Федеральное государственное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ И. В. Евдокимов
подпись инициалы, фамилия
«__» _____ 2018 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.04 «Программная инженерия»

Мобильное приложение для управления персональными заметками и распорядком дня

Руководитель	_____	доцент, к.т.н.	И.В. Евдокимов
	подпись, дата	должность, ученая степень	инициалы, фамилия
Выпускник	_____		А. И. Водянкин
	подпись, дата		инициалы, фамилия
Нормоконтролер	_____	доцент, к.т.н.	О. А. Антамошкин
	подпись, дата	должность, ученая степень	инициалы, фамилия

Красноярск 2018

СОДЕРЖАНИЕ

Введение

В современном мире одно из главных богатств человека это время – единственный ресурс, который не может восполнить человек, а его планирование очень важная часть жизни. Приложение Time Management System должно помочь контролировать расписание всего дня.

Актуальность приложения весьма высока, потому что вопрос контроля времени актуален с давних времен и многие люди помещены на таком контроле. В течение дня, каждый человек выполняет какое-то количество задач. Со временем, появляются новые задачи, а к некоторым старым приходиться возвращаться. Вместе с количеством задач растет и объем информации, которая становится сложной для запоминания: поневоле мы начинаем упускать из виду некоторые важные задачи, возможно, менее важные, чем остальные, либо начинаем делать их не своевременно. Для решения перечисленных проблем можно придерживаться нескольких правил [1-2]:

– каждый вечер планировать время согласно задачам, которые необходимо выполнить завтра;

– сортировать задачи на категории;

– придерживаться принципа «70/30», т.е. выделять 70% свободного времени на запланированные дела;

– выполнять большую части дел до обеда;

– находить время для отдыха и реалистично планировать время;

– разбивать комплексные и сложные задачи на более мелкие.

Возможно, кому-то достаточно придерживаться данных рекомендаций. Но в дополнение ко всему, можно использовать специализированное программное обеспечение.

Цель работы создать уникальное мобильное приложение с функциями распорядка дня, заметок и умной картой для русскоговорящей аудитории.

Объектом исследования является русскоговорящая целевая аудитория возрастом старше 6 лет.

Предметом исследования является процесс планирование временных ресурсов.

Итак, для достижения поставленной цели необходимо решить следующие задачи:

− изучить литературу по программированию клиент-серверных приложений для мобильных устройств;

− проанализировать рынок мобильных приложений, связанных с выбранной тематикой;

− повысить навык программирования;

− повысить навык менеджмента проекта;

− спроектировать архитектуру приложения;

− разработать дизайн интерфейса мобильного приложения;

− создать работоспособную команду с чётким разделением времени и обязанностей.

Для того чтобы создать работоспособную команду нужно определить роли в проекте, создать начальную документацию в виде планирования работ в проекте, и провести анализ требований аудитории к новому проекту. Перейдём к следующей главе, где описано как решили данные вопросы.

1 Проектирование мобильных приложений

Мобильные устройства уже давно стали частью жизни современного человека. С каждым годом они расширяют возможности пользователей. Вместе с этим повышаются требования к ним и, в частности, к программным продуктам, которые доступны на этих платформах. Поэтому важно грамотно проектировать приложения.

Перед тем как начать создавать проект необходимо составить чёткий план, разбить роли в команде, произвести анализ рынка, подготовить техническое задание, определить реальные сроки для каждого модуля проекта и подсчитать необходимые расходы и возможные доходы [3].

IEEE Standard Glossary of Software Engineering Terminology определяет требования как:

1) условия или возможности, необходимые пользователю для решения проблем или достижения целей;

2) условия или возможности, которыми должна обладать система или системные компоненты, чтобы выполнить контракт или удовлетворять стандартам, спецификациям или другим формальным документам;

3) документированное представление условий или возможностей для п. 1 и 2.

Требования к ПО состоят из трех уровней – бизнес-требования, требования пользователей и функциональные требования. Вдобавок каждая система имеет свои нефункциональные требования.

Бизнес-требования содержат высокоуровневые цели организации или заказчиков системы. Как правило, их высказывают те, кто финансируют проект, покупатели системы, менеджер реальных пользователей, отдел маркетинга. В этом документе объясняется, почему организации нужна такая система, то есть описаны цели, которые организация намерена достичь с ее помощью. Мне нравится записывать бизнес-требования в форме документа об образе и границах проекта, который еще иногда называют уставом

проекта или документом рыночных требований. Определение границ проекта представляет собой первый этап управление общими проблемами увеличения объема работ.

Требования пользователей описывают цели и задачи, которые пользователям даст система. К отличным способам представления этого вида требований относятся варианты использования, сценарии и таблицы «событие – отклик». Таким образом, в этом документе указано, что клиенты смогут делать с помощью системы.

Функциональные требования определяют функциональность ПО, которую разработчики должны построить, чтобы пользователи смогли выполнить свои задачи в рамках бизнес-требований. Иногда они называются требованиями поведения, они содержат положения с традиционным «должен» или «должна»: «Система должна по электронной почте отправлять пользователю подтверждение о заказе». Функциональные требования документируются в спецификации требований к ПО, где описывается так полно, как необходимо, ожидаемое поведение системы.

Системные требования – это высокоуровневые требования к продукту, которые содержат многие подсистемы. Говоря о системе, мы подразумеваем программное обеспечение или подсистемы ПО и оборудования. Люди — часть системы, поэтому определенные функции системы могут распространяться и на людей.

Бизнес-правила включают корпоративные политики, правительственные постановления, промышленные стандарты и вычислительные алгоритмы. Бизнес-правила не являются требованиями к ПО, потому что они находятся снаружи границ любой системы ПО. Однако они часто налагают ограничения, определяя, кто может выполнять конкретные ВИ, или диктовать, какими функциями должна обладать система, подчиняющаяся соответствующим правилам. Иногда бизнес-правила становятся источником атрибутов качества, которые реализуются в функциональности. Следовательно, вы можете отследить происхождение

конкретных функциональных требований вплоть до соответствующих им бизнес-правил.

Нефункциональные требования описывают цели и атрибуты качества. Атрибуты качества представляют собой дополнительное описание функций продукта, выраженное через описание его характеристик, важных для пользователей или разработчиков. К таким характеристикам относятся:

1) легкость и простота использования;

2) легкость перемещения;

3) целостность;

4) эффективность и устойчивость к сбоям;

5) внешние взаимодействия между системой и внешним миром;

6) ограничения дизайна и реализации, которые касаются выбора возможности разработки внешнего вида и структуры продукта.

Характеристика продукта – это набор логически связанных функциональных требований, которые обеспечивают возможности пользователя и удовлетворяют бизнес-цели. В области коммерческого ПО характеристика представляет собой узнаваемую всеми заинтересованными лицами группу требований, которые важны при принятии решения о покупке – элемент маркированного списка в описании продукта.

Разработка мобильных приложений состоит из нескольких этапов, среди которых проектирование. Проектированию подлежат архитектура программного обеспечения, устройство компонентов и пользовательские интерфейсы. Целью проектирования является выявление требуемых свойств программного обеспечения и дальнейший их анализ. Как правило, приложения должны обладать высокой доступностью, качественно выполнять требуемые функции и иметь адекватную стоимость. Эти характеристики, в той или иной степени, определяет архитектура ПО [4].

Архитектура – это базовая организация системы, воплощенная в ее компонентах, их отношениях между собой и с окружением, а также определяет принципы её проектирования и развития. Также, существуют

многие другие определения архитектуры, которые признают не только структурные элементы, но и их композиции, а также интерфейсы и другие соединительные звенья.

Система – это набор компонентов, объединенных для выполнения определенной функции или набора функций. Термин «система» охватывает отдельные приложения, системы в традиционном смысле, подсистемы, системы систем, линейки продуктов, семейства продуктов, целые корпорации и другие агрегации, имеющие отношение к данной теме. Система существует для выполнения одной или более миссий в своем окружении.

Архитектура определяет структуру программного обеспечения. Как правило, программисты работают со схемами, на которых изображены структурные аспекты системы – будь то архитектурные уровни, компоненты или распределенные узлы. Структурный элемент может быть подсистемой, процессом, библиотекой, базой данных, вычислительным узлом, системой в традиционном смысле, готовым продуктом. Каждый из этих элементов может быть представлен разными способами. Например, соединительное звено может представлять собой сокет, быть синхронным или асинхронным, быть связанным с конкретным протоколом и так далее. Примеры некоторых структурных элементов описываются с помощью UML-диаграмм.

Как и структуру, также архитектура определяет поведение системы, то есть взаимодействия между структурными элементами. Эти взаимодействия, в свою очередь, обеспечивают желаемое поведение системы. Описание поведение системы можно описывать с помощью диаграммы последовательности [5-6].

Несмотря на то, что архитектура определяет структуру и поведение, она определяет не все в структуре и не все в поведении. На первый план выходят более значимые элементы.

Значимые элементы – это элементы, которые имеют продолжительное и устойчивое действие, например:

– главные структурные элементы;

– элементы, связанные с основным поведением;

– элементы, которые определяют значимые свойства, такие как надежность и масштабируемость.

Как правило, архитектура не имеет отношения к гранулярным деталям этих элементов. В этом смысле архитектура – это некоторое обобщение системы, помогающее разработчику архитектуры управлять сложностью [7].

С течением времени, некоторые элементы изменяются при различных уточнениях поведения, но, в целом, относительная стабильность архитектуры – это признак качества, хорошо отлаженного процесса разработки и грамотных разработчиков.

Выделяют множество различных шаблонов архитектуры. В рамках данной работы, рассмотрим некоторые из них:

– многоуровневый шаблон;

– шаблон посредника;

– Model-View-Controller (MVC) шаблон;

– шаблон клиент/сервер.

Все шаблоны могут использоваться как отдельно, так и комбинации с другими, так как, в свою очередь они описывают лишь способ взаимодействия элементов системы.

В многоуровневом шаблоне система разбивается на уровни, которые на диаграмме изображаются один над другим. Каждый уровень может вызывать только уровень на 1 ниже него. Таким образом разработку каждого уровня можно вести относительно независимо, что повышает модифицируемость системы. Недостатками данного подхода являются усложнение системы и снижение производительности. Общий пример системы с использованием данного шаблона изображен на рисунке 1.

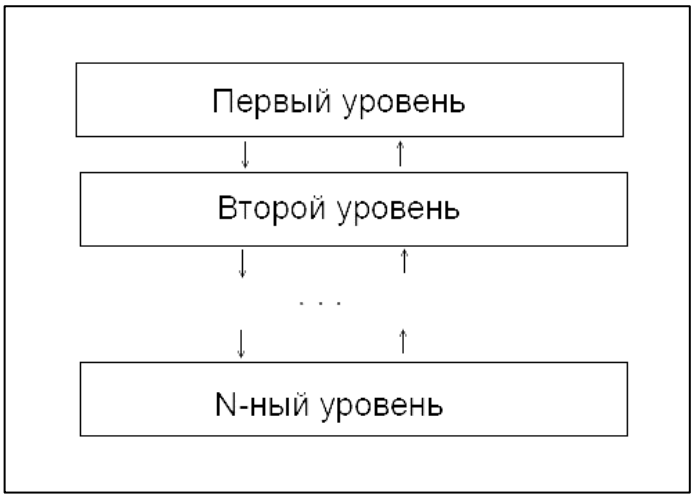

Рисунок 1 – Многоуровневый шаблон

Когда в системе присутствует большое количество модулей, их прямое взаимодействие друг с другом становится слишком сложным. Для решения проблемы вводится посредник, по которой модули общаются друг с другом. Таким образом, повышается функциональная совместимость модулей системы. Все недостатки вытекают из наличия посредника: он понижает производительность, его недоступность может сделать недоступной всю систему, он может стать объектом атак и узким местом системы [8].

В MVC шаблоне разделяется на 3 составляющих:

1) модель, хранящая данные;

2) представление, отображающее данные пользователю, а также позволяет ему взаимодействовать с ними;

3) контроллер, в свою очередь, является посредником между моделью и представлением.

При использовании MVC шаблона сохраняется корректное взаимодействие с данными (чтение, сохранение), в случае, когда требуется изменить представление данных. Для этого в шаблоне интерфейс отделен от данных. Это позволяет менять интерфейсы, равно как и создавать их разные варианты. Однако, концепция MVC имеет и свои недостатки. В частности, из-за усложнения взаимодействия падает скорость работы системы.

Использование шаблона «клиент/сервер» наиболее распространенный способ организации высокоуровневого сетевого взаимодействия. Этот шаблон позволяет повысить масштабируемость и доступность системы, т.к. клиент и сервер какого-либо ресурса могут находится как в рамках одной вычислительной системы, так и на различных компьютерах, связанных сетью. Но при этом сервер может стать узким местом системы, при его недоступности становится недоступна вся система [9].

Чтобы спроектировать мобильное приложение нужно знать, что такое архитектура, система, характеристика продукта, а также учитывать требования, которые будут предъявляется к новому проекту.

Теперь произведём анализ требований целевой аудитории, с которой предстоит столкнуться команде разработчиков.

1.1 Анализ требований целевой аудитории к разрабатываемому программному продукту

Требования могут выражаться в виде текстовых утверждений и графических моделей.

В классическом техническом подходе совокупность требований используется на стадии проектирования программного обеспечения (ПО). Требования также используются в процессе проверки ПО, так как тесты основываются на определённых требованиях.

Этапу разработки требований может предшествовать технико-экономическое обоснование или концептуальная фаза анализа проекта. Фаза разработки требований может быть разбита на выявление требований (сбор, понимание, рассмотрение и выяснение потребностей заинтересованных лиц), анализ (проверка целостности и законченности), спецификация (документирование требований) и проверка правильности.

Пользовательские требования – определяют набор пользовательских задач, которые должна решать программа, а также способы (сценарии) их

решения в системе. Пользовательские требования могут выражаться в виде фраз утверждений, в виде сценариев использования, пользовательских историй, сценариев взаимодействия.

На этом этапе разработчикам стоит определиться, зачем они планируют использовать приложение, какова итоговая цель разработки мобильного инструмента коммуникации с аудиторией [10]. Для более точного анализа был сформирован ряд вопросов, на которые необходимо ответить, прежде чем перейти к созданию технического задания.

Каких целей вы планируете достичь посредством создания и релиза собственного мобильного приложения?

Планируются ли продажи/конверсия переходов в продажу товаров и услуг в рамках приложения?

Кто ваша целевая аудитория и за счет кого она может пополниться?

Насколько высока конкуренция в сфере, в которой вы планируете работать (в том числе – с приложением)?

Какими приложениями пользуется ваша аудитория и аудитория ваших конкурентов, пересекаются ли они между собой? Готовы ли они пользоваться вашим приложением вместо приложений-аналогов?

Каков бюджет на разработку и продвижение полученного приложения?

На первом же командном собрании команда попыталась ответить на поставленные вопросы, а краткие ответы представлены ниже.

Создать уникальное приложение, которое поможет пользователю контролировать своё время, создавать заметки, которые точно не потеряются в отличии от бумажных собратьев, а также благодаря встроенной карте и push-уведомлениям напомнит сделать все дела и посетить все места, которые запланированы.

Основную прибыль приложению будет приносить договоры с рекламными компаниями.

Основная целевая аудитория – это люди, связанные с бизнесом, у которых каждая секунда на счету, дальше идут семейные люди, которым

очень часто необходимо делать какие-либо пометки и чаще всего они используют какие-то бумажки, которые потом могут легко потеряться. Так же планируется за счёт умной карты и возможности создания «общих» заметок привлечь пользователей, которым необходимо быстро общаться между собой, при этом экономя ценное время.

На рынке есть очень много приложений с такой тематикой, но разрабатываемое приложение будет иметь ряд преимуществ, за счёт этих преимуществ команда планирует выиграть конкуренцию на рынке и получить место в этой нише.

Самые популярные приложения – это Google-календарь системы Android и стандартное приложение от Apple на системе iOS. За счет дополнительных функций в разрабатываемом приложении планируется привлечь пользователей конкурентов, либо предложит свое приложения более крупным корпорациям.

Основная цель работы – это создание работоспособного продукта, который сможет приносить реальную помощь окружающим. Если проект будет действительно популярным, то за счет различных платных дополнений и рекламной интеграции, проект может стать финансово успешным.

Итак, благодаря сформулированным вопросам команда получила понимание того, с чем ей придётся столкнуться при создании проекта. Какой бюджет проекта, кто целевая аудитория, какие есть конкуренты и т.д.

Управление персоналом программного проекта очень важный пункт, о котором необходимо поговорить на начальных этапах проекта.

1.2 Управление персоналом программного проекта

Управление персоналом проекта – это процесс обеспечения эффективного использования человеческих ресурсов проекта, к которым относятся все участники проекта (спонсоры, заказчики, команда проекта, субподрядчики, подразделения компании и другие участники проекта).

Для успешного достижения целей проекта критически важным является следующее:

– идентифицировать состав участников проекта;

– определить роли участников проекта и порядок их взаимодействия;

– сформировать команду проекта и команду управления проектом;

– построить необходимую и достаточную для управления проектом организационную структуру.

Роль в проекте (проектная роль) – определенный набор функций и полномочий в проекте, созданный с целью распределения обязанностей между членами команды проекта [11]. Проектную роль можно рассматривать как временную должность в организации (компании).

Участники проекта – организации заказчика и исполнителя и специалисты от организаций заказчика и исполнителя, а также другие организации и лица, которые участвуют в работе проекта или чьи интересы могут быть затронуты при исполнении или завершении проекта [12]. Участники оказывают влияние на проект и его результаты.

Команда проекта – временная рабочая группа, выполняющая работы по проекту и ответственная перед руководителем проекта за их выполнение [13]. Команда проекта состоит из команды управления, участников проекта, выполняющих работы в рамках проекта, исполнителей проекта.

Команда управления проектом – члены команды проекта, уполномоченные принимать управленческие решения по управлению проектом [14].

Состав команды управления должен быть достаточным, чтобы осуществлять:

а) управление ресурсами проекта, в том числе:

1) определение требуемых для достижения целей проекта ресурсов;

2) подготовку предложений по изменению состава группы управления проектом;

3) утверждение персональных изменений в составе рабочих групп проекта;

4) оценку стоимости проекта, подготовку бюджетов проекта и отчетов об исполнении бюджетов.

б) управление сроками выполнения проекта, в том числе:

1) подготовку плана работ проекта;

2) контроль над выполнением проекта;

3) подготовку отчетов о ходе работ проекта.

в) управление качеством проекта, в том числе:

1) контроль соответствия разрабатываемых проектных решений техническому заданию;

2) организацию экспертизы проектных решений.

г) управление рисками проекта, в том числе:

1) анализ рисков проекта;

2) разработку планов мероприятий по снижению рисков;

3) реализацию мероприятий по снижению рисков.

д) управление проблемами проекта, в том числе:

1) анализ проблем проекта;

2) разработку мероприятий по разрешению проблем проекта;

3) реализацию мероприятий по разрешению проблем проекта.

е) контроль над организацией работ в проектных группах, в том числе:

1) согласование отчетов о ходе работ;

2) контроль над функционированием системы сбора и распределения информации;

3) контроль документирования проектных результатов.

Управление командой проекта это сложны процесс, где необходимо объявить роли каждого в проекте, разработать нужную документацию и определить, как будет контролироваться процесс хода работы.

Теперь необходимо объяснить роли каждого участника в проекте.

1.2.1 Распределение ролей в проекте

Адекватное и гибкое ролевое распределение – это эффективный метод повышения конкурентоспособности команды, ее устойчивости к негативному влиянию разнообразных внешних и внутренних факторов [15].

Другой, более тонкий, аспект распределения ролей заключается в том, чтобы дать каждому члену команды почувствовать свою значимость и перспективу роста. Это чрезвычайно важно, так как команда – это сообщество «равных». Однако за равенством иногда может потеряться индивидуальность каждого, т.к. дело общее. А как же индивидуальный вклад? И в этом случае ощущение своей роли как бы «страхует» индивидуальный вклад каждого члена команды в общее дело [16].

Главный ресурс команды заключается и в том, что члены команды могут «подстраховать» друг друга в сложной ситуации. Возможность «примерить» на себя разнообразные роли формирует дополнительный ресурс для выполнения членами команды своей «страховочной» функции. Чтобы сочетать «игровой момент» с ощущением индивидуального вклада в общее дело, к названиям ролей нужно подходить творчески, не скупясь на яркие образы, метафоры. К выделению, называнию и распределению ролей команда может посвятить специальное время. Такие дискуссии проходят весело и создают дополнительные ресурсы для поддержания «командного духа» [17].

Факторы, определяющие роли в команде: профессиональная деятельность, должностные обязанности; взаимодействие команды с внешними партнерами, клиентами; «склад ума» каждого члена команды и конкретные ситуации; процесс жизнедеятельности команды и динамика ее успешного развития.

Каждый член команды обладает определенными интеллектуальными особенностями. Один «фонтанирует» новыми идеями, другой лучше ориентируется среди готовых инструкций, третий склонен видеть все «в черном цвете», четвертый любит пофилософствовать.

Нередко эти особенности начинают раздражать членов команды. Однако если их грамотно использовать при решении проблем, это принесет команде ощутимую пользу. Важно правильно распределить роли [18].

Тестировщик – специалист, который оценивает качество программного обеспечения [19]. Он выполняет две функции: пользователя и эксперта ПО. Тестировщики занимаются поиском ошибок и сбоев в функционировании программ, а затем дают программистам обратную связь. Работа тестировщика кропотливая, а потому требует внимательности, терпения и настойчивости, готовности трудиться над совершенствованием программы от версии к версии. Тестер должен обладать отличной памятью и аналитическим мышлением, быть уравновешенным и рассудительным. Такому специалисту помогает коммуникабельность и умение работать в команде. Ценится готовность четко следовать правилам, но при этом приветствуется инициативность и любопытство, интерес к экспериментам. Тестировщик – это, по сути, инженер и пользователь в одном лице, поэтому он должен уметь анализировать продукт с позиций обоих [20].

Программист-разработчик – специалист, занимающийся написанием и корректировкой программ для компьютеров (любых вычислительных устройств), то есть программированием [21]. Большинство разработчиков ПО как рациональных интровертов, следует относить к флегматикам. Это означает, что программисты - люди спокойные как внешне, так и внутренне. Они настойчивые и упорные труженики, но им требуется время для раскачки, для сосредоточения внимания, для переключения внимания на другой объект. Интровертная рациональность делает их сдержанными и закрытыми, не отвлекающимися на внешние раздражающие факторы. Это вовсе не означает, что программист не способен вспылить. Просто, чтобы довести его

до такого состояния, надо потратить много сил. Это самый уравновешенный из всех темпераментов. Видимо, люди с другими темпераменты в условиях постоянной неопределенности и изменений, жесткого давления сроков и заказчиков, выживают не так успешно [22].

Архитектор программного обеспечения (ПО) – проектная роль в разработке ПО, профессия, возможно – позиция/должность, ключевая обязанность по которой – проектирование архитектуры ПО, т.е. принятие ключевых проектных решений относительно внутреннего устройства программной системы и её технических интерфейсов [23].

Тим-лидер (IT-менеджер проекта) руководит ИТ – отделом предприятия, который занимается внедрением информационной системы, позволяющей оценивать, контролировать и принимать решения в отношении состояния предприятия [24]. Задачей ИТ-менеджера является выбор необходимых предприятию средств автоматизации, с минимизацией затрат времени и ресурсов на их освоение, настройку и внедрение. В частности, он отвечает за автоматизацию таких областей, как управление сетевым оборудованием, серверами и корпоративными приложениями, хранением и безопасностью данных, управлением парком персональных компьютеров и службой поддержки.

Технический писатель – специалист, занимающийся документированием в рамках решения технических задач, в частности разработки программного обеспечения [25]. Основная задача технического писателя – написание документа, который бы удовлетворял определённым требованиям [26]. Требования могут определяться как нормативными актами, существующими в отрасли применения продукта, так и различными целями, которые организация-разработчик ставит перед собой. Например, сокращение расходов по сопровождению продукта путём разработки точного и понятного описания или обеспечение документированием процесса разработки для последующего подтверждения соответствия системе качества. Как правило, технические писатели компетентны как в области

языкознания, так и в технической области. Квалифицированный технический писатель умеет создавать, редактировать, иллюстрировать и адаптировать технический материал лаконично и понятно.

Интегратор – специалист, выполняющий объединение модулей в один проект. Требования – коммуникабельность, умение видеть систему «в целом» [27].

Дизайнер – человек занимающийся художественно-технической деятельностью в рамках какой-либо из отраслей дизайна [28].

Проектировщик – это специалист, занимающийся разработкой специальных планов и схем [29]. Особенность рассматриваемой специальности заключается в том, что работники в сфере построения проектов могут работать практически в любой профессиональной среде. лавная цель - разработка на заказ разного рода чертежей, схем, планов и др.

Команда состоит из двух человек и роли между ними были распределены следующим образом:

Водянкин А. И. – разработчик серверной части приложения, проектировщик, дизайнер, системный интегратор, системный архитектор.

Гуров И. И. – руководитель проекта, разработчик клиентской части приложения, тестировщик, технический писатель.

Итак, распределение ролей является важным этапом работы над проектом. Прежде чем приступить к нему, было необходимо узнать о каждой роли как можно больше, чтобы четко распределить ответственность.

Для того чтобы следить за ходом выполнения работы нужно большое внимание уделить современным инструментам интеграции программных модулей и выбрать лучше из этих средств, чтобы команда смогла максимальное комфортно разрабатывать новое приложения отвечая каждый за свой модуль.

1.2.2 Современные инструменты интеграции программных модулей

При проектировании, реализации и тестировании компонентов структурной иерархии, полученной при декомпозиции, применяют два подхода [30]:

– восходящий;

– нисходящий.

При использовании восходящего подхода сначала проектируют и реализуют компоненты нижнего уровня, затем предыдущего и т. д. По мере завершения тестирования и отладки компонентов осуществляют их сборку, причем компоненты нижнего уровня при таком подходе часто помещают в библиотеки компонентов.

Недостатки подхода:

– увеличение вероятности несогласованности компонентов вследствие неполноты спецификаций;

– наличие издержек на проектирование и реализацию тестирующих программ, которые нельзя преобразовать в компоненты;

– позднее проектирование интерфейса, а соответственно невозможность продемонстрировать его заказчику для уточнения спецификаций и т. д.

Исторически восходящий подход появился раньше, что связано с особенностью мышления программистов, которые в процессе обучения привыкают при написании небольших программ сначала детализировать компоненты нижних уровней (подпрограммы, классы). Это позволяет им лучше осознавать процессы верхних уровней. При промышленном изготовлении программного обеспечения восходящий подход в настоящее время практически не используют.

Нисходящий подход предполагает, что проектирование и последующая реализация компонентов выполняется «сверху-вниз», т. е. вначале проектируют компоненты верхних уровней иерархии, затем следующих и так далее до самых нижних уровней. В той же последовательности выполняют и реализацию компонентов. При этом в процессе программирования компоненты нижних, еще не реализованных уровней заменяют специально разработанными отладочными модулями – заглушками, что позволяет тестировать и отлаживать уже реализованную часть.

Нисходящий подход обеспечивает:

– максимально полное определение спецификаций проектируемого компонента и согласованность компонентов между собой;

– раннее определение интерфейса пользователя, демонстрация которого заказчику позволяет уточнить требования к создаваемому программному обеспечению;

– возможность нисходящего тестирования и комплексной отладки.

При использовании нисходящего подхода применяют методы определения последовательности проектирования и реализации компонентов:

– иерархический;

– операционный;

– комбинированный.

Иерархический метод предполагает выполнение разработки строго по уровням. Исключения допускаются при наличии зависимости по данным, т. е. если обнаруживается, что некоторый модуль использует результаты другого, то его рекомендуется программировать после этого модуля. Основной проблемой данного метода является большое количество достаточно сложных заглушек. Кроме того, при использовании данного метода основная масса модулей разрабатывается и реализуется в конце работы над проектом, что затрудняет распределение человеческих ресурсов.

Операционный метод связывает последовательность выполнения при запуске программы. Применение метода усложняется тем, что порядок выполнения модулей может зависеть от данных. Кроме того, модули вывода результатов, несмотря на их вызов последними в очереди, должны разрабатываться одними из первых, чтобы не проектировать сложную заглушку, обеспечивающую вывод результатов при тестировании. С точки зрения распределения человеческих ресурсов сложным является начало работ, пока не закончены все модули, находящиеся на так называемом критическом пути.

Комбинированный метод учитывает следующие факторы, влияющие на последовательность разработки:

1) достижимость модуля – наличие всех модулей в цепочке вызова данного модуля;

2) зависимость по данным – модули, формирующие некоторые данные, должны создаваться раньше обрабатывающих;

3) обеспечение возможности выдачи результатов – модули вывода результатов должны создаваться раньше обрабатывающих;

4) готовность вспомогательных модулей – вспомогательные модули, например, модули закрытия файлов, завершения программы, должны создаваться раньше обрабатывающих;

5) наличие необходимых ресурсов.

Кроме того, при прочих равных условиях сложные модули должны разрабатываться прежде простых, так как при их проектировании могут выявиться неточности в спецификациях, а чем раньше это произойдет, тем лучше.

Нисходящий подход допускает нарушение нисходящей последовательности разработки компонентов в специально оговоренных случаях. Так, если некоторый компонент нижнего уровня используется многими компонентами более высоких уровней, то его рекомендуют проектировать и разрабатывать раньше, чем вызывающие его компоненты. И,

наконец, в первую очередь проектируют и реализуют компоненты, обеспечивающие обработку правильных данных, оставляя компоненты обработки неправильных данных напоследок.

При объектно-ориентированном программировании также обычно используется нисходящий подход. В соответствии с рекомендациями подхода вначале проектируют и реализуют пользовательский интерфейс программного обеспечения, затем разрабатывают классы некоторых базовых объектов предметной области, а уже потом, используя эти объекты, проектируют и реализуют остальные компоненты.

Система Git была создана для управления разработкой ядра Linux и использует подход, который в корне отличается от CVS и SVN.

В основу Git закладывались концепции, призванные создать более быструю распределенную систему контроля версий, в противовес правилам и решениям, использованным в CVS. Так как Git разрабатывалась главным образом под Linux, то именно в этой ОС она работает быстрее всего.

Git также работает на Unix-подобных системах (как MacOS), а для работы на платформе Windows используется пакет mSysGit.

Программный код может быть недоступен, когда используется компьютер без репозитория. Для решения этой проблемы есть обходные пути, и некоторые разработчики полагают, что быстродействие Git является справедливой платой за неудобства.

Кроме того, в Git есть множество инструментов для навигации по истории изменений. Каждая рабочая копия исходного кода содержит всю историю разработки, что крайне полезно, когда программируешь без Интернет-соединения.

Преимущества:

– прекрасно подходит для тех, кто не приемлет CVS/SVN;

– значительное увеличение быстродействия;

– дешевые операции с ветками кода;

– полная история разработки доступная офлайн;

– распределенная, пиринговая модель.

Недостатки:

– высокий порог вхождения (обучения) для тех, кто ранее использовал SVN;

– ограниченная поддержка Windows (по сравнению с Linux).

Git это очень популярная и надёжная система для интеграции модулей проекта. Git позволяет создавать множество веток приложений. Так же для него разработаны различные desktop приложения, которые позволяют ускорить процесс обмена данными. Это очень важно для современной разработки.

Теперь необходимо грамотно распределить работы по всему проекту, для лучшего понимание целей каждого члена команды на каждом этапе, чтобы не было проблем ответственности за результат.

1.3 Планирование работ по проекту

Планирование проекта – непрерывный процесс, направленный на определение и согласование наилучшего способа действий для достижения поставленных целей проекта с учётом всех факторов его реализации [31].

Основным результатом этого этапа является план проекта. Однако, процесс планирования не завершается разработкой и утверждением первоначального плана проекта. В ходе осуществления проекта могут происходить изменения как внутри проекта, так и во внешнем окружении, которые требуют уточнения планов, а часто значительного перепланирования. Поэтому процессы планирования могут осуществляться на протяжении всего жизненного цикла проекта, начиная с предварительного укрупненного плана в составе концепции проекта, и заканчивая детальным планом работ завершающей фазы проекта [32].

Планирование – комплексная, многокритериальная функция, предполагающая рассмотрение, анализ и прогнозирование нескольких функциональных областей проекта [33]. Планирование проекта может включать следующие процедуры:

− планирование целей и содержания проекта;

− календарное планирование работ проекта;

− планирование затрат и финансирования проекта;

− планирование качества;

− организационное планирование;

− планирование коммуникаций;

− планирование управления рисками;

− планирование контрактов;

− разработка сводного плана проекта.

При создании проекта, его делят на 3 части. Первая часть предварительная работа по проекту, команда продумывает основные идеи проекта и как их реализовать. Сюда входят такие пункты как:

− анализ данных;

− изучение предметной области;

− разбиение ролей;

− заключение договора, создание необходимой документации.

На втором этапе происходит создание различных отдельных модулей приложения с затычками, что позволяет разрабатывать отдельные модули независимо друг от друга. Создаётся интерфейс программы.

На заключительном этапе происходит соединение всех разработанных модулей. Тестирования модулей и связей между ними. После этого проект показывают заказчику и делают финальные поправки.

Дальше проект отдаётся заказчику, и он делает релиз ПО, и может быть заключён договор о поддержки проекта, но это не обязательная часть проекта.

Таблица 1 – Планирование проекта

№	Веха	Плановая дата
1)	Анализ данных, изучение предметной области	09.17
2)	Проектирование архитектуры проекта	10.17
3)	Реализация интерфейса	11.17
4)	Реализация добавления, редактирования, удаления элементов расписания	12.17
5)	Реализация синхронизации расписания между необходимыми узлами	01.18
6)	Реализация «Интерактивной карты»	02.18
7)	Реализация системы уведомлений	03.18
8)	Тестирование и внедрение	04.18-05.18

В таблице 1 представлен краткий план разработки приложения TMS.

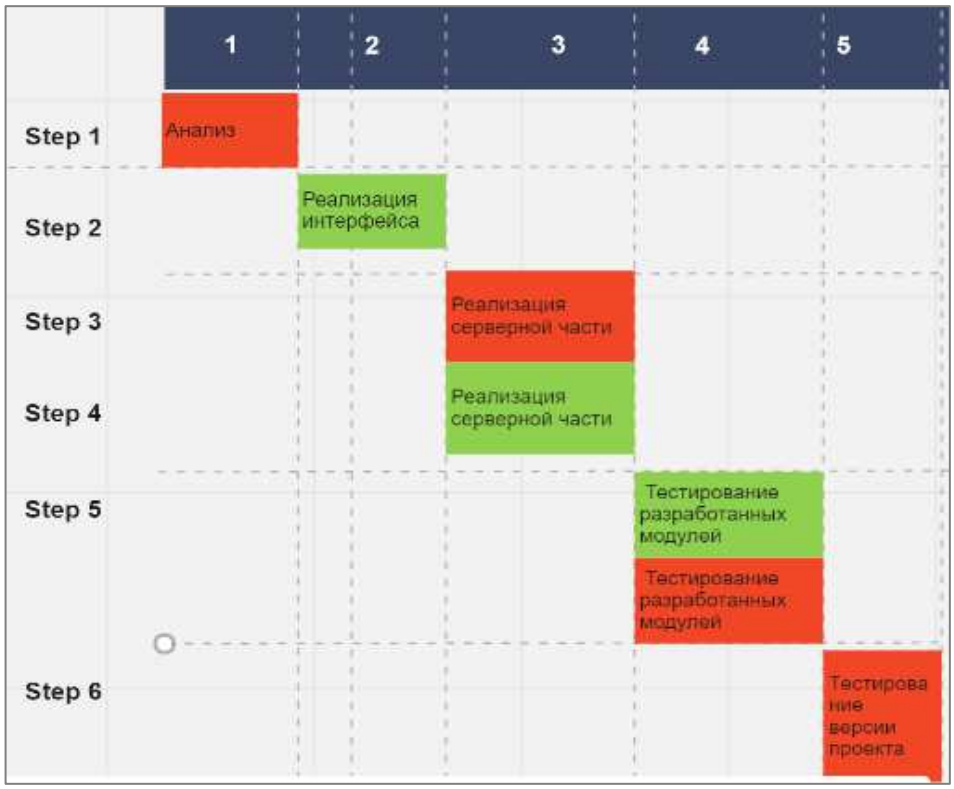

Рисунок 2 – Часть диаграммы Ганта

На рисунке 2 представлена часть диаграммы Ганта – это популярный тип столбчатых диаграмм (гистограмм), который используется для иллюстрации плана, графика работ по какому-либо проекту. Оранжевым

цветом отмечена часть, которую выполнял Гуров Иван, зелёной частью отмечена работа Водянкина Алексея.

Планирование работ – это один из первоочередных пунктов на ранних этапах разработки проекта, все время разработки можно поделить на три части: подготовка, разработка и сопровождение.

После того, когда в проекте распределены роли и обязанности, а также есть небольшое представление о новом проекте, необходимо провести глубокий анализ предметной области.

2 Системный анализ предметной области

Поведение потребителей ограничено не только денежными ресурсами, но и ресурсами времени.

Традиционно считалось, что временной бюджет, который имеют потребители, можно разделить на две части: рабочее время и свободное время.

В соответствии с современными представлениями, временной бюджет делится на три части: рабочее, неличное и свободное время. Рабочее время – это оплачиваемое время. Свободное время – это личное время потребителя, которое не оплачивается, но приносящее пользу потребителю. Неличное время – время, предназначенное для выполнения различных обязанностей (физических, социальных и моральных) [34].

Составляющие временных ресурсов потребителя конкурируют между собой так же, как и товары, предназначенные для различных видов деятельности.

Бизнес-процесс – это регулярно повторяющаяся последовательность взаимосвязанных мероприятий (операций, процедур, действий), при выполнении которых используются ресурсы внешней среды, создается ценность для потребителя и выдается ему результат.

Потребитель может быть как внешним, так и внутренним по отношению к организации. Внешний потребитель – это потребитель, который не входит в состав данной организации, а внутренний – тот потребитель, который находится в рамках данной организации.

Важно знать потребителя бизнес-процесса потому, что именно он явным или неявным образом задает требования к процессу и может оказывать влияние даже на сам факт существования конкретного процесса.

Моделирование бизнес-процессов осуществлялось при помощи CASE-средства ERwin [35].

На рисунке 3 изображена контекстная диаграмма.

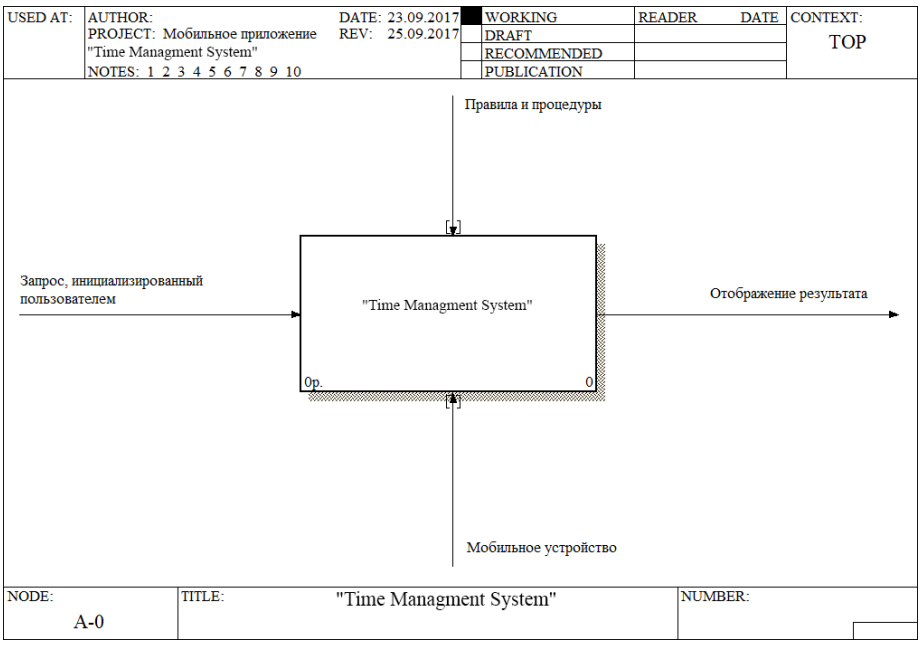

Рисунок 3 – Контекстная диаграмма (Уровень TOP)

Рисунок 4 содержит верхний уровень диаграммы декомпозиции A0. Данный уровень содержит такие работы, как: обработка запроса, обработка данных, обработка вывода.

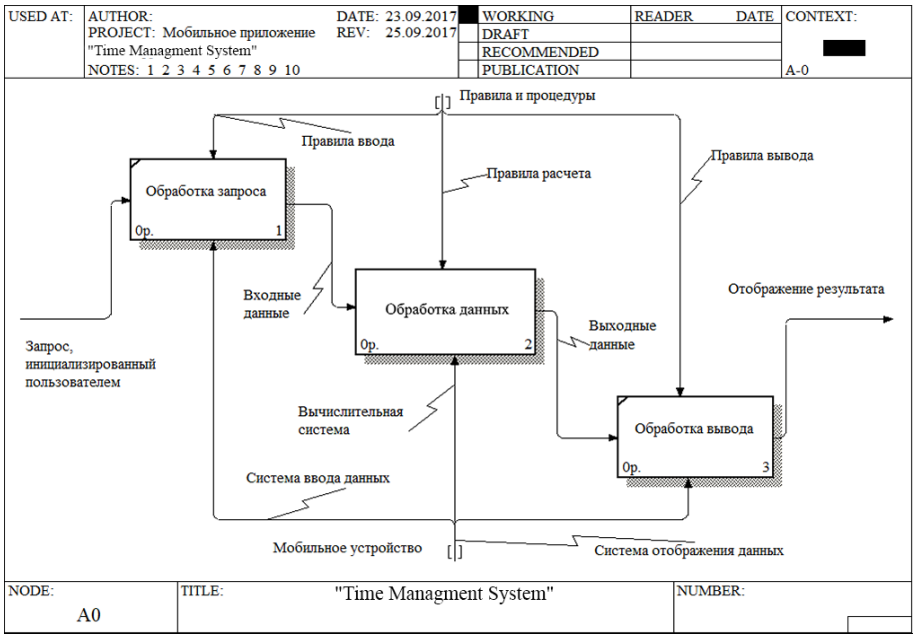

Рисунок 4 – Диаграмма декомпозиции A0

Диаграмма декомпозиции A2 отображена на рисунке 5. Здесь расположены работы: работа с расписанием, заметками и картой.

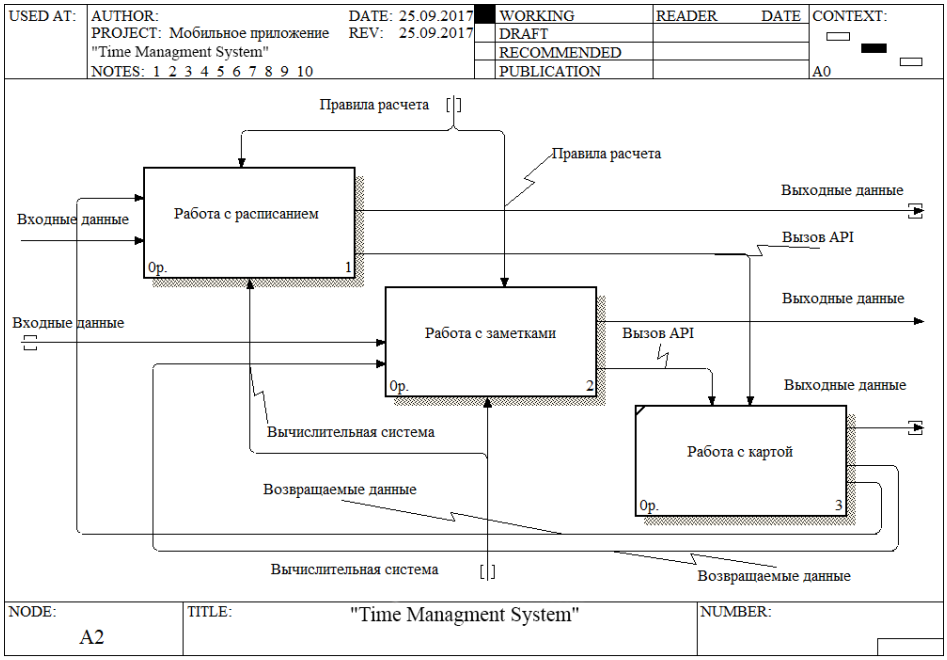

Рисунок 5 – Диаграмма декомпозиции A2

Декомпозиция работы с расписанием и заметками приведена на рисунке 6. Данные работы декомпозируются на: заполнение полей формы и сохранение данных.

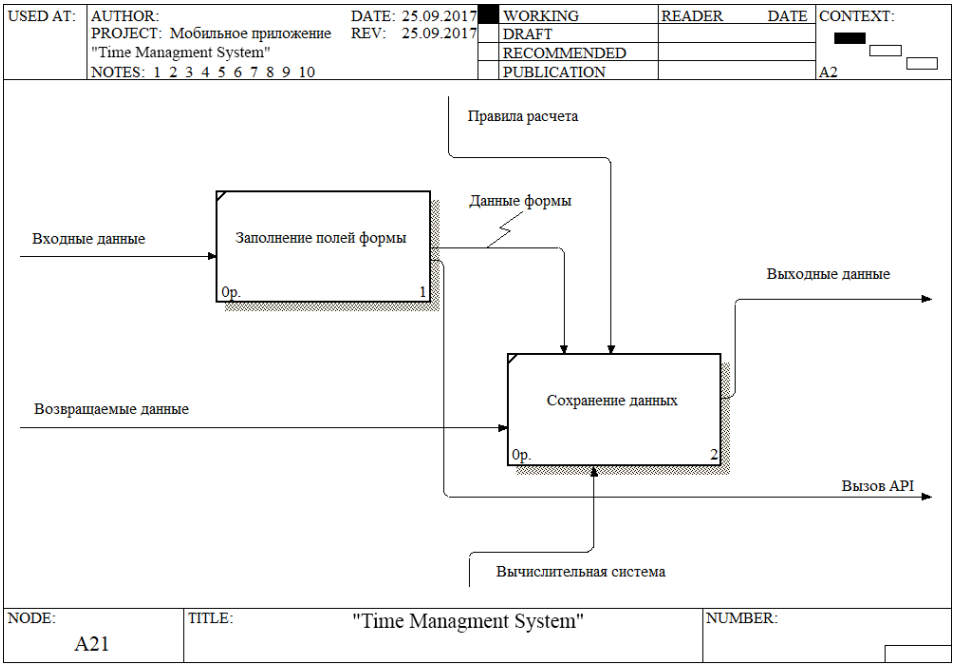

Рисунок 6 – Диаграмма декомпозиции A21

Для наглядного представления работ составлена диаграмма дерева узлов, показанная рисунком 7.

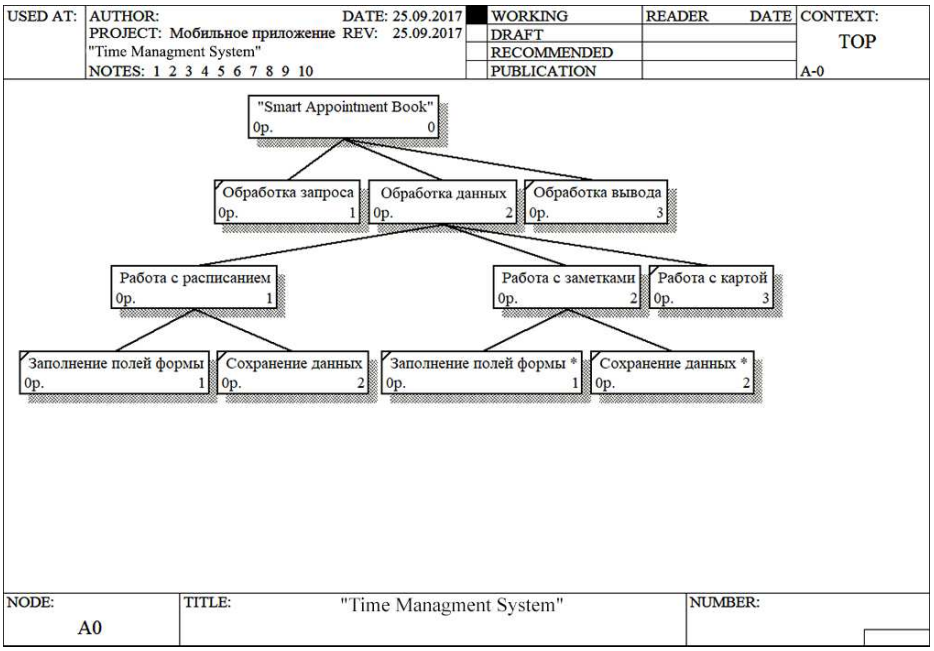

Рисунок 7 – Диаграмма дерева узлов

Анализ предметной области – это та часть проекта, которая описывает бизнес-процессы и показывает, как данные будут перемещаться по приложению и какие условия необходимы для этого.

Для того чтобы выиграть конкуренцию у других продуктов необходимо привнести что-то новое в проект, для этого нужно выбрать подходящие инструменты разработки.

2.1 Обоснование выбора инструментов разработки

Выбор средств разработки осуществляется на первых стадиях работы над проектом, после определения требований к создаваемому программному продукту. Он, как и анализ существующих альтернативных программных продуктов, является экономическим процессом, в ходе которого выбираются наиболее оптимальные средства разработки для конкретного проекта.

Например, по соотношению стоимости средства к необходимым функциям, по времени и стоимости обучения работы со средством разработки и т.д.

Существуют различные интегрированные среды разработки, предназначенные для создания мобильных приложений для операционной системы Android: IntelliJ IDEA, Netbeans, Eclipse, Android Studio, Visual Studio (последние версии) [36].

В зависимости от выбора среды разработки следует использовать соответствующий из доступных языков программирования: Java, C#, C/C++, Python, Kotlin, Lua.

Для разработки веб-приложений также существует не одно средство для написания программного кода. Здесь присутствуют, как полноценные интегрированные среды разработки, например, WebStorm и Visual Studio, так и простые текстовые редакторы с подсветкой синтаксиса [37].

Список доступных языков программирования также велик: JavaScript, Java, PHP, Python, Ruby, Perl, ASP.Net.

Язык программирования JavaScript – мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили [38]. Является реализацией языка ECMAScript (стандарт ECMA-262). JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам. Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

СУБД MySQL является наиболее приспособленной для применения в среде web-СУБД [39]. Не секрет, что для исполнения приложений клиента на большинстве хостинг площадок провайдеры предоставляют небольшое количество ресурсов (как вычислительных, так и дисковых). Поэтому для данного применения необходима высокоэффективная СУБД, обладающая

при этом высокой надежностью. Для общения с СУБД MySQL применяется язык SQL (Stuctured Query Language – язык структурированных запросов). В настоящее время SQL является стандартом работы с базами данных, и все основные СУБД понимают его. SQL включает много разных типов операторов, разработанных для взаимодействия с базами данных.

Программная платформа Node.js и Express – полностью самостоятельная платформа, включающая, кроме движка, встроенный сервер (HTTP и TCP/UDP/Unix-soket) и базовый набор библиотек, а также предоставляющей полностью асинхронную работу с файлами и сетевыми устройствами. Node.js имеет преимущества перед Java это было доказано компанией Plurk (азиатский аналог твиттера), которые полностью перенесли свой comet-сервер, изначально написанный на Java и солидном JBoss Netty, на Node.JS и, по отзывам, сократили потребление памяти буквально на гигабайты [40].

Платформа Android SDK – универсальное средство разработки мобильных приложений для операционной системы Android [41]. Отличительной чертой от обычных редакторов для написания кодов является наличие широких функциональных возможностей, позволяющих запускать тестирование и отладку исходных кодов, оценивать работу приложения в режиме совместимости с различными версиями ОС Android и наблюдать результат в реальном времени (опционально). Поддерживает большое количество мобильных устройств, среди которых выделяют: мобильные телефоны, планшетные компьютеры, умные очки (в том числе Google Glass), современные автомобили с бортовыми компьютерами на ОС Android, телевизоры с расширенным функционалом, особые виды наручных часов и многие другие мобильные гаджеты, габаритные технические приспособления.

Atom – бесплатный текстовый редактор с открытым исходным кодом для Linux, macOS, Windows с поддержкой плагинов, написанных на Node.js, и встраиваемых под управлением Git [42]. Большинство плагинов имеют

статус свободного программного обеспечения, разрабатываются и поддерживаются сообществом. Atom основан на Electron (ранее известный как Atom Shell) – фреймворке кроссплатформенной разработки с использованием Chromium и io.js. Редактор написан на CoffeeScript и LESS.

Git является распределенной системой для управления версиями разрабатываемых файлов. Создана она была в 2005 году автором ОС Linux. Эта система осуществляет синхронизацию работы с сайтом, а также сохраняет и обновляет изменения в файлах. Это очень удобный подход в случае работы над проектом нескольких разработчиков. На сегодняшний день во многих известных проектах используется именно Git [43].

Visual Studio Code – редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для кастомизации: пользовательские темы, сочетания клавиш и файлы конфигурации. Распространяется бесплатно, разрабатывается как программное обеспечение с открытым исходным кодом, но готовые сборки распространяются под проприетарной лицензией.

AsyncStorage это простая, незашифрованная, асинхронная, постоянная система хранения ключа-значения, глобальная для приложения. Его следует использовать вместо LocalStorage.

React Native - это JS-фреймворк для создания нативно отображаемых iOS- и Android-приложений. В его основе лежит разработанная в Facebook JS-библиотека React, предназначенная для создания пользовательских интерфейсов. Но вместо браузеров она ориентирована на мобильные платформы. Иными словами, если вы веб-разработчик, то можете использовать React Native для написания чистых, быстрых мобильных приложений, не покидая комфорта привычного фреймворка и единой кодовой базы JavaScript [42].

Облачная платформа Microsoft Azure была признана Compuware самой быстрой «облачной» платформой. Также тестовый пакет LINPACK показал у Microsoft Azure высокую производительность для масштабных вычислений – с результатами в 151,3 ТФлопс на 8064 ядрах с 90,2-процентной эффективностью. В отчетах-исследованиях провайдеров облачных сервисов хранения данных 2013 и 2015 года, проведенных компанией Nasuni, платформа Microsoft Azure является лидером в тестах производительности при записи и чтении данных из облака, доступности данных и минимальному числу ошибок. Есть возможность для студентов бесплатно использовать облачную платформу.

В данный момент рынок инструментов разработки очень разнообразен, поэтому сложностей при выборе технологий разработки не возникло, каждый может выбрать то, в чем ему комфортно работать.

После выбора инструментов разработки, необходимо произвести анализ конкурентов на рынке, найти их недостатки и плюсу, которые необходимо учесть при создании собственного проекта.

2.2 Сравнительный анализ рынка программного обеспеченья в сегменте планировщиков задач

Планирование своего времени сейчас очень важная задача для многих людей, потому что время – это тот ресурс человека, который люди не научилась восполнять и максимально расходовать. Поэтому и рынок приложений планировщиков очень обширен.

Контроль каждой части временного бюджета очень важны для современного человека, поэтому различные планировщики времени очень популярны у современного пользователя мобильного гаджета.

Сейчас среди русскоговорящих пользователей мобильных устройств очень популярны такие приложения:

1) Купи батон;

2) Мой день;

3) Remember the Milk;

4) Taasky;

5) Any.Do.

Список приложений, а также плюсы и минусы данных приложений был взят из статьи популярного интернет портала «The Village».

Купи батон – это приложение позволяет составлять подробные списки покупок, что приводит к безошибочным и молниеносным действиям во время операций по приобретению еды и других товаров. Плюсы:

− список покупок можно формировать из СМС;

− различные группы продуктов разделены по цветам.

Минусы:

− нельзя удалить весь список продуктов одним движением пальца;

− чтобы составить несколько списков покупок, придётся заплатить 66 рублей за полную версию;

− отсутствует функция добавления продуктов в список с помощью сканирования штрих кодов.

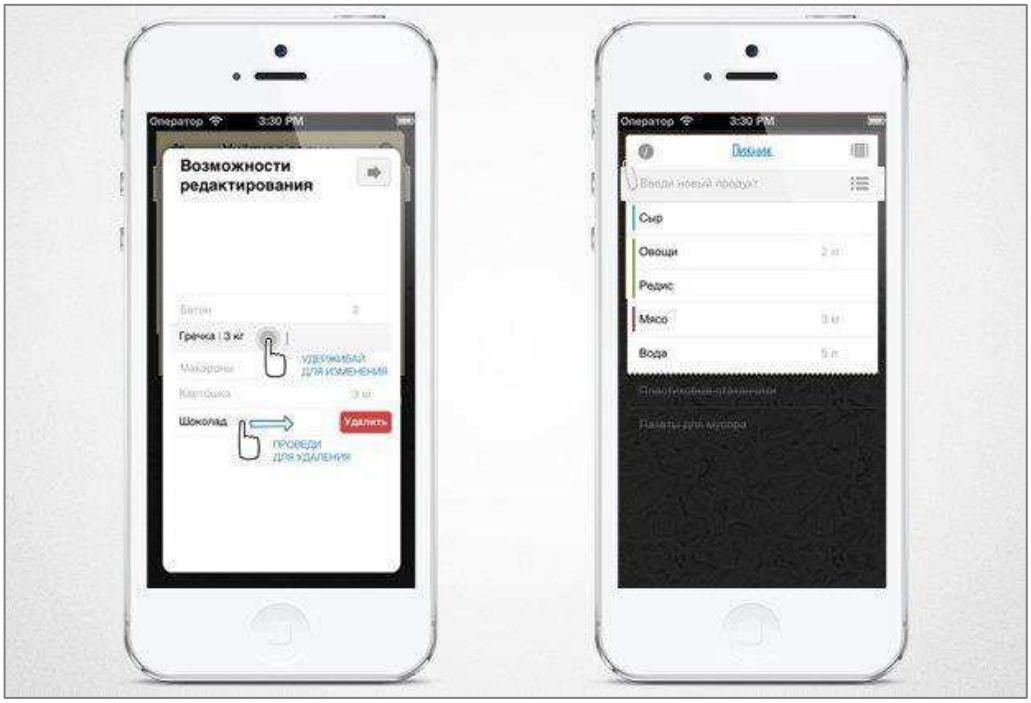

Рисунок 8 - Приложение «Купи батон»

Мой день – это приложение забив дату скорого свершения всех релевантных для вас событий, вы всегда будете иметь перед глазами воодушевляющий таймер, сообщающий, что, например, до отпуска осталось всего-то 560 дней. Плюсы:

− можно перенести дни рождения всех друзей из Facebook;

− предустановленные обои для событий можно заменять своими фотографиями;

− одновременно можно следить за неограниченным количеством событий.

Минусы:

− приложение не синхронизируется с данными из контактов телефона;

− маленькое количество предустановленных обоев.

Рисунок 9 - Приложение «Мой день»

Remember the Milk – не очень симпатичный, но проверенный временем планировщик для ретроградов, не желающих иметь ничего общего с трёхмерными интерфейсами, голосовыми командами и прочими дарами будущего. Плюсы:

− каждой задаче можно выставлять уровень важности, место, интервал и многое другое;

− не самый современный, но весьма функциональный дизайн.

Минусы:

− премиум-аккаунт, не сильно расширяющий возможности приложения, обойдётся вам в 799 рублей в месяц;

− бесплатно синхронизироваться можно только один раз в сутки.

Рисунок 10 – Приложение «Remember the Milk»

Taasky – новаторский планировщик из будущего. Первое, что бросается в глаза при знакомстве с Taasky – интересная реализация меню, выполненная

в виде подобия трёхмерного куба: проведя пальцем вниз, вы увидите поле для добавления новой задачи, а скользнув вправо, попадёте в список категорий задач. Плюсы:

− задачи можно разделять на несколько групп: дом, работа и так далее. Есть возможность добавлять новые;

− действительно удобный и простой интерфейс. Но понимание этого приходит не сразу.

Минусы:

− высокий порог вхождения для новых пользователей;

− есть риск от непонимания разбить телефон о свою голову;

− доступ к статистике вашей продуктивности будет стоить 33 рубля.

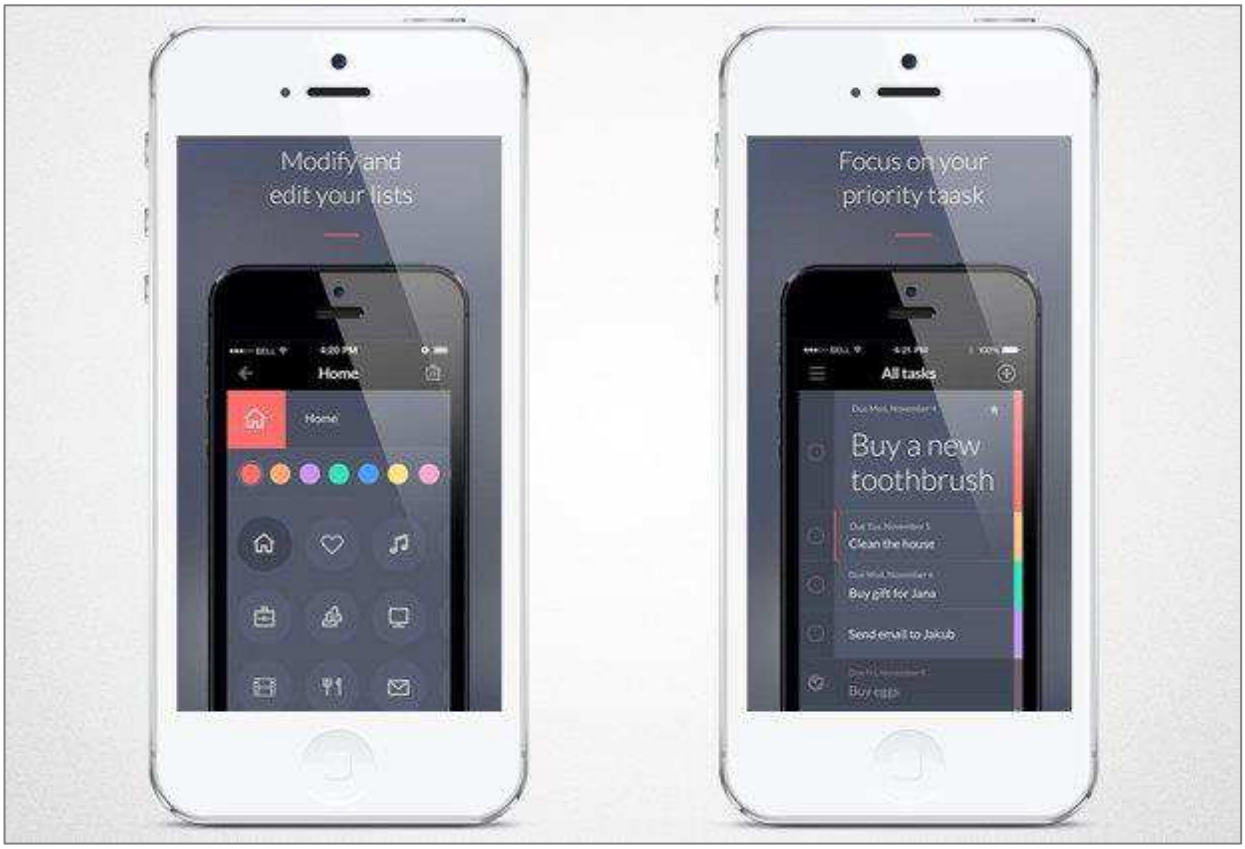

Рисунок 11 – Приложение «Taasky»

Any.Do – идеальная замена секретарше, а также всем предыдущим планировщикам из нашего списка. Если вы не любите минимализм, боитесь

киберпанковских интерфейсов из будущего и не хотите разбираться в перегруженном функционале, то у Any.Do есть все шансы подкупить вас балансом между приятным дизайном и в меру упитанным функционалом.

Плюсы:

− если у вас устали пальцы, задачи можно добавлять голосом;

− приложение синхронизируется с календарём телефона;

− любую задачу можно разделить между несколькими друзьями. Правда, для этого им тоже нужно будет зарегистрироваться в приложении;

− у Any.Do есть расширение для браузеров, позволяющее работать со списком задач даже на компьютере;

− функция Moment, которая спрятана в нижнем левом углу в виде четырёх разноцветных точек, которая волшебным образом позволит вам организовать все свои задачи в считанные секунды, указав на ошибки в их расстановке.

Минусы:

− нет функции напоминания за несколько минут до начала события;

− в версии для Android возможны вылеты.

Проведя анализ можно сделать несколько пунктов, которые необходимо учесть при создании приложения:

1) Различные группы разделены по цветам.

2) Каждой задаче можно выставлять уровень важности, место, интервал и многое другое.

3) Любую задачу можно разделить между несколькими друзьями. Правда, для этого им тоже нужно будет зарегистрироваться в приложении.

4) Приложение синхронизируется с календарём телефона.

5) Возможность просмотра и изменения задач с помощью компьютера [56].

Основным конкурентом в выбранной среде будет являться приложение Any.Do.

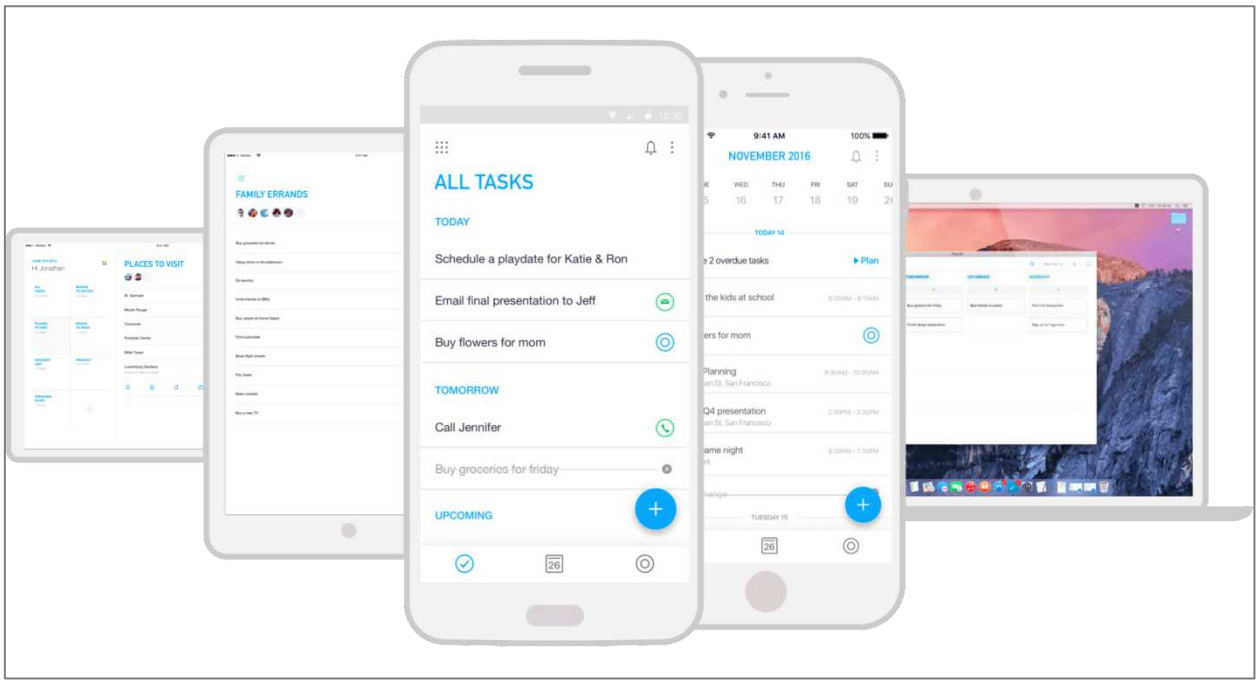

Рисунок 12 – приложение Any.Do

Благодаря анализу рынка мобильных приложений в сфере планировщик времени команде удалось узнать основного конкурента, а также минусы, которые необходимо избежать при создании приложения и плюсы, которые стоит внести в проект, если не в первой версии, то в следующих версиях программы.

Для того чтобы создать клиент-серверное приложение необходимо большое внимание уделить взаимодействию двух частей, которые не зависят друг от друга.

2.3 Проектировка приложения

После системного анализа и анализа требований идет важный этап жизненного цикла – проектирование. Именно здесь закладываются основные принципы построения приложения, поэтому необходимо уделять этому процессу должное время, чтобы минимизировать возникновение проблем и ошибок на последующих этапах создания приложения.

Проектирование программного обеспечения включает в себя следующие основные виды деятельности:

– выбор метода и стратегии решения;

– выбор представления внутренних данных;

– разработка основного алгоритма;

– документирование программного обеспечения;

– тестирование и подбор тестов;

– выбор представления входных данных.

Также, в процессе проектирования, учитывается то, что существует уже множество готовых пакетов и библиотек, которые реализуют те или иные задачи. Использование таких решений помогает минимизировать время трудоемкости. Но необходимо понимать, что лишь только грамотный выбор дает такую возможность. В противном случае, использование не нужных или не правильных библиотек может привести к увеличению затраченного времени на реализацию программного обеспечения.

Использование React Native позволяет разрабатывать приложения как под iOS, так и под Android. Поэтому хорошей практикой является проектирование экранов приложения под различные платформы. Плюс ко всему, необходимо учитывать особенности разных операционных систем, чтобы в контексте экосистемы все выглядело уместно.

Итогом на выходе этапа проектирования является набор обязательных функций и понимание общей механики с набросками ключевых экранов интерфейса. Данное приложение состоит из двух основных частей, поэтому необходимо организовать их взаимодействие, описанное в следующей главе.

2.3.1 Проектирование клиент-серверного взаимодействия

В концепции «клиент/сервер» участвует, как понятно из названия, две стороны: клиент и сервер. В нашем случае, клиент – это устройство,

оснащенное специальным программным обеспечением, которое позволяет пользователю задать запрос к другой машине и получить ответ. Сервер – это компьютер, оснащённый специальным программным обеспечением, которое позволяет решить задачи предоставления пользователю доступа к некоторым услугам и ресурсам, которыми владеет и управляет данный сервер.

Клиент-серверное взаимодействие – это обмен данными между клиентом и сервером. В свою очередь, существует архитектурный стиль взаимодействия компонентов REST, который используется для построения веб-служб. Системы, поддерживающие REST, называются RESTful-системами.

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

Отсутствие дополнительных внутренних прослоек означает передачу данных в том же виде, что и сами данные. Т.е. мы не заворачиваем данные в XML, как это делает SOAP и XML-RPC, не используем AMF, как это делает Flash и т.д. Просто отдаем сами данные.

Как происходит управление информацией сервиса – это целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протокол конечно же HTTP. HTTP методы составляют основную часть «единого интерфейса», ограничивающего и предоставляющего возможность осуществлять действия над ресурсом. Основными или наиболее часто используемыми HTTP методами являются POST, GET, PUT, и DELETE. Они соответствуют операциям создания, чтения, обновления и удаления (или в совокупности - CRUD). Есть еще и другие методы, но они используются реже, из них выделяются OPTIONS и HEAD.

HTTP метод GET используется для получения (или чтения) представления ресурса. В случае «удачного» (или не содержащего ошибок) адреса, GET возвращается представление ресурса в формате XML или JSON.

В соответствии спецификации HTTP, GET запросы используются только для чтения данных, не изменяя их. Таким образом, при соблюдении данного соглашения, они считаются безопасными. То есть они могут использоваться без риска изменения данных, вне зависимости от того, один раз данные были получены, или же 10, или ни разу вовсе.

Метод PUT обычно используется для предоставления возможности обновления ресурса. Тело запроса при отправке PUT-запроса к существующему ресурсу URI должно содержать обновленные данные оригинального ресурса (полностью, или только обновляемую часть).

Кроме того, метод PUT может быть использован для создания ресурса, в случае, когда идентификатор ресурса выбирает клиент, а не сервер, то есть запрос содержит не существующий идентификатор ресурса. Опять же, стоит помнить, что тело запроса должно быть модификацией оригинального ресурса.

Для создания новых экземпляров ресурса предпочтительнее использование POST запроса. В данном случае, при создании экземпляра будет предоставлен корректный идентификатор экземпляра ресурса в возвращенных данных об экземпляре.

DELETE запрос крайне прост для понимания. Он используется для удаления ресурса, идентифицированного конкретным URI, например, запрос содержит id.

Архитектура REST очень проста в плане использования. По виду пришедшего запроса сразу можно определить, что он делает. Данные передаются без применения дополнительных слоев, поэтому REST считается менее ресурсоемким, поскольку не надо парсить запрос, чтобы понять, что он должен выполнить, и не надо преобразовывать данные из одного формата в другой [44].

В совокупности с REST API можно использовать спецификацию JSON API, которая описывает, как клиент должен запрашивать, чтобы ресурсы были извлечены или изменены, и как сервер должен отвечать на эти запросы.

JSON API предназначен для минимизации количества запросов и количества данных, передаваемых между клиентами и серверами. Эта эффективность достигается без ущерба для удобочитаемости, гибкости или возможности обнаружения.

Одним из главных требований является использование медиа типа «application/json» в теле запроса. Тип указывает, что контент может быть проанализирован как JSON-объект.

Взаимодействие сервера с клиентом очень хрупкая, но при этом простая часть приложения, которое достаточно просто настроить.

При создании новой программы важным аспектом является выбор модели жизненного цикла разработки. При грамотном подходе во время разработки проект должен получить максимальное количество времени на каждом участке производства.

2.3.2 Проектирование архитектуры приложения

Для данного приложения был взят за основу архитектурный шаблон «клиент/сервер», что позволило разделить логику приложения. В результате чего было решено использовать отдельный сервер для централизованного хранения данных пользователей, а также логики «расшаривания» заметки или пунктов расписания доверенному кругу лиц.

Архитектура серверной части приложения изображена на рисунке 13. Модуль работы с базой данных отвечает за выполнение запросов к базе данных. Модуль авторизации реализует функции генерации токена авторизации, а также проверки токена на валидность. Модуль API – это основной контролирующий модуль, который организует взаимодействие

сервера с клиентом, а именно, принимает запросы, обрабатывает их и возвращает ответ.

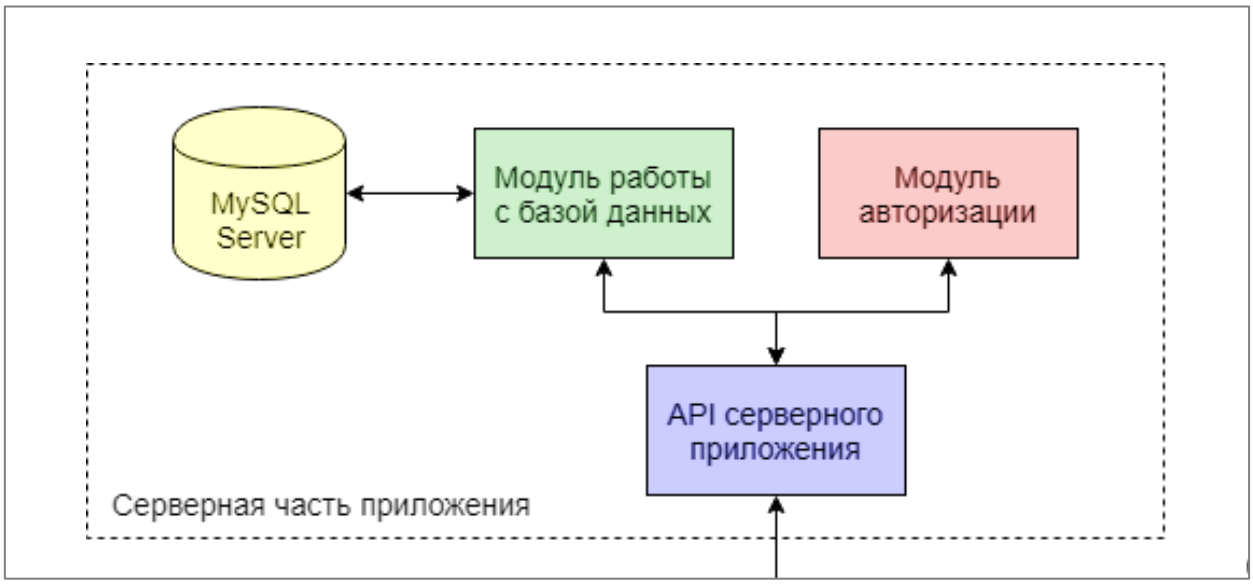

Рисунок 13 – Серверная часть архитектуры приложения

Мобильное приложение позволяет пользователю основной функционал данной системы. Здесь, описана вся логика авторизации пользователя, создания, редактирования и удаления заметок (пунктов распорядка дня). А также экран настроек приложения. Архитектура мобильного приложения отображена на рисунке 14.

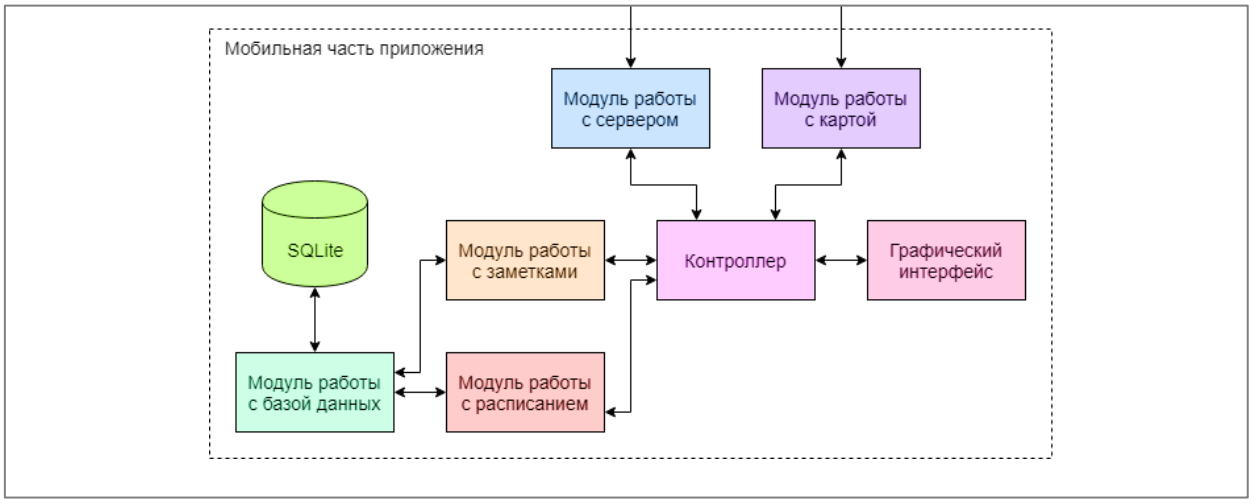

Рисунок 14 – Мобильная часть архитектуры приложения

Общая схема архитектуры представлена на рисунке 15.

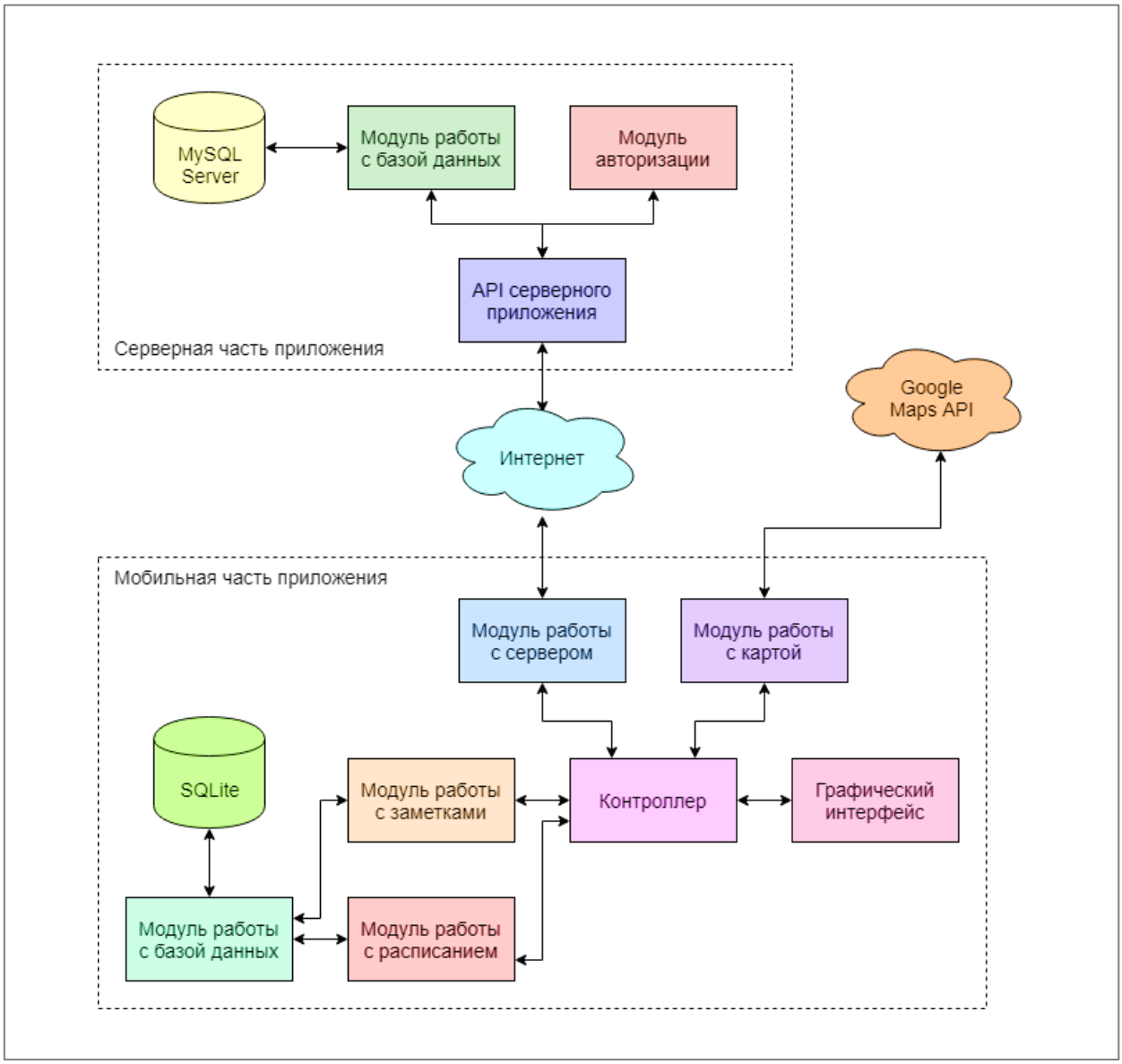

Рисунок 15 – Общая схема архитектуры приложения

Благодаря архитектуре приложения разработчики поймут, как взаимодействую модули между собой и как модули расположены в подсистемах приложения.

Дизайн приложения очень важен, потому что пользователь чаще всего выбирает то или иное приложения исходя из картинки, которая будет ему приятна.

2.3.3 Проектирование информационно-логической модели базы данных

Структура спроектированной базы данных представлена на рисунке 16.

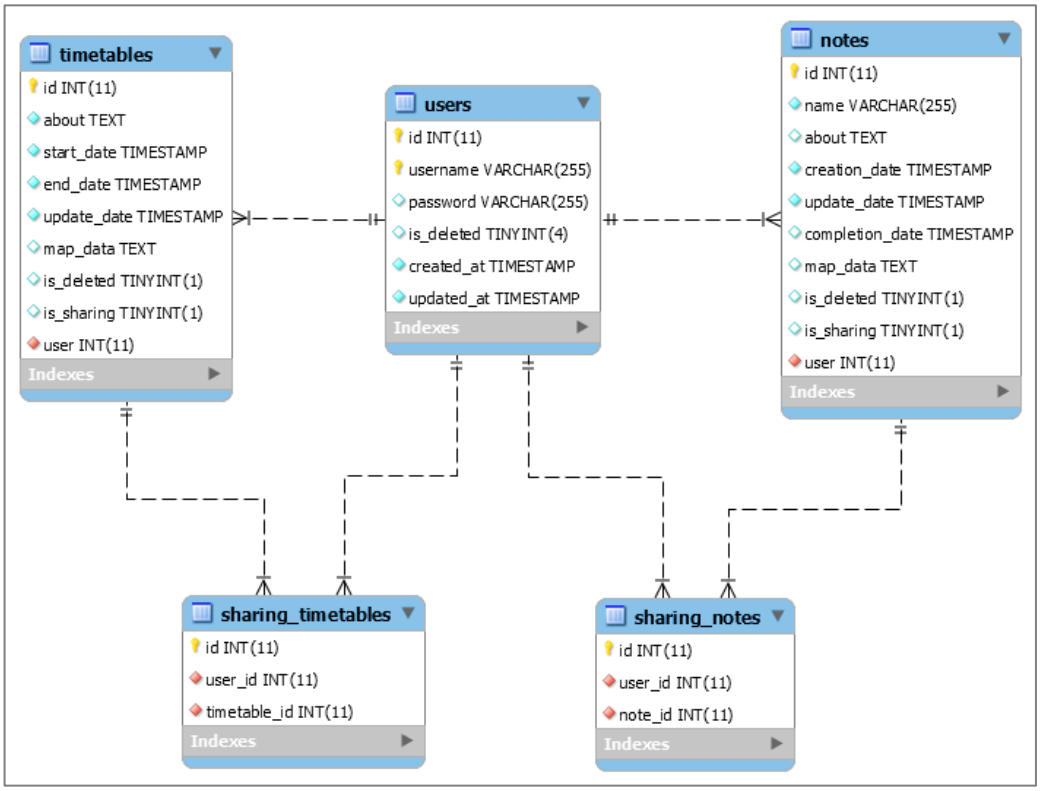

Рисунок 16 – Схема данных

Для хранения данных в серверной части разрабатываемой системы используется система управления базами данных MySQL, которая содержит следующие таблицы:

1) таблица пользователей, содержащая из полей id, имя пользователя, пароля, флаг удаления пользователя, а также два дополнительных поля – дату создания и последнего изменения;

2) таблица заметок, состоит из полей id, названия, содержания заметки, даты создания, даты последнего обновления, даты завершения, данных карты, флаг общего доступа, флаг удаления и id пользователя, который создал заметку;

3) таблица пунктов расписания включает в себя такие поля, как: id, дата начала пункта, дата завершения пункта, дата последнего обновления, текст пункта, данные карты, флаг общего доступа, флаг удаления и id пользователя, который создал пункт расписания;

4) таблица связей заметок, состоящая из id, id заметки и id пользователя, которому был дан доступ к заметке;

5) таблица связей пунктов расписания, содержащая id, id пункта расписания и id пользователя, которому был дан доступ к пункту расписания.

При проектировании таблиц были учтены такие особенности приложения, как предоставление доступа к заметке или пункту расписания другим пользователям, а также возможность добавления места.

2.4 Выбор модели жизненного цикла и технологий разработки

Scrum – это набор принципов [45], на которых строится процесс разработки, позволяющий в жёстко фиксированные и небольшие по времени итерации, называемые спринтами, предоставлять конечному пользователю работающее ПО с новыми возможностями, для которых определен наибольший приоритет (рисунок 17).

Рисунок 17– Методология Scrum

В таблице 2 представлены основные роли Scrum методологии, распределенные в команде.

Таблица 2 – Распределение ролей Scrum

Участник команды	Роли
Гуров Иван	Владелец продукта, скрам-команда
Водянкин Алексей	Скрам-мастер, Скрам-команда

Бэклог – это полный список работ, которые нужно выполнить [46]. Бэклог проекта представлен в таблице 3.

Таблица 3 – Бэклог продукта

ID	User Story	Importance (1-10)	How to demo
a-01	Как пользователь, я хочу получить доступ к приложению при помощи своей учетной записи (либо создать учетную запись, в случае ее отсутствия)	3	Демоверсия модуля
a-02	Как пользователь, я хочу создать, редактировать, удалять заметки и просматривать их список	7	Демоверсия модуля
a-03	Как пользователь, я хочу создавать, редактировать, удалять персональное расписание и просматривать его	7	Демоверсия модуля
a-04	Как пользователь, я хочу иметь круглосуточный доступ к своим данным	8	Совместная работа модулей a-02, A-03, a-04
a-05	Как пользователь, я хочу создавать автономную копию данных на своем мобильном устройстве	3	Совместная работа модулей a-02, A-03, a-05

Окончание таблицы 3

a-06	Как пользователь, я хочу предоставлять общий доступ к своим данным (заметкам или расписанию) для выбранного круга лиц	6	Совместная работа модулей a-02, A-03, a-06
a-07	Как пользователь, я хочу отмечать на карте места, упомянутые в заметке или расписании, и получать обратную связь при помощи геолокации	9	Совместная работа модулей a-02, A-03, a-07
a-08	Как пользователь, я хочу иметь простой и понятный графический интерфейс пользователя с удобной навигацией по функциям приложения	5	Демоверсия приложения

Длительность спринта для данного проекта была установлена 2 недели. Данное значение было выбрано, как среднее между наибольшим и наименьшим возможными значениями равными 3 и 1 неделя соответственно.

Визуализация процесса работы над проектом проводилась в web-приложении Trello.

Для получения полной информации о текущем состоянии проекта проводились ежедневные совещания.

По окончанию каждого спринта проводилось тестирование разработанного функционала, и собиралась демонстрационная версия разрабатываемой системы. На основе отзывов о демонстрационной версии проекта проводилась ретроспектива и планировался следующий спринт.

Спиральная модель представляет шаблон процесса разработки программного обеспечения, который сочетает идеи итеративной и каскадной моделей. Суть ее в том, что весь процесс создания конечного продукта представлен в виде условной плоскости, разбитой на 4 сектора, каждый из которых представляет отдельные этапы его разработки: определение целей, оценка рисков, разработка и тестирование, планирование новой итерации.

В спиральной модели жизненный путь разрабатываемого продукта изображается в виде спирали (рисунок 18), которая, начавшись на этапе планирования, раскручивается с прохождением каждого следующего шага. Таким образом, на выходе из очередного витка мы должны получить готовый протестированный прототип, который дополняет существующий билд. Прототип, удовлетворяющий всем требованиям – готов к релизу.

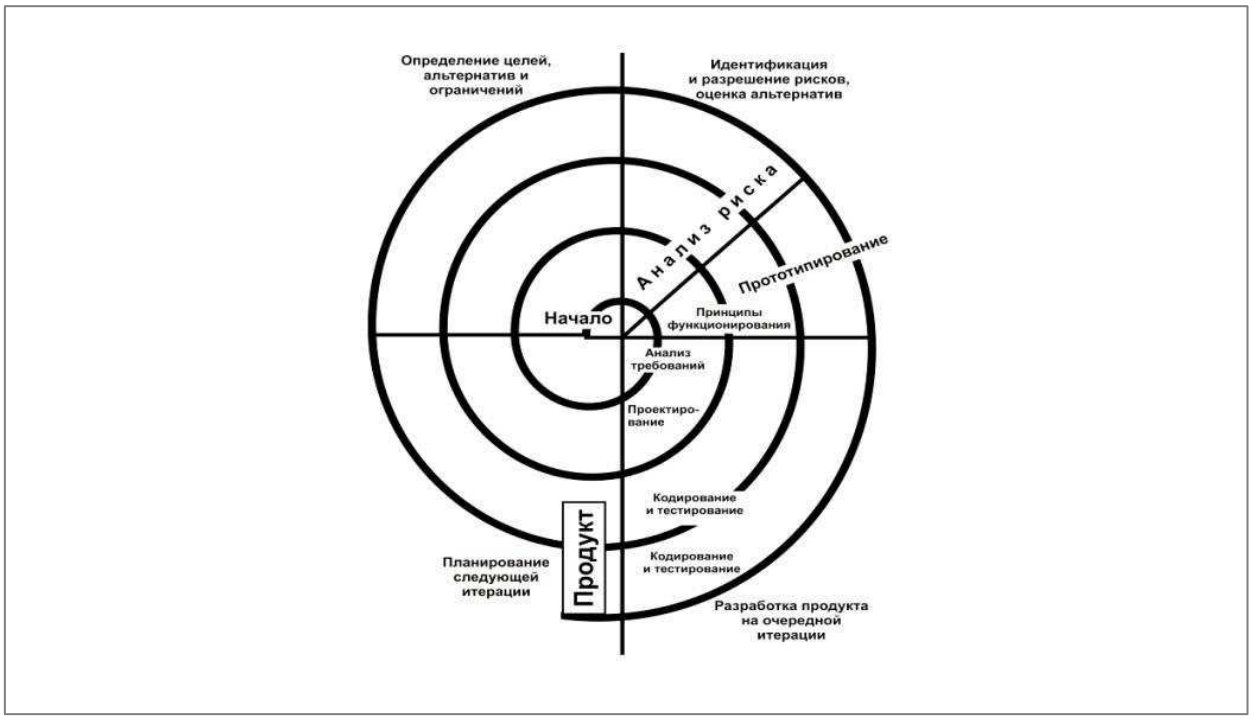

Рисунок 18 – Спиральная модель разработки

Главная особенность спиральной модели – концентрация на возможных рисках [47]. Для их оценки даже выделена соответствующая стадия.

Основные типы рисков, которые могут возникнуть в процессе разработки программного обеспечения:

– нереалистичный бюджет и сроки;

– дефицит специалистов;

– частые изменения требований;

– чрезмерная оптимизация;

– низкая производительность системы;

– несоответствие уровня квалификации специалистов разных отделов.

Плюсы спиральной модели:

– улучшенный анализ рисков;

– хорошая документация процесса разработки;

– гибкость – возможность внесения изменений и добавления новой функциональности даже на относительно поздних этапах;

– раннее создание рабочих прототипов.

Минусы спиральной модели:

– может быть достаточно дорогой в использовании;

– управление рисками требует привлечения высококлассных специалистов;

– успех процесса в большой степени зависит от стадии анализа рисков;

– не подходит для небольших проектов.

Когда использовать спиральную модель:

– когда важен анализ рисков и затрат;

– крупные долгосрочные проекты с отсутствием четких требований или вероятностью их динамического изменения;

– при разработке новой линейки продуктов.

Далее рассмотрим создание модулей приложения «Time manager system», применение средств организации командной работы над проектом, проектирование информационно-логической модули базы данных, применение клиент-серверной технологии и технико-экономическое обоснование эффективности программного проекта.

В итоге, для проекта была выбрана спиральная модель разработки, потому что она позволяет разрабатывать множество демо-версий находить в них изъяны и править их в новой версии, благодаря чему пользователь получит продукт, в котором вероятность ошибки очень мала, а если она и

есть, то цена её исправления может быть несоизмерима с возможностями проекта.

Теперь же необходимо рассказать о самом проекте, какие у него будут основные характеристики и какие подсистемы будут входить в него.

2.5 Концепция дизайна мобильного приложения

Для данного приложения было решено использовать рекомендованный компанией Google – материальный дизайн. На основе исходников и руководства официального сайта был спроектирован дизайн интерфейса приложения. Цветовая палитра, изображенная на рисунке 19, была выбрана с помощью инструмента COLOR TOOL.

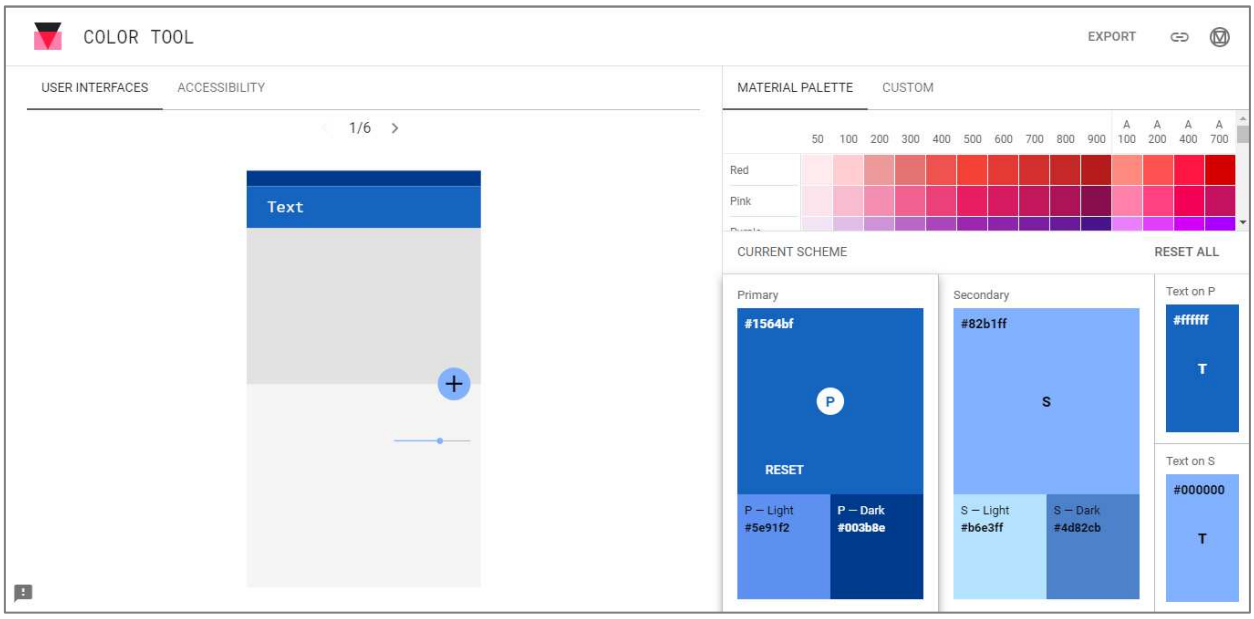

Рисунок 19 – Интерфейс инструмента COLOR TOOL

При создании основных экранов и элементов интерфейса были учтены рекомендации по создание дизайна интерфейса мобильных приложений. Полученный концепт дизайна представлен на рисунках 20-23.

Рисунок 20 – Макет экранов авторизации (слева) и регистрации (справа)

Для отображения задач было решено использовать один из самых популярных трендов дизайна интерфейсов – карточки. В контексте данного приложения, карточки представляют собой элементарную единицу, на которых отображена главная информация о состоянии задачи (рисунок 21).

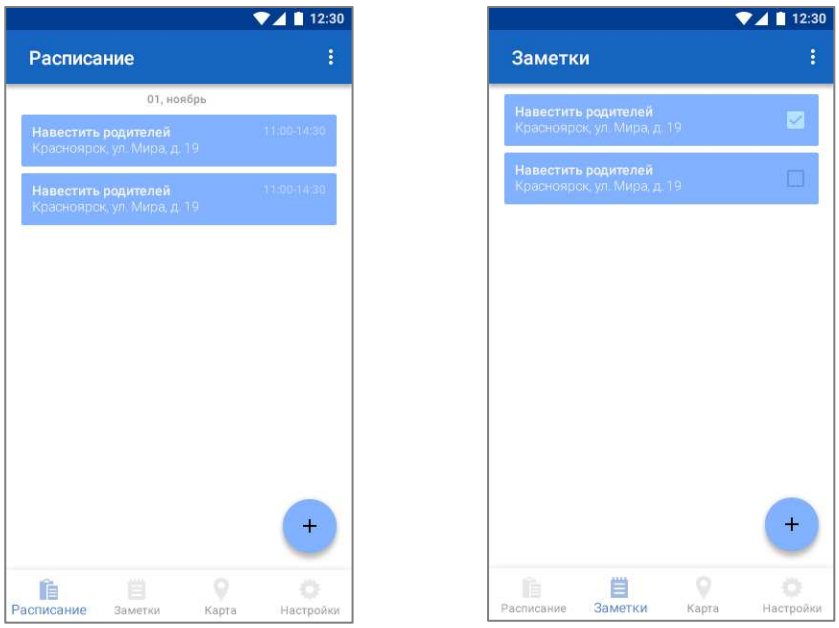

Рисунок 21 – Макет экранов отображения распорядка дня (слева) и заметок (справа)

Экраны редактирования заметок и пункта распорядка изображены ниже на рисунке 22.

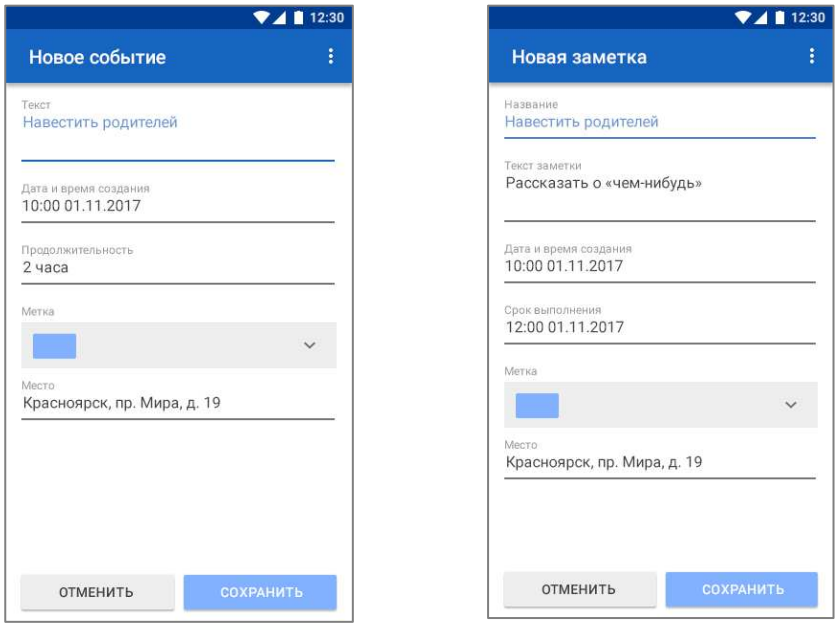

Рисунок 22 – Макеты экранов редактирования пункта распорядка дня (слева) и заметки (справа)

А также, по задумке, был спроектирован интерфейс отображения карты в приложении и экран настройки приложения (рисунок 23).

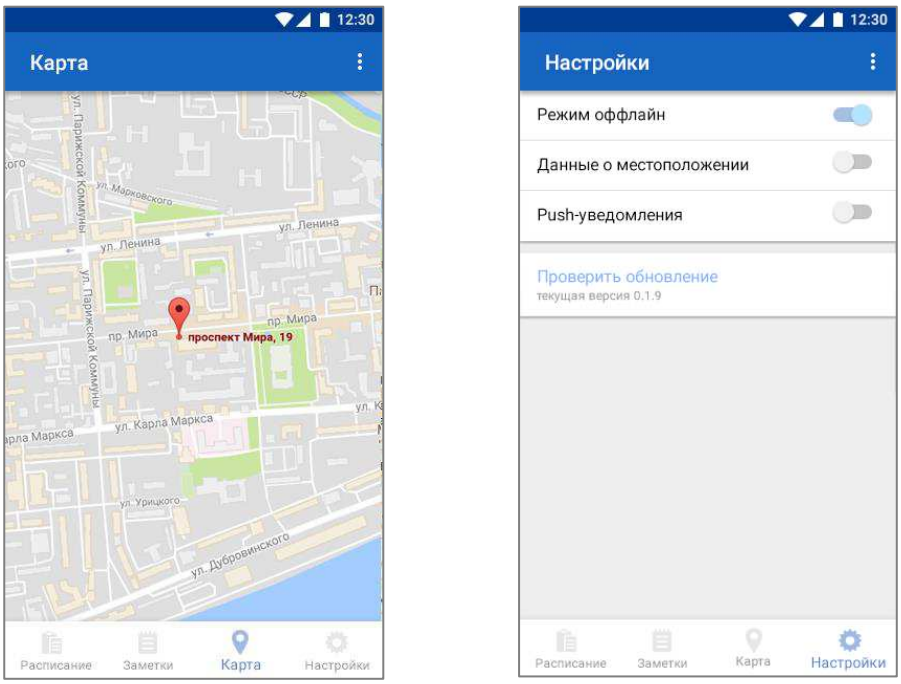

Рисунок 23 – Макеты экранов отображения карты (слева) и настроек (справа)

Дизайн приложения играет важную роль при создании приложения, макеты позволяют дать чёткое описание того, как видят разработчики будущее приложение.

Далее в следующей главе описано о процессе разработке всех модулей приложении и процессе тестирования.

3 Создание и тестирование модулей

Модуль – функционально законченный фрагмент программы, оформленный в виде отдельного файла с исходным кодом или поименованной непрерывной ее части (например, Active Oberon), предназначенный для использования в других программах. Модули позволяют разбивать сложные задачи на более мелкие в соответствии с принципом модульности [48]. Обычно проектируются таким образом, чтобы предоставлять программистам удобную для многократного использования функциональность (интерфейс) в виде набора функций, классов, констант. Модули могут объединяться в пакеты и, далее, в библиотеки. Удобство использования модульной архитектуры заключается в возможности обновления (замены) модуля, без необходимости изменения остальной системы. В большинстве случаев различные модули могут запускаться как на одном сервере, так и на разных, для распределения нагрузки и создания распределенной архитектуры.

Модульное программирование является особым способом разработки программы, которая строится при этом из нескольких относительно независимых друг от друга частей – модулей. Понятие модуля является одним из центральных при разработке программного обеспечения.

При разработке данного приложения используется метод, который называется «программирование сверху-вниз» или «пошаговая детализация». Принцип этого метода состоит в том, чтобы разбить исходную задачу на небольшие относительно независимые друг от друга подзадачи. В том случае, когда полученные подзадачи достаточно просты, то для каждой из них разрабатывается соответствующий алгоритм, иначе каждая такая сложная подзадача снова разбивается на более простые и т.д. Далее приступают к реализации каждой из полученных таким образом относительно простых подзадач на некотором языке программирования, и каждая такая реализация подзадачи и называется чаще всего (программным)

модулем. Таким образом, использование модулей является естественным способом разработки и построения сложных программ. Программный модуль допускает дальнейшую пошаговую детализацию, однако полученные подзадачи реализуются уже в виде процедур и функций. Таким образом, процедуры и функции являются инструментом пошаговой детализации при разработке программ на нижнем, более детальном уровне, а модули – на верхнем.

Модульное тестирование – процесс в программировании, позволяющий проверить на корректность единицы исходного кода, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки [49].

Для того чтобы команда могла хорошо функционировать необходимо организовать связь всей команды с помощью различных приложений.

3.1 Применение средств организации командной работы

Для обеспечения продуктивной работы членов команды разработчиков были применены следующие программные средства:

− система контроля версий Git;

− хостинг репозиториев Bitbucket;

− графический Git клиент SourceTree для доступа и взаимодействия с репозиторием проекта;

− web-приложение Trello для обеспечения управления командой разработчиков, размещения заданий и контроля их выполнения;

− облачное хранилище OneDrive для обеспечения общего доступа разработчиков к составленным техническим документам по проекту;

− облачный офисный макет OfficeOnline для совместного просмотра и редактирования документов и графиков без их предварительной загрузки на компьютер разработчика.

Одна из приоритетных проблем, с которой приходится сталкиваться команде разработчиков во время работы над большой программной системой – возможность параллельной, независимой работы каждого члена команды со своевременным доступом к актуальной версии исходного кода разрабатываемой системы [50].

Решить данную проблему помогает система контроля версий.

В текущем проекте была использована система контроля версий Git в комбинации с хостингом репозиториев Bitbucket и графическим клиентом SourceTree.

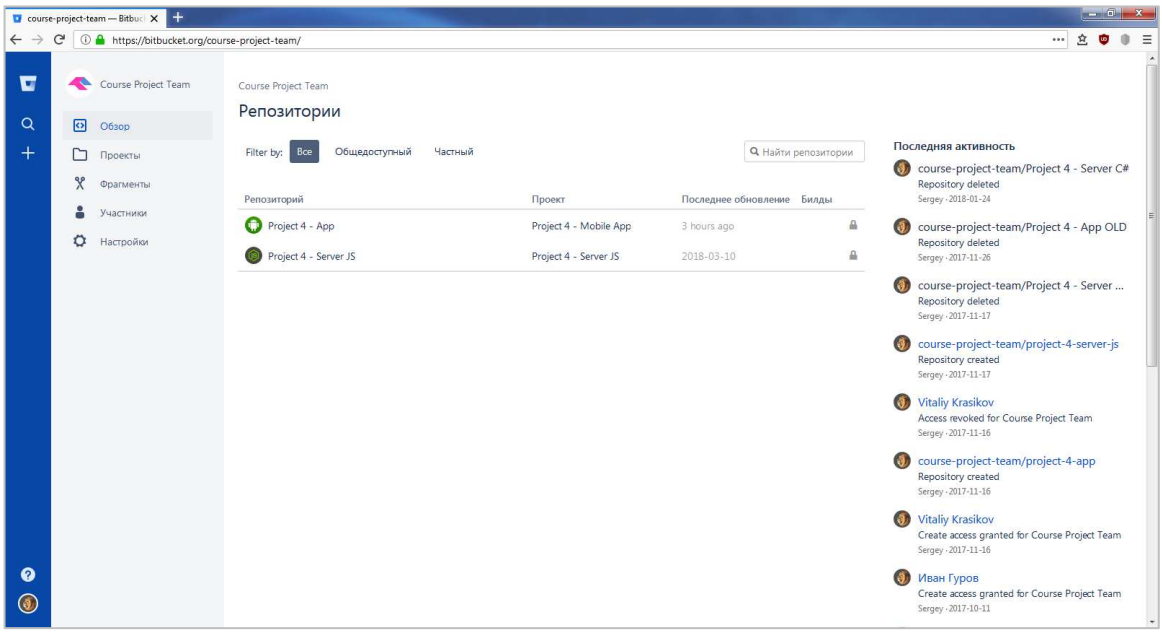

Рисунок 24 – Репозитории проекта

Использование Bitbucket:

− создана команда, в которую добавлены все разработчики;

− создано два проекта: для серверного и клиентского приложений;

− к соответствующим проектам прикреплен репозиторий;

− на начальном этапе в каждом репозитории созданы две ветки: master (для сборки и запуска завершенных модулей) и dev (для разработки и тестирования) [51].

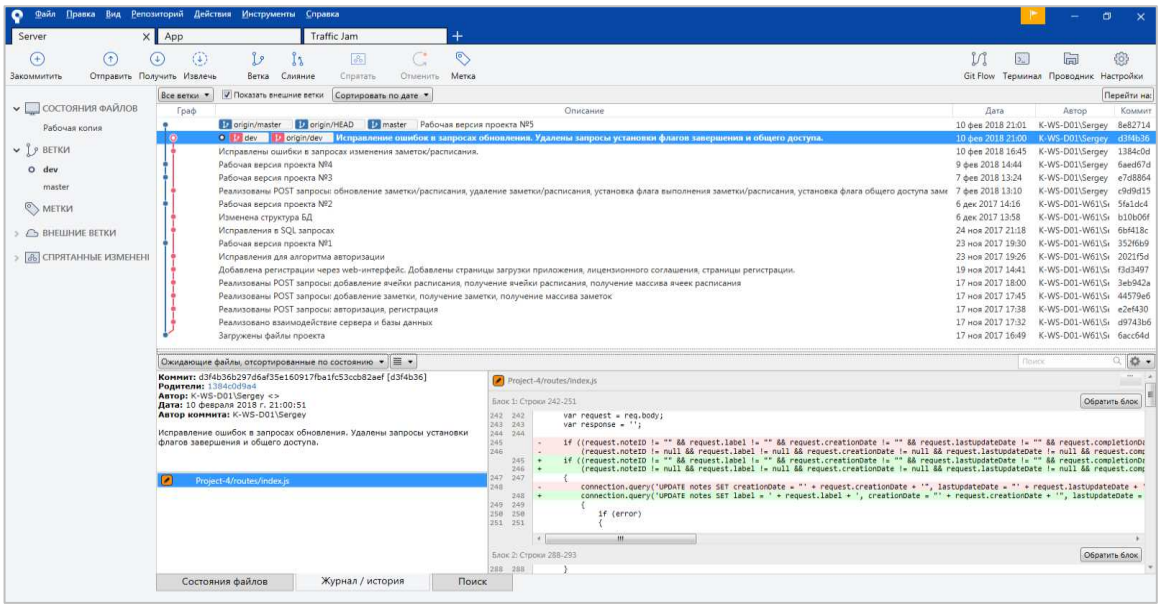

Рисунок 25 – Журнал изменений сервера в SourceTree

Графический клиент позволил отказаться от использование командной строки для взаимодействия с репозиторием, что существенно сэкономило время и упростило работу. Клиент позволил без проблем переключаться между двумя репозиториями, выбирать файлы для фиксации изменений, создавать комментарии, переключаться между ветками, принимать и отправлять изменения [52].

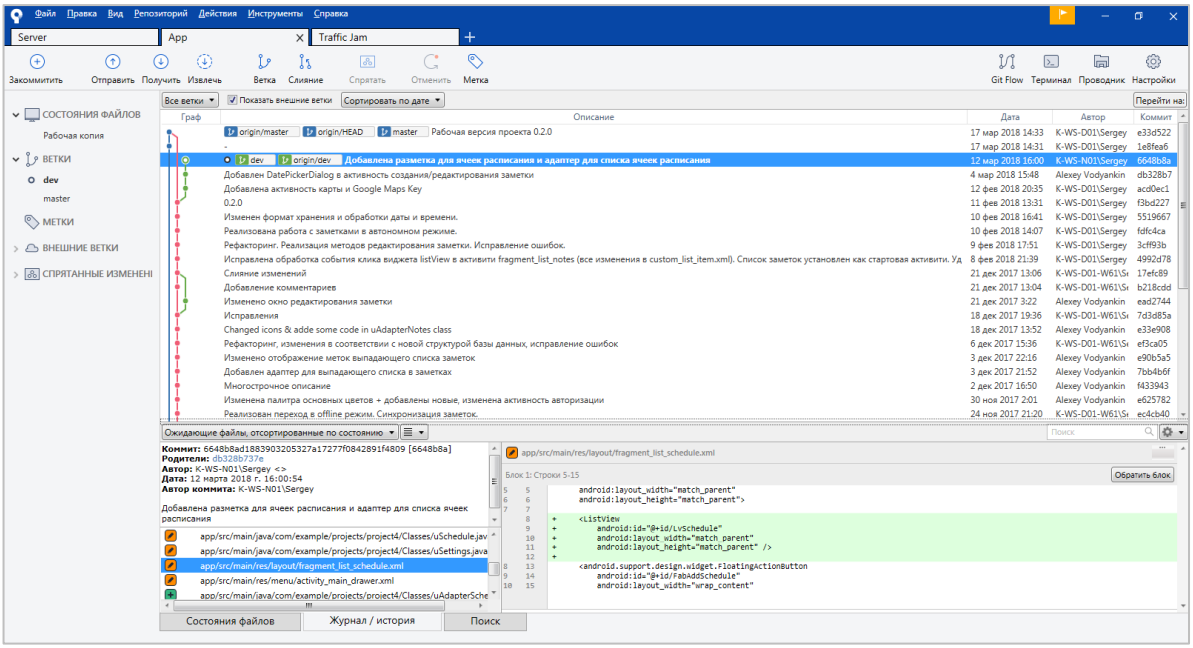

Рисунок 26 – Журнал изменений приложения в SourceTree

Следующим этапом было решение вопроса удобного и наглядного распределения обязанностей и работ между членами команды разработчиков. В результате было выбрано веб-приложения для управления проектами небольших групп Trello.

Данное приложение позволило создавать доску проекта, на которой были размещены списки и карточки.

Списки:

– ресурсы: ссылки на онлайн ресурсы по проекту, доступ к распределению ролей и модулей;

– план: общий план работ на весь период разработки;

– план работ по месяцам: ежемесячно обновляемый план работ на текущий период времени;

– выполнено: крупные завершенные этапы, результат которых потребуются в будущем;

– на доработку: выполненные этапу работ, но требующие определенной доработки.

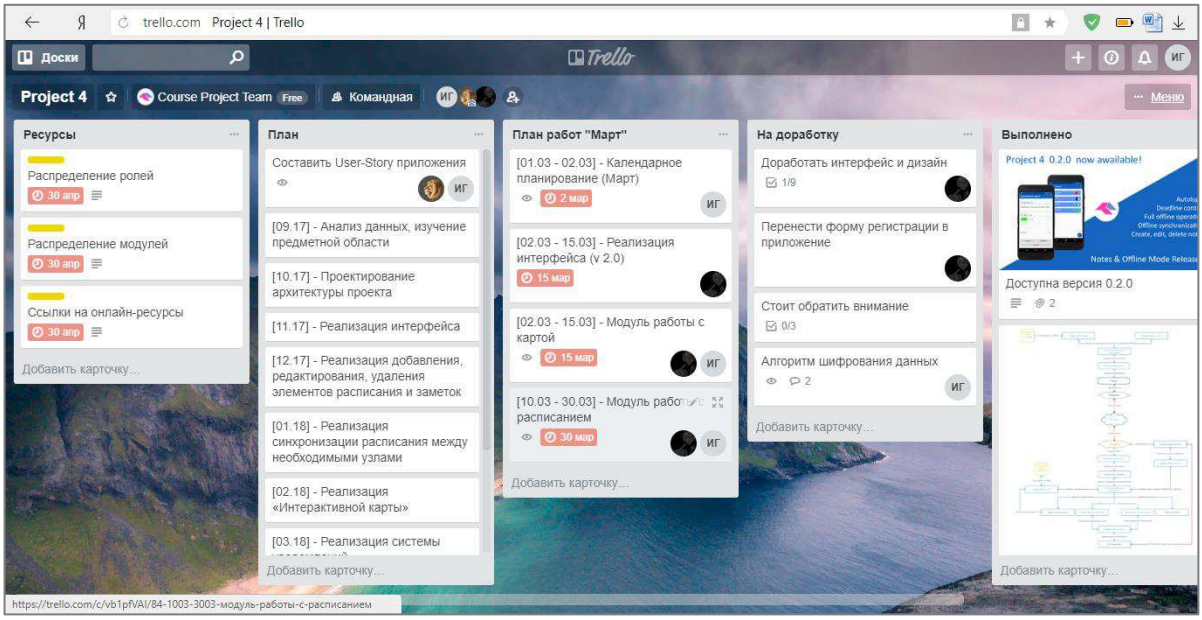

Рисунок 27 – Командная доска в Trello

Оставался нерешенным вопрос хранения календарного графика, а также создаваемой в ходе работы технической документации по проекту и доступ к ним. Для решения этого вопроса было принято решение использовать облачное хранилище OneDrive.

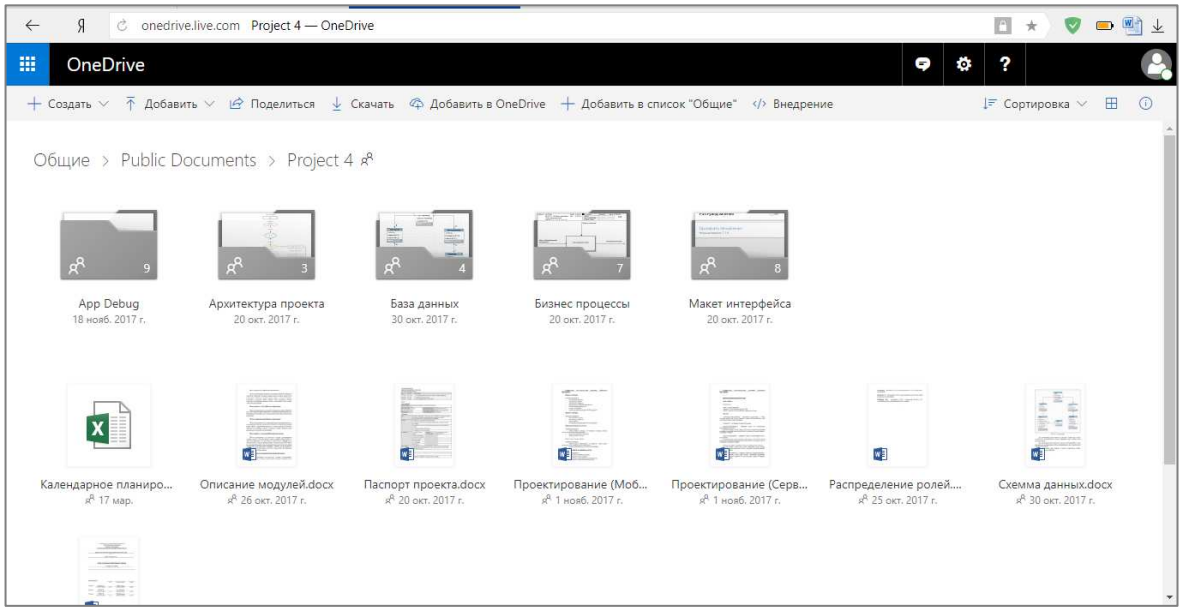

Рисунок 28 – Общий доступ к техническим документам проекта в OneDrive

Выбор был обусловлен такими факторами, как: наличие уже созданных учетных записей у всех разработчиков для доступа к этому хранилищу; возможность онлайн редактирования документов, без необходимости их загрузки; возможность совместного редактирования и возможность интеграции данного ресурса с другими [53].

Вопрос об общении команды разработчиков друг с другом упирается чаще всего в то, что у кого-то не создан аккаунт на популярных сайтах и системах.

3.2 Создание модулей

Разрабатываемое мобильное является клиент серверным, серверная и клиентская часть – это два независимых друг от друга приложения, которые

обмениваются между собой данными. Поэтому разработку можно поручить отдельным людям, которые будут контактировать друг с другом только тогда, когда будут настраивать обмен данными между сервером и клиентом.

Клиент-серверное приложение, в котором клиент взаимодействует с сервером при помощи приложения, а за сервер отвечает – веб-сервер. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются межплатформенными службами.

Программные компоненты взаимодействуют друг с другом посредством API. При этом обычно компоненты образуют иерархию – высокоуровневые компоненты используют API низкоуровневых компонентов, а те, в свою очередь, используют API ещё более низкоуровневых компонентов.

По такому принципу построены протоколы передачи данных по Internet. Стандартный протокол Internet (сетевая модель OSI) содержит 7 уровней (от физического уровня передачи пакетов бит до уровня протоколов приложений, подобных протоколам HTTP и IMAP). Каждый уровень пользуется функциональностью предыдущего уровня передачи данных и, в свою очередь, предоставляет нужную функциональность следующему уровню.

Для создания каждой части программист выбрал подходящие для него технологии.

Клиентская часть приложения – это то, с чем взаимодействует пользователь. Описание взаимодействия пользователя с мобильным приложением, а также основные методы, использованные при разработке приложения, изложены в следующей главе.

3.2.1 Клиентская часть приложения

Клиентская часть приложения – это мобильная часть, с которой взаимодействует пользователь. Главная цель клиентского приложения – это стабильный отклик на все действия пользователя.

Чтобы войти в систему пользователь должен быть зарегистрироваться, после чего происходит авторизация и данные о успешной авторизации хранятся неделю после последнего захода, то есть пользователю не надо будет каждый раз заново авторизоваться. Это возможно благодаря функции _loadInitialState (рисунок 29), в которой данные, хранящиеся в AsyncStorage, проверяются на наличие в кэше устройства.

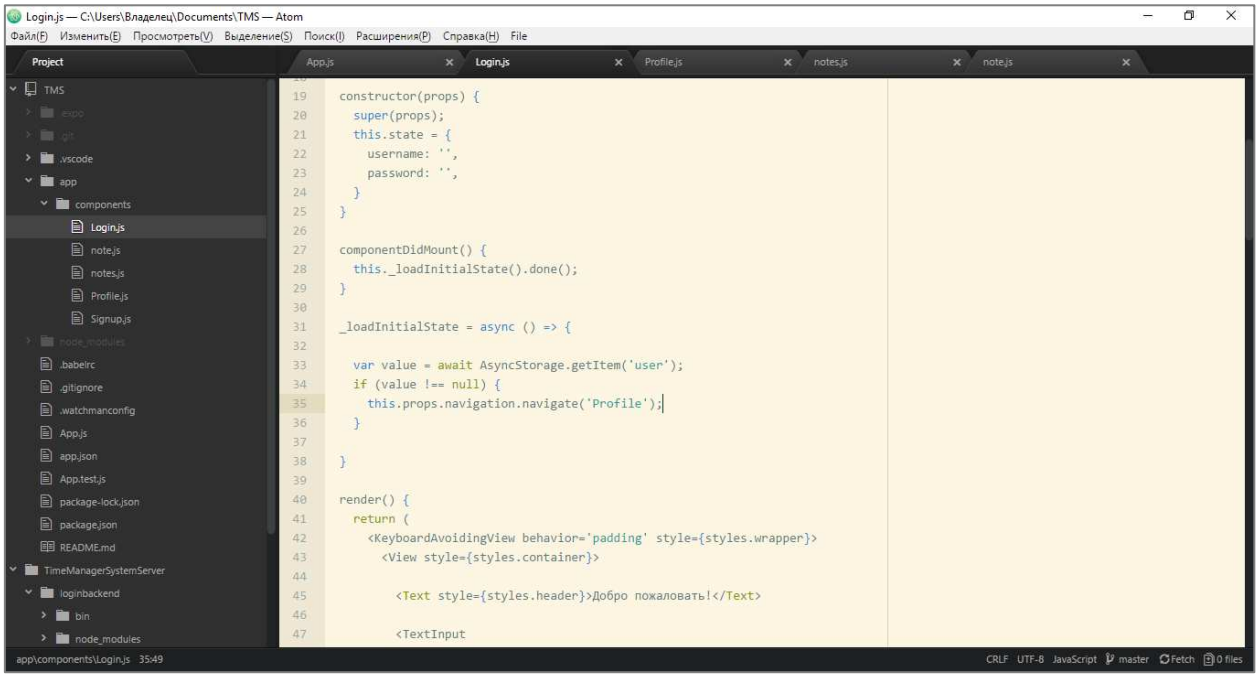

Рисунок 29 – Функция componentDidMount

После успешной авторизацией пользователю открываться список заметок пользователя. Где пользователь может создавать новые заметки и удалять старые заметки. Это возможно благодаря функциям addNote и deleteNote (рисунок 30).

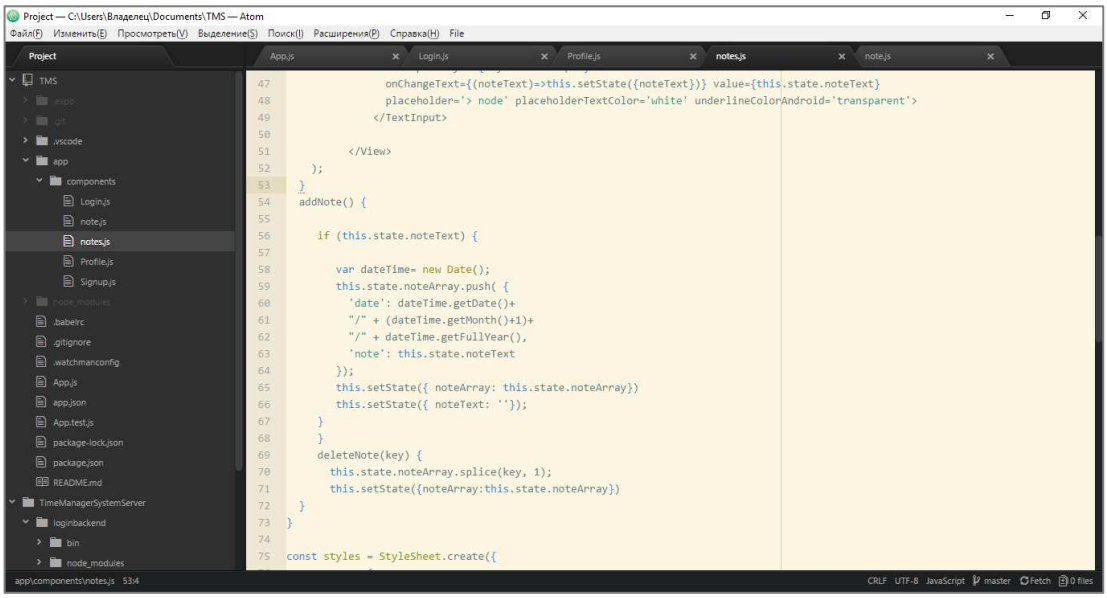

Рисунок 30 – Функции addNote и deleteNote

После каких-либо изменений в системе данные отправляться на сервер, где они хранятся. Это реализовано в классах заметки и расписание с помощью запросов к базе данных.

Основным преимуществом мобильного приложения – это работа с картой, которая напоминает пользователю о его задачах, что позволяет не забыть сделать запланированные дела. В классе Map (рисунок 30) показывается как система определяет геолокацию пользователя и получает данные о местах, которые расположены поблизости.

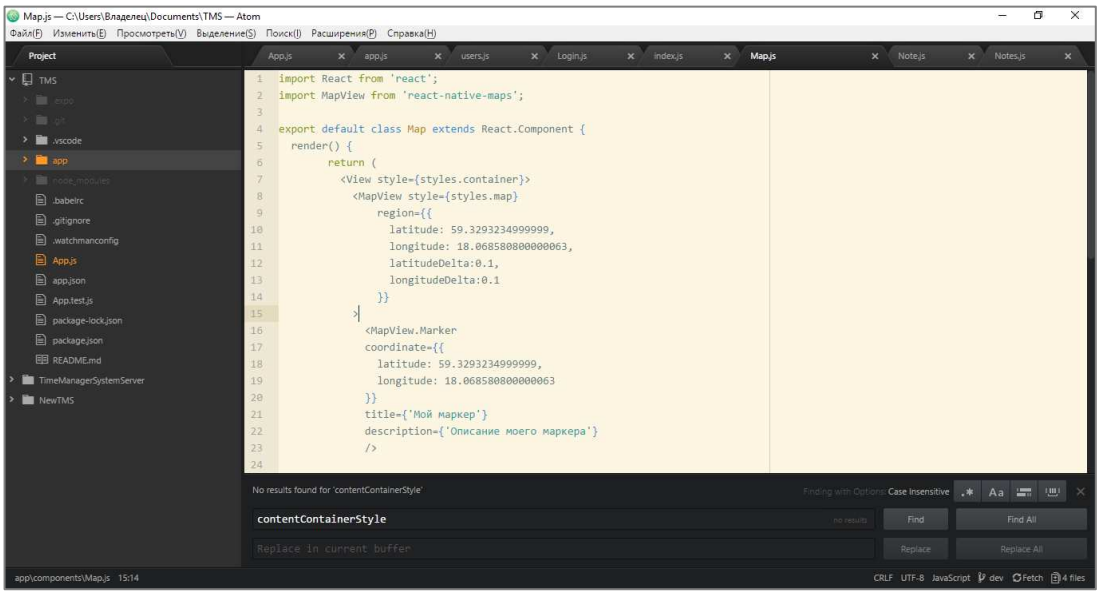

Рисунок 31 – Класс Map

Примеры стабильной работы приложения можно будет увидеть в главе тестирование приложения. В следующей главе рассказано об аспектах безопасности приложения.

3.2.2 Серверная часть приложения

Модуль работы с базой данных для серверного приложения предназначен для работы с базой данных MySQL на удаленном сервере. Выполняет добавление, изменение и получение данных. Данный модуль вызывается контроллером API серверного приложения.

Модуль API предназначен для работы с сетью. Данный модуль контролирует полученные POST/GET/DELETE/UPDATE запросы мобильного приложения, в ответе на них он посылает тело ошибки или предоставляет запрошенные данные сервера.

Модуль авторизации отвечает за генерацию токенов, которые предоставляют доступ к API приложения зарегистрированным пользователям. Также, данный модуль осуществляет верификацию токенов.

Методы доступа реализуют возможность авторизации и регистрации пользователей (рисунок 32).

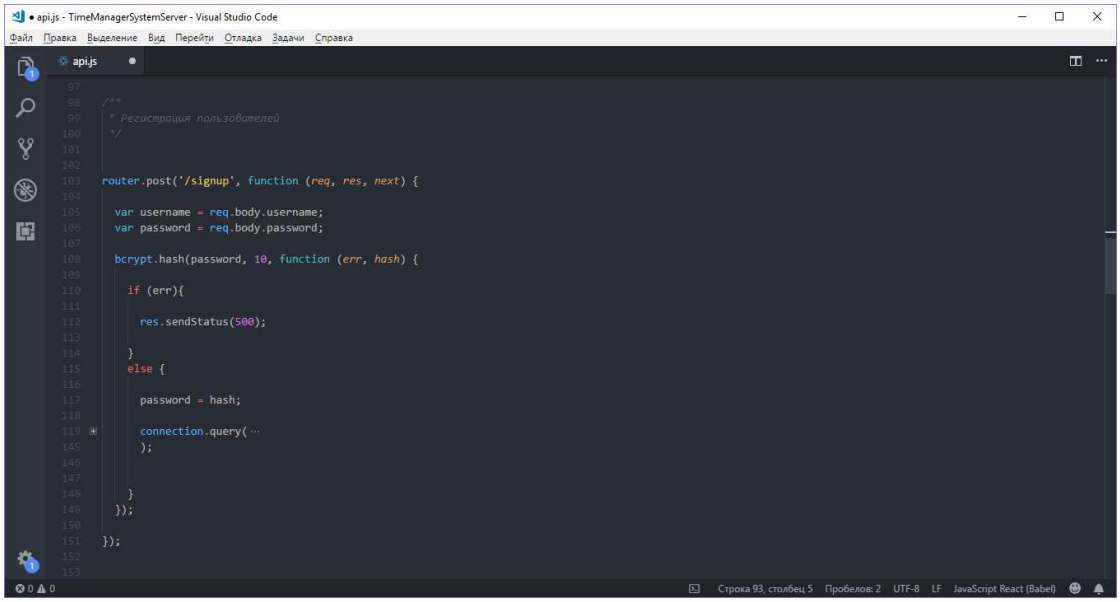

Рисунок 32 – Реализация метода регистрации пользователей

Методы работы с заметками и методы работы с ячейками расписания реализуют возможность добавления, изменения, удаления, получения одного объекта, получения списка объектов (рисунок 33).

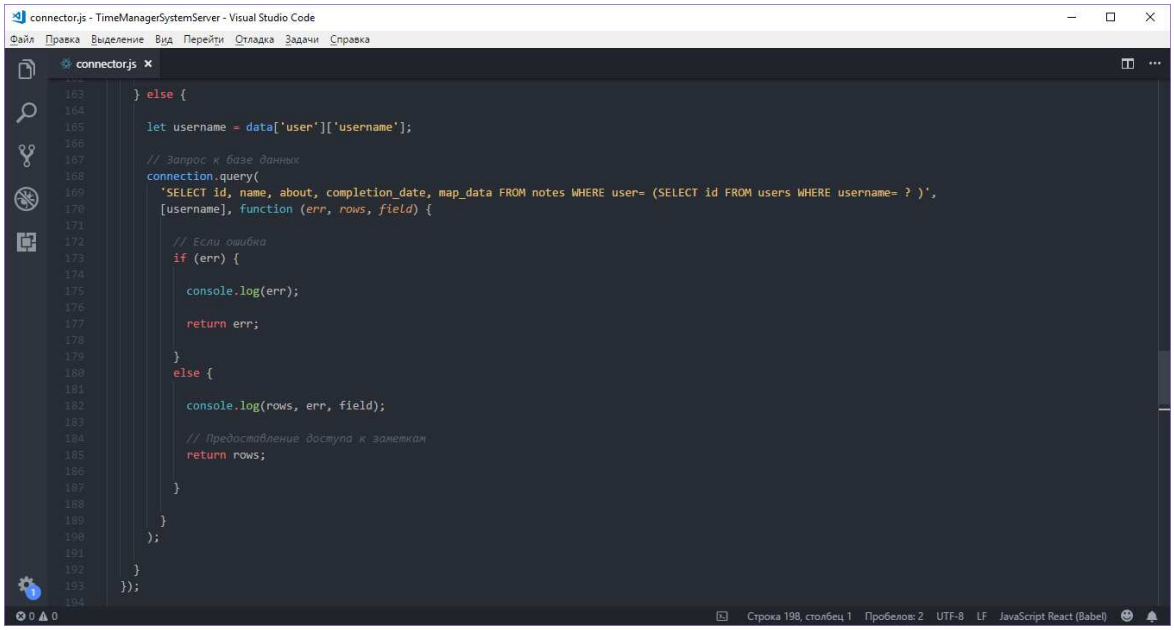

Рисунок 33 – Реализация модуля работы с базой данных

Методы валидации и генерации токенов изображены на рисунке 34.

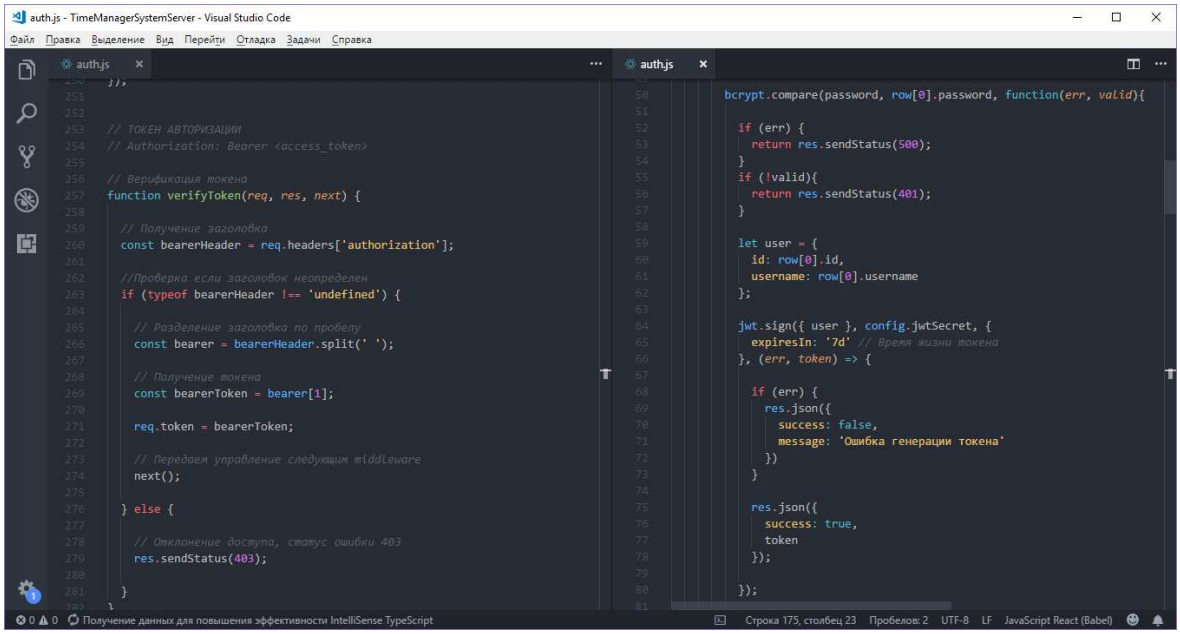

Рисунок 34 – Реализация модуля авторизации

Серверная часть приложения нужна, чтобы хранить все данные о пользователях и не просто хранить, а обеспечить безопасность личных данных.

Безопасность личной информации пользователей один из ключевых моментов, которым стоит уделять должное внимание. Поэтому многие компании используют шифрование и различные методы верификации пользователей, чтобы защитить данные.

Одним из самых распространенных вариантов аутентификации пользователей является открытый стандарт для передачи пакетов данных в интернет среде JSON Web Token. JSON Web Token (JWT) – это JSON объект, который определен в открытом стандарте RFC 7519. Он считается одним из безопасных способов передачи информации между двумя участниками. Для его создания необходимо определить заголовок (header) с общей информацией по токену, полезные данные (payload), такие как id пользователя, его роль и т.д. и подписи (signature). Иными словами, JWT – это лишь строка в следующем формате «header.payload.signature» [54].

Приложение использует JWT для проверки аутентификации пользователя следующим образом:

1) Пользователь посредством клиентской части приложения отправляет запрос на сервер для аутентификации с помощью пары логин/пароль.

2) Затем сервер аутентификации создает JWT и отправляет его обратно.

3) При последующих обращениях клиента к API приложения полученный ранее JWT добавляется к запросу.

4) После получении запроса к API серверная часть приложения может проверить по переданному JWT с запросом является ли пользователь тем, за кого себя выдает. В этой схеме сервер приложения сконфигурирован так, что сможет проверить, является ли входящий JWT именно тем, что был создан им.

Другим аспектом безопасности является хранение данных на сервере. Чтобы избежать утечки информации о пользователе используется

шифрование. Таким образом, в базе данных информация хранится в зашифрованном виде.

При шифровании данных была использована функция bcrypt – это адаптивная криптографическая хеш-функция используемая для защищенного хранения паролей. Разработчики: Нильс Провос и Дэвид Мэйзиэс. Функция основана на шифре Blowfish, впервые представлена на USENIX в 1999 году. Для защиты от атак с помощью радужных таблиц bcrypt использует соль (salt), кроме того, функция является адаптивной, время ее работы легко настраивается и ее можно замедлить, чтобы усложнить атаку перебором [55].

Шифр Blowfish отличается от многих алгоритмов вычислительно сложной фазой подготовки ключей шифрования. Провос и Мэйзиэс воспользовались этой особенностью, но изменили алгоритм подготовки ключей, получив шифр «Eksblowfish».

Количество раундов в подготовке ключей должно быть степенью двойки, конкретная степень может задаваться при использовании. Для шифрования данных в приложении была выбрана соль со значением 10.

При реализации данных аспектов использованы библиотеки Node.js: jsonwebtoken и bcrypt, которые распространяются по лицензии открытого программного обеспечения, что разрешает лицам, получившим копию данного программного обеспечения и сопутствующей документации, использовать их безвозмездно и без ограничений.

После создания всех модулей приложения необходимо провести полное тестирования созданного проекта, чтобы убедиться в безотказной работе приложения и при необходимости исправить найденные ошибки до релиза.

3.3 Тестирование приложения

Мобильные приложения – это приложения, тестирование которых значительно отличается от тестирования десктопных. Если приложение

вашего конкурента работает лучше вашего то, увы, пользователь выберет его, поскольку заинтересован в качестве. Мы стоим на его страже, поэтому давайте обсудим особенности тестирования мобильных приложений, чтобы минимизировать подобные ситуации [56].

При разработке мобильных приложений, а следовательно, и при тестировании, следует учитывать ряд определенных моментов:

− мобильные устройства – это системы, которые чаще всего имеют не шибко мощную начинку. Они по определению не могут работать как персональный компьютер, поскольку слабее в разы;

− прогресс в сфере информационных приложений движется очень быстро, поэтому операционные системы мобилок быстро устаревают. Кроме того, есть предел на обновление их ОС. К примеру, на iPhone 4 – это версия 7.1.2. Разработка клиентской части проверялась на версии Android 6.0.1;

− многообразие экранов, их расширений и цветов. В отличие от монитора компьютера, экран мобильных устройств может менять ориентацию, что также необходимо учесть при разработке и тестировании мобильных приложений;

− существует определенный список обязательных параметров мобильных приложений, которые создаются производителями устройств. Им следовать нужно обязательно;

− мобильное устройство чаще всего находится в движении, поэтому следует ожидать, что могут возникнуть какие-то случайные действия на устройстве (если оно не заблокировано, если щекой нажимаешь кнопки или кто-то тебя пинает). При разработке приложения для телефона нужно также учесть его пребывание в разных погодных условиях, при разном свете, поэтому нужно использовать контрастные цвета;

− необходимо помнить, что основной задачей, к примеру телефона, по-прежнему являются звонки, и приложение ну никак не должно мешать этой прямой и главной функции устройства;

– при тестировании мобильных приложений все-таки следует пренебречь эмуляторами, если у вас есть такая возможность. Дело в том, что в них не всегда функционал соответствует всем возможностям реального девайса;

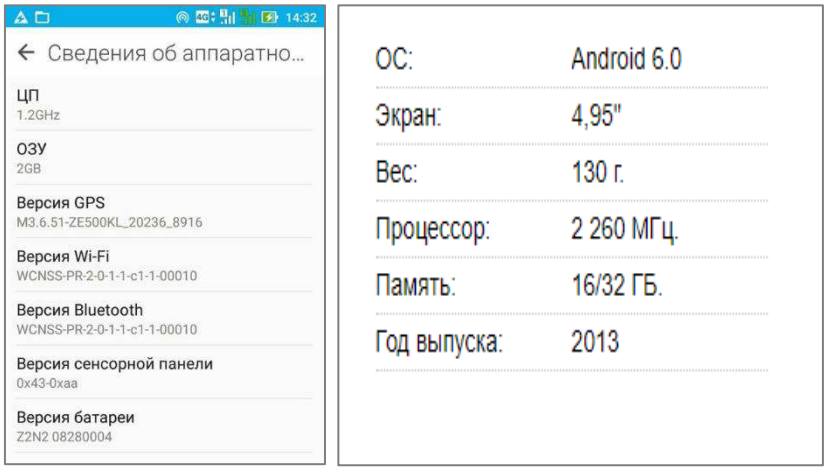

Рисунок 35 – Характеристики мобильных устройств (слева – Asus Zenfone, справа – Google Nexus 5) для тестирования приложения

– на мобильных устройствах может быть представлено большое разнообразие специфических операционных систем и конфигураций комплектующих;

– девайс постоянно пребывает в состоянии поиска сети. При тестировании следует проверить работу приложения на разных скоростях передачи данных.

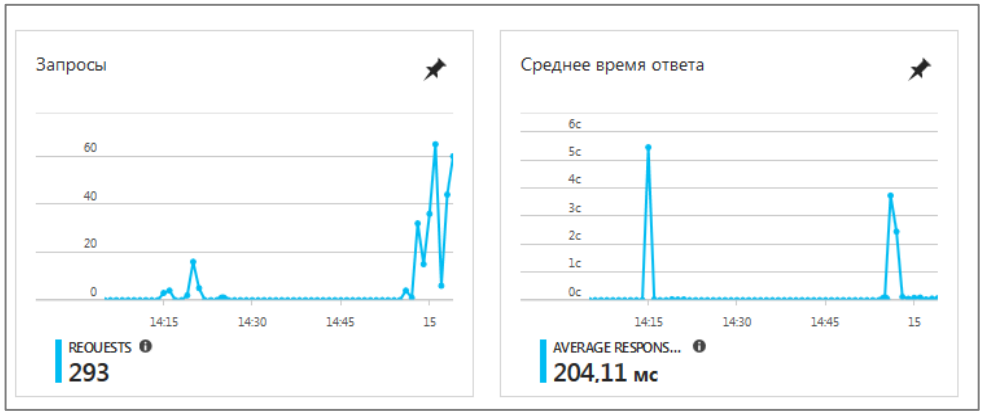

Рисунок 36 – Статистика запросов

Мобильное приложение должно быть интуитивно понятным, удобным, работать бесперебойно, безопасно, круглосуточно и достаточно быстро реагировать на действия пользователя. Итак, рассмотрим основные моменты и особенности тестирования мобильных приложений, на которые стоит обратить внимание при функциональном и GUI тестировании мобильных приложений.

Размер экрана и touch-интерфейс:

– удобный размер кнопок, чтобы не надо было искать ее на экране и попадать с третьего раза по ней;

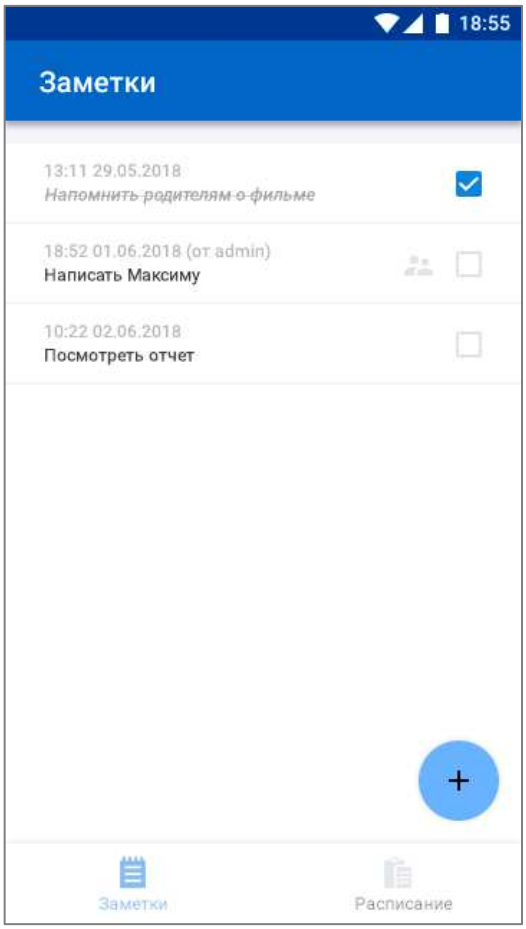

Рисунок 37 – Экран приложения «Заметки»

– скорость отклика элементов (высокая; нажатая клавиша должна визуально отличаться).

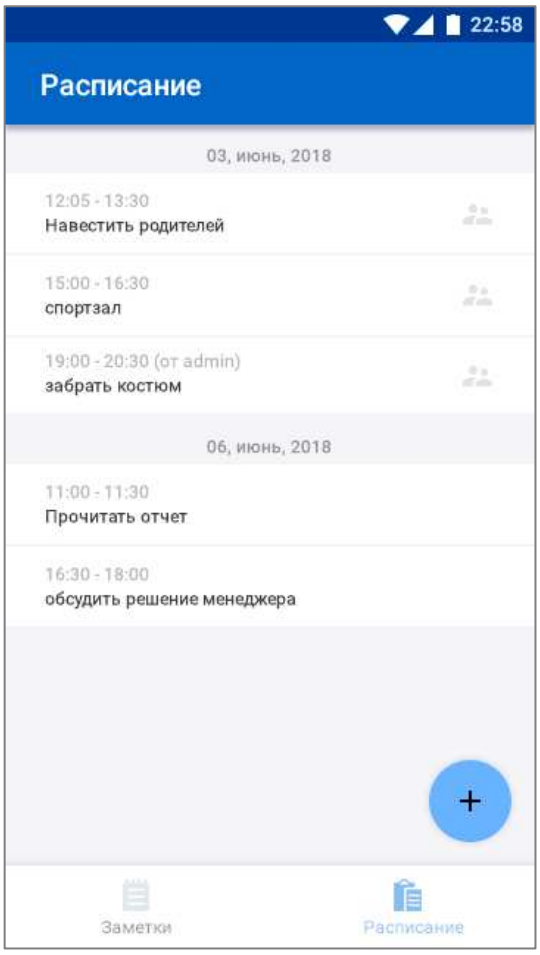

Рисунок 38– Экран приложения «Расписание»

Утечки памяти:

− можно проверить с помощью программы Instruments (стандартное приложение MacOS). Может быть не более 30мб на iPhone, примерно 70мб для всех девайсов до iPad 2;

− уделить внимание окнам с большим количеством информации, при длительном пребывании пользователя в приложении.

Проверка работы приложений на ретина экранах и различных версия OS:

− корректное отображение различных элементов на экранах ретина/не ретина;

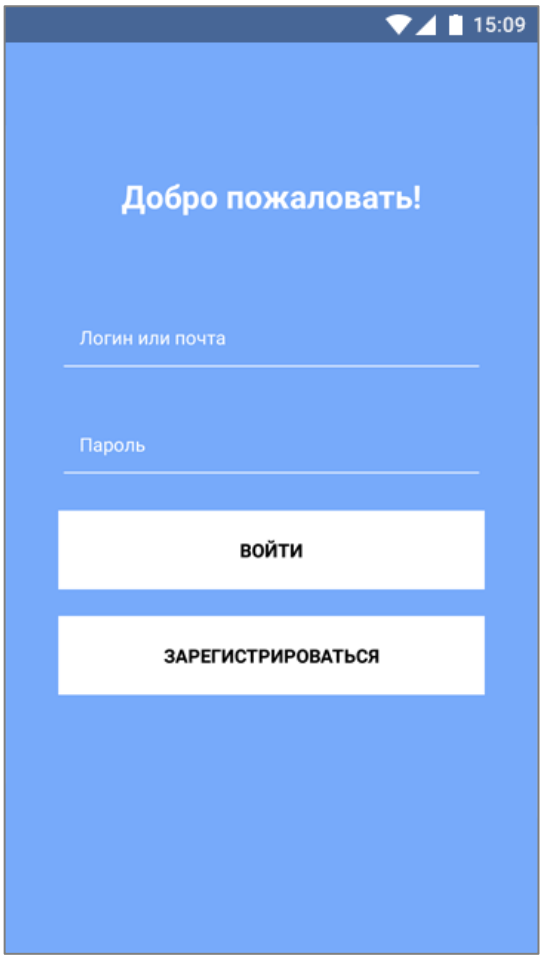

Рисунок 39 – Экран авторизации

– установка приложения на корректную версию OS;

– проверить установку на все возможные девайсы;

– различные функции на девайсах: отсутствие/наличие камеры (iPad) (автофокуса), отсутствие/наличие GPS.

Приложение было установлено и запущено в эмуляторах с различными версиями операционной системы. Единственно, что бросалось в глаза это цвет некоторых элементов немного менялся в зависимости от версии ОС.

Проверка типа покупок (восстанавливаемые, не восстанавливаемые)

– проверка восстановления покупки независимо от девайса, а с привязкой к учетной записи.

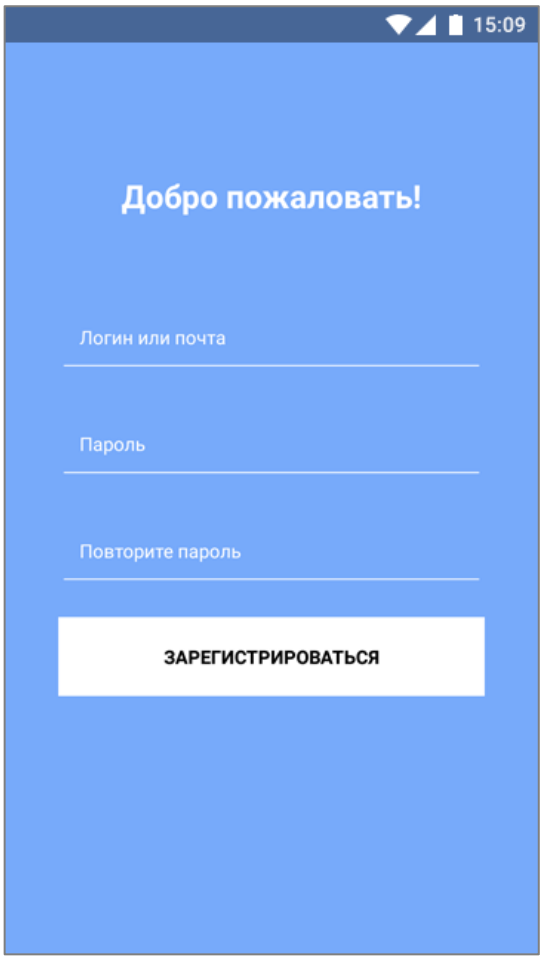

Рисунок 40 – Экран регистрации

Проверка работы обратной связи:

– сообщения при загрузке контента/прогресс;

– сообщения при ошибке доступа к сети;

– наличие сообщений при попытке удалить важную информацию;

– наличие экрана/сообщения при окончании процесса.

Проверка работы обновлений:

– проверка различных путей установки обновлений (wifi, bluetooth, usb);

– проверка работы установленных изменений, мест, куда они вносились;

– убедиться в поддержке обновлений более старыми операционками, чтобы элементы, которые используются в новой системе, работали хорошо, не вызывали ошибок на более старых версиях.

Чтобы приложения установилось и стабильно работало с новой версией необходимо удалить старое приложение и скачать новый apk, либо обновить приложение через подключение к компьютеру.

Проверка реакции приложения на внешние прерывания:

– входящие/исходящие СМС, ММС, звонки;

– разряд/изъятие батареи;

– отключение сети/Wi-Fi;

– подключение кабеля, карты, зарядки.

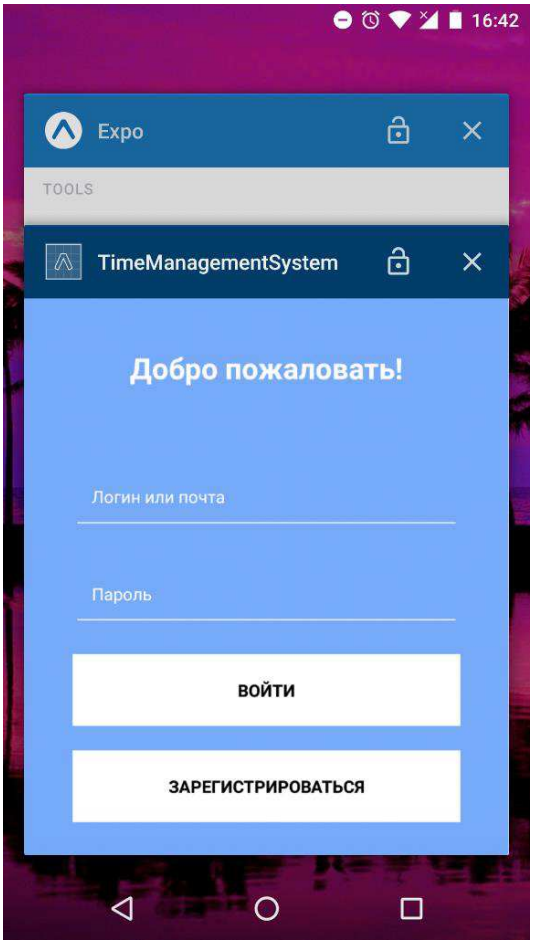

Рисунок 41 – Свернутое приложение

Реклама в мобильном приложении:

– реклама не должна перекрывать кнопки управления приложением;

– реклама должна иметь доступную кнопку закрытия, потому что чаще всего пользователь ее не ищет, а просто удаляет приложение с концами.

Проверка локализации:

− на другом языке на экране должно хватить места для текста. Приложение рассчитано для русскоговорящей аудитории, поддержка других языков может быть добавлена в следующих версиях проекта;

− даты должны соответствовать формату установленного региона;

− временные настройки должны быть соблюдены, благодаря UTC телефон сам определяет часовой пояс, в котором находиться устройство, а приложение берет эти данные и использует их.

Проверка энергопотребления: необходимо проверять насколько сильно ваше мощное приложение опустошает батарею устройства. Скорее всего пользователь удалит его, если из-за него мобильное устройство придется заряжать слишком часто.

Из-за того, что приложение постоянно определяет положения устройство на карте, это потребляет большое количество энергии и это может стать проблемой для владельцев телефон с маленькой ёмкостью батарей, поэтому необходимо оптимизировать этот аспект приложения.

Таковы основные моменты и особенности тестирования мобильных приложений, на которые стоит обратить внимание при тестировании мобильных приложений. Каждое устройство индивидуально ввиду установленных пользователем параметров и конфигураций. Но тем не менее следует придерживаться проверки вышеуказанных моментов на любом устройстве.

Создание и тестирование мобильного приложения – это долгий и трудоемкий процесс. Плюс у мобильных приложений есть свою нюансы, которые необходимо учитывать команде разработчиков при создании приложения.

После создания модулей, объединение их в одну программу и тестирование получившегося мобильного приложения необходимо посчитать затраченные деньги для создания и примерное время окупаемости проекта.

3.4 Экономические выкладки по проекту

Эффективность проекта оценивается с целью определения потенциальной привлекательности проекта для поиска возможных участников источников финансирования.

Разработка проекта заняла 9 месяцев. С 1 сентября 2017 года по 31 мая 2018 года. Над проектом трудится 2 человека.

МРОТ в городе Красноярске с 01.01.2018 равен 11968,96 рублей без вычета НДФЛ [57].

Один киловатт энергии в городе Красноярске с 01.01.2018 стоит 2.37 рублей [58].

Таблица 4 – Расходы проекта

№	Показатели	Всего (руб.)	Один месяц
1)	Затраты на заработную плату персонала, осуществляющего обслуживание ПО, с учетом страховых взносов	323161,92	35906,88
2)	Доступ в Интернет	4500	500
3)	Затраты на электроэнергию	2429,25	485,85
4)	Страховые взносы	46800	15 600
5)	Основное оборудование	36000	4000
6)	Затраты, связанные с ремонтом оборудования	5000	-
7)	Основное оборудование	180000	-
8)	Материальные затраты	1740	-
9)	Итог	599631,17	56492,73

Таблица 5 – Эксплуатационные расходы

Наименование показателя	Time Management System Сумма, руб.	Сторонние Решения Сумма, руб.	Годовая экономия Сумма, руб.
Затраты на заработную плату персонала, осуществляющего обслуживание ПО, с учетом страховых взносов	369961,92	542400	172438,08
Общепроизводственные расходы (облачный сервис Microsoft Azure)	103356	103356	0
Прочие расходы	-	-	-
Эксплуатационные расходы	473317,92	645756	172438,08

Годовой экономический эффект составляет:

$$Э = 172438{,}08 \text{ руб.} – 0{,}0825 \cdot 599631{,}17 \text{ руб.} = 122968{,}51 \text{ руб.}$$

Коэффициент эффективности капитальных затрат:

$$E_р = \frac{172438{,}08 \text{ руб.}}{599631{,}17 \text{ руб.}} = 0{,}29$$

В итоге результаты всех расчётов показали, что проект финансово целесообразен, теперь осталось попробовать привлечь пользователей, чтобы они могли принести прибыль разработчикам.

После того как команда научилась коммуницировать, проект был создан и протестирован, а также были подсчитаны итоговые затраты и срок окупаемости проекта необходимо подвести итог.

Заключение

Цель данной выпускной квалификационной работы было создание уникального мобильного приложения с функциями распорядка дня, заметок и умной картой для русскоговорящей аудитории. Для достижения поставленной цели был решен ряд задач.

Во время выполнения курсового проектирования команда разработчиков успешно выполнены следующие этапы разработки:

− изучены современные подходы к IT-разработке;

− выполнено грамотное распределение ролей между членами команды, в результате чего каждый студент получил ту роль, которая являлась для него посильной и выполнимой;

− составлен график работ, и распределение ресурсов во времени. Данный этап позволил предотвратить возникновение проблем на более поздних этапах разработки. Например, смещение сроков выполнения задач из-за того, что на кого-то был возложен непосильный объем работ;

− проведен сравнительный анализ программных средств интеграции модулей, что позволило выбрать наиболее подходящий инструмент для текущего проекта;

− проведен анализ предметной области, после чего были сформированы требования к программному обеспечению и описание объекта автоматизации;

− выбраны наиболее оптимальные средства разработки и язык программирования;

− спроектировано клиент-серверное взаимодействие между составными частями приложения;

− составлена программная документация на основе предпроектных исследований;

− разработаны модули программного обеспечения, и условные обозначения компонентов системы. Это значительно упростило

взаимодействие коллектива во время разработки, т.к. позволило команде общаться между собой на языке понятных друг другу терминов;

− изучены и применены программные средства для упрощения командной работы над проектом;

− проведено тестирование приложения;

− рассчитана экономическая эффективность проекта.

В итоге у команды получилось создать стабильно работающее клиент-серверное приложение с уникальными возможностями, которое может занять свою нишу на рынке мобильных приложений в категории планировщиков времени.

В процессе выполнения разработки программного проекта был выявлен ряд затруднений, решение которых позволило приобрести практический опыт и навыки разработки программных систем в команде, что позволит в дальнейшем использовать его в будущих проектах.

Существует ряд идей, которые можно реализовать в новой версии проекта. Программа изначально разрабатывалась под систему Android, но благодаря React Native должно работать и на системе iOS, но допускаются небольшие проблемы, которые можно будет устранить. При работе с картой выбранный диаметр поиска на деле может оказаться слишком большим или маленьким, но благодаря обратной связи с пользователями можно будет изменить его и сделать программу более точной для поиска. Также, можно добавить функции, которые есть у конкурентов, возможность синхронизировать приложение с календарем телефона, разработать веб-клиент просмотра списка задач.

Список использованных источников

1) Тестирование объектно-ориентированного программного обеспечения. Практическое пособие: Пер. с англ. / Джон Макгрегор, Девид Сайкс. – К.: ООО «ТИД ДС», 2002. – 432 с.

2) ГОСТ Р ИСО 21500-2014. Руководство по проектному менеджменту. Москва Стандартинформ, 2015. – 45 с.

3) Виды ИТ-проектов, их особенности. Определение целей ИТ-проекта, основные подходы. // e-educ.ru. – Режим доступа: https://e-educ.ru/pm4.html (дата обращения: 15.03.2018).

4) Система контроля версий (cvs) — // Time Doctor [Электронный ресурс]. – Режим доступа: https://biz30.timedoctor.com/ru/c%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8F-%D0%B2%D0%B5%D1%80%D1%81%D0%B8% D0%B9/ (дата обращения: 15.03.2018).

5) Системы управления версиями. Пособие для инженеров, художников и писателей // Habrahabr [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/post/122060/ (дата обращения: 15.03.2018).

6) Шелухин, О. И. Моделирование информационных систем: учебное пособие / О. И. Шелухин. - 2-е изд., перераб. и доп. - М.: Горячая линия - Телеком, 2011. - 536 с.

7) Михеева, Е. В. Информационные технологии в профессиональной деятельности: учебное пособие / Е. В. Михеева. - М.: Проспект, 2010. - 448 с.

8) Право интеллектуальной собственност: учебник / С. А. Судариков. - М.: Проспект, 2011. - 368 с.

9) Черников, Б. В. Управление качеством программного обеспечения: учебник / Б. В. Черников. - М.: Форум; Инфра-М, 2012. - 240 с. - (Высшее образование).

10) Черников, Б. В. Оценка качества программного обеспечения. Практикум: учебное пособие / Б. В. Черников, Б. Е. Поклонов. - М.: Форум; Инфра-М, 2012. - 400 с. - (Высшее образование).

11) Гагарина, Л. Г. Технология разработки программного обеспечения: учеб. пособие для вузов / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадул. - М.: ИНФРА-М, 2009. - 400 с. - (Высшее образование).

12) Балдин, К. В. Информационные технологии в менеджменте: учебник / К. В. Балдин. - М.: Академия, 2012. - 288 с. - (Бакалавриат).

13) Работаем с User stories // Habrahabr [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/company/friifond/blog/284032/ (дата обращения: 15.03.2018).

14) Курсовой проект // Первый шаг [Электронный ресурс]. – Режим доступа: http://www.pervyi-shag.narod.ru/Perwie/kursovoj.htm (дата обращения: 10.05.2016).

15) Белов, В. В. Проектирование информационных систем: учебник / В. В. Белов, В. И. Чистякова. - М.: Академия, 2013. - 352 с. - (Бакалавриат).

16) Затонский, А. В. Информационные технологии: учебник / А. Затонский. - М: Академия, 2001. – 217с.

17) 10 языков для Android-разработчика // GeekBrains [Электронный ресурс]. – Режим доступа: https://geekbrains.ru/posts/android_dev_langs (дата обращения: 15.03.2018).

18) Языки программирования для веб-разработки // Блог веб-программиста [Электронный ресурс]. – Режим доступа: http://juice-health.ru/programming/web-development/505-programming-languages- (дата обращения: 15.03.2018).

19) Android Studio The Official IDE for Android // Android Studio [Электронный ресурс]. – Режим доступа: https://developer.android.com/studio/index.html (дата обращения: 15.03.2018).

20) SQLite Documentation // SQLite [Электронный ресурс]. – Режим доступа: http://www.sqlite.org/docs.html (дата обращения: 15.03.2018).

21) SQLite — замечательная встраиваемая БД (часть 1) // Habrahabr [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/post/149356/ (дата обращения: 15.03.2018).

22) JavaScript.ru // Справочник JavaScript [Электронный ресурс]. – Режим доступа: http://javascript.ru/manual (дата обращения: 15.03.2018).

23) JavaScript.com // JavaScript [Электронный ресурс]. – Режим доступа: https://www.javascript.com/ (дата обращения: 15.03.2018).

24) About Node.js // Node.js [Электронный ресурс]. – Режим доступа: https://nodejs.org/en/about/ (дата обращения: 15.03.2018).

25) Установка: Node.js + Express // ITNote [Электронный ресурс]. – Режим доступа: http://itnote.ru/2014/11/01/setup-node-js-express/ (дата обращения: 15.03.2018).

26) MySQL Documentation // dev.mysql.com [Электронный ресурс]. – Режим доступа: https://dev.mysql.com/doc/ (дата обращения: 15.03.2018).

27) Документация по MySQL // MySQL.ru [Электронный ресурс]. – Режим доступа: http://www.mysql.ru/docs/ (дата обращения: 15.03.2018).

28) Приступая к работе с Azure // Microsoft Azure [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/azure/ (дата обращения: 15.03.2018).

29) Создание веб-приложений Node.js в Azure // Microsoft Azure [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/azure/app-service/app-service-web-get-started-nodejs (дата обращения: 15.03.2018).

30) 7. Лекция: Моделирование бизнес-процессов средствами BPwin // Интернет университет информационных технологий [Электронный ресурс]. – Режим доступа: http://old.intuit.ru/department/se/devis/7/ (дата обращения: 15.03.2018).

31) Информационные системы и технологии управления [Электронный ресурс]: учебник / Под ред. Г. А. Титоренко. - 3-е изд., перераб. и доп. - М.: ЮНИТИ-ДАНА, 2011. - 1 эл. опт. диск (CD-ROM).

32) Информационные системы и технологии в экономике и управлении: учебник / Под ред. В. В. Трофимова. - 4-е изд., перераб. и доп. - М.: Юрайт, 2013. - 542 с. - (Бакалавр. Базовый курс).

33) Буч, Г. Язык UML. Руководство пользователя: учебное пособие / Г. Буч, Дж. Рамбо, И. Якобсон; Пер. с англ. - 2-е изд. - М.: ДМК Пресс, 2007. - 496 с.

34) Гагарина, Л. Г. Технология разработки программного обеспечения: учеб. пособие для вузов / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадул. - М.: ИНФРА-М, 2009. - 400 с. - (Высшее образование).

35) Балдин, К. В. Информационные технологии в менеджменте: учебник / К. В. Балдин. - М.: Академия, 2012. - 288 с. - (Бакалавриат).

36) Creately // Diagram Maker [Электронный ресурс]. – Режим доступа: https://creately.com/ (дата обращения: 15.03.2018).

37) Гибкая методология разработки «Scrum» // Habrahabr [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/post/247319/ (дата обращения: 15.03.2018).

38) AgileRussia – Обзор методологии Scrum [Электронный ресурс]. – Режим доступа: http://agilerussia.ru/methodologies/обзор-методологии-scrum/ (дата обращения: 10.05.2016).

39) 3. MSDN – Scrum [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/library/dd997796(v=vs.100).aspx (дата обращения: 10.05.2016).

40) Технология программирования: Учебник для вузов. – М.: Изд-во МГТУ им. Н.Э. Баумфнф, 2002. – 320 с.

41) «Stack Overflow на русском» — сайт вопросов и ответов для программистов. // Stack Overflow на русском [Электронный ресурс]. – Режим доступа: https://ru.stackoverflow.com/ (дата обращения: 15.03.2018).

42) Learn, Share, Build // Stack Overflow. – Режим доступа: https://stackoverflow.com/ (дата обращения: 15.03.2018).

43) Утвержден ФГОС ВО по направлению подготовки 09.03.04 Программная инженерия (далее соответственно - программа бакалавриата, направление подготовки). // Портал Федеральных государственных образовательных стандартов высшего образования [Электронный ресурс]. – Режим доступа: http://fgosvo.ru/news/1/1086 (дата обращения: 26.05.2018).

44) Sibak // XXXIII Международной научно-практической конференции «Научное сообщество студентов: МЕЖДИСЦИПЛИНАРНЫЕ ИССЛЕДОВАНИЯ» (Россия, г. Новосибирск, 16 ноября 2017 г.) [Электронный ресурс]. – Режим доступа: https://sibac.info/studconf/science/xxxiii/86915 (дата обращения: 18.03.18).

45) Материалы конференции // СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ имени академика М.Ф.Решетнева «АКТУАЛЬНЫЕ ПРОБЛЕМЫ АВИАЦИИ И КОСМОНАВТИКИ» [Электронный ресурс]. – Режим доступа: https://apak.sibsau.ru/page/materials (дата обращения: 04.06.2018).

46) Моделирование архитектуры приложения // Developer Network [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/dd490886.aspx (дата обращения: 04.05.2018).

47) Базы данных и СУБД // Community by timeweb [Электронный ресурс]. – Режим доступа: https://timeweb.com/ru/community/articles/bazy-dannyh-i-subd-1 (дата обращения: 04.05.2018).

48) Когаловский М.Р. Энциклопедия технологий баз данных. — М.: Финансы и статистика, 2002. — 800 с. — ISBN 5-279-02276-4.

49) Кузнецов С. Д. Основы баз данных. — 2-е изд. — М.: Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний, 2007. — 484 с. — ISBN 978-5-94774-736-2.

50) Дейт К. Дж. Введение в системы баз данных = Introduction to Database Systems. — 8-е изд. — М.: Вильямс, 2005. — 1328 с. — ISBN 5-8459-0788-8 (рус.) 0-321-19784-4 (англ.).

51) Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика = Database Systems: A Practical Approach to Design, Implementation, and Management. — 3-е изд. — М.: Вильямс, 2003. — 1436 с. — ISBN 0-201-70857-4.

52) Гарсиа-Молина Г., Ульман Дж., Уидом Дж. Системы баз данных. Полный курс = Database Systems: The Complete Book. — Вильямс, 2003. — 1088 с. — ISBN 5-8459-0384-X.

53) habr // Разработка мобильных приложений: с чего начать [Электронный ресурс]. – Режим доступа: https://habr.com/company/mailru/blog/179113/ (дата обращения: 04.05.2018).

54) JSON Web Tokens - jwt.io [Электронный ресурс]. – Режим доступа: https://jwt.io/ (дата обращения: 04.05.2018).

55) Составные части хеш-строки bcrypt – RMCreative [Электронный ресурс]. – Режим доступа: https://jwt.io/ (дата обращения: 04.05.2018).

56) react-entity-editor // npm [Электронный ресурс]. – Режим доступа: https://rmcreative.ru/blog/post/sostavnye-chasti-kheshch-stroki-bcrypt (дата обращения: 04.05.2018).

57) Больше, чем React: Почему не следует использовать ReactJS для сложных интерактивных фронтенд-проектов // habr [Электронный ресурс]. – Режим доступа: https://habr.com/company/nixsolutions/blog/324748/ (дата обращения: 04.05.2018).

58) Полное руководство по ReactJS // Learn React JS [Электронный ресурс]. – Режим доступа: https://learn-reactjs.ru/home (дата обращения: 04.05.2018).

ПРИЛОЖЕНИЕ А – ПАСПОРТ ПРОЕКТА

Таблица 1 – Назначение проекта

<table>
<tr><td>Цель</td><td colspan="3">Мобильное приложение, упрощающее работу с личными задачами и расписанием пользователя. Наличие возможности обмена списками задач между пользователями и модуля интерактивных уведомлений.</td></tr>
<tr><td rowspan="5">Задачи</td><td colspan="3">1.Разработать интуитивно понятный интерфейс</td></tr>
<tr><td colspan="3">2.Реализовать возможность добавления, редактирования, удаления элементов расписания</td></tr>
<tr><td colspan="3">3.Реализовать возможность синхронизации расписаний между необходимыми узлами</td></tr>
<tr><td colspan="3">4.Реализовать «Интерактивную карту»</td></tr>
<tr><td colspan="3">5.Реализовать систему уведомлений</td></tr>
<tr><td rowspan="7">Состав работ</td><td>1.Изучение и анализ данных</td><td rowspan="7">Объекты автоматизации</td><td>1.Расписание и задачи пользователя</td></tr>
<tr><td>2.Поиск альтернатив и существующих решений</td><td></td></tr>
<tr><td>3.Проектирование</td><td></td></tr>
<tr><td>4.Кодирование</td><td></td></tr>
<tr><td>5.Тестирование</td><td></td></tr>
<tr><td>6.Внедрение</td><td></td></tr>
<tr><td>Комментарий</td><td colspan="3">Приложение позволит грамотно планировать свой день семье с возможностью удалённо вносить изменение каждому члену семьи, и так же отмечать на карте необходимые объекты.</td></tr>
</table>

В паспорте проекта зафиксированы задачи, которые необходимо решить команде.

Любой проект имеет бизнес-процессы, а их моделирование позволяет отобразить поток работы проекта графическим образом.

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ И. В. Евдокимов
подпись инициалы, фамилия
«19» июня 2018 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.04 «Программная инженерия»

Мобильное приложение для управления персональными заметками и распорядком дня

Руководитель	19.06.18г.	доцент, к.т.н.	И. В. Евдокимов
	подпись, дата	должность, ученая степень	инициалы, фамилия
Выпускник	19.06.18г.		А. И. Водянкин
	подпись, дата		инициалы, фамилия
Нормоконтролер	19.06.18	доцент, к.т.н.	О. А. Антамошкин
	подпись, дата	должность, ученая степень	инициалы, фамилия

Красноярск 2018