Санкт-Петербургский государственный университет

Ференц Данила Святославович

Выпускная квалификационная работа

Проектирование и реализация сервиса «Мой автомобиль» в мобильном банке

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование информационных систем»*

Основная образовательная программа *СВ.5006.2017 «Математическое обеспечение и администрирование информационных систем»*

Системное программирование

Научный руководитель:
доцент кафедры СП, к. т. н. Ю. В. Литвинов

Консультант:
ведущий разработчик ООО «Тинькофф Центр Разработки» А. Ю. Романов

Рецензент:
разработчик АО «Тинькофф Банк» К. А. Шумаков

Санкт-Петербург
2021

Saint Petersburg State University

Ferents Danila

Bachelor's Thesis

Design and implementation of «My Car» service in bank mobile application

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5006.2017 "Software and Administration of Information Systems"*

Profile: *System Programming*

Scientific supervisor:
candidate of Engineering Sciences, docent Yurii Litvinov

Adviser:
lead software engineer, Tinkoff Development Center Alexandr Romanov

Reviewer:
software engineer, Tinkoff Bank Shumakov Kirill

Saint Petersburg
2021

Оглавление

Введение

В последнее время стремительно растёт процент, который занимают смартфоны в мировом трафике интернета. На 2019 год он составлял 53% и уже на тот момент обгонял долю персональных компьютеров. В России ситуация похожа на мировую, ведь, согласно аналитической платформе gfk [1], уже к началу 2019 года 61% населения России пользовались Интернетом с мобильных устройств. Компании также понимают данную тенденцию, и сейчас уже редко встретишь продукт крупной компании, не имеющий, помимо интернет сайта, также и мобильного приложения.

Тем временем, согласно федеральной службе государственной статистики, в России постоянно растёт количество автомобилей на душу населения [2]. В 2020 году на 1000 россиян приходилось 309 машин [3], что почти на 50% выше, чем аналогичный показатель 10 лет назад. Учитывая степень автомобилизации населения, с каждым годом данная цифра будет лишь расти.

Стоит отметить, что по закону Российской Федерации автомобиль обязательно должен быть застрахован, владение им облагается налогом. Автомобилисту необходимо обслуживать свой автомобиль на станциях технического обслуживания, оплачивать при возникновении штрафы. Существует много отдельных приложений, позволяющих отслеживать налоги на автомобиль, страховать его, сообщать о дорожно-транспортных происшествиях. Однако было бы намного удобнее объединить все данные возможности и создать единый сервис, доступный на смартфоне, который всегда под рукой. Сервис, в котором можно добавить свой автомобиль, забронировать визит на станцию технического обслуживания, оплатить штрафы, налоги, купить страховку, посмотреть уже оформленные ранее страховые полисы и многое другое. И при этом можно отследить все затраты на автомобиль, что облегчает планирование бюджета.

Данный курс рынка заметила компания Тинькофф, которая яв-

ляется первопроходцем в России в области суперапповв[1] — приложений, отличающихся наличием собственной экосистемы, объединяющих несколько областей, таких как финансы, лайфстайл[2], досуг и другие. То есть приложение является порталом в другие сервисы и приложения, а его главная задача — как можно дольше удерживать пользователей.

В рамках данной работы компанией была поставлена задача расширить суперапп и реализовать в мобильном банке сервис «Мой Автомобиль», собирающий в единое целое и покрывающий большинство потребностей автомобилиста, тем самым значительно упрощая клиенту процесс владения автомобилем. В данной работе пойдёт речь о всех стадиях разработки данного сервиса. Начиная от анализа требований, заканчивая тестированием и релизом.

[1]от английского слова super app

[2]от англ. Lifestyle — образ жизни

1. Постановка задачи

Целью работы является проектирование и реализация сервиса «Мой Автомобиль» в мобильном банке на платформе iOS. Для выполнения поставленной цели были выделены следующие задачи:

1) произвести обзор существующих решений;
2) выявить функциональные и нефункциональные требования к сервису;
3) спроектировать сервис и его интеграцию в мобильном банке;
4) реализовать сервис на платформе iOS;
5) покрыть сервис автоматическими тестами.

2. Обзор

2.1. Обзор существующих решений

На данный момент существует много приложений, удовлетворяющих пользователю одну из нужд, возникающих в процессе владения автомобилем. Их все можно поделить на категории и их функциональность ограничена.

2.1.1. Мобильные приложения страховых компаний

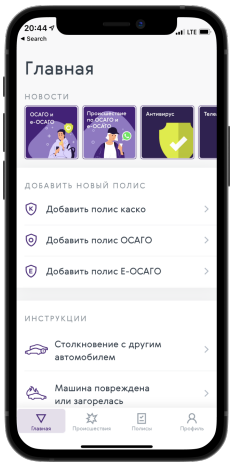

Рис. 1: Ренессанс. Авто [4]

Первыми, кто посмотрел в сторону собственных мобильных приложений, были страховые компании, которым необходимо было разгружать свой колл-центр и офисы продаж. Сейчас можно онлайн оформить страховку и заявить о страховом случае, сфотографировав ДТП и загрузив фотографии прямо в приложение, что намного удобнее, чем личный визит в офис.

Однако из-за того, что такие приложения создавались для предоставления удобного сервиса клиентам страховой компании, то их функциональность этим и ограничивается.

Самыми крупными примерами таких приложений являются «Ренессанс. Авто» (Рис. 1) и «АльфаСтрахование» (Рис. 2).

В рамках работы данные мобильные приложения интересны прежде всего с точки зрения того, какой функциональностью они обладают. Это позволяет убедиться, что разрабатываемый сервис её включает.

Таким образом, можно выделить, что приложения от страховых компаний дают возможность:

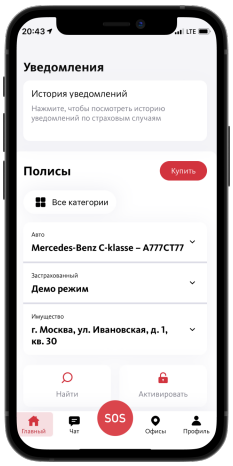

Рис. 2: Альфа [5]

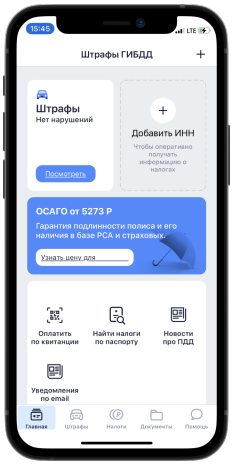

Рис. 3: Штрафы ГИБДД официальные ПДД [6]

- оформлять и оплачивать Каско, ОСАГО;
- заявлять о страховом случае;
- общаться с представителями страховой компании в чате;
- просматривать инструкции по алгоритму действий при страховом случае.

2.1.2. Сервисы для оплаты штрафов и налогов

Помимо обязанности иметь страховой полис, у водителей есть необходимость отслеживать и вовремя оплачивать штрафы, которые неизбежно появляются даже у самых аккуратных водителей. Для этого были разработаны и опубликованы приложения, которые заменили надобность отлавливать квитанции по адресу прописки, а затем оплачивать их в банке. Процесс намного упростился и теперь можно получить уведомления о штрафе и сразу оплатить его прямо в мобильном приложении.

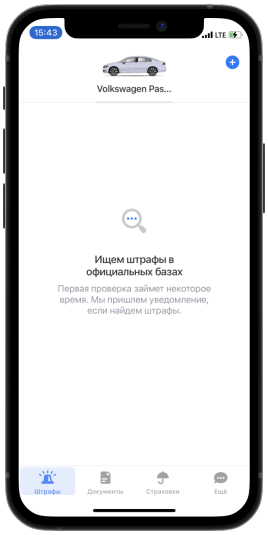

Рис. 4: Штрафы ГИБДД [7]

Штрафы ГИБДД официальные ПДД (Рис. 3) и Штрафы ГИБДД (Рис. 4) — популярнейшие приложения из данной категории.

С точки зрения функциональности, они позволяют:

- получать уведомления о штрафах и налогах и оплачивать их;
- просматривать оплаченные штрафы и налоги;
- настраивать уведомления о конце срока страховки;

- просматривать обновления в правилах дорожного движения.

2.1.3. Приложения для анализа трат

Также некоторые водители хотят отслеживать свои траты на автомобиль, чтобы затем планировать бюджет. Для данных нужд также выпускаются приложения, которые позволяют анализировать расходы. Однако они не берут данные автоматически, а лишь дают возможность вносить их вручную, что значительно сложнее и является менее точным, ведь можно что-то забыть внести. Несмотря на это, такой тип приложений популярен на рынке согласно оценкам в магазине приложений AppleStore [9].

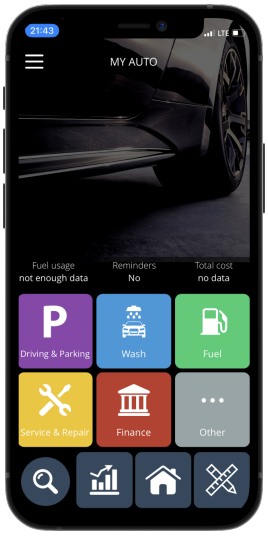

Рис. 5: My Auto - petrol usage [8]

Примером приложения из данной категории является приложение My Auto (Рис. 5). В нём заложена возможность:

- вносить траты по автомобилю;
- следить за динамикой расхода топлива;
- строить графики и диаграммы.

2.1.4. Сервисы, связанные с историей автомобиля

Все затронутые ранее категории являются не менее важными, но явно менее популярными, чем сервисы, позволяющие смотреть историю автомобиля перед покупкой, оценивать стоимость. В последние годы данные процедуры являются необходимыми перед продажей или покупкой, ведь они показывают все проблемы автомобиля и позволяют выбрать наиболее подходящий вариант. Обычно такие сервисы добавляют в онлайн площадки по продаже подержанных автомобилей, ведь именно на них заходит большинство вероятных пользователей.

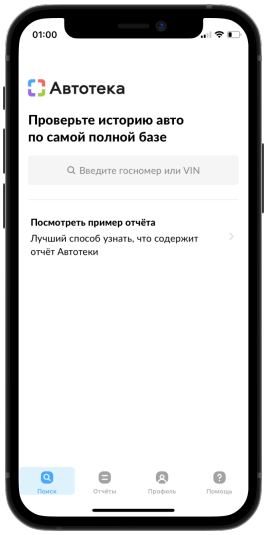

Рис. 6: Автотека [10]

Самое известное мобильное приложение с данной функциональностью — Автотека (Рис. 6)

Таким образом, хоть подобные сервисы и лишь косвенно касаются процесса владения автомобилем, они являются важными для прозрачности процессов покупки и продажи автомобиля. Их функциональность заключается в:

- возможности посмотреть юридические ограничения, сведения о ДТП, пробеге и другую информацию по автомобилю;
- оценить стоимость автомобиля.

2.1.5. Вывод

В обзоре были рассмотрены не все категории мобильных приложений, связанные с автомобилями. Таких приложений много, но не все они интересны в рамках работы. Например: навигаторы, трекеры передвижений, приложения для парковок и сервисных центров. Все они хоть и связаны с машинами, но либо не являются необходимыми и особенно популярными, либо не могут быть встроены в сервис. Стоит подчеркнуть, что в процессе обзора были выделены категории приложений, в которых приложения по объективным причинам оказались ограничены по функциональности. Из этого можно сделать вывод, что хоть в совокупности эти приложения и покрывают нужды владельцев автомобиля, однако каждое из них необходимо загрузить, внести туда свои данные, информацию об автомобиле и платёжную.

Тем не менее на рынке нет популярных сервисов, собирающих воедино функциональность различных категорий и предоставляющих её пользователям. Отчасти это связано с тем, что подобный сервис могут создать крупные компании, которые объединяют в себе как страховую компанию и банк, так и имеют возможность привлекать партнёров для совместной работы. А подобных компаний, которым интересен данный сервис и которые развиваются в направлении супераппов, ещё меньше.

Таким образом, создание такого сервиса покроет основные потребности владельцев автомобилей и позволит им экономить время и деньги.

2.2. Архитектура iOS-приложений

2.2.1. MVC

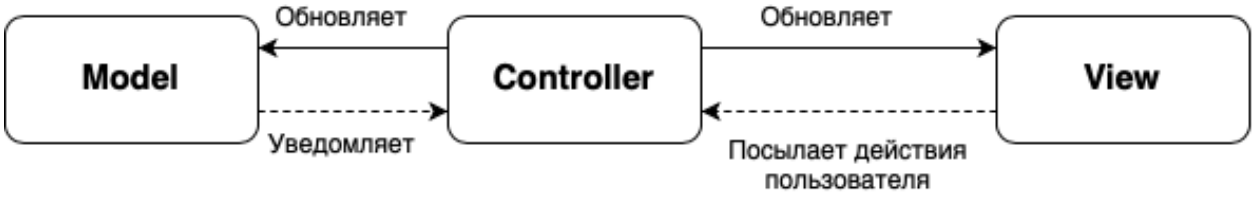

Рис. 7: Концепция Cocoa MVC от компании Apple [11]

На рисунке 7 представлена схема паттерна Model View Controller.

В данном паттерне модель (Model) отвечает за данные. К ней относятся хранилища, модели объектов, взаимодействие с сервером, конвертеры и многое другое, связанное с данными.

Вид или представление (View) отвечает за интерфейс (UI) в приложении, а также за обработку действий пользователя. Объекты, которые относятся к данному компоненту, легко переиспользуются.

Контроллер (Controller) выступает посредником между моделью и представлением, и обычно он не связан с каким-то конкретным видом. Достигается это через паттерн Делегат [12].

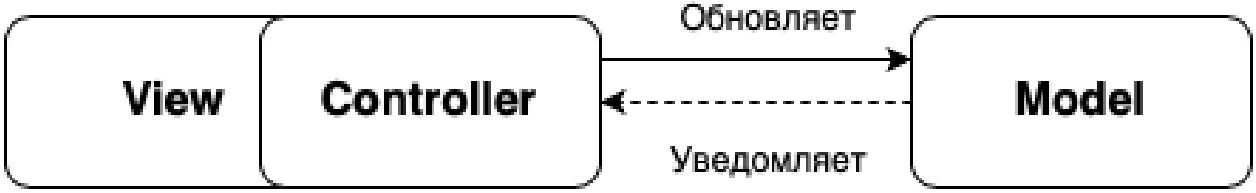

Рис. 8: Паттерн MVC в реальности

Однако теоретическая составляющая паттерна MVC немного отличается от практической (Рис. 8), так как на деле вид тесно связан с контроллером и объясняется это особенностью фреймворка UIKit [13].

В нём контроллер содержит вид, и поэтому они воспринимаются как единое целое.

Данный паттерн, хоть несомненно и обладает преимуществами, такими как простота использования и высокая скорость разработки, имеет большое количество недостатков.

- Во-первых, со временем данный паттерн превращается из Model View Controller в Massive View Controller, так как контроллер будет быстро расти, что будет сильно усложнять добавление новой функциональности, а значит повлечёт за собой плохую масштабируемость. Это обусловлено тем, что в данном подходе слабо разделена ответственность, и контроллер пытается взять на себя как бизнес-логику, так и логику отображения.

- Во-вторых, в данном подходе сложно покрывать программу юнит тестами, что на самом деле вытекает напрямую из первого недостатка и является следствием того, что контроллер берёт слишком много ролей на себя.

Таким образом, паттерн MVC не подходит сервису, разрабатываемому в рамках работы, по критериям, сформулированным в начале раздела с архитектурой. (Стр. 21)

2.2.2. MVVM

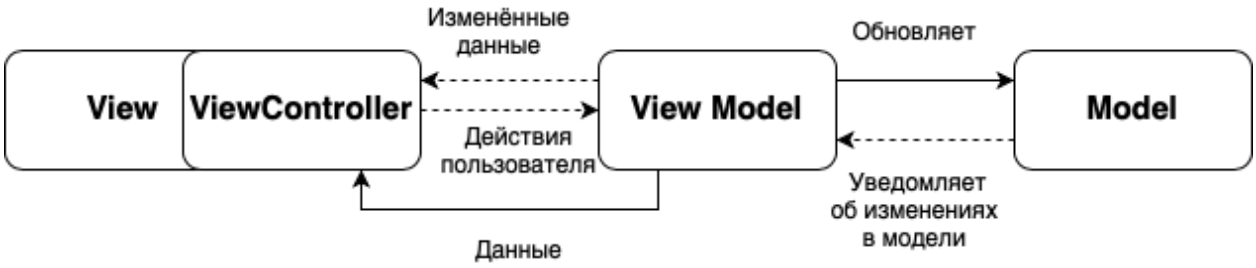

Рис. 9: Паттерн MVVM

Следующим рассмотрим паттерн Model View ViewModel (Рис. 9). Как и MVC, паттерн разделяется на три составляющие:

1. Вид (View) является графическим интерфейсом. Если пользователь взаимодействует с элементом интерфейса, то вызывается соответствующий метод в модели вида (View Model). Также, если меняется свойство в модели вида, то об этом оповещается вид и он обновляется.

2. Модель (Model) также отвечает за данные в приложении и всё, что с ними связано.

3. Модель вида (View Model) отражает текущее состояние View в любой момент времени (является некоторой абстракцией вида) и должна содержать команды, которые использует вид для влияния на модель. Для того, чтобы обеспечить связывание вида и модели через модель вида используется реактивное программирование.

Подход MVVM имеет хорошее распределение ответственности (из-за связывания данных, через которое View через ViewModel обновляет своё состояние) и хорошо тестируется (даже лучше, чем подход, который будет выбран). Но для того, чтобы использовать данный паттерн, необходимо внедрять реактивные фреймворки в проект. Использование таких фреймворков связано с огромным количеством трудностей, особенно на больших проектах. Поэтому отказ от данного паттерна обусловлен тем, что данных подход не используется в мобильном банке, в котором необходимо внедрять сервис.

2.2.3. MVP

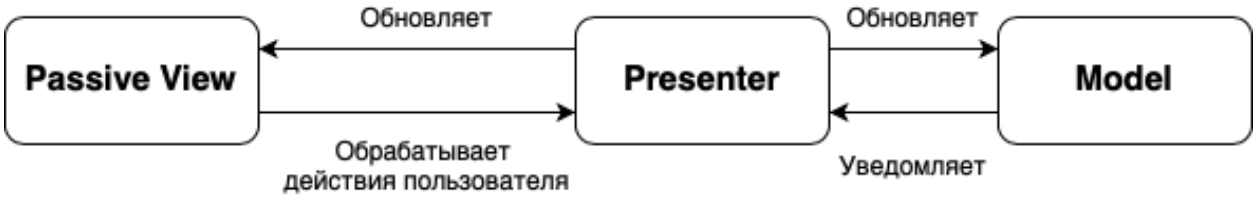

Рис. 10: Паттерн MVP

Паттерн Model View Presenter (Рис. 10) является развитием идей MVC. Хоть презентер (аналогично контроллеру) берёт на себя роль

посредника, он не так тесно связан с видом, поэтому в MVP вид и имеет прилагательное пассивный. Презентер не управляет жизненным циклом View Controller, что сильно облегчает его тестируемость.

Данный паттерн подходит по критериям и мог быть использован при разработке сервиса, однако предпочтение было отдано паттерну VIPER из-за большего разделения ответственности.

2.2.4. VIPER

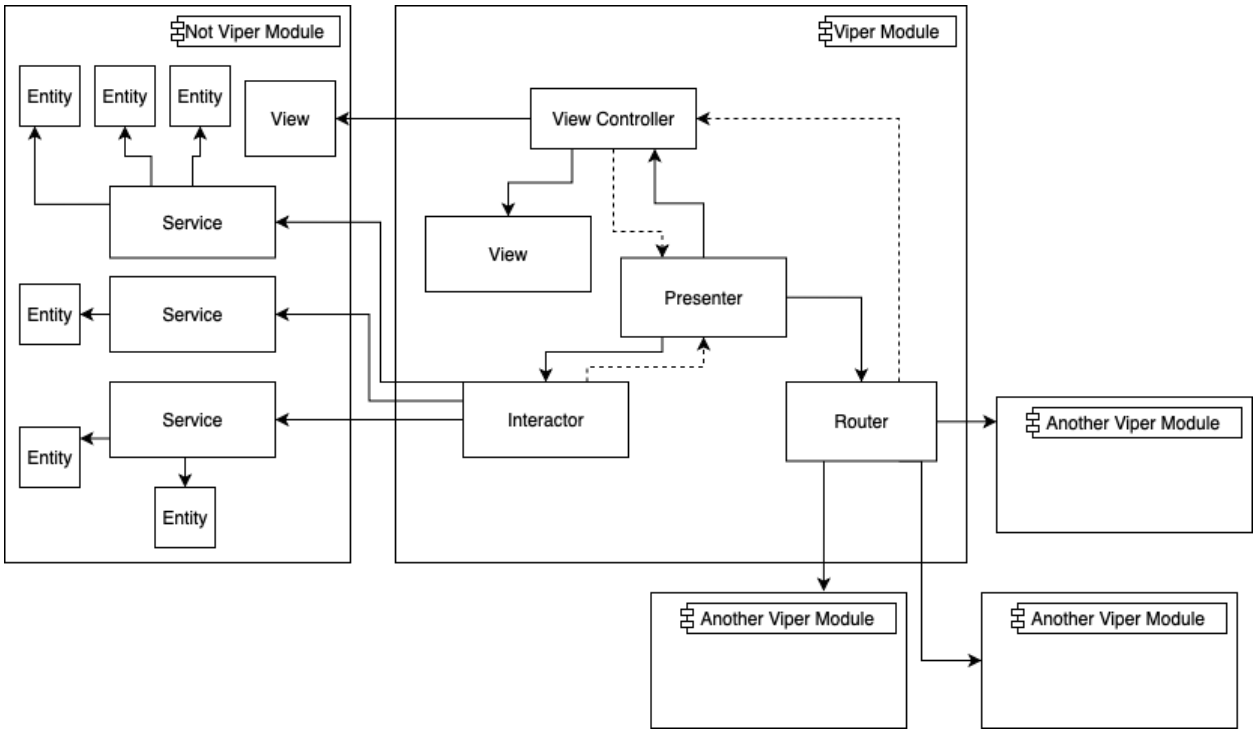

Рис. 11: Паттерн Viper

Паттерн VIPER (View Interactor Presenter Entity Router) продолжает идеи MVC и MVP, однако имеет намного менее сильную связанность программного кода.

В данном подходе приложение разбивается на модули, а пользователь, переходя по экранам, на самом деле переключает модули. В данном паттерне число слоёв больше.

- Маршрутизатор (Router) отвечает за переходы между модулями.

- Интерактор (Interactor) берёт на себя логику по работе с данными. Через сервисы он получает данные, взаимодействуя с сервером. Обычно сервисы не включаются в модуль VIPER из-за того, что они часто переиспользуются несколькими разными модулями.

- Объект (Entity) многие не относят к модулю VIPER, а скорее привязывают к сервисам, которые в этот модуль не входят. Однако интерактор всё же знает о нём. Сюда относятся объекты данных, однако данный слой не регулирует доступ к данным, этим занимается интерактор. Может также входить логика конвертации из JSON структуры в объект.

- Вид (View) похож на прошлые паттерны и отвечает за графическую составляющую и взаимодействие с пользователем.

- Презентер (Presenter) выступает в роли менеджера слоёв и через интерактор взаимодействует с сервером и базами данных, а также через вид взаимодействует с пользователем. Стоит отметить, что всё взаимодействие происходит через протоколы (интерфейсы класса).

Главное отличие паттерна VIPER от семейства MV-X паттернов в том, что вводится отдельный слой «Интерактор», который берёт на себя взаимодействие с данными, а также появляется Роутер, который занимается маршрутизацией между модулями, разгружая тем самым презентер.

Основными преимуществами данного подхода являются:

- независимость модулей друг от друга, что позволяет удобно их обновлять и удалять, а также независимо разрабатывать;

- удобство тестирования из-за менее сильной связанности;

- переиспользуемость компонентов.

К минусам можно отнести большой объём программного кода, что негативно влияет на скорость разработки, а также явно более сложную структуру, что поднимает порог вхождения.

Тем не менее паттерн VIPER наиболее точно подходит заявленным критериям и именно он был использован при разработке сервиса.

2.3. Использованные технологии

Перед тем как приступить к реализации сервиса, необходимо было продумать подход к работе с ресурсами (локализация, картинки), а также к разработке UI компонентов.

Компания Apple предлагает собственный подход к локализации и работе с изображениями. Предоставляется файл Localizable.strings, в котором хранятся пары «ключ-значение». А для изображений создаются отдельные модули в проекте. Если необходимо использовать картинку или локализованный текст, то надо в программном коде указать ключ. Однако такая концепция имеет существенные недостатки. Если был неверно указан ключ, то в процессе компиляции разработчику не будет помечена ошибка, а в этом месте в пользовательском интерфейсе будет локализационный ключ вместо текста. Это происходит из-за того, что ресурсы добавляются в пакет (от англ. Bundle[3]) уже после компиляции проекта. Таким образом, ошибки подобного вида можно увидеть лишь в процессе ручного тестирования, потому что такую проверку сложно автоматизировать.

Для предотвращения подобного рода ошибок были созданы специальные библиотеки, использующие кодогенерацию для решения проблемы. Самые популярные из них: R.swift [14], Swiftgen [15] и Natalie [16].

Инструмент Natalie не подходил для сервиса из-за того, что был создан для узкоспециализированных задач, связанных с кодогенерацией элементов UI. Swiftgen и R.swift же, позволяют работать со шрифтами, картинками, локализацией строк, цветами и многим другим, поэтому любой из них мог быть использован при реализации сервиса. Выбран же был R.swift из-за того, что позволяет также работать с xml, json, plist файлами.

[3]Bundle — совокупность данных, объединённых по признаку.

Что же касается UI компонентов, то в разработке на платформе iOS используется несколько разных подходов: с помощью Storyboard[4] и прямо в программном коде, используя библиотеку SnapKit [17].

Оба подхода имеют ряд достоинств и недостатков. К преимуществам Storyboard относятся:

- визуализация экрана (причём всех размеров) и отступов;
- моментальные изменения без компиляции проекта;
- наглядность списка атрибутов у UI элементов, нет необходимости смотреть в документацию.

Однако недостатки более существенны:

- Storyboard сложно переиспользовать;
- проявляются замедления при увеличении количества экранов;
- сложно решать конфликты, возникающие в файлах при слиянии изменений, особенно в больших командах.

При использовании подхода ручной вёрстки достоинства и недостатки предыдущего подхода меняются местами. Следовательно, принимая во внимание то, что переиспользуемость компонент была заявлена в нефункциональных требованиях к продукту, а также то, что сервис будет поддерживаться и дополняться большой командой, в которой необходимо будет параллельно разрабатывать новые экраны, был выбран подход ручной вёрстки.

[4]Storyboard — графический инструмент от Apple, позволяющий визуализировать экраны

3. Требования к сервису

После обсуждения с бизнес руководителями и командой были выделены следующие требования.

3.1. Функциональные требования

В первой версии продукта решено было сосредоточиться на наборе самой необходимой функциональности.

- Добавление и удаление автомобиля и документов.
- Наличие внутренних сервисов:
  - ремонт;
  - заправки.
- Отображение и оплата штрафов и налогов по каждому автомобилю.
- Оформление, оплата и пролонгация страховых продуктов банка.
- Возможность заявить о страховом случае по автомобилю.
- Помощь на дороге[5].
- Возможность встраивать баннеры с информацией о продуктах банка.
- Отображение информации о расходах на автомобиль по категориям, спецпредложениях, бонусах, кэшбеках.
- Возможность предложить пользователю добавить в сервис автомобиль, информация по которому уже имеется.

[5]Помощь по составлению европротокола, эвакуатор ...

3.2. Нефункциональные требования

- Язык разработки — Swift. Данное требование обусловлено тем, что в проекте мобильного банка в компании разработка ведётся на Swift.

- Лёгкое встраивание новой функциональности в сервис. Из-за желания компании пораньше запустить сервис, в первую версию вошла не вся функциональность, которая планируется. Поэтому одним из главных нефункциональных требований стала лёгкая расширяемость сервиса.

- Поддержать возможность сбора аналитических данных. Также является очень важным требованием, потому что в совокупности со статистикой MAU[6] и количеством клиентов банка, имеющих автомобиль, можно отследить, заметили ли пользователи новый сервис и, возможно, больше его рекомендовать.

- Кеширование информации для ускорения работы сервиса.

- Поддержать deeplinks[7].

[6]Monthly Active Users — количество активных пользователей приложения в месяце

[7]Особый вид ссылок, позволяющий направлять пользователя на конкретную страницу в приложении

4. Проектирование архитектуры сервиса

4.1. Требования к архитектуре

Для того, чтобы спроектировать подходящую для сервиса архитектуру и выбрать подходящий паттерн, необходимо сформулировать критерии, которые к ней предъявляются.

- Гибкость. Основной критерий, который коррелирует с требованием лёгкой встраиваемости новой функциональности в сервис и означает, что можно легко разрабатывать и встраивать новые возможности, переиспользовать старые компоненты.

- Лёгкая тестируемость, которая вытекает из распределения обязанностей и принципа единственности ответственности. Не менее важная характеристика для архитектуры, так как она влияет на покрытие программного кода юнит тестами, а соответственно, на скорость выпуска новой функциональности для пользователя.

Также есть и критерии, которыми можно пренебречь, ради основных.

- В первую очередь это скорость разработки. Как правило, чем сложнее архитектура и чем больше в ней распределены обязанности, тем дольше идёт процесс разработки. Однако в данном случае необходимо учитывать, что данный сервис встраивается в уже существующее приложение, которое является монолитом. Отсюда вытекает, что в нём будет появляться всё больше и больше функциональности, что означает, что при неверном подходе к архитектуре и желании как можно быстрее разрабатывать новые продукты, не уделяя должного внимания архитектуре, придётся потратить не меньше времени в дальнейшем на пересмотр архитектуры, что может привести к полной остановке разработки.

- Также придётся пренебрегать порогом входа. Чем сложнее архитектура, тем сложнее войти новым людям в проект и команду, а значит тем сложнее его поддерживать.

4.2. Архитектура сервиса

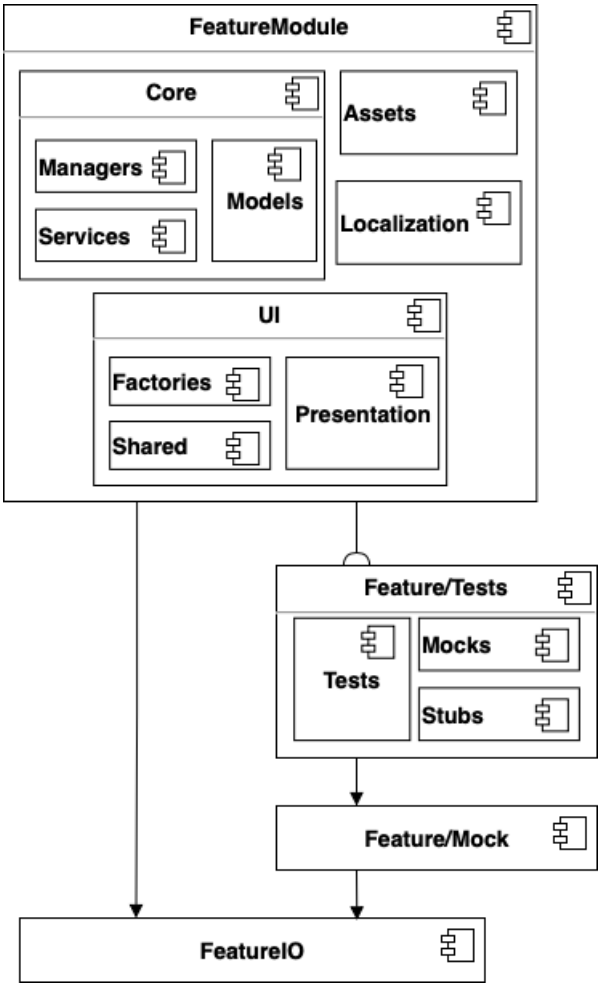

Рис. 12: Архитектура сервиса

Кодовая база сервиса состоит из четырёх модулей (Рис. 12).

- Основной функциональный модуль (FeatureModule). Делится на ядро (Core), слой интерфейса и вспомогательные. Ядро содержит менеджеры, сервисы и связанные с ними модели. Интерфейс включает фабрики (Factories), общие для экранов контроллеры и виды (Shared) и главный модуль представления (Presentation). Также в модуль входят компоненты, отвечающие за локализацию и ресурсы (Localization и Assets).

- Модуль Feature/Tests, который содержит как UI, так и юнит-

тесты. Также туда входят Mocks[8] и Stubs[9].

- Модуль Feature/Mocks содержит Mock и Stub объекты, которые могут быть переиспользованы в других функциональных модулях.

- Модуль FeatureIO включает в себя публичные интерфейсы объектов из основного модуля, которые могут быть переиспользованы другими функциональными модулями.

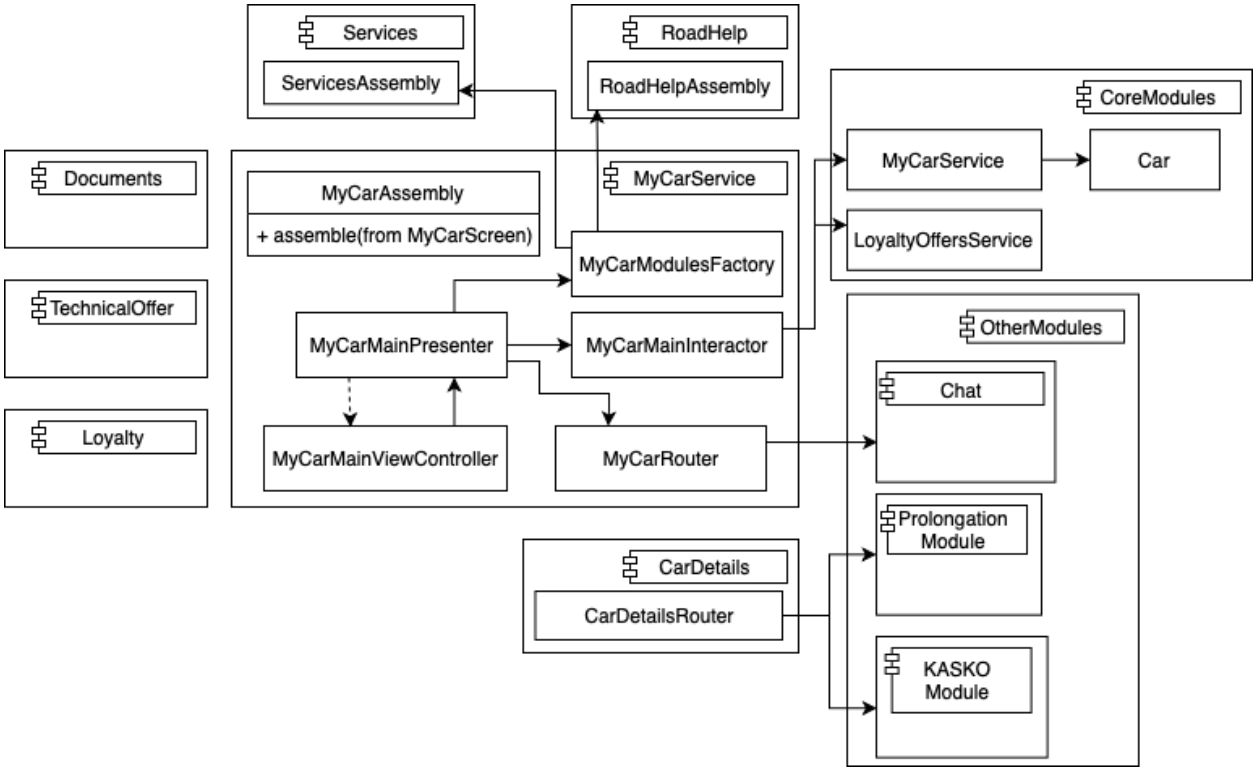

Рис. 13: Архитектура слоя представления

В модуле представления для реализации бизнес-логики решено было использовать также модульный подход и применять паттерн VIPER (Рис. 13). Основным модулем сервиса является Главная страница,

[8]Mock — (от англ. объект-имитация) объекты, фиктивно реализующие интерфейс для тестирования взаимодействия. Позволяют задавать ответ на запросы и подсчитывать вызовы.

[9]Stub — (от англ. заглушка) объекты, реализующие интерфейс и в которых заранее предопределяют ответ на вызовы.

к которой подключаются модули, отвечающие за внутренние сервисы, документы, детали машины, помощь на дороге и другую функциональность. Чтобы добавить новую функциональность, необходимо реализовать соответствующий модуль и встроить его в фабрику MyCarModulesFactory.

5. Реализация сервиса

5.1. Процесс разработки сервиса

Процесс разработки сервиса вёлся по методу Kanban с определёнными изменениями и дополнениями. Данный метод позволяет обеспечить гибкость и адаптироваться к изменениям. Спроектированный сервис был разделён на блоки функциональности, которые также были декомпозированы на маленькие задачи. Каждая задача имела свой приоритет.

Основополагающим для процесса является выпуск обновления продукта (релиз), который происходит в банке раз в месяц. Между ними находятся 11 дней разработки и 9 дней регрессионного тестирования. Заранее планируется какие задачи должны войти в следующий релиз, однако данное правило не является строгим и задачи могут пересматриваться в соответствии с приоритетами. Три раза в неделю проводились встречи, в которых обсуждались сложности с текущими задачами и что можно сделать для их скорейшего решения. После окончания процессов разработки и регрессионного тестирования организовывалась ретроспективная встреча, в которой обсуждалось, что можно улучшить в процессе.

Для удобства процесса разработки использовался инструмент Jira Atlassian [18], который является электронной версией канбан-доски. На рисунке 14 представлен процесс разработки. После написания технического задания задача попадает в статус «Необходимо сделать». Затем, после написания тестовых сценариев задача готова к разработке. После того как она была выполнена, программный код был отрецензирован и принят, а также были реализованы тесты, задача должна пройти рецензирование команды тестирования и дизайна. После всех процессов задача попадает в статус «Готова» и ждёт выпуска обновления приложения.

Также для оптимизации процесса были введены WIP-лимиты[10], что-

[10]WIP — Work in progress, ограничения на количество задач в определённом статусе

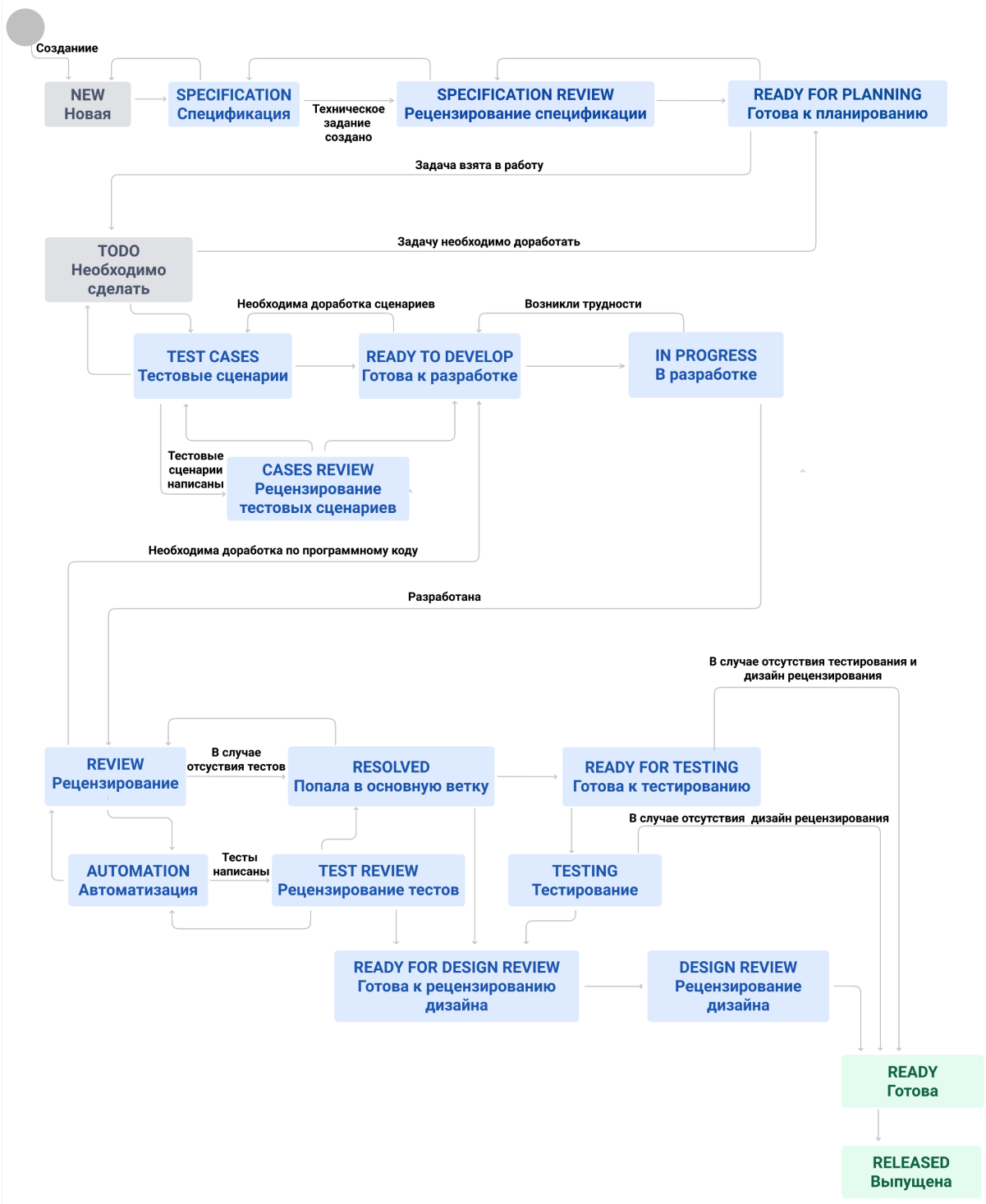

Рис. 14: Процесс разработки

бы обращать внимание на ситуации, когда много задач находятся в одном статусе и превышают возможности команды.

5.2. Особенности реализации

Основной компонентой сервиса выступает главный экран (Рис. 15), который собирает в себе остальные модули.

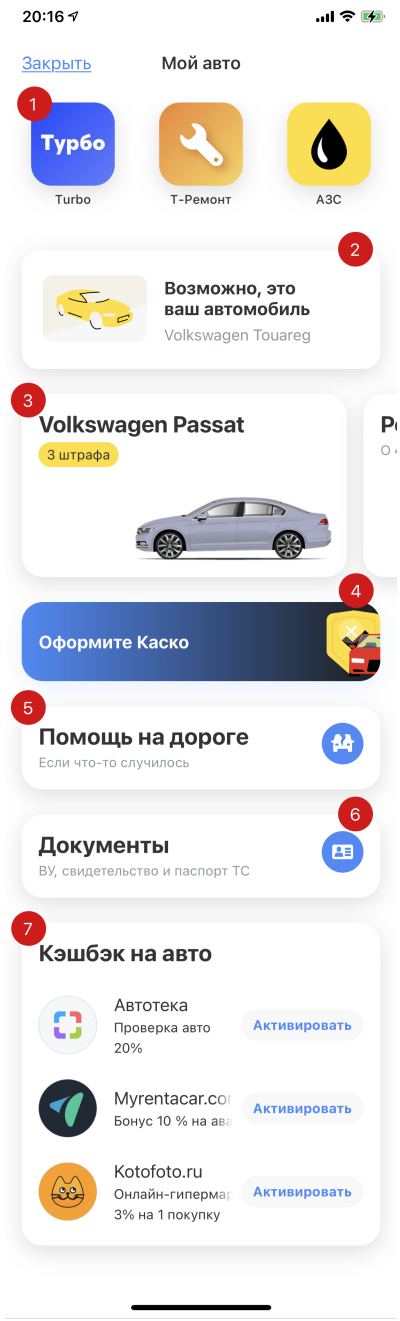

Рис. 15: Главный экран сервиса

1. В верхней части экрана представлены сервисы. Их набор подбирается индивидуально из доступных сервисов в городе клиента. Каждый из них ведёт в отдельный модуль. Чтобы прикрепить новый сервис, требуется лишь добавить его модель на сервер.
2. (Опционально). Если у банка уже имеется информация о машине клиента, то предлагается её добавить, воспользовавшись «техническим оффером». «Оффер» (Предложение) прикрепляется к клиенту на сервере и содержит, например, такие данные, как: марка и модель машины, идентификационный и регистрационный номер, тип двигателя и другие. Если в предложении содержится часть информации, то после добавления автомобиля на его карточке отображается соответствующий баннер, а информацию предлагается дополнить.
3. Чуть ниже располагается карусель с машинами клиента или кнопка «Добавить автомобиль», если они ещё не добавлены. Нажатие на картинку автомобиля ведёт на детали с полисами и штрафами, если они есть, о чём сообщает баннер.
4. (Опционально). «Универсальный оф-

фер» с информацией о другом продукте банка. Могут прикреплять маркетологи через специальный сайт. Можно настраивать различные шрифты, цвета, картинки. Предложения могут быть различных типов: скрываемые и нет, с заголовком, описанием и без, вести по нажатию на другой экран, открывать сайт или историю.

5. Модуль «Помощь на дороге». Предоставляет сведения о полисах, действиях при ДТП, европротоколе, содержимом аптечки и другую информацию, необходимую автомобилистам.

6. Блок документов. Даёт возможность добавлять паспорт транспортного средства, свидетельство о регистрации и водительское удостоверение. При этом их можно сканировать с помощью камеры, чтобы не вводить данные вручную.

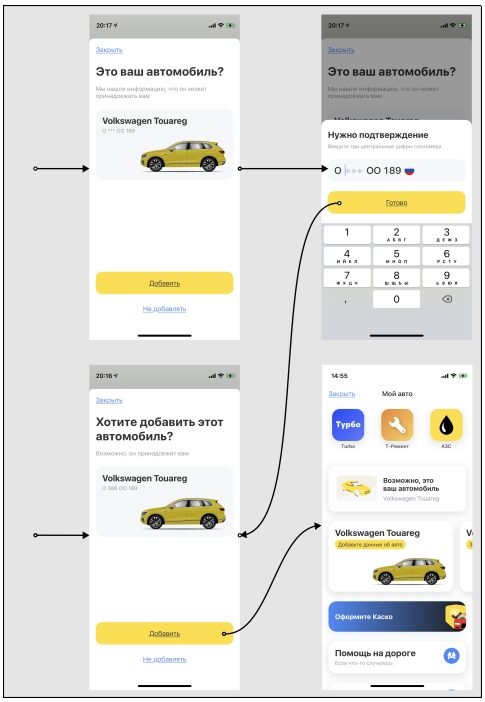

Рис. 16: «Технический оффер»

7. Модуль предложений с кэшбеком. В данный раздел попадают все спецпредложения по кэшбеку, связанные с категорией автомобилей. Их можно просмотреть и активировать прямо в сервисе.

На рисунке 16 представлен процесс добавления автомобиля по «Техническому офферу». Если банк не уверен, что автомобиль принадлежит клиенту, то номер маскируется, а пользователю предлагается ввести три цифры номера, чтобы подтвердить, что автомобиль в его собственности. Клиенту даётся три попытки. Количество использованных попыток необходимо было сохранять при повторной авторизации пользователя в приложение. Поэтому был написан сервис, который привязывается к идентификатору пользователя и хранит информацию по ключу, не удаляя при этом информацию при авторизации под другим логином.

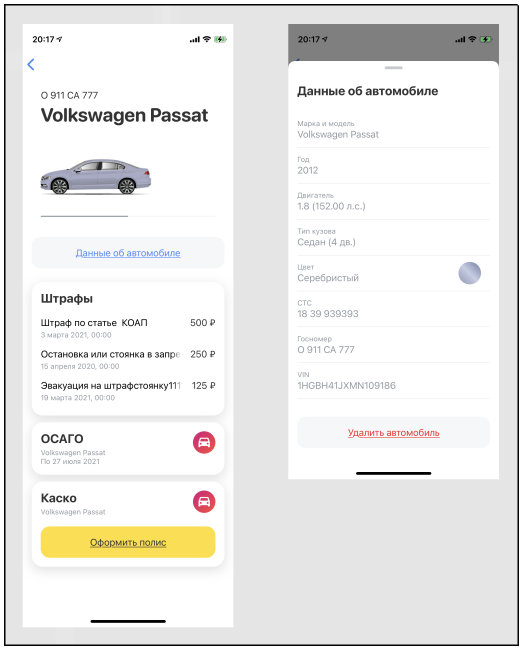

Рис. 17: Детали автомобиля

Блок «помощь на дороге» содержит информацию о различных ситуациях, которые могут возникнуть на дороге. (Рис. 18). Данный блок реализован так, что он берёт отображаемую информацию и изображения с сервера, чтобы их можно было изменить, не выпуская новую версию приложения для обновления.

Вторым главным компонентом выступают детали автомобиля (Рис. 17). В них можно оплатить штрафы, оформить полис, посмотреть детали полиса и зарегистрировать ДТП.

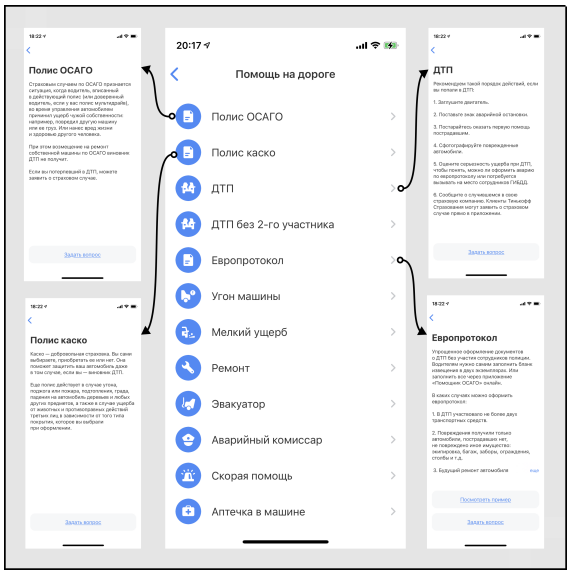

Рис. 18: «Помощь на дороге»

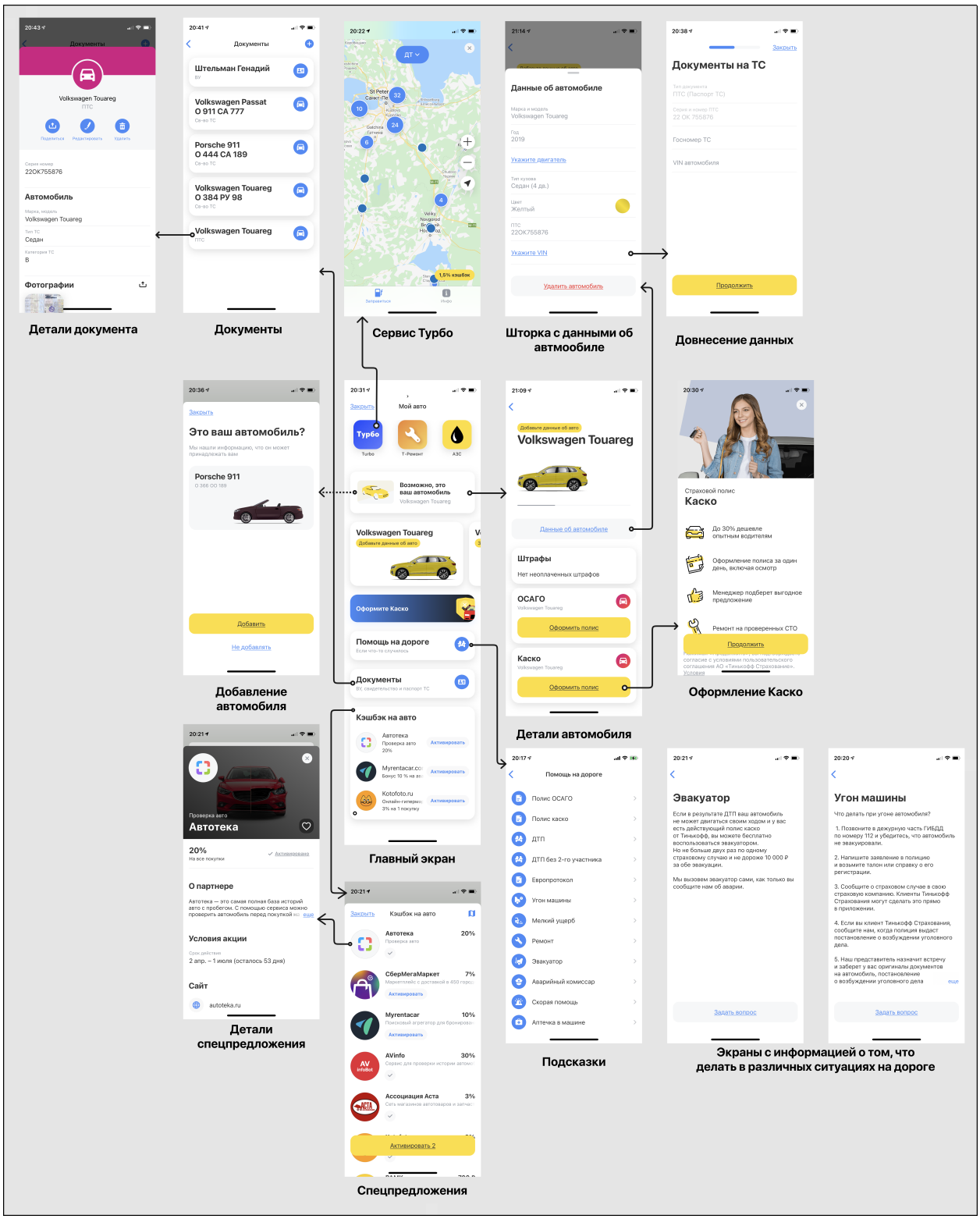

Рис. 19: Схема навигации

На рисунке 19 представлена схема навигации в сервисе.

6. Тестирование сервиса

Рис. 20: Пирамида тестирования

В мобильном банке процессу тестирования выделяется большая роль из-за того, что сбои в приложении ведут к финансовым потерям для клиентов. Помимо этого, наличие автоматических тестов влияет напрямую на скорость разработки новой функциональности и продолжительность регрессионного тестирования приложения. При покрытии программного кода применяется подход пирамиды тестирования (Рис. 20). Данный принцип обусловлен тем, что юнит тесты более быстрые, чем UI тесты и их намного легче выполнить. Однако далеко не все случаи можно проверить юнит тестами (Например проверить слой View/ViewController), поэтому такие случаи покрываются UI тестами, которых, правда значительно меньше. Всё, что не удаётся проверить автоматическими тестами, например: переход в другое приложение, если оно установлено на устройстве, проверяется командой тестирования на этапе регрессионного тестирования приложения.

Для покрытия тестами сервиса было написано более 200 тестовых случаев. Автоматические тесты покрыли более 70% исходного кода. Использовался встроенный фреймворк XCTest [19]. Для автоматической генерации тестовых данных применялась библиотека Fakery [20].

Тесты запускаются после сборки проекта в момент создания запроса на внесение изменений в основную ветку продукта. Процесс представлен на рис. 21

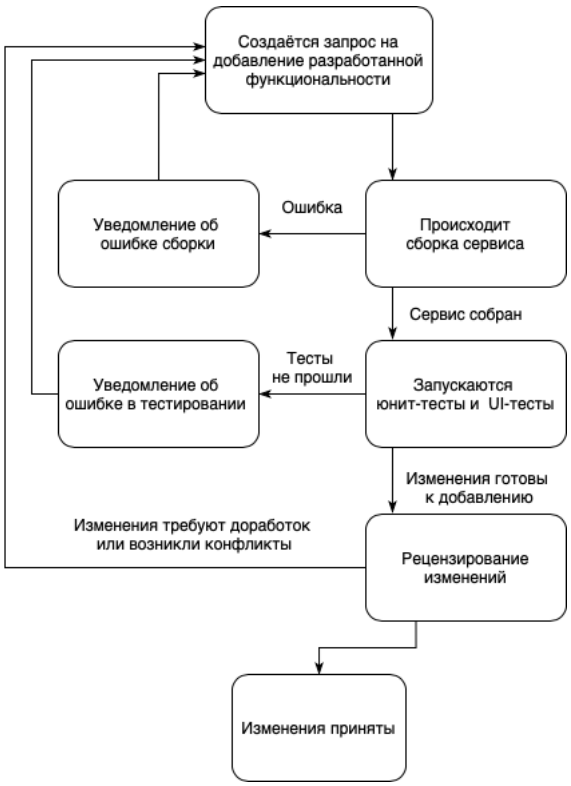

Рис. 21: CI/CD процесс

Сборка проекта и тестирование производится при помощи связки сервисов Attlassian Stash [21] и TeamCity [22].

Заключение

В данной работе были достигнуты следующие результаты:

- проанализированы существующие приложения и сервисы, связанные с автомобилями;
- выявлены требования к сервису;
- спроектирован сервис и его интеграция в мобильном банке;
- сервис реализован на платформе iOS;
- сервис покрыт UI и юнит-тестами.

Сервис вошёл в релиз приложения Тинькофф Мобильный банк [23], в версию 5.14.5.

Список литературы

[1] Аналитическая платформа Gfk. Проникновение Интернета в России. — 2019. — Режим доступа: https://www.gfk.com/ru/press/issledovanie-gfk-proniknovenie-interneta-v-rossii (дата обращения: 01.05.2021).

[2] Федеральная служба государственной статистики. Число легковых автомобилей на 1000 человек. — 2020. — Режим доступа: http://www.gks.ru/bgd/regl/b12_13/IssWWW.exe/Stg/d1/06-38.htm (дата обращения: 01.05.2021).

[3] Автостат. Сколько автомобилей приходится на тысячу жителей в России? — 2020. — Режим доступа: https://www.autostat.ru/news/46352/ (дата обращения: 01.05.2021).

[4] Ренесанс. Авто. Мобильное приложение страховой компании Ренесанс. — 2021. — Режим доступа: https://apps.apple.com/ru/app/ренессанс-авто/id1456449980 (дата обращения: 01.05.2021).

[5] АльфаСтрахование. Мобильное приложение в сфере страхования от Альфа-Групп. — 2021. — Режим доступа: https://apps.apple.com/ru/app/альфастрахование-мобайл/id1039418352 (дата обращения: 01.05.2021).

[6] Payment Systems. Штрафы ГИБДД официальные ПДД. — 2021. — Режим доступа: https://apps.apple.com/ru/app/штрафы-гибдд-официальные-пдд/id824748040 (дата обращения: 01.05.2021).

[7] АО «Тинькофф Банк». Штрафы ГИБДД официальные. — 2021. — Режим доступа: https://apps.apple.com/ru/app/штрафы-гибдд-официальные/id887937919 (дата обращения: 01.05.2021).

[8] KinKin Ltd. My Auto - petrol usage. — 2021. — Режим доступа: https://apps.apple.com/by/app/my-auto-petrol-usage/id1073705524 (дата обращения: 01.05.2021).

[9] Apple Inc. Магазин приложений Apple Store. — 2021. — Режим доступа: https://www.apple.com/ru/app-store/ (дата обращения: 01.05.2021).

[10] Автотека. Автотека - сервис проверки автомобиля по VIN коду и госномеру. — 2021. — Режим доступа: https://autoteka.ru (дата обращения: 01.05.2021).

[11] Apple Inc. Cocoa MVC от компании. — 2021. — Режим доступа: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html (дата обращения: 01.05.2021).

[12] Swiftblog. Делегирование в Swift. — 2021. — Режим доступа: https://swiftblog.org/delegirovanie-v-swift/ (дата обращения: 01.05.2021).

[13] Apple Inc. Фреймворк UIKit. — 2021. — Режим доступа: https://developer.apple.com/documentation/uikit (дата обращения: 01.05.2021).

[14] Mathijs Kadijk. R.swif. — 2021. — Режим доступа: https://github.com/mac-cain13/R.swift (дата обращения: 01.05.2021).

[15] Olivier Halligon. Swiftgen. — 2021. — Режим доступа: https://github.com/SwiftGen (дата обращения: 01.05.2021).

[16] Marcin Krzyzanowski. Natalie. — 2021. — Режим доступа: https://github.com/krzyzanowskim/Natalie (дата обращения: 01.05.2021).

[17] Robert Payne. SnapKit. — 2021. — Режим доступа: https://github.com/SnapKit/SnapKit (дата обращения: 01.05.2021).

[18] Atlassian. Jira. — 2021. — Access mode: https://www.atlassian.com/ru/software/jira (online; accessed: 01.05.2021).

[19] Apple Inc. XCTest. — 2021. — Режим доступа: https://developer.apple.com/documentation/xctest (дата обращения: 01.05.2021).

[20] vadymmarkov. Fakery. — 2021. — Режим доступа: https://github.com/vadymmarkov/Fakery (дата обращения: 01.05.2021).

[21] Atlassian. Bitbucket. — 2021. — Режим доступа: https://www.atlassian.com/ru/software/bitbucket (дата обращения: 01.05.2021).

[22] JetBrains s.r.o. TeamCity. — 2021. — Режим доступа: https://www.jetbrains.com/ru-ru/teamcity/ (дата обращения: 01.05.2021).

[23] АО «Тинькофф Банк». Тинькофф Мобильный банк // Официальный сайт. — 2021. — Режим доступа: https://apps.apple.com/ru/app/тинькофф-мобильный-банк/id455652438 (дата обращения: 01.05.2021).