, дало значительный прирост в стабильности приложения. Теперь, при изменении в этих сущностях, которые влияли на их поведение, это можно было с легкостью отследить и предугадать как это отразится на

поведении приложения. Подобные изменения нежелательны, но все же иногда необходимы.

После покрытия unit тестами всей бизнес логики пришло время браться за следующую ступень тестирования, это интеграционные тесты. Для интеграционных тестов был выбран фреймворк Enzyme, из-за того, что он идеально подходит для тестирования приложений, написанных на React, так как предоставляет множество гибких инструментов, позволяющих отрендерить компоненты максимально эффективно для нужного вида тестирования, а также был использован jest для тестирования остального функционала. Отличие интеграционного от unit тестирования заключается в том, что при интеграционном тестировании мы уже не мокаем данные, а тестируем взаимодействие между сущностями или же модулями. Этот этап пирамиды тестирования более медленный из-за того, что среде приходится работать не с одной, а несколькими сущностями и не исключено, что в результате тестирования могут идти запросы, занимающие намного больше времени, чем прогон функции при unit тестировании.

Финальным же этапом стало добавление e2e тестов. Для e2e тестов был выбран cypress, так как практически без настроек позволил развернуть мощные e2e тесты. Как оказалось это самые долгие и тяжелые тесты и в них было принято решение тестировать только самые важные сценарии. E2e тестирование — это тестирование пользовательского интерфейса, путем прохождения написанных сценариев. Это напоминает ручное тестирование, но только вместо человека сценарии проходит

робот. Низкая скорость подобных тестов обуславливается тем, что среде тестирования приходится для каждого теста запускать полноценное приложение и проходить пользовательские сценарии как это делал бы пользователь, что, разумеется, намного дольше, чем те же интеграционные тесты, которые проходят в облегченной среде всего с несколькими сущностями.

В итоге удалось добиться достаточно высокой уверенности при написании кода, что новый код не ломает ничего из уже существующего функционала, так что временные затраты на написание тестов окупились сэкономленным временем на отладку и ручной проход сценариев.

2.9. CI/CD Доставка пользователям

Для того чтобы выпуск новых версий стал возможным нужно изначально правильно подготовить среду разработки. А именно настроить возможность интеграции и доставки пользователям, подобные настройки называются настройками CI/CD, а именно continuous integration и continuous delivery. Правильно настроенный процесс CI/CD позволит без проблем добавлять новый функционал в приложение, обновлять или исправлять уже существующий. Визуализация процесса непрерывной интеграции и доставки представлена на рисунке 8.

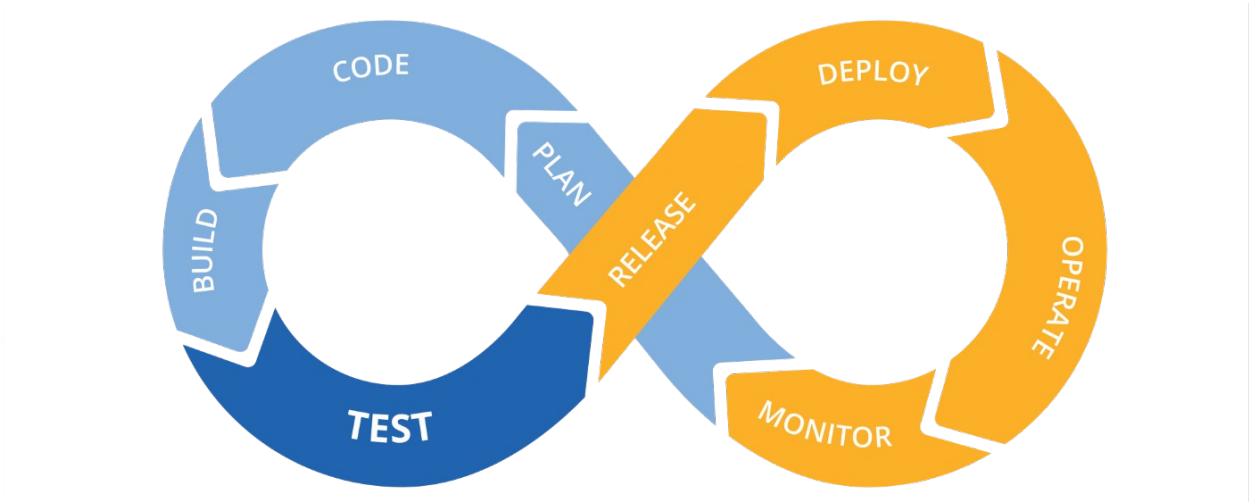

Рисунок 8 - Визуализация процесса CI/CD

Настройка этого процесса с нуля вручную займет уйму времени даже у продвинутого devOps специалиста, поэтому было принято решение использовать пред настроенные реализации.

Неплохим вариантом оказался Gitlab. Он предоставляет возможности внедрить в существующий git репозиторий функционал CI/CD без особых проблем. Описание работы CI/CD в гитлабе представлена на рисунке 9.

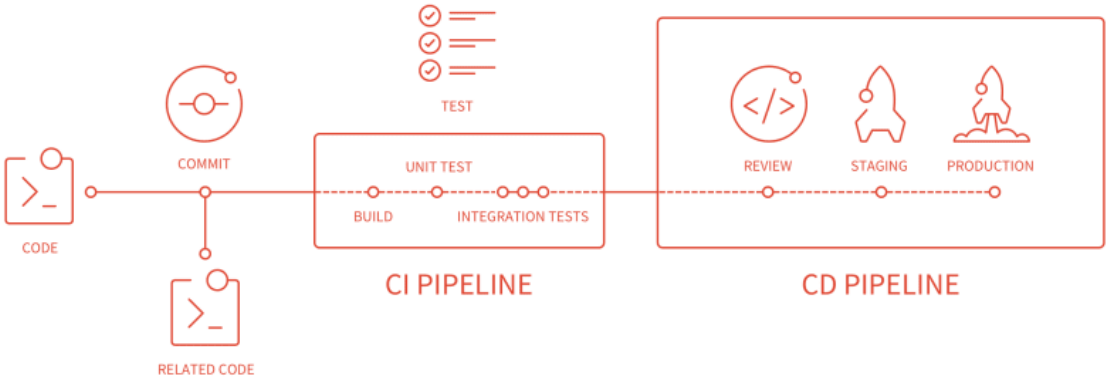

Рисунок 9 - Схема процесса CI/CD в gitlab

Мне для этого понадобилось создать небольшой конфиг примерно на 150 строк, который позволил обеспечить возможность непрерывной доставки продукта, а именно PWA приложения, конечному пользователю. Настроенный процесс доставки и интеграции позволяет автоматически тестировать сборку и функциональность приложения при добавлении нового кода и является последним барьером, позволяющим предотвратить катастрофу попадания нерабочего кода к потребителю. Без настроенного механизма непрерывной доставки есть большая вероятность, что, либо к пользователю попадет сломанное приложение, либо процесс тестирования и отладки займет недели. Настроенный процесс непрерывной интеграции позволяет постоянно тестировать малейшие изменения и сразу выявлять проблему, что ускоряет ее исправление и скорость разработки.

3. ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ ПОСТРОЕНИЯ ПРИЛОЖЕНИЯ

3.1. React

Используем реакт ^16.8.0 версии, из-за внедренного в него с 8 минорной версии движка fiber, которые значительно оптимизирует работу с virtual DOM и позволяет рендерить намного более производительно сложные анимации и в принципе ведет себя более оптимизированно, по крайней мере он так ощущается пользователем.

Новый движок отрисовки позволяет нам более гибко работать с отрисовкой сложных HTML элементов, таких как виртуальный список или же функционал перетаскивания HTML элементов. Оптимизация происходит путем уменьшения количества перерисовок, благодаря продвинутому механизму реконсиляции, который способен определять изменился ли контент элемента и в зависимости от этого вызывать перерисовку элемента или оставлять его старый вид.

Использование React последней версии позволило оптимизировать приложение для слабых мобильных устройств без особых усилий. Для этого необходимо грамотно расставлять метки-подсказки для механизма реконсилсяции, чтобы он мог гораздо эффективнее выполнять свою задачу. К таким меткам относятся параметры key для элементов массивов и указание чистых компонентов, а в некоторых случаях добавление своей реализации функции сравнения

состояния с предыдущим. Схема работы virtual DOM представлена на рисунке 10.

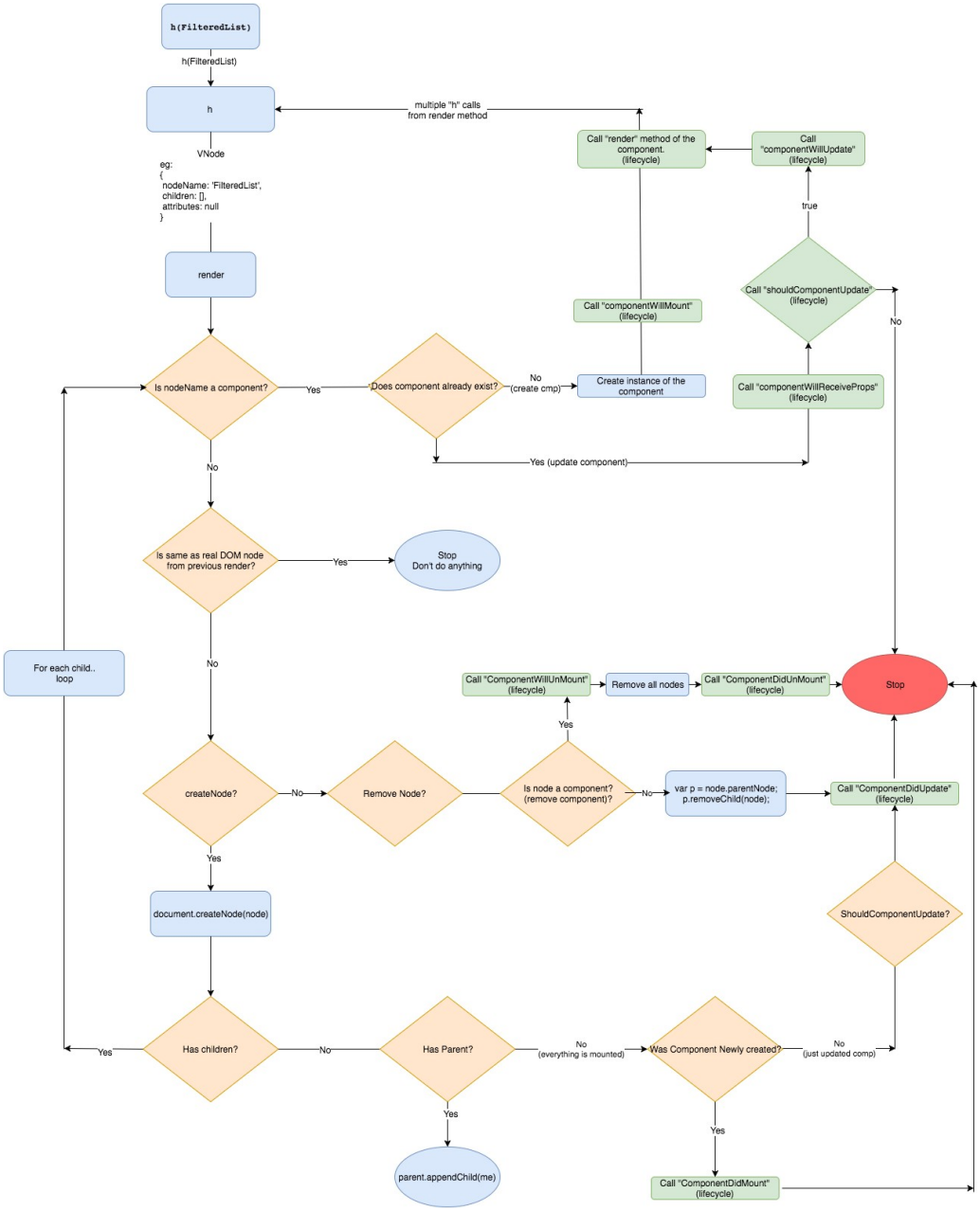

Рисунок 10 - Схема virtual DOM

3.2. Redux

Для работы с данными приложения используем паттерн redux, в частности его имплементацию react-redux. У нас ожидается serverless архитектура и нам необходимо много сложных операций, при отсутствии интернет-соединения, делать на фронте и хороший стор будет кстати. В двух словах

данный паттерн позволяет управлять потоками данных превратив их в один, что делает его (поток) намного менее хаотичным и более управляемым. В своей реализации паттерн redux использует такие паттерны как observer и singleton. Observer позволяет нам диспачить действия в стор, которые приводят к определенным изменениям и подписываться на эти изменения в сторе. А singleton всего лишь следит за тем, чтобы у нас был один единственный экземпляр стора в приложении. Есть паттерны, такие как flux, позволяющие работать похожим образом с данными при нескольких сторах, но redux считается более простым и прогрессивным решением, лишенным недостатков flux-а. Схема текущей реализации redux паттерна представлена на рисунке 11.

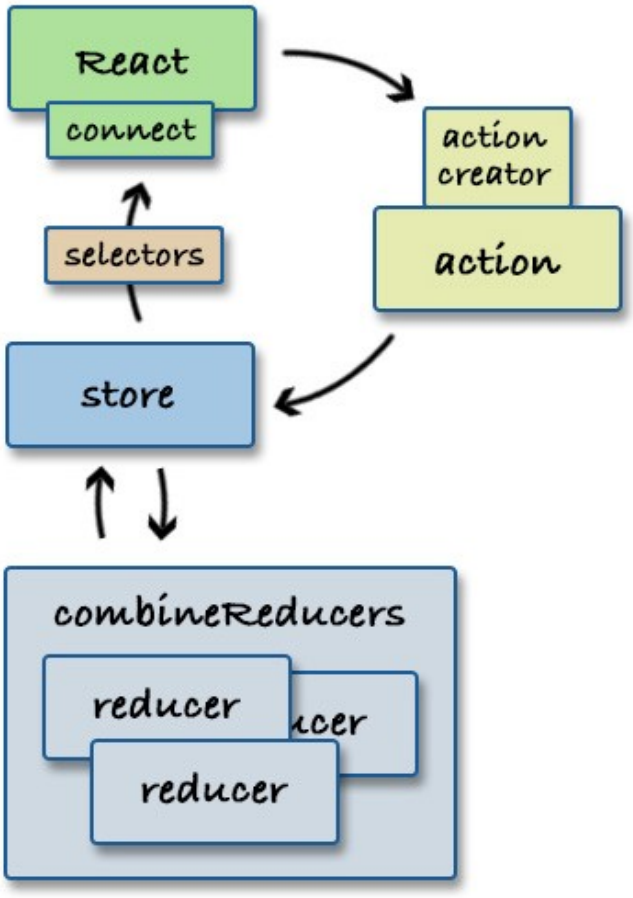

Рисунок 11 - Схема реализации redux паттерна

3.3. React-Router

Как и любое другое SPA, наше не может обойтись без роутинга. Здесь особого выбора нет, потому-как react-router, пожалуй, единственное доступное популярное решение. Роутер позволяем нам легко и гибко управлять отображениями приложения в зависимости от роута в браузере, перенаправлять пользователя по приложению и блокировать нежелательные переходы. Фрагмент кода приведен в Приложении Б.

3.4. Самописный D&D

Для реализации функции drad and drop, нужной нам для обеспечения пользователю возможности просто и интуитивно управлять списками дел в приложении решено было использовать свою реализацию по нескольким причинам.

Во-первых, хочется облегчить приложение по максимуму, так как оно будет храниться у пользователя локально и не хочется занимать у него много места, так же хочется, во-вторых, чтобы новые пользователи не сталкивались с проблемами долгого ожидания получения первого пользовательского опыта. К тому же своя реализация такого функционала сделает его более гибким и

оптимизированным для нашего приложения. Фрагмент кода представлен в Приложении В.

3.5. Самописный Virtualized list

Для реализации виртуальных списков тоже было решено выбрать свою реализацию, что даст нам скорость работы и загрузки приложения, при этом не увеличивая, а даже уменьшая его вес. В сумме все рукописные решения для реализации необходимой функциональности помогут добиться неплохого прироста в таких параметрах как:

- производительность;
- снижение веса приложения;
- гибкость архитектуры уровня React компонентов.

Фрагмент кода приведен в Приложении Г.

3.6. Typescript

JavaScript - это язык с динамической типизацией, что полезно для написания небольших приложений, но у нас приложение в себе содержит несколько самописных библиотек, сложные модели данных, которые нам приходится обрабатывать на клиенте, так как нужно обеспечить возможность работы при отсутствии интернет соединения, а так же использует сложные и объемные сервисы firebase, что нивелирует все преимущества возможности быстрого прототипирования на JavaScript из-за необходимости ручной проверки типов для нашей сложной работы с данными. Нам просто необходима статическая типизация, чтобы не завязнуть в вечной отладке приложения.

Из вариантов статической типизации для приложения на React у нас существует Flow и TypeScript. Было решено остановиться на TypeScript, так как Flow менее популярен и более сложен, хоть у него и есть типизация в рантайме, но сложность написания на нем сильно замедлила бы разработку, что не желательно при прототипировании приложений. К тому же TypeScript намного более выразителен, что касается работы со сложными типами данных как у нас. Фрагмент кода приведен в Приложении Д.

3.7. Webpack

Для сборки приложения было решено использовать Webpack. Это наиболее мощное и популярное решение дна текущий момент. С нуля настройка сборки может занять приличное время, поэтому было решено использовать npx create-react-app, которые позволяет развернуть готовое к разработке приложение на реакте и легко подключить TypeScript для статической типизации и Jest для тестирования. Схема сборка бандла вебпаком представлена на рисунке 12.

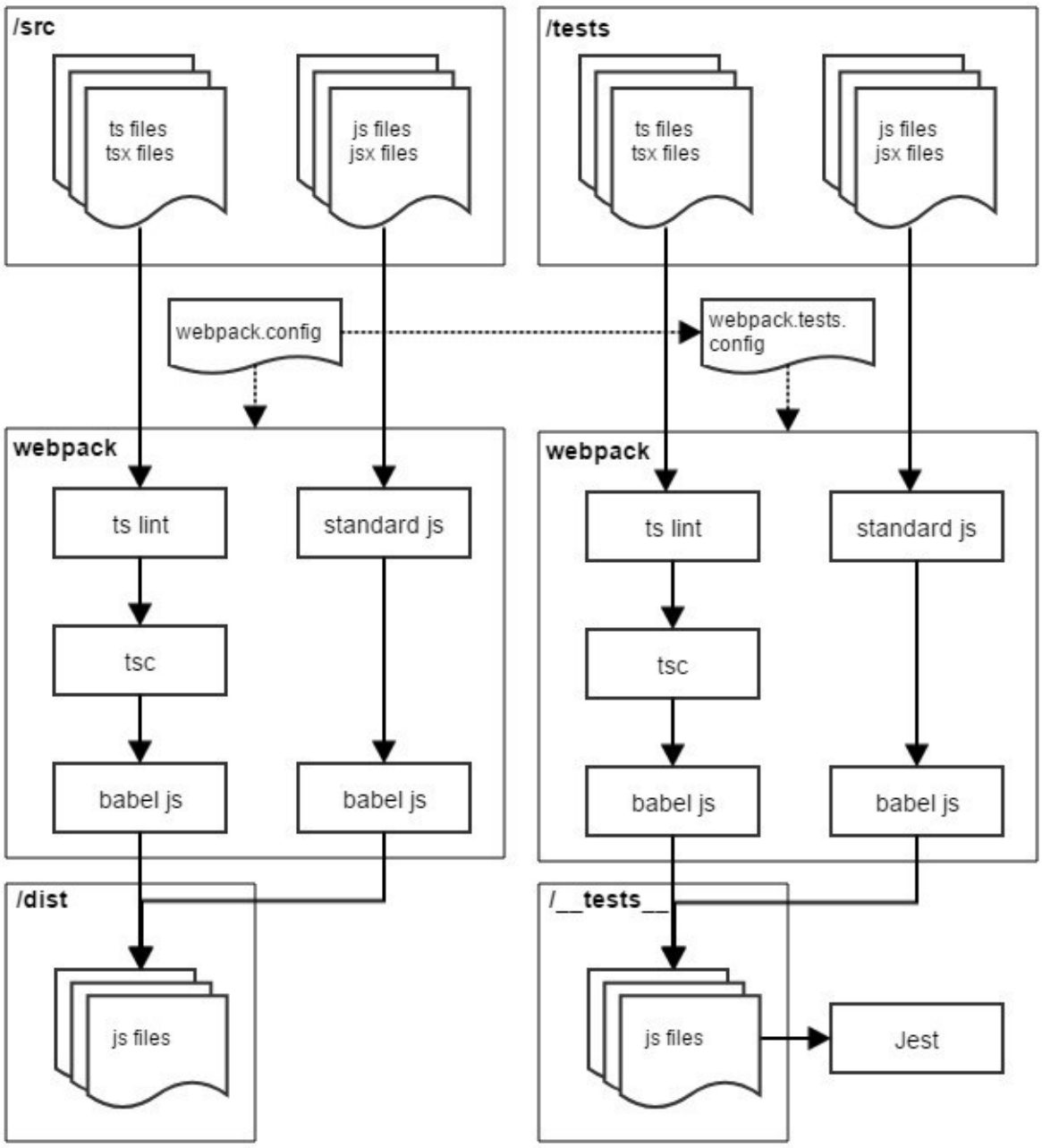

Рисунок 12 - Схема сборка бандла вебпаком (Webpack)

3.8. Git

Как система контроля версий был выбран git, как наиболее популярное решение. Использованный git flow схематично представлен на рисунке 13.

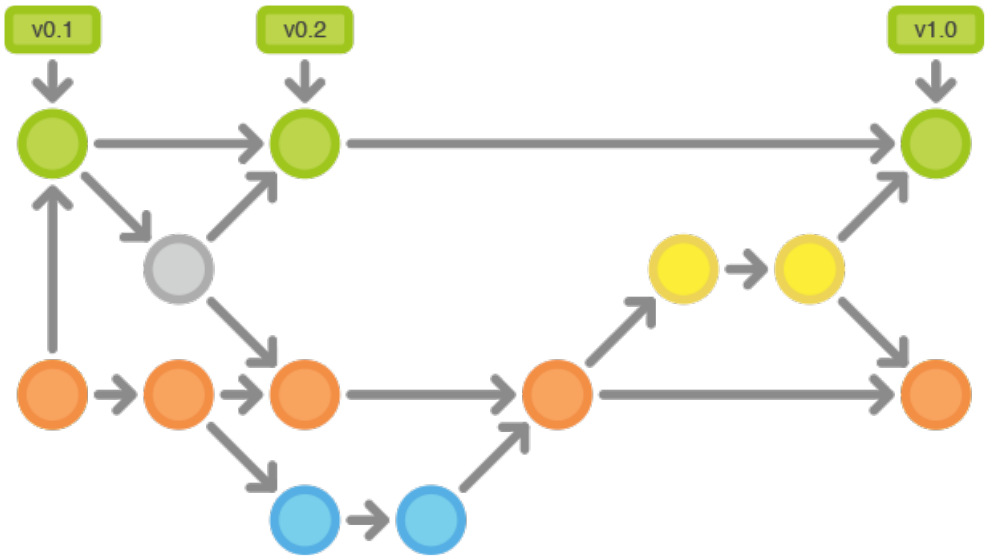

Рисунок 13- Использованный git flow

3.9. Git flow

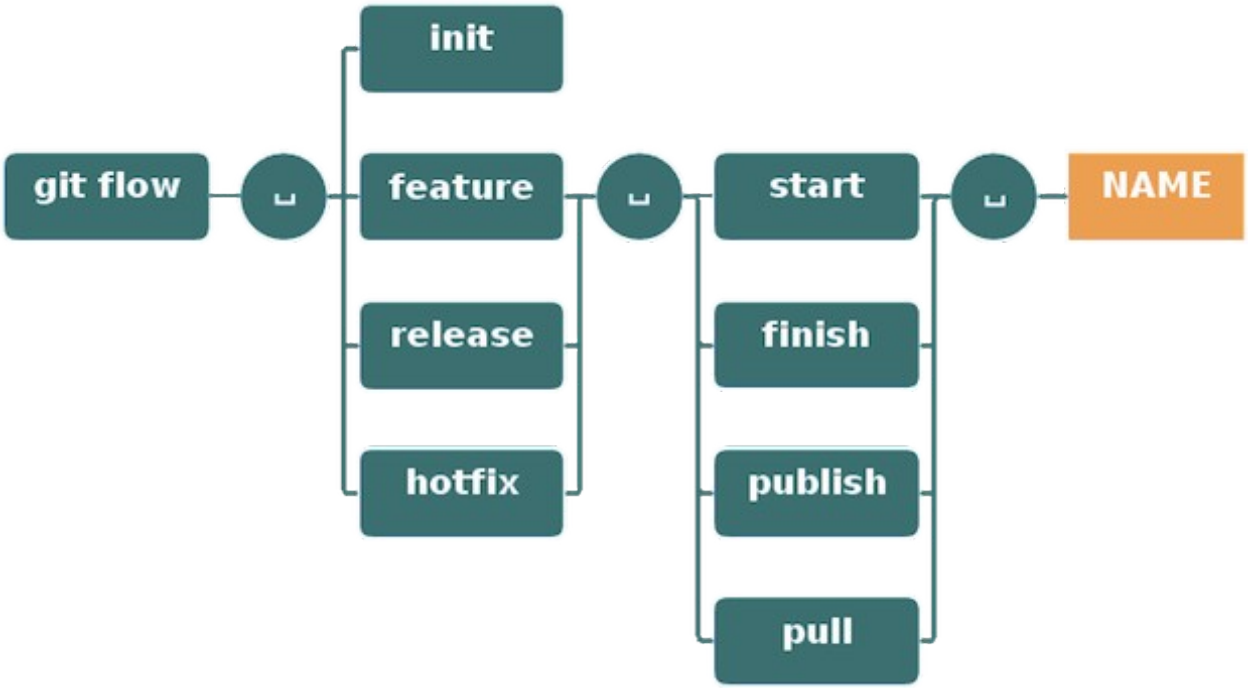

Рисунок 14 - Git flow команды

Git-flow- представляет собой пакет расширений git которые представляют собой высокоуровневые операции над репозиторием для поддержки ветвления Vincent Driessen. Дерево построения команд для управления git flow представлено на рисунке 14.

Git-flow-командная строка со справкой и улучшенным выводом, он основан на слиянии веток фич, не используя rebase.

Перед работой нужно инициализировать Git-flow. Сначала нужно проинициализировать внутри git-репозитория:

git flow init

затем отвечаем на вопросы о способах именования веток. Лучше оставить значения по умолчанию.

Фичи.

- Разработка фич для релизов следующих релизов;
- Присутствуют в репозиториях разработчиков.

Начало новой фичи

Из ветки «develop» разрабатываем новую ветку фичи:

Git flow feature start MYFEATURE

Завершение фичи:

- Слияние ветки MYFEATURE в «develop»;
- Удаление ветки фичи;
- Переключение на ветку «develop»

Git flow feature finish MYFEATURE

Публикация фичи

Чтобы фичу могли использовать другие пользователи, ее публикуют на удаленном сервере

Git flow feature publish MYFEATURE

Получение опубликованной фичи

Git flow feature pull origin MYFEATURE

Отслеживание в репозитории origin происходит

Git flow feature track MYFEATURE

Создание релиза

- Подготовка нового релиза продукта
- Устранение багов, подготовка метаданных для релиза.

Начало релиза

Используется команда для ответвления от «develop»

Git flow release

Git flow release start RELEASE (BASE)

Можно указать (BASE)-коммит, принадлежащего к ветке «develop», в виде хеша sha-1 и начать с него релиз.

Ветку релиза публикуем после ее создания для доступа других пользователей. Команда для публикации:

Git flow release publish RELEASE

Отслеживать удалённый релиз можно по команде:

Git flow release track RELEASE

Завершение релиза- реализуется в несколько действий:

- ветка релиза сливается в ветку «master»;
- релиз отмечается тегом равным его имени;
- ветка релиза обратно сливается с «develop»;
- удаляется ветка релиза.

Git flow release finish RELEASE

Отправляем изменения в тегах по команде:

git push –tags

4. РАЗРАБОТКА КРОССПЛАТФОРМЕННОГО PWA ПРИЛОЖЕНИЯ

4.1. Вход/регистрация

Как любое приложение наше начинается со страницы входа/регистрации. Помимо возможности использования приложения не авторизованными пользователями мы сразу предлагаем им зарегистрироваться, чтобы в будущем, при подключении облачного хранения данных у них не возникало проблем с переносом/миграцией локальных данных в облако. Миграция здесь необходима для привязки дел/списков дел к учетной записи пользователя.

Для этого мы сначала показываем неавторизованному пользователю форму входа, так как вход в систему более частый случай, чем регистрация и нагружать пользователя лишними действиями не лучшая идея.

Форма входа в приложение представлена на рисунке 15.

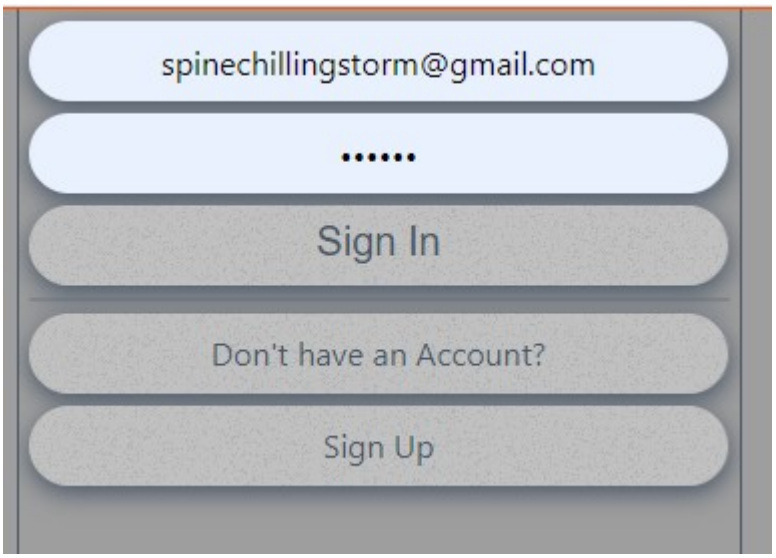

Рисунок 15 – Форма входа в приложение

На картинке представлена форма входа с полями для ввода почтового ящика и пароля для учетной записи. Поле для e-mail-а имеет валидацию на правильность ввода почтового ящика и проверяет соответствует ли он существующим шаблонам. Поле пароля проверяется на допустимые символы. Эти проверки позволяют нам снизить нагрузку на сервера, так как отметают совершенно невалидные данные.

Так же снизу есть кнопка для перехода на форму создания учетной записи для пользователей, у которых ее еще нет. При нажатии на кнопку нас переносит на форму создания учетной записи. Для подсказки неопытным пользователям присутствует надпись с вопросом, “Don’t have an Account?”, расположенная над кнопкой перехода на форму регистрации.

4.2. Форма регистрации

При переходе на форму регистрации, нам показывается страница с вышеупомянутой формой, на которой присутствует поле для ввода почтового ящика с его валидацией. Для валидации используется поиск соответствия при помощи механизма regExp, который проверяет содержит ли строка в себе все необходимые для почтового ящика символы в нужном порядке и в нежном количестве. Так же присутствуют поля ввода и подтверждения пароля. Форма регистрации представлена на рисунке 16.

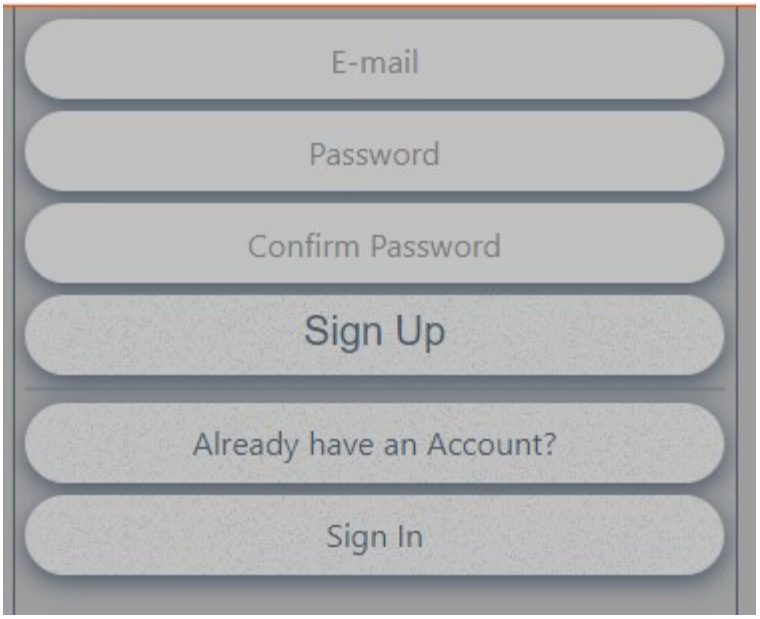

Рисунок 16 – Форма регистрации в приложении

При заполненных полях почтового ящика, пароля и подтверждения нам становится доступна кнопка регистрации. При удачной регистрации нас перекинет на страницу входа, а также на почту придет извещение.

Помимо полей ввода и кнопки регистрации присутствует ссылка для перехода на страницу входа для пользователей, которые не хотят регистрировать новый аккаунт.

При ошибках регистрации на клиенте (например ошибки валидации почтового ящика или недопустимых символах в пароле) или серверной ошибке регистрации мы получим об этом оповещение через всплывающее окно

4.3. Оповещения об ошибках.

Есть ошибки, которые не являются неожиданными, а скорее являются обычной частью работы приложения, например неправильный ввод пользователя, проблемы с сервером авторизации или другие предвиденные ошибки,

которые мы можем обработать и сообщить пользователю, что что-то пошло не так и, если он может это исправить (например ввести валидные данные) подсказать как это сделать.

Для таких ошибок у нас предусмотрен специальный слой в архитектуре приложения. Он отвечает за обработку предвиденных ошибок и выводит пользователю уведомления о них. Уведомления представлены на рисунке 17.

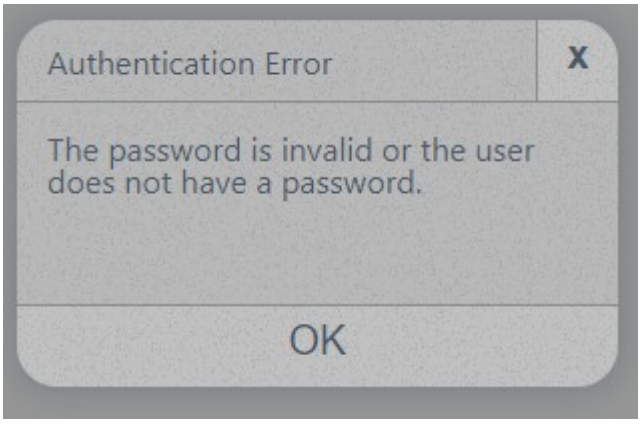

Рисунок 17 – Пример ошибки выводимой для пользователя

В этих уведомлениях мы выводим саму ошибку, из которой должно быть понятно может ли пользователь ее исправить, как например неправильный ввод, либо причину неполадок, на которую пользователь не может по каким-то причинам повлиять, но должен быть предупрежден, что приложение может вести себя слегка некорректно.

Помимо самой ошибки пользователь может закрыть сообщение несколькими способами: нажав крестик или кнопку OK.

4.4. Классическое представление списков

Для пользователей, которые хотят простые списки дел без лишнего функционала у нас есть страница, позволяющая с ними работать. Она состоит из списков для текущего периода, которые пользователь может создавать или удалять. Списки в приложении выглядят как представлено на рисунке 18.

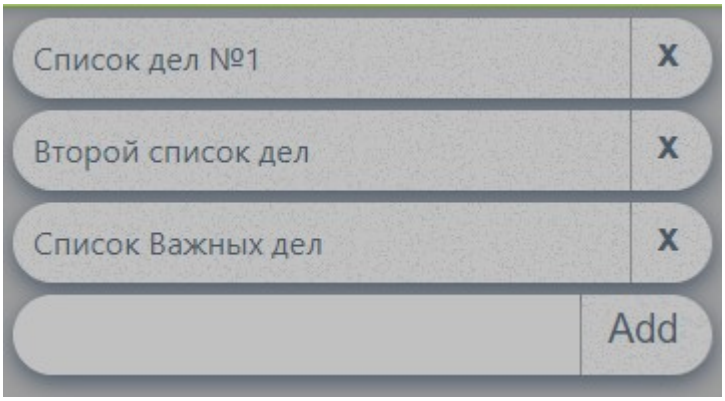

Рисунок 18 - Представление списка

На ней представлен список существующих списков дел, мы можем их сортировать путем перетаскивания мышкой, для этого мы используем самописную библиотеку. Присутствуют анимации для переноса списков, и что не мало важно этот список является виртуальным, что означает значительное снижение нагрузки для слабых устройств при наличии внушительного количества списков.

Для удаления списка нам достаточно нажать на крестик у элемента, представляющего список.

Для создания нам нужно вписать название списка и нажать кнопку добавить.

При изменениях в списках мы будем видеть, как сверху подсветка хэдера загорается красным, это сигнализирует,

что список не синхронизирован с облаком при работе с интернетом.

Синхронизация происходит каждые 5 секунд или при подключении интернета, если тариф пользователя позволяет.

4.5. Поиск по спискам

Со временем у пользователя может накопиться приличное количество списков дел и самих дел, что может привести к проблемам навигации по ним. Для этого в приложении предусмотрен функционал поиска по спискам и элементам списков. Поиск по спискам и элементам списков представлен на рисунке 19.

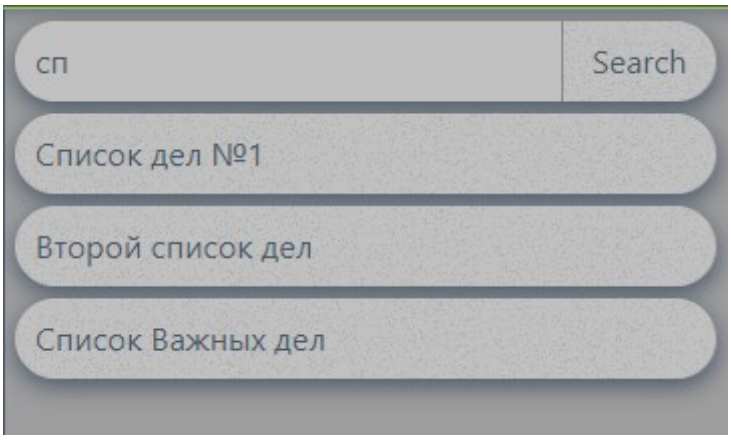

Рисунок 19 - Поиск по задачам и спискам

На изображении мы видим процесс поиска по существующим элементам списков и дел. При вводе символов в строку поиска мы получаем результаты в реальном времени, соответствующие введенным данным.

Поиск идет по подстроке и не учитывает регистр символов. Поиск идет только по элементам простого

функционала, продвинутый функционал канбана здесь не учитывается.

При клике по найденному элементу нас перекинет к нему соответственно нашей структуре.

4.6. Список задач

При переходе в список задач мы видим его элементы – задачи. Они могут выглядеть на первый взгляд просто, но несут в себе достаточно продвинутый функционал. Список задач представлен на рисунке 20.

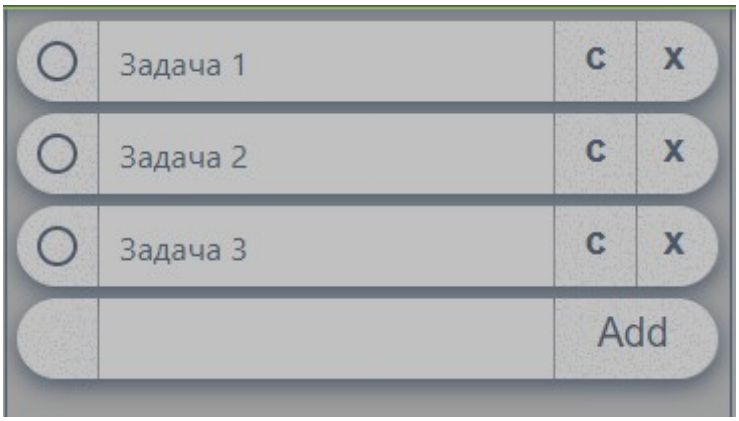

Рисунок 20 - Список задач в списке

У каждой задачи есть чекбокс, название, кнопка для редактирования и кнопка для удаления.

Чекбокс служит для отметки задачи выполненной. При нажатии на него он подсвечивается зеленым и запускается таймер, по истечению которого задача переместится в архив с другими уже выполненными задачами.

Название задачи – это поле, которое доступно для редактирования только в режиме редактирования. Это служит для удобства пользования функционалом сортировки

дел путем переноса их мышкой. Если название не помещается, то мы его сокращаем и обозначаем это сокращение путем добавления в конец многоточия.

Для просмотра полного названия дела и его описания можно нажать на кнопку редактирования, что приведет к появлению полного представления элемента дела. Появление происходит при использовании анимации вращения для более плавного перехода и встраивания элемента другого размера в дерево элементов. Элемент задачи в развернутом виде автоматически подстраивается под размеры введенного текста. В развернутом виде элемент не теряет возможностей переноса мышкой, хоть это и менее удобно.

Для удаления дела нужно нажать на крестик у элемента дела. Развернутый элемент дела представлен на рисунке 21.

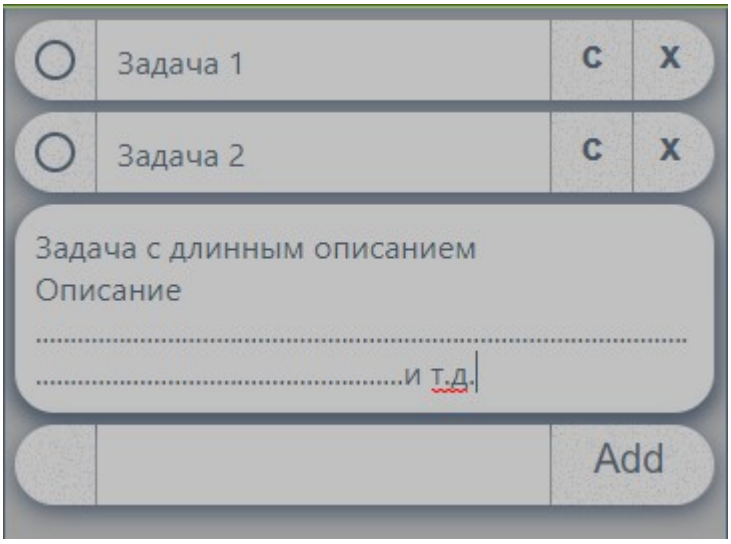

Рисунок 21 - редактирование задачи

Здесь представлен упрощенный вид лишь для демонстрации.

У развернутого вида задачи скрываются кнопки удаления, редактирования и чикбокс, чтобы не мешать

пользователю. Выход из режима редактирования происходит по нажатию все редактируемого элемента.

При любых изменениях в списке дел или в самих задачах, эти изменения добавляются в историю изменений, которая в будущем будет отправлена на облако для синхронизации. При наличии несохраненной истории изменений мы оповещаем об этом пользователя при помощи красной подсветки хэдера страницы. Это предупреждает пользователя, что ему нужно немного подождать, если он хочет, чтобы его изменения успели синхронизироваться перед выходом.

4.7. Архив готовых дел

Пользователям может понадобиться посмотреть на список уже завершенных дел, но смотреть на них постоянно скорее всего они не захотят, к тому же это быстро перерастет в беспорядочную кашу из дел. Для упорядочивания дел была добавлена функциональность архива, в который автоматически переносятся все, отмеченные как завершенные пользователем, дела с течением времени.

Архив представляет собой такой же список дел, как и обычный список, но в нем все дела отмечены как завершенные. Архив так же, как и другие списки виртуален и содержит функционал сортировки перетаскиванием. Архив с выполненными делами представлен на рисунке 22.

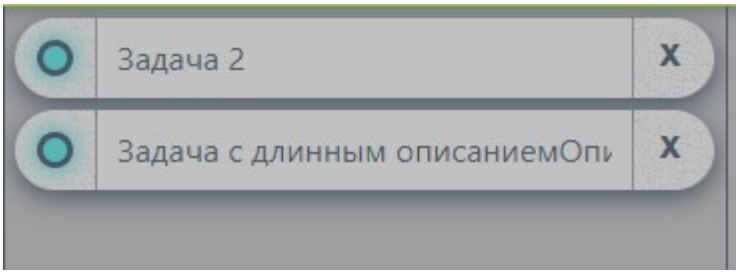

Рисунок 22 - Задачи в архиве

Здесь представлены всего две задачи в архиве в целях демонстрации, но их может быть очень много. Для управления задачами в архиве существуют 3 механизма: ручная сортировка, возвращение задачи в работу и удаление ненужных задач. Удаление было решено не автоматизировать для того, чтобы случайно не лишить пользователя важной информации.

Для возврата задачи в работу пользователю нужно нажать на чекбокс тем самым отменив выполненный статус задачи, после чего в течении пяти секунд задача вернется в список дел, из которого она поступила. Задержка сделана для того, чтобы пользователь не мог отправить задачу снова в работу случайно.

4.8. Выход из учетной записи

Для смены учетной записи или в целях безопасности пльзователю может понадобиться вручную выйти из приложения. Для этого в приложении предусмотрен механизм выхода из учетной записи.

В правом верхнем углу есть кнопка Sign Out. Она служит для начала процесса выхода из учетной записи. При нажатии на нее показывается всплывающее окно с предупреждением,

что будет совершен выход из учетной записи, и кнопками подтверждения и отмены. Кнопка выхода представлена на рисунке 23.

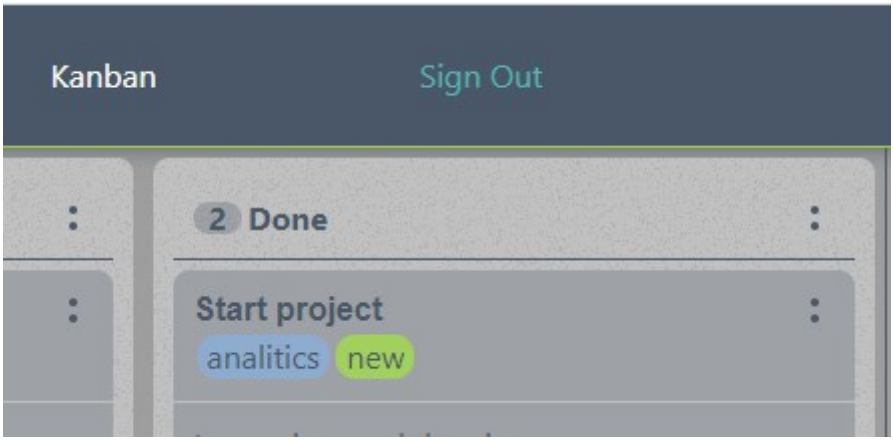

Рисунок 23 - Кнопка выхода

Всплывающее окно с предупреждением представлено на рисунке 24.

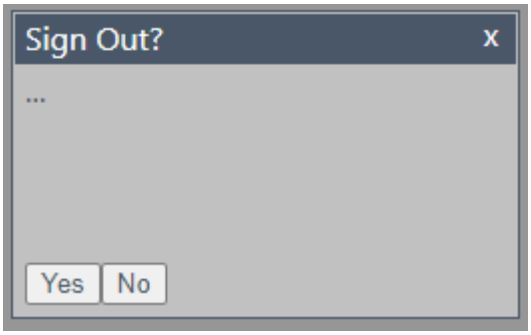

Рисунок 24 - Подтверждение выхода

При нажатии на Yes произойдет выход из учетной записи, при нажатии на крестик или No всплывающее окно скроется, и пользователь останется в системе.

4.9. Работа с большими списками

Со временем у пользователя может накопиться много активных, запланированных или выполненных дел и

отобразить их на одной станице одновременно не получится. Для этого можно использовать списки с прокруткой, но они со временем могут начать медленно работать, так как стандартные списки отрисовывают элементы даже за пределами видимости.

На самом нам не обязательно отрисовывать элементы, которые пользователь не видит. Для этого мы будем использовать подход, связанный с виртуальными списками. Проанализировав готовые решения, было принято решение сделать собственную реализацию виртуального списка, так как нам не нужен лишний функционал и хочется по максимуму уменьшить занимаемое приложением место.

В результате у нас получилось очень быстрое даже для слабых устройств приложение, которое может отрисовать огромное количество элементов списка. Благодаря этому даже на слабых устройствах пользователи не будут ощущать дискомфорт при работе с большим количеством дел в списке. Пролистывание списка представлено на рисунке 25.

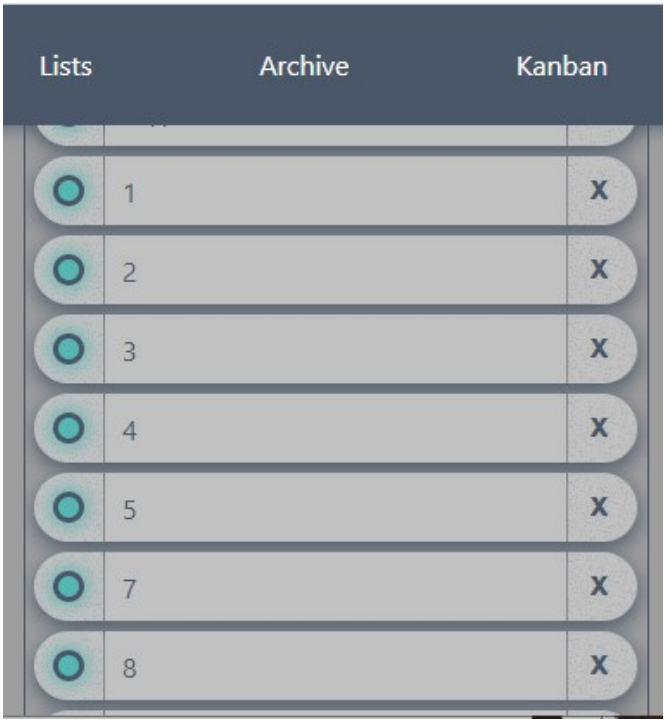

Рисунок 25 - Демонстрация пролистывания списков

Как можно заметить при пролистывании список ведет себя точно так же, как обычный и для пользователя разницы не видно, но он гораздо более оптимизирован.

4.10. Хранение данных локально

Для того, чтобы, при отсутствии интернет-соединения, пользователь имел возможность пользоваться приложением, мы должны как-то хранить данные на стороне клиента. Для этого современны браузеры предоставляют нам такие инструменты как: IndexedDB, localStorage и sessionStorage.

Для долгосрочного хранения нам точно не подходит sessionStorage, так как он очищает данные каждый раз при закрытии вкладки, хотя мы и можем хранить в нем данные нужные нам только в течении определенной сессии.

Среди IndexedDB и localStorage было решено использовать оба хранилища для разных целей. LocalStorage используется для данных еще не авторизованного пользователя и данных, которые имеют вспомогательных характер и не относятся к основной бизнес-логике. В то время как IndexedDB мы используем на полную при помощи средств firebase библиотек для работы с сервисами firebase API, так как в них полно встроенных средств для хранения уже полученных данных локально и подтягивания обновлений с сервера в случае восстановления соединения.

Что касается структуры получившихся данных, то для списка еще не авторизованного пользователя она выглядит следующим образом:

В приведенном коде показана лишь часть localStorage хранилища, используемая в тестовом окружении. IndexedDB достаточно оптимизирована и сжата средствами firebase и показывать ее смысла нет.

В будущем эти данные преобразуются в полноценную структуру для indexedDB при регистрации пользователем учетной записи.

4.11. Расширенный функционал приложения

При необходимости зарегистрированный пользователь может воспользоваться более широким спектром возможностей приложения. К ним относится канбан лист.

Функционал канбан листа – это визуализация подхода для управления задачами в небольшой команде, но также он отлично подходит и для одного человека. Выглядит он как несколько столбцов, в зависимости от ситуации их количество может варьироваться. У каждого столбца есть свое название, как правило — это название статуса, в котором находится задача.

Канбан лист помогает лучше воспринимать статус дел, а также легко управлять им при помощи перетаскивания дел из одного столбца-статуса в другой.

Реализация канбан листа в нашем приложении достаточно классическая, однако он отвязан от тяжелой инфраструктуры, обычно сопутствующей подобному функционалу. Внешний вид канбан листа, реализованного в приложении представлен на рисунке 26.

Рисунок 26 - Функционал канбан листа

В данном примере он заполнен случайными данными, только лишь в целях демонстрации. Для наглядности созданы три классических столбца-статуса для задач, это статусы “To Do”, “In Progress”, “Done”, которые символизируют самый простой стандартный жизненный цикл задачи.

При перетаскивании между столбцами задач, задачи меняют свой статус соответственно названию столбца, в который их поместили.

4.12. Работа с канбан листом

Продемонстрированный канбан лист – это не предел его возможностей. В нем заложены мощные инструменты для его модификаций под нужды каждого человека. Стандартные столбцы не всегда предоставляют полную и актуальную информацию по текущему состоянию задачи, так как некоторые задачи, которые взяты в работу, могут быть, например, заблокированы другими, либо отложены на неопределенный срок и возвращать их просто так в статус “Должны быть сделаны” не всегда лучшая практика. Или возьмем пример, когда подобными инструментами пользуются команды разработчиков. Для разных стадий разработки могут понадобиться разные статусы, такие как “проработка командой”, “тестирование”, “аналитическая проработка” и т.д. Поэтому неплохо было бы иметь возможность гибко настраивать используемый для этого инструмент.

В нашем случае мы предоставляем множество функций для этого. Одной из таких функций является настройка столбцов-статусов канбан листа. Всплывающее меню при нажатии на кнопку настроек листа представлено на рисунке 27.

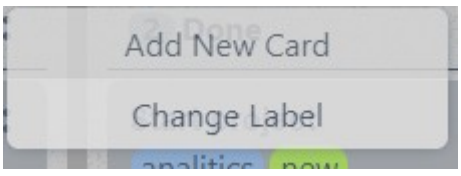

Рисунок 27 - Меню настроек столбца-статуса

В данном меню находится функционал по созданию новых карточек с предопределенным статусом и функционал для смены названия выбранного списка-статуса.

При нажатии на смену названия пользователь получает возможность редактирования названия столбца. После окончания редактирования столбец примет новое название, как и задачи, находящиеся в нем, поменяют статус в соответствии с выбранным названием. А также эти изменения добавятся в очередь для отправки в базу данных для синхронизации, если у пользователя будет приобретен соответствующий функционал.

При нажатии на добавление новой карточки задачи у пользователя появится конструктор новых задач. В нем он сможет задать название для задачи, ее теги и описание. При отмене создания задачи конструктор задач исчезнет, а при подтверждении создания у пользователя в соответствующем столбце появится новая задача со статусом, соответствующим названию столбца, в котором она была создана.

Создание задач с предопределенным не начальным статусом может понадобиться в случае начала работы с данным функционалом в середине уже начатой работы. Например, при внедрении нового подхода к уже построенным бизнес-процессам в фирме или переходу с ранее использовавшихся аналогов подобных журналов.

Помимо этого, есть возможность добавления или удаления уже существующих списков-статусов. Для добавления нового списка пользователю нужно кликнуть на свободной области страницы канбан листов и у него

откроется схожий с созданием задач функционал создания списков. В нем пользователь сможет задать название-статус для нового списка и при нажатии подтверждения этот список добавится на страницу.

Для удаления уже существующего списка пользователю необходимо нажать ту же кнопку, что и для добавления новых задач, но уже на пустом столбце. В этом случае у пользователя появится дополнительная возможность удаления списка. Было решено скрывать эту кнопку для списков, в которых есть задачи, чтобы пользователь не мог случайно или специально их удалить, тем самым нарушив процесс управления задачами. При нажатии на пункт удаления высветится окошко с подтверждением удаления списка. Подтвердив действие, пользователь уберет у себя из доступных списков выбранный, а приложение поставит его в очередь для синхронизации.

Помимо управления такими большими сущностями как списки, пользователю может понадобиться и более гибкое управление самими сущностями задач, помимо простого создания и смены их статуса.

Для более гибкого управления задачами в приложении реализованы такие функциональности как просмотр задачи, управление ее контентом, смена названия задачи и ее тегов. Внешний вид задачи представлен на рисунке 28.

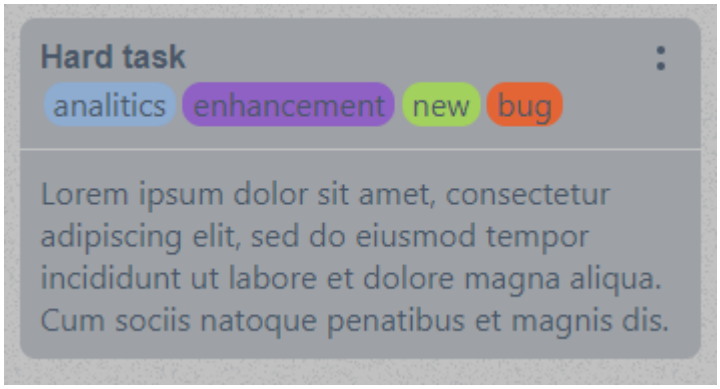

Рисунок 28 - Внешний вид задачи

Здесь представлена задача, используемая в ручном тестировании функционала с на генерированными данными в целях демонстрации.

У задачи присутствует название, теги и описание. Чтобы приступить к ее редактированию, пользователю необходимо нажать на кнопку, находящуюся в правом верхнем углу задачи. При нажатии на кнопку редактирования у пользователя появится всплывающее меню со следующими пунктами: “смена названия”, “редактирование контента”, “удаление”. Всплывающее меню, при нажатии на кнопку настроек задачи, представлено на рисунке 29.

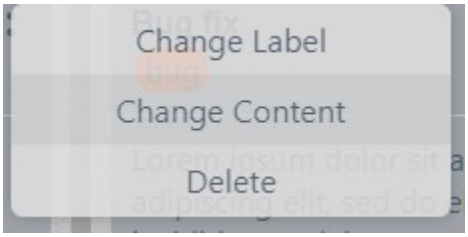

Рисунок 29 - Меню настройки задачи

При нажатии на изменение названия у пользователя появится возможность редактировать название задачи и ее теги. После окончания редактирования задача сменит название и теги, а также новые изменения автоматически встанут в очередь для синхронизации с облаком.

При нажатии на кнопку изменения контента задача изменит вид наподобие с задачами из облегченного представления списков и у пользователя появится возможность редактирования описания задачи. По окончанию редактирования задача изменит описание и изменения отправятся в очередь для синхронизации.

При нажатии пользователем на кнопку удаления появляется всплывающее окно, как при удалении списка или выхода из учетной записи с подтверждением действия.

4.13. Общий вид приложения

Приложение имеет возможность навигации по страницам, которая выглядит как навигация по обычному сайту, за исключение того, что у нас не запрашивается статика при каждом переходе. Могут лишь подгружаться более актуальные данные для страниц из облака при наличии у пользователя подписки на сохранение данных в облаке и интернет-соединения.

Для навигации в приложении присутствует верхнее навигационное меню, в котором пользователь может выбирать часть приложения, с которой в данный момент желает взаимодействовать. Навигационное меню представлено на рисунке 30.

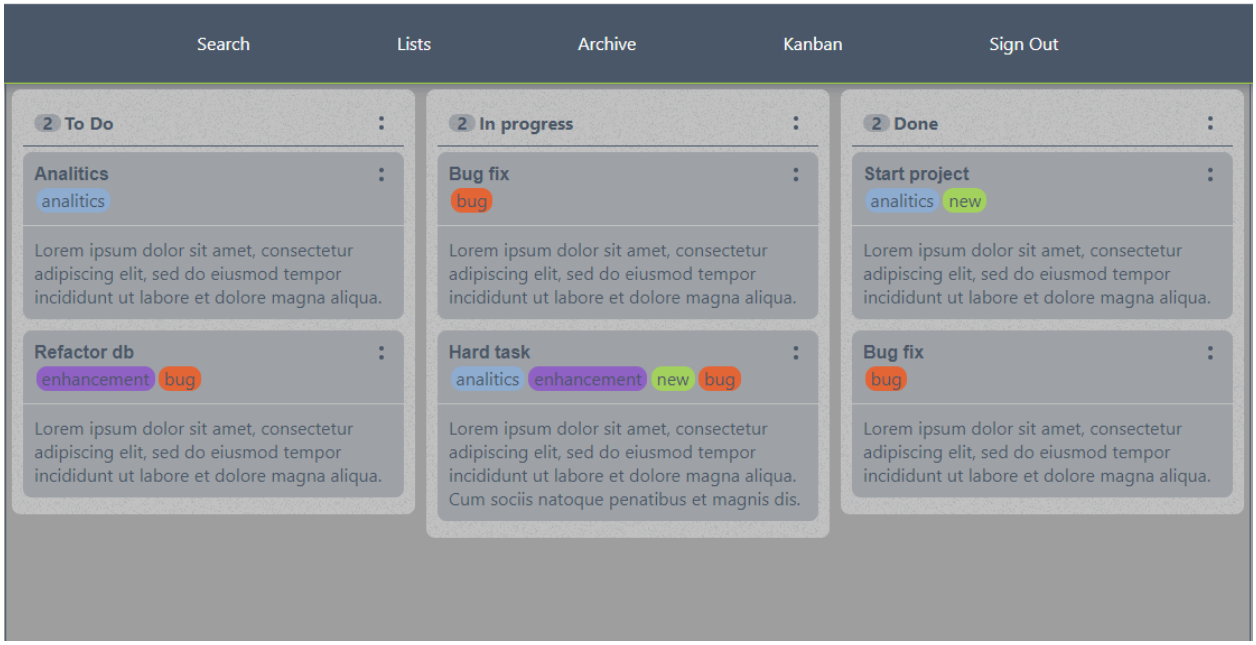

Рисунок 30 - Общий вид сайта (страница с канбан листом)

В навигационном меню присутствуют ссылки на поиск по приложению, упрощенное представление дел, архив, куда попадают завершенные дела и канбан лист, продемонстрированные выше. А также кнопка для выхода из приложения.

Сейчас все приложение представлено в английской локализации, так как рассчитывается на повсеместное использование, но в архитектуру уже заложена возможность переключения на другие языки, которые можно будет добавлять со временем. Архитектурно это будет реализовано через подгрузку локализации с сервера, чтобы не хранить все на стороне клиента и иметь доступ к возможности, без обновления всего клиента, вносить правки в локализацию и добавлять новые локализации.

Возможность смены локализации клиентом предполагается из меню настроек профиля.

5. БЕЗОПАСНОСТЬ ПРИЛОЖЕНИЯ

Рассмотрим какие могут быть угрозы для мобильных приложений:

- каналы передачи информации небезопасны;
- наличие отладочного кода;
- секретные данные в открытом виде;
- неправильная расстановка прав доступа;
- слабая криптография;
- отсутствие проверок входящих данных;
- внедрение SQL-инъекции;
- атаки межсайтового стриптинга (XSS).

Рассмотрим методы защиты мобильных приложений:

- обфускация или запутывание кода;
- аутентификация-проверка и принятие введенной информации на сервер;
- аутентификация и контроль параметров устройства;
- шифрование средствами ОС, приложения, МВМ;
- при утере устройства предусмотреть инструменты блокировки и очистки его;
- ограничение доступа к возможным точкам доступа;
- ограничение доступа по времени к КОРП ресурсам.

Рассмотрим атаки на веб-сайты:

- одна из самых частых форм взлома - подмена главной страницы сайта;
- удаление файловой системы, когда исчезает вся информация;

- подмена информации-могут подменить данные владелица сайта;
- троянские программы-вредоносные программы, которые проникают в компьютер под видом легального ПО, могут осуществлять переадресацию сайта, заражать вирусами, кражу персональных данных.
- создание высокой нагрузки, приводящей к падению ОС сервера;
- рассылка спама, использование веб-сайта для рассылки спама.

Рассмотрим методы защиты веб-сайтов:

- своевременное обновление ПО как сервисного, так любое, запущенное на сайте;
- использование параметризованных запросов для защиты от атак SQL-инъекци;
- при создании сообщения об ошибке нужно использовать «неправильное имя пользователя» или «неправильный пароль», т.е. не давать лишней информации, которой может воспользоваться взломщик;
- использовать комплексные пароли с буквами и цифрами;
- установить firewall-технологический барьер для защиты от несанкционированного сообщения между компьютерными сетями и хостами;
- установить DMZ, обеспечивающую доступ к порту 80 и 443;
- для загрузки файлов на сервер использовать протоколы SFTP, SSH, SSL.

Для обеспечения безопасности созданного приложения используется HTTPS, в котором защита данных обеспечивается протоколом SSL/TLS, позволяющий устанавливать защищенное соединение через незащищенный канал. Браузер запрещает использование HTTP. Использование данной системы позволит сохранить целостность и конфиденциальность данных пользователя.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было разработано PWA приложение "Bullet Journal", удовлетворяющее всем требованиям современных PWA приложений, а именно возможностью работы при отсутствии интернета, возможностью кроссплатформенной работы, как на мобильных устройствах, так и на персональных компьютерах пользователей. Был получен опыт разработки архитектуры приложения, настройки service worker-ов, настройки непрерывной интеграции и доставки, настройки пирамиды тестирования, работы с облачными сервисами, лямбда функциями и разработки клиентской части PWA приложения.

Был полностью реализован необходимый функционал и учтены все технические требования.

На этапе проектирования архитектуры приложения были проанализированы имеющиеся виды архитектуры кроссплатформенных приложений, выбрана подходящая архитектура для создания PWA приложения с поставленными требованиями. Ею оказалась Serverless архитектура, так как необходимо хранить большую часть бизнес логики на стороне клиента, из-за необходимости работы приложения при отсутствии интернета. Так же на этом этапе было принято решение использовать облачные сервисы, в частности firebase, так как он предоставляет весь необходимый функционал для разработки приложений с Serverless архитектурой.

Во время разработки приложения были проанализированы аналоги, имеющиеся на рынке. Были

учтены их недостатки и преимущества. Были проанализированы существующие способы реализации кроссплатформенных приложений, проведено сравнение существующих фреймворков и выбран React как наиболее подходящий, для реализации клиентской части приложения.

Приложение было протестировано в процессе разработки при помощи пирамиды тестирования, состоящей из unit, интеграционных и e2e тестов. Так же проведено ручное тестирование. Для проведения автоматического тестирования был настроен процесс непрерывной интеграции и доставки, позволяющий быстрее и качественнее доставлять новый функционал конечному потребителю.

Список литературы

1. Typescript документация [Электронный ресурс]. - Режим доступа: https://www.typescriptlang.org/.
2. WC3 стандарты [Электронный ресурс]. - Режим доступа: https://www.w3.org/standards/.
3. Самоучитель JavaScript [Электронный ресурс]. - Режим доступа: https://learn.javascript.ru/.
4. Документация React [Электронный ресурс]. - Режим доступа: https://ru.reactjs.org/.
5. Документация JavaScript [Электронный ресурс]. - Режим доступа: https://developer.mozilla.org/ru/.
6. Справочник по HTML5 [Электронный ресурс]. - Режим доступа: http://htmlbook.ru/.
7. Учебник по HTML5 CSS3 [Электронный ресурс]. - Режим доступа: https://htmlacademy.ru/.
8. Документация по react-redux [Электронный ресурс]. - Режим доступа: https://redux.js.org/basics/usage-with-react.
9. Bakaus P. What Are Progressive Web AMPs? [Электронный ресурс] // 2016. URL: https://www.smashingmagazine.com/2016/12/progressive-web-amps/ .
10. Dean A. H. Progressive Web Apps. N.Y.: Meap, 2017. P. 2-5.
11. Markov D. Progressive Web Apps. Everything You Should Know About Progressive Web Apps [Электронный ресурс] // 2016. URL: http://tutorialzine.com/2016/09/everything-you-should-know-aboutprogressive-web-apps/ .

12. Демьяненко М. Что такое Progressive Web Apps и какие возможности они открывают для вашего бизнеса [Электронный ресурс] // 2016. URL: https://netpeak.net/ru/blog/chto-takoe-progressive-web-apps-i-kakievozmozhnosti-oni-otkryvayut-dlya-vashego-biznesa/ .

13. Сальников М., Липатцев А., Кардава З., Пугачев С. Progressive Web Apps Day. Онлайн конференция [Электронный ресурс] // 2016. URL: pwaday.ru.

14. Мобильные приложения. Анализ защищенности мобильных приложений [Электронный ресурс] // 2016. URL: https://dsec.ru/services/securityanalysis/mobile-applications.

15. 10 важных советов безопасности для защиты сайта [Электронный ресурс] // 2016. URL: http://alexdev.ru/1025.

16. Green I. From AMP to PWA [Электронный ресурс] // 2016. URL: h

17. Кедлек Т. Адаптивный дизайн. Делаем сайты для любых устройств; Питер - М., 2013. - 288 с.

18. Фрейен Бен HTML5 и CSS3.Разработка сайтов для любых браузеров и устройств; Питер - М., 2014. - 304 с.

19. Мациевский Николай. Разгони свой сайт. Методы клиентской оптимизации веб-страниц; Интернет-университет информационных технологий, Бином. Лаборатория знаний - М., 2009. - 264 с.

20. Гарднер Л., Григсби Д. Разработка веб-сайтов для мобильных устройств; Питер - М., 2013. - 397 с.

21. Дакетт, Д. HTML и CSS. Разработка и дизайн веб-сайтов / Д. Дакетт. - М.: Эксмо, 2018. - 208 с.

22. Якоб Нильсен, Хоа Лоранжер «Web-дизайн. Удобство использования Webсайтов»Издательство «Вильямс». 2007г.- 376 с.

23. Мэтью МакДональд «Создание Web-сайтов. Основное руководство» 2010 г. Издательство: Эксмо, - 768 с.

24. Чебыкин Ростислав Разработка и оформление текстового содержания сайтов; БХВ-Петербург - М., 2011. - 528 с.

25. Сырых, Ю. Современный веб-дизайн. Настольный и мобильный / Ю. Сырых. - М.: Диалектика, 2019. - 384 с.

26. Рассел Р. Защита от хакеров коммерческого сайта; Книга по Требованию - М., 2014. - 548 с.

27. Андерсон С. Приманка для пользователей. Создаем привлекательный сайт; Питер - М., 2013. - 537 с.

28. Киселев, С.В. Веб-дизайн: Учебное пособие / С.В. Киселев. - М.: Academia, 2018. - 384 с.

29. Нильсен, Я. Веб-дизайн: книга Якоба Нильсена / Я. Нильсен. - М.: Символ, 2015. - 512 с.

30. Гарретт, Д. Веб-дизайн. Элементы опыта взаимодействия / Д. Гарретт. - СПб.: Символ-плюс, 2015. - 192 с.

ПРИЛОЖЕНИЕ А

Спецификация

Обозначение	Наименование	Примечание
	Документация	
ВКР-09.04.02-181323-20 81	Пояснительная записка	

Приложение Б

Фрагмент кода с настройкой react-router

```
render() {
    const { modalState, authState, dataSaveStatus } = this.props;

    return (
        authState.requestStatus === requestStatus.default
            ? <Preloader />
            : (
                <>
                    <App>
                        {authState.isSignedIn && (
                            <Nav>
                                <NavLink path="/" value="Search" />
                                <NavLink path="/view" value="Lists" />
                                <NavLink path="/archive" value="Archive" />
                                <NavLink path="/kanban" value="Kanban" />
                                <NavLink onClick={this.handleClick} path="/signin" value="Sign Out" />
                            </Nav>
                        )}
                        <StatusBar status={dataSaveStatus.isSaved ? 'fulfilled' : 'rejected'} />
                        <Main>
                            <Switch>
                                <Route exact path="/" component={Search} />
                                <Route exact path="/view" component={TodoSelector} />
                                <Route path="/view/:id" component={TodoList} />
                                <Route path="/archive" component={Archive} />
                                <Route path="/kanban" component={KanbanBoard} />
                                <Route path="/signin" component={SignIn} />
                                <Route path="/signup" component={SignUp} />
                            </Switch>
                        </Main>
                    </App>
                    <Modal {...modalState} />
                </>
            )
    );
}
```

Приложение В

Фрагмент кода реализации Drag & Drop функционала

```
render() {
    const { nameSpaces } = this.state;

    return (
        <Provider
            onDragStart={this.handleDragStart}
            onDragOverDraggable={this.handleDragOverDraggable}
            onDragLeaveDroppable={() => {}}
            onDragEnterDroppable={this.handleDragEnterDroppable}
            onDragEnd={this.handleDragEnd}
        >
            <div style={{ display: 'flex' }}>
                <ul style={{ margin: '5px' }} className={style.list}>
                    <Droppable
                        nameSpace="main1"
                    >
                        {nameSpaces.main1.map((value) => (
                            <Draggable key={value} id={value}>
                                <li className={style.item}>{value}</li>
                            </Draggable>
                        ))}
                    </Droppable>
                </ul>
                <ul style={{ margin: '5px' }} className={style.list}>
                    <Droppable
                        nameSpace="main2"
                    >
                        {nameSpaces.main2.map((value) => (
                            <Draggable key={value} id={value}>
                                <li className={style.item}>{value}</li>
                            </Draggable>
                        ))}
                    </Droppable>
                </ul>
            </div>
        </Provider>
    );
}
```

Приложение Г

Фрагмент кода реализации виртуального списка

```
handleWheel = (e) => {
    let { list } = this.state;

    if (list === null) {
        list = e.currentTarget;
        this.setState({ list });
    } else {
        const { children } = list;
        const childHeight = this.getChildHeight();
        const isNegative = e.deltaY < 0;
        const delta = Math.abs(e.deltaY);
        const maxDiff = 17;
        const minDiff = 10;

        let topDiff;
        if (delta > maxDiff) {
            topDiff = isNegative ? -maxDiff : maxDiff;
        } else if (delta < minDiff) {
            topDiff = isNegative ? -minDiff : minDiff;
        } else {
            topDiff = isNegative ? -delta : delta;
        }

        const lastChild = children[children.length - 1];
        const marginBottom = +getComputedStyle(lastChild).marginBottom.slice(0, -2);
        const shift = -this.calcShift(list, childHeight);

        this.setState((prevState) => {
            const listPosition = list.getBoundingClientRect();
            const lastChildPosition = lastChild.getBoundingClientRect();
            let top = prevState.top - topDiff;
            let position = -Math.floor(top / childHeight) - 1;

            if (listPosition.bottom > lastChildPosition.bottom + marginBottom - topDiff) {
                top = prevState.top;
                position = prevState.position;
            }

            if (position < 0) {
                top = 0;
                position = prevState.position;
            }

            return {
                position,
                top,
                shift,
            };
        });
```

Приложение Д

Фрагмент кода настроек сборки Typescript

```
{
  "compilerOptions": {
    "target": "ESNext",
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react",
    "sourceMap": true,
    "outDir": "./build",
    "removeComments": true,
    "importHelpers": true,
    "pretty": true,
    "noEmitHelpers": true,

    /* Strict Type-Checking Options */
    "noImplicitAny": false,
    "strictNullChecks": true,
    "strictFunctionTypes": false,
    "strictBindCallApply": true,
    "strictPropertyInitialization": false,
    "noImplicitThis": false,
    "alwaysStrict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true,
    "paths": {
      "*": ["src/*"]
    },
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true
  },
  "include": ["src"],
  "exclude": ["node_modules", "build", "assets"],
  "buildOnSave": false,
  "compileOnSave": false
}
```