Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа экономики и управления
Кафедра «Информационные технологии в экономике»

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой, проф., д.т.н.
_____Б.М. Суховилов
«___»_____ 2019 г.

Управление рисками и процессом разработки мобильного приложения

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(магистерская диссертация)

ЮУрГУ – 38.03.01.2019.301/70.ВКР

Руководитель работы,
д.т.н., ст. научный сотрудник
_____Б.М. Суховилов
«___»_____ 2019 г.

Автор работы,
студент группы ЭУ–235
_____А.В. Казьмина
«___»_____ 2019 г.

Нормоконтролер,
ст. преподаватель кафедры
_____ Е.Н. Горных
«___»_____ 2019 г.

Челябинск 2019

АННОТАЦИЯ

Казьмина А.В. Управление рисками и процессами разработки мобильного приложения. – Челябинск: ЮУрГУ, ЭУ-235, 67 с., 22 ил., 1 табл., библиогр. список – 12 наим.

Выпускная квалификационная работа выполнена с целью повышения эффективности управления процессами разработки для снижения рисков и повышения производительности команды.

В выпускной квалификационной работе были рассмотрены теоретические вопросы, касающиеся инструментов управления проектами.

Был проведен анализ различных инструментов. Выбраны оптимальные инструменты для данного проекта.

Проанализирован процесс разработки, разработаны шаги по повышению эффективности управления и повышению производительности команды.

ANNOTATION

Kazmina A.V. Management of risk and development of the mobile app. – Chelyabinsk: SUSU, EU-235, 67 p., 22 pic., 1 tabl., bibliogr. list – 12 points.

The aim of the graduation qualification work is increase the efficiency of the management of development processes to reduce risks and increase the team productivity.

Theoretical issues about the project management tools was considered in the graduation qualification work.

The analyses of various tools was conducted, high tools were determined.

The development process was analyzed, the steps for increasing efficiency of management and development of the team were identified.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Эффективное управление процессами разработки мобильного приложения приводит к сокращению сроков разработки и улучшению качества, а его отсутствие – к высоким рискам. Это подтверждает актуальность данной темы. "Управление процессами разработки" - это комплексное, многоплановое понятие, к которому, однако, еще не сформировалось общепринятого подхода. Некоторые специалисты рассматривают его как вопрос управления, и прежде всего людьми. Другие считают, что это в первую очередь управление непосредственно самой разработкой. Довольно много внимания уделяется необходимости автоматизировать процесс выстраивания коммуникаций с клиентами.

В компанию «Наполеон IT» поступила заявка на разработку мобильного приложения. Проект «YouFix» - это сервис для мексиканского рынка, который поможет найти местного специалиста для ремонта или другой работы. Оценка объемов работ показала, что проект будет длиться более четырех месяцев и потребует команду из восьми человек.

Основная цель компании – создать востребованный продукт, который полностью будет удовлетворять требованиям заказчика и который будет выполнен в срок.

Наиболее эффективным решением для данного проекта является выделение команды на проект. Для организации работы команды, проработки требований проекта, согласование деталей выделяется отдельная роль – менеджер проекта.

Основные риски проекта заключаются в возможности бесконечной переработки задач в связи с быстро меняющимися требованиями клиента, различном понимании задач из-за высокой взаимосвязанности и сложности

функций, понижение лояльности клиента, если снизится скорость разработки, возрастет количество ошибок или процесс разработки будет не прозрачным.

Целью магистерской диссертации является повышение эффективности управления процессами разработки для снижения рисков и повышения производительности команды.

Для достижения цели были поставлены следующие задачи:

- рассмотреть существующие инструменты для управления проектом.
- выбрать наиболее подходящие инструменты.
- провести анализ процесса разработки.
- установить наиболее эффективные решения.
- построить оптимальные процессы управления разработкой данного проекта.

В первой главе дипломной работы изучены теоретические основы инструментов: рассмотрены методы управления проектом, различные варианты архитектуры и баз данных, а также различные брокеры запросов.

Во второй главе выбраны основные фреимворки, инструменты для управления разработкой, дизайна, а также определены архитектура проекта.

В третьей главе описаны различные кейсы процессов разработки проекта, а именно установление последовательности, выбор промежутка времени для показов клиенту, перемещение времени планирования, необходимость документации, относительная оценка, проведение ретроспектив с командой.

1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗЛИЧНЫХ ИНСТРУМЕНТОВ ДЛЯ УПРАВЛЕНИЯ ПРОЕКТАМИ

1.1 Методы управления проектом

Существует несколько методологий управления проектом. Гибкая и каскадная модели разработки проекта (Agile и Waterfall) — одни из наиболее популярных.

Agile — система идей и принципов «гибкого» управления проектами, на основе которых разработаны популярные методы Scrum, Kanban и другие. Ключевой принцип — разработка через короткие итерации (циклы), в конце каждого из которых заказчик (пользователь) получает рабочий код или продукт.

Waterfall — методика управления проектами, которая подразумевает последовательный переход с одного этапа на другой без пропусков и возвращений на предыдущие стадии.

1.1.1 Каскадная модель управления проектом

Каскадная (водопадная) модель разработки подразумевает последовательное прохождение процесса, разбитого на стадии. Переход к новому этапу возможен только после завершения предыдущего (Рисунок 1).

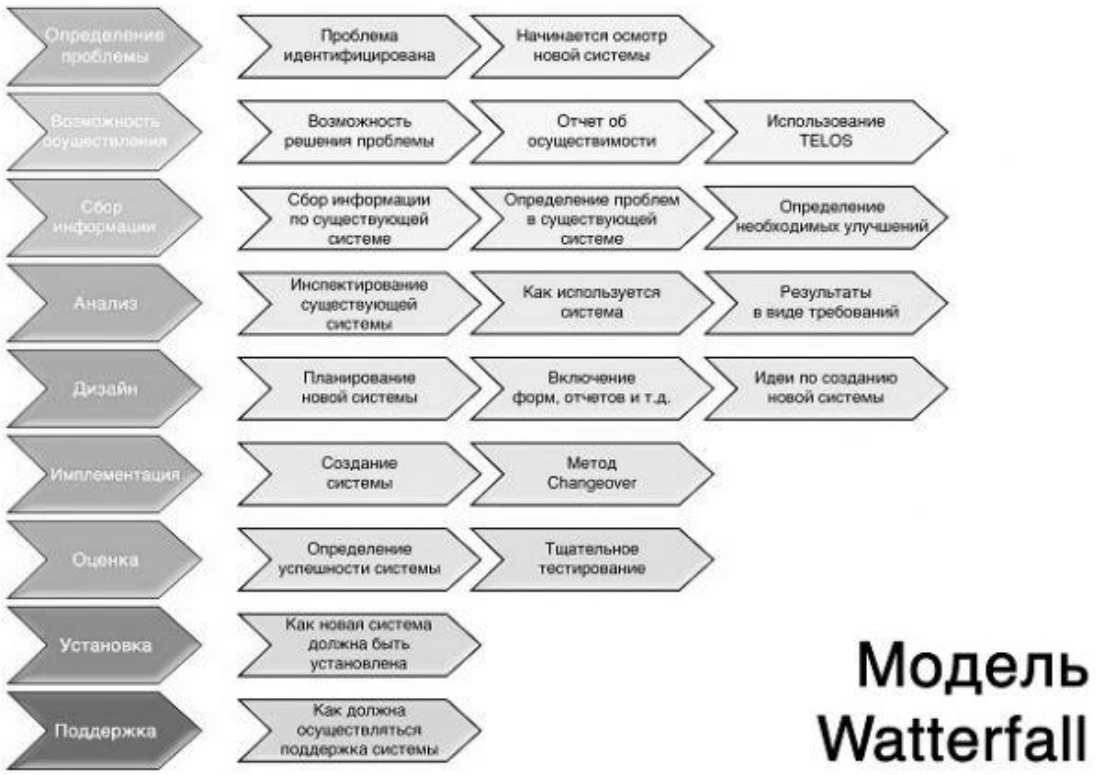

Рисунок 1 - Каскадная модель разработки

В оригинальной работе Уолкера «Managing the development of large software systems» описаны 6 стадий разработки продукта, которые в 1985 году Департамент защиты США закрепил в стандартах работы с разработчиками программного обеспечения:

1. Системные и программные требования: закрепляются в PRD (документе требований к продукту).
2. Анализ: воплощается в моделях, схемах и бизнес-правилах.
3. Дизайн: разрабатывается внутренняя архитектура программного обеспечения, способы реализации требований. Это не только об интерфейсе и внешнем виде ПО, но и о его внутренней структурной логике.

4. Кодинг: непосредственно пишется код программы, идёт интеграция программного обеспечения.
5. Тестирование: баг-тестеры (тестировщики) проверяют финальный продукт, занося в трекеры сведения о дефектах кода программы или функционала. В случае ошибок и наличия времени/финансов происходит исправление багов.
6. Операции: продукт адаптируется под разные операционные системы, регулярно обновляется для исправления обнаруженных пользователями багов и добавления функционала. В рамках стадии также осуществляется техническая поддержка клиентов.

Компания Toyota, популяризовавшая методологии Lean и Kanban, до конца 2000-ых пользовалась каскадной моделью разработки ПО для нужд производства.

Преимущества и недостатки Waterfall.

В число наибольших преимуществ методики Waterfall вошли:

1. понятная и простая структура процесса разработки — это снижает порог вхождения для команд.
2. удобная отчётность — можно легко отследить ресурсы, риски, затраченное время и финансы благодаря строгой этапности процесса разработки и детальной документации проекта.
3. стабильность задач — задачи, которые стоят перед продуктом, ясны команде с самого начала разработки, и остаются неизменными на протяжении всего процесса.

4. оценка стоимости и сроков сдачи проекта — сроки выпуска готового продукта, как и его итоговая стоимость могут быть просчитаны до момента запуска разработки.

Среди недостатков водопадного метода можно выделить:

1. лишенный гибкости процесс — так, если проект требует больше временных и финансовых ресурсов, чем возможно, то под нож пойдёт фаза тестирования. Согласно исследованиям консалт-группы Rothman, стоимость исправления багов после выпуска продукта выше в среднем в 20 раз, чем во время полноценного многоэтапного тестирования в процессе разработки
2. «стойкость» к изменениям — жёсткий каркас из этапов разработки и условие предоставление только готового продукта определяют невозможность вносить изменения во время разработки
3. инерционность — на первых стадиях прогноз временных и финансовых трат может измениться в сторону увеличения, но изменить проект в сторону оптимизации затрат, изменения функционала или концепции до выпуска готового продукта невозможно
4. повышенный риск — классическая система тестирования подразумевает отдельно тестирование каждого из компонентов проекта, в том числе, во взаимодействии с другими. При использовании Waterfall происходит тестирование готового продукта [1].

1.1.2 Agile

Принципы Agile-разработки:

1. Люди и взаимодействие важнее процессов и инструментов.

2. Работающий продукт важнее исчерпывающей документации.
3. Сотрудничество с заказчиком важнее согласования условий контракта.
4. Готовность к изменениям важнее следования первоначальному плану.

Agile стал основой для целого ряда гибких методик, среди которых наиболее известны Scrum, kanban.

Преимущества и недостатки метода Agile

К преимуществам метода относятся:

- короткие и понятные итерации — циклы разработки длятся от 2 недели до 2 месяцев, по окончанию которых заказчик получает рабочую версию продукта;
- высокая степень вовлечения исполнителей, организаторов и заказчиков проекта;
- рабочий продукт как основной показатель прогресса;
- минимизация рисков благодаря гибкой системе внесения изменений.

Недостатки Agile:

- стимулирование постоянных изменений проекта: гибкость разработки продукта может привести к тому, что он никогда не дойдёт до финальной версии;
- повышенные требования к квалификации и опыту команды: помимо непосредственно создания продукта команда должна анализировать возможные способы улучшения эффективности собственной работы,

беспрерывно обмениваться информацией по проекту, быть мотивированной и самоорганизованной. Далеко не всегда ресурсы проекта позволяют привлечь таких специалистов;

- философский характер методологии: Agile — это не чёткая инструкция к действию, а целая философская концепция. Команда не может механически применить механики «гибкой» разработки, нужно принять ключевые принципы системы;
- сложность подсчёта итоговой суммы работы: стимуляция изменений и усовершенствования конечного продукта приводит к плавающему значению стоимости проекта.

1.1.2.1 Подход Scrum

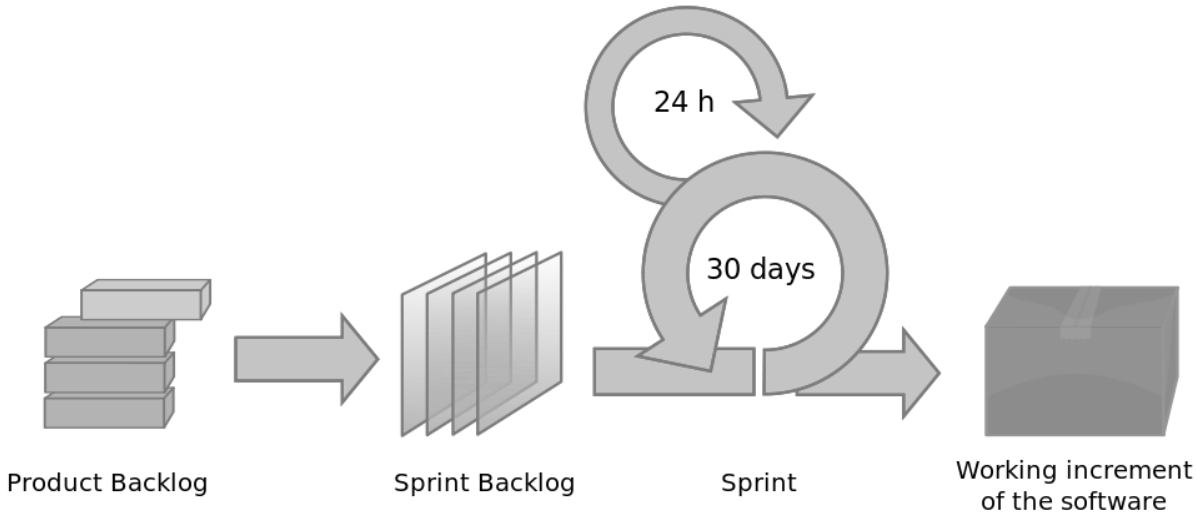

Рисунок 2 - Процесс Scrum

Основой Scrum является спринт, в течение которого выполняется работа над продуктом. По окончанию спринта должна быть получена новая рабочая

версия продукта. Спринт всегда ограничен по времени (1-4 недели) и имеет одинаковую продолжительность на протяжении все жизни продукта (Рисунок 2).

Перед началом каждого спринта производится планирование, на котором производится оценка содержимого беклог и формирование беклога спринта – это задачи, которые должны быть выполнены в текущем спринте. Каждый спринт должен иметь цель, которая является мотивирующим фактором и достигается с помощью выполнения задач из беклога спринта.

Каждый день производится митинг, на котором каждый член команды отвечает на вопросы «что я сделал вчера?», «что я планирую сделать сегодня?», «какие препятствия на своей работе я встретил?». Задача митинга — определение статуса и прогресса работы над спринтом, раннее обнаружение возникших препятствий, выработка решений по изменению стратегии, необходимых для достижения целей спринта.

По окончанию спринта производятся ревью и проводится ретроспектива, задача которых оценить эффективность (производительность) команды в прошедшем спринте, спрогнозировать ожидаемую эффективность (производительность) в следующем спринте, выявлении имеющихся проблем, оценки вероятности завершения всех необходимых работ по продукту и другое.

Рекомендуемый размер команды — 7 (плюс-минус 2) человека. Согласно идеологам Scrum, команды большего размера требуют слишком больших ресурсов на коммуникации, в то время как команды меньшего размера повышают риски (за счет возможного отсутствия требуемых навыков) и уменьшают размер работы, который команда может выполнить в единицу времени [2-5].

1.1.2.2 Подход Kanban

В упрощенном варианте, Kanban включает в себя два простых правила:

1. Производственная станция имеет план производства деталей («backlog»). План отсортирован по приоритету, и может меняться в любое время.
2. Количество задач, выполняемых на станции одновременно ограничено. Это ограничение необходимо для управления скоростью производства на станции, а также скоростью реагирования на изменения плана.

Задача Kanban – сбалансировать разных специалистов внутри команды и избежать ситуации, когда дизайнеры работают сутками, а разработчики жалуются на отсутствие новых задач.

Главный показатель эффективности в kanban – это среднее время прохождения задачи по доске. Задача прошла быстро – команда работала продуктивно и слаженно. Задача затянулась – надо думать, на каком этапе и почему возникли задержки и чью работу надо оптимизировать.

Для визуализации agile-подходов используют доски: физические и электронные. Они позволяют сделать рабочий процесс открытым и понятным для всех специалистов, что важно, когда у команды нет одного формального руководителя [6].

1.2 Варианты архитектуры проекта

1.2.1 Монолитная архитектура

Помимо выбора подхода управления процессом разработки проекта необходимо определить технические характеристики. Одним из ключевых моментов является выбор архитектуры.

Монолитный сервер — распространенный способ построения систем. Логика по обработке запросов выполняется в единственном процессе, при этом существует возможность использования любого языка программирования для разделения приложения на классы, функции и namespace-ы. Есть возможность запускать и тестировать приложение на машине разработчика и использовать стандартный процесс развертывания для проверки изменений перед выкладыванием их в использование. Масштабировать монолитное приложение возможно горизонтально, путем запуска нескольких физических серверов за балансировщиком нагрузки.

Все большую популярность набирает развертывание монолитных приложений в облаке. Однако есть недостатки. Любые изменения, даже самые небольшие, требуют пересборки и развертывания всего монолита. С течением времени, становится труднее сохранять хорошую модульную структуру, изменения логики одного модуля имеют тенденцию влиять на код других модулей. Масштабировать приходится все приложение целиком, даже если это требуется только для одного модуля этого приложения.

Монолитная архитектура не соответствовала потребностям бизнеса в плане ускорения, так как крупные сервисы сложно быстро изменять и адаптировать. Для ускорения наиболее логичным решением является создание типовых микросервисов, которые по-разному реализованы в крупных сервисах и выделены в самостоятельные компоненты. Так каждый крупный сервис может использовать микросервисы по необходимости. Исключение дублирования типовых функций сделало программы более легкими и гибкими.

1.2.2 Микросервисная архитектура

Архитектурный стиль микросервисов — это подход, при котором единое приложение строится как набор небольших сервисов, каждый из которых работает в собственном процессе и коммуницирует с остальными, используя легковесные механизмы, как правило HTTP. Эти сервисы построены вокруг бизнес-потребностей и развертываются независимо с использованием полностью автоматизированной среды. Существует абсолютный минимум централизованного управления этими сервисами. Сами по себе эти сервисы могут быть написаны на разных языках и использовать разные технологии хранения данных.

Преимущества микросервисов:

1. Жесткие границы модулей: четкая модульная структура, что крайне удобно для больших команд.
2. Независимое развертывание: микросервисы проще в развертывании, и поскольку они автономны, меньше шансов вызвать сбои системы, когда они работают некорректно.
3. Технологическое разнообразие: С микросервисами легко смешивать языки программирования, среды разработки и технологии хранения данных.

Недостатки микросервисов:

1. Распределенность: Распределенные системы сложнее программировать, так как удаленные вызовы работают медленно и невозможно исключить риск сбоя.
2. Согласованность в конечном счете: Для распределенной системы сложно поддерживать строгую согласованность. Поэтому каждый обеспечивает ее самостоятельно.

3. Сложность эксплуатации: Для управления микросервисами, которые регулярно переразвертываются, требуется зрелая группа эксплуатации/поддержки.

Микросервисная архитектура устранила недостатки монолитного ПО, обеспечив:

- изоляцию и минимизацию изменений;
- ускорение разработки;
- возможность легко подстраивать ПО под структуру бизнеса [7].

1.3 Различные базы данных

1.3.1 PostgreSQL

PostgreSQL - объектно-реляционная СУБД. Это даёт ему некоторые преимущества над другими SQL базами данных с открытым исходным кодом, такими как MySQL, MariaDB и Firebird.

Фундаментальная характеристика объектно-реляционной базы данных — это поддержка пользовательских объектов и их поведения, включая типы данных, функции, операции, домены и индексы. Это делает PosrgreSQL гибким и надежным. Среди прочего, он умеет создавать, хранить и извлекать сложные структуры данных.

PosrgreSQL стремится соответствовать стандарту ANSI-SQL:2008, отвечает требованиям ACID (атомарность, согласованность, изолированность и надежность) и известен своей ссылочной и транзакционной целостностью. Первичные ключи, ограничивающие и каскадные внешние ключи, уникальные ограничения, ограничения NOT NULL, проверочные ограничения и другие

функции обеспечения целостности данных дают уверенность, что только корректные данные будут сохранены.

У PosrgreSQL множество возможностей. Созданный с использованием объектно-реляционной модели, он поддерживает сложные структуры и широкий спектр встроенных и определяемых пользователем типов данных. Он обеспечивает расширенную ёмкость данных и целостность данных [8].

1.3.2 MongoDB

MongoDB реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи объектно-реляционным базам данных.

В отличие от реляционных баз данных MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать.

Вся система MongoDB может представлять не только одну базу данных, находящуюся на одном физическом сервере. Функциональность MongoDB позволяет расположить несколько баз данных на нескольких физических серверах, и эти базы данных смогут легко обмениваться данными и сохранять целостность.

Одним из популярных стандартов обмена данными и их хранения является JSON (JavaScript Object Notation). JSON эффективно описывает сложные по структуре данные. Способ хранения данных в MongoDB в этом плане похож на JSON, хотя формально JSON не используется. Для хранения в MongoDB применяется формат, который называется BSON (БиСон) или сокращение от binary JSON.

BSON позволяет работать с данными быстрее: быстрее выполняется поиск и обработка. Хотя надо отметить, что BSON в отличие от хранения данных в формате JSON имеет небольшой недостаток: в целом данные в JSON-формате занимают меньше места, чем в формате BSON, с другой стороны, данный недостаток с лихвой окупается скоростью.

MongoDB написана на C++, поэтому ее легко портировать на самые разные платформы. MongoDB может быть развернута на платформах Windows, Linux, MacOS, Solaris. Можно также загрузить исходный код и самому скомпилировать MongoDB, но рекомендуется использовать библиотеки с офсайта.

Если реляционные базы данных хранят строки, то MongoDB хранит документы. В отличие от строк документы могут хранить сложную по структуре информацию. Документ можно представить как хранилище ключей и значений.

В MongoDB есть коллекции. И если в реляционных БД таблицы хранят однотипные жестко структурированные объекты, то в коллекции могут содержать самые разные объекты, имеющие различную структуру и различный набор свойств.

Система хранения данных в MongoDB представляет набор реплик. В этом наборе есть основной узел, а также может быть набор вторичных узлов. Все вторичные узлы сохраняют целостность и автоматически обновляются вместе с обновлением главного узла. И если основной узел по каким-то причинам выходит из строя, то один из вторичных узлов становится главным.

Отсутствие жесткой схемы базы данных и в связи с этим потребности при малейшем изменении концепции хранения данных пересоздавать эту схему значительно облегчают работу с базами данных MongoDB и дальнейшим их масштабированием. Кроме того, экономится время разработчиков. Им больше не надо думать о пересоздании базы данных и тратить время на построение сложных запросов [9].

1.4 Брокеры запросов

AMQP (Advanced Message Queuing Protocol) — открытый протокол для передачи сообщений между компонентами системы. Основная идея состоит в том, что отдельные подсистемы (или независимые приложения) могут обмениваться произвольным образом сообщениями через AMQP-брокер, который осуществляет маршрутизацию, гарантирует доставку, распределение потоков данных, подписку на нужные типы сообщений.

Протокол AMQP основан на трёх понятиях:

1. Сообщение (message) — единица передаваемых данных, основная его часть (содержание) никак не интерпретируется сервером, к сообщению могут быть присоединены структурированные заголовки.
2. Точка обмена (exchange) — в неё отправляются сообщения. Точка обмена распределяет сообщения в одну или несколько очередей. При этом в точке обмена сообщения не хранятся.
3. Очередь (queue) — здесь хранятся сообщения до тех пор, пока не будут забраны клиентом. Клиент всегда забирает сообщения из одной или нескольких очередей.

Преимущества AMPQ перед JMS

AMPQ часто сравнивают с JMS (Java Message Service), более популярную систему обмена сообщениями в Java сообществе. Ограничение JMS заключается в том, что API определено, а формат сообщений - нет. В отличие от AMPQ, JMS не имеет требований к тому, как сообщение будет сформировано и передано. Фактически, каждый JMS брокер может создавать сообщения различного формата. Они просто должны использовать одно и то же API.

Поэтому Pivotal выпустила собственный JMS проект Rabbit, библиотеку, которая реализует JMS API, но использует RabbiMQ, AMQP брокер для передачи сообщений.

AMQP публикует эти спецификации в доступном для загрузки XML формате. Такая доступность делает её простой для библиотеки, поддерживаемую генерацию API по спецификации, а также автоматическую генерацию алгоритмов для сериализации и десериализации сообщений.

Эти преимущества и открытость спецификации вдохновили на создание множества брокеров, которые поддерживают AMQP, среди которых:

- RabbiMQ
- ActiveMQ
- Qpid

AMPQ и JMS терминология

JMS имеет очереди и списки. Сообщение, отправленное в JMS очередь, предназначено не более чем одному клиенту. Сообщение, отправленное в JMS список, может быть предназначено нескольким потребителям. AMQP имеет только очереди. Пока AMQP очереди взаимодействуют с единственным получателем, AMQP не производит публикацию в очереди напрямую. Сообщение, опубликованное для обмена, может быть отправлено одной или несколькими очередями, эффективно эмулируя JMS очереди и списки.

JMS и AMQP имеют эквивалентные заголовки сообщения, предоставляя средства для сортировки и маршрутизации сообщений.

JMS и AMQP имеют брокеров, ответственных за получение, маршрутизацию и, в конечном счете, распространения сообщений потребителям.

AMQP имеет средства обмена, маршрутизации и очереди. Сообщения вначале публикуются для обмена. Маршрутизаторы определяют, в какой очереди будет передано сообщение. Потребители подписываются на эту очередь для получения копии сообщения. Если подписывается более одного потребителя к одной и той же очереди, то сообщения доставляются по кругу каждому из потребителей [10].

В результате можно сделать следующие выводы:

1. Существуют различные методы управления проектом. Наиболее популярные – это водопадная модель и Agile. Выбор метода осуществляется в зависимости от характеристик проекта и потребностей заказчика.
2. Выбор архитектуры проекта зависит от технических особенностей проекта. Микросервисная архитектура позволяет масштабировать отдельные сервисы по необходимости. Монолитная более проста в использовании, при ней проще поддерживать согласованность код и взаимосвязь между модулями.
3. Также различные характеристики имеют базы данных. PosrgreSQL поддерживает сложные структуры и широкий спектр встроенных и определяемых пользователем типов данных. Он обеспечивает расширенную ёмкость данных и целостность данных. В отличие от реляционных баз данных MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать.
4. Выбор брокера запросов также определяется исходя из потребностей проекта. Были рассмотрены JMS и AMQP брокеры.

2 ВЫБОР ИНСТРУМЕНТОВ ДЛЯ УПРАВЛЕНИЯ ПРОЕКТОМ

2.1 Написание технического задания

В ходе проведенного анализа были определены преимущества и недостатки различных инструментов для управления проектом. Для наиболее эффективного управления были выбраны подходящие инструменты.

Первый этап проекта заключается в написании технического задания.

Техническое задание (ТЗ) — исходный документ на проектирование технического объекта (изделия). ТЗ устанавливает основное назначение разрабатываемого объекта, его технические характеристики, показатели качества и технико-экономические требования, предписание по выполнению необходимых стадий создания документации (конструкторской, технологической, программной и т. д.) и её состав, а также специальные требования.

Техническое задание является юридическим документом — как приложение включается в договор между заказчиком и исполнителем на проведение проектных работ и является его основой: определяет порядок и условия работ, в том числе цель, задачи, принципы, ожидаемые результаты и сроки выполнения. То есть должны быть объективные критерии, по которым можно определить, сделан ли тот или иной пункт работ или нет.

Все изменения, дополнения и уточнения формулировок ТЗ обязательно согласуются с заказчиком и им утверждаются. Это необходимо и потому, что в случае обнаружения в процессе решения проектной задачи неточностей или ошибочности исходных данных возникает необходимость определения степени вины каждой из сторон-участниц разработки, распределения понесенных в связи с этим убытков.

Техническое задание, как термин в области информационных технологий – это юридически значимый документ, содержащий исчерпывающую информацию, необходимую для постановки задач исполнителям на разработку, внедрение или

интеграцию программного продукта, информационной системы, сайта, портала либо прочего ИТ сервиса [11].

После составления технического задания клиент получает:

- Ясную картину желаемого результата;
- Снижение риска: несовпадение ожиданий и результата, снижение затрат на дополнительную разработку;
- Контроль разработки: есть четкий порядок действий к выполнению

Написание ТЗ ведется с постоянным контактом с клиентом. Для более удобной работы был выбран инструмент Miro (онлайн-доска).

Miro, экс-RealtimeBoard — платформа для визуальной коллаборации распределённых команд. Создана для UX-дизайнеров, продакт-менеджеров и agile-коучей для эффективной работе в команде.

Miro позволяет выстраивать бизнес-логику, добавлять скетчи экранов и любые материалы, относящиеся к проекту. Доступ к доске выдается ограниченному кругу лиц по почте. Так, каждый участник проекта может видеть всю актуальную информацию по проекту, так как все изменения на доске происходят в реальном времени.

Структура технического задания:

1. Предварительный анализ.
2. Бизнес-моделирование (user story development).
3. Системный анализ.
4. Проектирование интерфейсов.

Предварительный анализ включает в себя:

- Описание целей и маркетинговых задач проекта;
- KPI проекта;
- Краткое описание проекта и его целевой аудитории;
- Описание основного функционала.

Результат: полная маркетинговая картина проекта.

Бизнес-моделирование:

- Подготовка сценариев использования (Рисунок 4);
- Описание бизнес-логики работы приложения (Рисунок 3).

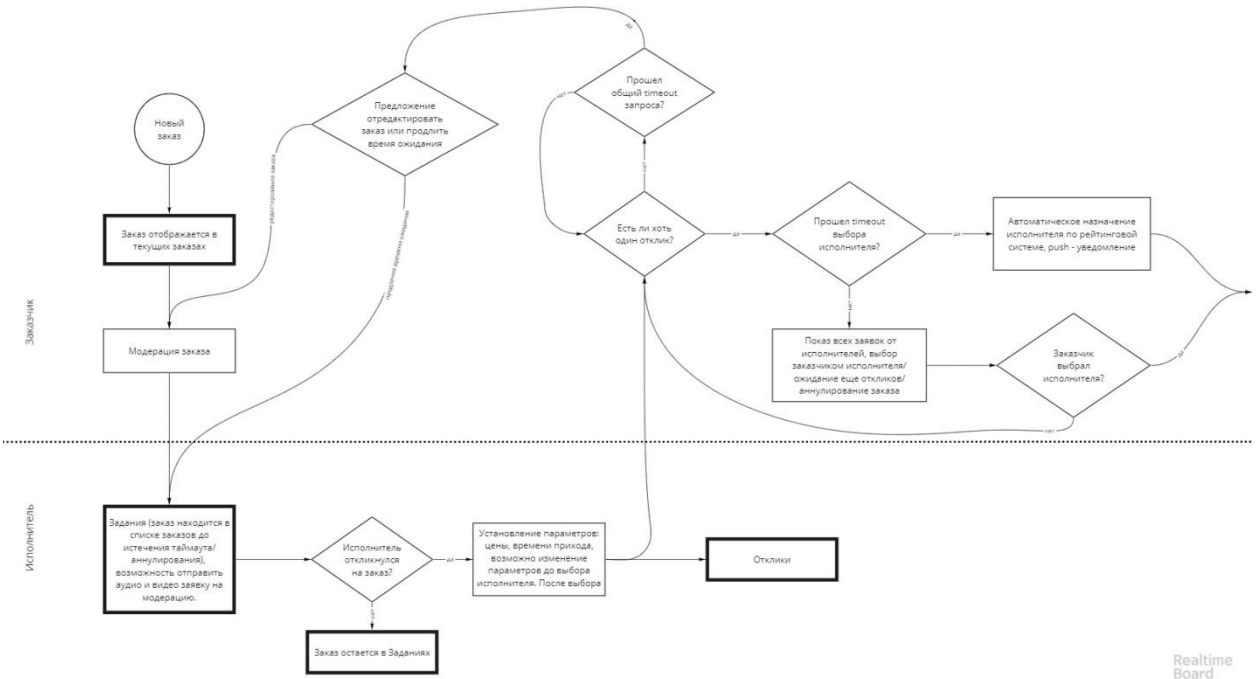

Рисунок 3 - Пример описания бизнес-логики

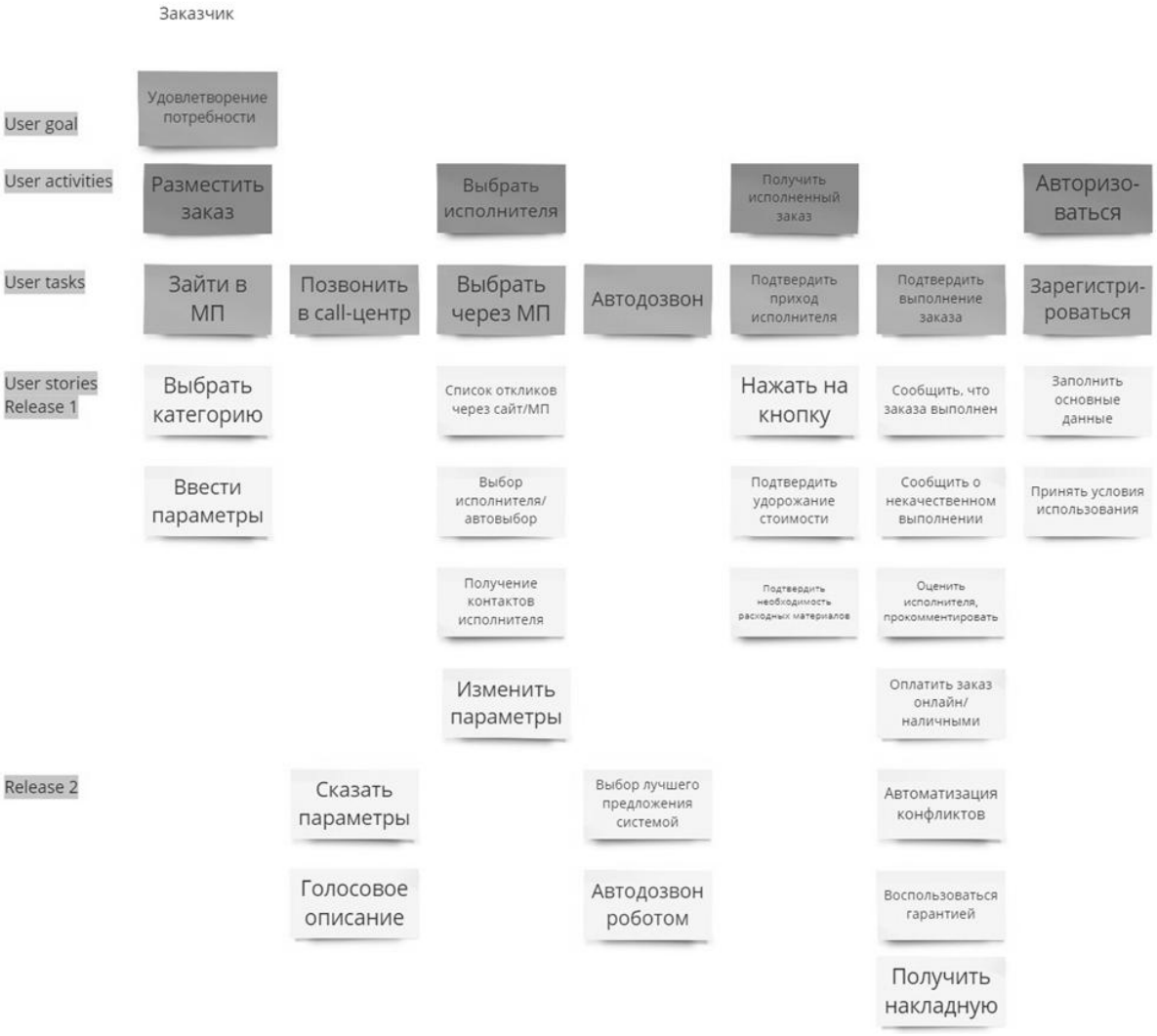

Рисунок 4 - Карта сценариев использования

Системный анализ:

- Карта внешнего окружения — схема взаимодействия между частями системы (Рисунок 5);
- Карта внутреннего окружения – схема взаимодействия внутренних частей системы (Рисунок 5);
- Анализ задействованных платформ и версий операционных систем;
- Разработка технических требований к самому приложению и сторонним сервисам;

- Требования к БД и CRM;
- Требования к протоколам интеграции;
- Инструментарий мобильной и серверной части;

Результат: полная картина технической составляющей проекта.

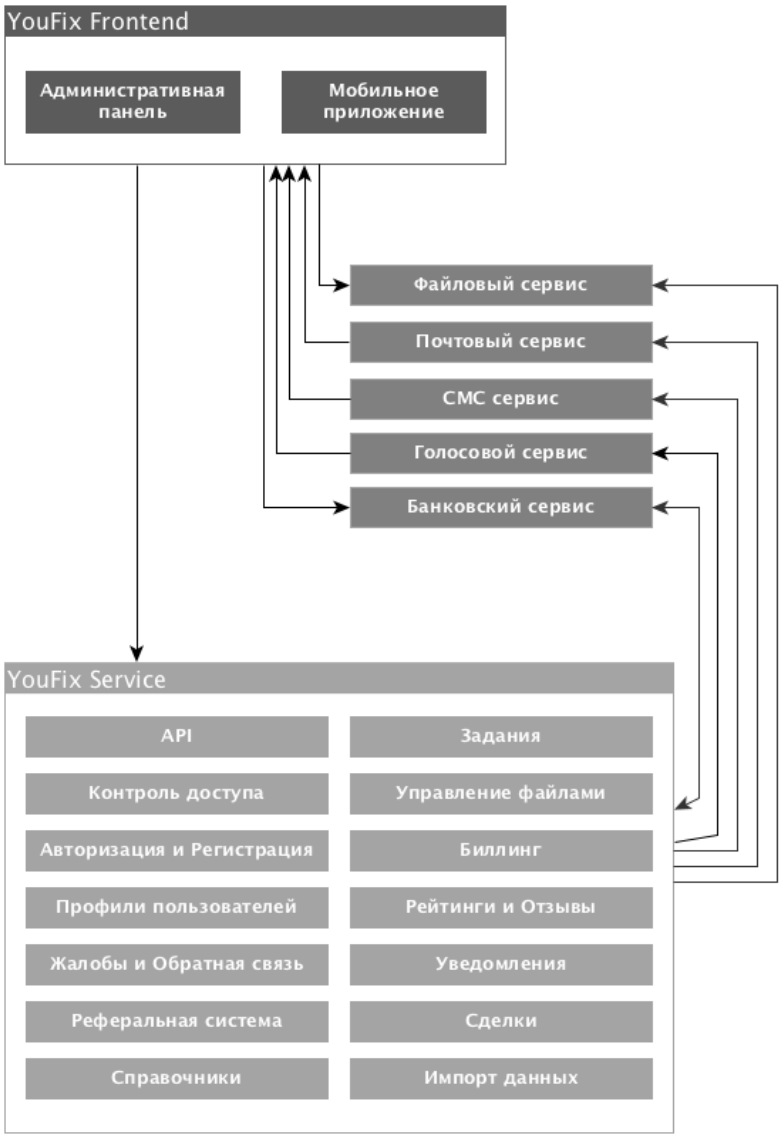

Рисунок 5 - Карта внешнего и внутреннего окружения

Проектирование интерфейсов:

- Проектирование основных экранов приложения (Рисунок 6);
- Разработка схемы взаимодействия со всеми переходами (Рисунок 7);
- Текстовое описание поведения экранов.

Результат: полная картина визуальной части приложения, взаимодействие экранов и их описание.

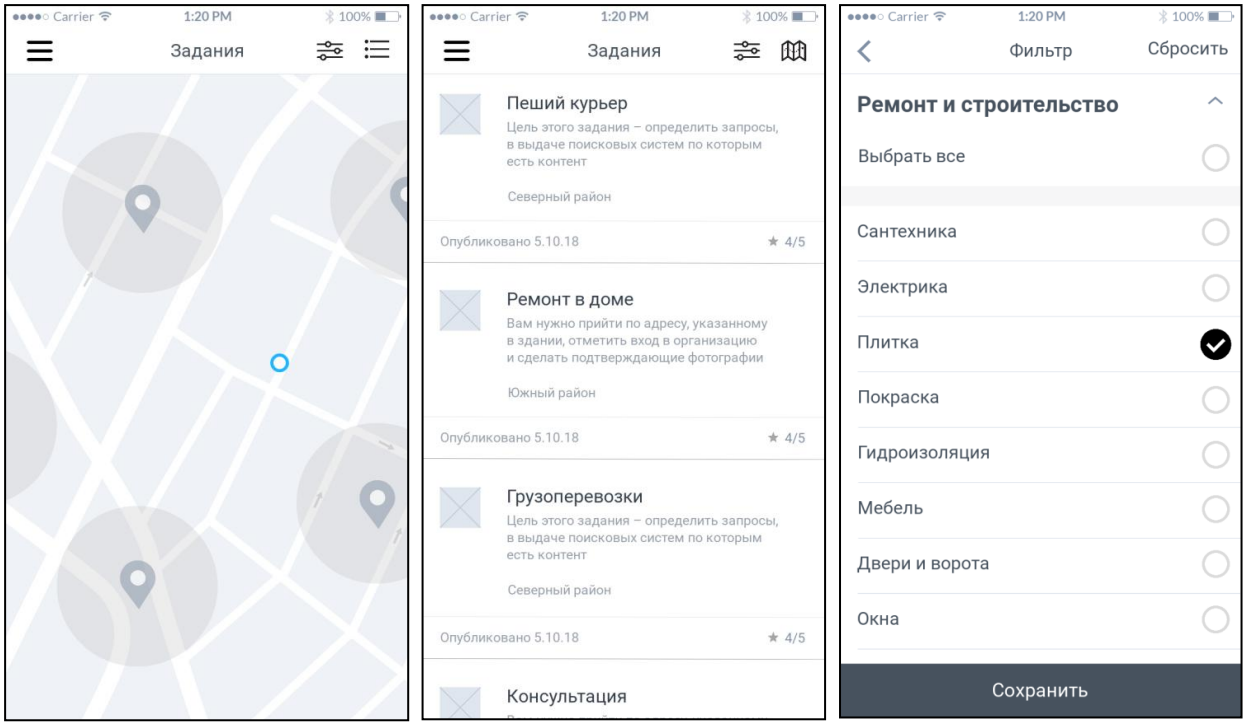

Рисунок 6 - Примеры проектирования интерфейсов

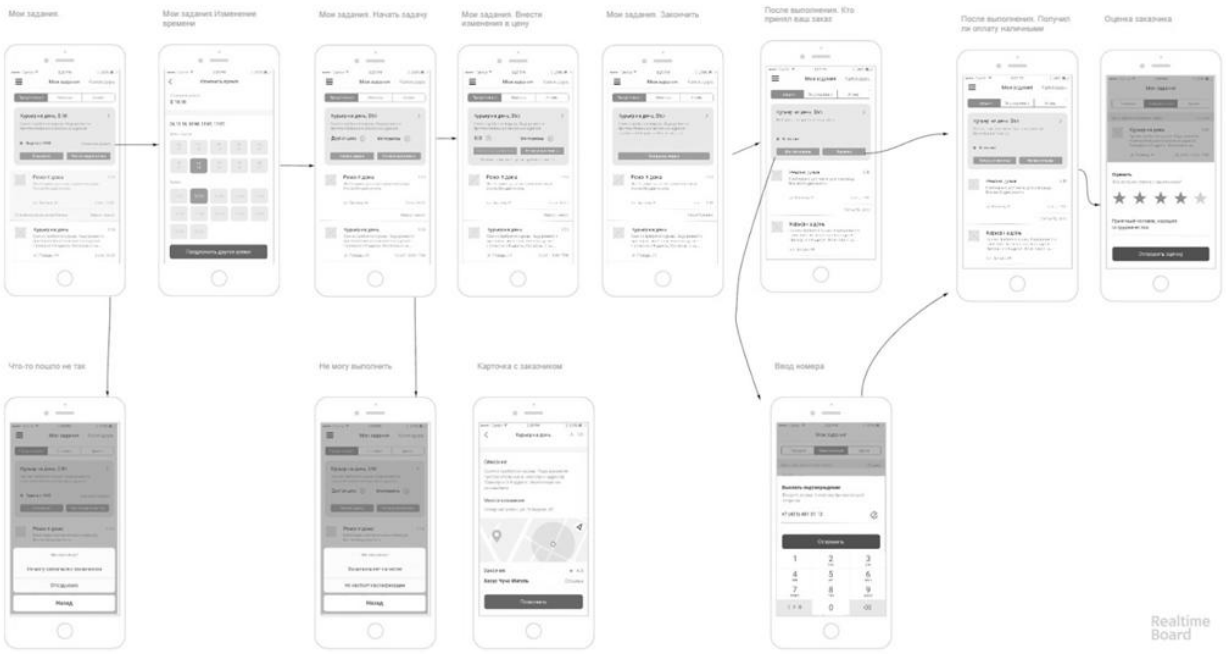

Рисунок 7 - Схема взаимодействия между экранами

2.2 Выбор основных фреимворков

Основная цель компании – разработка качественного продукта в минимально короткие сроки в соответствии с пожеланиями заказчика. В ходе написания ТЗ было установлено, что клиент заинтересован сделать продукт, востребованный на рынке, поэтому требования клиента довольно часто менялись.

В связи с этим было принято решение выбрать метод управления проектами Scrum. А систему оплаты Time&Material. Так, требования, прописанные в ТЗ, жестко не фиксируются, и клиент в любой момент может внести новую функцию или требования, а команда возьмет его в работу. При этом оплачивается фактически затраченное время на реализацию функций.

Разработка ведется итерационно. По окончанию каждого спринта заказчик может посмотреть сборку приложения на реальном мобильном устройстве и убедиться в работоспособности и правильности выполнения задач.

2.3 Инструмент для управления ходом работ

Система гибкой разработки и оплаты по факту предполагает доверие со стороны клиента, но также необходимо сделать процесс разработки прозрачным. Чтобы заказчик всегда мог посмотреть, что происходит в проекте и за что он платит.

В качестве инструмента для управления ходом работ был выбран redmine.

Redmine — это информационная система управления проектами с веб интерфейсом (онлайн), включающая в себя полный набор средств для совместной работы над проектами. Система позволяет вести одновременно несколько проектов, отслеживать их состояния, управлять шагами проекта, задачами, приоритетами, гибко назначать роли участникам (Рисунок 8).

Рисунок 8 - Доски проекта

Также в одном проекте можно создавать несколько досок. Для текущего проекта были разделены доски для backend, iOS, Android, design.

Преимущества системы Redmine и решение задач:

1. Организует единый центр ведения проектов, программ и портфелей проектов в компании с гибкими настройками ролей участников – один и тот же сотрудник может играть разные роли в разных проектах. Обеспечивается единый стандарт ведения проектов в организации.

2. Позволяет вовлечь участников проекта в процесс, обеспечить визуальное представление задач, сроков, вех проекта. Все знают, что делать дальше и видят цель.

3. Гибкая отчетность по проектам: кто, что и когда делал, делает и будет делать. Видимость загруженности ресурсов, контроль сроков, история задач. Автоматическое построение диаграммы Ганта и отображения задач на календарном плане.

4. Простота доступа к информации из любой точки, в том числе для географически удаленных сотрудников и подразделений. Возможность доступа заинтересованных лиц, спонсоров и других участников проекта, явно не связанных с выполнением задач, к информации и отчетности в режиме просмотра.

5. Система решает задачу социального взаимодействия в проектах, предоставляя встроенные проектные форумы (средства для обсуждений), доски новостей, базы знаний и возможность комментировать и обсуждать задачи.

6. Возможность настройки продукта на любую предметную область бизнеса, путем введения новых справочников, дополнительных полей к задачам, схем обработки последовательности задач.

7. Инструмент не только для проектного менеджера, но и всех участников проектной команды, предоставляет доступ к проекту всем всегда и везде, в том числе с мобильных устройств.

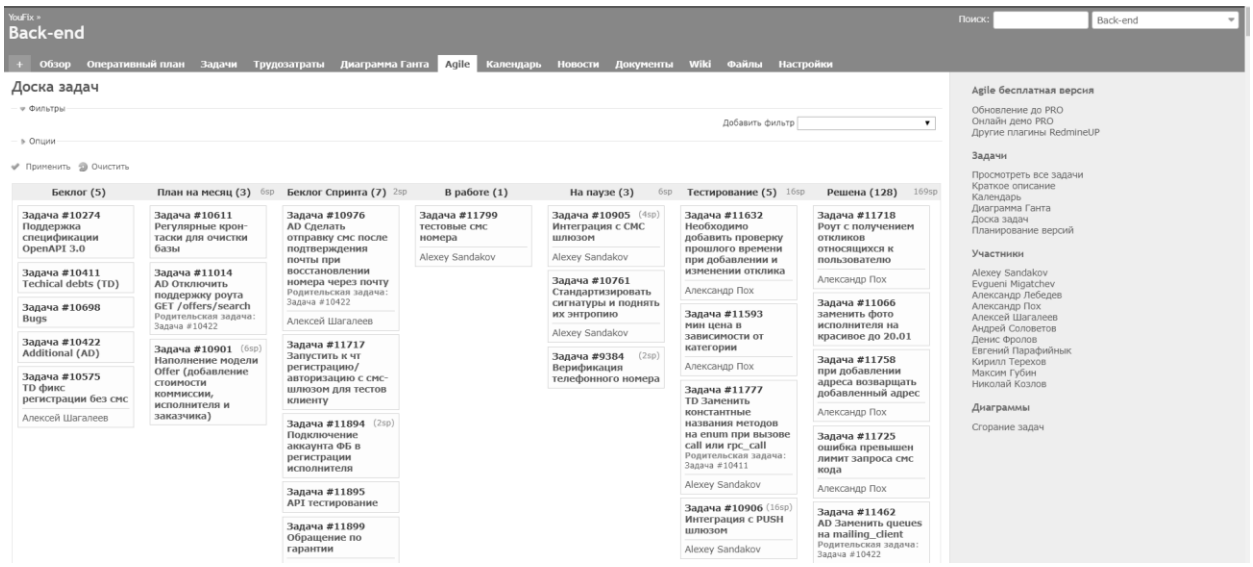

Рисунок 9 - Доска задач в redmine

Разработчики всегда поддерживают доску в актуальном состоянии (Рисунок 9). Так можно понять на какой стадии находится та или иная задача, отследить проблемные места. Все задачи переносятся в тестирование, QA тестирует и закрывает успешно реализованные, а задачи с ошибками возвращает в беклог спринта.

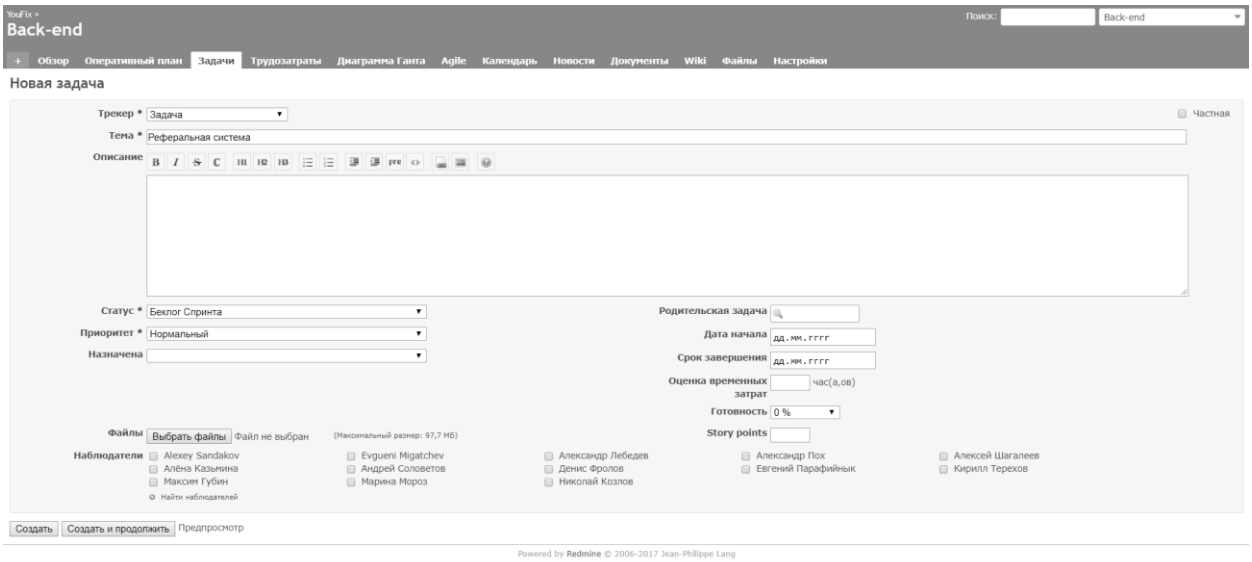

Рисунок 10 - Создание задачи в redmine

В самой задаче можно написать развернутое описание (Рисунок 10), выставить приоритет, назначить задачу на конкретного разработчика, указать дату

начала и завершения работы, story points и другие параметры. По ходу выполнения разработчики указывают трудозатраты и готовность задачи.

Также к проекту был выдан доступ клиенту. Это делает процесс разработки более прозрачным, и клиент всегда может посмотреть, какие задачи сейчас в работе и сколько на них по факту затрачено времени.

2.4 Инструменты дизайна

Выбор программы для выгрузки экранов был между zeplin и simply. В simply существует проблема некорректной передачи размеров экранов. В результате разработчикам приходится вручную адаптировать размеры, при этом экраны могут в итоге иметь разный вид. В связи с этим выбор был остановлен на zeplin.

Zeplin.io – это десктопное приложение, которое помогает дизайнерам интерфейсов и фронтенд разработчикам эффективно работать в команде, экономя свое время. Программа берет файл с дизайном, а затем генерирует руководство по стилю, исходники и спецификации, которые нужны разработчикам (Рисунок 11).

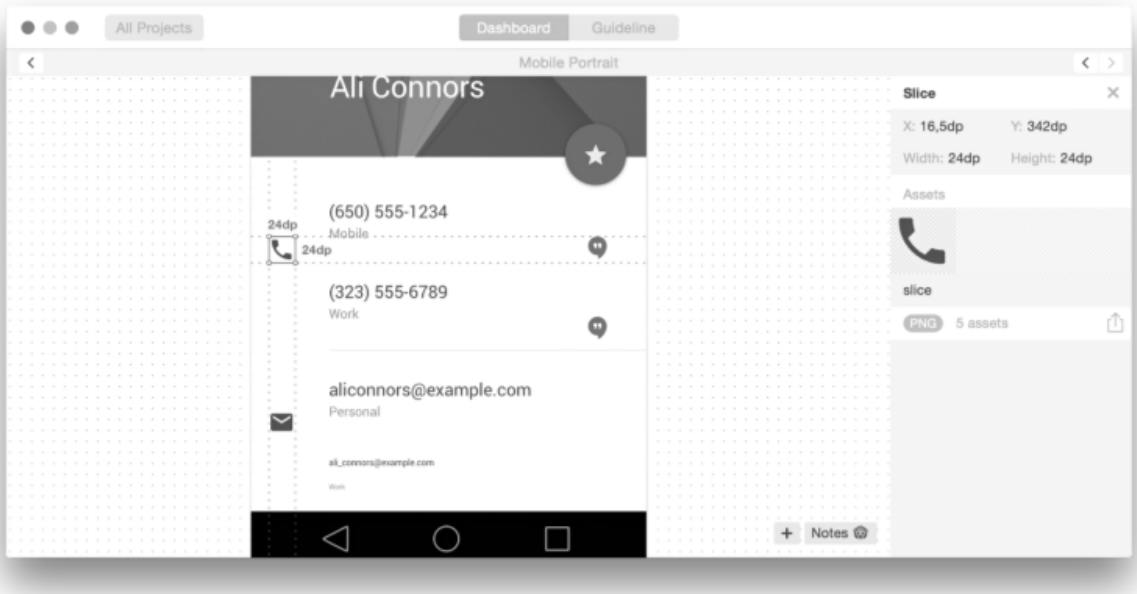

Рисунок 11 - Пример экрана в zeplin

2.5 Выбор архитектуры проекта

Для проекта была выбрана микросервисная архитектура. Так как, во-первых, она позволяет масштабировать отдельные микросервисы, на которых будет наибольшая нагрузка, а во-вторых, микросервисы позволяют работать над бекенд частью проекта нескольким разработчикам изолировано.

В качестве базы данных была выбрана PosrgreSQL . Это реляционная база данных, обладает высокой скоростью, низкой ценой, имеет возможность выстраивать связи между таблицами, интеграцию со сборщиками и автомиграции.

Также был выбран брокер запросов - AMQP-брокер, который осуществляет маршрутизацию, гарантирует доставку, распределение потоков данных, подписку на нужные типы сообщений.

Итоговый вариант архитектуры приведен на рисунке 12.

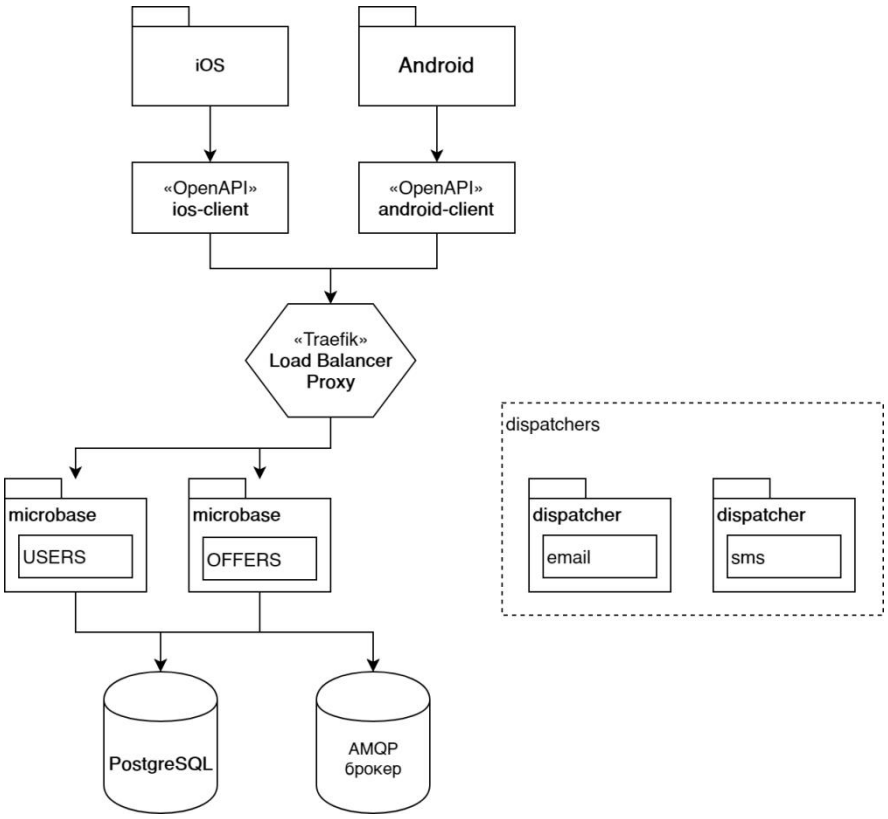

Рисунок 12 - Архитектура проекта

В результате можно сделать следующие выводы:

1. Написание технического задания – первый этап проекта. В нем прорабатываются требования, описывается логика работы приложения, проектируются прототипы, предлагается первичный вариант архитектуры проекта.

2. В связи с особенностям проекта было принято решение выбрать метод управления проектами Scrum. А систему оплаты Time&Material.

3. Для постановки задач и обеспечения прозрачности ходы выполнения проекта был выбран инструмент redmine.

4. В качестве программы для работы с дизайном экранов был выбран zeplin.io, так как это наиболее удобный инструмент для разработчиков.

5. Была определена итоговая архитектура проекта. Для проекта была выбрана микросервисная архитектура. В качестве базы данных была выбрана PosrgreSQL . Также был выбран брокер запросов - AMQP-брокер.

3 ПРОЦЕСС РАЗАРАБОТКИ

Выстраивание процессов разработки происходило опытным путем. Изначально в основу брались общепринятые процессы. По ходу развития проекта процессы адаптировались. Так, были проведены ряд мероприятий, которые позволили достичь большей эффективности.

3.1 Установление четкой последовательности разработки

В общем представлении последовательность разработки после написания ТЗ и прототипирования следующая: параллельно может идти дизайн и backend, с небольшой задержкой начинается разработка на IOS и андроид. backend выполняет задачи на пару недель раньше, чтобы мобильная разработка могла разрабатываться не завися от backend.

Основная проблема гибкого подхода разработки – требования от заказчика могут поменяться. Опытным путем было выяснено, что клиент склонен менять требования, когда он видит конечный дизайн экранов, а не прототипы.

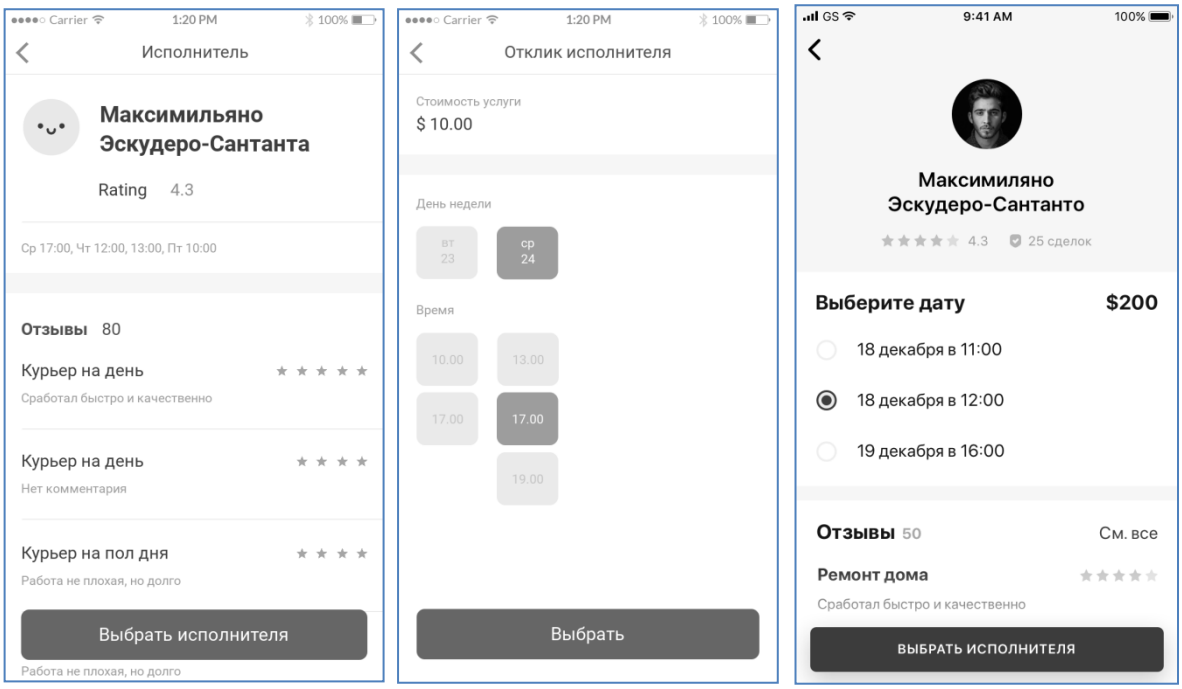

Рисунок 13 - Пример различия прототипов и конечного дизайна

На приведенном сравнении прототипов и конечного экрана видны различия (Рисунок 13). Во-первых, изначально предполагалось, что это будет два отдельных экрана. Во-вторых, добавилось новое поле – количество выполненных заданий.

Также поскольку backend-часть не видна клиенту напрямую, у него не складывается представление, что работа уже была выполнена. Однако команду демотивировало постоянное переделывание работы, которую они сделали по предоставленным первоначально требованиям.

Ситуацию позволило скорректировать жесткое фиксирование процесса работы над задачами: полностью утверждается дизайн экранов определенного блока, после этот блок может взять в работу backend, и только потом мобильная разработка.

Результат:

1. Клиент имеет конечное представление, как выглядит экран и визуально лучше представляет функциональность, поэтому требования утверждаются сразу.
2. Команда не тратит время на переработку функций, в результате возрастает скорость разработки.
3. У разработчиков всегда новые задачи, они не демотивированы переделыванием старых задач.

3.2 Выбор длины спринта

Классический Agile-подход предлагает зафиксировать длину спринта от одной до четырех недель. При старте разработке длина спринта была выбрана равной четыре недели.

Плюсы:

1. Если для задачи не хватает информации, либо не готов дизайн, ее можно поставить на паузу, в это время взять в работу другую задачу, а эту задачу завершить позже.
2. Планирование занимает меньше времени.
3. Оценка проходит более укрупнено, а у разработчика есть ответственность выполнить поставленный объем работ, поэтому скорость разработки может быть больше.
4. Поскольку сборка отправляется клиенту раз в четыре недели, команда нервничает меньше.

Недостатки:

1. Дизайн не успевал подготовить все экраны до планирования, поэтому планирование вынуждено проводилось по прототипам. В результате после получения дизайна могли поменяться требования, а задача могла уже быть сделана на backend.

2. Также без дизайна разработчики менее точно могли оценить время работы на задачу.

3. Из-за планирования задач блоками упускались детали, и в ходе разработки могло выясниться, что не хватает какого-либо маршрута или параметра в базе данных на backend или дизайн экрана некорректный, и нужны доработки. На это требовалось время, а задача была вынуждена быть на паузе. Наиболее негативно это сказывалось в конце спринта.

4. По человеческому фактору разработка велась менее продуктивно и расслаблено в первой части месяца, и в авральном режиме последнюю неделю. Это повлекло за собой некоторые неудачные решения, которые в последствие нуждались в переделке, а также переработку часов разработчиками, что негативно сказывалось на их дальнейшей продуктивности.

5. Получение большого количества обратной связи от клиента раз в месяц. В порядке вещей после реальной проверки приложения, заказчик мог дать пожелания об изменении логики той или иной функции. Поскольку мы работаем по гибкой разработке, это считается нормой и позитивным моментом, так как основной целью является сделать качественный продукт, который во всех аспектах удовлетворял бы заказчика. Но это приводит к ряду проблем. Во-первых, обратная связь от клиента была, когда уже был запланирован объем на следующий спринт, то есть команда должна была сделать эти задачи перерабатывая, либо по scrum требуется замена какой-то задачи на спринт, что, безусловно,

расстраивает клиента. Во-вторых, клиент получает исправленный вариант задачи только со следующей сборкой – через месяц.

В результате было принято решения перейти на недельные спринты.

Преимущества:

1. Полностью готов и утвержден дизайн на задачи.
2. Детальное планирование функционала каждого экрана.
3. Получение обратной связи от клиента раз в неделю по небольшому объему задач. Небольшие корректировки брались в работу сразу, и уже через неделю клиент видел результат. Более значительные изменения за неделю прорабатывались, и планировались на следующий спринт. Так, через две недели заказчик полностью получал все свои пожелания в адекватный срок.
4. Постоянная выдача сборок не дает расслабляться команде, поэтому работа идет равномерно в течение недели.
5. Разработчики полностью представляют свой объем работ и могут для себя планировать время.

Недостатки:

1. Сокращение времени фактической разработки и, как следствие, сокращение скорости. Так как задача клиенту должна была быть выдана протестированная, без критичных ошибок, то сборки по возможности выдавались на тестирование на 4-й день, а финальные тесты проводились в 5-й день. Так, время фактической разработки было 3 дня, и 2 дня – исправление ошибок и рефакторинг.
2. Если в ходе разработки выяснялось, что для завершения задачи чего-то не хватает, то задача в экстренном порядке проходила все стадии

(дизайн - backend – мобильная разработка), это держало команду в напряжении, а мобильные разработчики и тестировщики, как финальная стадия, зачастую вынуждены были перерабатывать в последний день.

В результате было принято решение попробовать вариант с двухнедельными спринтами.

Преимущества:

1. Оптимальное время разработки. Для более равномерного тестирования функционала, было введено внутренне тестирование после первой недели с частью функционала. Итоговая сборка на тесты выдается за два дня до показа клиенту, если нужны значительные правки, то задействуются оба дня, если на тестировании нет критичных ошибок, то у разработчиков остается время на рефакторинг или они могут взять новые задачи в работу.
2. Объем работ известен, и разработчики могут его распределять заранее.
3. В случае нехватки информации для завершения задачи, есть достаточное количество времени, чтобы задача прошла по всем стадиям от дизайна до мобильной разработки.

Недостатки:

1. Разработка в первые дни спринта может идти несколько медленнее.

Результат: в качестве оптимального и наиболее удобного варианта команда выбрала для себя спринты равные двум неделям (Таблица 1.1).

Таблица 1.1 – Сравнение спринтов разной длины

Параметры	Продолжительность спринта		
	Месяц	Неделя	2 недели
Заменяемость задач	При недостаточной информации можно поставить задачу на паузу	Критично, объем задач небольшой	Оптимально. Задача успевает пройти по всем стадиям, но долго не задерживается на паузе из-за дедлайнов.
Планирование	в общей сумме занимает меньше времени	Проходит быстрее, но в сумме занимает больше времени	Из-за детализации занимает больше времени. Под конец планирования снижается эффективность команды. В целом занимает больше времени
Оценка	Укрупненно из-за отсутствия дизайна - не всегда достоверно	Точная	Точная
Дизайн	Готов не весь дизайн, требование могло поменяться, но уже сделано на бекенде	Полностью утвержден	Утвержден на все основные экраны, состояния экранов успевают доработаться.
Детализация задач	Из-за отсутствия дизайна невозможно полностью спроектировать задачу, упускались детали, которые могли обнаружится в конце спринта	Детализация каждой задачи	Детализация каждой задачи
Равномерность разработки	Менее продуктивно в начале спринта, с переработками в конце	Мобильная разработка и тестировщики часто перерабатывали в последние дни	Может незначительно быть ниже в первые дни, но компенсируется в последние
Обратная связь от клиента	Большой объем обратной связи, замена задач на запланированный спринт, получение правок только через месяц	Небольшой объем, получение всех пожеланий через две недели без замены задач	Нормальный объем, получение небольших пожеланий через 2 недели, а которые требуют проработки через 1 спринт.
Скорость разработки	Выше	сокращение фактического времени разработки из-за необходимости тестирования	Оптимальное, введено внутренне тестирование в середине для равномерности
Команда	Меньше нервничает	Может распланировать задачи для себя	Может распланировать задачи для себя. Меньше нервничает.

3.3 Сдвиг планирования

Классический подход в Scrum предполагает проведение планирования в первый день спринта (Рисунок 14). Однако в ходе разработки часто встречалась ситуация, что задача не готова для работы для мобильной разработки – некорректный дизайн, не хватает состояния экрана, нет маршрутов или полей в БД на backend. Эта ситуация была описана при выборе длины спринта.

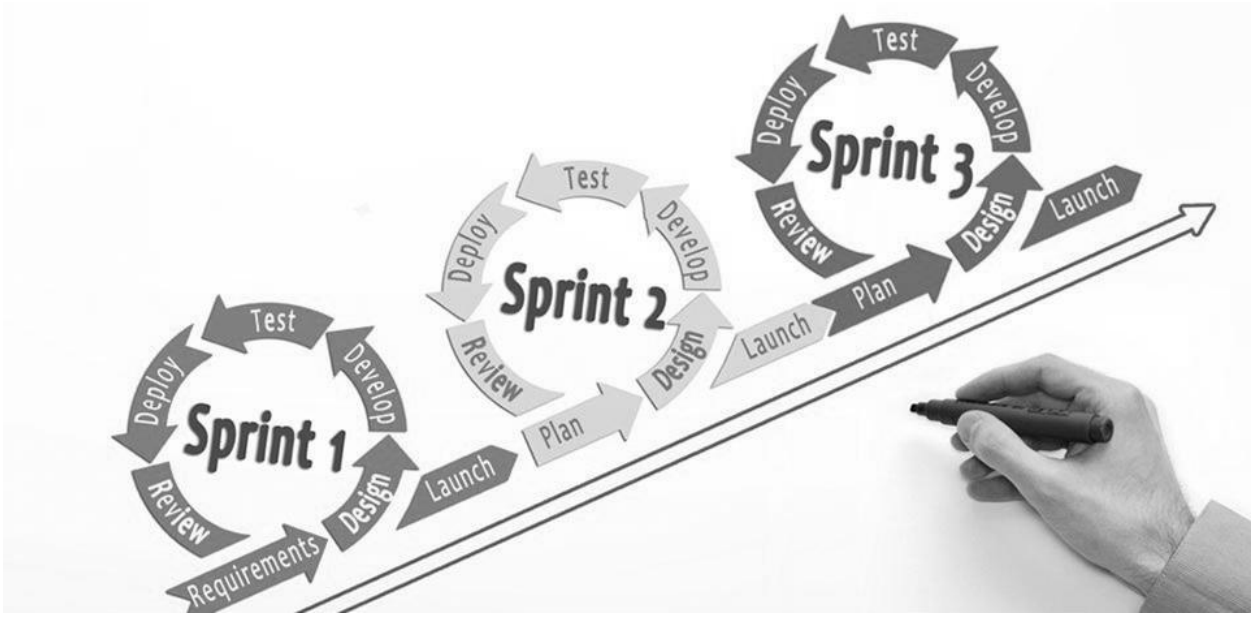

Рисунок 14 - Классическое время планирования в Scrum

Чтобы максимально минимизировать такие случаи, планирование было сдвинуто на неделю раньше (Рисунок 15).

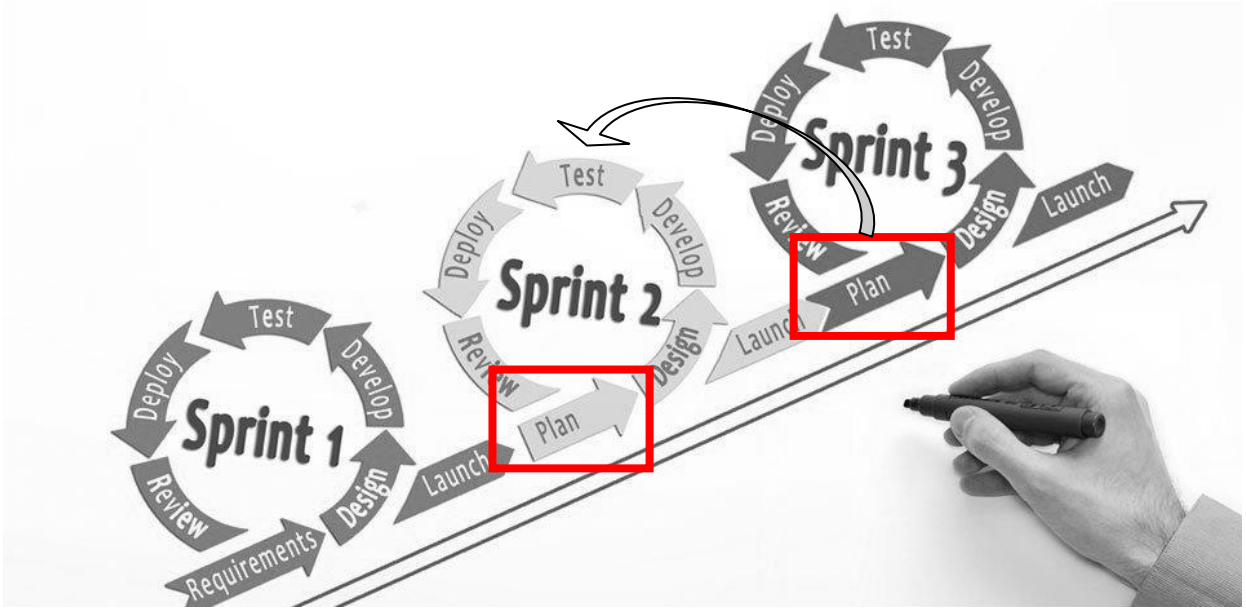

Рисунок 15 - Сдвиг планирования

Так, в ходе планирования выяснялись пробелы в задаче, и в течение недели дизайнеры успевали по необходимости переработать экран (переработки обязательно проходили согласование с клиентом), доработать состояния, адаптировать недостающие экраны под андроид. Backend, в свою очередь, реализовывал необходимый функционал, а также, немаловажная часть, проводил слияние данных и выкладывал работоспособный код на сервер, а также проводил изменение в спецификациях.

Результат: таким образом, задача полностью готова к реализации для мобильной разработки ко времени начала спринта.

3.4 Документация

3.4.1 Необходимость документации

Важной частью любого проекта, является документирование. Обязательной частью является написание технической документации – спецификации, с постоянной ее актуализацией и дополнением. Такая документация велась сразу.

Документация необходима для:

1. Документация обеспечивает «общее пространство» проекта. Любой участник в любой момент времени может получить необходимую информацию как по конкретной задаче, так и по общему направлению работы.

2. Команда говорит «на одном языке» — ведь гораздо проще понять человека, сообщающего «об ошибке в функции, описанной в Use Case №12», чем «о трех экранах, которые Вася месяц назад делал».

3. Документирование позволяет четко разграничить зоны ответственности между участниками проекта.

4. Только тщательно описанные требования могут быть проверены на полноту и непротиворечивость.

Был выработан набор базовых правил, которые позволяют эффективно документировать, планировать и контролировать разработку в комфортных для всех участников условиях.

1. Документация не должна быть избыточной и объемной. Избыточное количество текста – раздражает и затрудняет восприятие.

2. Вся схема документирования проекта должна быть взаимоувязанной и логичной. Если в схеме существует документ, который не связан ссылкой с каким бы то ни было другим документом, то его можно безболезненно из схемы исключить.

3. Вся оценка трудозатрат должна производиться только на основании описанных задач. Чем мельче оцениваемый элемент – тем точнее будет агрегированная оценка.

4. Всегда необходимо формировать списки оповещения заинтересованных участников. Это позволяет сохранить общую цель у команды и минимизировать демотивацию разработчиков [12].

3.4.2 Документирование задачи в тикете в redmine.

Для корректного и полного выполнения задачи необходимо четкое описание. Так как инструментом для работы была выбрана система redmine, все задачи ставятся и описываются там.

Тикет – это поставленная в системе управления (в нашем случае redmine) задача.

В начале проекта, поскольку планирование проводилось укрупненными блоками, задачи ставились соответствующим образом (Рисунок 16). Такая постановка задач привела к разному выполнению на iOS и андроид. Также в ходе могли разработки могли быть изменения, которые не всегда фиксировались в задачах. Поскольку дизайн отрисовывается для экранов на iOS, а для андроид они адаптировались после согласования с клиентом, периодически для андроид не были своевременно адаптированы экраны, либо в них не были внесены изменения.

В результате тестировщик не мог определить правильность реализации функции, либо считал ошибкой то, что было согласовано, но не задокументировано.

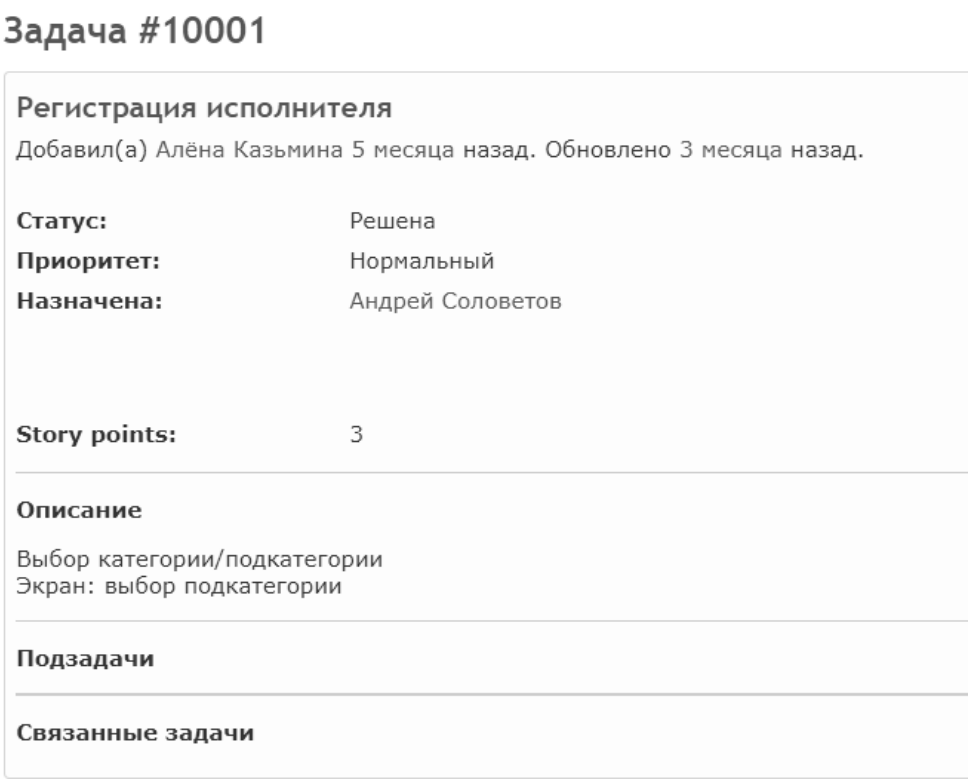

Рисунок 16 - Пример изначального тикета.

После получения обратной связи от разработчиков, было принято решение фиксировать максимально полное описание в тикетах.

В качестве дополнительной проверки полноты всех экранов, ссылки на экраны в zeplin прикреплялись к тикету. Если на экране появлялись незначительные корректировки, то они отмечались комментариями в zeplin (Рисунок 17).

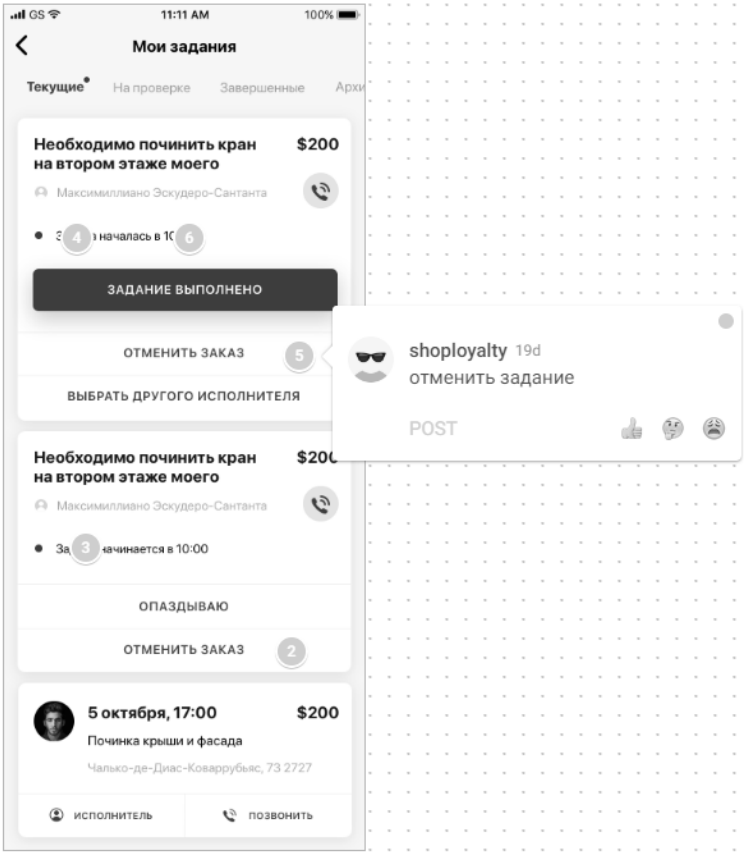

Рисунок 17 - Пример комментария в zeplin

При возникновении возможной вариативной реализации задачи, обговаривались все варианты, выбирался наилучший и обязательно заносился в тикет для обеих платформ (Рисунок 18).

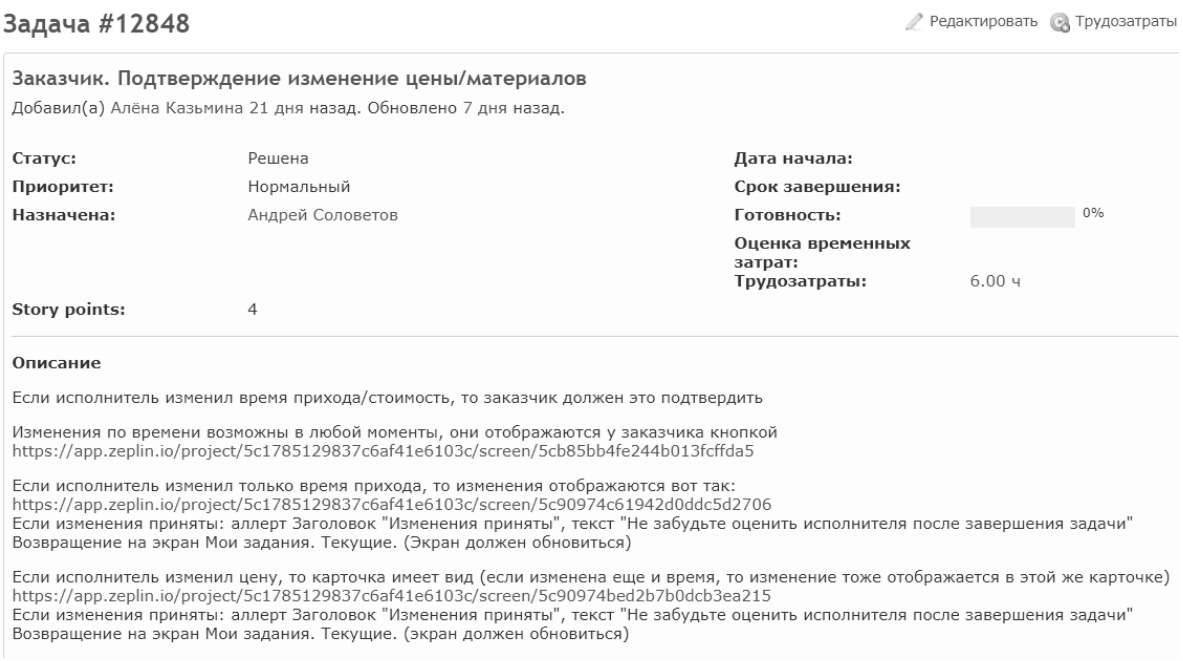

Рисунок 18 - Тикет с зафиксированной информацией

Результат: такая система позволила создать одинаковое понимание задач для всех участников процесса и существенно облегчила процесс контроля корректного выполнения.

3.4.3 Расположение User Flow на онлайн-доске Miro

Одним из наиболее эффективных инструментов является доска Miro. Основное преимущество доски – онлайн – формат. Так, каждый участник может видеть всегда актуальную информацию и все изменения.

На ней расположена карта экранов, из которой выстраивается User Flow.

User flows (пользовательские сценарии) - это наглядные материалы, которые иллюстрируют весь путь пользователя в продукте целиком.

В основе User Flow , как правило, лежит порядок действий, которые должен выполнить пользователь. User Flow помогает понять, все ли процессы в

продукте имеют логическое завершение. Глядя на него, команда может сразу понять, в чем суть решения, которое предлагает продукт.

User flow могут быть нелинейными — в них есть точки принятия решений, пути, режимы и петли, при помощи которых мы иллюстрируем все возможные взаимодействия с продуктом.

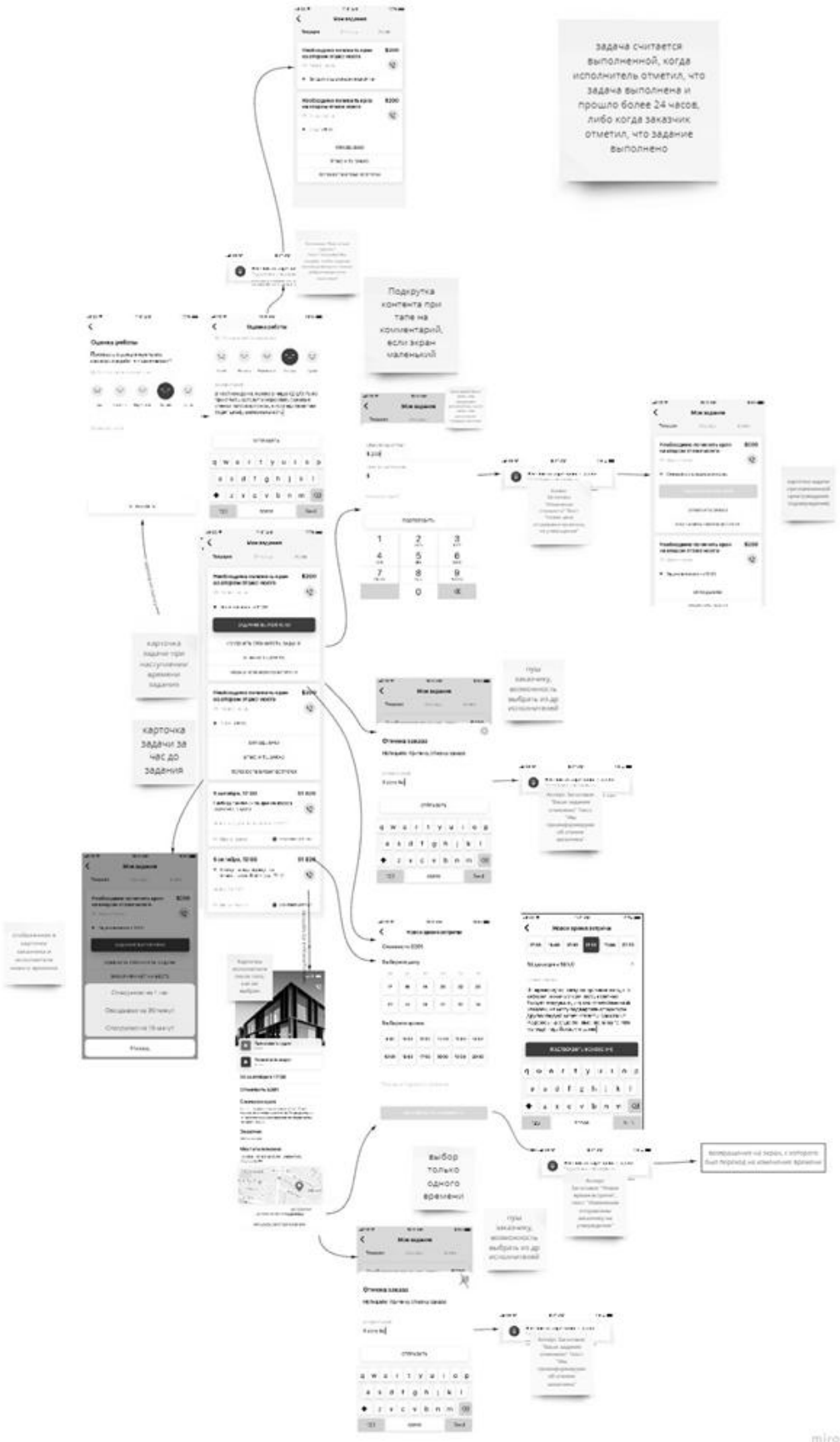

Рисунок 19 - Первоначальный вид User Flow (фрагмент)

Как показала практика, такой формат user flow оказался не вполне удобным. Во-первых, «красивые стрелки по дуге» занимали больше пространства, на карте терялась четкость, в результате терялись некоторые важные моменты, к примеру, не всегда было понятно на какой экран возвращаться после совершения того или иного действия. Во-вторых, не были зафиксированы особенности экранов (Рисунок 19).

В итоге user flow принял более структурированный вид – стрелки заменены на квадратную форму, экраны выстроены преимущественно по законам построения схем (вход слева и сверху, выход справа и снизу), с доски убраны состояния экранов (они отмечены в zeplin), отмечены неочевидные переходы по экранам и необходимые уведомления. А также все ключевые моменты были зафиксированы на зеленых стикерах (Рисунок 20).

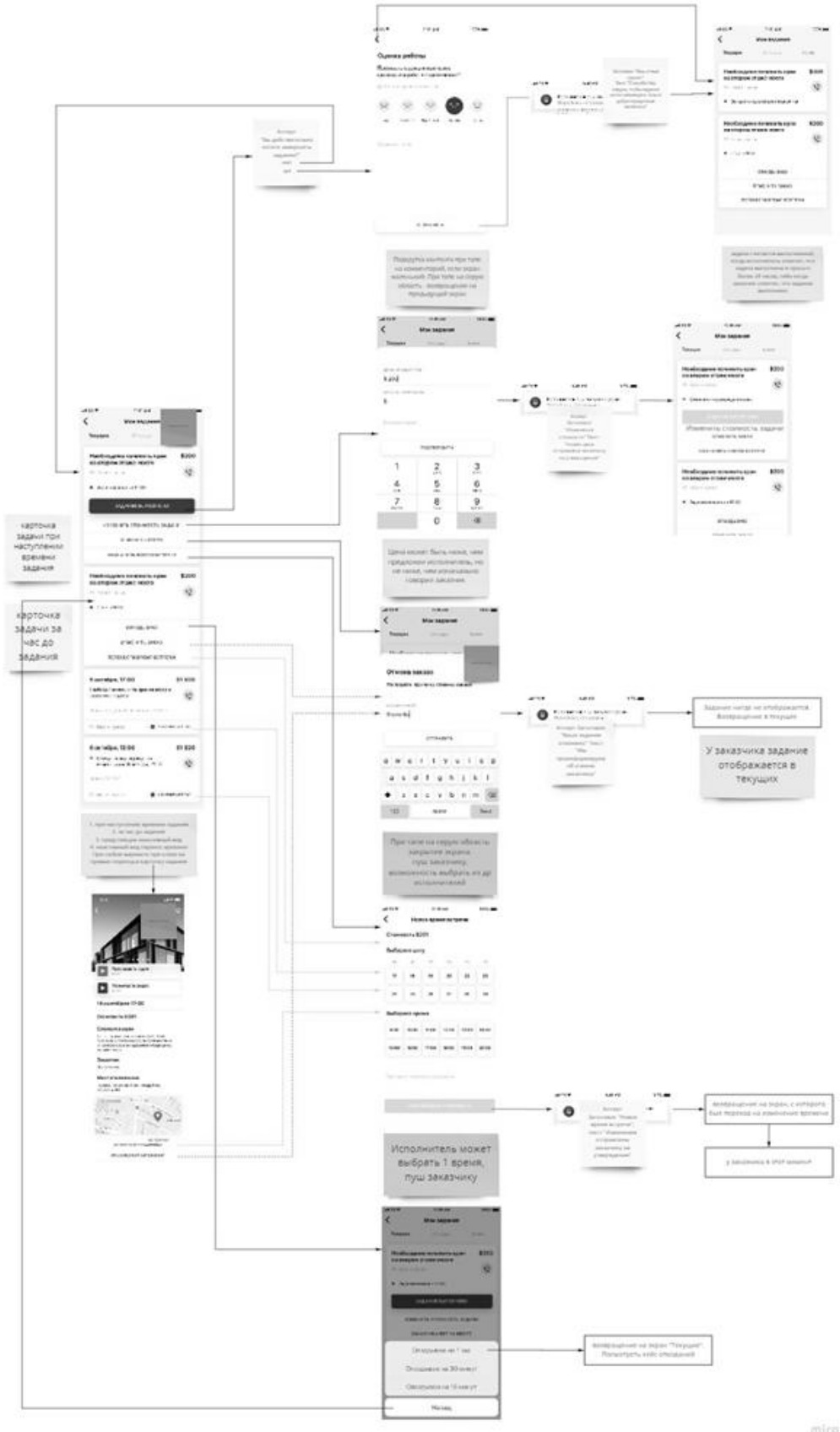

Рисунок 20 – Вид фрагмента User Flow после структурирования

Результат: этот инструмент также позволяет создать единое понимание функциональности задач.

3.4.4 Документирование особенностей поведения

Приложение едино как для заказчиков, так и для исполнителей. Часть функционала у заказчика/исполнителя зависима от поведения контрагента. К примеру, при отказе от выполнения задачи исполнителем, который уже выбран, задание у данного исполнителя скрывается из всех разделов, оно снова становится доступно для отклика другим исполнителям, а у заказчика принимает вид «ожидания выбора исполнителя».

Вариативности отражения возникло довольно большое количество, что привело к необходимости документации.

В качестве инструмента был выбран Google - документ – таблица Excel (Рисунок 21). Каждый член команды может видеть актуальный утвержденный вариант, а также вносить корректировки и дополнения. Такой вариант возможен, когда есть полное доверие в команде.

	A	B	C	D	E	F	G	H	I	J	K
1		Действие		Отображение							Текущий статус (результат тестирования)
2				Заказчик		Исполнитель					
3		Заказчик	Исполнитель	Текущие	Завершенные	Текущие	Отклики	Задания (общий список у исп)	Архив	Общий список у всех	
4	задача размещена, откликов нет	удалил задачу	-	удаляется		-	-	удаляется		удаляется	
5	оставлен отклик исполнителем	удалил задачу	-	удаляется		-	удаляется	удаляется		удаляется	
6	выбран исполнитель	удалил задачу	-	удаляется		удаляется	-	удаляется		удаляется	
7	выбран исполнитель	смена исполнителя	-	отклик меняемого исполнителя не показывается		удаляется	удаляется (статус cancel)	удаляется		доступен для остальных исполнителей	
8	выбран исполнитель	-	отказывается от выполнения	отображаются отклики др исполнителей (никто не назначен) статус selection		удаляется	удаляется cancel	удаляется		Доступно др исполнителем	
9											
10											
11											
12	**Завершение задачи**										
13		завершил задачу	-	переходит в завершенные	отображается	переходит в архив			отображается здесь		
14		-	завершил задачу	ожидает подтверждения 24 часа		меняется вид карточки, ожидает подтверждения заказчика 24ч					https://redmine.gitlab.itnap.ru/issues/12978
15		подтвердил в течении 24 часов	завершил задачу	переходит в завершенные	отображается	переходит в архив			отображается здесь		https://redmine.gitlab.itnap.ru/issues/12978
16		не подтвердил в течении 24 часов	завершил задачу	переходит в завершенные	отображается	переходит в архив			отображается здесь		https://redmine.gitlab.itnap.ru/issues/12978
17											Активный вид, исполнитель, изменить время. Перенес в

Рисунок 21 - Пример документации в Google - документах (Excel)

3.4.5 Другая необходимая документация

Помимо документации, которая используется всей командой, существует документация для клиента. К ней относится:

1. Смета оплат: в данной таблице указывается количество специалистов, занятых в месяце, планируемое количество часов, фактически отработанное количество часов, ставки и итоговые значения.
2. Запланированные функции на спринт: в документе расписываются функции и примерная оценка в часах по ним.
3. Закрытие функций за спринт: в документе указываются реализованные функции, фактически затраченное на них время.
4. Таблица релизов: в документе описывается функционал сделанный за спринт, особенности функций, на которые стоит обратить внимание клиенту, дата отправки клиенту, багги, найденные клиентом, причина показа сборки позже срока, если вдруг был перенос.
5. Product road map – карта развития продукта. Составляется для понимания загрузки разработчиков и своевременной проработки функций.

3.5 Оценка в Story point

Story points – это сравнительная оценка, которая показывает «размер» требований относительно друг друга. Т.е. важны относительные значения и не может быть эталона. «Пункты» не имеют физических единиц измерения.

Преимущества:

1. По исследованиям люди не достоверно оценивают предполагаемые трудозатраты в часах.
2. Разработчики оценивают непосредственно сложность задачи, не думая, что другой разработчик может делать задачу медленнее.
3. Замеряется общая скорость команды (количество Story point на всю команду), это направляет разработчиков на командную работу.
4. По затраченному времени нельзя определить количество реализованных задач, в то время как по Story point – это основной показатель.

3.6 Проведение ретроспективы

Для улучшения процессов внутри команды примерно раз в месяц устраиваются ретроспективы.

Проведение ретроспектив – это активность, которую каждая agile-команда проводит для того, чтобы решать свои проблемы. Ретроспектива - это регулярная встреча, на которой команда обсуждает свой рабочий процесс и что-то в нем меняет.

Основные вопросы ретроспективы:

1. Плюсы. Что шло хорошо в прошлой итерации?
2. Минусы. Какие проблемы были в прошлой итерации?
3. Идеи. Какие идеи появились по ходу ретроспективы?
4. План. Какие улучшения мы запланируем на следующую итерацию?

По результатам ретроспективы формируется четкий план действий для улучшения процессов.

Ретроспектива может проводиться в разном формате. Один из проведенных вариантов представлен на рисунке 22.

Рисунок 22 - Проведение ретроспективы

Так, по результатам ретроспективы совместно с командой принималось решение о длине спринта, отмечались положительные результаты после переноса планирования на более раннее время, четкой постановки тикетов, документирования необходимой информации.

Также определяется общий настрой команды – позитивный ли он, можно узнать моменты, на которые стоит обратить внимание и понять конкретные переживания каждого разработчика.

Помимо определения возможных улучшений, проведение подобных мероприятий способствует сплочению команды, появляется эффект синергии. Важна не только нацеленность на результат, но и дружественная атмосфера внутри коллектива.

В ходе получении практического опыта и тестирования различных сценариев разработки проекта были предприняты следующие шаги:

1) Зафиксирована четкая последовательность разработки: дизайн- backend – mobile.
2) Выбрана продолжительность спринта в две недели.
3) Планирование перенесено на неделю раньше начала спринта.
4) В обязательном порядке документируются и постоянно актуализируются наиболее важные аспекты приложения.
5) Ежемесячно собирается обратная связь с команды на ретроспективе.

ЗАКЛЮЧЕНИЕ

Для разработки проекта используются успешные методологии, удобные и современные инструменты. Но не далеко не все методологии и инструменты подходят, некоторые могут нуждаться в модернизации и адаптации под конкретный проект.

В ходе выполнения работы были выполнены задачи:

- Рассмотрены и выбраны инструменты для управления проектом.
- Проведен анализ процесса разработки.
- Установить наиболее эффективные решения.
- Построены оптимальные процессы управления разработкой данного проекта

Также были протестированы различные гипотезы и предприняты шаги, направленные на достижение результатов:

1. Скорость разработки команды возросла практически в 2 раза с 42 до 64 Story point.
2. Оперативное внедрение пожеланий от клиента в сборки, как следствие увеличение лояльности со стороны клиента.
3. Создание единого понимания функционала приложения у всех участников процесса разработки.
4. Минимизирование изменений в реализованный функционал, благодаря четко выстроенным процессам.
5. Высокая замотивированность и нацеленность на результат разработчиков.

В результате была достигнута цель – процесс разработки показывает свою эффективность и не допускает основные риски проекта.

Проект сейчас находится на стадии доработки второстепенных функций. Предполагаемая дата релиза июль 2019 года.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Борута, Я.Б. Agile или WaterFall. Worksection – 13.06.2017 // https://worksection.com/blog/waterfall-vs-agile.html

2 Гибкая методология разработки Scrum – 04.01.2015 – Habr – https://habr.com/ru/post/247319/

3 Schwaber , K. The Scrum Guide. The definitive Guide to Scrum: The Rules of the Game. / - K. Schwaber, J. Sutherland

4 Трусь, А.А. Психология управления: учебное пособие. / А.А.Трусь. – М.: Элитариум, 2015 – 423 с.

5 Sutherland, J. How a Traditional Project Manager Transforms to Scrum: PMBOK vs. Scrum. / J. Sutherland, N. Ahmad

6 Зыкова, С. Agile, Scrum, Kanban: в чем разница и для чего использовать: статья – 09.10.2017 – https://rb.ru/story/agile-scrum-kanban/

7 Использование микросервисной архитектуры. Hawk House Integration – http://hawkhouse.ru/blog/kogda-opravdano-ispolzovanie-mikroservisnoj-arhitektury/

8 Друзьягин, Р. Чем PostgreSQL лучше других SQL баз данных с открытым исходным кодом: статья – 29.04.2016 – Habr – https://habr.com/ru/post/282764/

9 Введение в MongoDB: учебное пособие – https://metanit.com/nosql/mongodb/1.1.php

10 Понимание AMPQ: учебные материалы. Spring – http://spring-projects.ru/understanding/amqp/

11 Что такое техническое задание: статья. – 26.04.2016 – Habr – https://habr.com/ru/post/300420/

12 Документирование в разработке ПО: статья. Анализ и проектирование систем. – 14.03.2014 – Habr – https://habr.com/ru/post/215837/