

**МИНОБРНАУКИ РОССИИ**  
**Федеральное государственное бюджетное образовательное**  
**учреждение высшего образования**  
**«Чувашский государственный университет имени И.Н. Ульянова»**  
**(ФГБОУ ВО «ЧГУ им. И.Н. Ульянова»)**

**Факультет информатики и вычислительной техники**  
**Кафедра вычислительной техники**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**  
**(БАКАЛАВРСКИЙ ПРОЕКТ)**  
по направлению подготовки  
09.03.01 Информатика и вычислительная техника,  
направленность (профиль) «Программное обеспечение средств  
вычислительной техники и автоматизированных систем»

на тему *«Разработка кроссплатформенного приложения магазина-поставщика на flutter»*

Обучающийся

\_\_\_\_\_  
подпись, дата

Петров А.А.

\_\_\_\_\_  
ФИО

Руководитель,  
доцент кафедры  
вычислительной техники,  
к.т.н., доцент

\_\_\_\_\_  
подпись, дата

Обломов И.А.

\_\_\_\_\_  
ФИО

Заведующий кафедрой  
вычислительной техники,  
к. пед.н., доцент

\_\_\_\_\_  
подпись, дата

Щипцова А.В.

\_\_\_\_\_  
ФИО

Работа выполнена на базе ООО «Бокус»

Чебоксары 2025

## АННОТАЦИЯ

Данная выпускная квалификационная работа посвящена разработке кроссплатформенного мобильного приложения для магазина-поставщика строительных материалов «binevir» для одноименной компании с разработкой строительного калькулятора для расчета товаров по заданным параметрам.

Стек технологий включает в себя кроссплатформенный фреймворк Flutter для разработки кроссплатформенных решений. Использовалась Figma для проектирования пользовательского интерфейса, среда разработки Visual Studio Code с встроенной системой контроля версий.

В ходе работы был проведен анализ предметной области, определены требования и тех задание, разработан интерфейс и функциональный калькулятор, а также настроено взаимодействие с сервером. Последний этап включал тестирование приложения.

## ANNOTATION

This graduate qualification work is dedicated to the development of a cross-platform mobile application «binevir», same as company's name, construction materials supplier store, featuring the implementation of a construction calculator for product estimation based on specified parameters.

The technology stack includes the Flutter framework for cross-platform development. Figma was used for UI design, and Visual Studio Code served as the primary IDE with integrated version control.

The work involved domain analysis, requirement specification, and technical task definition, followed by the development of the user interface and functional calculator, along with server integration setup. The final stage encompassed application testing.

# СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ.....	6
ВВЕДЕНИЕ .....	8
1. Аналитический раздел .....	10
1.1 Описание предметной области .....	10
1.2 Обоснование целесообразности и постановка задачи на разработку .....	11
1.3 Техническое задание на разработку ПО .....	13
1.3.1 Наименование программы .....	13
1.3.2 Основание для разработки .....	13
1.3.3 Назначение .....	14
1.3.4 Требования .....	14
1.3.4.1 Функциональные требования .....	14
1.3.4.2 Требования по надежности .....	15
1.3.4.3 Требования к условиям эксплуатации .....	16
1.3.4.4 Требования к составу технических средств .....	16
1.3.4.5 Требования по эргономике и технической эстетике .....	16
1.3.5 Безопасность при эксплуатации ПЭВМ .....	17
1.3.6 Стадии и этапы разработки .....	17
Выводы по разделу .....	17
2 Проектно-конструкторский раздел .....	19
2.1 Выбор средств разработки .....	19
2.1.1 Выбор подхода к разработке .....	19
2.1.2 Выбор фреймворка .....	20
2.1.3 Принципы работы фреймворка Flutter .....	22
2.1.4 Выбор языка программирования .....	24
2.1.5 Библиотеки .....	26
2.1.6 Выбор среды разработки .....	28
2.1.7 Выбор среды тестирования .....	28
2.2 Проектирование архитектуры .....	29
2.2.1 Архитектура серверной части .....	29

2.2.2	Архитектура приложения .....	30
2.3	Проектирование пользовательского интерфейса .....	32
2.4	Разработка мобильного приложения .....	35
2.4.1	Устройство структуры Flutter-проекта .....	35
2.4.2	Построение дерева экранов .....	37
2.4.3	Разработка пользовательского интерфейса .....	43
2.4.4	Взаимодействие с сервером .....	46
2.4.5	Работа калькулятора .....	55
	Выводы по разделу .....	61
3	Экспериментальный раздел .....	63
3.1	Тестирование .....	63
3.1.1	Экраны приложения .....	65
3.1.1.1	Сплэш-экран .....	65
3.1.1.2	Онбординг-экран .....	66
3.1.1.3	Главный экран .....	68
3.1.1.4	Экран каталог, .....	70
3.1.1.5	Экран с подробной информацией о товаре .....	71
3.1.1.6	Экран профиля .....	72
3.1.1.7	Экран калькулятора .....	74
3.2	Проверка других функциональных требований .....	76
3.2.1	Корректное лого, нативный сплэш, наименование приложения .....	76
3.2.2	Поддержка локализации .....	76
3.2.3	Поддержка светлой и темной темы .....	77
3.2.4	Поддержка разных систем измерений .....	79
3.2.5	Загрузка фото в профиль .....	80
	Выводы по разделу .....	80
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	83
	ПРИЛОЖЕНИЕ А .....	85
	ПРИЛОЖЕНИЕ Б .....	86
	ПРИЛОЖЕНИЕ В .....	91
	ПРИЛОЖЕНИЕ Г .....	94

## ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей работе применяются следующие термины с соответствующими определениями.

Android	Операционная система для смартфонов, а также других цифровых устройств, например телевизоры
iOS	Операционная система для смартфонов и других устройств, выпускаемых компанией Apple
Google	Транснациональная корпорация из США, занимающаяся разработкой, развитием и инвестированием в технические продукции.
Фреймворк	Программное обеспечение, облегчающее разработку за счёт использования готовых компонентов и методов
Flutter	Фреймворк для создания кроссплатформенных приложений, разрабатывается и развивается корпорацией Google, использует язык программирования Dart.
Вёрстка	Процесс организации и построение пользовательского интерфейса
Виджет	Наименьший целый компонент, выполняющий определенные функции. Используются для отображения информации

В настоящей работе применяют следующие сокращения и обозначения.

Сокращение	Расшифровка
API	Application Programming Interface – программный интерфейс, описание методов взаимодействия одной программы с другой
BLoC	Business Logic Component – принцип управления состояниями, разделяющий бизнес-логику и пользовательский интерфейс
JSON	JavaScript Object Notation – текстовый формат обмена данными в виде объектов JavaScript
UI	User Interface – пользовательский интерфейс
ОС	Операционная система
ПО	Программное обеспечение
ПЭВМ	Персональная электронно-вычислительная машина
БД	База данных

## ВВЕДЕНИЕ

Современные тенденции развития электронной коммерции требуют от компаний все большего взаимодействия с клиентами. Использование программного обеспечения позволяет повысить лояльность, упростить взаимодействие, оптимизировать бизнес-процессы, увеличить доступность информации, а также иметь специфические применения или функционал. Мобильные приложения в этом плане имеют сильное преимущество, благодаря распространённости исполнительных устройств, почти каждый имеет в распоряжении смартфон.

Необходимость иметь собственное ПО есть у бизнесов разных сфер, в том числе строительных. Им важно повысить эффективность взаимодействия между поставщиком и клиентом, а также автоматизировать рутинные процессы – от подбора материалов, до расчета их количества. Разработка собственного мобильного приложения позволяет учесть требования нетипичные требования бизнеса, снизить зависимость от подписочных и сторонних решений, а также повысить совместимость с существующими ИТ-продуктами.

Цель выпускной квалификационной работы – разработка кроссплатформенного приложения магазина поставщика с возможностью расчёта количества продукции с использованием встроенного калькулятора.

Для достижения цели были поставлены следующие задачи:

- провести анализ предметной области;
- обосновать целесообразность разработки;
- определить функциональные и нефункциональные требования к приложению;
- выбрать инструменты реализации;



- разработать кроссплатформенное мобильное приложение с соответствующими требованиями, организовать работу с сервером и функционал строительного калькулятора;

- провести тестирование и отладку функционала и отображения.

Выпускная квалификационная работа состоит из трех разделов, решающих поставленные задачи:

- в аналитическом разделе рассматривается востребованность прикладных мобильных приложений в бизнесе и составляется техническое задание;

- в проектно-конструкторском происходит выбор средств разработки и отладки, проектирование приложения, разработка пользовательского интерфейса и функционала приложения;

- в экспериментальном разделе проводится тестирование приложения, описываются разработанные виджеты и экраны, проверяется соответствие требованиям.

В завершении представлено заключение, подводящее итоги проделанной работы, описываются достигнутые результаты и обозначаются возможности дальнейшего развития.

# 1 Аналитический раздел

## 1.1 Описание предметной области

Современный рынок строительных материалов требует от поставщиков высокой скорости обслуживания, прозрачности в коммуникации и простоты взаимодействия с клиентами. Мобильные приложения чрезвычайно эффективным инструментом для достижения этих целей.

В данной предметной области рассматривается разработка мобильного приложения для компании «Биневир». Она уже сотрудничала с профильной организацией, на базе которой выполняется выпускная квалификационная работа. Ранее для неё был разработан сайт-визитка [1]. Часть главного экрана сайта представлена на рисунке 1.

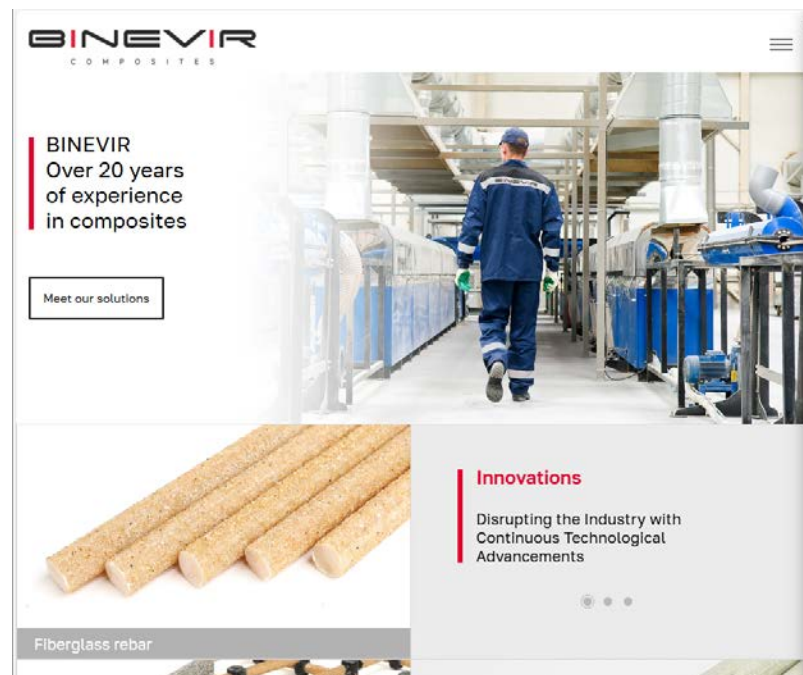


Рисунок 1 – Изображение сайта-визитки Биневир

Компания является международным производителем и поставщиком строительных материалов, ориентированной на строительные компании. Задача сайта-визитки – представление организации и подробное описание производимой продукции.

Одной из главных задач приложения является автоматизация и ускорение процессов расчёта количества материалов, необходимых для реализации строительных проектов. Для этого в приложение встроен специализированный строительный калькулятор, позволяющий быстро производить вычисления по заданным параметрам (применение, толщина, длина, объём и т.д.).

Мобильное приложение также играет важную роль в повышении качества клиентского сервиса. Оно обеспечивает удобную навигацию по товарам, предоставляет быстрый доступ к информации о продукции, ценам, сферах применения и характеристикам. Наличие такой платформы повышает доступность поставщика, позволяет клиентам спланировать покупки.

Кроме того, цифровизация бизнес-процессов способствует повышению лояльности клиентов, ускоряет процесс принятия решений и минимизирует ошибки при оформлении заказов. Использование мобильных решений значительно оптимизирует внутренние процессы компании, снижая нагрузку на менеджеров и других специалистов.

Таким образом, разработка и внедрение мобильного приложения в сфере продажи строительных материалов – это шаг к повышению конкурентоспособности, улучшению клиентского опыта и созданию более гибкой, удобной и эффективной бизнес-модели.

## **1.2 Обоснование целесообразности и постановка задачи на разработку**

Наличие мобильного приложения у бизнеса сегодня это важное конкретное преимущество. Однако перед большинством компаний встаёт

выбор: использовать готовое коммерческое решение или разрабатывать собственное приложение. В рамках рассматриваемой задачи создание своего продукта представляется более целесообразной по ряду причин.

Во-первых, финансовые затраты на использовании сторонних решений могут значительно превышать стоимость разработки. Готовые платформы требуют регулярной оплаты подписки, комиссии с продаж и прочее, а также могут быть ограничены в добавлении специфичных модулей. Это может быть удобным, если бизнес только запускается и перспективы туманны, в этом же случае компания уже функционирует и обладает клиентской базой. Разработка собственного приложения позволяет инвестировать средства в уникальный продукт, который полностью соотносится с существующими бизнес-процессами и не требует постоянных платежей.

Во-вторых, готовые решения имеют ограниченный набор функций и могут не подходить по потребностям компаний. В случае магазина-поставщика строительных материалов важно наличие специализированного строительного калькулятора, поддерживающего расчёт по специальным формулам по пользовательским параметрам.

В-третьих, разрабатываемое решение будет иметь интеграцию с существующими инфраструктурой, разработанной ранее для сайта-визитки. Новое приложение будет использовать тот же сервер, используя те же данные, что сэкономит ресурсы на интеграцию.

Таким образом, разработка нового мобильного приложения является обоснованным и целесообразным.

Приложение будет включать следующие варианты использования:

- ввод личных данных;
- просмотр личных данных;
- редактирование личных данных;
- просмотр главного экрана;

- просмотр каталога;
- просмотр подробной информации товара;
- переход к калькулятору с экрана товара с выбранным способом применения;

- просмотр экрана калькулятора;
- выбор типа расчёта в калькуляторе;
- проведение расчётов в калькуляторе;
- отправка результатов расчётов на электронную почту;

Для определения основных функций, которые должны быть реализованы, на основе вариантов использования строится диаграмма вариантов использования. Готовая диаграмма представлена на рисунке 2.

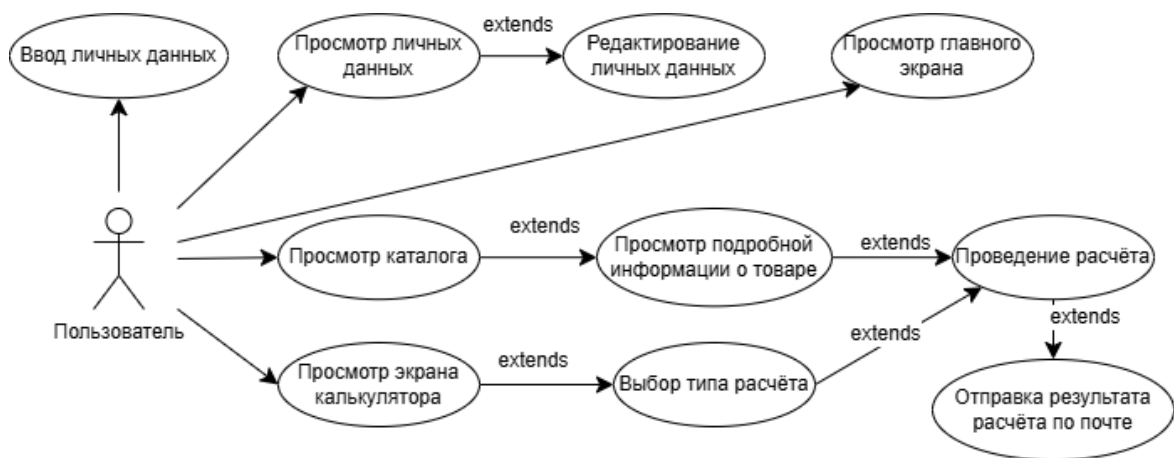


Рисунок 2 – Диаграмма вариантов использования разрабатываемого приложения

### 1.3 Техническое задание на разработку ПО

#### 1.3.1 Наименование программы

Мобильное приложение «binevir».

### **1.3.2 Основание для разработки**

Основанием для разработки является задание на выпускную квалификационную работу.

### **1.3.3 Назначение**

Результатом работы будет готовое кроссплатформенное мобильное приложение магазина-поставщика композитных материалов. Приложение будет предоставлять информацию о производимых товарах, а также иметь функциональный строительный калькулятор для расчета необходимого товара при заданных параметрах.

### **1.3.4 Требования**

В ходе выполнения аналитического раздела, изучения технического задания и просмотра предоставленных прототипов и дизайнов были сформулированы основные требования к разрабатываемому приложению, подробно описанные в следующих подразделах

#### **1.3.4.1 Функциональные требования**

Были сформулированы следующие функциональные требования:

- лого и запуск: значок приложения должен быть фирменной буквой «В», экран загрузки приложения должен содержать брендовый логотип «binevir»;
- сплеш экран: должен присутствовать экран с логотипом компании;
- онбординг: при первом входе в приложение пользователя должен встретить приветственный экран, который знакомит его с возможностями приложения и особенностями компании;

- главный экран: должен быть «первым» при полном запуске приложения, на котором пользователь должны иметь возможность просмотреть различное наполнение, такое как последние расчеты, новости компании, новые возможности, заказы пользователя и т.д.
- каталог: пользователь должен иметь возможность просмотреть каталог предоставляемой продукции;
- подробная информация о товаре: пользователь должен иметь возможность ознакомиться с подробной информацией о товаре: трехмерная модель продукта, продажа листами или рулонами, покрытие песком, описание товара, сферы применения, проекты, в которых использовался товар;
- калькулятор: пользователь должен иметь возможность выбрать необходимое применение в строительстве, ввести параметры и увидеть результат в соответствующей системе измерений: В международной системе единиц (SI) или в американской системе мер (US customary units – USCU/USA), а также отправить результат измерений по почте.
- профиль: пользователь должен иметь возможность заполнить информацию о себе: страна, компания, должность, телефон, электронная почта. Также должна быть возможность изменить личную информацию.
- информативный подсказка: пользователь должен иметь возможность просмотреть контакты компании и её представителей.
- персональная настройка: пользователь должен иметь возможность изменить тему приложения с «светлой» на «тёмную» и наоборот, используемый язык приложения.

#### **1.3.4.2 Требования по надежности**

Приложение должно регулярно испытываться на устойчивость к

нагрузкам. При нахождении критических ошибок незамедлительно начинать их исправление. Плановые работы и обновления проводить во время минимальной активности пользователей.

Приложение должно сохранять устойчивую работу при нестабильном или отсутствующем интернет-соединении. При соответствующих проблемах приложение не должно аварийно завершать работу, а уведомлять пользователя о состоянии и предпринимать попытки повторного подключения.

#### **1.3.4.3 Требования к условиям эксплуатации**

Приложение требует подключение к интернету независимо от источника (Wi-Fi или мобильный интернет).

Устройство должно быть не перегруженным другими процессами, настройки питания не должны жестко ограничивать работу приложения.

Приложение должно регулярно обновляться.

#### **1.3.4.4 Требования к составу технических средств**

Минимальные системные требования включает в себя наличие оперативной памяти не менее свободного в количестве 1 ГБ, а также достаточного количества свободного места на диске для установки и последующего обновления и кеширования часто используемых данных.

Для корректной работы операционные системы устройства должны быть версии 7.0 и выше для Android, 12.0 и выше для iOS.

Серверная часть должна обладать веб-сервером и средствами обработки запросов и взаимодействия с базой данных.

#### **1.3.4.5 Требования по эргономике и технической эстетике**

Интерфейс должен быть интуитивно понятен и прост. Пользователь



должен легко понимать, что является прокручиваемым, когда что-либо изменяется, был ли достигнут лимит.

Длительные операции, такие как обращение к серверной части и отображение данных, должны снабжаться индикатором ожидания.

### **1.3.5 Безопасность при эксплуатации ПЭВМ**

Пользователь приложения должен быть ознакомлен с работой мобильных устройств и инструкцией по их безопасной эксплуатации, не допускать к устройству посторонних лиц, а также избегать использования приложения через небезопасные сети.

### **1.3.6 Этапы разработки**

Были определены следующие этапы разработки:

- анализ предметной области и технического задания;
- проведение исследований по аналитическому разделу;
- разработка по проектно-конструкторскому разделу;
- введение экспериментального раздела
- заполнение заключения и приложений;
- корректировка и редактирование;
- представление выпускной квалификационной работы научному

руководителю

### **Выводы по разделу**

В данном разделе был проведен анализ предметной области мобильного приложения. Выполненная работа позволила обосновать актуальность и востребованность разработки собственного решения.

Современный подход к ведению бизнеса предъявляет высокие требования к качеству и скорости проведения услуг, из-за чего простые и полезные мобильные приложения служат эффективным инструментом для автоматизации бизнес-процессов

Так же была оценена целесообразность разработки с экономической точки зрения из-за возможной интеграции с существующей инфраструктурой и потребностью в особых функциях, таких как строительный калькулятор.

В ходе анализа были сформулированы функциональные и не функциональные требования, следственно и техническое задание

Данный этап важен в определении общей концепции проекта и является основой для последующих этапов разработки.

## **2 Проектно-конструкторский раздел**

### **2.1 Выбор средств разработки**

Выбор средств разработки является важным этапом в процессе создания программного продукта. От корректности принятого решения зависит эффективность создания, масштабируемость, удобство обслуживания и развития.

#### **2.1.1 Выбор подхода к разработке**

При разработке мобильного приложения обязательно поднимается вопрос о выборе подхода к созданию ПО: нативная или кроссплатформенная разработка.

Нативная разработка предполагает использование инструментов специфичной для неё. Для iOS это Xcode [2] на языках Swift [3] или Objective-C [4], Android Studio [5] на Kotlin [6] или Java [7] соответственно. Такой подход даёт максимальную интеграцию с платформой, в следствие чего конечный продукт будет иметь наивысшую производительность и точно соответствовать поставленным требованиям. Однако в таком случае значительно увеличится объем связанных работ, ведь возникнет необходимость создания и поддержки приложения отдельно как на Android, так и на iOS.

Кроссплатформенная разработка в свою очередь представляет собой создание единого кода, который будет использоваться на любой платформе, что значительно снижает время разработки. Для этого существуют специальные фреймворки, например React Native [8], Xamarin [9] или Flutter [10]. Однако в таком случае может упасть качество конечного продукта: проблемы с отображением интерфейсов, низкая производительность из-за использования «обёрток», ограниченный доступ к нативным API,

несвоевременное обновление под новые версии системы.

На основании сравнений был принят кроссплатформенный подход для реализации мобильного приложения «binevir».

### **2.1.2 Выбор фреймворка**

Выбор фреймворка тоже не маловажное решение. При выборе стоит учитывать производительность, качество документации и размер сообщества разработчиков, функциональные возможности, а также потенциал развития.

Существует множество кроссплатформенных фреймворков для мобильной разработки со своими особенностями. Вот самые популярные на данный момент:

Flutter – принадлежит Google, работает на языке программирования Dart [11]. Особенностью является использование собственного движка для рендера – Skia. Основной компонент разработки – виджет. Разработчики могут разрабатывать и настраивать свои виджеты, однако есть большой набор решений. Заинтересованной компании-родителя, а также растущее сообщество ведёт к наличию значительного количества сторонних библиотек для расширения функционала, а также упрощения разработки. Наличие полной документации, а также инструкции для разработчиков из смежных тем очень привлекает новую аудиторию;

React Native – принадлежит Facebook, работает на языке программирования Javascript [12] и библиотеки для веб-разработки React. Фреймворк предполагает преобразование JavaScript кода в нативные UI-элементы целевой платформы, что дает относительно высокую производительность. Он активно поддерживается сообществом и имеет поддерживаемую документацию;

Ionic [13] – фреймворк типа Hybrid-Web, в котором разработанное приложение работает в специальной оболочке, в UIWebView на iOS, WebView на android. Основывается на веб-технологиях (HTML, CSS,

JavaScript) и использует фреймворк Angular [14]. Доступно множество готовых компонентов, что ускоряет разработку. Однако из минусов замечают низкую интеграцию с нативными функциями устройств (Камера, вибрация и т.д.) из-за выполнения «в браузере»;

Apache Cordova [15] – фреймворк также, как и Ionic, базируется на использовании веб-технологий, однако является гибридным методом разработки. Веб-приложения выполняются в специальном контейнере, сохраняя доступ к функциям устройства. Это достигается из-за использования унифицированного JavaScript API;

Xamarin– фреймворк работает на C# [16] и использует платформы .NET [17]. Состоит из библиотек классов для каждой платформы – Xamarin.iOS и Xamarin.Android, а также соответствующие компиляторы и «свою» среду разработки – «Xamarin Studio». Данный фреймворк может похвастаться высокой производительностью сравнимой лишь с нативными приложениями, высокой совместимостью и полным инструментарием. Однако из-за этого приложения занимают значительно больше места на устройстве, а также могут быть недоступные функции, которые не предоставляются платформой, и могут быть задержки ввода поддержки с обновленными платформами.

Сравнив популярные фреймворки, было решено использовать именно Flutter. Он набирает популярность среди разработчиков, а поддержка компанией-владельцев только усиливается, что делает его наиболее перспективным и надёжным выбором для долгосрочных проектов. О актуальности Flutter можно судить о его росте популярности как инструмента кроссплатформенной мобильной разработки ссылаясь на график «Statista» [18], изображенного на рисунке 3, можно посмотреть процент использований различных фреймворков в последние года.

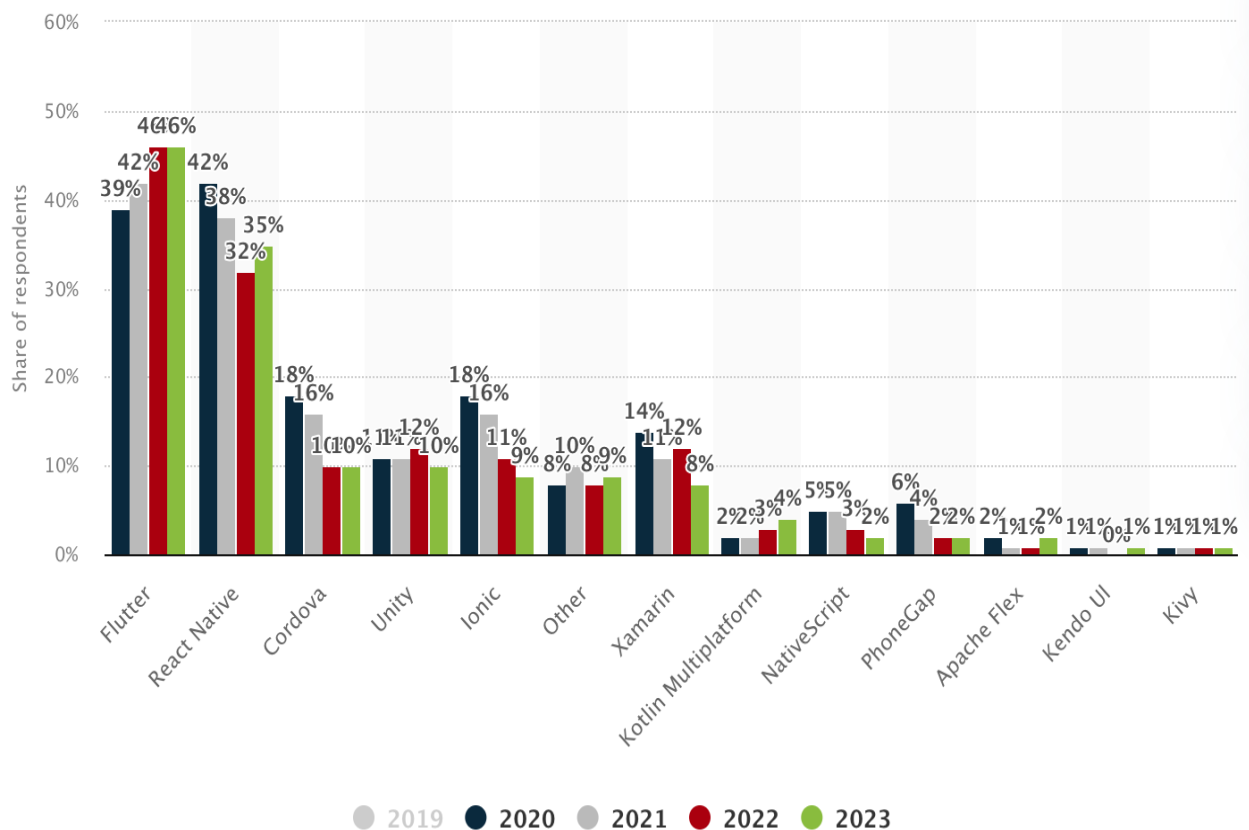


Рисунок 3 – Процент использования различных фреймворков для разработки мобильных приложений

### 2.1.3 Принципы работы фреймворка Flutter

Flutter – фреймворк с открытым исходным кодом, разработанный Google, позволяющий разрабатывать приложения разной сложности для Android, iOS, Windows, macOS, Linux и Web с использованием единого кода.

Архитектура состоит из трех основных слоев, представлена на рисунке 4:

- Dart Framework – фреймворк верхнего уровня, работает на Dart, содержит набор библиотек для создания виджетов – компонентов интерфейса.
- Flutter Engine – движок, средний уровень приложения, работает

на C++, обрабатывает ввод и вывод, обеспечивает рендеринг на базе Skia и обработка событий.

– Embedder – внедритель, нижний уровень, встраиваемый компонент, запускающий приложение на разных устройствах и обрабатывающий взаимодействие с ОС.

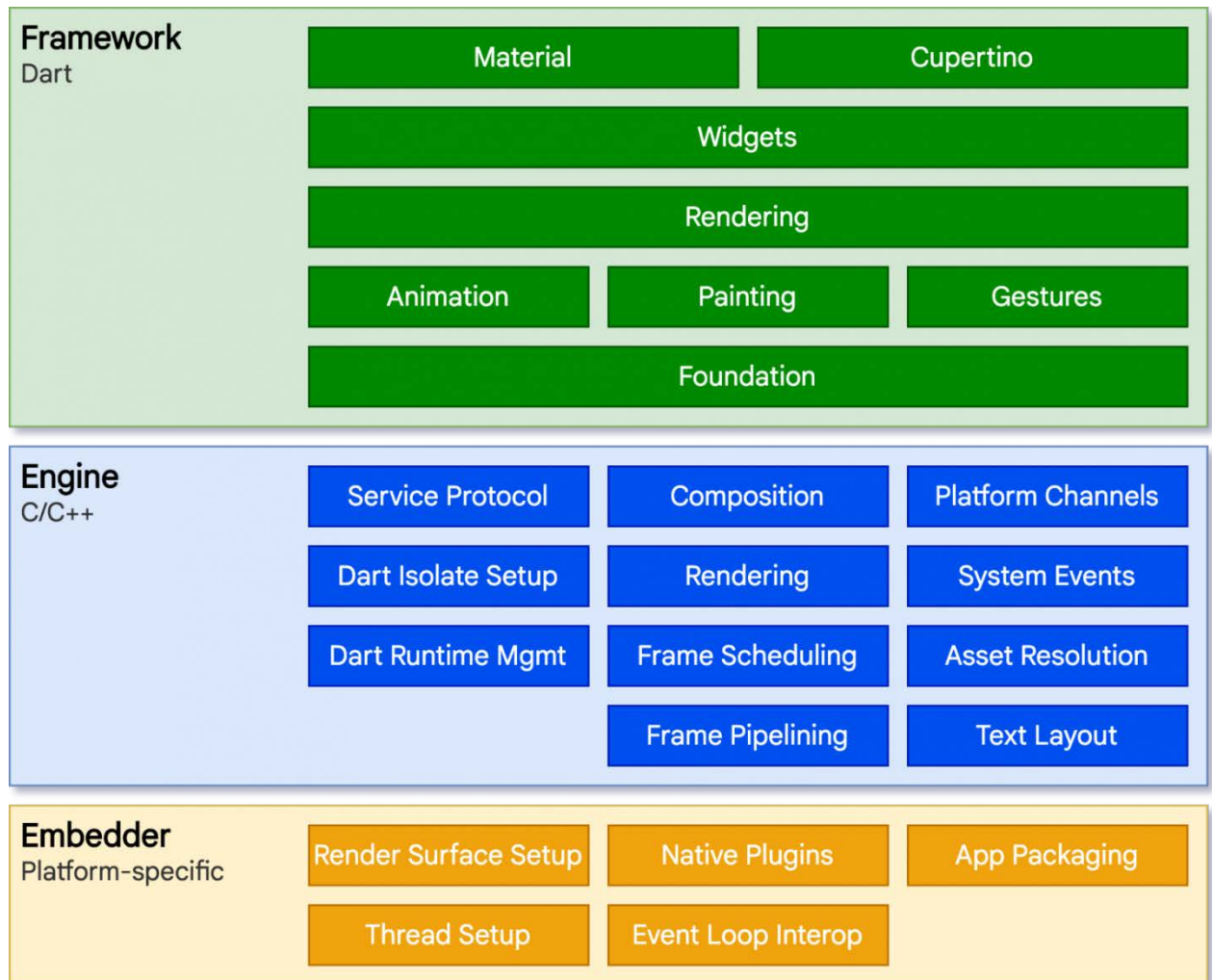


Рисунок 4 – Архитектура фреймворка Flutter

Вдохновленный React [19], Flutter использует реактивные принципы программирования. UI обновляется в ответ на изменения состояния приложения, вместо того, чтобы вручную обновлять элементы, фреймворк сам перестраивает изменившуюся часть, тем самым увеличивая отзывчивость

приложения, а также делать код более предсказуемым.

Виджеты – основные компоненты интерфейса, существуют два вида:

- `stateless widget` – виджет без состояния, не может быть изменен во время выполнения приложения (текст, иконка, и т.д.).
- `stateful widget` – виджет с состоянием, меняющийся в ходе работы приложения (Поля ввода, результаты запросов и т.д.).

Кроме того, виджеты можно комбинировать, вкладывая в друг друга, для управления положением на экране используются Flex-подобные модели, а также предоставлены инструменты для адаптивных интерфейсов. Всё это позволяет Flutter создавать приложения с сложными интерфейсами, но понятно разработанными, а также поддерживать корректное отображение на экранах разных размеров.

Также стоит отметить функцию `Hot Reload`, позволяющую мгновенно отобразить изменения в UI без необходимости перезапуска и полной пересборки проекта, что существенно ускоряет процесс разработки интерфейсов разработчиками. Всё это доступно благодаря JIT-компиляции (`Just-In-Time`), то есть компиляцией во время выполнения. По сравнению с AOT(`Ahead-Of-Time`) будет меньшая скорость запуска, производительность и потребуется использование виртуальной машины, однако функция `Hot Reload` незаменима как функция для быстрой разработки интерфейсов.

#### **2.1.4 Выбор языка программирования**

Так как в качестве фреймворка выбран Flutter выбор языка программирования ограничен лишь одним вариантом – Dart. Сам он разработан ещё в 2011-ом году, ориентирован на разработку быстрых приложений. Ключевые особенности Dart:



- Простой синтаксис – легок в освоении, напоминает популярные C-подобные языки, такие как JavaScript, C# или Java, что облегчает переход разработчиков на него
- ООП – поддержка классов, наследования, интерфейсов и абстракций, обеспечивая модульность и понятное переиспользование кода.
- Асинхронное программирование – используются специальные ключевые слова (`async`, `await`) и объекты (`Future`) для работы с длительными задачами.
- JIT (Just-In-Time) и AOT (Ahead-Of-Time) компиляции – ускорение разработки с использованием функции `Hot Reload` в режиме JIT, построение финальной сборки с преобразованием кода Dart в нативный конкретный платформы в режиме AOT.
- Типы и Null Safety – Dart поддерживает как строгую, так и динамическую типизацию, что повышает приложения и снижает вероятность ошибок, не ограничивая разработчика в гибкости. С новейшие версии Dart поддерживают Null Safety, позволяющая явно указывать какие значения могут принимать `null`, снижая количество ошибок и исключения, приводящих к аварийным завершениям.
- Сборка мусора – язык включает в себя и автоматическую отчистку памяти
- Platform channels – платформенные каналы, механизм позволяющий Dart-коду работать с нативным кодом Android и iOS, недоступные во Flutter. Схема работы каналов представлена на рисунке 5.

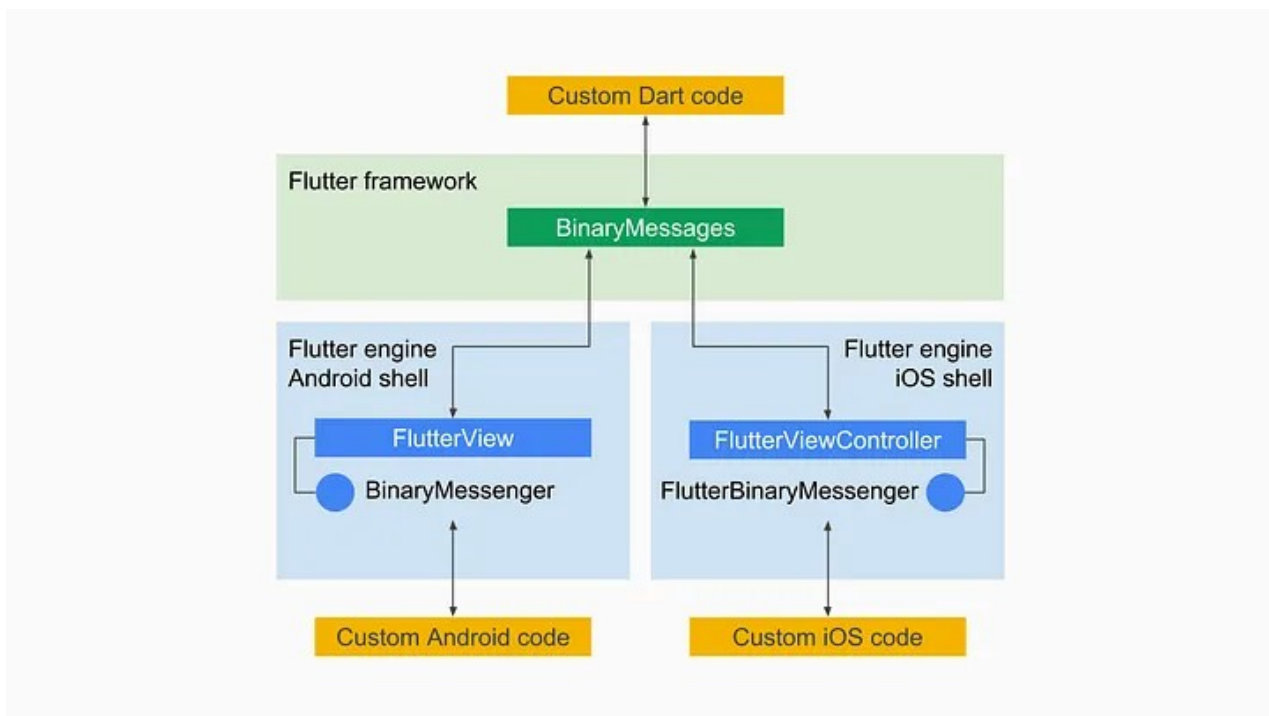


Рисунок 5 – Схема работы Platform Channels

### 2.1.5 Библиотеки

В приложениях на базе Flutter почти всегда используются дополнительные библиотеки, разработанные Google или другими независимыми программистами. Их использование позволяет расширять функционал без лишних затрат и для ускорения разработки.

Можно выделить основные используемые библиотеки:

- `go_router` – переопределяет и дополняет маршрутизацию во Flutter, позволяет легко настраивать навигацию;
- `hive_flutter` – легковесная и быстрая NoSQL база данных, не требует сложной настройки, хранит данные в разных форматах. Используется для кеширования данных и сохранения настроек;
- `flutter_bloc` – реализация паттерна BLoC (Business Logic Component) для управления состоянием приложения. Повышает читаемость и

надежность кода;

- `l10n` (Localization) – система локализации Flutter, позволяющая поддерживать множество переводов приложения;

- `dio` – мощный пакет для работы с HTTP запросами и взаимодействия с веб-серверами, содержит инструменты для сетевых операций.

Другие библиотеки и их назначение:

- `cupertino_icons` – стилизованные иконки под iOS
- `custom_navigation_bar` – настраиваемый компонент для навигации;

- `flutter_native_splash` – упрощенное создание нативного экрана загрузки;

- `get_it` – сервис для управления зависимости;

- `cached_network_image` – загрузка и автоматическое кеширование изображений из сети;

- `internet_connection_checker` – проверка доступности интернета;

- `image_picker` – выбор изображений с устройства пользователя;

- `dropdown_button2` – виджет выпадающего списка;

- `flutter_svg` – работа с SVG изображениями;

- `url_launcher` – открытие ссылок и приложений по умолчанию;

- `hive_generator` – генератор моделей для Hive;

- `model_viewer_plus` – отображение 3D моделей;

- `webview_flutter` – встраивание веб-контента;

- `path` – работа с файловыми путями;

- `keyboard_actions` – управление клавиатурой;

### **2.1.6 Выбор среды разработки**

Существует несколько сред разработки:

- Android Studio – официальная среда разработки под Android от Google. Имеет мощный инструментарий по разработке, тестированию и отладке мобильных приложений. Поддержка Flutter и Dart из коробки, однако требует много системных ресурсов, сложна в использовании для новичков, многих функционал избыточен.

- IntelliJ IDEA [20] – среда разработки от JetBrains, поддерживает Flutter и Dart посредством установки плагинов. Среди минусов также высокая нагрузка системы, а также платная версия с расширенными функциями.

- Visual Studio Code [21] – мощный редактор кода от Microsoft. Наиболее популярны и поддерживает множество языков программирования. Из плюсов можно выделить легкость программы, простую установку дополнительных плагинов, полностью бесплатна, поддерживает контроль версий и простую интеграцию с GitHub [22]. Однако может требовать дополнительных настроек.

В результате сравнения выбор пал на использование VSC (Visual Studio Code), поскольку она является наилучшим вариантом по отношению к производительности, а все недостатки компенсируются возможностью простой модернизации и расширения функционала с помощью расширений

### **2.1.7 Выбор среды тестирования**

При разработке мобильных приложений, в том числе на Flutter, используются различные эмуляторы. Они предоставляются Android Studio, позволяя настраивать различные виртуальные девайсы. Visual Studio Code

имеет возможность интегрироваться с ними, тем самым позволяет запускать и тестировать приложение с компьютера. Интерфейс менеджера устройств Android Studio представлен на рисунке 6.

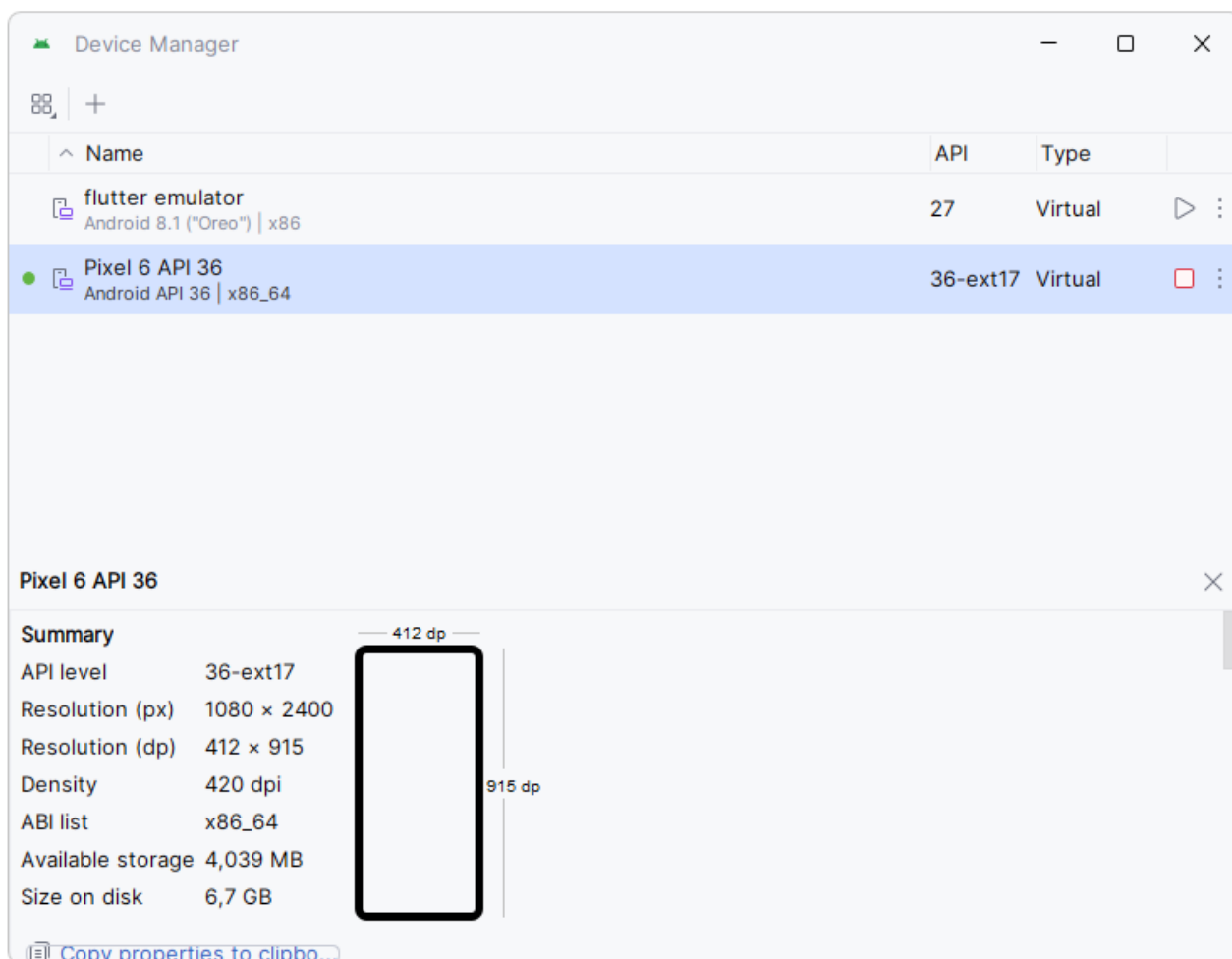


Рисунок 6 – Характеристики выбранного эмулятора

## 2.2 Проектирование архитектуры

### 2.2.1 Архитектура серверной части

Серверная часть построена на Docker [25] – контейнерное решение. Использование таких технологий обеспечивает масштабируемость проекта, а также делает его развёртывание удобнее и быстрее.

Основные компоненты серверной части:

- API-сервис – контейнер на базе PHP, реализующий логику приложения и API, к которому обращается уже созданный сайт-визитка и разрабатываемое мобильное приложение. Сервис содержит в себе PHP расширения и зависимости для исполнения необходимого функционала;
- База данных MariaDB – реляционная база данных, в которой хранятся все основные данные проекта;
- Веб-сервер Nginx [23]– сервер на базе Alpine Linux для принятия через себя HTTP-запросов и перенаправления их к PHP-приложению;
- Redis – обеспечивает кеширование и работу с очередями, что повышает производительность сервера.

### **2.2.2 Архитектура приложения**

Архитектура Flutter-приложения представляет собой многоуровневую структуру, обеспечивающую взаимодействие между кодом и системными компонентами. Ключевые уровни:

- Приложение Dart – реализация интерфейса и логики;
- Фреймворк Flutter – предоставление готовых компонентов (Виджеты, обработка жестов);
- Движок – отрисовка интерфейса и работа с API;
- Встраиваемый слой (Embedder) – связывает движок с ОС;
- Исполнитель (Runner) – запускает приложения, собирая все слои в адаптированном виде под целевую платформу.

Схема Flutter приложения представлена на рисунке 7.

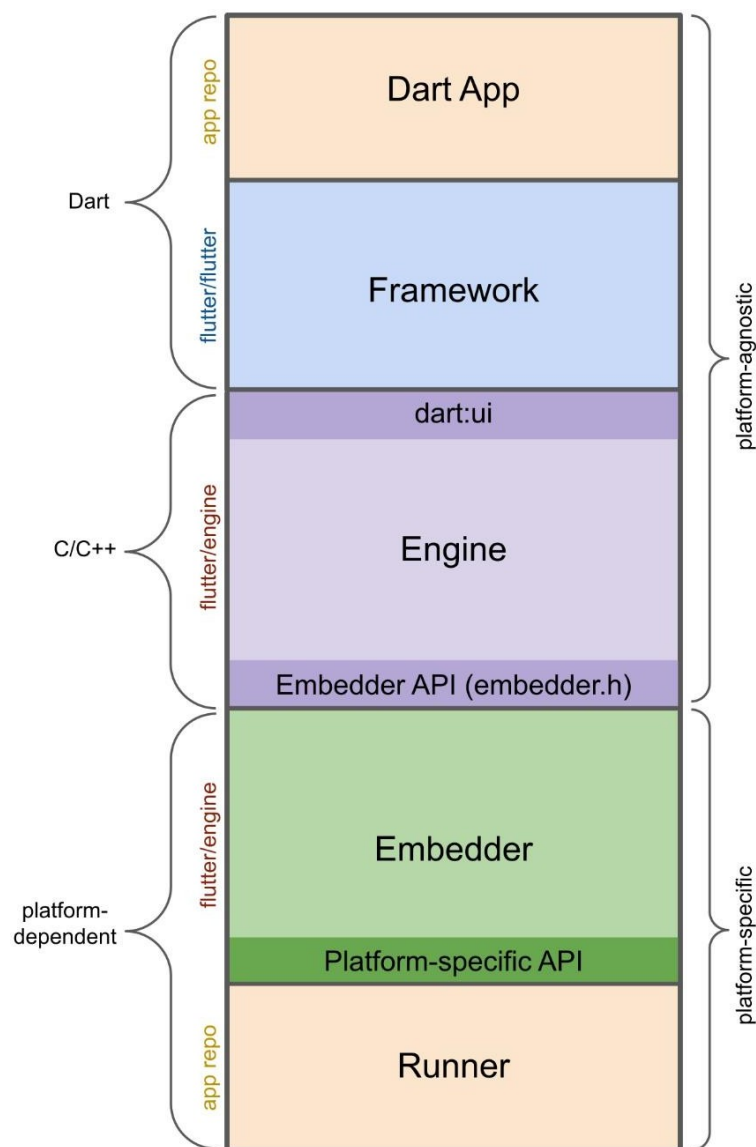


Рисунок 7 – Схема элементов Flutter-приложения

Рассматривая архитектуру приложения, стоит описать управление состояниями. Для упрощения используется шаблон BLoC. Он является одним из наиболее популярных подходов к управлению, так как просто помогает отделить бизнес-логику, улучшая масштабируемость кода.

Концепции BLoC:

- **Events** – события, действия которые отправляются в BLoC (Нажатие кнопки, жест и т.д.);

- States – состояния, отображаемые данные на экране в зависимости от события
- BLoC – компонент, который получает события и выполняет определенную логику.

Схема работы BLoC представлена рисунке 8.

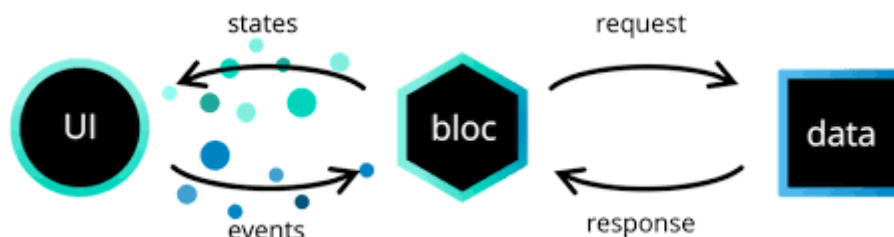


Рисунок 8 – Схема работы BLoC

### 2.3 Проектирование пользовательского интерфейса

Проектирование пользовательского интерфейса (UI) – важнейший этап в разработке любого приложения, от которого напрямую зависит удобство пользователя. Грамотный дизайн не только будет привлекательным, но ещё позволит упростить взаимодействие с приложением, увеличив его интуитивность, а вследствие повысит пользовательский опыт.

В рамках этой работы для проектирования интерфейсов выбран инструмент Figma – онлайн-сервис для создания макетов, компонентов для UI, а также прототипов приложений и программ. В ходе проектирования пользовательского интерфейса были подготовлены:

- UI-кит – наборы типовых интерфейсных компонентов, используемые цвета, шрифты;
- Макеты экранов – визуальные схемы всех ключевых интерфейсов приложения;
- Прототип – интерактивная модель, позволяющая оценить



сценарии взаимодействия для корректной программной реализации.

Примеры UI-китов, макетов экрана и отображение прототипа изображены на рисунках 9, 10, 11 соответственно.

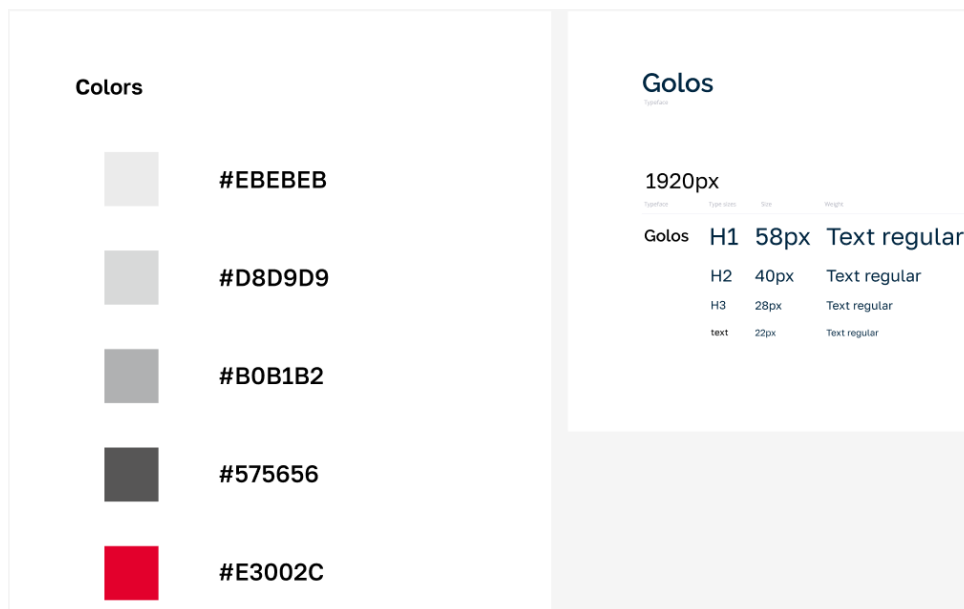


Рисунок 9 – Пример UI-кита по требуемым цветам и шрифту

contacts\_1

Photo

First name: Ivan

Second name:

Surname:

Country: Turkey

Company: SBP group

Job title:

Phone:

E-mail: i@sbpgroup.com

CONTINUE

skip

Рисунок 10 – Пример макета экрана с заполнением данных

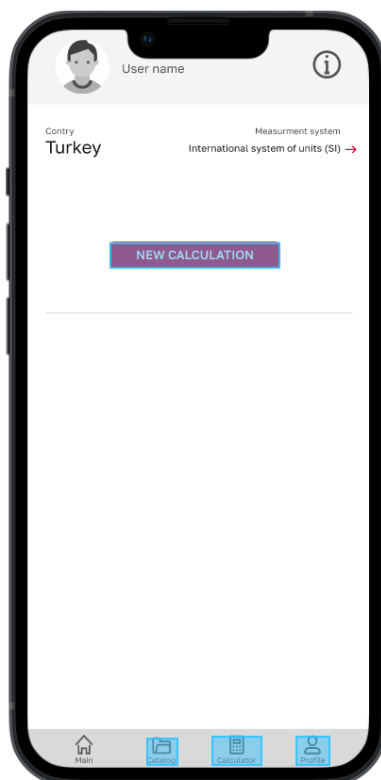


Рисунок 11 – Прототип приложения с элементами, на которые можно нажать

## 2.4 Разработка мобильного приложения

Этап разработки мобильного приложения – центральная часть проекта, на котором происходит реализация всей запланированной функциональности с использованием выбранных инструментов и средств.

Важно уделить особое внимание на соблюдение определенных ранее требований, так как качество их реализации напрямую влияет на пригодность использования законченного продукта.

### 2.4.1 Устройство структуры Flutter-проекта

Структура проекта играет ключевую роль в обеспечении читаемости, удобства сопровождения и масштабируемости приложения. Правильная организация структуры упрощает ориентирование в коде, что упрощает разработку.

При создании проекта автоматически генерируется базовая структура:

- `.dart_tool/` – внутренняя папка для Dart и Flutter, хранит метаданные о зависимостях, кеш и информация о сборках;
- `.idea/`, `.vscode/` – папки для хранения настроек проекта внутри соответствующей среды разработки;
- `build/` – генерация при сборке проекта;
- `.flutter-plugins`, `.flutter-plugins-dependencies` - автоматически файлы подключенных файлов и их зависимости;
- `.metadata` – метаданные о версии Flutter;
- `android/`, `ios/`, `web/`, `windows/`, `macos/`, `linux/` – папки для поддержки сборки приложения на соответствующих платформах;
- `pubspec.yaml` – файл конфигурации проекта, в котором указываются зависимости, ассеты и настройки проекта;

- pubspec.lock – зафиксированные версии зависимостей;
- test/ - модульные тесты приложения
- lib/main.dart – точка входа в приложение;
- .gitignore – файл исключений системы контроля версий;
- analysis\_options.yaml – настройки анализа кода Dart;
- devtools\_options.yaml – конфигурации для инструментов разработчика;

Основная логика и организация создается разработчиком вручную. Структура папки lib/:

- binevir\_app.dart – корневой виджет приложения, запускаемый из main.dart. Содержит базовые настройки темы и лока, делегированием функций маршрутизации.
- components/ – содержит в себе повторно используемые виджеты;
- constants/ – содержит константы путей, используемых цветов и т.д.;
- data/ – содержит обрабатывает данные, имеет подпапки: models/ – хранит модели данных, http/ – сервис API запросов, построенный на dio, repository/ – слой между BLoC и источниками данных;
- di/ – внедрение зависимостей через get\_it, регистрирует доступ к репозиториям как к синглтон объекту;
- helpers/ – содержит вспомогательные функции, такие как работу с файлами устройства, перевода из одной системы измерений в другое, математические функции и форматирование пользовательского ввода;
- l10n/ – локализация приложения на разные языки на основе arb-формата;

- router/ – роутер маршрутов;
- screens/ – содержит экраны приложения, каждый из которых расположен в отдельной папке. Папка экранов содержит dart-файл экрана, а также подпапки в зависимости от логики и сложности: bloc – содержит BLoC обработку логики, widgets – содержит в себе повторяющиеся или громоздкие виджеты на экране;

Вне папки lib/ добавлены следующие каталоги:

- assets/ – содержит иконки и изображения;
- fonts/ – содержит пользовательские шрифты.

## **2.4.2 Построение дерева экранов**

Построение дерева экранов – частая практика при разработки многостраничных приложений. Эта концепция позволяет логично структурировать логику переходов и в последующем упростить реализацию маршрутизации. В большинстве случаев дерево можно строить на основе Use cases – пользовательских сценариев, так как каждый из них может отражать логический путь в приложении.

Открыв приложение пользователя, встречает стартовый экран – сплэш, отображающий лого компании. Пока пользователю отображается сплэш-экран, приложение может загрузить последующие необходимые для работы данные в фоновом режиме.

Далее, если пользователь не ввел свои данные (Или запустил приложение впервые), то ему отобразится экран с последовательностью обучающих слайдов – онбординг экран, в конце которого ему будет предоставлена возможность заполнить информацию о себе. На каждом этапе у пользователя есть возможность пропустить следующие слайды и заполнение информации о себе. В любом из случаев он будет перемещен на

главный экран.

Главный экран пока не имеет конкретного функционала, на нём могут отображаться новости компании, последние расчёты и прочее. Весь функционал может быть легко добавлен по требованию заказчика. Здесь впервые пользователь увидит навигационный виджет, с помощью которого сможет перемещаться по другим экранам самостоятельно с выбором: главная, каталог, калькулятор, профиль.

Каталог позволяет просмотреть продукцию организации и отобразить их в зависимости от предпочтений пользователя – списком или плитками. Кроме того, пользователь может перейти экрану подробной информации о товаре, выбрав его в каталоге.

Экран товара содержит подробную информацию о нём: его возможные характеристики, описание, сферы применения, в каких проектах использовался.

Калькулятор – главная особенность приложения, позволяет пользователю рассчитать необходимое количество товара по заданным параметрам в зависимости в выбранной сферы применения, с отображением результата и возможностью сохранить его, отправив на почту.

Профиль отображает личную информацию о пользователе, которую он мог заполнить на экране онбординга, контакты менеджера, а также имеет возможность отредактировать их и настроить приложение.

Построенное дерево экранов можно увидеть на рисунке 12.

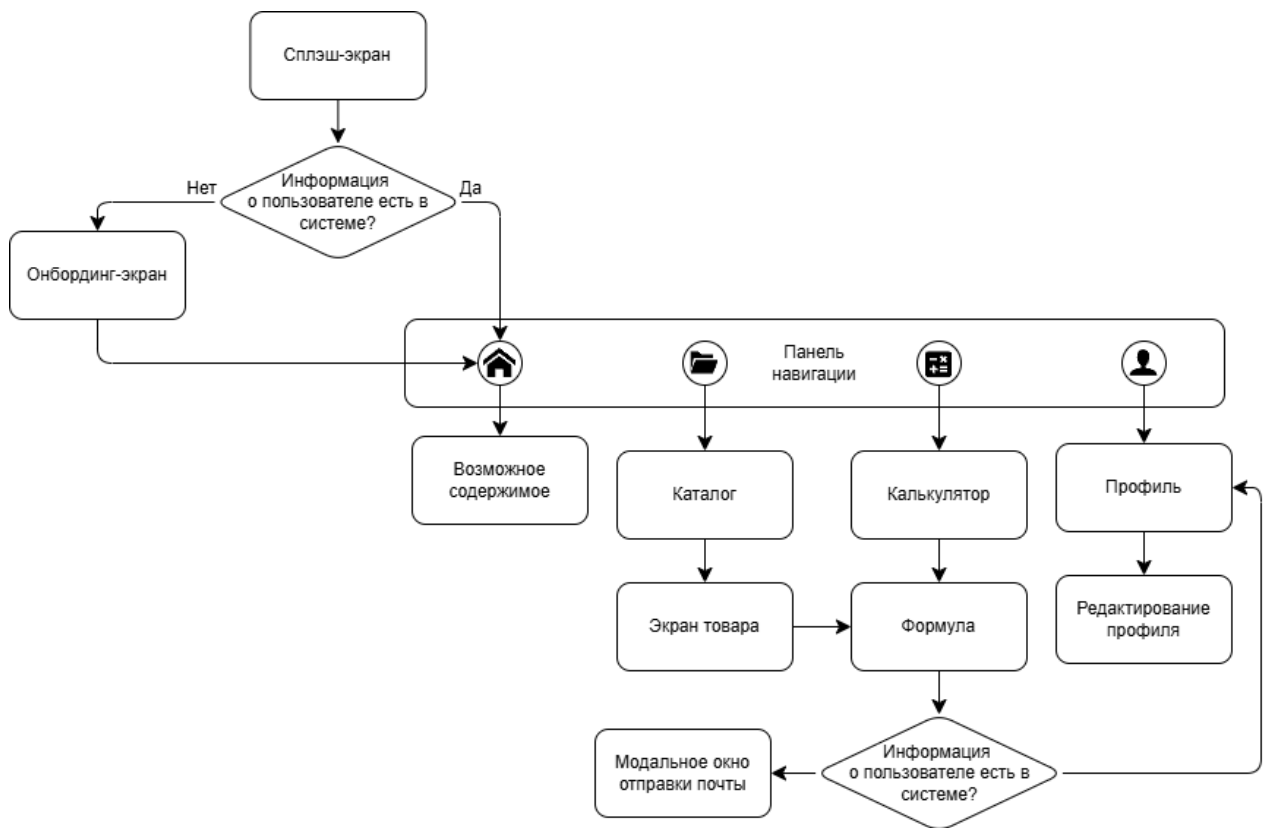


Рисунок 12 – Дерево экранов приложения

Для простой реализации маршрутизации и управления перехода используется библиотека `go_router`, вся логика которого размещена в `lib/router/router.dart` проекта. На рисунке 13 можно увидеть его реализацию.

```

final GlobalKey<NavigatorState> _shellNavigatorKey = GlobalKey<NavigatorState>({
  debugLabel: 'shell',
});

final GoRouter appRouter = GoRouter(
  initialLocation: '/splash',
  errorBuilder: (context, state) => ErrorScreen(),
  routes: <RouteBase>[
    ShellRoute( // ShellRoute ...

      GoRoute(
        path: '/welcome',
        builder: (context, state) => const OnboardingScreen(),
      ), // GoRoute
      GoRoute(
        path: '/splash',
        builder: (context, state) => const SplashScreen() // GoRoute
      ), // <RouteBase>[]
    ], // <RouteBase>[]
); // GoRouter

```

Рисунок 13 – Реализация роутера приложения

В реализации роутера определены следующие настройки:

- начальный маршрут – /slash;
- экран ошибки – ErrorScreen();
- список маршрутов – routes;
- обертку некоторых маршрутов – ShellRoute;

Для обеспечения навигации пользователем используется компонент ScaffoldWithNavBar, так как данный виджет требуется размещать на нескольких экранах, поэтому принято решение обернуть некоторые пути в ShellRoute с его указанием в билдере для отображения одного и того же элемента на разных страницах. Такой подход не только избавит проект от повторяющегося кода, но улучшит графическую составляющую и производительность, ведь Flutter не придется строить новый виджет при переходе. Отображение и фрагмент кода реализации компонента навигации,



а также структуру `ShellRoute` можно увидеть на рисунках 14, 15, 16 соответственно.



Рисунок 14 – Компонент навигации приложения

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Stack(
      children: [ ...
    ), // Stack
    bottomNavigationBar: Container(
      color: const Color.fromRGBO(216, 217, 217, 1),
      padding: const EdgeInsets.only(left: 20, right: 20),
      height: 65,
      child: CustomNavigationBar(
        iconSize: 20.0,
        elevation: 0,
        scaleFactor: 0.2,
        opacity: 1,
        selectedColor: const Color.fromRGBO(227, 0, 44, 1),
        strokeColor: const Color(0x30040307),
        unSelectedColor: const Color.fromRGBO(87, 86, 86, 1),
        backgroundColor: Colors.transparent,
        onTap: (int idx) => _onItemTapped(idx, context),
        currentIndex: _calculateSelectedIndex(context),
        items: [
          CustomNavigationBarItem(
            icon: const Icon(AppIcons.home),
            title: _navLabel(AppLocalizations.of(context)!.main),
          ), // CustomNavigationBarItem
          CustomNavigationBarItem( // CustomNavigationBarItem ...
          CustomNavigationBarItem( // CustomNavigationBarItem ...
          CustomNavigationBarItem( // CustomNavigationBarItem ...
        ],
      ), // CustomNavigationBar
    ), // Container
  ); // Scaffold
}
```

Рисунок 15 – Часть кода реализации компонента навигации

```

routes: <RouteBase>[
  ShellRoute(
    navigatorKey: _shellNavigatorKey,
    builder: (context, state, child) {
      return ScaffoldWithNavBar(child: child);
    },
    routes: [
      GoRoute( // GoRoute ...
      GoRoute( // GoRoute ...
      GoRoute( // GoRoute ...
      GoRoute( // GoRoute ...
    ],
  ), // ShellRoute

```

Рисунок 16 – обёртка маршрутов в ShellRoute

Также go\_router имеет понятную механику передачи параметров и может поддерживать «ветвистость» в URL. Реализация передачи параметров и веток изображены на рисунке 17.

```

      return ScaffoldWithNavBar(child: child);
    },
    routes: [
      GoRoute( // GoRoute ...
      GoRoute(
        path: '/catalog',
        builder: (context, state) => const CatalogScreen(),
        routes: [
          GoRoute(
            path: ':id',
            builder: (context, state){
              final idParam = state.pathParameters['id'];
              final id = int.tryParse(idParam ?? '');
              if(id == null) return const ErrorScreen();
              return ProductScreen(id: id);
            },
          ), // GoRoute
        ],
      ), // GoRoute
      GoRoute( // GoRoute ...
      GoRoute(
        path: '/profile',
        builder: (context, state) => const ProfileScreen(),
        routes: [
          GoRoute(
            path: 'edit',
            builder: (context, state) => const ProfileScreenEdit(),
          ), // GoRoute
        ],
      ), // GoRoute
    ],
  ),

```

Рисунок 17 – особые маршруты в обёртке

Так открытие подробной информации о товаре будет находиться по относительному адресу например – /catalog/1, а редактирование профиля – /profile/edit. .

### 2.4.3 Разработка пользовательского интерфейса

При разработке часто приходилось ссылаться на макеты в Figma, для визуального соответствия разработанному дизайну. Figma также предоставляет возможности для быстрого переноса свойств реализуемых компонентов. Так можно посмотреть соотношения элемента, узнать точный цвет любой части и просто экспортировать какой-то элемент в выбранном и удобном формате, например лого или иконку. На рисунке 18 изображен просмотр свойств элемента макета в Figma.

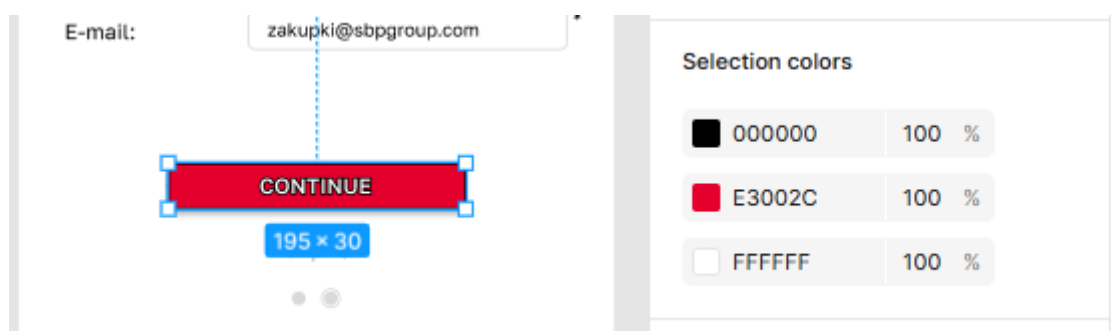


Рисунок 18 – Просмотр свойств элемента в Figma

Для работы консистентного и логически правильного отображения создаются константные значения в папке constant/, так разработчику не придется вручную жёстко задавать каждому виджету своё свойство. Определение констант стандартных цветов и фрагмент кода светлой темы представлены на рисунках 19, 20 соответственно.

```
import 'package:flutter/material.dart';

Color hexToColor(String hex) {
  assert(RegExp(r'^#([0-9a-fA-F]{6})|([0-9a-fA-F]{8})$').hasMatch(hex) ||
    'hex color must be #rrggbb or #rrggbbbaa');

  return Color(
    int.parse(hex.substring(1), radix: 16) +
    (hex.length == 7 ? 0xff000000 : 0x00000000),
  );
}

class ThemeColors {
  static Color white = const Color.fromRGBO(255, 255, 255, 1);
  static Color black = const Color.fromRGBO(0, 0, 0, 1);
  static Color red = const Color.fromRGBO(227, 0, 44, 1);
  static Color primaryColor = hexToColor('#C4C4C4');
}
```

Рисунок 19 – Файл theme\_color, содержащий константные значения цветов и метод перевода из шестнадцатеричной системы кодирования цветов в RGB

```
class AppTheme {
  static ThemeData get lightTheme {
    return ThemeData(
      scaffoldBackgroundColor: ThemeColors.white,
      fontFamily: 'Golos',
      primaryColor: ThemeColors.red,
      primarySwatch: Colors.red,
      hintColor: ThemeColors.red,
      colorScheme: ColorScheme.fromSwatch(
        primarySwatch: Colors.red,
      ).copyWith( // ColorScheme.fromSwatch
        secondary: ThemeColors.red,
      ),
      elevatedButtonTheme: ElevatedButtonThemeData( // ElevatedButtonThe
      bottomNavigationBarTheme: BottomNavigationBarThemeData(
        elevation: 0,
        showUnselectedLabels: true,
        backgroundColor: const Color.fromRGBO(216, 217, 217, 1),
        unselectedItemColor: const Color.fromRGBO(87, 86, 86, 1),
        selectedItemColor: ThemeColors.red,
        unselectedIconTheme: const IconThemeData(),
        unselectedLabelStyle: const TextStyle(
          fontSize: 10.8,
        ), // TextStyle
        selectedLabelStyle: const TextStyle(
          fontSize: 10.8,
        ), // TextStyle
      ), // BottomNavigationBarThemeData
      buttonTheme: ButtonThemeData(
```

Рисунок 20 – Файл theme\_date с «гет» методом получения цветов темы

Такой подход способствует снижению количества ошибок, а также упрощает переработку дизайна, если у компании произойдет ребрендинг. Кроме того, как и в большинстве мобильных приложений, подобным способом можно легко организовать смену тем, между тёмной и светлой.

Как уже говорилось, Flutter полностью состоит из виджетов, вёрстка же состоит из правильного их комбинирования и вкладывания друг в друга. Flutter использует Flex-подобную модель вёрстки, подобную CSS веб-приложениях, что дает спроектировать самый сложный интерфейс с поддержкой адаптивности под разные размеры экранов, что куда важнее в мобильной разработке приложений. Отображение готового результата вёрстки экрана «Каталог» и часть его кода представлено на рисунках 21 и 22.

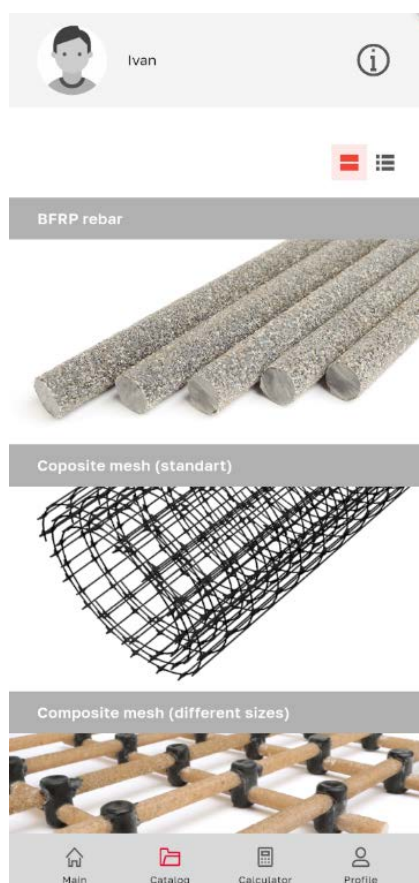


Рисунок 21 – Разработанный экран каталога

```

return Screen(
  body: Column(
    children: [
      Container(
        padding: const EdgeInsets.symmetric(horizontal: 25),
        margin: const EdgeInsets.symmetric(vertical: 10.0),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.end,
          children: [
            ToggleButtons( // ToggleButtons ...
          ],
        ), // Row
      ), // Container
      Expanded(
        child: RefreshIndicator(
          onRefresh: () async {
            final completer = Completer();
            _catalogBloc.add(LoadCatalog(completer: completer));
            return completer.future;
          },
          child: BlocBuilder<CatalogBloc, CatalogState>(
            bloc: _catalogBloc,
            builder: (context, state) {
              if (state is CatalogLoaded) {
                return ListView.builder(
                  itemCount: state.products.length,
                  itemBuilder: (BuildContext context, int index)
                {
                  return InkWell(
                    onTap: () { ...
                      child: CatalogCard( // CatalogCard ...
                    ); // InkWell
                }, // ListView.builder
              } else if (state is CatalogLoadingFailure) {
                return LoadingFailureWidget(
                  onRetry: () { ...
                ); // LoadingFailureWidget
              }
              return platformIndicator();
            },
          ), // BlocBuilder
        ), // RefreshIndicator
      ),
    ],
  ),
);

```

Рисунок 22 – Фрагмент кода экрана каталога

#### 2.4.4 Взаимодействие с сервером

Взаимодействие с сервером позволяет приложению обмениваться с ним данными. Большинство данных поступают извне, для поддержки актуальности информации, независимо от версии ПО. В данном случае интеграция с бэкендом происходит через API, путём отправки HTTP запросов. Для этого в приложении настроен клиент Dio для работы с http, реализована система репозитория с внедрением зависимостей GetIt и простым кешированием с использованием Nive.

К API нет конкретной документации, однако есть файл routes/api.php с сервера, содержащий в себе доступные эндпоинты. Его содержание изображено на рисунке 23.

```

<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
    |     return $request->user();
});

Route::get('/settings', [\App\Http\Api\Controllers\SettingsController::class, 'index']);

Route::get('/countries', [\App\Http\Api\Controllers\CountryController::class, 'index']);

Route::get('/products', [\App\Http\Api\Controllers\AppProductController::class, 'index']);

Route::get('/products/{id}', [\App\Http\Api\Controllers\AppProductController::class, 'getProduct']);

Route::get('/guides', [\App\Http\Api\Controllers\AppGuideController::class, 'index']);

Route::get('/applications', [\App\Http\Api\Controllers\AppApplicationController::class, 'index']);

```

Рисунок 23 – Доступные API эндпоинты бэкенда

Настройка Dio позволяет лучше отлаживать систему и позволяет легче масштабировать функционал. Для этого в каталоге data/http/ используются следующие файлы:

- dio\_client.dart - непосредственно сам сервис, с реализацией всех запросов с аргументами, получаемыми из других файлов. Фрагмент изображен на рисунке 24;
- endpoints.dart – содержит непосредственные эндпоинты к API. Изображен на рисунке 25;
- dio\_exception.dart – перехватывает ошибки запросов и переводит их в консоль отладки;
- api/... .dart – файл реализации запроса, содержит типы и параметры запроса. Например product.dart изображен на рисунке 26.

```

class DioClient {
  final Dio _dio;

  DioClient(this._dio) {
    _dio
      ..options.baseUrl = Endpoints.baseUrl
      ..options.connectTimeout = Endpoints.connectionTimeout
      ..options.receiveTimeout = Endpoints.receiveTimeout
      ..options.responseType = ResponseType.json;
  }

  Future<Response> get(
    String url, {
    Map<String, dynamic>? queryParameters,
    Options? options,
    CancelToken? cancelToken,
    ProgressCallback? onReceiveProgress,
  }) async {
    try {
      final Response response = await _dio.get(
        url,
        queryParameters: queryParameters,
        options: options,
        cancelToken: cancelToken,
        onReceiveProgress: onReceiveProgress,
      );
      return response;
    } catch (e) {

```

Рисунок 24 – Часть клиента Dio

```

class Endpoints {
  Endpoints._();

  // base url
  static const String baseUrl = 'http://binevir.bokus.ru/';
  static const Duration receiveTimeout = Duration(milliseconds: 5000);
  static const Duration connectionTimeout = Duration(milliseconds: 5000);
  static const String products = '/api/products';
  static const String countries = '/api/countries';
  static const String applications = '/api/applications';
  static const String settings = '/api/settings';
  static const String guides = '/api/guides';
  static const String calculator = '/api/calculator/formula';
}

```

Рисунок 25 – Содержимое endpoints.dart



```

import 'package:binevir/data/http/dio_client.dart';
import 'package:dio/dio.dart';
import '../endpoints.dart';

class ProductApi {
  final DioClient dioClient;

  ProductApi({required this.dioClient});

  Future<Response> getProductApi(int id) async {
    try {
      final Response response =
        await dioClient.get('${Endpoints.products}/${id}');
      return response;
    } catch (e) {
      rethrow;
    }
  }

  Future<Response> getProductsApi() async {
    try {
      final Response response = await dioClient.get(Endpoints.products);
      return response;
    } catch (e) {
      rethrow;
    }
  }
}

```

Рисунок 26 – Содержимое api/product.dart

Для каждого типа данных существует свои репозиторий и модель. Репозиторий это слой приложения, который содержит в себе обработку и хранения данных, предоставляя удобные интерфейсы для взаимодействия, что чрезвычайно полезно при работе в команде и делегированию обязанностей по проектированию сложных проектов. В разрабатываемом приложении будет использоваться кеширование с использованием Nive. Логика проста: если к репозиторию обратятся за данными, то он делает апи-запрос для получения данных, при успехе сохраняет их в локальное зрелище и возвращает, при ошибке получения данных – вернёт сохраненные в Nive. Это повысит стабильность работы при отсутствующем интернете, так как данные хоть как-то будут получены, пусть и неактуальные. Код одного из

репозитория представлен на рисунке 27.

```
class GuideRepository {
  final GuideApi guideApi;
  late List<Guide> guides;
  bool isFromCache = true;
  GuideRepository(this.guideApi, {this.isFromCache = false});

  Future<List<Guide>> getGuideRequested() async {
    try {
      final response = await guideApi.getGuidesApi();
      final guides = (response.data['result']['guides'] as List)
        .map((e) => Guide.fromJson(e))
        .toList();
      final box = await Hive.openBox('guides');

      await box.put('guides', guides.map((e) => e.toJson()).toList());
      return guides;
    } on DioException catch (e) {
      final box = await Hive.openBox('guides');
      final cached = box.get('guides');
      if (cached != null && cached is List) {
        return cached
          .map((e) => Guide.fromJson(Map<String, dynamic>.from(e)))
          .toList();
      }
      final errorMessage = DioExceptions.fromDioError(e).toString();
      throw errorMessage;
    }
  }
}
```

Рисунок 27 – Код репозитория инструкций для экрана онбординга

Для наглядного представления порядка взаимодействия компонентов при работе с сервером и обработке данных, на рисунке 28 представлена swimlane-диаграмма, демонстрирующая последовательность действий между приложением, репозиторием, HTTP-клиентом (Dio) и локальным хранилищем Hive.

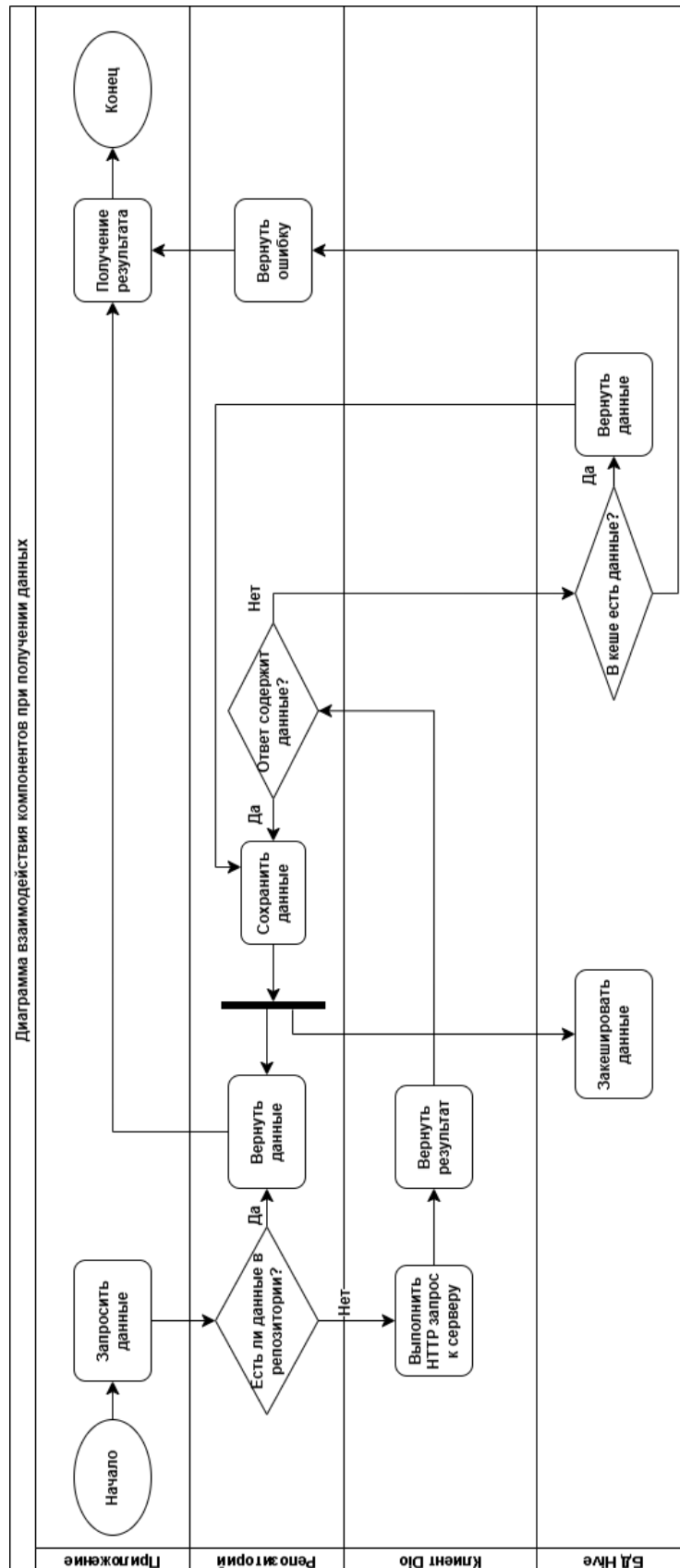


Рисунок 28 – Диаграмма взаимодействия компонентов при получении данных

Можно заметить, что список всех ответов нестандартного типа – Guide. Это модель соответствующего типа. Репозитории используют модели – специальные классы, описывающие структуру данных, получаемые с сервера или хранящихся локально. Они обеспечивают безопасность данных, а также помогают с сериализацией и десериализацией формата JSON. Пример модели Guide, представлен на рисунке 29.

```
class Guide {  
    final int id;  
    final String image;  
    final String? title;  
    final String? description;  
    final int sort;  
  
    Guide({  
        required this.id,  
        required this.image,  
        this.title,  
        this.description,  
        required this.sort,  
    });  
  
    static Guide fromJson(Map<String, dynamic> json) {  
        return Guide(  
            id: json['id'] as int,  
            image: json['image'].toString(),  
            title: json['title'].toString(),  
            description: json['description'].toString(),  
            sort: json['sort'] as int,  
        );  
    }  
}  
  
Map<String, dynamic> toJson() {  
    return {  
        'id': id,  
        'image': image,  
        'title': title,  
        'description': description,  
        'sort': sort  
    };  
}  
  
Guide copy({  
    int? id,  
    String? title,  
    String? image,  
    String? description,  
    int? sort,  
}) {  
    return Guide(  
        id: id ?? this.id,  
        title: title ?? this.title,  
        image: image ?? this.image,  
        description: description ?? this.description,  
        sort: sort ?? this.sort,  
    );  
}
```

Рисунок 29 – Код модели Guide

Так как одни и те же данные используются во многих компонентах имеет смысл задуматься о централизации логики для упрощения контроля над получением данных. С этим отлично справляется внедрение зависимостей – патерн проектирования с регистрацией синглтон (Единственные экземпляры) объектов, доступных из любой точки приложения, для этого используется функционал библиотеки GetIt. Так вместо того, чтобы создавать объекты репозитория с апи-клиентом и делать запрос возможно зарегистрировать их как зависимость и использовать в

любой точке кода. Код сервиса по внедрению представлен на рисунке 30.

```
final getIt = GetIt.instance;

Future<void> setup() async {
  getIt.registerSingleton(Dio());
  getIt.registerSingleton(DioClient(getIt<Dio>()));
  getIt.registerSingleton(CalculatorApi(dioClient: getIt<DioClient>()));
  getIt.registerSingleton(CalculatorRepository(getIt.get<CalculatorApi>()));
  getIt.registerSingleton(ProductApi(dioClient: getIt<DioClient>()));
  getIt.registerSingleton(ProductRepository(getIt.get<ProductApi>()));
  getIt.registerSingleton(GuideApi(dioClient: getIt<DioClient>()));
  getIt.registerSingleton(GuideRepository(getIt.get<GuideApi>()));
  getIt.registerSingleton(ApplicationApi(dioClient: getIt<DioClient>()));
  getIt.registerSingleton(ApplicationRepository(getIt.get<ApplicationApi>()));
  getIt.registerSingleton(SettingsApi(dioClient: getIt<DioClient>()));
  getIt.registerSingleton(SettingsRepository(getIt.get<SettingsApi>()));
  getIt.registerSingleton(CountryApi(dioClient: getIt<DioClient>()));
  getIt.registerSingleton(CountryRepository(getIt.get<CountryApi>()));
  getIt.registerSingleton(DataRepository());
}
```

Рисунок 30 – Код di/service\_locator.dar

Продолжая рассматривать экран онбординга, можно проанализировать логику его работы. Сначала при переходе на экран в BLoC отправляется событие загрузки данных, то есть запускается метод `getGuideRequested` из своего репозитория. Пока данные загружаются, BLoC находится в состоянии `OnboardingLoading` – в UI отображается индикатор загрузки. Внутри репозитория идет обращение через API к серверу, если не получилось получить ответ – вернуть сохраненные в Hive данные, иначе ничего. Если данные вернулись успешно, состояние меняется на `OnboardingLoaded` – в UI загружены слайды, иначе отображается виджет ошибки, позволяющий повторить попытку загрузки. Отображение экрана онбординга в трёх состояниях предоставлено на рисунках 31, 32, 33.

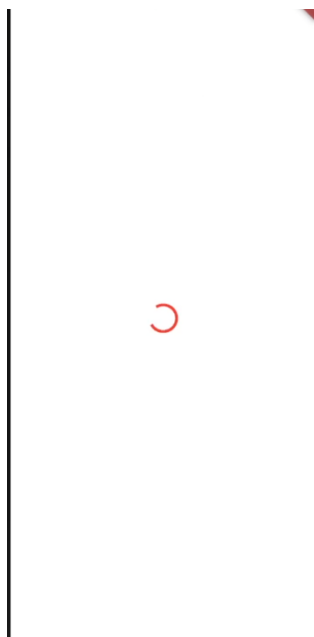


Рисунок 31 – Онбордиг в состоянии загрузки данных

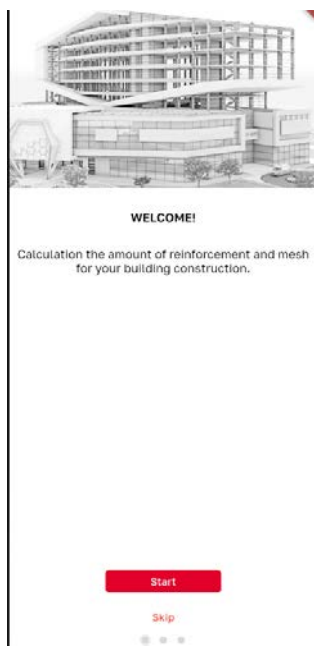


Рисунок 32 – Онбордиг в состоянии загруженных данных



Рисунок 33 – Онбордиг в состоянии ошибки получения данных

#### **2.4.5 Работа калькулятора**

Одним из наиболее функционально насыщенных компонентов приложения является калькулятор строительных материалов, предназначенный для расчёта количества товара при заданных параметрах. При входе на страницу калькулятора пользователю предоставляется возможность выбрать из выпадающего меню сферу применения (Например установка подпорных стен, строительство заборов, дорог, мостов, изготовление торкретбетона и другое). Сам расчет должен происходить по заранее подготовленным формулам из технического задания, представленными блок-схемами. Пример блок-схемы формулы для расчета представлен на рисунке 34.

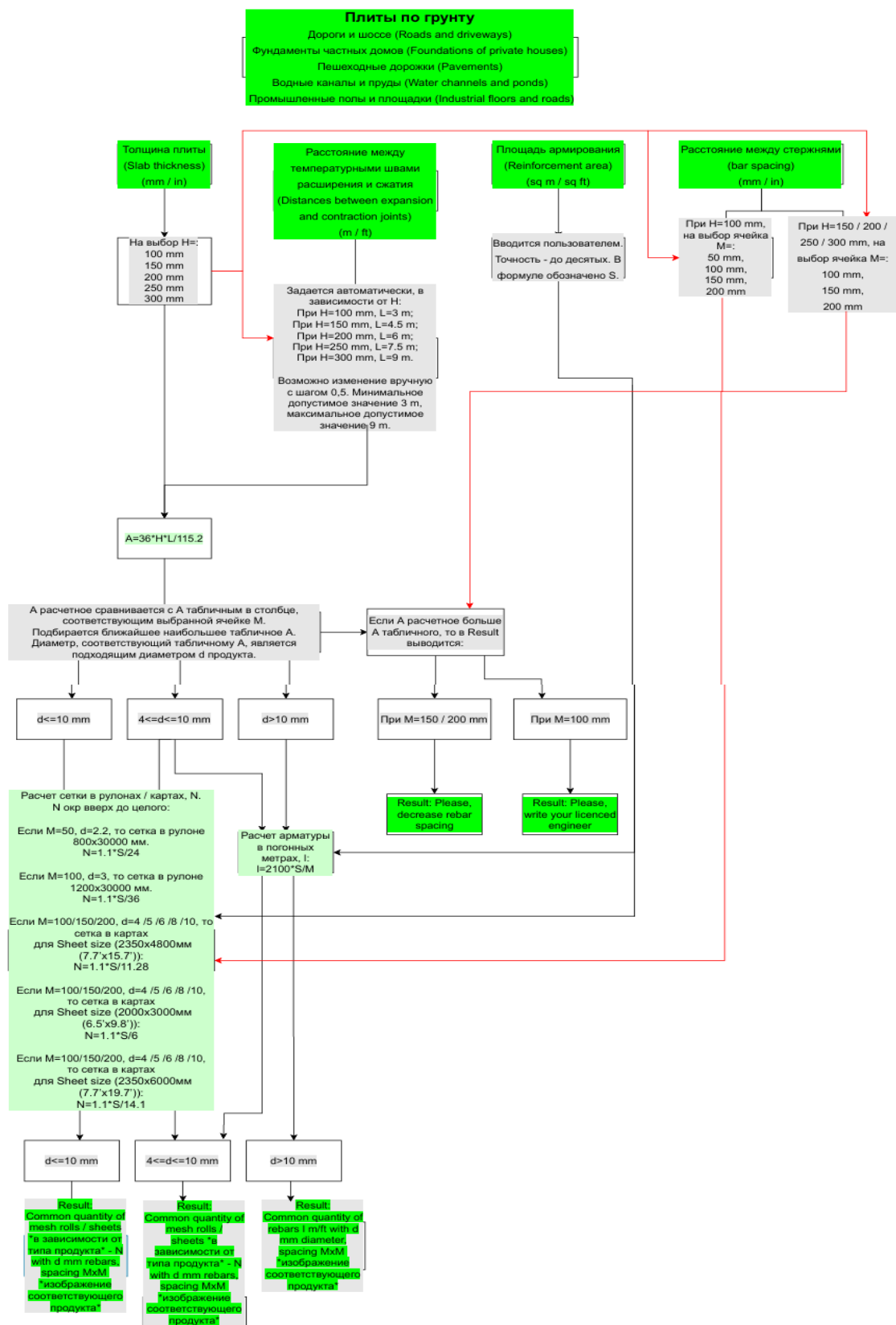


Рисунок 34 – Блок-схема расчета количества продукции для изготовления плит по грунту в зависимости от вводных данных



Часть кода калькулятора можно увидеть на рисунке 35.

```
void _onResultChanged(Widget resultWidget) {
  setState(() {
    _resultWidget = resultWidget;
  });
  WidgetsBinding.instance.addPostFrameCallback((_) { ...
}

Widget _buildFormulaWidget() {
  if (_selectedApplication == null) return const SizedBox();
  switch (_selectedApplication?.calculation_type) {
    case 'type_1':
      return ShotcreteFormula(
        onResultChanged: _onResultChanged,
        isSI: isSI);
    case 'type_2': ...
    case 'type_3': ...
    case 'type_4': ...
    case 'type_5': ...
    default: ...
  }
}

final Box person = Hive.box('person_box');
bool isSI = true;
final _countryRepository = getIt<CountryRepository>();
@override
void initState() {
  super.initState();
  final appUser = person.get('AppUser');
  final measurementCode = _countryRepository.countries.firstWhereOrNull(
    (country) => country.code == appUser?.country_code,)
    ?.measurement
    .name;

  isSI = (measurementCode == 'International system of units (SI)');
}
```

Рисунок 35 – Фрагмент кода экрана калькулятора с логикой отображения формулы и обновления результата

В калькуляторе важно обратить внимание на следующие составляющие:

- классу экрана создается пустой виджет resultWidget – он и будет хранить в себе результирующие значения;

– в объявлении присутствует выборочный билдер формулы `buildFormulaWidget`, который зависит от выбранной сферы применения. Внутри виджета формулы передаётся колбек изменяющая виджет и система измерений;

– Система измерений выбирается в зависимости от выбранной страны пользователя.

После выбора сферы применения загружается соответствующий тип формулы. В каждом из них нужно обратить внимание на следующие особенности:

– Возвращаемый виджет состоит из кастомных полей ввода `NumberInput` – виджет созданный специально для работы с вводом величин и их отображением в кратком виде как суффикс. Каждое поле ввода параметров так или иначе при изменении вызывает функцию `_recalculate()` – функция подсчёта результата по формуле. Код простейшего поля ввода с его отображением представлен на рисунке 36.



Рисунок 36 – Отображение поля ввода и его код

– Отображение величин в формах и в результате происходит в зависимости от текущей системы измерений. Весь подсчет происходит в СИ. Для учёта американской системы мер в начале и конце функции `_recalculate()` есть перевод в СИ и обратно с корректным отображением. Код одной из формул и соответствующей ей блок-схеме алгоритма расчета представлены на рисунках 37 и 38.

```
void _recalculate() {
    double S = double.TryParse(_sController.text.replaceAll(',', '.')) ?? 0;
    double N = 0, d = 0, meshX = 0, meshY = 30000;
    String spacingText = '', meshSizeText = '';
    String imageUrl = '';

    if (!widget.isSI) {
        S = ConversionHelper.convertArea(S, UnitSystem.us, UnitSystem.si);
    }
    if (_mValue == 50.0) {
        d = 2.2;
        N = (1.1 * S / 24).ceilToDouble();
        meshX = 800;
    } else if (_mValue == 100.0) {
        d = 3;
        N = (1.1 * S / 36).ceilToDouble();
        meshX = 1200;
    }
    imageUrl =
        'https://bivir.bokus.ru/storage/app_products/d0pGfN5Ch0oEkdX';

    if (!widget.isSI) {
        d = ConversionHelper.convertLengthMmInch(d, UnitSystem.si, UnitSystem.us);
        meshX = ConversionHelper.convertLengthMmInch(meshX, UnitSystem.si, UnitSystem.us);
        meshY = ConversionHelper.convertLengthMmInch(meshY, UnitSystem.si, UnitSystem.us);
        spacingText =
            '${ConversionHelper.convertLengthMmInch(_mValue, UnitSystem.si, UnitSystem.us)} mm';
    } else {
        spacingText = '${_mValue.toInt()}x${_mValue.toInt()} mm';
    }
    meshSizeText = '${meshX.toStringAsFixed(0)}x${meshY.toStringAsFixed(0)}';
    final resultWidget = Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
            Text( // Text ...
            Text( // Text ...
            Text( // Text ...
```

Рисунок 37 – Функция подсчета количества продукции для торкретирования в коде

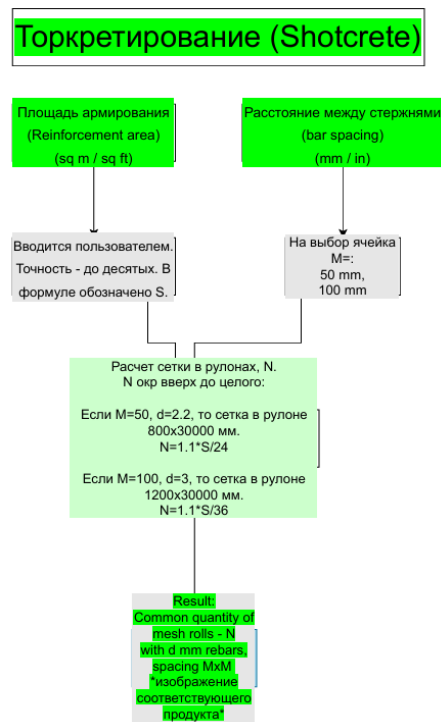


Рисунок 38 – Блок-схема алгоритма расчёта продукции для торкретирования

– Для корректной разработки был создан специальный помощник для перевода из измерительных систем – `conversion_helper.dart`, избавляющий код от повторений. Одна из функций классов переводов и отображения суффикса величины измерения представлена на рисунке 39.

```

enum UnitSystem { si, us }

class ConversionHelper {
  // Площадь
  static double convertArea(double value, UnitSystem from, UnitSystem to) {
    if (from == to) return value;
    return from == UnitSystem.si
      ? value * 10.7639 // m² -> ft²
      : value / 10.7639; // ft² -> m²
  }
  static String suffixArea(UnitSystem system) => system == UnitSystem.si ?
  
```

Рисунок 39– Фрагмент кода `conversion_helper.dart`

Для большего понимания алгоритма работы калькулятора была построена соответствующая блок-схема, представленная на рисунке 40.

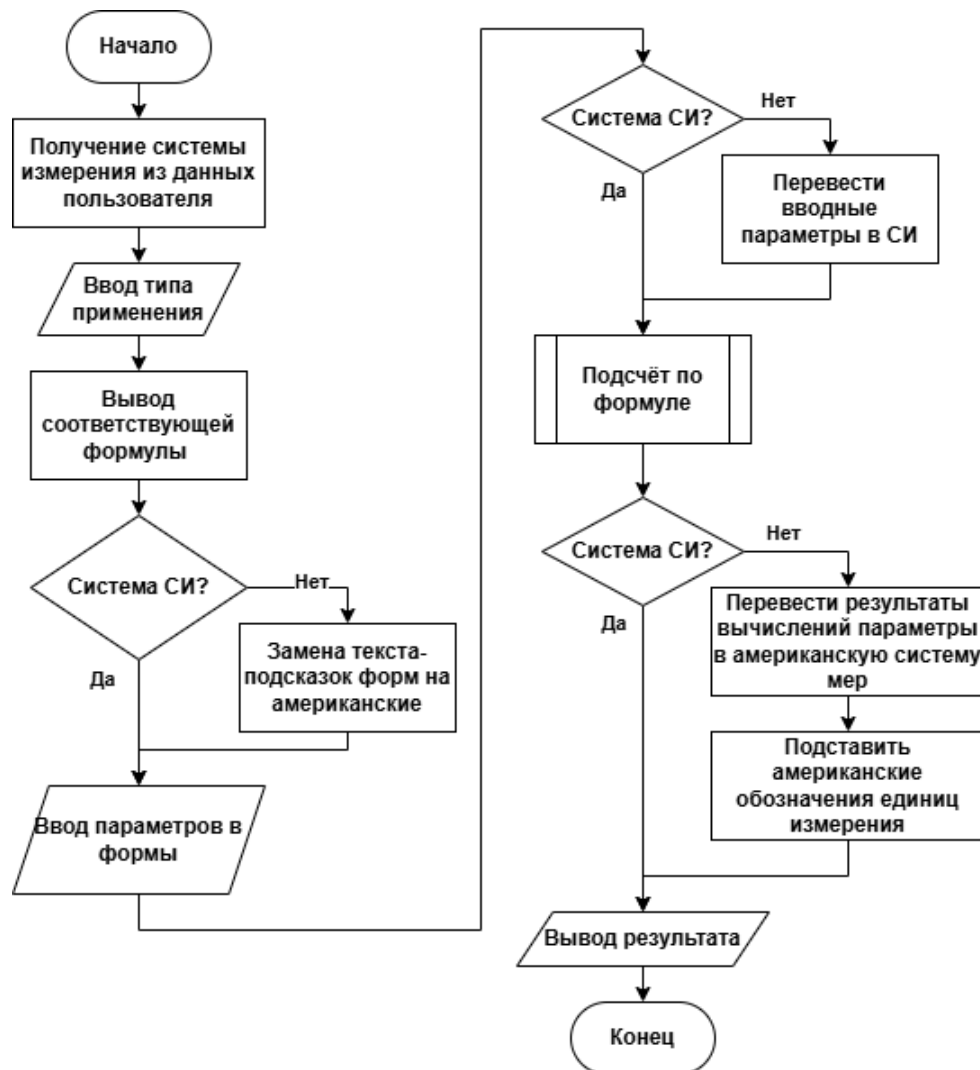


Рисунок 40 – Блок-схема алгоритма работы калькулятора

### Выводы по разделу

В рамках данного раздела была выполнена всесторонняя работа, включающая в себя выбор инструментов разработки, проектирование архитектуры проекта и UI, а саму программную реализацию мобильного приложения.

Исходя из требований к приложению и анализа существующих решений для реализации проекта был выбран фреймворк Flutter, который

позволяет охватить все мобильные платформы и позволит сократить время разработки, не потеряв много в качестве и производительности. Кроме того, отмечена перспективность данного инструмента и растущая популярность в последние года.

Пользовательский интерфейс проектировался на основе макетов и прототипов Figma, что позволило не отвлекаться на продумывание дизайна во время разработки. Маршрутизация полностью соответствует прототипу и понятно реализовано с использованием сторонней библиотеки.

Для построения архитектуры мобильного приложения были использованы шаблон BLoC для управления состоянием и реализаций API-сервисов, репозитория и внедрения зависимостей. Такой подход обеспечил модульность и способствовал упрощенному масштабированию. Также использованы такие библиотеки ответов запросов хранить, получать результаты и строить интерфейсы даже при отсутствии сети, что повышает стабильность приложения и улучшает пользовательский опыт.

В результате проведенной работы было создано мобильное приложение с устойчивой архитектурой, адаптивным дизайном, настроенным взаимодействием с бэкендом, а также реализован ряд функциональных модулей, как анимации, обработка ошибок, сохранение данных пользователя и настроек локально, поддержка локализации и создания тем, а также строительный калькулятор, выполняющий расчёты по заданным параметрам с поддержкой разных систем измерений.

### 3 Экспериментальный раздел

#### 3.1 Тестирование

Тестирование является последним, но не менее важным этапом разработки. На этом этапе проводится проверка функционала приложений и программ, корректное отображение, производительность и соответствие другим функциональным и не функциональным требованиям. Так выявляются ошибки и происходит их устранение.

Результаты тестирования можно увидеть в таблице 1.

Таблица 1 - Результаты тестирования функционала приложения

Функционал	Результат тестирования	Примечание
Заполнение личных данных пользователем	Успешно	
Изменение личных данных пользователя	Успешно	
Открытие ссылки на сайт после нажатия на адрес	Успешно	Используется стандартный браузер
Автоматическая подстановка номера после нажатия на него	Успешно	Используется стандартного приложение для звонков
Автоматическое открытие почты с адресатом	Успешно	Используется стандартного приложение для электронных писем
Валидация данных, вводимых пользователем о себе	Успешно	Возможно улучшение и поддержка большего формата номеров
Сохранение данных пользователя на сервере	Не реализовано	Нет поддержки сохранения клиентов в БД на сервере

Продолжение таблицы 1

Переход на другие экраны с помощью панели навигации	Успешно	
Получение и отображение данных с сервера с помощью API-запросов	Успешно	Запросы на получения не имеют параметров и всегда содержат информацию. Ожидается реализация отправки данных с учётом языка на сервере
Просмотр каталога в двух форматах	Успешно	
Просмотр экрана товара	Успешно	Для некоторых товаров не пока доступна информация о их характеристиках
Отображение и взаимодействие с трехмерной моделью товара	Успешно	Может сильно влиять на производительность и сильно разряжать аккумулятор
Выбор применения в калькуляторе	Успешно	
Подсчёт количества товара по заданным параметрам	Успешно	
Учёт системы измерений SI и US для ввода параметров и вывода результата	Частично	Пока не все формулы поддерживают расчёт в системе US
Отправка результатов подсчётов по почте	Не реализовано	На сервере пока не реализована поддержка SMTP
Поддержка тёмной темы	Частично	Требуются правки по дизайну и подбору цветов
Поддержка локализаций	Успешно	Пока переводы жестко вшиты в приложение. Ожидается добавление функционала на сервере получения нужных локализаций



### 3.1.1 Экраны приложения

Далее будут описаны все разработанные экраны приложения, их предназначения и взаимодействие с пользователем.

#### 3.1.1.1 Сплэш-экран

Первым экраном после запуска приложения является сплэш-экран. Представляет собой стильную визитку с логотипом компании-заказчика. Во время его отображения в фоновом режиме происходят запросы к серверу для последующего их использования. Экран изображен на рисунке 41.

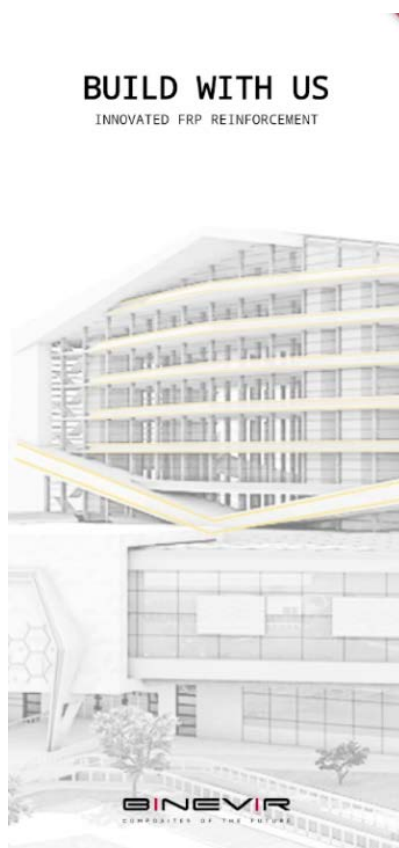


Рисунок 41 – Сплэш-экран

### 3.1.1.2 Онбординг-экран

После, если о пользователе нет информации, выходит онбординг-экран, содержащий краткую справку о приложении, а также располагает слайд с вводом данных пользователя. На каждой странице онбординга присутствует кнопка продвижения по логике («Начать» или «Продолжить на странице заполнения данных»), а также кнопка пропуска, ведущего к главному экрану, а также индикатор продвижения, показывающий расположение в очереди текущей страницы относительно других. Отдельная кнопка на слайде ввода данных необходима из-за другого функционала, так как она проводит до другого слайда, но ещё и содержит логику валидации данных пользователя

На рисунке 42 представлено отображение обучающего слайда, на 43 слайда ввода данных, а на рисунках 44 и 45 результат валидации вводимых данных.

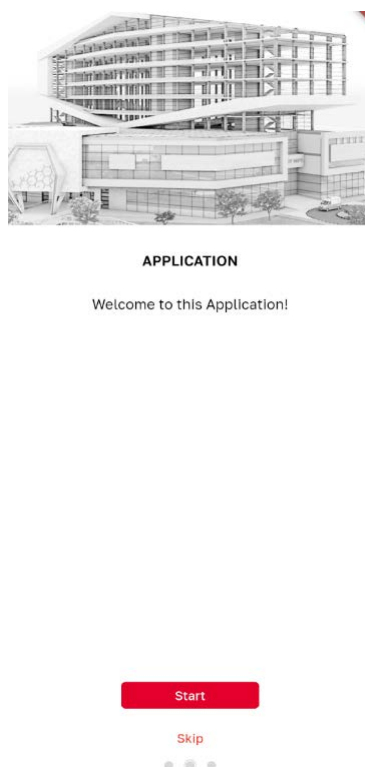


Рисунок 42 – Один из обучающих слайдов





Photo 

First name  \*

Second Name

Surname

Country  ▾

Company  \*

Job title

Phone

Email  \*

[Continue](#)

[Skip](#)




Рисунок 43 – Слайд с вводом данных пальзователя





Photo 

First name  \*

First name is required!

Second Name

Surname

Country  ▾

Company  \*

Job title

Phone

Email  \*

[Continue](#)

[Skip](#)




Рисунок 44 – Валидация данных по обязательным полям

The image shows a mobile application registration screen. At the top, there is a user profile icon and a vertical menu with icons for voice call, chat, confirmation, and settings. Below the profile is a 'Photo' button with a pencil icon. The form contains the following fields: 'First name' (filled with 'Ivan'), 'Second Name' (empty), 'Surname' (empty), 'Country' (dropdown menu showing 'Russia'), 'Company' (filled with 'ООО Zakupka'), 'Job title' (empty), 'Phone' (filled with '1111111111111111', highlighted with a red border and a red error message 'Error phone format' below it), and 'Email' (filled with 'testmail@mail.com'). At the bottom, there are two buttons: a red 'Continue' button and a red 'Skip' button. Below the buttons are three small circular indicators, with the first one being filled.

Рисунок 45 – Валидация данных по формату номера телефона

### 3.1.1.3 Главный экран

Главный экран представлен на рисунке 46. В текущей версии он не содержит специфического функционала, однако его структура позволяет легко дополнить его необходимыми возможностями по требованию заказчика.

Можно заметить, что на следующих описанных страницах присутствует те же компоненты, благодаря общей обёртке маршрутов (Панель навигации), и использованию обёрточного виджета Screen – Экран, на котором размещены ещё повторяющиеся виджеты: `user_panel` – верхняя панель с информацией о пользователе, `network_banner` – баннер, отображающийся, в зависимости от состояния сети. На рисунке 47 можно увидеть отображение главного экрана в зависимости с включенной и отключенной сетью.

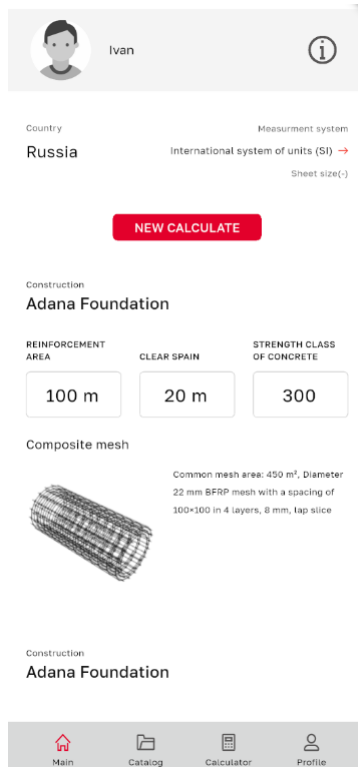


Рисунок 46 – Отображение главного экрана

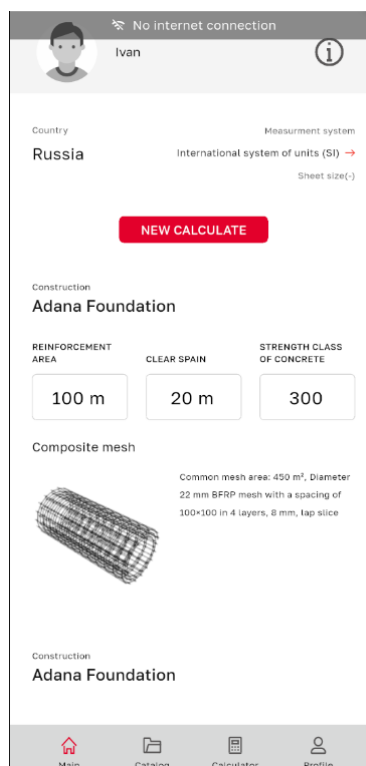


Рисунок 47 – Отображение главного экрана без доступа к интернету

### 3.1.1.4 Экран каталога

Экран каталога предоставляет возможность просмотреть доступные товары компании. Доступен выбор отображения в виде списка или плиток с помощью специального переключателя. Его значение передается в конструктор карточек товаров – `catalog.card`, и они отображаются в заданном виде. На данный момент доступно лишь 4 товара, однако сама страница построена с использованием `ListView.builder`. – виджета отображающего элементы в списке. Так будут загружаться только те карточке, которые непосредственно можно увидеть на экране для повышения производительности, а также доступен функционал для реализации ленивой загрузки. Из-за правильно организованной структуры сервисов, будет легко ввести загрузку большего количества товаров с обработкой страниц в запросе. Отображения экрана представлено на рисунках 48 и 49.

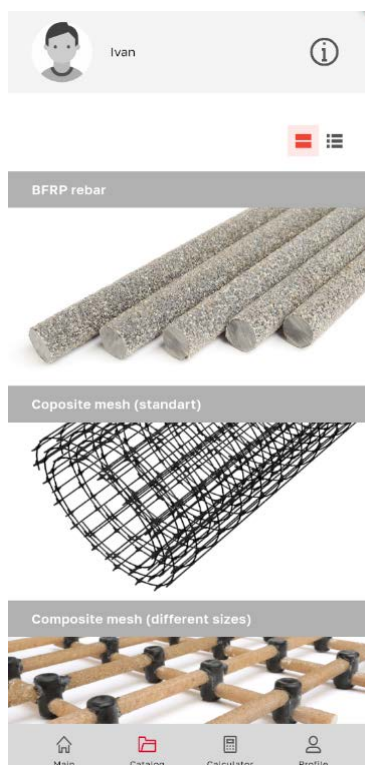


Рисунок 48 – Отображение каталога в режиме плитки

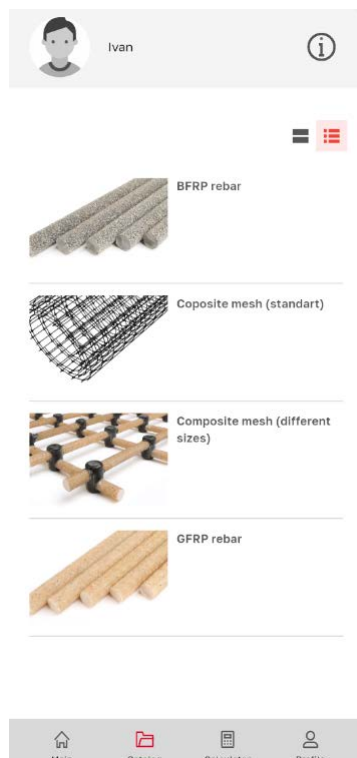


Рисунок 49 – Отображение каталога в режиме списка

### 3.1.1.5 Экран с подробной информацией о товаре

Нажав на какую-либо карточку в каталоге, пользователь переместится на экран с подробной информацией о товаре. Он предназначен для отображения характеристик и параметров о конкретной единице продукции.

Экран реализован на основе виджета `SingleChildScrollView` – обеспечивающий вертикальную прокрутку независимо от размера экранов.

Используются кастомные компоненты:

- `ProductModel` – отображает трехмерную модель товара или проекта (Реализована лишь она модель товара, используется как заглушка для все моделей);
- `NumberInput` – обеспечивает ввод данных числовых данных с системами измерений для сопоставления характеристик товара относительно друг друга;

- CustomCheckBox – в двух представлениях, говорящий о особенностях эксплуатации и упаковки товаров;
- ProductDescription – отображает иконки возможных сфер применений конкретного товара;
- ProductCalcButton – переход к калькулятору с переносом выбранных параметров и методов применения;
- ProductProject – проекты с примерами реализованных проектов с использованием данного продукта.

Отображение экрана продукта представлено на рисунке 50.

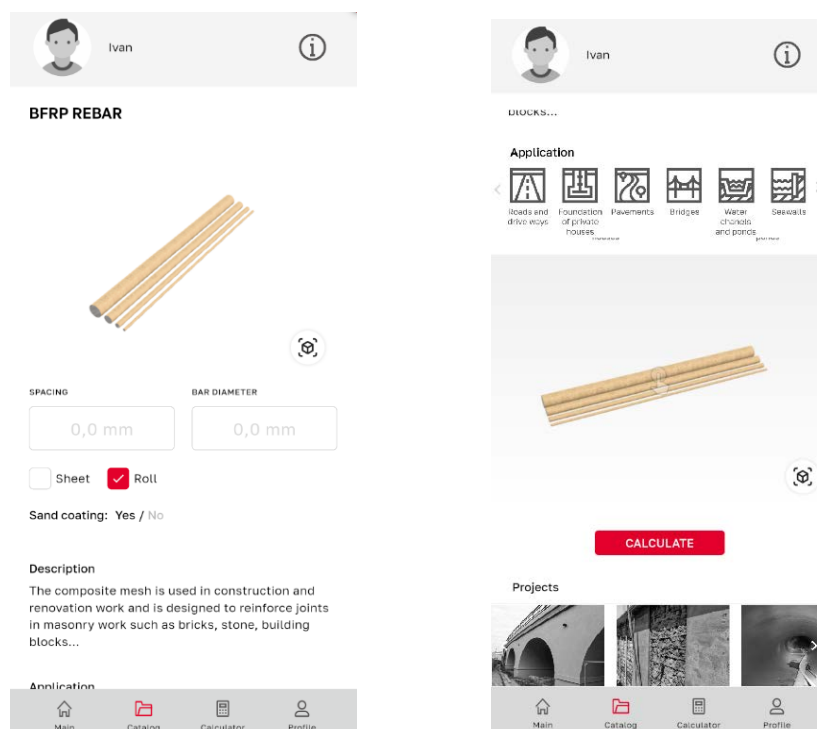


Рисунок 50 – Экран подробной информации о товаре

### 3.1.1.6 Экран профиля

Экран профиля предоставляет просмотр данных пользователя, а также их редактирование. Состоит из двух частей – просмотр профиля



(ProfileScreen), редактирование профиля (ProfileScreenEdit).

В основной части также отображаются данные менеджера, а в режиме редактирования можно сменить тему приложения и локализацию. Кнопка «Сохранить» имеет логику валидации данных, такой же как на онбординг-экране.

Отображение экранов профиля и его редактирования представлены на рисунке 51 и 52.

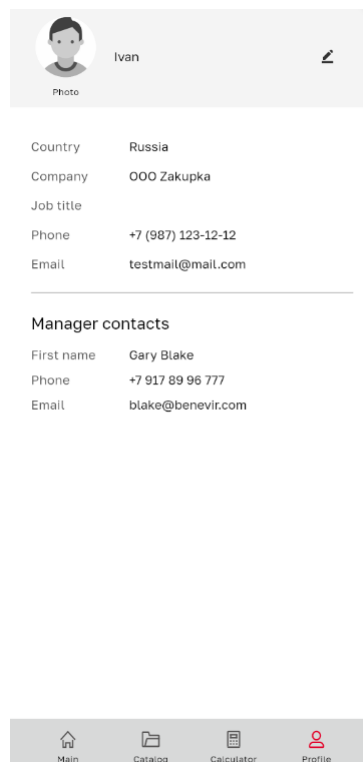


Рисунок 51 – Отображение экрана профиля в режиме просмотра

Photo Ivan

First name Ivan

Second Name

Surname

Country Russia

Company OOO Zakupka

Job title

Phone +7 (987) 123-12-12

Email testmail@mail.com

SAVE

App settings

Language English

Theme Light

Main Catalog Calculator Profile

Рисунок 52 – Отображение экрана профиля в режиме редактирования.

### 3.1.1.7 Экран калькулятора

Экран калькулятора используется для размещения строительного калькулятора. Подробный алгоритм работы и логика отображения формул описаны в проектно-конструкторском разделе, в подразделе посвященном работе калькулятора. Экран содержит кастомный виджет для отправки результатов на почту (На данный момент бэкенд не содержит SMTP сервера, кнопка является временной заглушкой, готовой к получению функциональности).

Отображение калькулятора и модального окна отправки письма представлено на рисунках 53 и 54.

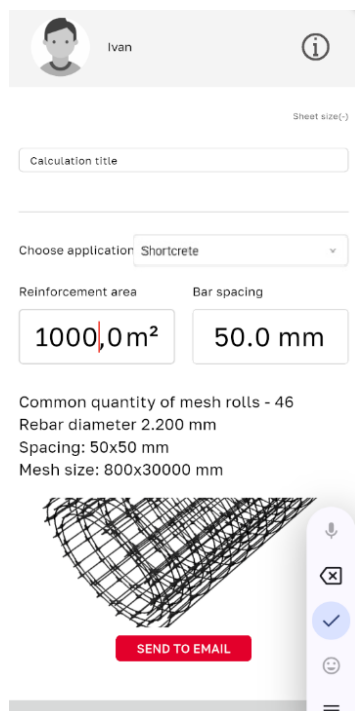


Рисунок 53 – Отображение формулы в калькуляторе

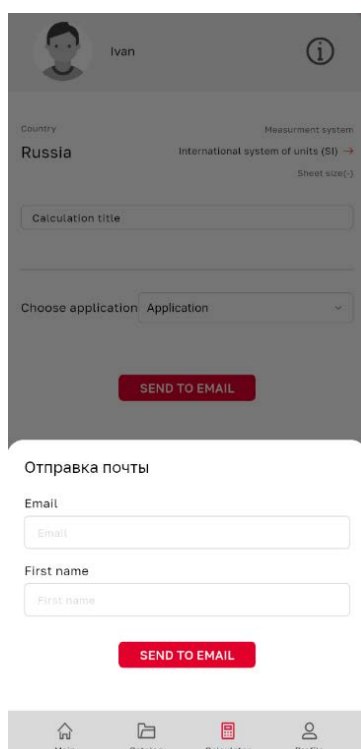


Рисунок 54 – Отображение модального окна для отправки на электронную почту на экране калькулятора

## **3.2 Проверка других функциональных требований**

### **3.2.1 Корректное лого, нативный сплэш, наименование приложения**

Настроенное лого и наименование приложения на устройстве и нативный сплэш запуска изображены на рисунках 55 и 56.



Рисунок 55 – Брендочая буква «В» на иконке приложения



Рисунок 56 –Нативный сплэш с лого компании при запуске приложения

### **3.2.2 Поддержка локализации**

Отображение одного и того же экрана с разной локализацией представлено на рисунках 57 и 58.



Рисунок 57 – Экран профиля в русской локализации



Рисунок 58 – Экран профиля в английской локализации

### 3.2.3 Поддержка светлой и темной темы

Отображение одного и того же экрана с разной темой представлена на рисунках 59 и 60 (Темная тема требует доработок для обеспечения соответствия дизайну и приятному внешнему виду).

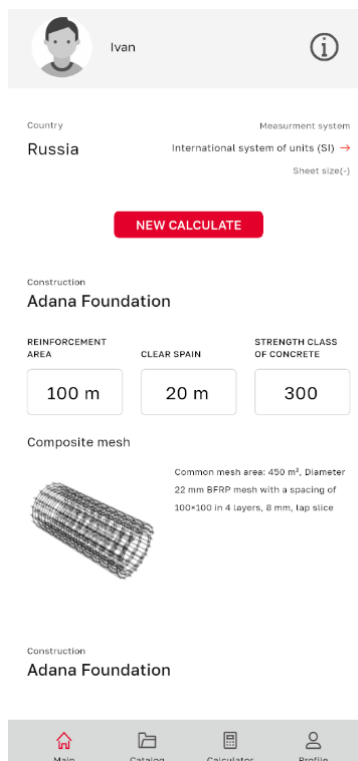


Рисунок 59– Главный экран с светлой темой

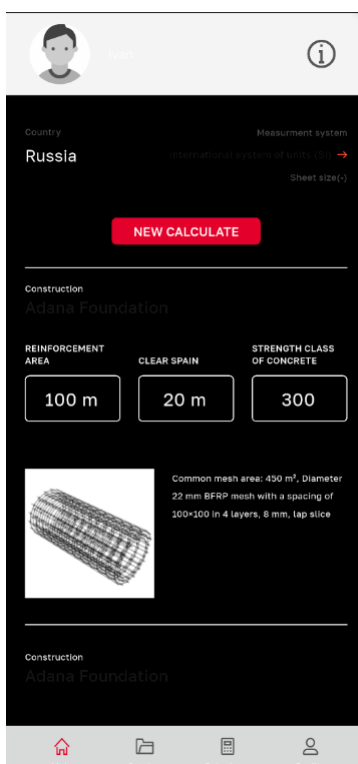


Рисунок 60– Главный экран с тёмной темой

### 3.2.4 Поддержка разных систем измерений

Отображение одной и той же формулы в разных системах измерения – в СИ и в Американской системе мер, представлено на рисунках 61 и 62.

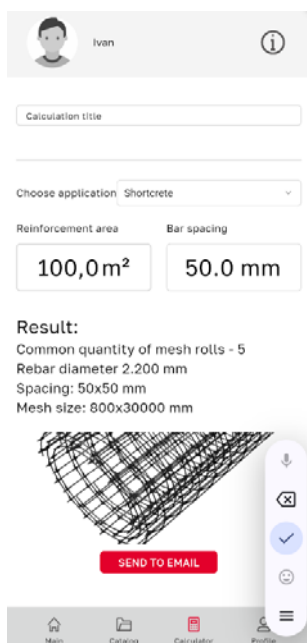


Рисунок 61 – Вычисление по формуле в системе СИ

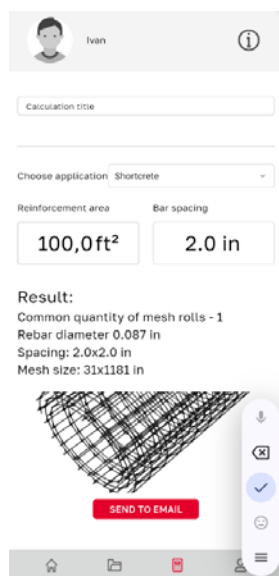


Рисунок 62 – Вычисление по формуле в американской системе мер

### 3.2.5 Загрузка фото в профиль

Модальное окно загрузки изображения и вид панели после добавления изображены на рисунках 63 и 64.

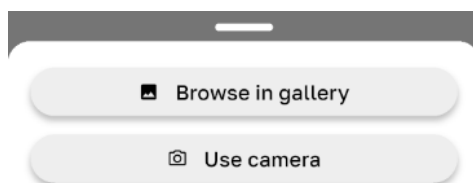


Рисунок 63 – Модальное окно добавление изображения

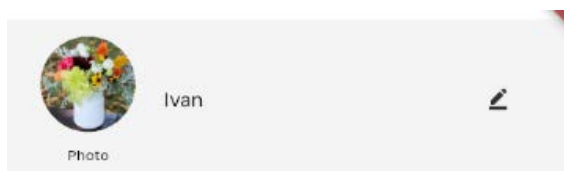


Рисунок 64 – Отображение выбранного изображения в профиле

#### Выводы по разделу

В ходе тестирования был проверен основной функционал каждого из компонентов. Реализованная в ходе разработки структура демонстрирует готовность к расширению функциональности.

Некоторые функции на текущем этапе не реализованы полностью ввиду отсутствия необходимой обработки на серверной части (Отправка писем через SMTP сервер, локализация отправляемых данных, отправка и установка файлов локализации, пагинация данных). Однако благодаря гибко спроектированной клиентской части, интеграция может быть выполнена по мере доработки бэкенда без необходимости в значительных изменениях.

Приложение демонстрирует высокий уровень к масштабированию и скорому выпуску полнофункциональной версии.



## ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы была успешно достигнута поставленная цель – разработка кроссплатформенного мобильного приложения «binevir», предназначенное для повышения эффективности взаимодействия между клиентом и магазином-поставщиком, повышения точности расчётов, автоматизацию подбора композитных материалов из ассортимента.

В ходе работы выполнено следующее:

- Проведен анализ предметной области с обоснованием актуальности и целесообразности разработки;
- Сформулированы функциональные и нефункциональные требования для тех задания;
- Выполнен выбор средств и инструментов для разработки приложения, в том числе и фреймворк – Flutter;
- Проработана архитектура управления состоянием на основе BLoC, проработаны сервисы для взаимодействия с сервером, методы хранения и использования данных, что обеспечило модульность приложения и расширяемость всего проекта;
- Реализован пользовательский в соответствии с дизайном проекта Figma;
- Создан и проработан строительный калькулятор, позволяющий производить расчеты необходимого количества строительных материалов, учитывающий систему измерения в стране пользователя;
- Организована поддержка разных языков и тем.

Во время тестирования был проверен весь основной функционал. Несмотря на то, что часть функций на текущем этапе не реализована из-за

отсутствия необходимой поддержки на сервере, архитектура проекта позволяет легко расширить функционал приложения для выполнения необходимых задач. Перспективы развития также включают в себя и улучшения интерфейса, в некоторых местах оптимизацию логики.

Выполненная работа имеет практическую значимость и перспективность в сфере строительной торговли. Разработанное программное обеспечение после внедрения и доработки функционала способно повысить конкурентоспособность компании-заказчика за счёт улучшения клиентского сервиса и оптимизации бизнес-процессов.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Веб-сайт Vinevir от команды Bokus [Электронный ресурс]. – Режим доступа: <https://vinevir.com/#about> (дата обращения 15.04.2025).
2. Документация Xcode [Электронный ресурс]. – Режим доступа: <https://developer.apple.com/documentation/xcode/> (дата обращения 16.04.2025).
3. Документация Swift [Электронный ресурс]. – Режим доступа: <https://developer.apple.com/documentation/swift> (дата обращения 16.04.2025).
4. Документация Objective-C [Электронный ресурс]. – Режим доступа: <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html> (дата обращения 16.04.2025).
5. Документация Android Studio [Электронный ресурс]. – Режим доступа: <https://developer.android.com/develop?hl=ru> (дата обращения 17.04.2025).
6. Документация Kotlin [Электронный ресурс]. – Режим доступа: <https://kotlinlang.org/docs/home.html> (дата обращения 17.04.2025).
7. Документация Java [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/en/java/> (дата обращения 17.04.2025).
8. Документация React Native [Электронный ресурс]. – Режим доступа: <https://reactnative.dev/docs/getting-started> (дата обращения 18.04.2025).
9. Документация Xamarin [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/previous-versions/xamarin/> (дата обращения 19.04.2025).
10. Документация Flutter [Электронный ресурс]. – Режим доступа: <https://docs.flutter.dev/> (дата обращения 20.04.2025).
11. Документация Dart [Электронный ресурс]. – Режим доступа: <https://dart.dev/docs> (дата обращения 20.04.2025).
12. Документация Javascript [Электронный ресурс]. – Режим доступа: <https://developer.mozilla.org/ru/docs/Web/JavaScript> (дата обращения 18.04.2025).
13. Документация Ionic [Электронный ресурс]. – Режим доступа: <https://ionicframework.com/docs> (дата обращения 21.04.2025).

14. Документация Angular [Электронный ресурс]. – Режим доступа: <https://angular-doc.ru/docs> (дата обращения 21.04.2025).
15. Документация Apache Cordova [Электронный ресурс]. – Режим доступа: <https://cordova.apache.org/docs/en/12.x/> (дата обращения 22.04.2025).
16. Документация C# [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения 22.04.2025).
17. Документация .NET [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/dotnet/> (дата обращения 22.04.2025).
18. Публикация statista [Электронный ресурс]. – Режим доступа: <https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/> (дата обращения 23.04.2025).
19. Документация React [Электронный ресурс]. – Режим доступа: <https://ru.legacy.reactjs.org/docs/getting-started.html> (дата обращения 24.04.2025).
20. Документация IntelliJ IDEA [Электронный ресурс]. – Режим доступа: <https://www.jetbrains.com/ru-ru/idea/resources/> (дата обращения 24.04.2025).
21. Документация – Visual Studio Code [Электронный ресурс]. – Режим доступа: <https://code.visualstudio.com/docs> (дата обращения 28.04.2025).
22. Документация GitHub [Электронный ресурс]. – Режим доступа: <https://docs.github.com/ru> (дата обращения 28.04.2025).
23. Документация Nginx [Электронный ресурс]. – Режим доступа: <https://nginx.org/ru/docs/> (дата обращения 29.04.2025).
24. Документация Figma [Электронный ресурс]. – Режим доступа: <https://www.figma.com/community/file/813826100927416632/figma-russian> (дата обращения 20.04.2025).
25. Документация Docker [Электронный ресурс]. – Режим доступа: <https://docs.docker.com/> (дата обращения 25.05.2025).

## ПРИЛОЖЕНИЕ А

### Пример ответа сервера на API-запроса для получения данных для онбординг-экрана

```
{
  "success": true,
  "result": {
    "guides": [
      {
        "id": 1,
        "title": "Welcome!",
        "description": "Calculation the amount of reinforcement and mesh
for your building construction.",
        "image":
"https://binevir.bokus.ru/storage/guides/kHilIPd5iCdMU14It7RWHpJuCQcwy
e-metaaw1hZ2UgMS5qcGc=-.jpg",
        "sort": 500
      },
      {
        "id": 2,
        "title": "Application",
        "description": "Welcome to this Application!",
        "image":
"https://binevir.bokus.ru/storage/guides/wZ0nJ5TztPXHMtNiymALpIABw1nR
df-metaaw1hZ2UgMS5qcGc=-.jpg",
        "sort": 500
      }
    ]
  }
}
```

## ПРИЛОЖЕНИЕ Б

### Код сплэш-крана

```
import 'dart:async';
import 'package:binevir/components/loading_failure.dart';
import 'package:binevir/data/repository/application_repository.dart';
import 'package:binevir/data/repository/country_repository.dart';
import 'package:binevir/data/repository/settings_repository.dart';
import 'package:binevir/di/service_locator.dart';
import 'package:binevir/screens/splash/bloc/splash_bloc.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:go_router/go_router.dart';
import 'package:hive_flutter/hive_flutter.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';

class SplashScreen extends StatefulWidget {
  const SplashScreen({super.key});

  @override
  State<SplashScreen> createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
  final Box personBox = Hive.box('person_box');

  bool settingsLoaded = false;
  bool applicationsLoaded = false;
  bool countriesLoaded = false;

  final _splashBlock = SplashBloc(
```

```
getIt<CountryRepository>(),  
getIt<ApplicationRepository>(),  
getIt<SettingsRepository>());
```

```
@override  
void initState() {  
  _splashBlock.add(LoadData());  
  super.initState();  
}
```

```
@override  
void dispose() {  
  super.dispose();  
}
```

```
@override  
void didChangeDependencies() {  
  super.didChangeDependencies();  
}
```

```
@override  
Widget build(BuildContext context) {  
  return Container(  
    width: double.infinity,  
    decoration: const BoxDecoration(  
      color: Colors.white,  
    ),  
    child: SafeArea(  
      child: BlocConsumer<SplashBloc, SplashState>(  
        bloc: _splashBlock,  
        listener: (context, state) {
```

```

if (state is DataLoaded) {
  final appUser = personBox.get('AppUser');
  Timer(const Duration(seconds: 1), () async {
    // if (appUser == null) {
    //   context.go('/welcome');
    // } else {
    //   context.go('/');
    // }
    context.go('/welcome');
  });
}
},
builder: (context, state) {
  if (state is DataLoadingFailure) {
    return LoadingFailureWidget(
      onRetry:(){
        _splashBlock.add(LoadData());
      }
    );
  }
  return Column(
    crossAxisAlignment: CrossAxisAlignment.center,
    children: [
      const SizedBox(
        height: 50,
      ),
      Text(
        AppLocalizations.of(context)!.splashTitle.toUpperCase(),
        style: const TextStyle(
          fontSize: 32.0,
          fontWeight: FontWeight.w800,

```



```

        height: 1.2,
        decoration: TextDecoration.none,
        color: Colors.black,
    ),
),
const SizedBox(
    height: 5.0,
),
Text(
    AppLocalizations.of(context)!.splashDescription.toUpperCase(),
    style: const TextStyle(
        fontSize: 14.0,
        fontWeight: FontWeight.w400,
        height: 1.2,
        decoration: TextDecoration.none,
        color: Colors.black,
    ),
),
Expanded(
    child: Stack(
        children: [
            Container(
                decoration: BoxDecoration(
                    color: Colors.white,
                    image: DecorationImage(
                        image:
                            Image.asset('assets/images/splash.png').image,
                        fit: BoxFit.cover,
                    ),
                ),
        ],
    ),
),

```

```

        Positioned(
          bottom: 50,
          left: 0,
          right: 0,
          child: Container(
            alignment: Alignment.center,
            child: Image.asset(
              'assets/images/logo.png',
              width: 169.0,
            ),
          ),
        ),
      ],
    ),
  ],
);
},
),
),
);
}
}

```

## ПРИЛОЖЕНИЕ В

### **BLoC составляющая сплеш-экрана**

```
import 'dart:async';

import 'package:binevir/data/models/country.dart';
import 'package:binevir/data/models/product.dart';
import 'package:binevir/data/models/settings.dart';
import 'package:binevir/data/repository/application_repository.dart';
import 'package:binevir/data/repository/country_repository.dart';
import 'package:binevir/data/repository/settings_repository.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

part 'splash_event.dart';
part 'splash_state.dart';

class SplashBloc extends Bloc<SplashEvent, SplashState> {
  SplashBloc(this.countryRepository, this.applicationRepository,
    this.settingsRepository)
    : super(SplashInitial()) {
    on<LoadData>((event, emit) async {
      try {
        if (state is! DataLoaded) {
          emit(DataLoading());
        }
        final countries = await countryRepository.getCountriesRequested();
        final applications =
          await applicationRepository.getApplicationsRequested();
        final settings = await settingsRepository.getSettingsRequested();
        emit(DataLoaded(
          countries: countries,
          settings: settings,
          applications: applications,
```

```

    ));
  } catch (e) {
    emit(DataLoadingFailure(exception: e));
  } finally {
    event.completer?.complete();
  }
});
}
final CountryRepository countryRepository;
final ApplicationRepository applicationRepository;
final SettingsRepository settingsRepository;
}
part of 'splash_bloc.dart';

```

```

abstract class SplashEvent {}

```

```

class LoadData extends SplashEvent {
  final Completer? completer;

  LoadData({
    this.completer,
  });
}
part of 'splash_bloc.dart';

```

```

abstract class SplashState {}

```

```

class SplashInitial extends SplashState {}

```

```

class DataLoaded extends SplashState {
  DataLoaded({
    required this.countries,
    required this.applications,
    required this.settings,
  });
}

```

```

    });
    final List<Country> countries;
    final List<Application> applications;
    final Settings settings;
}

class DataLoading extends SplashState {}

class DataLoadingFailure extends SplashState {
    DataLoadingFailure({
        required this.exception,
    });
    final Object? exception;
}

```

## ПРИЛОЖЕНИЕ Г

### Графические материалы

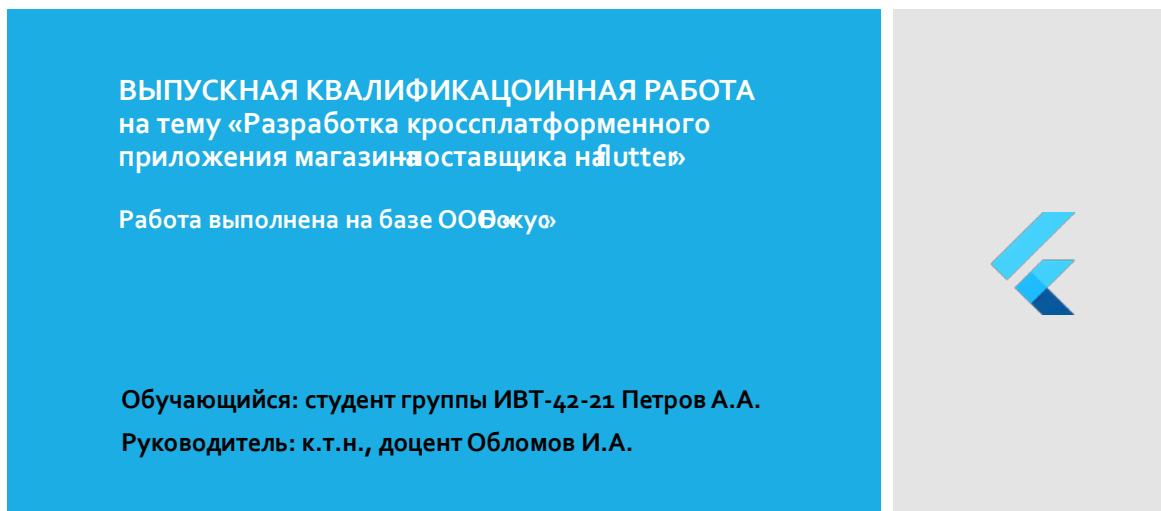


Рисунок Г.1 – Слайд 1. Титульный слайд

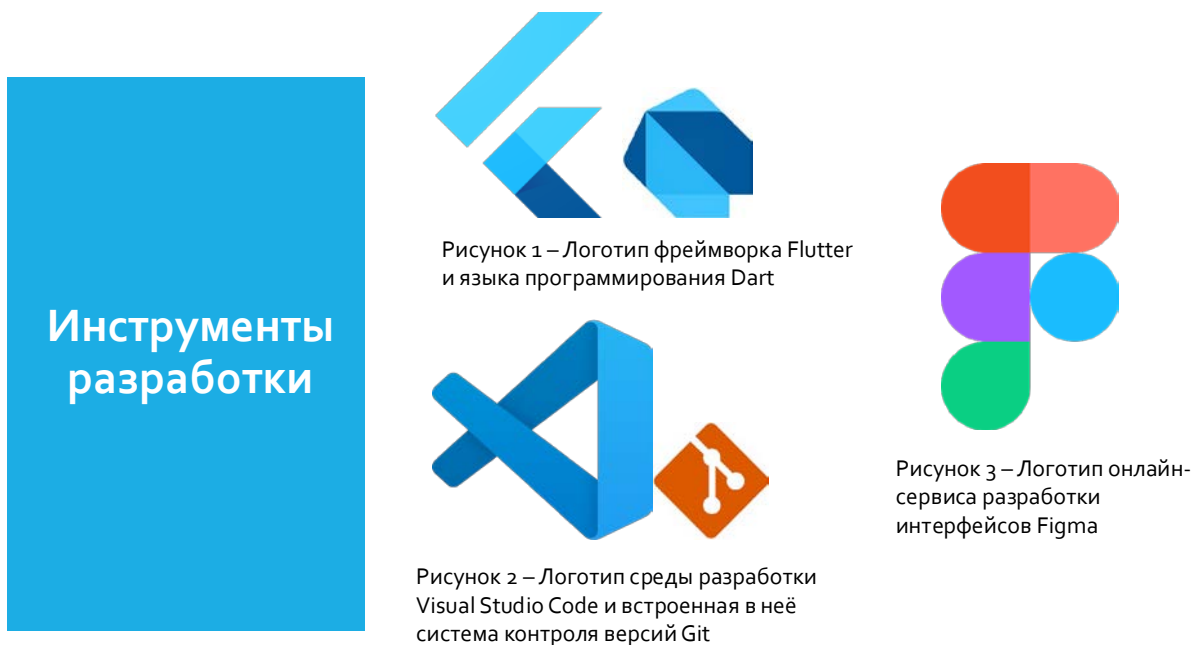


Рисунок Г.2 – Слайд 2. Инструменты разработки

## Диаграмма вариантов использования приложения

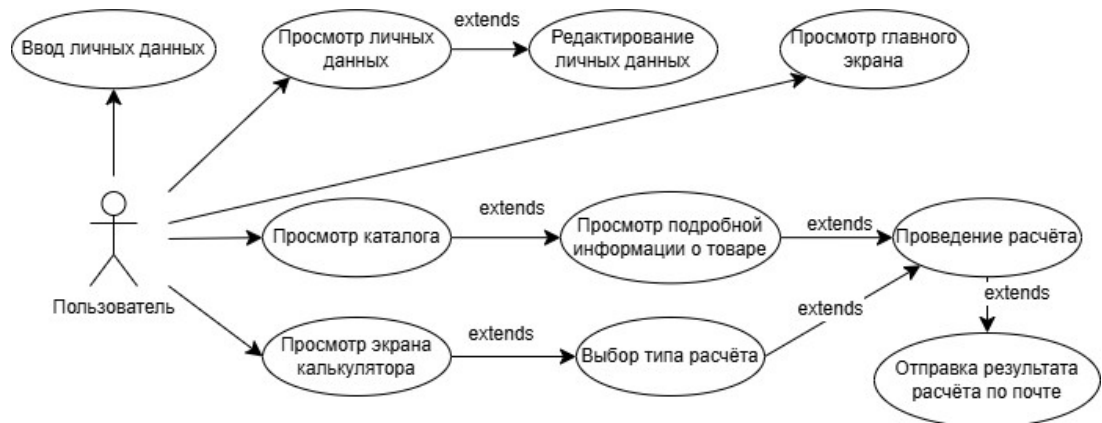


Рисунок Г.3 – Слайд 3. Диаграмма вариантов использования

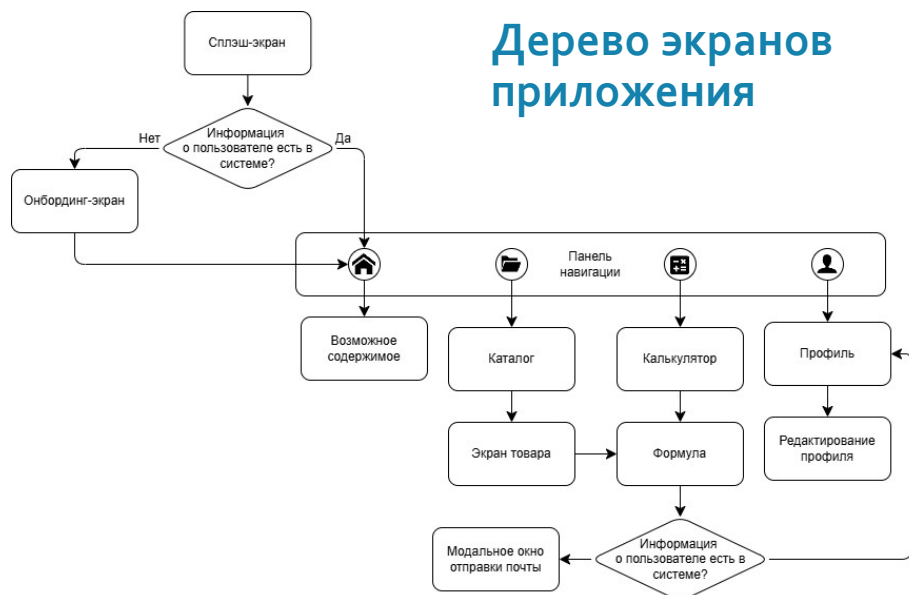


Рисунок Г.4 – Слайд 4. Дерево экранов приложения

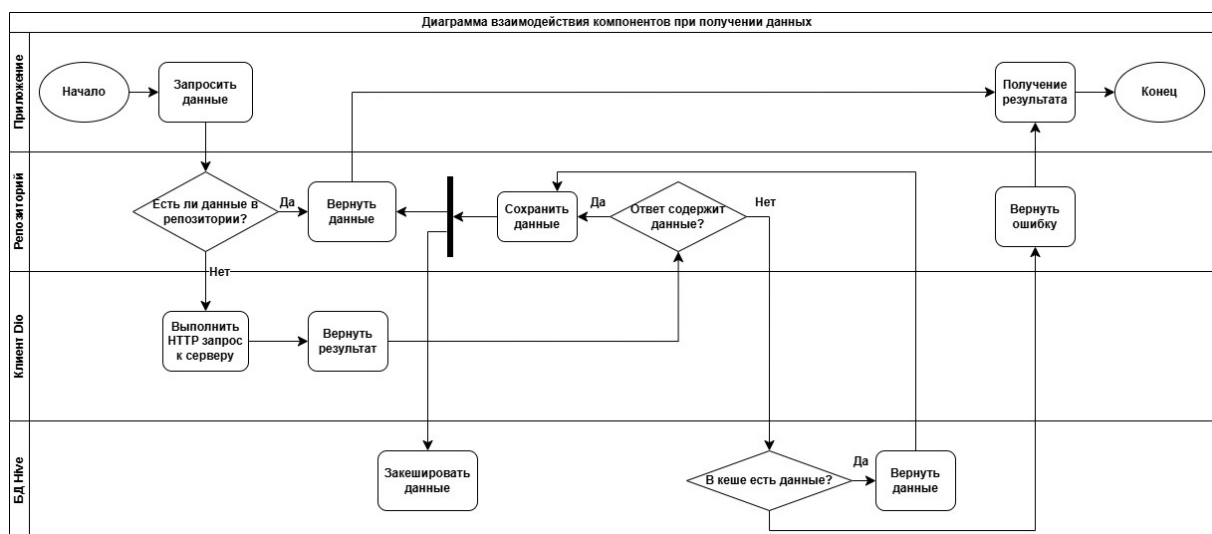


Рисунок Г.5 – Слайд 5. Диаграмма взаимодействия компонентов при получении данных

## Работа калькулятора

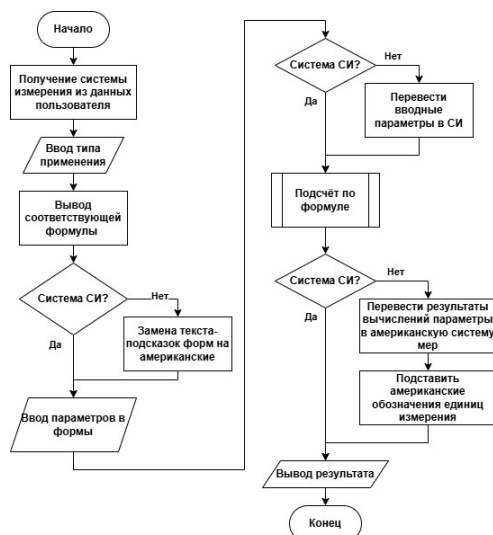


Рисунок 1 – Блок-схема алгоритма работы калькулятора

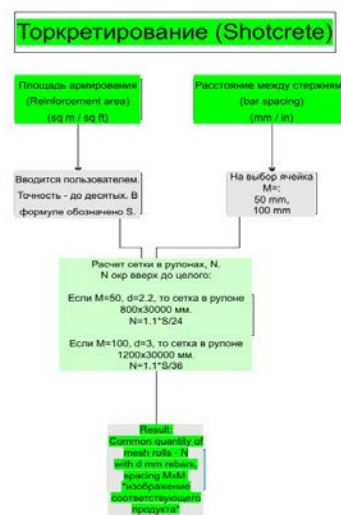


Рисунок 2 – Блок-схема алгоритма расчёта продукции для торкретирования

Рисунок Г.6 – Слайд 6. Работа калькулятора



## Отображение расчета в калькуляторе с разными системами измерения

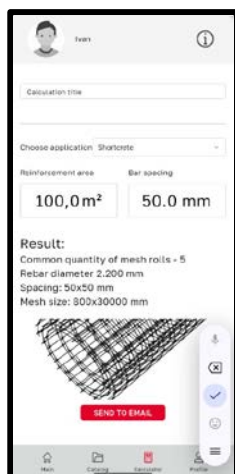


Рисунок 1 – Подсчет материалов в СИ

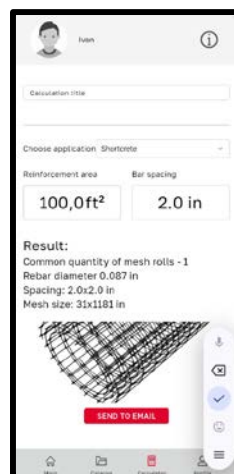


Рисунок 2 – Подсчет материалов в американской системе мер

Рисунок Г.7 – Слайд 7. Отображение расчёта в калькуляторе с разными системами измерения