

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему компьютерный ассистент менеджера по подбору площадок
для киносъемок

Выполнил:

студент группы БПМИ196_



Подпись

А.В.Войба

И.О. Фамилия

22.05.2021

Дата

Принял:

руководитель проекта Андрей Андреевич Паринов

Имя, Отчество, Фамилия

Младший научный сотрудник

Должность, ученое звание

МНУЛ ИССА НИУ ВШЭ

Место работы (Компания или подразделение НИУ ВШЭ)

Дата проверки
22.05.2021

9

Оценка
(по 10-тибалльной шкале)



Подпись

Москва 2021

Содержание

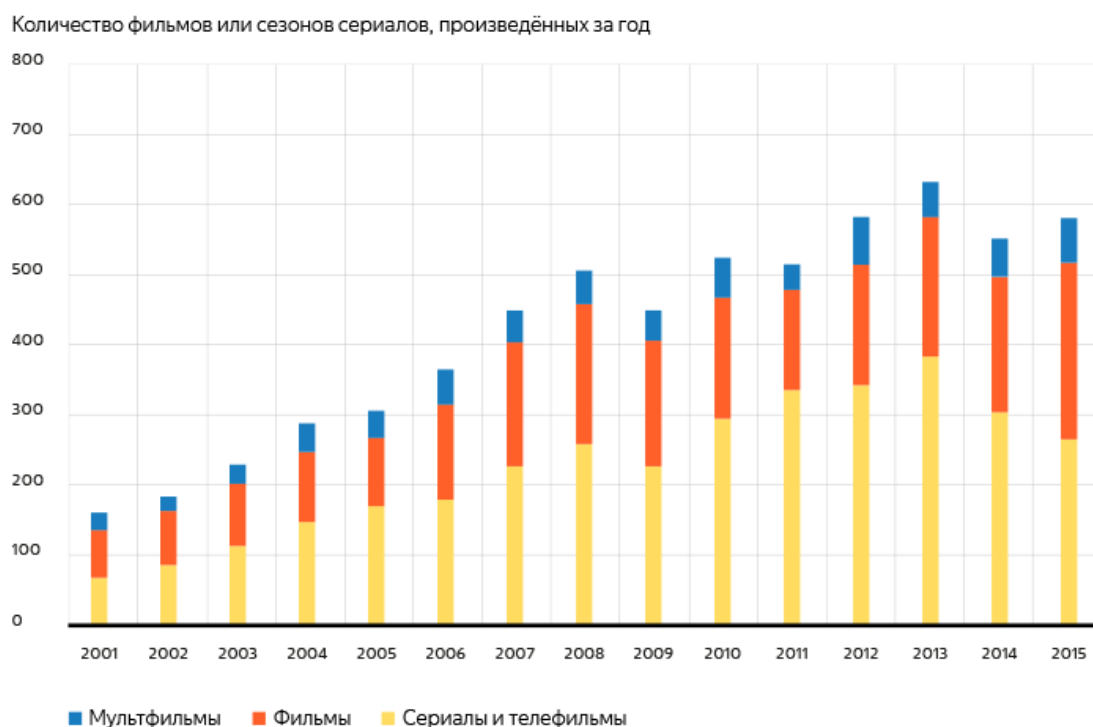
Выполнил:	1
1. Основные термины и определения.....	3
2. Введение.....	4
3. Ход работ	5
4. Сравнение библиотек beautiful soup, scrapy, selenium.....	7
Beautiful soup	7
Scrapy	7
Selenium	8
5. Сравнение фреймворков Django, Flask, Vue.js.	8
Django	8
Flask	8
Vue.js	8
6. Сравнение баз данных Firebase, MongoDB, SQLite	9
Firebase	9
MongoDB	9
SQLite	9
7. Архитектура программной системы	9
8. Модель монетизации сервиса	12
9. Описание функциональных и нефункциональных требований к программному проекту.....	13
7.1 Требования к веб сервису	13
7.2 Требования к подсистеме сбора данных	14
7.3 Интерфейс пользователя	14
7.4 Базы Данных	14
10. Разработка главной страницы	14
11. Разработка страницы авторизации/регистрации	18
12. Разработка страницы личного кабинета.....	19
13. Разработка страницы добавления объекта	21
14. Разработка страницы оплаты.....	23
15. Репозиторий	24
16. Список источников:.....	24

1. Основные термины и определения

- **Парсинг** - выделение нужной информации из всей.
- **Скраппер** - программа для автоматического сбора данных с веб сервера и последующего парсинга.
- **Краулер** - программа для автоматического перемещения рекурсией по веб-страницам.
- **API** - интерфейсы прикладного программирования (application programming interfaces), предлагающие удобные средства для обмена информацией между различными приложениями, программами.
- **HTML** - формат файла, в котором представлены коды веб-сайтов.
- **HTTP** - протокол передачи данных.
- **URL** - специальная форма, обозначающая адрес веб-сайта.
- **Фреймворк** - “каркас” для объединения различных компонентов программы.
- **Python** - язык программирования. Основной для нашего проекта
- **Dart** – C-подобный язык программирования
- **Flutter** – фреймворк написанный на языке Dart
- **Django** - фреймворк, написанный на Python.
- **Firebase** – дочерняя компания Google, занимающаяся облачными серверами и хранилищами
- **Cloud Firestore** – облачная база данных Firebase
- **Firebase Storage** – облачная база данных Firebase, удобная для хранения фото и видео
- **Firebase Authentication** - облачная база данных Firebase для хранения данных пользователей. Также предоставляет сервис для обработки авторизаций и логинов
- *Используемые Библиотеки:*
 - **Firebase core** - библиотека для Flutter, для работы с продуктами Firebase
 - **Firebase storage** - библиотека для Flutter, для работы с Firebase Storage
 - **Cloud firestore** - библиотека для Flutter, для работы с Firebase Cloud Firestore
 - **Firebase auth** - библиотека для Flutter, для работы с Firebase Authentication
 - **Flutter login** - библиотека для Flutter, помогающая создать окно регистрации/входа на сайт
 - **Photo view** - библиотека для Flutter, для лучшей работы с фотографиями
 - **Image picker** - библиотека для Flutter, создающая инструмент, который позволяет пользователю сайта загружать на него различные файлы
 - **Mime** - библиотека для Flutter, для распознавания типа файла по его битовой записи
 - **Flutter credit card** - библиотека для Flutter, создающая форму для ввода данных банковской карты
 - **Url launcher** - библиотека для Flutter, позволяющая работать с внешними ссылками
 - **Google.cloud** - библиотека для Python, для работы с серверами Google (в том числе Firebase)
 - **Beautiful soup** - библиотека для Python, помогающая в скрапперах.
 - **NumPy** - библиотека для Python, помогающая с математикой.
 - **Pillow** - библиотека для Python, помогающая с обработкой изображений.
 - **Scrapy** - библиотека для Python, помогающая в скрапперах.
 - **Selenium** - библиотека для Python, помогающая в скрапперах.
 - **Requests** - библиотека для Python, помогающая отправлять HTTP - запросы.
 - **Urllib2** - библиотека для Python, помогающая в открытии URL - адресов.

2. Введение

Несмотря на то, что количество стартапов в наше время растет с невиданной до этого скоростью, нашей команде не удалось найти ни одного проекта, занимающегося разработкой платформы для поиска и подбора площадок для фото- и киносъемок. Однако данная область в России, по данным различных исследований, неуклонно развивается. Растет как количество высокобюджетных профессиональных фильмов, так и количество малобюджетных авторских и короткометражных фильмов. Также благодаря развитию интернета стало сниматься в разы больше клипов и видео.



Таким образом, востребованность и актуальность данного проекта не подлежит сомнениям.

Для создания платформы были выделены основные задачи:

- 1) Создание Веб краулеров и Веб скрапперов
- 2) Создание основной системы и интерфейса (разработка веб сайта)
- 3) Создание баз данных пользователей и площадок

Вся работа происходила в несколько этапов. Первый этап – обучение. Были изучены различные справочные материалы для сравнения и освоения различных методов, с помощью которых выполнялись поставленные задачи. Таким образом к КТ1 выполнены следующие задачи:

- 1) Изучение книги Райана Митчелла “Скраппинг веб-сайтов с помощью Python”
- 2) Сравнение библиотек Python для скраппинга данных
- 3) Выбор сайтов, с которых будет проводиться скраппинг данных.
- 4) Сравнения инструментов и фреймворков, выбор конкретного.
- 5) Создание прототипа сайта в сервисе «Balsamiq Wireframes».
- 6) Создание и тестирование HTML-проекта в образовательных целях.

После первого этапа команда приступила непосредственно к самой разработке. К КТ2 были выполнено следующее:

- 1) Создание веб скрапперов
- 2) Сбор данных с помощью скрапперов
- 3) Создание страницы регистрации пользователей на сайте.
- 4) Создание страницы авторизации пользователей на сайте.

К КТ3 были завершены следующие части:

- 1) Переработка страницы регистрации и авторизации на другом фреймворке
- 2) Добавление подтверждения регистрации по электронной почте
- 3) Добавление начальная страница сайта со списком всех локаций
- 4) Добавление фильтров для поиска локаций
- 5) Добавление возможности авторизованного пользователя добавлять локации
- 6) Добавление личного кабинета
- 7) Добавление возможности выхода из аккаунта
- 8) Создание базы данных локаций
- 9) Создание базы данных с фотографиями локаций
- 10) Создание базы данных пользователей
- 11) Создание скрипта позволяющего выгрузить накопленные данные о локациях в новые базы данных
- 12) Созданы прототипы рекомендательных систем

3 Ход работ

В первый этап была прочитана и изучена книга Райана Митчелла “Скраппинг веб-сайтов с помощью Python”. В ней описываются три различные библиотеки python, которые можно использовать для веб скраппинга, их дальнейшее описание и сравнение находится ниже в пункте 4. Однако, в итоге, была выбрана библиотека BeautifulSoup. Также часть команды занималась изучением фреймворков для разработки сайта. Описание и сравнение фреймворков находятся ниже в пункте 5. После долгих сомнений был выбран фреймворк Django, после чего мои коллеги приступили к детальному изучению данного фреймворка.

Во время второго этапа команда приступила к разработке. Мною было создано 2 скраппера, обработавших сайты Cian.ru и Gdeetotdom.ru, и собравших суммарно более 500 объявлений, которые были использованы в дальнейшем для создания баз

данных. Другие участники также создали еще 4 парсера, обработавших 4 других сайта. Единственную проблему вызвал сайт Яндекс Недвижимость из-за проблем с форматом, однако мои коллеги быстро с этим справились. Всего было собрано более 3 тысяч объявлений. Остальная команда занялась созданием сайта. Ими были созданы страницы регистрации и входа на сайт. Однако во время разработки возникло множество проблем: специфические ошибки, информацию по которым не удавалось найти, ошибки из-за конфликтов версий Django двух моих коллег и т. д. Устарелость большинства справочных материалов также негативно сказывалась на ходе работы.

Во время третьего этапа, из-за возникших опасений по поводу большого количества ошибок, связанных с разработкой сайта, команда решила бросить все свои силы на занятие сайтом. Довольно быстро было принято решение изменить все кардинально и поменять фреймворк. Командой было решено перейти на Flutter – фреймворк, написанный на Dart – С-подобном языке программирования от Google. Основным средством в Flutter – виджеты, на которых строятся все части сайта. Виджеты и варианты их применения будут описаны далее.

Примечательно, что Flutter позволяет переделать сайт в мобильное приложения для Android или IOS за короткий промежуток времени.

Сменив фреймворк дело пошло куда быстрее. Изучение языка и переработка уже созданных моделей на языке Django заняло всего неделю. Далее за несколько недель была сделана основная страница со списком всех локаций, добавлена возможность искать локации по фильтрам, добавлен личный кабинет и страница, через которую пользователь может загрузить свою локацию на сайт. Также добавлена отдельная страница для оплаты выставления объявления с помощью банковской карты. До выхода приложения на рынок эта форма останется непривязанной к каким-либо сервисам оплаты и является исключительно декорацией, для демонстрации всех возможностей сайта.

Однако вернемся на пару недель назад, когда команда только заново создала страницу с регистрацией и авторизацией. После этого стало понятно, что откладывать работу над базами данных нет никакого смысла, ведь если они появятся уже сейчас – работать над сайтом станет намного проще, ведь его отдельные его части можно будет тестировать ровно в том виде, в котором они появятся на релизе. Изучив различные варианты, команда решила остановиться на продукции дочерней компании Google – Firebase. Такой выбор был обоснован тем, что Firebase мог бесплатно предоставить все необходимые нам функции, был очень дружелюбен для работы с Dart (что легко понять, так как большинство продуктов Google друг с другом хорошо коммуницируют). После этого был создан скрипт на Python, который перенес все данные собранные скрапперами в базу данных Cloud Firestore. Все картинки (которые скрапперами были получены в виде ссылок) тем же скриптом были загружены в базу данных Firebase Storage уже в виде файлов. Для картинок была выбрана другая база данных, так как Storage – база данных, созданная для хранения картинок, что позволило избежать лишних проблем во время разработки. Базу данных пользователей также было решено хранить в специализированной базе данных – Firebase Authentication. Данная база данных не только привлекательна тем, что в ней удобно хранить логины и пароли, она также поддерживает систему регистрация/логин. То есть Authentication самостоятельно проверяет корректность паролей, наличие пользователя в базе и т. д. Так же

система сама хэширует пароли пользователей, что позволяет сделать сайт более безопасным. В качестве бонуса в Authentication есть возможность рассылки почты, что позволяет создать проверку e-mail-ов во время регистрации, а также сбрасывать пароли.

Также команда задумалась о хостинге сайта. Изначально было предложено использовать Github Pages, однако, изучив данный вопрос, было обнаружено, что Firebase также предоставляет и хостинг, а так как вся работа, связанная с базами данных, уже была проделана, то запустить сайт на сервере Firebase не составляло никакого труда. Так и было сделано.

Была начата разработка рекомендательных систем.

Сам сайт расположен по ссылке:

kino-locations.web.app

4 Сравнение библиотек beautiful soup, scrapy, selenium.

Beautiful soup

Эта библиотека может извлекать данные из файлов HTML и XML. Но требует определенных модулей для работы, а именно:

- 1) Необходима библиотека для отправки запроса на веб-сайт, потому что сам BeautifulSoup не может сделать запрос на конкретный сервер. Библиотеки Requests или urllib2 помогут сделать запрос к серверу.
- 2) После загрузки данных HTML или XML на наш локальный компьютер BeautifulSoup требуется внешний анализатор для анализа загруженных данных. Наиболее известные парсеры - это XML-parser lxml, HTML-parser lxml, HTML5lib, html.parser.

Когда речь идет о небольшом проекте, или о низкоуровневом сложном проекте BeautifulSoup может выполнить задачу довольно хорошо. Это помогает нам поддерживать наш код простым и гибким.

Scrapy

Scrapy – это платформа для совместной работы с открытым исходным кодом для извлечения данных с веб-сайтов. Его производительность невероятно высока, и это одна из самых мощных доступных библиотек. Одним из ключевых преимуществ scrapy является то, что он построен на основе Twisted, асинхронной сетевой структуры, что означает, что scrapy использует неблокирующий механизм при отправке запросов пользователям.

Одним из самых больших преимуществ Scrapy является то, что мы можем очень легко перенести наш существующий проект в другой проект. Поэтому для больших / сложных проектов Scrapy - лучший выбор для разработки.

Selenium

Selenium предназначен для автоматизации тестирования веб-приложений. Он позволяет разработчику писать тесты на нескольких популярных языках программирования, таких как C#, Java, Python, Ruby и другие. Эта среда разработана для автоматизации браузера

5 Сравнение фреймворков Django, Flask, Vue.js.

(Flutter в сравнение не вошел, так на момент сравнения фреймворков не рассматривался как кандидат. Также его преимущества подробно описаны выше, а методы разработки - ниже)

Django

Главное преимущество – наличие большего количества источников информации и относительная легкость в изучении и использовании. Также он несколько старше «конкурентов», что влечет за собой полноту функциональности. В связи с этим и была выбрана именно это веб-инфраструктура. Основная ее задача - создание сложных веб-сайтов на базе баз данных. Она имеет библиотеку с большим количеством функций и полной и надежной документацией, что позволяет быстрее писать код и находить ошибки. Дизайн приложений, разработанных в этой среде лаконичен. На ее базе написаны такие популярные сервисы, как Coursera и Instagram

Flask

Flask – это относительно молодой фреймворк, предлагающий основные функции веб-приложения. Инструмент хорошо подходит для разработки простых приложений с небольшим функционалом. Он имеет возможность простой интеграции базы данных, поддерживает визуальную отладку, и дает возможность легко поддерживать проекты. Специалисты также утверждают, что он лучше многих подходит именно для легкой кодовой базы. Его тоже используют крупные компании для реализации таких сервисов, как Netflix, Reddit и другие.

Vue.js

Vue.js – это фреймворк, который, в отличие от «Django» и «Flask», работает по принципу генерации страниц на стороне клиента. Вследствие этого Vue.js требует больше времени на начальную загрузку, но этот минус перекрывается скоростью последующего переключения между страницами. Vue.js имеет достаточно полную документацию, но, к

сожалению, его сообщество значительно меньше, чем у Django и Flask, что означает большие затраты времени на освоение материала.

6 Сравнение баз данных Firebase, MongoDB, SQLite

Firestore

Firestore – это облачная база данных для хранения информации и синхронизации данных в реальном времени. Благодаря этому БД может легко уведомить об изменении все устройства за короткий период времени. Это также дает гибкость для доступа к данным с любого устройства. Благодаря тому, что данные размещаются в облаке, обслуживание сервера не осуществляется. Firestore можно использовать в автономном режиме.

MongoDB

MongoDB - документно-ориентированная система управления базами данных, не требующая описания схемы таблиц. Как и Firestore, является одним из примеров NoSQL-систем с JSON-подобными документами.

MongoDB позволяет располагать базы данных на нескольких физических серверах, и эти базы легко могут обмениваться содержимой информацией между собой и сохранить целостность.

SQLite

SQLite - автономная база данных без сервера SQL. Вся база данных хранится в одном файле, что облегчает перемещение.

Если приложение использует SQLite, связь между БД и программой производится с помощью функциональных и прямых вызовов файлов, содержащих данные, что повышает скорость и производительность операций.

В качестве базы данных для проекта, благодаря размещению данных в облаке, автономности и возможности доступа с любого устройства, мы решили использовать Firestore.

7 Архитектура программной системы

Архитектура системы состоит из следующих частей:

- **Подсистема сбора данных**

Сбор данных осуществляется парсерами, которые пишутся отдельно для каждого сайта. Не со всех сайтов есть возможность сбора данных машинным образом.

Также фотографии, адреса локаций, контакты владельцев спрятаны в разных местах.

- **Хранилище данных**

Все данные, которые используются сайтом, хранятся в трех базах данных:

- *База данных пользователей* -- содержит информацию о каждом авторизованном посетителе сайта. То есть логин, пароль, электронная почта и остальная дополнительная информация. Все вышеперечисленное заполняются пользователем вручную.
- *База данных локаций* -- содержит информацию о локациях. То есть адрес, цена, площадь и т. д. В нее данные могут попадать двумя способами. Первый из них – результат парсинга разных сайтов. Второй – через специальную форму добавления объекта киноплощадки.
- *База данных изображений локаций* – содержит фотографии локаций. Данные в базу данных попадают аналогично предыдущему пункту.

- **Модель для рекомендательной системы**

Для каждого клиента сайта будет предложена возможность отсортировать все локации по различным фильтрам, например по сроку аренды, цене, адресу, площади и т. д.

- **Программа бизнес-логики**

Сравнение микросервисной и монолитной архитектуры приложения:

- **Монолитные приложения** – приложения, построенные как единое целое, где вся логика по обработке запросов помещается внутрь одного процесса. Разумеется, монолиты могут иметь модульную структуру — содержать отдельные классы, функции, namespace (в зависимости от выбранного языка программирования). Но связи между этими модулями настолько сильны, что изменение каждого из них неизбежно отражается на работе приложения в целом.
 - **Достоинства:**
Большим преимуществом монолита является то, что его легче реализовать. В монолитной архитектуре вы можете быстро начать реализовывать свою бизнес-логику, вместо того чтобы тратить время на размышления о межпроцессном взаимодействии.
Монолитные приложения просты в развертывании и легко масштабируемы. Для развертывания вы можете использовать скрипт, загружающий ваш модуль и запускающий приложение.
 - **Недостатки:**
Даже небольшое изменение для одной из бизнес-функций будет приводить к необходимости сборки и развертывания новой версии всего приложения.

Масштабировать приходится все приложение целиком, даже если это необходимо отдельно взятому компоненту с наименьшей производительностью.

Отказ одного модуля чаще всего сказывается на всей работе в целом в силу тесных связей внутри приложения.

Любое изменение в базе данных будет сказываться на всем приложении и требовать значительных изменений кода.

- **Микросервисная архитектура** – наиболее распространенный подход к разработке программного обеспечения, когда приложение разбивается на небольшие автономные компоненты (микросервисы) с четко определенными интерфейсами. Каждый из сервисов отвечает за конкретную бизнес-задачу, имеет собственное хранилище данных и общается с другими сервисами через простые API-интерфейсы для решения более сложных задач. Микросервисы в значительной степени получили свое название из-за того, что сервисы здесь меньше, чем в монолитной среде. Каждый сервис развертывается самостоятельно.
 - **Достоинства:**
Микросервисы меньше, и благодаря этому их легче понять и проверить. Меньшие размеры помогают, когда речь идет о времени компиляции, времени запуска и времени, необходимом для выполнения тестов.
Можно расширять только те сервисы, которые в этом нуждаются, то есть сервисы с наименьшей производительностью, оставляя работать остальные части системы на менее мощном оборудовании.
Отказ одного сервиса не приводит к остановке системы в целом.
 - **Недостатки:**
Распределенная система имеет свою сложность: в ней вам приходится иметь дело с частичным отказом, более затруднительным взаимодействием при тестировании (тесты E2E), а также с более высокой сложностью при реализации взаимодействия между сервисами.
Существуют эксплуатационные накладные расходы, а множество микросервисов сложнее в эксплуатации, чем несколько экземпляров сигнального монолита

Итог: предварительно мы с командой решили выбрать микросервисную архитектуру. В целом процессы в различных частях проекта связаны максимально просто.

Есть 4 основных части нашего проекта: скрапер, база данных, сайт и рекомендательная система. Скрапер, как упоминалось выше, просто собирает нужную информацию и передает ее в базу данных, связывать его с сайтом и рекомендательной системой не нужно. Сайт связан с базой данных и рекомендательной системы, от нее он получает всю необходимую информацию о локациях (картинки, адреса, контакты, цена). Если

происходит регистрация клиента, то это отображается на сайте и на базе данных. Обращение к какой-либо странице на сайте может повлиять на рекомендательную систему, которая берет всю информацию из базы данных.



8 Модель монетизации сервиса

Нашей командой было обдумано несколько сценариев монетизации:

- 1) Объявления выставляются бесплатно, менеджеры, находящиеся в поисках локаций, оплачивают ежемесячную/ежегодную подписку на постоянно дополняющийся сервис с локациями. Этот способ имеет ряд минусов. Во-первых, менеджеры будут иметь неуверенность при регистрации, так как сначала они платят, а потом получают доступ к просмотру локаций. Этаким кот в мешке. Во-вторых, те, кто выставляют локацию должны либо так же купить подписку, либо тоже отправить свои данные неизвестно куда. В-третьих, платная подписка подразумевает отсутствие дополнительной внутренней рекламы, которая так же может приносить доход. Это повлечет дефицит продюсеров киноплощадки, и по итогу придется скрапить данные все чаще и со всего большего количества сервисов, что со временем сделает базу неактуальной и засоренной локациями незаинтересованных в таком роде сдачи арендодателей.
- 2) Объявления выставляются бесплатно, просматриваются бесплатно, но получение контактов арендодателя имеет цену. Тут минусы еще более необычные. Во-первых, существуют технологии поиска в интернете по картинке, их уже давно внедрили такие компании, как Яндекс, Google и прочие. То есть, найдя интересующий объект, пользователь может просто сохранить или сделать скриншот изображения и добраться до контактов пользователей, минуя оплату в сервисе. Это лишает нас какой-либо монетизации, кроме показа контекстной рекламы. Эту проблему можно решить, используя необратимые фильтры на изображениях, но это исказит изображения и может сделать объявление менее привлекательным. Также, это потребует больших мощностей и может значительно замедлить выгрузку объявления.
- 3) Объявления выставляются платно, становятся доступны всем. Такая модель лишена вышеописанных минусов и является довольно популярной в различных сервисах (например, Авито). Также в ней рационально добавить возможность платного

продвижения локации за небольшую стоимость и нет факторов, противоречащих наличию дополнительной рекламы.

По итогу, было принято решение выбрать третий тип монетизации, так как он показался наиболее прибыльным и рациональным.

9 Описание функциональных и нефункциональных требований к программному проекту

7.1 Требования к веб сервису

1 Роли

- a. Простой пользователь (может просматривать локации)
- b. Авторизованный пользователь (может добавлять локации после оплаты)

2 Сценарии работы

- a. Простой пользователь
 - i. Регистрация
 - ii. Отображение всех объектов в виде списка
 - iii. Поиск объектов
- b. Авторизованный пользователь
 - i. Авторизация (логин, пароль)
 - ii. Кабинет пользователя
 - iii. Отображение объектов только этого пользователя в виде списка на карте
 - iv. Добавление объекта
 - 1. Название (описание)
 - 2. Имя пользователя
 - 3. Цена
 - 4. Адрес
 - 5. Этаж
 - 6. Площадь
 - 7. Контакты (добавляются автоматически)

3 Функциональные требования

Система должна поддерживать

- i. Регистрацию
- ii. Авторизацию
- iii. Отображение кабинета пользователя
- iv. Просмотр всех локаций
- v. Поиск локаций с возможностями фильтрации и сортировки
- vi. Просмотр своих локаций
- vii. Возможность добавление локации
- viii. Возможность взимать плату за использование услуги добавления локации

7.2 Требования к подсистеме сбора данных

Сбор названия, адреса, фотографий, дополнительной информации о локации, контакта владельца с сайтов

7.3 Интерфейс пользователя

1. Возможность авторизации с помощью логина и пароля
2. Отображение информации о подключенных устройствах
3. Отображение информации о носимых устройствах
4. Возможность сортировки по цене, размеру, адресу, сроку аренды.

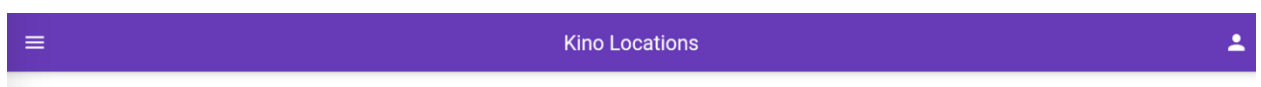
7.4 Базы Данных

- а. Пользователя
 - i. Логин
 - ii. Пароль
 - iii. Электронная почта
 - iv. Имя (по умолчанию)
 - v. Фамилия (по умолчанию)
 - vi. Дополнительная информация (по умолчанию)
- б. Локации
 - i. Название (описание)
 - ii. Имя арендодателя (пусто в случае скраппинга)
 - iii. Цена
 - iv. Адрес
 - v. Площадь
 - vi. Этаж / Общее число этажей
 - vii. Контакт (данные пользователя, разместившего локацию, или ссылка на исходный сайт, в случае скраппинга)

10 Разработка главной страницы

Функционал страницы включает в себя скроллинг карточек локаций, на которых расположена информация из баз данных локаций и изображений, возможность фильтрации и отбора киноплощадок по количеству комнат/общей площади и т. п., возможность попасть в личный кабинет или авторизоваться.

Для разработки были изучены принципы работы статических (StatelessWidget) и динамических (StatefulWidget) виджетов. Статическим, например, является виджет MaterialApp, отвечающий за верхнюю строку на сайте:



Примером динамического виджета является виджет, содержащий фильтры локаций:

Количество комнат

1 комната	<input checked="" type="checkbox"/>
2 комнаты	<input type="checkbox"/>
3 комнаты	<input checked="" type="checkbox"/>
4 комнаты или более	<input type="checkbox"/>

Площадь

Цена (за сутки)

Здесь используются “двойные” ползунки – виджет `RangeSlider`, и вертикальный список `checkbox`-ы – виджет `CheckboxListTile`. Эти виджеты являются динамическими так как пользователь может изменять их состояние (отмечать нужные пункты или двигать ползунков).

Архитектура страницы (`HomePage`) представляет из себя следующее:

Так как начальная страница включает в себя виджет с фильтрами, то сама страница является `Stateful` виджетом. Код `Stateful` виджета отличается от `Stateless` тем, что `Stateless` является самым обычным виджетом, включающим в себя всё своё наполнение, в то время как в `Stateful` всего лишь передается его состояние (пример кода)

`Stateless`:

```
class Auth extends StatelessWidget { # класс
# ...
  @override
  Widget build(BuildContext context) {
    # основной код наполнения страницы
  }
}
```

`Stateful`:

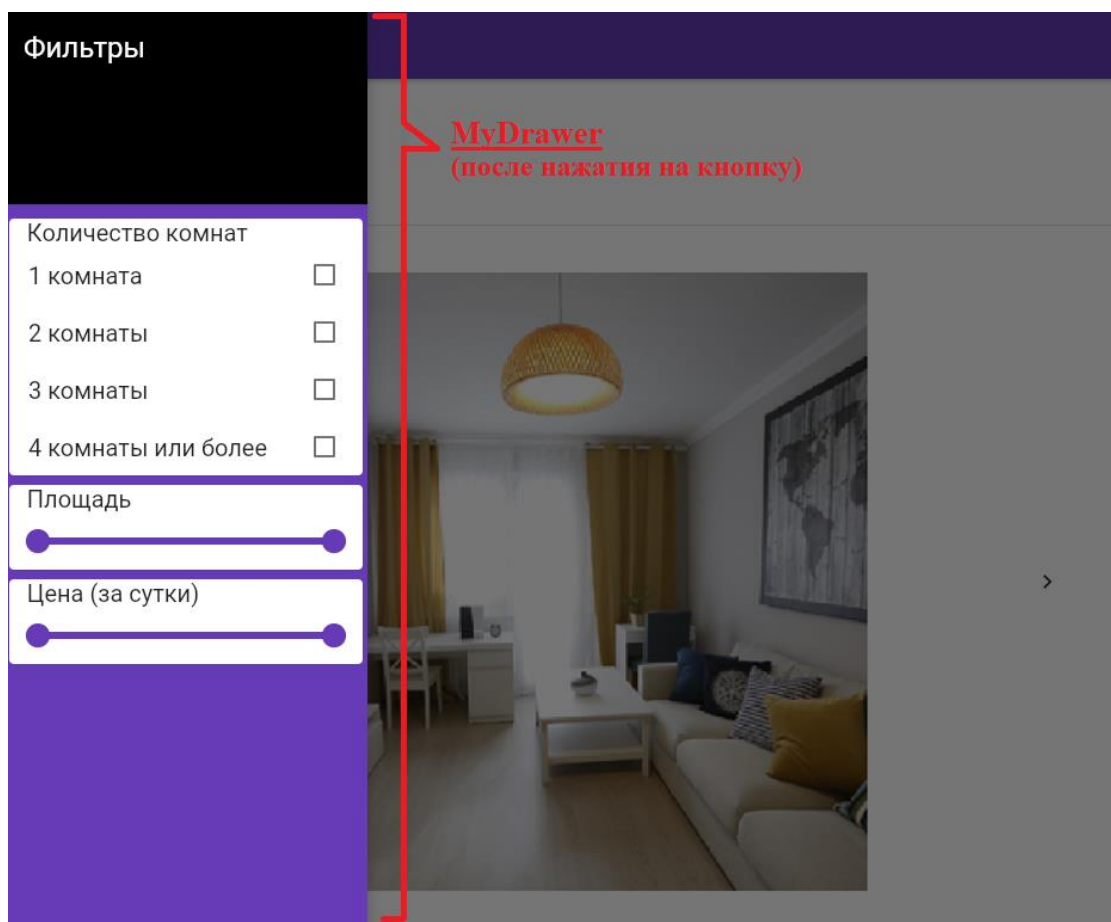
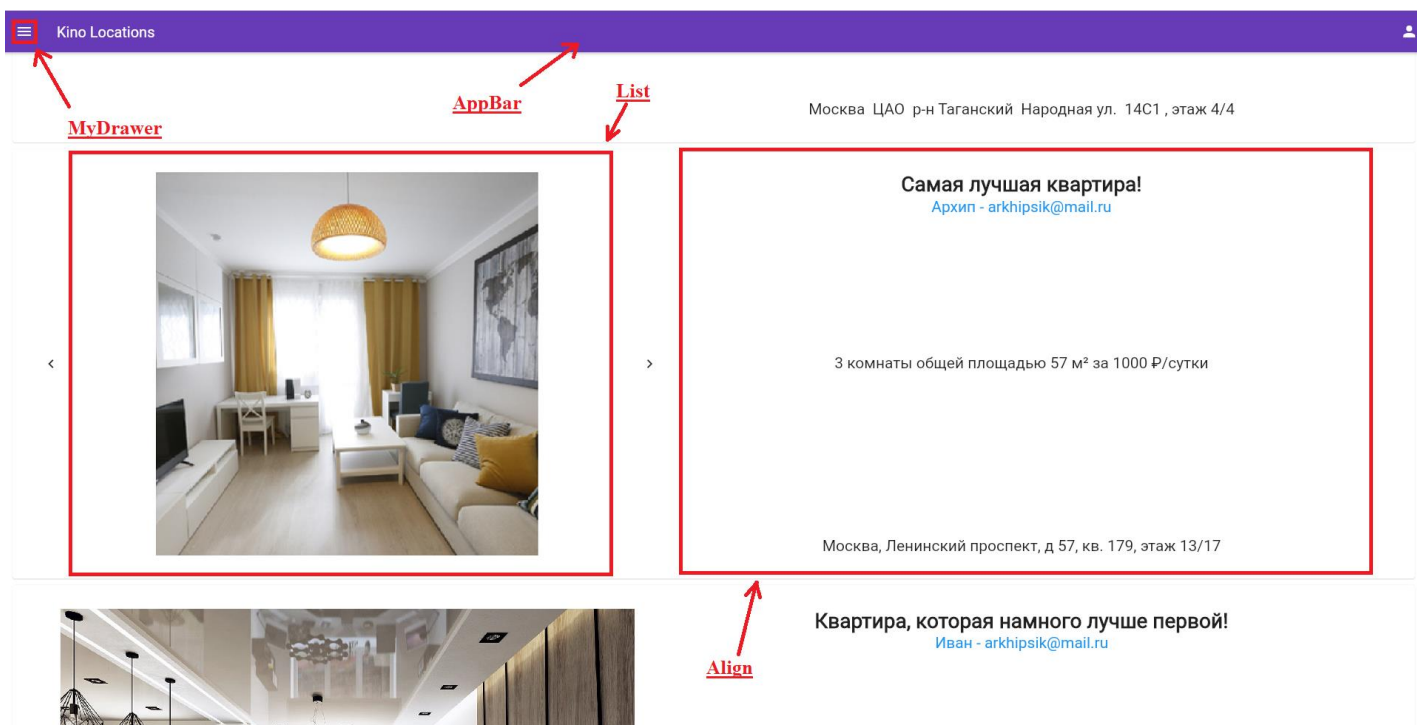
```
class HomePage extends StatefulWidget { # класс
# внешние переменные
  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> { # состояние
# ...
  @override
  Widget build(BuildContext context) {
    # основной код наполнения страницы
  }
}
```

Далее состояние возвращает виджет Scaffold – по сути сама страница. Scaffold же содержит в себе три вещи: AppBar – виджет для полосы сверху, MyDrawer – штука слева, которая выезжает по кнопке и содержит в себе фильтры, и виджет Center – по сути просто обертка, которая содержит основную часть сайта.

- AppBar – содержит кнопки перехода в личный кабинет или на страницу авторизации. Также название сайта и кнопку для MyDrawer
- MyDrawer – виджет (не из стандартной библиотеки – он написан вручную) который содержит в себе фильтры – каждый фильтр состоит из виджета Card. Далее фильтр комнат из виджета CheckboxListTile (см. выше), виджеты цены и площади из виджетов RangeSlider (см. выше). Все карточки обернуты в виджет Column, чтобы они были расположены вертикально.
- Center – стандартный виджет, который центрирует на экране своего ребенка – виджет ListView, который похож на виджет Column, однако не вмещает на экран всех своих детей, а выделяет каждому своему ребенку необходимое ему место и позволяет пользователю листать его (в данном случае в вертикальном направлении). ListView содержит карточки (виджет Card) с данными о локациях. Каждая карточка содержит в себе виджет Row (аналог Column, но горизонтальный). В ряд помещаются List – со списком фотографий, виджеты с кнопками прокрутки фото и Align – виджет который располагает по экрану своих детей в соответствии с настройками (в данном случае это тексты с описаниями локаций (кстати, любой текст также является отдельным виджетом)). При этом ListView обернут в виджет FutureBuilder – виджет для асинхронного программирования. Это нужно для того, чтобы все локации подгружались независимо, а пользователь мог увидеть сайт не загружая всю нашу базу данных. FutureBuilder контактирует с базами данных Cloud Firestore и Firebase Storage при помощи следующих библиотек:

```
firebase_core: ^1.1.0
cloud_firestore: ^1.0.7
firebase_storage: ^8.0.5
```

11 Разработка страницы авторизации/регистрации

Страница регистрации имеет весь необходимый функционал:

При переходе на страницу авторизации у пользователя запрашивается электронная почта и пароль. Тут же имеется опция восстановления пароля и кнопка «зарегистрироваться». При выборе опции регистрации нового пользователя появляется дополнительное окно «подтверждения пароля»:

После нажатия кнопки «зарегистрироваться» на почту будет выслана ссылка-подтверждения регистрации и валидности электронной почты. Проходя по ссылке, пользователь попадает на главную страницу сайта. При попытке сбросить пароль, вам также придет ссылка-восстановление на вашу почту.

Архитектура страницы имеет следующий вид:

Все состоит из Stateless виджета, который содержит в себе Scaffold (см. выше). В нем лежат виджеты AppBar (см. выше) и FlutterLogin – виджет из неофициальной библиотеки:

```
flutter_login: ^2.0.0-nullsafety.0
```

Этот виджет и создает прекрасное окно авторизации и логина. Также начальный Stateless виджет содержит пару функций:

```
String? emailValidator(String? value) {  
  # ...  
}  
  
String? passwordValidator(String? value) {  
  # ...  
}  
  
Future<String?>? _authUser(LoginData data) {  
  # ...  
}  
  
Future<String?>? _regUser(LoginData data) {  
  # ...  
}  
  
Future<String?>? _recoverPassword(String email) {  
  # ...  
}
```

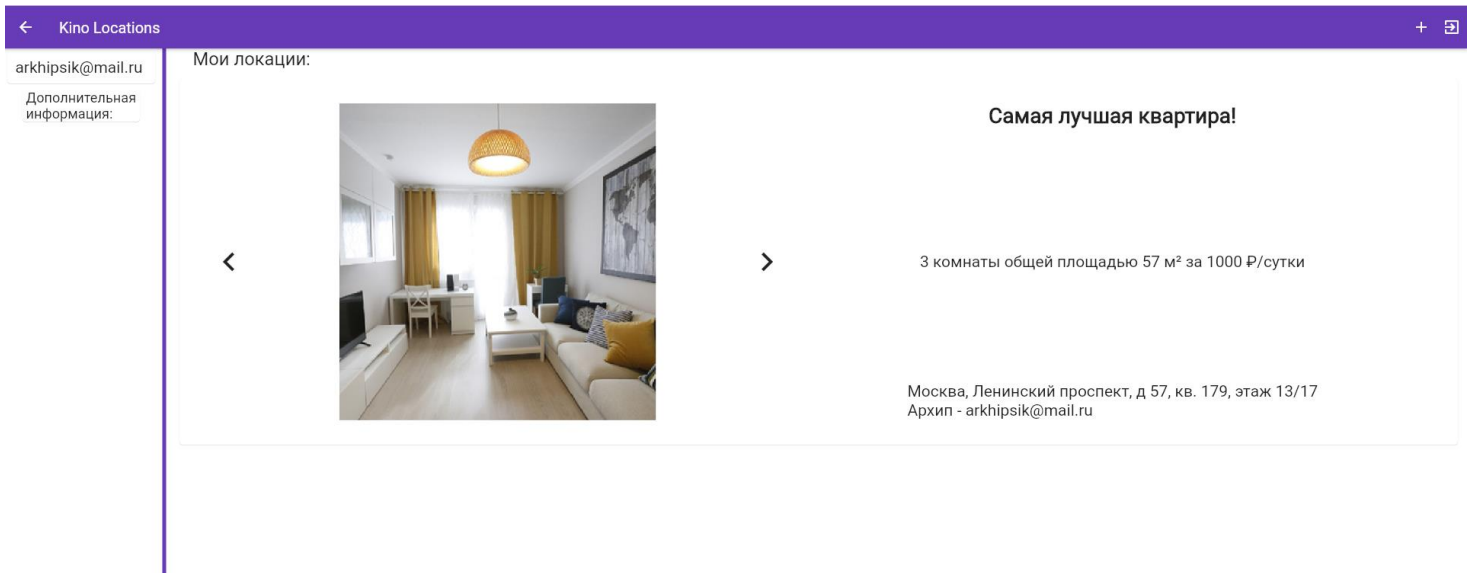
Данные принимают данные из виджета FlutterLogin, обрабатывают их нужным образом и связываются с базой данных Firebase Authentication при помощи библиотеки

```
firebase_auth: ^1.1.3
```

12 Разработка страницы личного кабинета

В личном кабинете отображается информация о данном пользователе и локациях, загруженные на сайт этим пользователем.

Также на странице присутствует кнопка выхода из аккаунта и кнопка, ведущая на страницу добавления новой локации от своего лица.



Архитектура страницы имеет следующий вид:

Страница опять же состоит из виджета Scaffold, который включает в себя виджеты AppBar (см. выше) и Row.

Row – содержит в себе 3 виджета: Column, в который попали виджеты Card с различной информацией о пользователе; VerticalDivider – виджет разделитель (вертикальная фиолетовая линия) и виджет FutureBuilder, который содержит в себе ListView. Сам ListView практически такой же, как и на домашней странице, однако к подгружаемым локациям применен один фильтр – значение “Contact” каждой локации должно совпадать с почтой, под которой зарегистрирован пользователь:

```
User? user = widget.auth.currentUser; # user – данные о текущем пользователе
String _email = '';
if (user != null) {
  String? tmpEmail = user.email;
  if (tmpEmail != null) _email = tmpEmail;
}
# ...
return ListView.builder(
  if (loc['contact'] != null &&
    loc['contact'] != '' &&
    loc['contact'] == _email) {
    # ...
```

(Во время загрузки объекта на сайт пользователем значение локации “Contact” в базе данных автоматически заполняется электронной почтой данного пользователя. В случае, если локация получена путем скраппинга – данное поле заполняется ссылкой на объявление на сайте, с которого была получена локация.)

13 Разработка страницы добавления объекта

На странице есть поля для ввода названия локации, имени, посуточно цены, адреса, этажа, количества этажей в здании, количества комнат и общей площади. Также возможность добавления изображения в галерею и его удаления в случае неверной загрузки.

После нажатия на кнопку добавления объекта в правом нижнем углу, пользователя переадресовывает на страницу оплаты публикации, где ему будет предложено ввести данные банковской карты (информация об этой странице в пункте 12).

The screenshot shows a mobile application interface titled "Kino Locations". It features a form for adding a new location. The form fields are as follows:

- Имя:** Василий Пупкин
- Цена:** 10000
- Адрес:** Москва, Покровский бульвар, д. 11
- Этаж:** 1
- Всего этажей:** 10
- Количество комнат:** (Field with red error text: "Укажите количество комнат в локации", "Пожалуйста, заполните это поле")
- Площадь:** (Field with red error text: "Площадь очень большая", "Пожалуйста, введите число")

Below the form is a photo gallery showing a modern building interior. At the bottom right is a purple "+" button for adding more photos. At the bottom left are buttons "Загрузить фото" and "Удалить".

Архитектура страницы имеет следующий вид:

Страница состоит из виджета Scaffold (см. выше), который содержит в себе следующие виджеты: AppBar (см. выше), FloatingActionButton – кнопка в правом нижнем углу, которая загружает указанные данные в базу данных (все происходит аналогично их загрузке из базы данных, об этом можно почитать в пункте 8) и отправляет пользователя на страницу с оплатой, и виджет Center, в котором лежит виджет Card, в котором находится виджет Form – форма, которую заполняет пользователь. В Form, лежит ListView, который содержит в себе виджеты TextFormField – каждый такой виджет – это строка, которую нужно заполнить. TextFormField состоит из трех частей: decoration (содержит в себе текст над полем для заполнения и текст-подсказка, появляющийся при нажатии на поле), onSave (функция отвечающая за то – что должно произойти при сохранении данных, в данном случае введенная строка сохраняется в словарь, который позже будет передан в базу данных) и validator (он проверяет корректность введенных данных и в случае их некорректности выдает под полем для заполнения инструкцию для пользователя как правильно заполнять данное поле).

```

TextFormField( # пример использования виджета TextFormField
  decoration: const InputDecoration(
    icon: Icon(Icons.attach_money),
    hintText: 'Укажите цену за сутки в рублях',
    labelText: 'Цена',
  ),
  onSave: (String? value) {
    if (value != null) loc['price'] = int.parse(value);
  },
  validator: (String? value) {
    if (value == '' || value == null)
      return 'Пожалуйста, заполните это поле';
    if (!isNumeric(value)) return 'Пожалуйста, введите число';
    return null;
  },
),

```

Последним элементом ListView является виджет Row, который содержит в себе двух детей ElevatedButton и TextButton – это кнопки, одна для загрузки фото, другая для их удаления (методы данных виджетов не отличаются функционалом, их различия только визуальные). TextButton для удаления обернута в виджет Visibility. Этот виджет позволяет скрывать или показывать своего ребенка в зависимости от условий. В данном случае кнопка “Удалить” скрывается, если счетчик количества загруженных фотографий равен 0. ElevatedButton при нажатии позволяет вам выбрать фото на вашем устройстве, для этого применяется класс PickedFile и функция imagePicker.

```

PickedFile? newImage = await imagePicker.getImage(source: ImageSource.gallery);

```

Данный класс и функция добавлены пользовательской библиотекой:

```

image_picker: ^0.7.5+2

```

Также при обработке фото возникли проблемы с передачей и определением расширения загруженного файла. Для определения расширения по байтовой записи файла была использована библиотека:

```

mime: ^1.0.0

```

14 Разработка страницы оплаты

Данная страница не несет в себе какого-либо функционала и служит лишь примером того, как можно выстроить экономическую модель. Вся страница является лишь украшением, а данные никак не обрабатываются и не сохраняются.

← Kino Locations

5757 **** * 5757

06/23

Василий Пу

Номер карты

MM / ГГ
06/23

CVV код

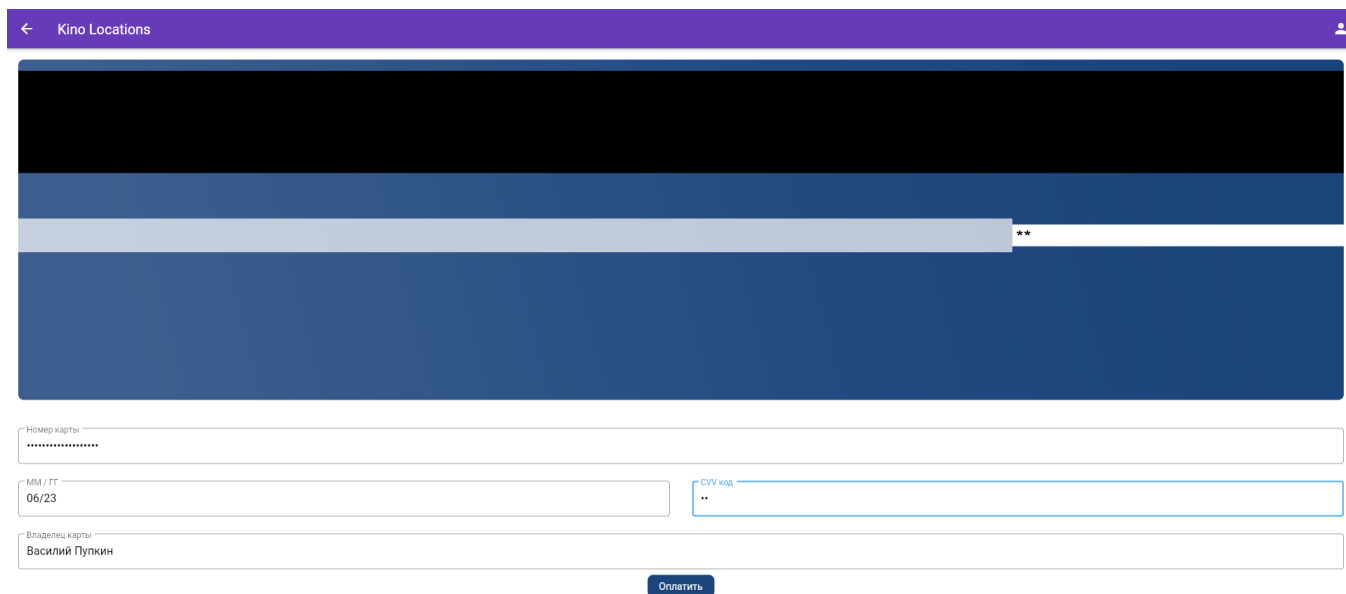
Владелец карты
Василий Пу

Оплатить

Архитектура страницы имеет следующий вид:

Аналогично странице с авторизацией, данная страница по сути является двумя большими виджетами из сторонней библиотеки (если не считать AppBar – про него см. выше, и кнопки “оплатить” – данная кнопка концептуально ничем не отличается от остальных кнопок на сайте и перебрасывает пользователя на начальную страницу). Первый – это виджет CreditCardWidget, он добавляет картинку банковской карты, которая заполняется по мере того, как вы заполняете форму. Второй – это виджет CreditCardForm, который является формой для заполнения. Оба виджета предоставлены библиотекой

```
flutter_credit_card: ^2.0.0
```



(Виджет CreditCardWidget преворачивает картинку с карточкой в зависимости от полей, которые мы заполняем)

15 Репозиторий

- Весь код сайта можно найти по ссылке:
github.com/57Ark/kino_project
- Все остальные части проекта находятся по ссылке:
github.com/57Ark/kino_project_materials
- Сам сайт расположен по ссылке:
kino-locations.web.app

16 Список источников:

[Сайт] Scrapy 2.4 documentation- <https://docs.scrapy.org/en/latest/>

[Сайт] Selenium with Python- <https://selenium-python.readthedocs.io/>

[Сайт] Python: Scrapy, Selenium, BeautifulSoup что лучше для парсинга веб сайтов-
<https://dev-gang.ru/article/python-scrapy-selenium-beautiful-soup-czto-luczshe-dlja-parsinga-veb-saitov-rmv3n1q3us/>

[Сайт] BeautifulSoup-
<http://wiki.python.su/%D0%94%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D0%B8/BeautifulSoup>

[Сайт] Scrapy Vs Selenium Vs BeautifulSoup for Web Scraping. - <https://medium.com/analytics-vidhya/scrapy-vs-selenium-vs-beautiful-soup-for-web-scraping-24008b6c87b8>

[Книга] Web Scraping with Python, Ryan Mitchell

[Книга] «Django 3 by Example» - Antonio Mele

[Книга] «Django crash course» - Daniel Roy Greenfield and Audrey Roy Greenfield

[Книга] «Flask by example» - Gareth Dwyer

[Книга] “Django 3.0 Практика создания веб-сайтов на Python” – Владимир Дронов

[Сайт] Celery- <https://docs.celeryproject.org/en/stable/getting-started/introduction.html>

[Сайт] Microservices vs Monolith: which architecture is the best choice for your business?- <https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/#:~:text=While%20a%20monolithic%20application%20is,as%20perform%20the%20specific%20functions.>

[Сайт] Flutter <https://flutter.dev/docs/development/ui/widgets>

[Сайт] Firebase Docs <https://firebase.flutter.dev/docs/overview>

[Сайт] Пользовательские библиотеки Flutter <https://pub.dev/packages>