МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка мобильного приложения для столовой «Восточный дракон»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-020.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____ Н.С. Силкина

Автор работы,
студент группы КЭ-404
_____ Н.В. Стецкий

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-404
Стецкому Никите Вадимовичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка кроссплатформенного мобильного приложения для столовой «Восточный дракон».
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Flutter Documentation. [Электронный ресурс] URL: https://flutter.dev/docs (дата обращения: 31.01.2024 г.).

3.2. Dart Documentation. [Электронный ресурс] URL: https://dart.dev/guides (дата обращения: 31.01.2024 г.).

3.3. Сайт кафе «Восточная столовая». [Электронный ресурс] URL: https://cafe-38130.business.site/ (дата обращения: 31.01.2024 г.).

3.4. Страница кафе «Восточный дракон» на 2gis. [Электронный ресурс] URL: https://2gis.ru/chelyabinsk/firm/70000001040636558 (дата обращения: 31.01.2024 г.).

4. Перечень подлежащих разработке вопросов

4.1. Изучить предметную область.

4.2. Провести обзор аналогичных приложений.

4.3. Спроектировать архитектуру мобильного приложения.

4.4. Спроектировать UI/UX-дизайн для мобильного приложения.

4.5. Разработать клиентскую часть мобильного приложения.

4.6. Протестировать мобильное приложение.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
ст. преподаватель кафедры СП Н.С. Силкина

Задание принял к исполнению Н.В. Стецкий

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В наше время быстрого развития технологий и постоянных изменений в предпочтениях потребителей, общественное питание становится неотъемлемой частью нашей повседневной жизни. Кафе и рестораны стремятся адаптироваться к новым требованиям рынка, внедряя инновационные решения для улучшения качества обслуживания.

В эпоху цифровых технологий, оказывается, что ресторанный бизнес предполагает не только качественные блюда и обслуживание, но и должен приспосабливаться к потребностям современного потребителя. Онлайн-заказы и доставка еды становятся неотъемлемой частью этой индустрии, а предприятия, которые игнорируют этот тренд, рискуют отстать от конкурентов [1].

Российский фудтех [2] предлагает разнообразные решения для удовлетворения требований современного ресторанного бизнеса. Маркетплейсы еды [3] позволяют ресторанам оперативно взаимодействовать со своими клиентами, однако это может сопровождаться потерей части прибыли и зависимостью от работы платформы. Облачные продукты, такие как мобильные приложения для ресторанов, предлагают доступный функционал для организации доставки и автоматизации процессов, но при этом могут ограничивать предприятия в свободе действий.

Агрегаторы доставок [4] предлагают решение в виде готового и отлаженного процесса доставки, что освобождает рестораны от необходимости организовывать доставку самостоятельно. Однако здесь также есть недостатки, такие как зависимость от комиссий и разработчиков.

Разработка собственного мобильного приложения для ресторана представляет собой значимое решение в современном ресторанном бизнесе. Такой подход дает возможность управлять всем процессом обслуживания и взаимодействия с клиентами без посредников. Собственное приложение

позволяет ресторану сохранить контроль над прибылью и не зависеть от работы сторонних платформ. Кроме того, разработка собственного приложения открывает возможность для индивидуализации сервиса, улучшения качества обслуживания и формирования собственной базы клиентов. Важно отметить, что хотя разработка собственного приложения может потребовать больших инвестиций и временных затрат, в долгосрочной перспективе это может оказаться наиболее выгодным и эффективным решением для ресторана.

Из вышеупомянутых решений для ресторанного бизнеса, кажется, наиболее перспективным и выгодным вариантом является разработка собственного кроссплатформенного мобильного приложения [5]. Хотя маркетплейсы еды и агрегаторы доставок предоставляют определенные преимущества в виде готовых решений и отлаженных процессов, они могут сопровождаться высокими комиссиями и зависимостью от сторонних платформ и разработчиков.

В свою очередь, собственное мобильное приложение позволит ресторану контролировать процесс обслуживания и доставки, а также взаимодействовать непосредственно с клиентами. Это дает большую гибкость и возможность предоставления индивидуализированных услуг, что может способствовать улучшению качества обслуживания и удовлетворению потребностей клиентов.

Таким образом, инвестирование в разработку собственного мобильного приложения может быть ключевым шагом для ресторанов в адаптации к современным тенденциям рынка и обеспечении конкурентоспособности в индустрии общественного питания.

В этом контексте, разработка кроссплатформенного мобильного приложения для столовой «Восточный Дракон» становится актуальной и важной задачей.

Столовая «Восточный Дракон» представляет собой уникальное заведение, которое успешно привлекает внимание жителей центра и гостей города. Специализируясь на блюдах китайской кухни, столовая также предлагает адаптированные под российского потребителя вариации, что позволяет удовлетворить разнообразные вкусовые предпочтения посетителей.

Одним из особых преимуществ столовой «Восточный Дракон» является система обедов, представленных в формате конструктора. Посетитель может самостоятельно собрать обед, выбрав базовый компонент, такой как рис, и дополнив его различными горячими блюдами и салатами. Этот подход позволяет каждому гостю создать именно тот обед, который соответствует его предпочтениям и потребностям.

Кроме того, столовая «Восточный Дракон» является популярным местом среди студентов-китайцев, что свидетельствует о высоком качестве кухни и атмосферы заведения.

На популярных платформах, таких как 2gis и Яндекс Карты, отмечается высокая оценка столовой за ее чистоту, вкус, цены и скорость обслуживания. Однако, проведенный анализ заведения и опрос клиентов выявил несколько аспектов, требующих внимания и усовершенствования.

Среди выявленных потенциальных возможностей для улучшения бизнес-процессов в столовой «Восточный Дракон» выделяются следующие: необходимость сокращения времени ожидания заказов, увеличение удобства заказа для групповых посещений и повышение уровня информированности клиентов о предлагаемых блюдах.

Одной из главных проблем, выявленных в ходе анализа, является ограничение возможностей заказа: в ресторане можно заказывать блюда только на месте. Это приводит к длинным очередям в часы-пик из-за подготовки и упаковки заказов, что в свою очередь ухудшает общее впечатление клиентов от посещения заведения. Также часто возникает ситуация, когда клиенты не знают, какое блюдо перед ними, что приводит к задержкам и дополнительным вопросам на раздаче.

Для решения этих проблем и повышения удовлетворенности клиентов представляется актуальной разработка мобильного приложения для столовой «Восточный Дракон». Приложение позволит клиентам удобно ознакомиться с меню и описанием блюд, делать предварительные заказы и оплачивать их онлайн. Это сократит время ожидания заказов, увеличит удобство заказа для групповых посещений и повысит уровень информированности клиентов о предлагаемых блюдах, что в итоге приведет к улучшению общего опыта посещения столовой «Восточный Дракон».

Постановка задачи

Целью данной выпускной квалификационной работы является разработка кроссплатформенного мобильного приложения, которое позволит столовой «Восточный Дракон» оптимизировать процессы обслуживания клиентов и удовлетворить их потребности в максимальной степени. Для достижения этой цели поставлены следующие задачи:

1) изучение предметной области;
2) проведение обзора аналогичных приложений;
3) проектирование архитектуры мобильного приложения;
4) проектирование интерфейса мобильного приложения;
5) разработка клиентской части мобильного приложения;
6) тестирование и анализ эффективности приложения.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 51 страницу, объем списка литературы – 20 источников.

В первой главе описывается предметная область разработки мобильных кроссплатформенных приложений. Также в этой главе был проведен анализ приложений доставки еды и выделен список функций, которыми должно обладать приложение такого типа.

Вторая глава посвящена проектированию мобильного приложения. В ней были тщательно определены варианты использования, функциональные

и нефункциональные требования. Процесс определения требований включал анализ потребностей пользователей, изучение конкурирующих приложений и проведение опросов среди потенциальных пользователей. В результате этого анализа была создана исчерпывающая спецификация, включающая в себя список всех необходимых функций и возможностей. Спроектированы архитектура и макеты приложения. Архитектура приложения была разработана с учетом требований к масштабируемости, надежности и производительности, что обеспечивает возможность дальнейшего расширения функциональности без значительных изменений в базовом коде. Макеты, созданные на этом этапе, дают четкое представление о том, как будет выглядеть и функционировать пользовательский интерфейс, а также позволяют выявить возможные проблемы на ранней стадии разработки.

В ходе работы над третьей главой описана реализация мобильного приложения: разработка бизнес-логики, UI-интерфейса приложения и внедрение push-уведомлений.

В четвертой главе содержится описание функционального тестирования приложения.

В приложении А содержатся спроектированные окна мобильного приложения.

В приложении Б содержатся листинги основных моделей мобильного приложения.

В приложении В содержатся результаты Golden-тестов основных экранов мобильного приложения.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Технологии реализации мобильных приложений

При выборе подхода к разработке мобильного приложения важно учитывать множество факторов, включая цели проекта, бюджет, требования пользователей и возможности технической реализации. Существует два основных подхода: нативная разработка и кроссплатформенная разработка [6]. В данном анализе будут рассмотрены преимущества кроссплатформенного подхода в определенных ситуациях.

Одним из основных преимуществ кроссплатформенной разработки является низкая стоимость разработки и обслуживания. За счет использования единой кодовой базы и уменьшения необходимости в найме разработчиков с разными навыками, затраты на разработку и поддержку приложения снижаются значительно. Это особенно важно для небольших компаний или стартапов с ограниченным бюджетом, которые стремятся минимизировать затраты на разработку.

Кроме того, кроссплатформенная разработка позволяет ускорить процесс выхода приложения на рынок. Создание единого приложения для нескольких платформ позволяет сократить время разработки и тестирования, что особенно ценно для бизнесов, стремящихся быстро привлечь пользователей и провести тестирование гипотез.

Другим важным преимуществом кроссплатформенной разработки является возможность приложения сразу появиться на нескольких платформах. Это позволяет охватить более широкую аудиторию пользователей и увеличить потенциальную прибыль от приложения. Пользователи с устройствами на разных операционных системах смогут опробовать ваше приложение, что способствует его популяризации и распространению.

Кроме того, при кроссплатформенной разработке возможно повторное использование кодовой базы. Это позволяет сэкономить время и ре-

сурсы на разработке других приложений в будущем, а также снизить затраты на исправление ошибок и внедрение нового функционала благодаря использованию проверенных и оптимизированных решений.

На рынке существует множество платформ для кроссплатформенной разработки, из которых наиболее популярными являются Flutter, React Native и Cordova. [7] Недавно востребованные Xamarin и Ionic теперь редко используются, уступая место более современным и гибким решениям. На рисунке 1 представлена аналитика популярности поисковых запросов по данным технологиям за последние пять лет.

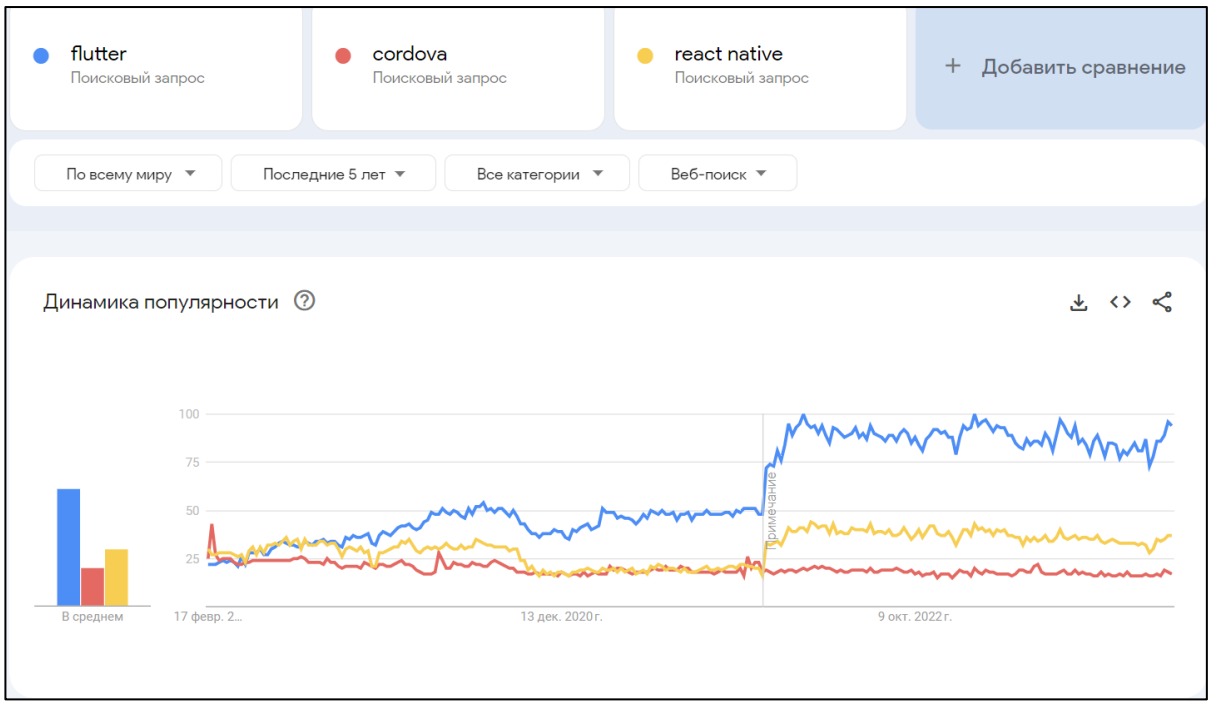

Рисунок 1 – Статистика поисковых запросов
для кроссплатформенных решений

React Native [8] – это популярная платформа мобильных приложений на основе JavaScript, которая позволяет создавать мобильные приложения с собственным интерфейсом для iOS и Android. Фреймворк позволяет создавать приложения для различных платформ, используя одну и ту же кодовую базу.

Flutter [9] – бесплатный и открытый набор средств разработки мобильного пользовательского интерфейса, созданный компанией Google и выпущенный в мае 2017 года. Проще говоря, с помощью Flutter возможно создать собственное мобильное приложение с одним массивом кода. Это означает, что для создания двух приложений (IOS и Android) можно использовать единый язык программирования и одну базу кода.

Cordova [10] – это фреймворк с открытым исходным кодом, работающий на стандартных веб-технологиях: JavaScript, HTML5 и CSS3. Приложение, написанное на Cordova – это совокупность HTML-страниц в нативной «оболочке» платформы. Благодаря активному сообществу разработчиков и солидному возрасту, у этого фреймворка есть множество бесплатных плагинов.

Исходя из анализа кроссплатформенных технологий для создания мобильных приложений, для разработки приложения для заведения «Восточная столовая №1» был выбран Flutter, т.к. он является самым новым и перспективным фреймворком на текущий момент.

1.2. Обзор аналогичных мобильных приложений

На рынке представлено много мобильных приложений, созданных для доставки еды. Это приложения-агрегаторы и собственные разработки ресторанов. Рассмотрим некоторые из них.

Яндекс Еда

Приложение разработано компанией Яндекс и является сервисом доставки еды из различных заведений. Можно выделить несколько основных функциональных блоков:

1) главная;
2) каталог;
3) детальная страница товара или комбо-набора;
4) корзина;
5) оформление заказа;

6) профиль пользователя;
7) форма обратной связи;
8) оплата заказа онлайн;
9) авторизация/регистрация;
10) push-уведомления.

Скриншоты приложения изображены на рисунке 2.

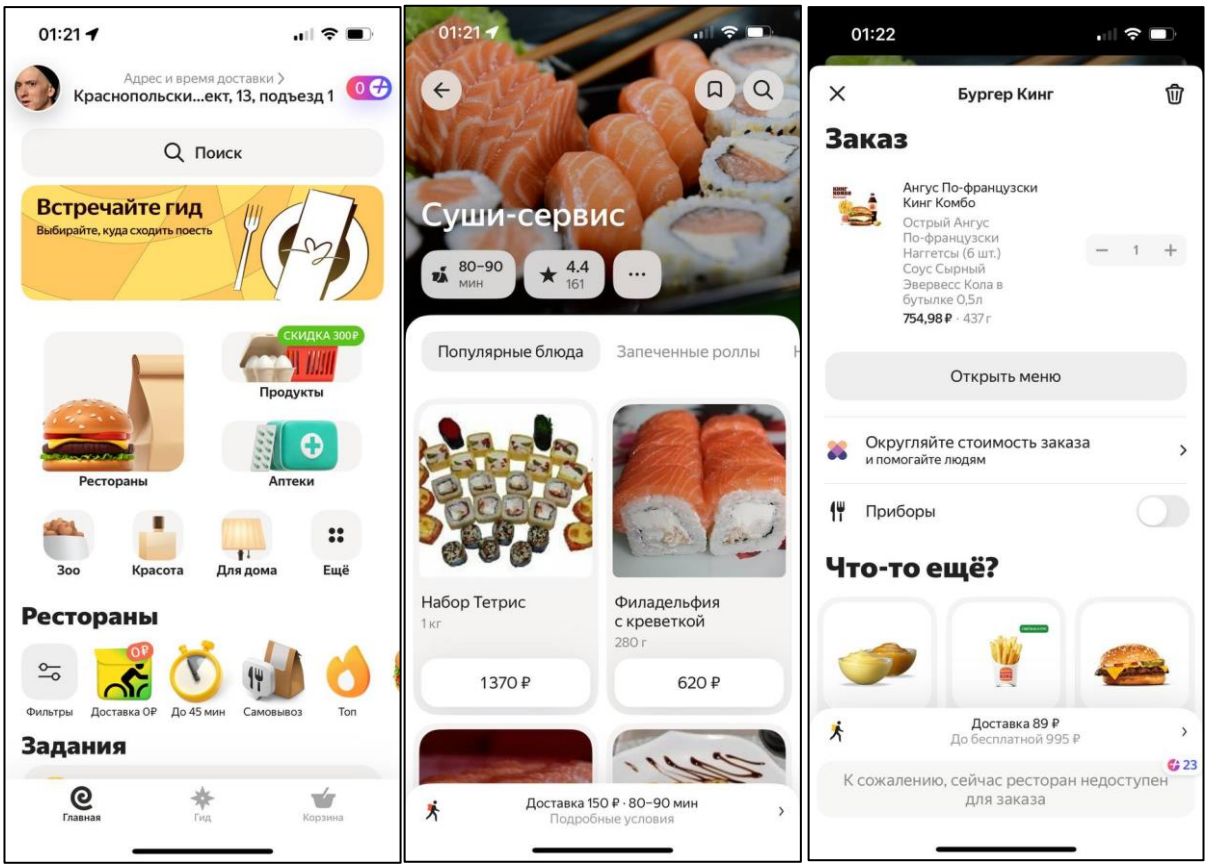

Рисунок 2 – Скриншоты приложения «Яндекс Еда»

Приложение хорошо себя показало с точки зрения наполнения дизайна и проработки интерфейса. Не возникает ситуации, когда непонятно, что приложение хочет от пользователя. Есть хорошая программа лояльности. Но слишком агрессивная реклама и интеграция других сервисов компании Яндекс отталкивает.

Жизньмарт

Это собственное приложение сети магазинов готовой еды «Жизньмарт». В нем можно выделить несколько основных функциональных блоков:

1) каталог;
2) детальная страница товара;
3) корзина;
4) оформление заказа;
5) профиль пользователя;
6) форма обратной связи;
7) оплата заказа онлайн;
8) авторизация/регистрация;
9) push-уведомления.

Скриншоты приложения изображены на рисунке 3.

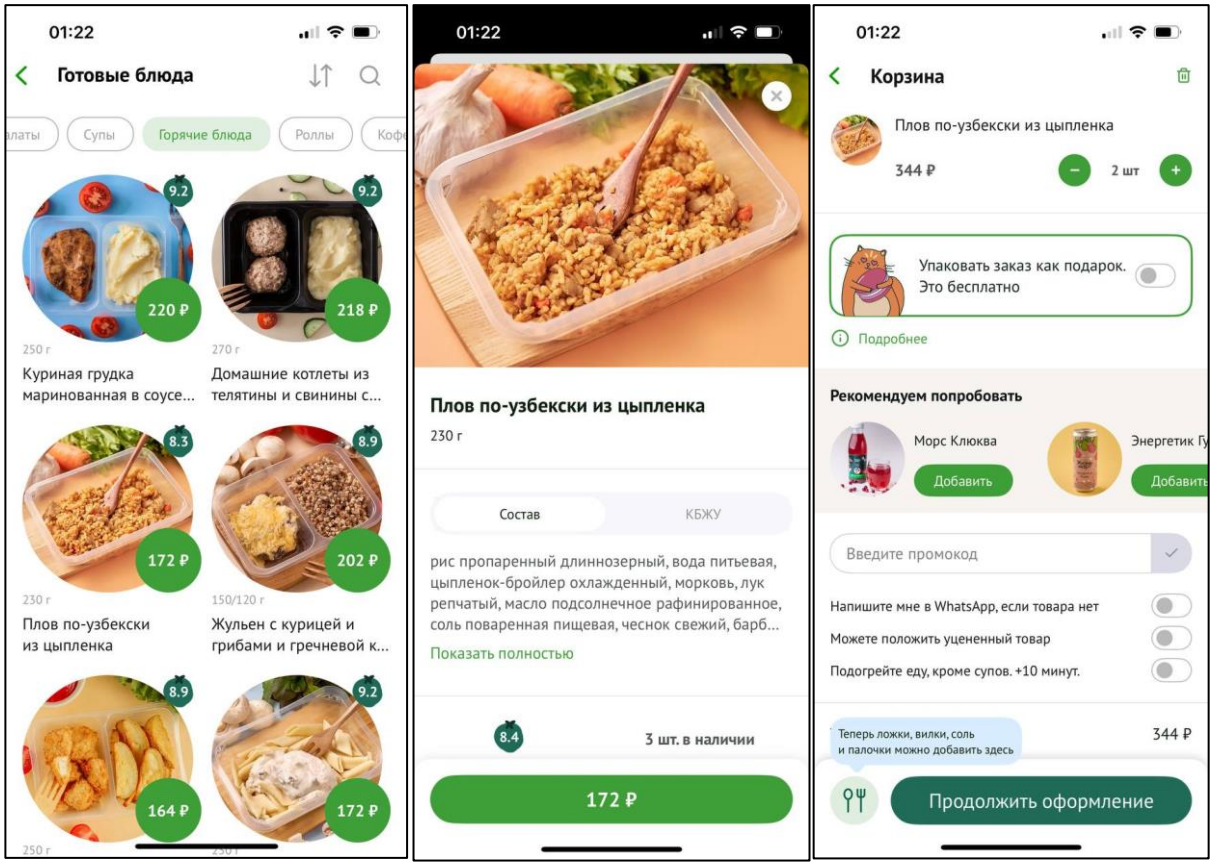

Рисунок 3 – Скриншоты приложения «Жизньмарт»

Приложение не требовательное, хорошо работает на старых смартфонах. Есть программа лояльности. Интерфейс не всегда понятный, и пользователь может уйти, что уменьшает конверсию.

Додо Пицца

Это собственное приложение сети пиццерий «Додо Пицца». В нем можно выделить несколько основных функциональных блоков:

1) каталог;
2) детальная страница товара или комбо-набора;
3) корзина;
4) оформление заказа;
5) форма обратной связи;
6) оплата заказа онлайн;
7) авторизация/регистрация;
8) push-уведомления.

Скриншоты из приложения изображены на рисунке 4.

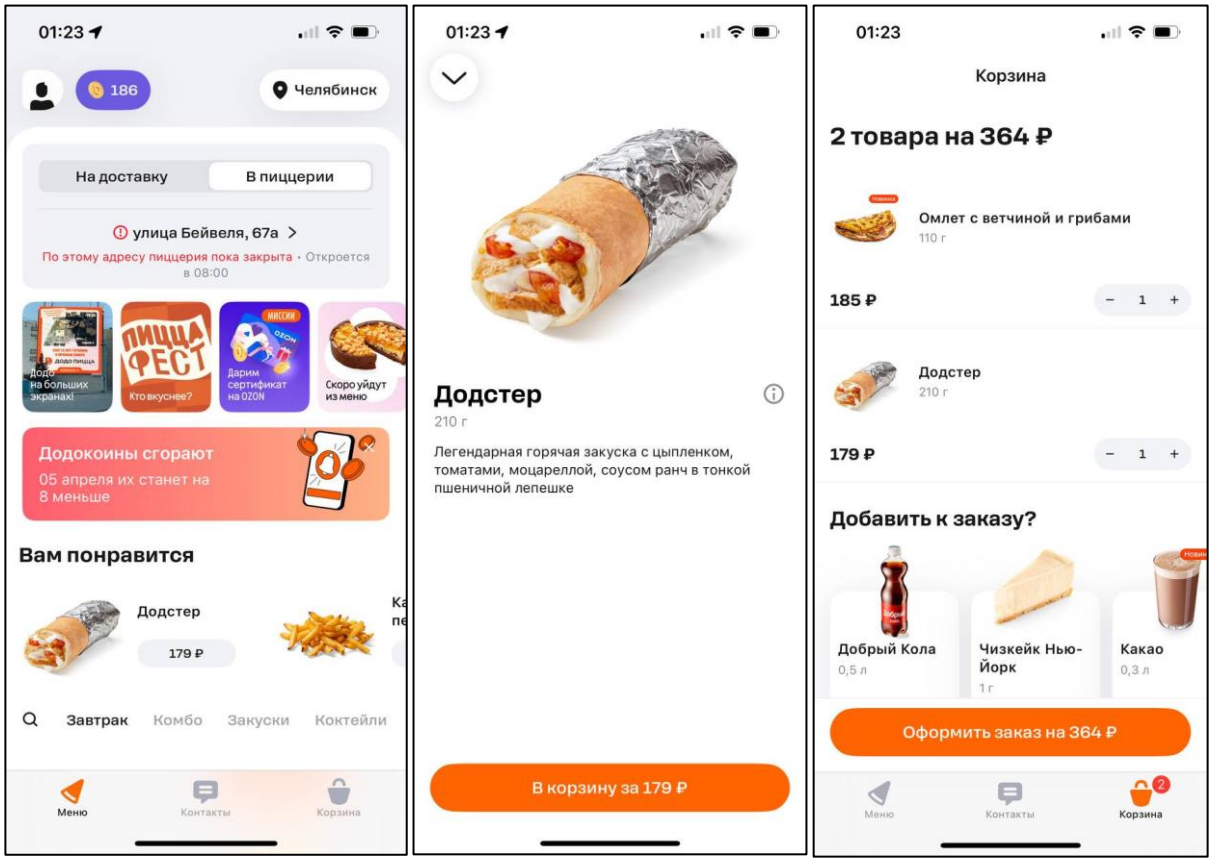

Рисунок 4 – Скриншоты приложения «Додо Пицца»

В приложении задействуются функции вибрации смартфона, это ощущается интересно и подчеркивает различные действия пользователя. С точки зрения дизайна приложение так же хорошо сделано. Недостатков замечено не было.

Таким образом, рассмотренные приложения функционально мало отличаются друг от друга, поэтому можно выделить список основных функциональных блоков, которые ожидает пользователь от подобного типа приложений:

1) каталог;
2) детальная страница товара или комбо-набора;
3) корзина;
4) оформление заказа;
5) профиль пользователя;
6) форма обратной связи;
7) оплата заказа онлайн;
8) авторизация/регистрация;
9) push-уведомления.

Вывод по первой главе

В результате анализа предметной области для разработки мобильного приложения столовой «Восточный дракон» был выбран фреймворк Flutter, а также произведен анализ аналогичных приложений, который позволил выделить основные функции, которые должно предоставлять пользователю приложение такого типа.

2. ПРОЕКТИРОВАНИЕ

2.1. Требования и варианты использования

Выявление функциональных и нефункциональных требований

В ходе анализа предметной области были выявлены требования к разрабатываемой системе. Были выявлены следующие функциональные требования:

1) система должна предоставлять возможность регистрации;
2) система должна предоставлять доступ к каталогу товаров;
3) система должна давать возможность оформить заказ онлайн;
4) система должна давать возможность просматривать профиль пользователя;
5) система должна предоставлять доступ к корзине пользователя;
6) система должна предоставлять функцию рассылки push-уведомлений.

Были выявлены следующие нефункциональные требования:

1) система должна быть написана на языке программирования Dart;
2) система должна быть разработана для операционных систем Android и iOS.

Диаграмма вариантов использования

В соответствии с требованиями была построена диаграмма вариантов использования [11], которая представлена на рисунке 5. Диаграмма отражает модель взаимодействия актеров «Пользователь» и «Гость».

Пользователь может зарегистрироваться в приложении, используя адрес электронной почты. Если пользователь ранее не авторизовывался в приложении, то система потребует от него ввести свои личные данные.

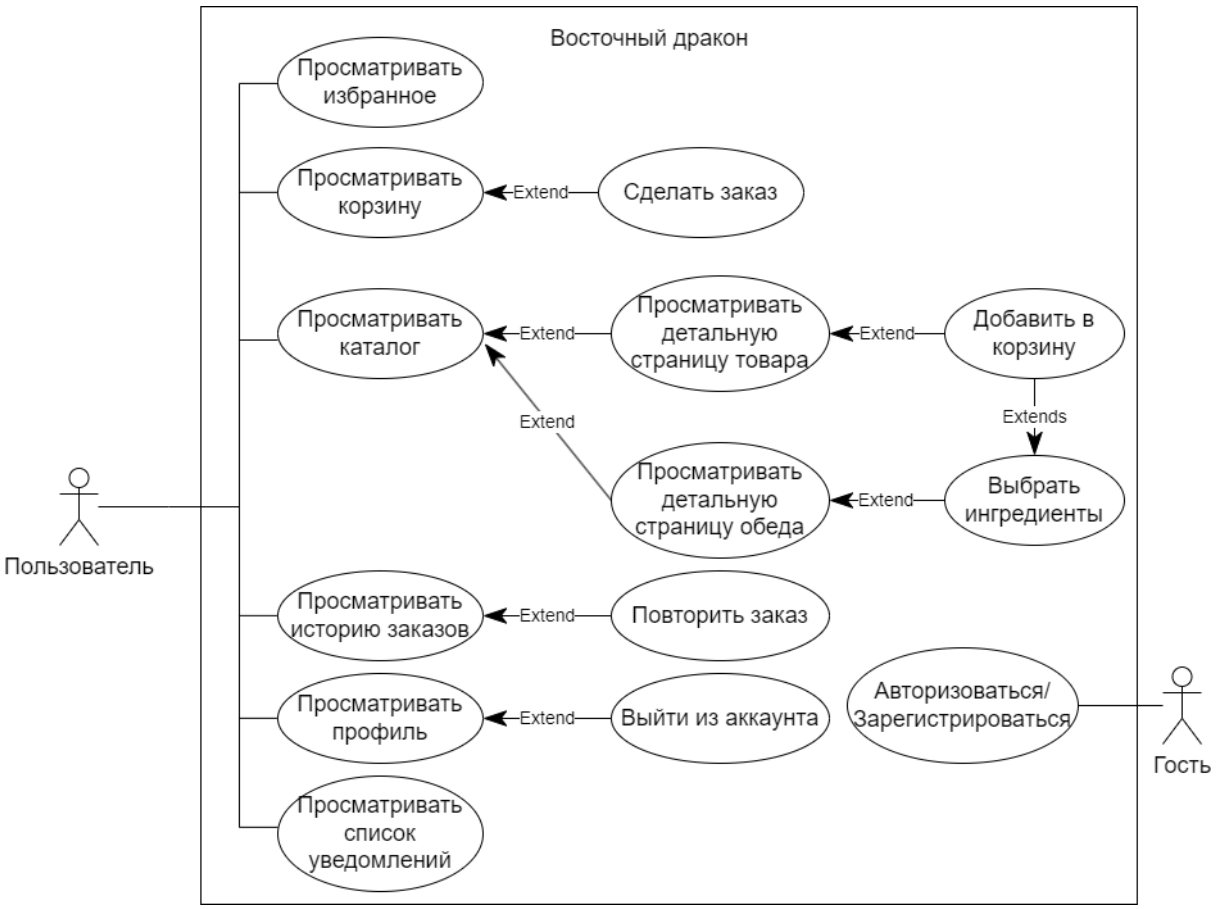

Рисунок 5 – Диаграмма вариантов использования

После регистрации пользователь может просматривать каталог товаров и обедов. При нажатии на элемент каталога пользователь попадает на детальную страницу товара или обеда, на которой находится описание. На странице товара можно добавить нужное количество единиц товара в корзину. На странице обеда пользователю необходимо собрать свой обед из ингредиентов, выбрав которые пользователь может добавить составленный обед в корзину.

Пользователь может просматривать страницу профиля, на которой указана личная информация, указанная при регистрации. Пользователь может сменить адрес электронной почты и другие данные. Также на странице профиля есть кнопка выхода из аккаунта.

Пользователю доступна страница избранных товаров и страница уведомлений, где собран список push-уведомлений, отправленных на данный аккаунт.

Если пользователь добавил хотя бы один товар в корзину, то в нижней части экрана каталога появляется кнопка перехода на страницу корзины. На этой странице содержится список товаров, находящихся в корзине и доступных для заказа. Пользователь может создать заказ из находящихся в корзине товаров.

2.2. Проектирование архитектуры приложения

В качестве архитектуры приложения была выбрана MVVM [12]. Этот архитектурный паттерн разделяет приложение на три основных слоя.

1. Model (Модель) отвечает за запрос и хранение данных. Модель может включать в себя операции с сервисами и репозиториями.
2. View (Представление) отвечает за отображение данных и взаимодействие с пользователем. Это компонент, с которым пользователь взаимодействует, и он визуализирует данные, предоставляемые ViewModel.
3. ViewModel (Модель представления) служит посредником между Model и View. ViewModel преобразует данные из Model в формат, который может быть легко отображен в View, и обрабатывает пользовательские действия, перенаправляя их в Model. ViewModel также позволяет реализовать связь данных между Model и View.

Для реализации паттерна MVVM была выбрана библиотека Elementary [13]. В качестве слоя представления здесь выступает ElementaryWidget, в качестве модели – ElementaryModel, а в качестве модели представления – WidgetModel. На рисунке 6 представлена общая схема устройства Elementary.

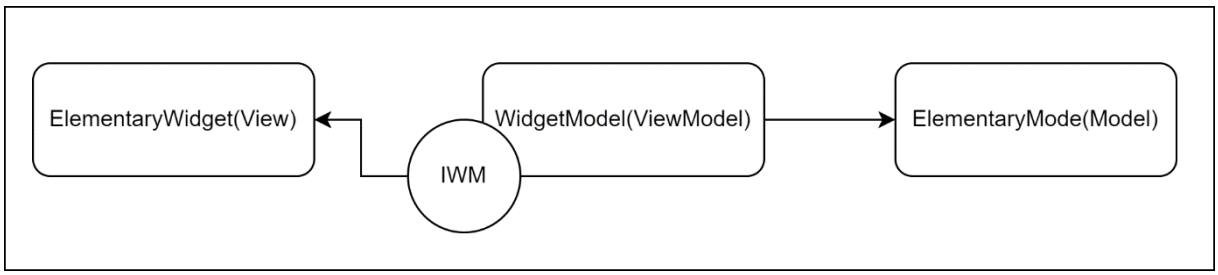

Рисунок 6 – Общая схема Elementary.

Схема работы Elementary в рамках мобильного приложения «Восточный дракон» приведена на примере экрана каталога на рисунке 7.

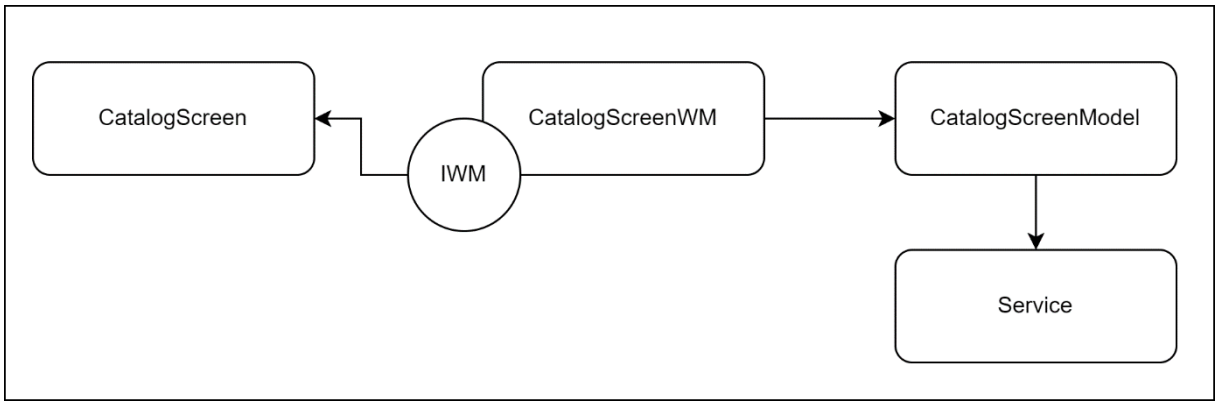

Рисунок 7 – Схема Elementary на примере экрана каталога.

Здесь CatalogScreen – виджет, который взаимодействует с виджет моделью с помощью интерфейса IWM(IWidgetModel) и слушает ее изменения.

CatalogScreenWM – это класс, который хранит в себе состояния для отображения данных на экране.

CatalogScreenModel – это класс, который содержит методы для отправки данных на сервер и их получения. Запросы производятся с помощью глобального сервиса, который существует в единичном экземпляре на все приложение.

Модель обращается к сервису, сервис возвращает данные в сыром виде, модель преобразует данные в формат, нужный для модели представления.

Остальные экраны и виджеты в приложении устроены аналогичным образом.

2.3. Проектирование интерфейса приложения

В соответствии с выбранными функциональными требованиями и вариантами использования были разработаны макеты страниц мобильного приложения. Разработка дизайна приложения проходила в онлайн графическом редакторе Figma [14].

Были спроектированы следующие страницы приложения:

1) ввод адреса электронной почты;
2) ввод кода подтверждения;
3) ввод имени пользователя;
4) каталог товаров;
5) детальная страница обеда;
6) детальная страница товара;
7) корзина;
8) список заказов;
9) оформление заказа.

Процесс создания макетов включал не только разработку визуальной части интерфейса, но и проработку пользовательских сценариев для каждого экрана. Это позволило обеспечить логическую последовательность действий пользователя и удобство навигации по приложению. Особое внимание было уделено адаптивности дизайна, чтобы страницы корректно отображались на различных устройствах с разными разрешениями экранов. Таким образом, макеты были протестированы на различных моделях смартфонов, чтобы убедиться в их функциональности и удобстве использования.

Макет экрана каталога представлен на рисунке 8. Макеты остальных экранов представлены на рисунках 1–6 приложения А.

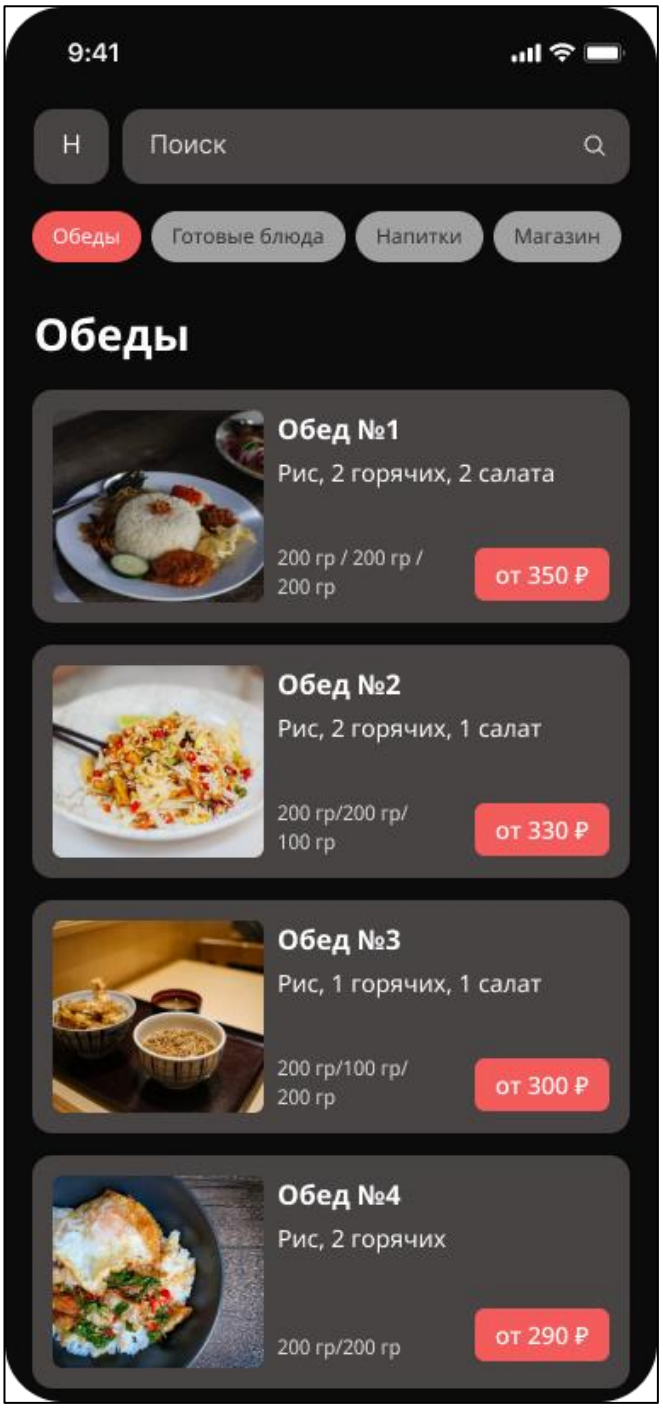

Рисунок 8 – Макет экрана каталога

Вывод по второй главе

В ходе проектирования были определены функциональные и нефункциональные требования к системе. На основе этих требований была построена диаграмма вариантов использования, определены основные актеры, взаимодействующие с системой. Также были спроектированы макеты страниц приложения.

3. РЕАЛИЗАЦИЯ

Для реализации мобильного приложения «Восточный дракон» были использованы следующие инструменты:

1) среда разработки Visual Studio Code;
2) фреймворк Flutter;
3) язык программирования Dart;
4) библиотеки:

– dio для работы с сетью;
– elementary для управления состоянием;
– json_serializable для кодогенерации;
– go_router для навигации.

3.1. Реализация компонента Model

Сервер использует REST архитектуру для выполнения запросов [15]. Ответы от сервера приходят в формате, представленном в листинге 1.

Листинг 1 – Формат ответа от сервера

```
{
    "data": data
    "message": null,
    "success": true,
    "code": 200
}
```

`Data` – это данные, которые ожидает клиент, `message` – это сообщение от сервера в случае ошибки, `success` – состояние успеха отработки запроса, `code` – код ответа от сервера.

В случае, если `success` равен `true`, клиент делает попытку десериализации поля `data`.

В соответствии с API были реализованы модели – описания файлов json формата, которые приходят в ответ на запрос. Пример реализации модели элемента каталога представлен в листинге 2.

Листинг 2 – Модель элемента каталога

```
import 'package:eastern_dragon/core/common/data/exceptions/re-
sponse_parse_exception.dart';
import 'package:json_annotation/json_annotation.dart';
part 'catalog_item_model.g.dart';

@JsonSerializable(createToJson: false)
class CatalogItemModel {
  final int id;
  final String name;
  final String compound;
  final List<num> weights;
  final String? image;
  final num price;
  @JsonKey(defaultValue: false)
  final bool isLunch;
  num get totalWeight => weights.reduce((a, b) => a + b);

  CatalogItemModel({
    required this.id,
    required this.name,
    required this.compound,
    required this.weights,
    required this.price,
    required this.isLunch,
    this.image,
  });
}
```

Модель элемента каталога содержит следующие поля.

– Уникальный идентификатор (строка).

– Название (строка).

– Состав (строка).

– Вес (массив целых чисел).

– Цена (число).

– Флаг обеда (логический тип).

– Ссылка на картинку (строка).

Всего было реализовано 7 моделей. Примеры реализации основных моделей приложения представлены в приложении Б.

3.2. Реализация компонента WidgetModel

После создания моделей и методов, которые обращаются к сервису, необходимо связать `ElementaryModel` с `WidgetModel`, которая слушает дей-

ствия пользователя. Для того, чтобы из модели представления вызвать метод модели, нужно обратиться к ней через `model.methodName`(). В листинге 3 приведен код метода, который вызывается, когда пользователь нажимает на кнопку отправки кода на странице авторизации.

Листинг 3 – Метод нажатия на кнопку отправки кода

```
Future<void> trySendCode() async {
    await executor.execute(
      () => model.auth(codeController.text),
      before: () => _isButtonAvailableState.accept(false),
      after: () => _isButtonAvailableState.accept(true),
      onSuccess: (user) {
        userAuthEntity.setUserState(user!);
      },
      onError: (e) => ToastShower.showError(context, e.title),
    );
  }
```

Перед отправкой кода состояние кнопки `_isButtonAvailableState` переводится в неактивное, чтобы пользователь не мог несколько раз нажать на кнопку. В случае успешной отработки запроса и корректности отправленных данных сервер вернет модель пользователя. В противном случае будет показано сообщение с текстом ошибки.

В листинге 4 представлен метод отправки кода подтверждения на сервер, который будет вызван методом нажатия на кнопку. В случае успешного прохождения авторизации, в ответ сервер вернет модель пользователя.

Листинг 4 – Метод отправки кода подтверждения на сервер

```
Future<UserModel> auth(String code) async {
  final response = await requestService.post(
    RestConstants.sendEmailCode,
    data: FormData.fromMap(
      <String, dynamic>{
        'code': code,
      },
    ),
  );

final baseRes = BaseResponseModel.fromJson(response.data as Map<String, dy-
namic>);

final user = UserModel.fromJson(baseRes.data as Map<String, dynamic>);
return user;
}
```

Метод `auth()` принимает на вход введенный пользователем код подтверждения и возвращает `UserModel` в случае успеха, иначе – возвращает ошибку. Запрос отправляется с помощью класса `RequestService`, который содержит методы для отправки запросов к REST API(Get, Post, Put, Delete). В качестве тела запроса передается объект типа `FormData`, который содержит ключ `code`, а в качестве значения подставляется переданный код. Полученный от сервера ответ преобразовывается в базовую модель ответа на запрос. Из этой модели берется поле `data`, которое преобразовывается в модель пользователя.

Всего было реализовано 17 методов. Методы работы с пользователем включают в себя:

1) запрос кода на адрес электронной почты;
2) отправку кода на сервер;
3) получение данных пользователя;
4) изменение данных пользователя;
5) выход из аккаунта.

Методы взаимодействия с API для работы с каталогом включают в себя следующие:

1) запрос данных каталога;
2) поиск в каталоге по названию;
3) запрос данных товара;
4) запрос данных обеда.

Методы взаимодействия с API для работы с корзиной включают в себя следующие:

1) добавление товара/обеда в корзину;
2) изменение количества позиции в корзине;
3) запрос данных корзины.

Методы взаимодействия с API для работы с заказами включают в себя следующие:

1) оформление заказа;

2) запрос списка заказов пользователя;

3) повторение заказа.

Также были реализованы следующие вспомогательные методы:

1) запрос списка уведомлений;

2) запрос списка избранных товаров.

3.3. Реализация компонента Widget (View)

Верстка страниц приложений во Flutter представляет собой дерево виджетов, а взаимодействие с приложением – это обход дерева. На рисунке 9 показано, как выглядит дерево виджетов приложения при входе на экран авторизации. Можно заметить, что почти в самом корне дерева находится `ResponsiveScaledBox`, который отвечает за респонсивную верстку.

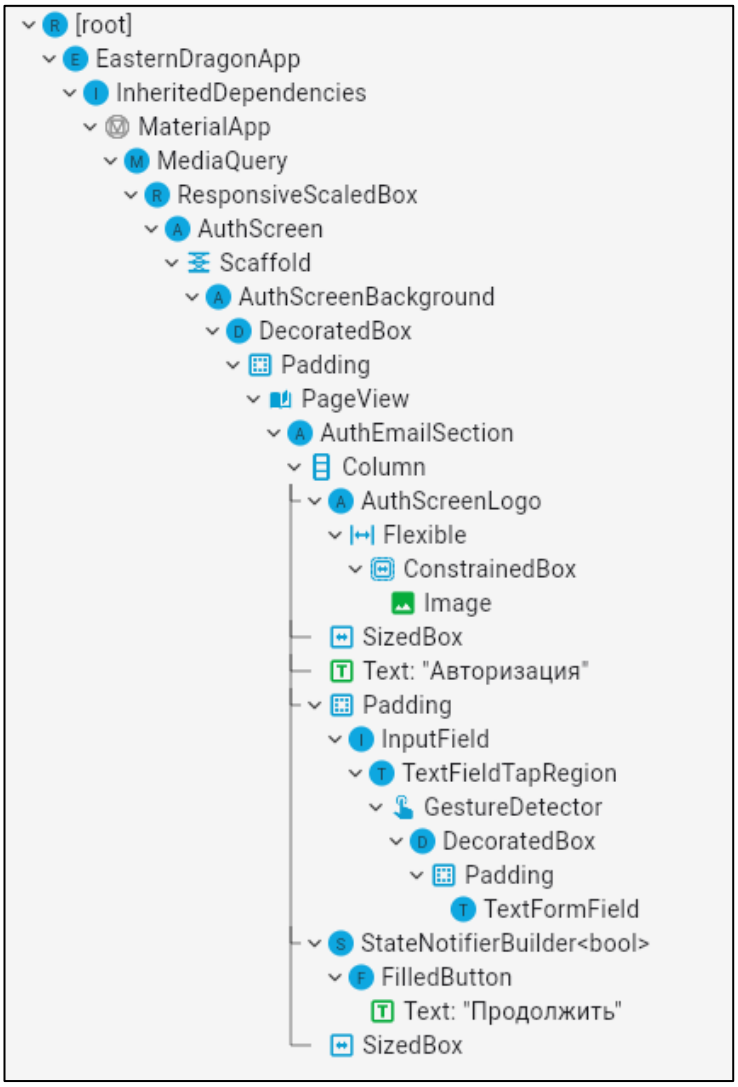

Рисунок 9 – Дерево виджетов приложения

Согласно требованиям, также были разработаны следующие страницы мобильного приложения:

1) `auth_screen.dart` – экран авторизации;
2) `user_data_screen.dart` – экран ввода личных данных;
3) `catalog_screen.dart` – экран каталога;
4) `cart_screen.dart` – экран корзины;
5) `order_screen.dart` – экран оформления заказа.

Код верстки страницы авторизации приведен в листинге 5.

Листинг 5 – Верстка экрана авторизации

```
import 'package:elementary/elementary.dart';
import 'package:flutter/material.dart';
class AuthScreen extends ElementaryWidget<IAuthScreenWM> {
  const AuthScreen({super.key})
      : super(
          defaultAuthScreenWMFactory,
        );
  @override
  Widget build(IAuthScreenWM wm) {
    return Scaffold(
      body: AuthScreenBackground(
        child: PageView(
          controller: wm.pageController,
          clipBehavior: Clip.none,
          physics: const NeverScrollableScrollPhysics(),
          children: [
            AuthEmailSection(wm: wm),
            AuthCodeSection(wm: wm),
          ],
        ),
      ),
    );
  }
}
```

Экран авторизации отрисовывается на устройстве с помощью метода `build()`, который является частью фреймворка. Метод возвращает дерево виджетов, корневым элементом которого является `Scaffold` (так же предоставляется фреймворком). Внутри него содержится `AuthScreenBackground` – виджет с градиентом и логотипом. Далее следует `PageView` – виджет, позволяющий перелистывать дочерние элементы, в данном случае он нужен, чтобы переключать секции авторизации (ввод адреса электронной

почты и ввод кода). `PageView` управляется с помощью `PageViewController`, экземпляр которого находится в виджет модели.

Для вышеперечисленных экранов также была реализована верстка.

Респонсивная верстка

В приложении используется респонсивная верстка, т.е. все элементы с фиксированной шириной на устройствах с разным расширением экрана будут иметь одинаковую ширину и высоту, пропорциональную этой ширине. Для этого используется пакет `responsive_framework`. В этом приложении был использован класс `ResponsiveScaledBox`, параметр `width` у которого равен 375 в соответствие с макетами Figma. Также во всем приложении размер текста принудительно сделан равным 1, чтобы у пользователей, у которых стоит увеличение текста в системе, не ломалась верстка. Реализация респонсивной верстки приведена в листинге 6.

Листинг 6 – Респонсивная верстка

```
return MediaQuery(
       data: MediaQuery.of(context).copyWith(
       textScaler: const TextScaler.linear(1.0),
       ),
       child: ResponsiveScaledBox(
       width: 375,
       child: child!,
   ),
);
```

3.4. Push – уведомления

Push уведомления – это короткие сообщения, всплывающие на экране компьютера или мобильного телефона с различной информацией [16]. Push уведомление является коротким текстом с важной информацией для пользователя (оповещение о сообщении, статусе заказа и акциях).

В ходе анализа аналогичных мобильных приложений были выявлены ключевые особенности приложений данного сегмента, и push – уведомления присутствуют в каждом продукте. Они выполняют не только оповестительную функцию, но и маркетинговую.

На данном этапе разработки планируется использование уведомлений для своевременного информирования пользователя об изменении статуса заказа. Для отправки уведомлений используется решение от Firebase [17], а именно пакет `firebase_messaging`, он используется для приема и отображения уведомлений в фоне. Для отображения уведомлений, когда приложение находится в работе, используется пакет `flutter_local_notifications`.

Принцип работы подписки на уведомления: при входе в приложение запрашивается разрешение пользователя на отправку уведомлений, если разрешение еще не было выдано. Затем происходит проверка на авторизацию пользователя в системе. Если пользователь не авторизован, то ничего не происходит. Если авторизован, то запрашивается уникальный токен устройства Firebase и отправляется на сервер. Также токен запрашивается при авторизации пользователя. Если токен успешно отправился, то сервер будет присылать уведомления, когда это необходимо. В листинге 7 представлен класс, который выполняет вышеописанную логику.

Листинг 7 – Подписка на push-уведомления

```
abstract class FirebaseNotificationHandler {
  /// Инициализация методов для работы с FirebaseMessaging.
  static Future<void> init({
    required RequestService requestService,
  }) async {
    final messaging = FirebaseMessaging.instance;
    await messaging.requestPermission();
    await messaging.setForegroundNotificationPresentationOptions(
      alert: true,
      badge: true,
      sound: true,
    );
    final token = await messaging.getToken();
    TokenRefresher.periodicallyRefresh(
      token, requestService: requestService,
    );
    // Слушаем сообщения
    FirebaseMessaging.onMessage.listen(
      (message) {
        if (Platform.isAndroid) {
          LocalNotificationHandler.showFlutterLocalNotification(message);
        }
      },
    );
    // Слушаем токен, если он вдруг обновится
    messaging.onTokenRefresh.listen(
      (data) => TokenRefresher.periodicallyRefresh(
```

```
          data,
          requestService: requestService,
        ),
      );
    }
}
```

3.5. Оплата

Одной из проблем, выявленных в ходе анализа предприятия и общения с заказчиком, является невозможность оформить заказ онлайн. Для решения данной задачи был выбран сервис Yookassa, который берет на себя весь цикл обработки платежа. Для интеграции с Yookassa используется пакет `yookassa_payments_flutter`. Он предоставляет несколько вариантов оплаты: ЮMoney, SberPay, ЯндексPay, Банковские карты и Система Быстрых Платежей (СБП). Также в будущем сервис можно будет интегрировать с платежными терминалами, которые находятся непосредственно на кассе и на кухне в столовой.

На рисунке 10 представлена диаграмма последовательности проведения платежа в мобильном приложении.

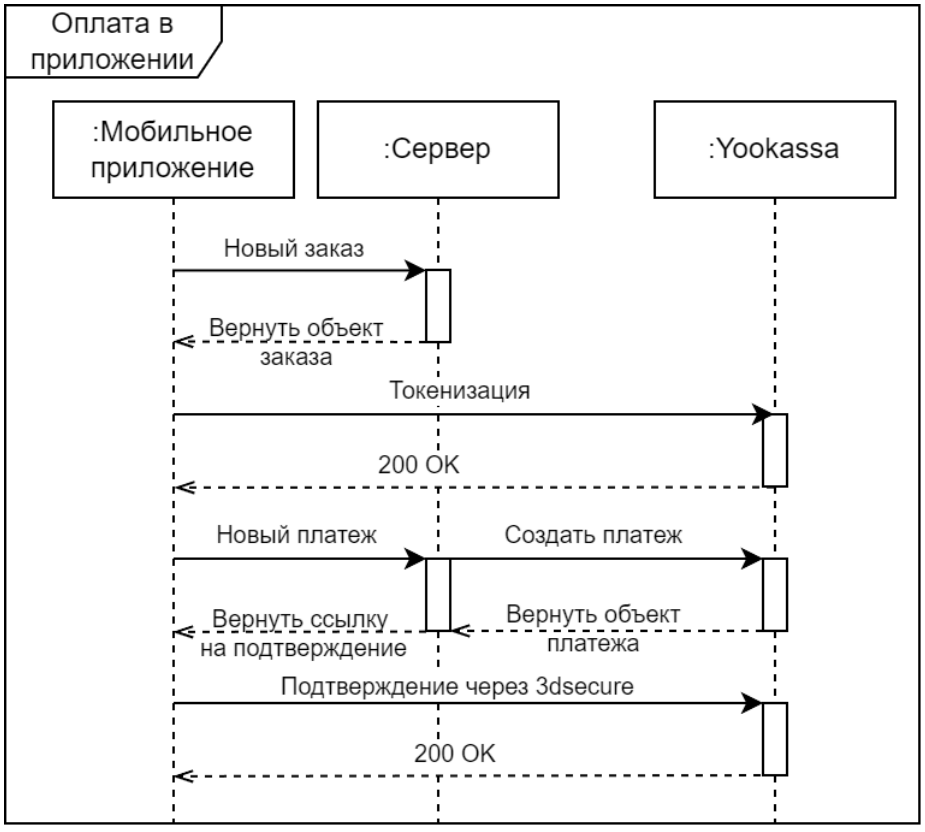

Рисунок 10 – Проведение платежа в мобильном приложении

Мобильное приложение отправляет запрос на создание заказа на сервер. В случае успеха в ответ возвращается модель заказа, а также данные для проведения токенизации (идентификатор магазина, ключи для Yookassa).

Процесс токенизации: приложение отправляет данные, полученные на предыдущем шаге, в Yookassa, пользователь выбирает нужный способ оплаты, вводит данные карты или получает токен СБП, а также у пользователя есть возможность выбрать, сохранить ли способ оплаты для последующих платежей, т.е. подключить автооплату. В случае, если возникли какие-то ошибки, то будет показан виджет ошибки с кнопкой «Повторить». В случае, если все введено верно и не возникло ошибок, Yookassa возвращает в ответ токен платежа. Этот токен передается на сервер, где создается платеж.

Если для данного способа оплаты подключена проверка через код подтверждения, то в ответе от сервера возвращается ссылка на подтверждение оплаты через 3-D Secure [18]. Иначе – приходит пустая строка или null. Если пришла ссылка на подтверждение, то мобильное приложение подтверждает платеж через пакет `yookassa_payments_flutter`, т.е. через WebView открывается веб-страница с возможностью ввода кода подтверждения. Если процесс подтверждения завершен или ссылка на подтверждение не пришла, то происходит перенаправление на экран статуса заказа.

Методы для работы с пакетом были вынесены в отдельный сервис, который представлен в листинге 8.

Листинг 8 – Сервис для работы с Yookassa

```
class PaymentService {
  Future<TokenizationResult> tokenization({
    required CreateOrderResponseModel data,
  }) async {
    var amount = Amount(value: data.order.price.toString(), currency: Cur-
rency.rub);
    var tokenizationModuleInputData = TokenizationModuleInputData(
      clientApplicationKey: data.clientApplicationKey,
      title: 'Заказ №${data.order.id}',
      subtitle: data.order.compound,
      amount: amount,
      shopId: data.shopId,
      moneyAuthClientId: data.moneyAuthClientId,
      savePaymentMethod: SavePaymentMethod.userSelects,
```

```
      applicationScheme: 'vh80732.rdock.ru',
    );
     final result = await YookassaPaymentsFlutter.tokenization(tokeniza-
tionModuleInputData);
    return result;
  }
  Future<void> confirmation({
    required String confirmationUrl,
    required String clientApplicationKey,
    required String shopId,
    PaymentMethod? paymentMethod,
  }) async {
    await YookassaPaymentsFlutter.confirmation(
    confirmationUrl,
    paymentMethod,
    clientApplicationKey,
    shopId,
    );
  }
}
```

Вывод по третьей главе

В ходе реализации было разработано кроссплатформенное мобильное приложение столовой «Восточный дракон» в соответствие с определенными ранее функциональными требованиями и вариантами использованиями системы. Приложение было реализовано с помощью фреймворка Flutter и языка программирования Dart. Для управления состоянием был использован архитектурный паттерн MVVM. Клиентская часть мобильного приложения была интегрирована с серверной частью в соответствии с API. Также в проекте были использованы push-уведомления с использованием Firebase Messaging и проведение оплаты заказа с помощью сервиса для обработки платежей Yookassa.

4. ТЕСТИРОВАНИЕ

Функциональное тестирование

Функциональное тестирование – это тестирование программного обеспечения в целях проверки реализуемости функциональных требований, то есть способности программного обеспечения в определенных задачах решать задачи, необходимые пользователям [19].

Для функционального тестирования приложения была создана страница в магазине приложений Google Play, а также был открыт доступ для внутренних тестировщиков. В качестве команды тестирования выступила команда разработки. Тестировщикам была выслана таблица со списком необходимых тестов и ожидаемыми результатами. Список тестов представлен в таблице 1.

Все тесты были пройдены успешно.

Таблица 1 – Набор тестов

№	Название теста	Шаги	Ожидаемый результат
1	Авторизация с корректными данными	1. Ввести корректный E-mail. 2. Нажать кнопку «Продолжить». 3. Ввести корректный код подтверждения.	Откроется экран ввода личных данных либо каталог
2	Авторизация с некорректным кодом	1. Ввести корректный E-mail. 2. Нажать кнопку «Продолжить». 3. Ввести некорректный код подтверждения.	Появится уведомление о том, что код неверный
3	Авторизация с некорректным E-mail	Ввести некорректный E-mail.	Кнопка «Продолжить» неактивна
4	Корректное отображение товаров в каталоге	1. Открыть каталог. 2. Пролистать все позиции.	Все изображения и весь текст должны отображаться без ошибок верстки
5	Добавление товара в корзину из каталога	1. Открыть каталог. 2. Нажать на кнопку с ценой товара.	Вместо кнопки с ценой товара появилась кнопка «+/-», в низу экрана появилась кнопка перехода в корзину

Продолжение таблицы 1

№	Название теста	Шаги	Ожидаемый результат
6	Добавление товара в корзину из детальной страницы товара	1. Открыть каталог. 2. Нажать на товар. Нажать кнопку «Добавить».	Страница товара закрылась, вместо кнопки с ценой товара появилась кнопка «+/-», в низу экрана появилась кнопка перехода в корзину
7	Добавление собранного обеда в корзину	1. Открыть каталог. 2. Нажать на обед. 3. Выбрать все необходимые опции. 4. Нажать «Добавить».	Страница обеда закрылась, вместо кнопки с ценой товара появилась кнопка «+/-», в низу экрана появилась кнопка перехода в корзину
8	Добавление несобранного обеда в корзину	1. Открыть каталог. 2. Нажать на обед. 3. Ничего не выбирать.	Кнопка внизу экрана затемненная и с текстом «Выберите опции»
9	Добавление/удаление товара на странице корзины	1. Открыть корзину. 2. Нажать на кнопку «+» или «–» возле элемента товара.	Количество товара изменилось, общая сумма изменилась
10	Удаление всей позиции из корзины	1. Открыть корзину. 2. Нажать на кнопку с изображением мусорной корзины возле элемента товара.	Элемент корзины исчез, общая сумма изменилась Если это был единственный товар в корзине, то экран корзины закрылся
11	Изменение данных пользователя	1. Открыть экран профиля. 2. Нажать на поле с текущим именем. 3. Изменить имя. 4. Нажать кнопку «Сохранить» внизу экрана.	На кнопке «Сохранить» появился индикатор загрузки, затем появилось уведомление «Данные успешно изменены»
12	Создание заказа	1. Добавить товар в корзину. 2. Перейти в корзину. 3. Нажать «Перейти к оплате». 4. Нажать «Оплатить». 5. Нажать кнопку «Назад»	Произойдет перенаправление на экран неуспешного статуса заказа. В списке заказов появится заказ со статусом «Не оплачен»
13	Оплата заказа	1. Добавить товар в корзину. 2. Перейти в корзину. 3. Нажать «Перейти к оплате». 4. Нажать «Оплатить». 5. Выбрать способ оплаты. 6. Ввести данные для оплаты.	Произойдет перенаправление на экран успешного статуса заказа. В списке заказов появится заказ со статусом «Готовится»

Окончание таблицы 1

№	Название теста	Шаги	Ожидаемый результат
14	Смена статуса заказа	1. Добавить товар в корзину. 2. Перейти в корзину. 3. Нажать «Перейти к оплате». 4. Нажать «Оплатить». 5. Выбрать способ оплаты. Ввести данные для оплаты. 6. Подождать примерно 7 минут. Перейти на страницу «Заказы».	В списке заказов новый заказ будет со статусом «Готов».

Тест респонсивной верстки

Для тестирования корректного отображения элементов на экране в приложении на Flutter обычно используют Golden-тесты [20]. Golden-тест проверяет отдельные виджеты и целый экран. Визуальное представление компонента сравнивается с предыдущими результатами тестов.

Данный вид тестов важен для кроссплатформенных приложений, так как пользовательский интерфейс должен оставаться консистентным и удобным независимо от устройства, на котором он используется. Также важно автоматизировать этот процесс, чтобы сократить время на тестирование и повысить общую эффективность работы команды разработчиков

Для автоматического тестирования верстки экранов приложения столовой «Восточный дракон» был использован пакет golden_toolkit. С его помощью можно отрисовать виджет на различных разрешениях экрана, пакет сделает скриншоты и сохранит их в один png файл, а разработчик сможет сравнить полученные результаты.

Для теста были выбраны следующие разрешения: 375 на 667, 414 на 896, 1125 на 2436, 1440 на 3200 и 1080 на 2340. Это разрешения популярных на данный момент мобильных девайсов: iPhone X, iPhone 11, Samsung Galaxy S20, Google Pixel 4a. Результаты тестирования экрана корзины показаны на рисунке 11.

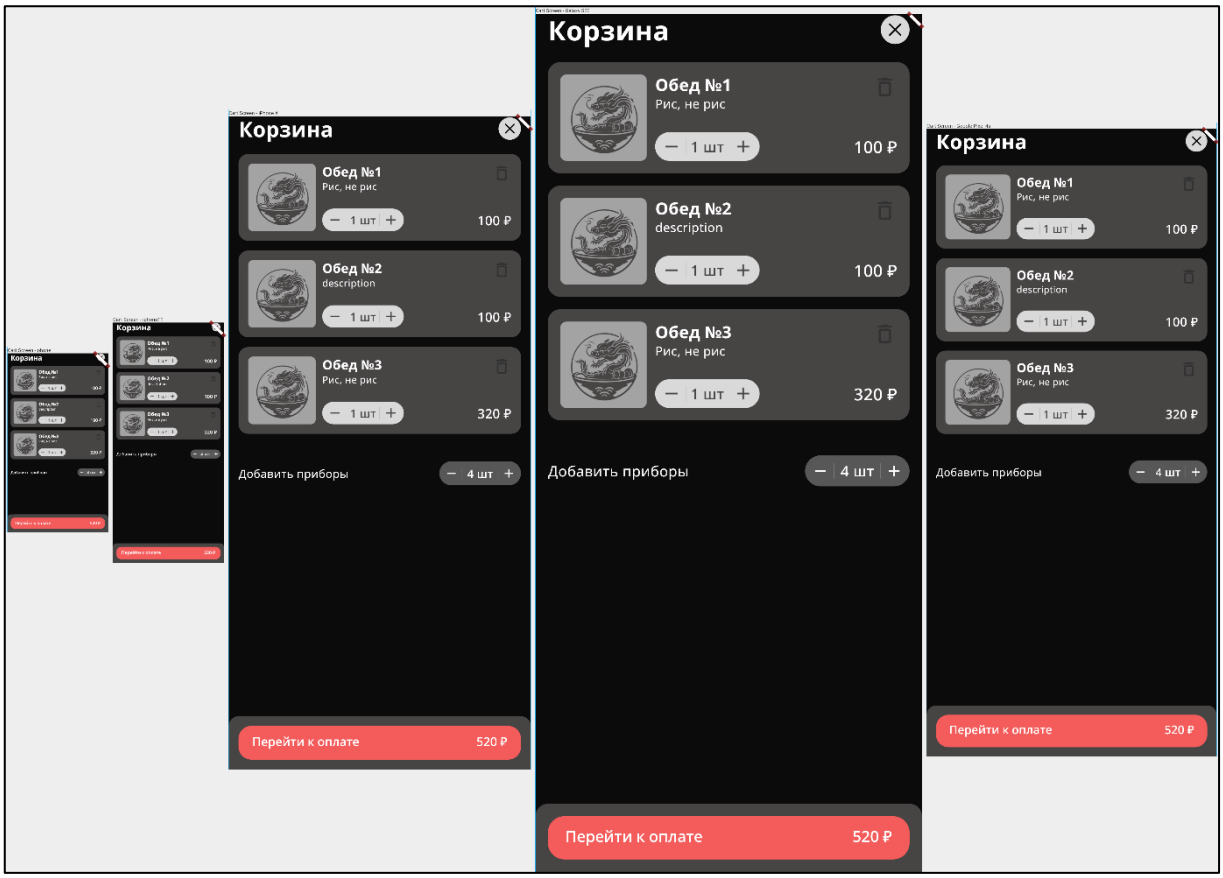

Рисунок 11 – Golden – тест экрана корзины

Из скриншотов видно, что элементы выглядят одинаково на разных разрешениях экрана.

Таким же образом были протестированы экраны авторизации и каталога. Результаты представлены в приложении В. Остальные экраны не имело смысла тестировать, ввиду малого количества элементов на них.

Вывод по четвертой главе

Было проведено функциональное тестирование мобильного приложения, а также тестирование респонсивной верстки. Оба вида тестирования были пройдены успешно.

ЗАКЛЮЧЕНИЕ

В рамках данной выпускной квалификационной работы было разработано кроссплатформенное мобильное приложение для столовой «Восточный дракон». В ходе реализации были решены следующие задачи.

1. Изучена предметная область.
2. Проведен обзор аналогичных приложений.
3. Спроектирована архитектура мобильного приложения.
4. Спроектирован UI/UX-дизайн для мобильного приложения.
5. Разработана клиентская часть мобильного приложения.
6. Протестировано мобильное приложение.

В ходе реализации была изучена архитектура MVVM для разработки кроссплатформенных мобильных приложений, а также была произведена интеграция со сторонними сервисами: Firebase Cloud Messaging для получения push-уведомлений и Yookassa для проведения онлайн платежей.

В дальнейшем планируется расширение функционала приложения. В частности, рассматривается возможность интеграции с платежными терминалами в заведении, что позволит клиентам оплачивать заказы непосредственно через приложение, находясь в столовой. Это не только улучшит пользовательский опыт, но и сократит время обслуживания. Также планируется смена способа авторизации на более быстрый и удобный, например, с помощью ввода кода из СМС. Такой подход обеспечит дополнительное удобство и безопасность для пользователей, упрощая процесс входа в приложение. В дополнение к этому, в ближайшем будущем планируется внедрение программы лояльности, которая будет включать систему накопительных бонусов и специальных предложений для постоянных клиентов. Это не только повысит лояльность клиентов, но и будет способствовать увеличению количества повторных заказов.

ЛИТЕРАТУРА

1. Особенности разработки приложений для сферы фудтех в 2023 году. Как создать IT-продукт, сэкономив время, деньги и нервы. [Электронный ресурс] URL: https://vc.ru/u/331647-elena-nazarova/602672-osobennosti-razrabotki-prilozheniy-dlya-sfery-fudteh-v-2023-godu-kak-sozdat-it-produkt-sekonomiv-vremya-dengi-i-nervy (дата обращения: 11.05.2024 г.).

2. Еда будущего: что такое фудтех и зачем он нужен. [Электронный ресурс] URL: https://cek.vkusvill.ru/media/journal/eda-budushchego-chto-takoe-fudtekh-i-zachem-on-nuzhen.html (дата обращения: 11.05.2024 г.).

3. Что такое маркетплейс: все, что нужно знать новичку. [Электронный ресурс] URL: https://partner.market.yandex.ru/blog/start-on-marketplace/chto-takoe-marketplace/ (дата обращения: 11.05.2024 г.).

4. Почему не нужно запускать агрегатор доставки еды? [Электронный ресурс] URL: https://vc.ru/opinions/287294-pochemu-ne-nuzhno-zapuskat-agregator-dostavki-edy (дата обращения: 11.05.2024 г.).

5. Ступина Т.А., Грицунова С.В. Инновационные технологии в ресторанном бизнесе: фирменные мобильные приложения. // Оригинальные исследования, 2016. – Т. 2, № 2. – С. 35–36.

6. Жуковская А.Н., Заушицина А.С. Особенности разработки кроссплтаформенных мобильных приложений. // Решетневские чтения, 2017. – №21. – С. 330–331.

7. Мишагин Д.В. Сравнительный анализ кроссплатформенных технологий для разработки мобильных приложений. // Оригинальные исследования, 2020. – № 5. – С. 189–198.

8. Документация фреймворка ReactNative. [Электронный ресурс] URL: https://reactnative.dev/docs/getting-started (дата обращения: 17.05.2024 г.).

9. Документация фреймворка Flutter. [Электронный ресурс] URL: https://flutter.dev/ (дата обращения: 17.05.2024 г.).

10. Документация фреймворка Cordova [Электронный ресурс] URL: https://cordova.apache.org/docs/en/latest/ (дата обращения: 17.05.2024 г.).

11. Фаулер М. UML. Основы. 3-е издание. // Символ-Плюс, 2005, 192 с.

12. Раджабов Н.Н., Рысин М.Л. Чистая архитектура и паттерн MVVM в практике разработки Android-приложения. // Актуальные проблемы науки и образования в условиях современных вызовов, 2023. – С. 21–27.

13. Документация библиотеки Elementary. [Электронный ресурс] URL: https://pub.dev/packages/elementary/ (дата обращения: 17.05.2024 г.).

14. Figma Help-Center. // Figma. [Электронный ресурс].URL: https://help.figma.com/hc/en-us (дата обращения: 17.05.2024 г.).

15. Massé M. REST API Design Rulebook. // O'Reilly Media, Inc., 2012. – 96 с.

16. Push уведомления. Как они работают и для чего они нужны. [Электронный ресурс] URL: https://habr.com/ru/articles/706190/ (дата обращения: 17.05.2024 г.).

17. Документация Firebase. [Электронный ресурс] URL: https://firebase.google.com/docs/ (дата обращения: 17.05.2024 г.).

18. Михайлова О.Ю. Современные технологии защиты онлайн платежей. // Современные информационные технологии и информационная безопасность, 2022. – С. 93–97.

19. Функциональное тестирование программного обеспечения. [Электронный ресурс] URL: https://unetway.com/tutorial/funkcionalnoe-testirovanie (дата обращения: 14.06.2024 г.).

20. Introduction to golden tests in Flutter. [Электронный ресурс] URL: https://medium.com/appunite-edu-collection/intro-to-golden-tests-in-flutter-3d23bb12d056/ (дата обращения: 14.06.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Макеты окон приложения

Спроектированные окна приложения представлены на рисунках 1–5.

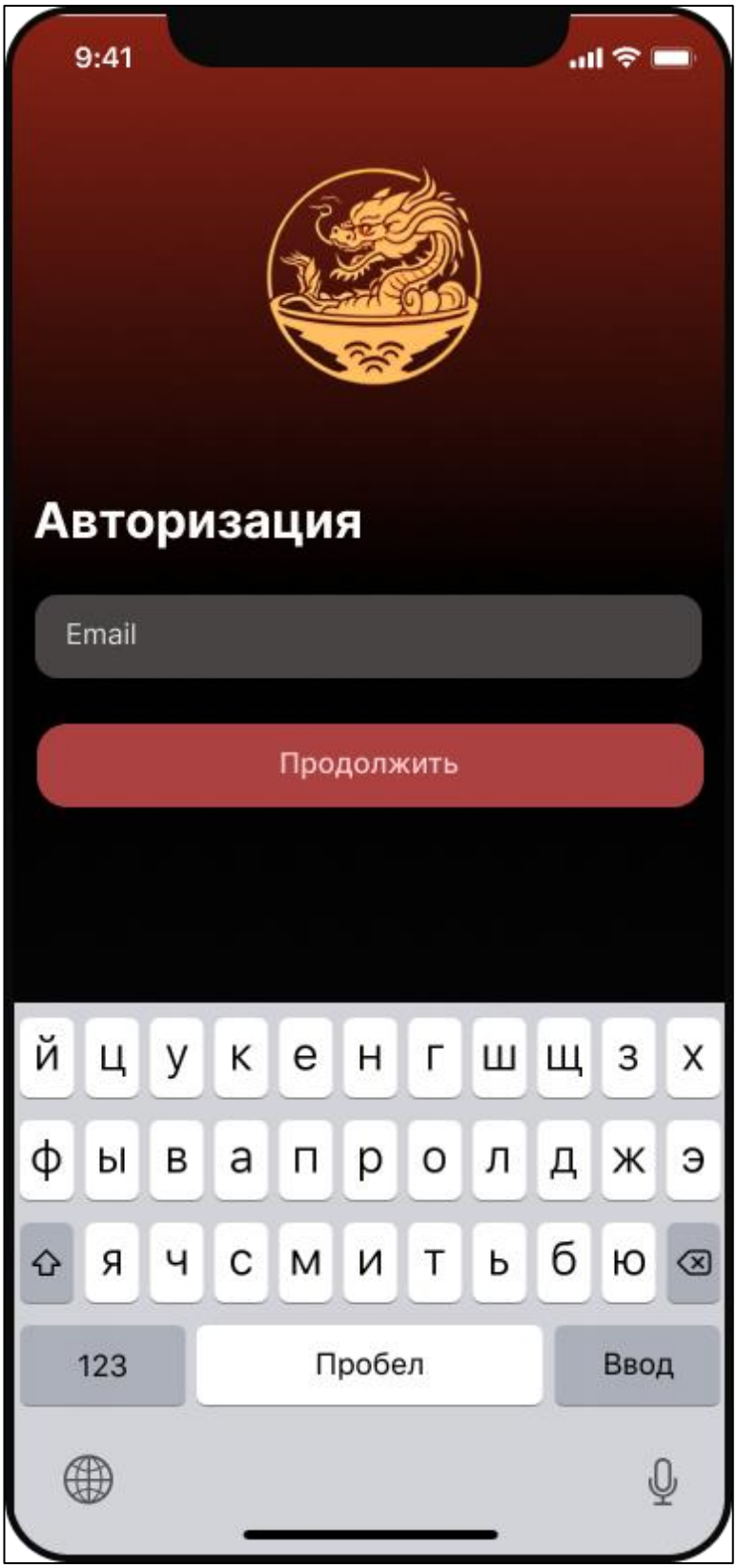

Рисунок 1 – Макет экрана авторизации

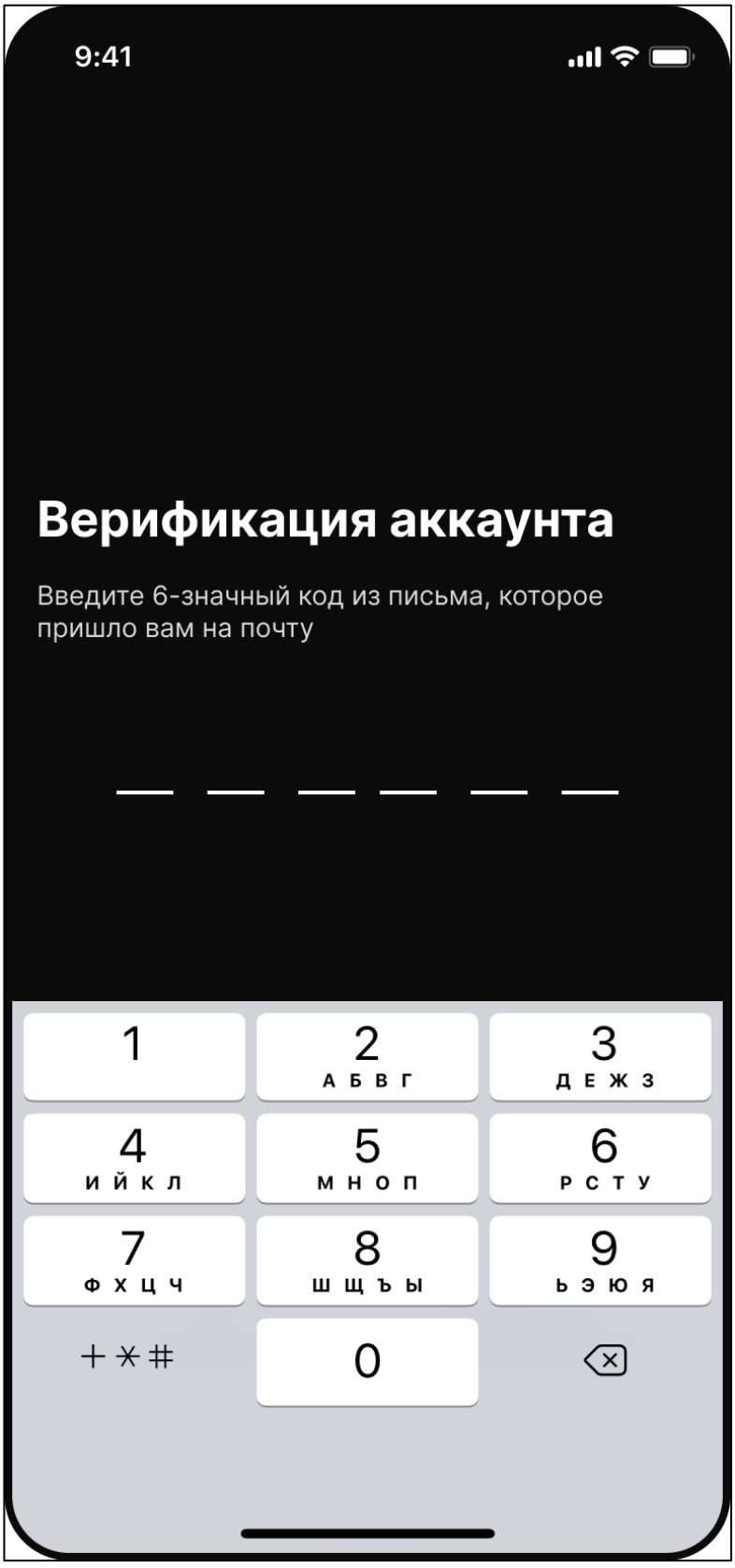

Рисунок 2 – Макет экрана ввода кода

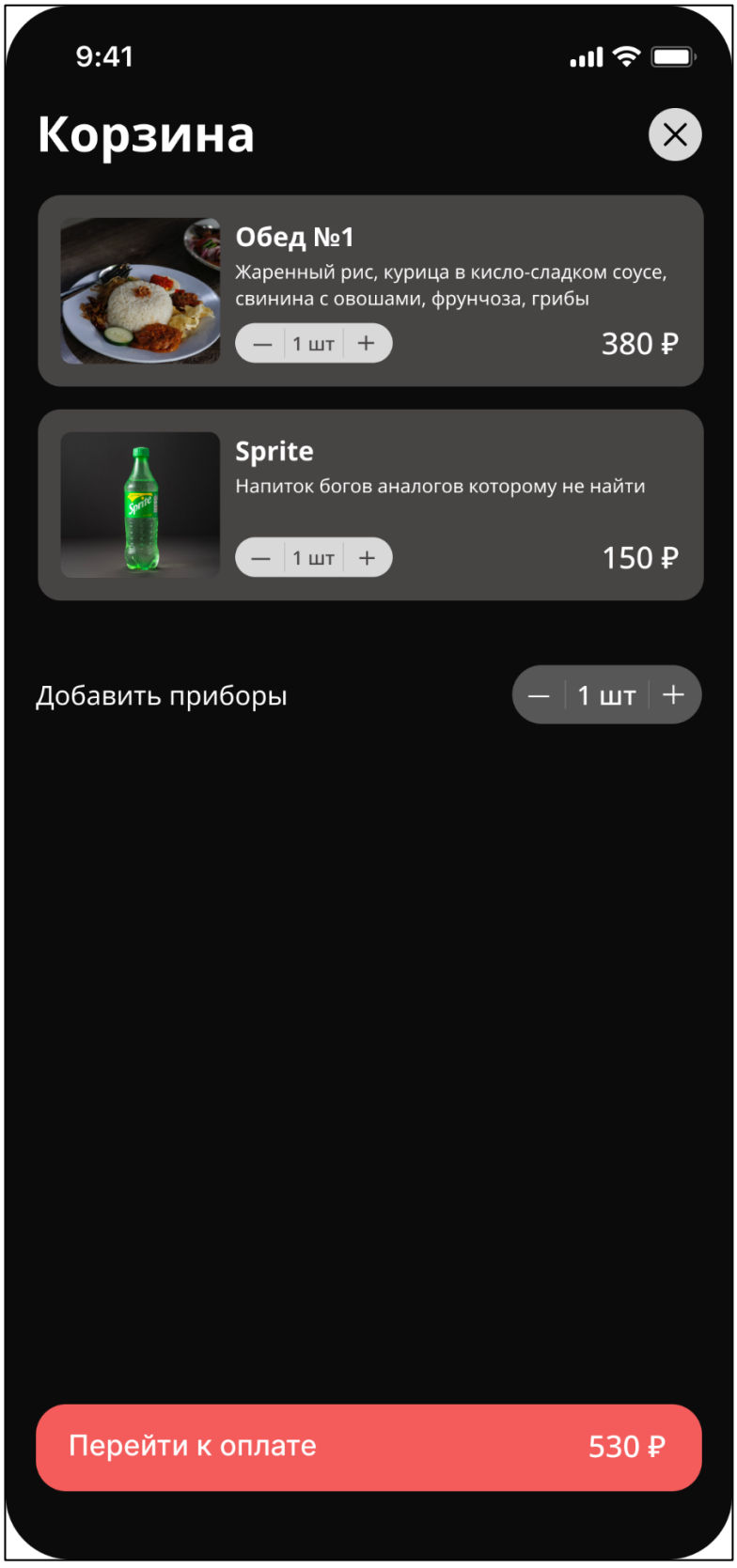

Рисунок 3 – Макет экрана корзины

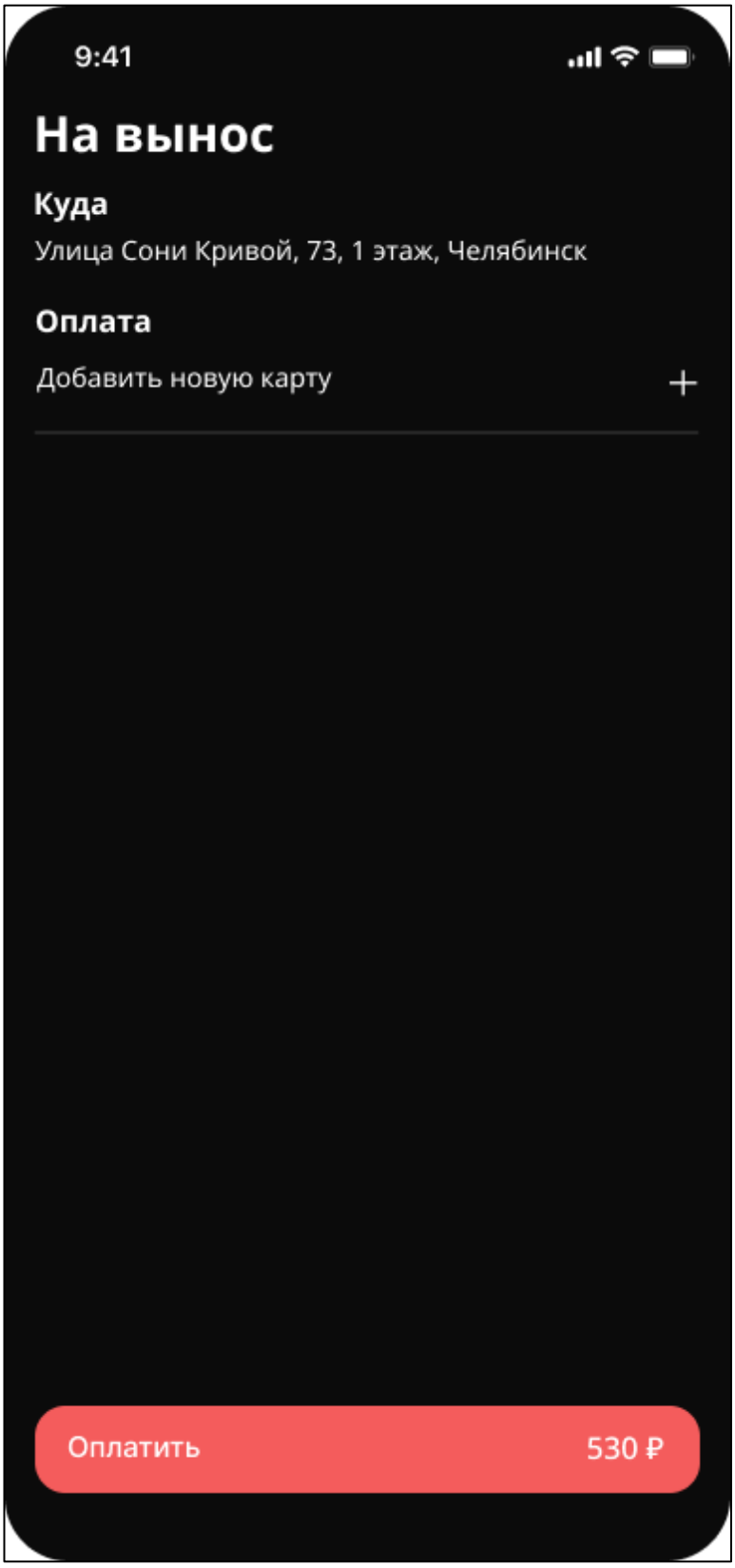

Рисунок 4 – Макет экрана оформления заказа

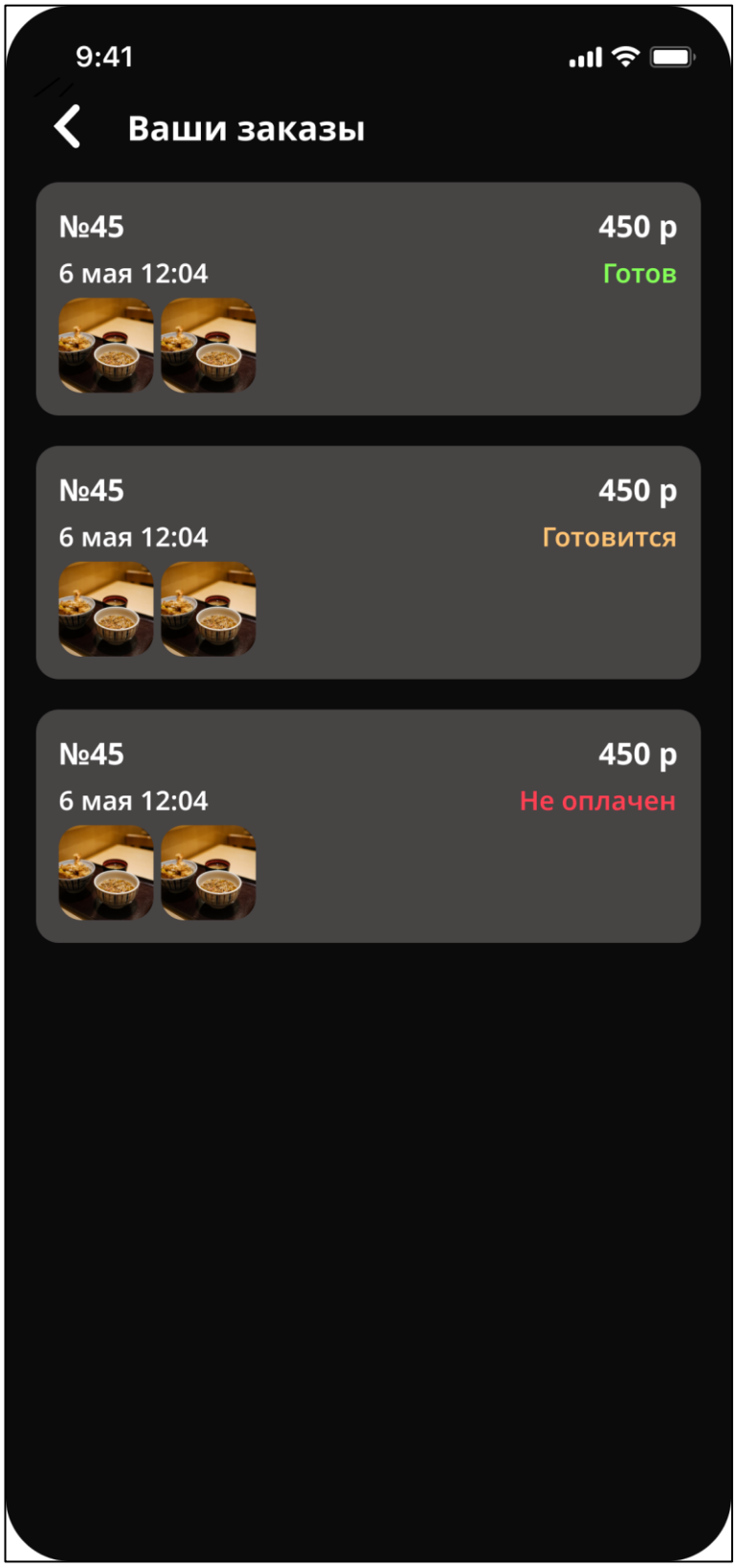

Рисунок 5 – Макет экрана списка заказов

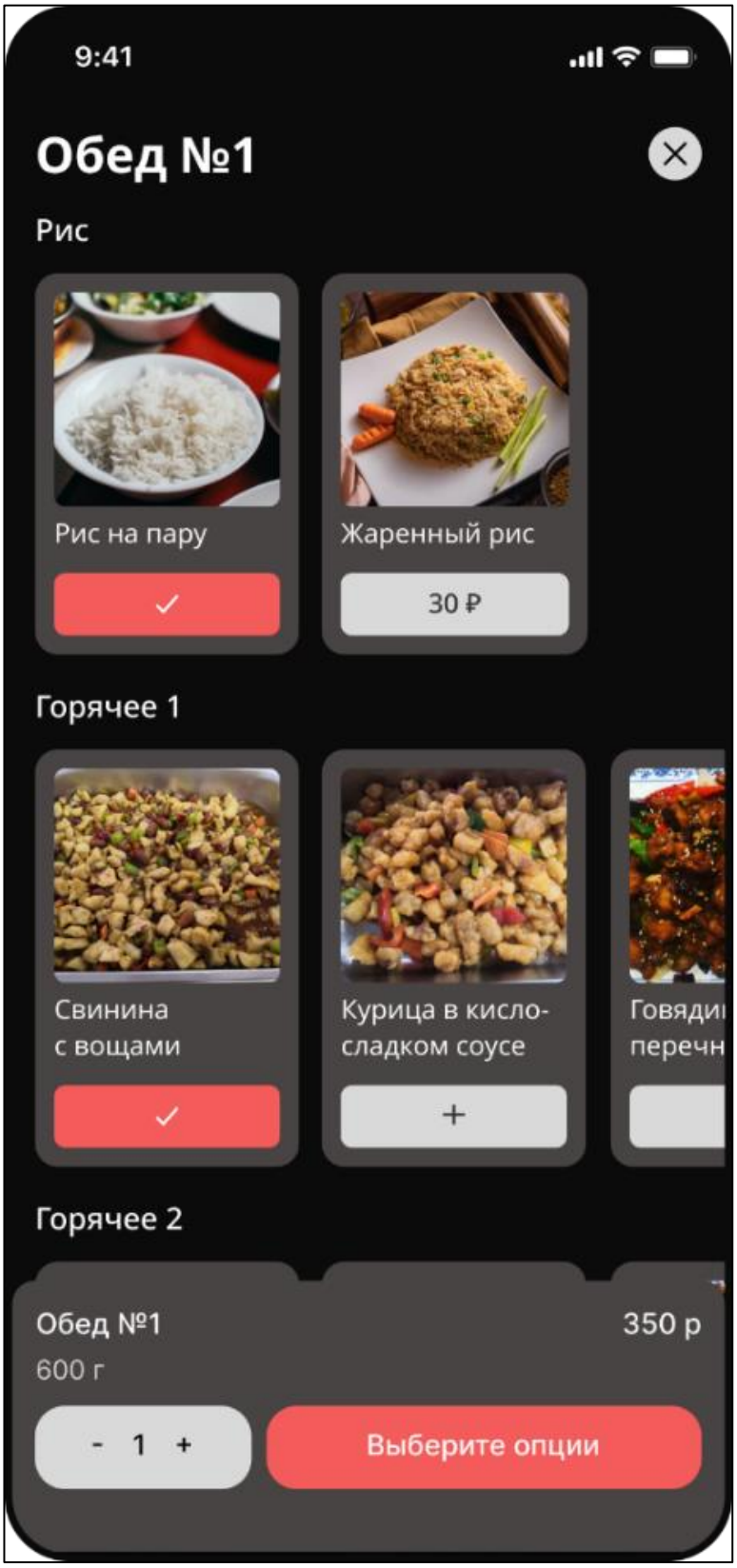

Рисунок 6 – Макет экрана обеда

Приложение Б. Листинги основных моделей

Код основных моделей, используемых в приложении, представлен в листингах 1–6.

Листинг 1 – Базовая модель ответа на запрос

```
@JsonSerializable()

/// Модель ответа на запрос
class BaseResponseModel {
  const BaseResponseModel({
    required this.data,
    required this.success,
    this.code,
    this.message,
  });

  factory BaseResponseModel.fromJson(Map<String, dynamic> json) {
    final code = json['code'] as int?;
    if (code == 401 || code == 403) {
      throw AccessException(
        'Ошибка доступа',
        StackTrace.current,
      );
    }

    if (json['success'] is! bool) {
      throw ResponseParseException(
        'Ответ от сервера не содержит success',
        StackTrace.current,
      );
    }

    if (json['success'] == false) {
      throw SuccessFalse(
        json['message'] as String? ?? 'Произошла ошибка',
        StackTrace.current,
      );
    }

    try {
      final res = _$BaseResponseModelFromJson(json);

      return res;
    } catch (e) {
      throw ResponseParseException('BaseResponseModel: $e');
    }
  }

  factory BaseResponseModel.empty() {
    return const BaseResponseModel(
      data: <dynamic>[],
      success: true,
    );
  }

  /// данные в ответе на запрос
  final dynamic data;

  /// результат выполнения запроса
  final bool success;
```

```
  /// некое сообщение
  /// обычно присутствует если [success] == false
  final String? message;

  final int? code;

  Map<String, dynamic> toJson() => _$BaseResponseModelToJson(this);
}
```

Листинг 2 – Модель пользователя

```
import 'package:eastern_dragon/core/common/data/exceptions/re-
sponse_parse_exception.dart';
import 'package:json_annotation/json_annotation.dart';

part 'user_model.g.dart';

@JsonSerializable(createToJson: false)

/// Сущность пользователя
class UserModel {
  /// id
  final int id;

  /// Адрес почты
  final String? email;

  /// Имя пользователя
  final String? name;

  UserModel({
    required this.id,
    this.email,
    this.name,
  });

  factory UserModel.fromJson(Map<String, dynamic> json) {
    try {
      return _$UserModelFromJson(json);
    } on Object catch (e) {
      throw ResponseParseException('UserModel: $e');
    }
  }

  UserModel copyWith({
    String? email,
    String? name,
  }) {
    return UserModel(
      id: id,
      email: email ?? this.email,
      name: name ?? this.name,
    );
  }
}
```

Листинг 3 – Детальная модель обеда

```
class LunchDetailModel {
  final String name;

  final String? image;

  final List<LunchIngridientModel> ingredients;

  LunchDetailModel({
    required this.name,
    required this.ingredients,
    this.image,
  });

  factory LunchDetailModel.fromJson(Map<String, dynamic> json) {
    try {
      return _$LunchDetailModelFromJson(json);
    } on Object catch (e) {
      throw ResponseParseException('LunchDetailModel: $e');
    }
  }
}
```

Листинг 4 – Модель ингредиента обеда

```
class LunchIngridientModel {
  final String name;

  final List<LunchIngridientOptionModel> options;

  LunchIngridientModel({
    required this.name,
    required this.options,
  });

  factory LunchIngridientModel.fromJson(Map<String, dynamic> json) {
    try {
      return _$LunchIngridientModelFromJson(json);
    } on Object catch (e) {
      throw ResponseParseException('LunchIngridientModel: $e');
    }
  }
}
```

Листинг 5 – Модель варианта ингредиента

```
class LunchIngridientOptionModel {
  final int id;

  final String name;

  final String? image;

  final num extraPrice;

  LunchIngridientOptionModel({
    required this.id,
```

```
    required this.name,
    required this.extraPrice,
    this.image,
  });

  factory LunchIngridientOptionModel.fromJson(Map<String, dynamic> json) {
    try {
      return _$LunchIngridientOptionModelFromJson(json);
    } on Object catch (e) {
      throw ResponseParseException('LunchIngridientOptionModel: $e');
    }
  }
}
```

Листинг 6 – Модель корзины

```
@JsonSerializable(createToJson: false)
class CartModel {
  final List<CartItemModel> items;

  final num totalPrice;

  CartModel({
    required this.items,
    required this.totalPrice,
  });

  CartModel.empty()
      : items = [],
        totalPrice = 0;

  factory CartModel.fromJson(Map<String, dynamic> json) {
    try {
      return _$CartModelFromJson(json);
    } on Object catch (e) {
      throw ResponseParseException('CartModel: $e');
    }
  }
}
```

Приложение В. Результаты Golden – тестов

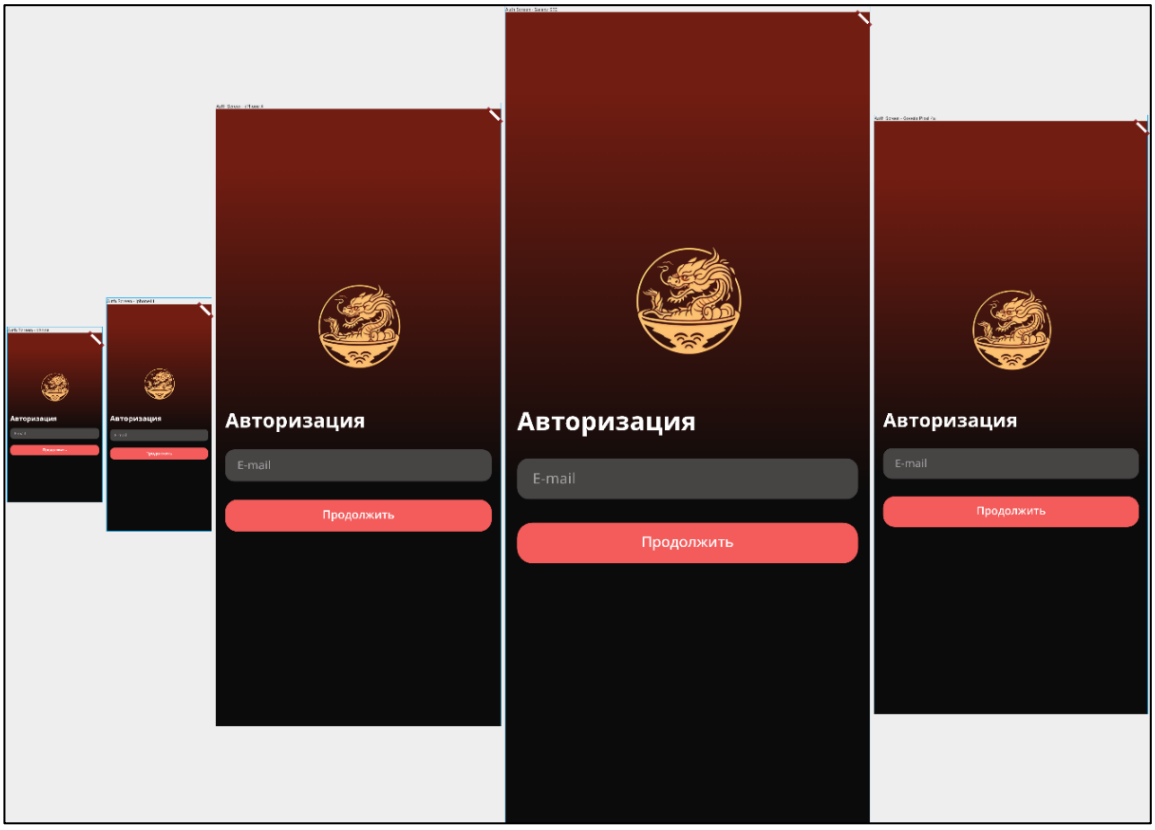

Рисунок 1 – Golden – тест экрана авторизации

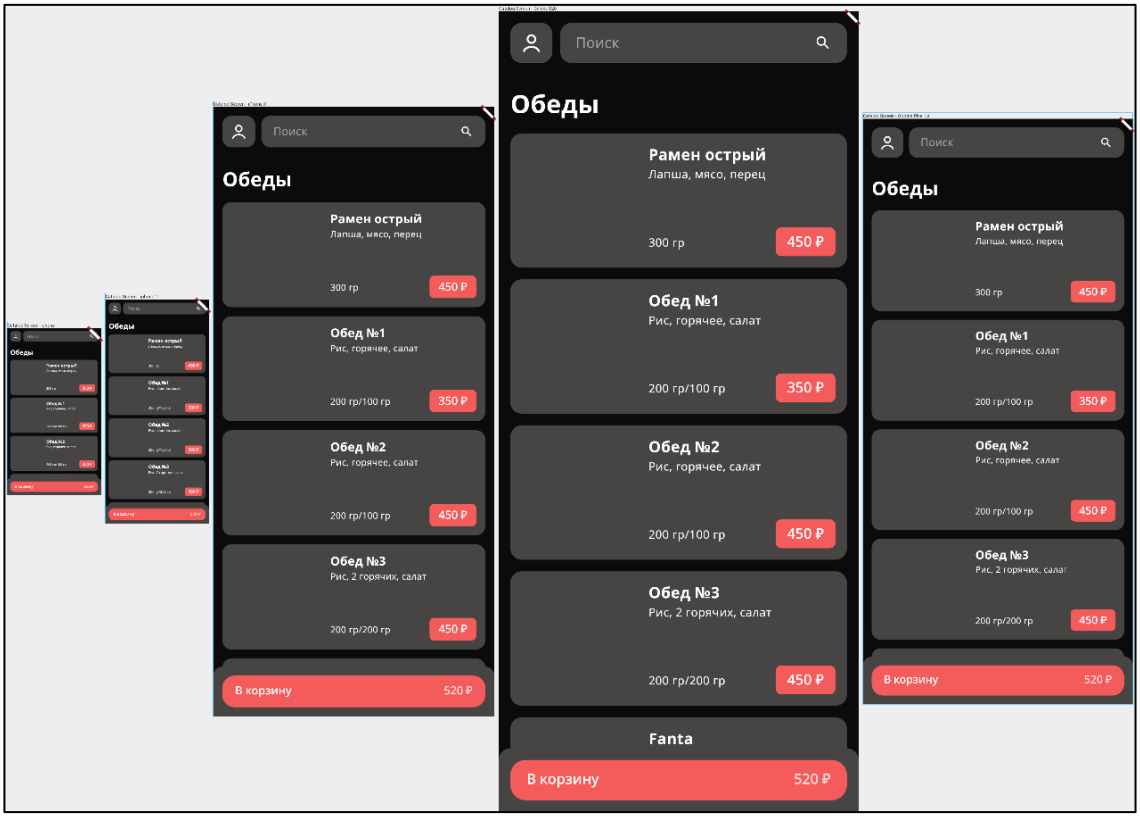

Рисунок 2 – Golden – тест экрана каталога