МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Мобильное приложение компьютерного клуба

Исполнитель

студент 3 курса 7 группы Двоскин Арсений Тимофеевич

(Ф.И.О.)

Руководитель работы преп.-стажер, Уласевич Николай Иванович

(учен. степень, звание, должность, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель

(подпись)

Минск 2024

Содержание

[Введение 3](#_Toc167098915)

[1 Обзор аналогов и постановка задачи 4](#_Toc167098916)

[1.1 Основные сведения по теме «Мобильное приложение» 4](#_Toc167098917)

[1.2 BookMe 4](#_Toc167098918)

[1.3 SeatMe 6](#_Toc167098919)

[1.4 ClubReserve 8](#_Toc167098920)

[1.5 PlaceBook 10](#_Toc167098921)

[1.6 Вывод 11](#_Toc167098922)

[2 Проектирование мобильного приложения и выбор технологий 13](#_Toc167098923)

[2.1 Выбор платформы и технологий 13](#_Toc167098924)

[2.1.1 Flutter 13](#_Toc167098925)

[2.1.2 Dart 13](#_Toc167098926)

[2.1.3 Widgets 14](#_Toc167098927)

[2.1.4 Isar 15](#_Toc167098928)

[2.1.5 Выбор IDE для разработки 17](#_Toc167098929)

[2.2 Проектирование базы данных 18](#_Toc167098930)

[2.3 Проектирование структуры проекта 20](#_Toc167098931)

[2.4 Диаграмма вариантов использования 20](#_Toc167098932)

[3 Реализация приложения. 22](#_Toc167098933)

[3.1 Разработка базы данных 22](#_Toc167098934)

[3.2 Разработка серверной части проекта 23](#_Toc167098935)

[4 Тестирование приложения 28](#_Toc167098936)

[5 Руководство пользователя 31](#_Toc167098937)

[Заключение 37](#_Toc167098938)

[Список использованных источников 38](#_Toc167098939)

[Приложение А 39](#_Toc167098940)

[Приложение Б 43](#_Toc167098941)

[Приложение В 49](#_Toc167098942)

### Введение

Современные компьютерные клубы играют важную роль в жизни геймеров и IT-энтузиастов, предоставляя место для игр, обучения и общения. В условиях растущей популярности мобильных технологий и приложений, компьютерные клубы сталкиваются с необходимостью интеграции новых цифровых решений для улучшения своих сервисов и повышения уровня удовлетворенности клиентов. Мобильное приложение для компьютерного клуба может стать эффективным инструментом для оптимизации работы клуба и предоставления пользователям удобного доступа к его услугам.

В последние годы наблюдается значительное увеличение числа пользователей мобильных устройств, что приводит к росту спроса на мобильные приложения, предоставляющие быстрый и удобный доступ к различным услугам. Компьютерные клубы, несмотря на свою популярность, часто не используют полностью возможности, которые предоставляют мобильные приложения. Создание мобильного приложения для компьютерного клуба позволит улучшить взаимодействие с клиентами, упростить процесс бронирования игровых мест, предоставлять актуальную информацию о мероприятиях и акциях, а также создавать персонализированные предложения для пользователей.

Целью данного курсового проекта является разработка мобильного приложения для компьютерного клуба, которое будет удовлетворять потребности пользователей и способствовать улучшению сервиса клуба. Для достижения этой цели поставлены следующие задачи:

Анализ потребностей пользователей и функциональных требований: Изучение текущих потребностей пользователей компьютерного клуба и определение основных функций, которые должны быть реализованы в приложении.

Проектирование пользовательского интерфейса, разработка интуитивно понятного и привлекательного интерфейса, обеспечивающего удобство использования приложения, разработка и внедрение функционала для бронирования игровых мест, тестирование и оптимизация приложения.

Для разработки мобильного приложения будет использован фреймворк Flutter, который позволяет создавать кросс-платформенные приложения с единой кодовой базой для iOS и Android. Язык программирования Dart будет использоваться для написания кода.

# Обзор аналогов и постановка задачи

В ходе выполнения курсового проекта были проанализированы аналоги приложения для бронирования мест в компьютерном клубе, выявлены их отличительные особенности.

## **1.1 Основные сведения по теме «Мобильное приложение»**

Мобильное – это программное обеспечение, разработанное для работы на мобильных устройствах, таких как смартфоны и планшеты. Эти приложения могут выполняться на различных операционных системах, таких как Android и iOS, и могут предоставлять пользователям широкий спектр функциональных возможностей, от общения и развлечений до работы и образования.

Типы мобильных приложений:

1. Нативные приложения:

Разрабатываются специально для конкретной операционной системы (например, Java/Kotlin для Android, Swift/Objective-C для iOS).

Преимущества: Высокая производительность, доступ к аппаратным ресурсам устройства (камера, GPS и т.д.), лучший пользовательский опыт.

Недостатки: требуется разработка отдельной версии для каждой платформы, что увеличивает затраты на разработку и поддержку.

1. Кроссплатформенные приложения:

Создаются с использованием фреймворков, которые позволяют писать код один раз и запускать его на разных платформах (например, Flutter, React Native, Xamarin).

Преимущества: Снижение затрат на разработку и поддержку, единая кодовая база.

Недостатки: потенциально ниже производительность по сравнению с нативными приложениями, возможны ограничения в доступе к некоторым аппаратным ресурсам.

## **1.2 BookMe**

Приложение BookMe [1] предназначено для бронирования мест в различных заведениях, включая компьютерные клубы (рисунок 1.1).

Вот его основные плюсы и минусы:

Плюсы:

1. Удобство использования:

Интуитивный интерфейс: Пользователи могут легко ориентироваться в приложении благодаря простому и понятному дизайну.

Мобильная доступность: Приложение доступно как на iOS, так и на Android, что позволяет использовать его на различных устройствах.

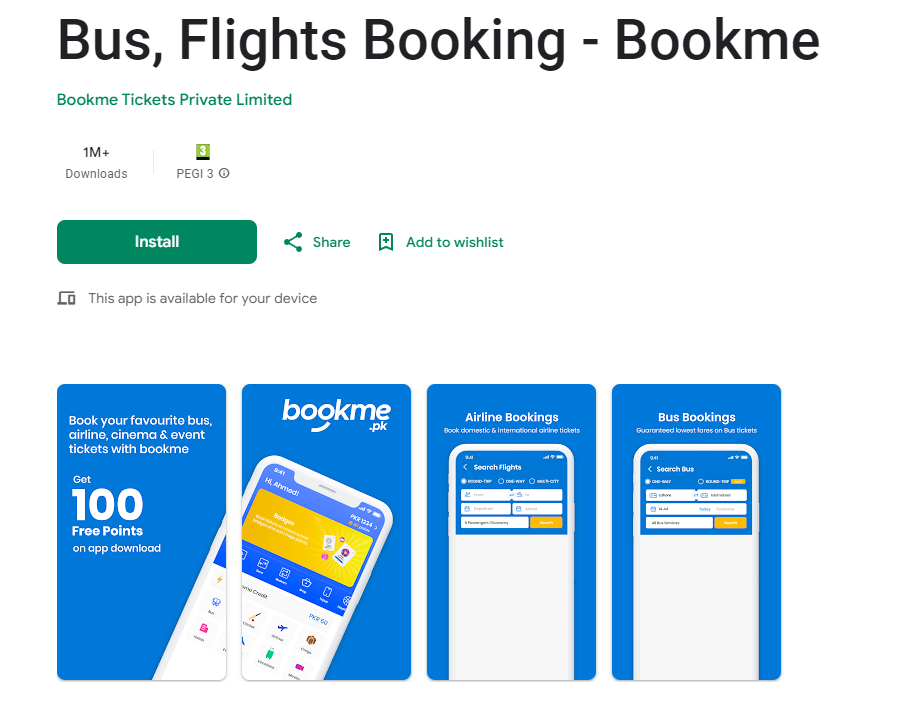


Рисунок 1.1 – BookMe

1. Функциональность:

Гибкость бронирования: Пользователи могут бронировать места на определенные дни и время, выбирая из доступных опций.

Уведомления и напоминания: Приложение отправляет уведомления о предстоящих бронированиях и напоминания, чтобы пользователи не забывали о своих визитах.

Интеграция с календарем: Возможность интеграции с календарем пользователя для синхронизации всех бронирований.

1. Управление бронированиями:

История бронирований: Пользователи могут просматривать свои прошлые и предстоящие бронирования.

Отмена и изменение бронирований: Легкость в управлении бронированиями позволяет пользователям при необходимости быстро отменять или изменять свои резервации.

1. Дополнительные функции:

Отзывы и рейтинги: Пользователи могут оставлять отзывы о посещенных местах и читать отзывы других пользователей.

Поддержка клиентов: Быстрая и эффективная поддержка пользователей для решения возникающих проблем.

Минусы:

1. Стоимость:

Платные функции: Некоторые функции могут быть доступны только в премиум-версии приложения, что может потребовать дополнительных затрат.

1. Зависимость от интернета:

Онлайн-режим: Для использования приложения требуется постоянное интернет-соединение, что может быть неудобно в местах с плохим покрытием.

1. Ограниченная локализация:

Региональные ограничения: Приложение может быть недоступно или иметь ограниченную функциональность в некоторых регионах.

1. Конкуренция:

Рынок насыщен аналогами: Существуют и другие приложения с похожей функциональностью, что может затруднить выбор в пользу BookMe.

В целом, BookMe является удобным и функциональным инструментом для бронирования мест в компьютерных клубах и других заведениях, но важно учитывать его стоимость и необходимость интернет-соединения.

## **1.3 SeatMe**

SeatMe — это приложение, предназначенное для бронирования мест в различных заведениях, включая компьютерные клубы (рисунок 1.2).

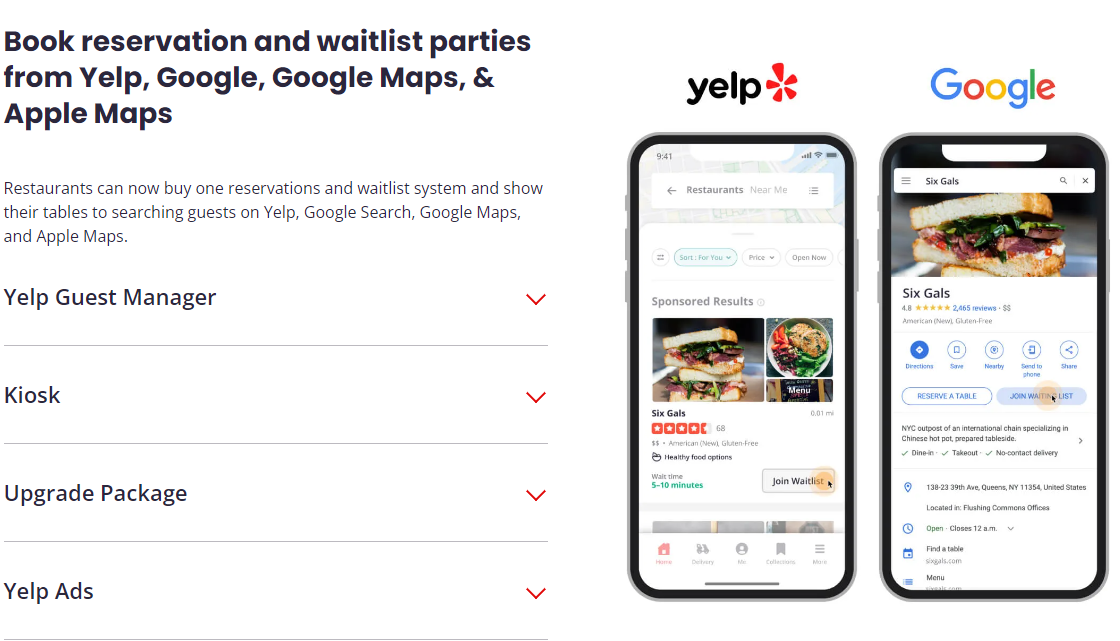


Рисунок 1.2 – SeatMe

Рассмотрим его основные плюсы и минусы. Плюсы:

1. Интерфейс и удобство использования:

Интуитивно понятный интерфейс: Простота и удобство использования позволяют пользователям быстро ориентироваться в приложении.

Доступность на различных платформах: Приложение доступно на iOS и Android, что делает его доступным для широкой аудитории.

1. Функциональность:

Гибкость бронирования: Возможность бронировать места на определённое время и дату, выбирая из доступных вариантов.

Настройки предпочтений: Пользователи могут указывать свои предпочтения, такие как любимые места или зоны в клубе.

1. Уведомления и напоминания:

Своевременные уведомления: Приложение отправляет уведомления о предстоящих бронированиях и напоминает пользователям о времени посещения.

1. Интеграция с календарем: Возможность синхронизации с личным календарем пользователя для удобного планирования.
2. Управление бронированиями:

История и управление бронированиями: Пользователи могут легко просматривать и управлять своими текущими и прошлыми бронированиями.

Возможность отмены и изменения: Простота в изменении или отмене бронирований по мере необходимости.

1. Дополнительные функции:

Отзывы и рейтинги: Функция оставления отзывов и чтения мнений других пользователей о заведениях.

Поддержка пользователей: Круглосуточная поддержка клиентов для решения возникающих вопросов.

Минусы:

1. Стоимость:

Подписка на премиум-функции: Некоторые функции могут быть доступны только по подписке, что может потребовать дополнительных затрат.

1. Зависимость от интернета:

Необходимость постоянного подключения: Для работы приложения требуется постоянное интернет-соединение, что может быть неудобно в местах с плохим покрытием.

1. Ограниченная локализация:

Региональные ограничения: В некоторых регионах функциональность приложения может быть ограничена или недоступна.

1. Конкуренция на рынке:

Наличие аналогов: Существует множество аналогичных приложений с похожими функциями, что может затруднить выбор пользователей в пользу SeatMe.

1. Сложности с интеграцией:

Интеграция с другими системами: В некоторых случаях могут возникать сложности при интеграции с внутренними системами управления бронированиями заведений.

Заключение:

SeatMe — это удобное и функциональное приложение для бронирования мест в компьютерных клубах и других заведениях. Оно предлагает широкий спектр функций и удобный интерфейс, но пользователям следует учитывать стоимость премиум-версии и необходимость постоянного интернет-соединения.

## **1.4 ClubReserve**

ClubReserve [3] — это специализированное приложение для бронирования мест в развлекательных заведениях, таких как компьютерные клубы, боулинг-центры, киберспортивные арены и другие (рисунок 1.3).

Плюсы:

1. Специализация:

Ориентация на развлекательные заведения: Приложение разработано специально для таких мест, как компьютерные клубы, что делает его особенно удобным для этих целей.

Поддержка различных типов заведений: Помимо компьютерных клубов, приложение поддерживает бронирование в других развлекательных центрах, что делает его универсальным.

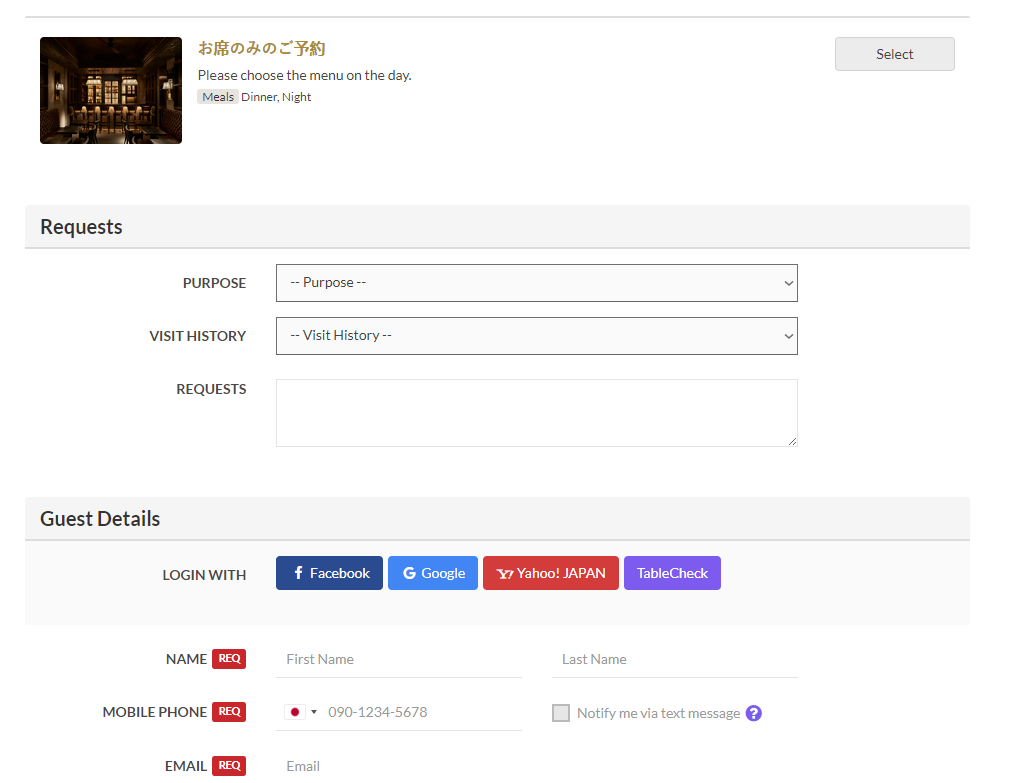


Рисунок 1.3 – ClubReserve

1. Функциональность:

Расширенные возможности бронирования: Возможность выбирать конкретные места или столы, а также бронировать их на определенное время.

Интеграция с системами управления: Приложение может интегрироваться с внутренними системами управления заведений, что обеспечивает точность и своевременность бронирований.

1. Удобство использования:

Интуитивный интерфейс: Простой и понятный дизайн, который позволяет легко ориентироваться и делать бронирования.

Мобильная доступность: Приложение доступно на платформах iOS и Android.

1. Уведомления и напоминания:

Своевременные уведомления: Приложение отправляет уведомления о предстоящих бронированиях и напоминает пользователям о времени посещения.

Интеграция с календарем: Возможность синхронизации с личным календарем пользователя для удобного планирования.

1. Управление бронированиями:

История и управление бронированиями: Пользователи могут легко просматривать и управлять своими текущими и прошлыми бронированиями.

Возможность отмены и изменения: Простота в изменении или отмене бронирований по мере необходимости.

Минусы:

1. Стоимость:

Платные функции: Некоторые расширенные функции могут быть доступны только по подписке или за дополнительную плату, что может потребовать дополнительных затрат.

1. Зависимость от интернета:

Необходимость постоянного подключения: Для работы приложения требуется постоянное интернет-соединение, что может быть неудобно в местах с плохим покрытием.

1. Ограниченная локализация:

Региональные ограничения: В некоторых регионах функциональность приложения может быть ограничена или недоступна.

4) Конкуренция на рынке:

Наличие аналогов: Существует множество аналогичных приложений с похожими функциями, что может затруднить выбор пользователей в пользу ClubReserve.

Заключение:

ClubReserve — это мощное и удобное приложение для бронирования мест в различных развлекательных заведениях, включая компьютерные клубы. Оно предлагает широкий спектр функций и специализированный подход к бронированиям в развлекательных центрах. Пользователи могут оценить его удобство и функциональность, но им следует учитывать возможные затраты на подписку и необходимость постоянного интернет-соединения.

## **1.5 PlaceBook**

PlaceBook [2] — это приложение для управления резервациями мест в различных заведениях, включая компьютерные клубы, рестораны, конференц-залы и другие (рисунок 1.4).

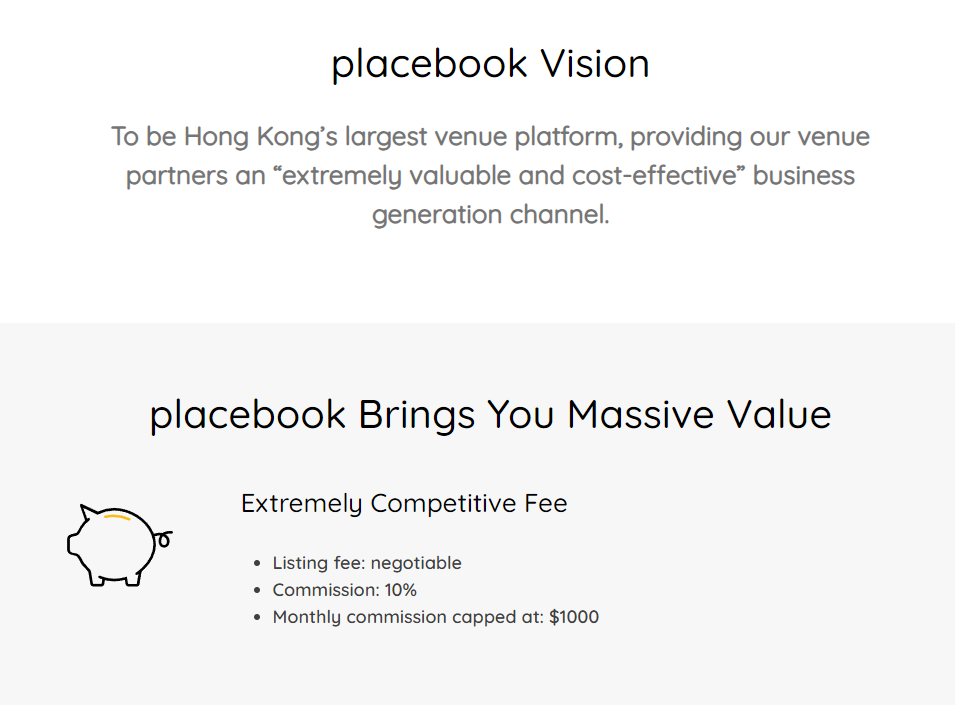


Рисунок 1.4 – PlaceBook

Рассмотрим его основные плюсы и минусы:

Плюсы:

1. Универсальность:

Широкий спектр заведений: Приложение подходит для бронирования в самых разных местах — от компьютерных клубов до ресторанов и конференц-залов.

Гибкость настроек: Возможность настройки под различные типы заведений делает его универсальным инструментом.

1. Функциональность:

Детализированные бронирования: Возможность выбора конкретных мест или столиков, а также бронирования на определенное время и дату.

Интеграция с календарями: Приложение можно синхронизировать с календарями пользователей для удобного планирования.

1. Удобство использования:

Интуитивный интерфейс: Простой и понятный интерфейс, который облегчает процесс бронирования.

Доступность на мобильных устройствах: Приложение доступно на платформах iOS и Android, что делает его удобным для использования на ходу.

1. Управление бронированиями:

История бронирований: Пользователи могут просматривать свои прошлые и текущие бронирования, что позволяет легко отслеживать посещения.

Изменение и отмена бронирований: Простота в изменении или отмене бронирований при необходимости.

1. Поддержка пользователей:

Круглосуточная поддержка: Приложение предлагает поддержку пользователей для решения возникающих вопросов.

Отзывы и рейтинги: Возможность оставлять отзывы и просматривать рейтинги других пользователей для принятия обоснованных решений.

Минусы:

1. Стоимость:

Платные функции: Некоторые функции и возможности могут быть доступны только в платной версии приложения, что требует дополнительных затрат.

1. Зависимость от интернета:

Необходимость постоянного подключения: Для работы приложения требуется постоянное интернет-соединение, что может быть неудобно в условиях слабого покрытия.

1. Ограниченная локализация:

Региональные ограничения: Приложение может иметь ограниченную функциональность или быть недоступным в некоторых регионах.

1. Конкуренция:

Рынок насыщен аналогами: Существует множество других приложений с похожими функциями, что может затруднить выбор в пользу PlaceBook.

Заключение:

PlaceBook — это универсальное и удобное приложение для бронирования мест в различных заведениях. Оно предлагает широкий спектр функций, включая детализированные бронирования, интеграцию с календарями и поддержку пользователей. Пользователям следует учитывать стоимость платных функций и необходимость постоянного интернет-соединения. Приложение отлично подходит для тех, кто часто пользуется услугами различных заведений и нуждается в удобном инструменте для управления своими бронированиями.

## **1.6 Вывод**

При выборе приложения для бронирования мест в компьютерных клубах и других заведениях, стоит учитывать множество факторов, таких как функциональность, удобство использования, стоимость и наличие поддержки. Рассмотрим краткий общий вывод по четырем упомянутым приложениям — BookMe, SeatMe, ClubReserve и PlaceBook.

1. BookMe:

Плюсы: Интуитивный интерфейс, гибкость бронирования, уведомления и напоминания, интеграция с календарями.

Минусы: Некоторые функции доступны только в премиум-версии, зависимость от интернета, ограниченная локализация.

1. SeatMe:

Плюсы: Простота использования, мобильная доступность, уведомления, интеграция с календарями, управление бронированиями.

Минусы: Платные функции, необходимость постоянного интернет-соединения, региональные ограничения.

1. ClubReserve:

Плюсы: Специализация на развлекательных заведениях, интеграция с системами управления, расширенные возможности бронирования.

Минусы: Платные функции, зависимость от интернета, наличие аналогов.

4)PlaceBook:

Плюсы: Универсальность, детализированные бронирования, интеграция с календарями, удобный интерфейс, поддержка пользователей.

Минусы: Платные функции, необходимость постоянного подключения к интернету, региональные ограничения.

Если вам нужна специализированная платформа для развлекательных заведений, таких как компьютерные клубы, ClubReserve может быть наилучшим выбором благодаря своей направленности и интеграции с системами управления.

Для универсальности и широкого спектра заведений, PlaceBook предоставляет отличные возможности бронирования и управления, подходящие для различных типов заведений.

Если важны простота и доступность, SeatMe предлагает удобный и интуитивный интерфейс с полезными функциями управления бронированиями и интеграцией с календарями.

Для тех, кто ценит гибкость и продвинутые уведомления, BookMe предоставляет удобные функции бронирования с хорошей интеграцией с календарями и напоминаниями.

Каждое из этих приложений имеет свои сильные и слабые стороны, и выбор конкретного зависит от ваших конкретных потребностей и предпочтений.

# Проектирование мобильного приложения и выбор технологий

## **Выбор платформы и технологий**

* + 1. Flutter

Flutter [4] — это фреймворк с открытым исходным кодом, разработанный Google, предназначенный для создания кросс-платформенных мобильных приложений, веб-приложений и настольных приложений. Вот основные моменты, которые стоит знать о Flutter:

Язык программирования: Flutter использует язык программирования Dart, также разработанный Google. Dart предоставляет высокую производительность и компиляцию в машинный код, что позволяет приложениям работать быстрее.

Основные компоненты:

Widgets: Основная строительная единица Flutter. Все, начиная от кнопок до текстовых полей, является виджетом. Flutter предлагает большое количество виджетов для создания интерфейсов.

Flutter Engine: Ядро фреймворка, написанное на C++, обеспечивает высокую производительность и поддержку различных платформ.

Кросс-платформенность: Одно из главных преимуществ Flutter — возможность создавать приложения, которые работают на различных платформах (iOS, Android, Web, Windows, macOS, Linux) с единой кодовой базой.

Hot Reload: Flutter поддерживает функцию Hot Reload, позволяющую разработчикам мгновенно видеть изменения в коде без необходимости перезапуска приложения, что значительно ускоряет процесс разработки.

Материальный дизайн и Cupertino: Flutter поддерживает два набора визуальных компонентов: Material Design, разработанный Google, и Cupertino, который соответствует стилю интерфейса iOS.

Большое сообщество и поддержка: Вокруг Flutter сформировалось большое сообщество разработчиков, что облегчает получение поддержки и доступ к многочисленным библиотекам и пакетам, упрощающим разработку.

Использование Flutter позволяет разработчикам создавать красивые, высокопроизводительные приложения с привлекательным пользовательским интерфейсом, минимизируя затраты на разработку и поддержку кода для различных платформ.

* + 1. Dart

Dart [5] — это объектно-ориентированный язык программирования, разработанный компанией Google. Он предназначен для разработки как серверных, так и клиентских приложений, особенно хорошо подходит для создания мобильных и веб-приложений с помощью фреймворка Flutter. Вот основные характеристики языка Dart:

Dart имеет C-подобный синтаксис, что делает его знакомым и удобным для разработчиков, знакомых с языками C, C++, Java и JavaScript. Он поддерживает как объектно-ориентированное, так и функциональное программирование.

Dart поддерживает статическую типизацию, что позволяет выявлять ошибки на этапе компиляции. Типы переменных могут быть указаны явно, но Dart также поддерживает вывод типов. Объектно-ориентированный: Все в Dart является объектом, включая числа, функции и null.

Богатая стандартная библиотека: Включает утилиты для коллекций, асинхронного программирования, работы с HTML и многого другого. Асинхронное программирование: Dart имеет встроенную поддержку асинхронного программирования с помощью Future и Stream, что упрощает работу с асинхронными операциями.

JIT и AOT компиляция: Dart поддерживает как JIT (Just-In-Time) компиляцию для быстрого запуска и отладки кода, так и AOT (Ahead-Of-Time) компиляцию для повышения производительности готовых приложений.

Dart широко используется для разработки кросс-платформенных мобильных приложений с помощью Flutter. Также он применяется для разработки веб-приложений и серверных приложений.

Основные компоненты синтаксиса:

Классы и объекты: Dart полностью поддерживает ООП с классами и наследованием.

Функции: Dart поддерживает как именованные, так и анонимные функции.

Коллекции: Включает списки, множества и карты (словари).

Генераторы: Dart поддерживает генераторы для удобного создания коллекций.

Dart имеет собственный пакетный менеджер, называемый pub, который используется для управления зависимостями и библиотеками. Поддержка во многих интегрированных средах разработки (IDE), таких как IntelliJ IDEA, Android Studio, Visual Studio Code.

Dart — это мощный и гибкий язык программирования, который предлагает высокую производительность и широкие возможности для разработки современных приложений.

* + 1. Widgets

В Flutter виджеты (Widgets) являются основными строительными блоками интерфейса пользователя. Практически все элементы интерфейса в приложении на Flutter являются виджетами.

Виджеты — это неизменяемые (immutable) компоненты, которые описывают часть пользовательского интерфейса. Каждый виджет описывает свою внешность и поведение.

Существует несколько типов виджетов.

StatelessWidget: Виджеты без состояния. Они описывают часть интерфейса, которая не меняется во времени. Примеры: Text, Icon.

StatefulWidget: Виджеты с состоянием. Они могут изменяться во времени в ответ на действия пользователя или другие события. Примеры: Checkbox, Slider.

Flutter поощряет использование композиции, что означает создание сложных интерфейсов путем объединения простых виджетов. Каждый виджет может содержать один или несколько дочерних виджетов.

Стоит рассказать и о иерархии виджетов. Виджеты образуют дерево, которое называется деревом виджетов (widget tree). Корневой виджет (root widget) является начальной точкой, и от него идут все остальные виджеты.

Благодаря Hot Reload разработчики могут быстро вносить изменения в код виджетов и мгновенно видеть результаты без необходимости перезапускать приложение.

Есть различные виджеты которые способны помочь в написании приложения.

Container: Один из самых универсальных виджетов, который может содержать другие виджеты и управлять их размером, отступами, границами и цветом фона.

Row и Column: Виджеты для размещения дочерних виджетов в горизонтальном (Row) или вертикальном (Column) порядке.

Text: Виджет для отображения текста.

Button: Различные виды кнопок, такие как RaisedButton, FlatButton, IconButton.

Темизация и стилизация: Flutter позволяет легко изменять внешний вид виджетов с помощью тем и стилей, что помогает создавать визуально привлекательные и единообразные интерфейсы. Виджеты в Flutter обеспечивают гибкость и мощь при создании пользовательских интерфейсов, позволяя разработчикам эффективно строить как простые, так и сложные приложения.

* + 1. Isar

Isar [6] (Instant Sync and Asynchronous Realtime) — это современная кроссплатформенная база данных, разработанная для мобильных приложений, особенно тех, которые построены с использованием Flutter и Dart. Isar ориентирована на высокую производительность и низкое потребление ресурсов, что делает её подходящей для использования на устройствах с ограниченными ресурсами, таких как мобильные телефоны (рисунок 2.1).

Основные особенности Isar:

1. Высокая производительность: Isar оптимизирована для быстрого доступа к данным и их обработки, обеспечивая минимальные задержки и высокую скорость чтения и записи данных.

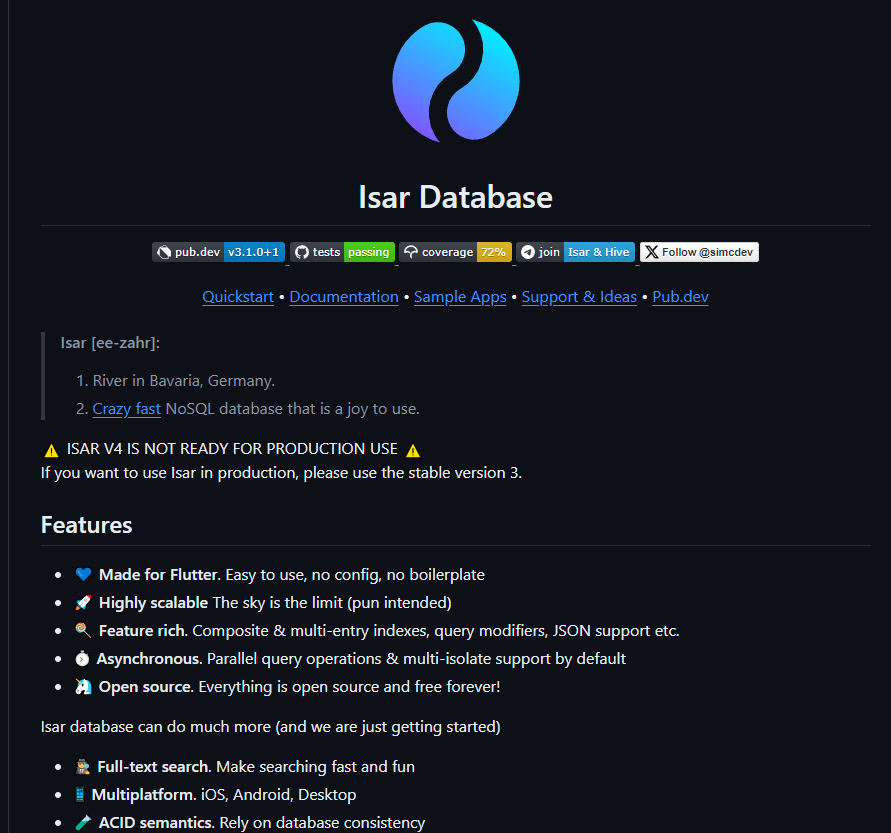


Рисунок 2.1 – Isar Database

1. Кроссплатформенность: Isar поддерживает несколько платформ, включая iOS, Android, и Web, что делает её универсальной для разработки современных приложений.
2. Легкость использования: Isar предоставляет удобный API для работы с базой данных, что упрощает процесс интеграции и использования в приложениях.
3. Асинхронная работа: Isar поддерживает асинхронные операции, что позволяет избежать блокировок в основном потоке и улучшает общую отзывчивость приложения.
4. Поддержка сложных запросов: Isar позволяет выполнять сложные запросы и манипуляции с данными, обеспечивая гибкость при работе с базой данных.
5. Низкое потребление ресурсов: Isar спроектирована таким образом, чтобы эффективно использовать ресурсы устройства, обеспечивая долгую работу приложения без значительного расхода батареи или памяти.

Isar предоставляет мощные инструменты для разработчиков Flutter и Dart, позволяя создавать высокопроизводительные приложения с минимальными затратами на управление данными.

2.1.5 Выбор IDE для разработки

Visual Studio Code (VS Code) — это бесплатная, кроссплатформенная интегрированная среда разработки (IDE), разработанная Microsoft, которая широко используется для разработки мобильных приложений. Она обладает множеством функций, делающих ее отличным инструментом для этой задачи. Вот основные возможности и преимущества VS Code для разработки мобильных приложений:

Расширения: VS Code поддерживает широкий спектр языков программирования через расширения, что позволяет разработчикам работать с языками Dart, JavaScript, TypeScript, Kotlin, Swift и другими.

Кроссплатформенность: работает на Windows, macOS и Linux, что обеспечивает гибкость и доступность для разработчиков на разных системах.

Интеграция с инструментами разработки мобильных приложений.

Flutter: с помощью расширения Flutter разработчики могут легко создавать, отлаживать и тестировать мобильные приложения на Flutter прямо в VS Code.

React Native: Поддержка React Native через соответствующие расширения позволяет разрабатывать приложения на этой популярной платформе.

Xamarin: Для разработки кроссплатформенных мобильных приложений на C# также доступны расширения.

VS Code предоставляет встроенные инструменты для отладки приложений, включая возможность установки точек останова, отслеживание выполнения кода, просмотр переменных и выполнение пошаговой отладки.

Специфичные для платформы расширения (например, для Flutter или React Native) добавляют дополнительные возможности для отладки мобильных приложений.

Интеграция с системами контроля версий: поддержка Git и других систем контроля версий позволяет разработчикам легко управлять изменениями кода, работать в команде и автоматизировать процессы развертывания.

Интеллектуальная автодополнение, подсветка синтаксиса и встроенные линтеры помогают писать код быстрее и с меньшим количеством ошибок.

Возможность настройки рабочего пространства, использование сниппетов кода и многочисленных плагинов для улучшения продуктивности.

Интеграция с CI/CD: поддержка DevOps процессов через интеграцию с такими инструментами как GitHub Actions, Azure Pipelines и другими, что позволяет автоматизировать тестирование и развертывание приложений.

## **Проектирование базы данных**

Для обеспечения управления приложением с использованием базы данных необходимо разработать удобный и интуитивно понятный интерфейс, который позволит пользователю взаимодействовать с базой данных и эффективно управлять данными. Это может включать в себя разработку оптимизированных запросов для вставки, обновления и удаления данных, а также разработку механизмов для извлечения и обработки информации из базы данных.

Диаграмма базы данных таблиц (Database Table Diagram) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных представлена на рисунке 2.2.

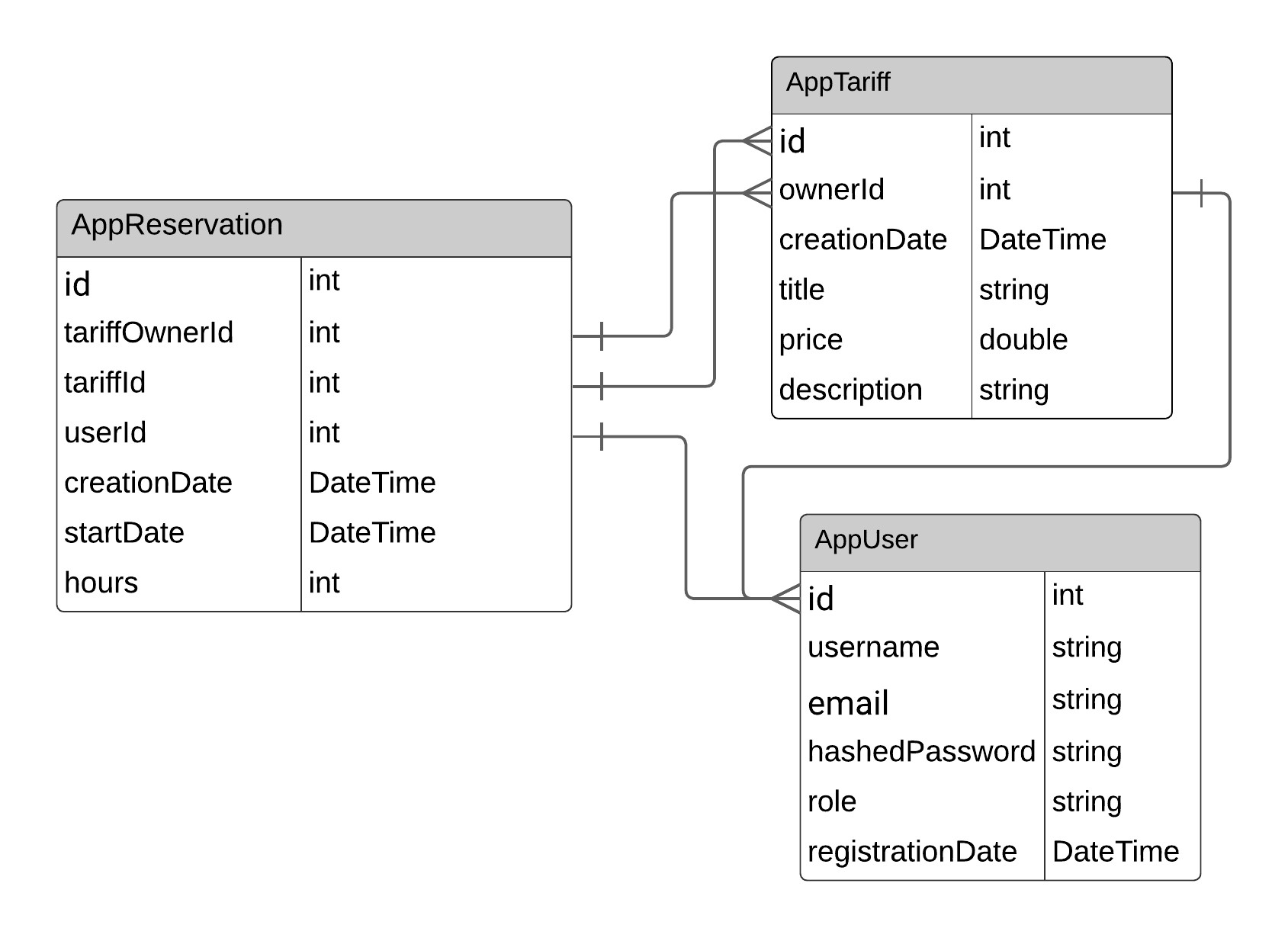


Рисунок 2.2 – Логическая схема базы данных

Таблица *AppUser* содержит словари. Перечень полей таблицы *AppUser* приведен в таблице 2.1.

Таблица 2.1 – Описание полей таблицы *AppUser*

|  |  |
| --- | --- |
| Поле | Типы данных, ограничения целостности |
| *id* | *Int, identity, primary key* |
| *username* | *string* |
| *email* | *string* |
| *hashedPassword* | *string* |
| *role* | *string* |
| *registrationDate* | *DateTime* |

Описание каждого поля таблицы *AppUser*:

* *id* – отвечает за идентификацию пользователя;
* *username* – имя пользователя;
* *email* – почта пользователя;
* *role* – роль пользователя в системе;
* *registrationDate* – дата регистрации пользователя;
* *hashedPassword* – пароль пользователя.

Таблица *AppTariff* содержит словари. Перечень полей таблицы *AppTariff* приведен в таблице 2.2.

Таблица 2.2 – Описание полей таблицы *AppTariff*

|  |  |
| --- | --- |
| Поле | Типы данных, ограничения целостности |
| *id* | *Int, identity, primary key* |
| *ownerId* | *Int* |
| *creationDate* | *DateTime* |
| *title* | *string* |
| *price* | *double* |
| *description* | *string* |

Описание каждого поля таблицы *AppUser*:

* *id* – отвечает за идентификацию тарифа;
* *ownerId* – идентификатор создателя тарифа;
* *creationDate* – время создания тарифа;
* *title* – название тарифа;
* *price* – стоймость 1 часа в тарифе;
* *description* – описание тарифа.

Таблица *AppReservation* содержит словари. Перечень полей таблицы *AppReservation* приведен в таблице 2.3.

Таблица 2.3 – Описание полей таблицы *AppReservation*

|  |  |
| --- | --- |
| Поле | Типы данных, ограничения целостности |
| *id* | *Int, identity, primary key* |
| *tariffOwnerId* | *Int* |
| *tariffId* | *Int* |
| *userId* | *Int* |
| *creationDate* | *DateTime* |
| *startDate* | *DateTime* |
| *hours* | *Int* |

Описание каждого поля таблицы *AppUser*:

* *id* – отвечает за идентификацию брони;
* *tariffOwnerId* – идентификатор создателя тарифа;
* *tariffId* – идентификатор тарифа;
* *userId* – идентификатор пользователя кто будет использовать бронь;
* *creationDate* – время создания брони;
* *startDate* – начало брони;
* *hours* – время действия брони.

База данных в виде сущностей представлена в приложении А

## **Проектирование структуры проекта**

Bloc [7] (Business Logic Component) – это паттерн управления состоянием, используемый в разработке приложений на Flutter. Он помогает отделить бизнес-логику приложения от пользовательского интерфейса и управлять состоянием приложения путем отправки событий в блок и обновления пользовательского интерфейса в соответствии с полученными данными.

Основные компоненты Bloc включают в себя:

События (Events): События представляют собой входные данные, которые инициируют изменение состояния в блоке.

Состояния (States): Состояния представляют собой данные, которые определяют текущее состояние приложения. Каждый блок может иметь различные состояния в зависимости от бизнес-логики.

Блок (Bloc): это класс, который принимает входные события, обрабатывает их и возвращает новые состояния. Он отвечает за управление бизнес-логикой и состоянием приложения.

Поток состояний (State Stream): Поток, который отправляет новые состояния из блока в пользовательский интерфейс для обновления.

Использование Bloc помогает упростить управление состоянием в приложениях Flutter, делает код более модульным и улучшает его тестируемость. Он часто применяется в средних и крупных приложениях, где сложная бизнес-логика требует эффективного управления состоянием.

## **Диаграмма вариантов использования**

Диаграмма вариантов использования показывает все возможные варианты использования приложения и взаимодействия пользователей с различными ролями.

В приложении присутствуют 2 основные роли:

* пользователь;
* администратор.

На рисунке 2.3 представлена диаграмма вариантов использования приложения.

Как видно из диаграммы использования функционал как админа так и пользователя включает в себя логин и регистрацию в приложении, изменение профиля бронирование и отмену бронирования.

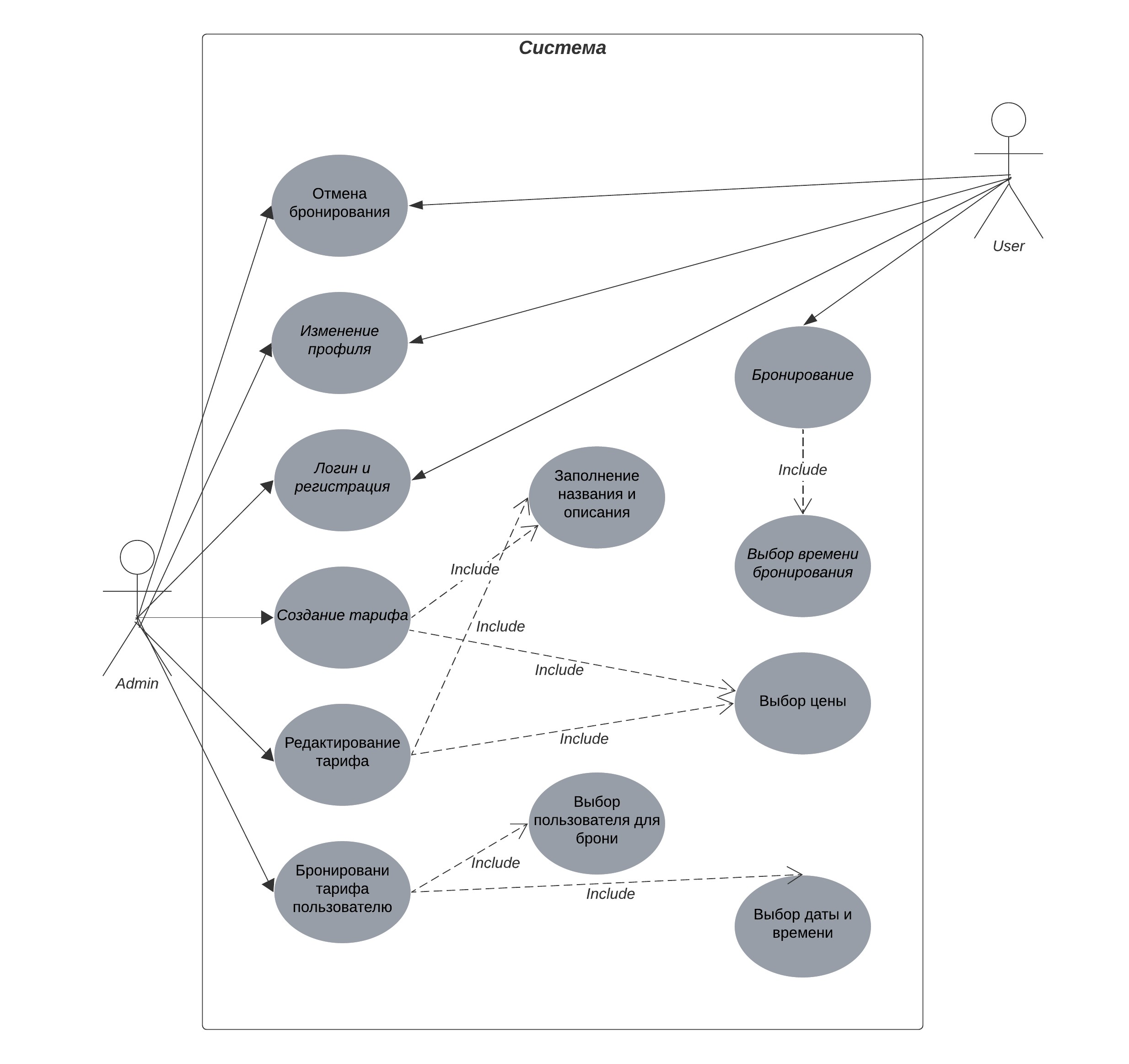


Рисунок 2.3 – Диаграмма вариантов использования

Администратор ко всему может дополнительно бронировать пользователю из своего аккаунта время для посещения, создавать и редактировать тарифы, а также отменять или завершать бронирование.

# Реализация приложения.

## **Разработка базы данных**

NoSQL базы данных - это класс баз данных, которые отличаются от реляционных баз данных (SQL) в том, что они не используют традиционные таблицы со строками и столбцами для хранения данных. Вместо этого они используют различные модели хранения данных, такие как документы, ключ-значение, столбцы или графы. Они часто применяются в случаях, когда необходимо обрабатывать большие объемы данных, требуется гибкая схема или высокая доступность.

ISAR (IndexedDB Storage, Advanced and Rapid) - это плагин для Flutter, который обеспечивает доступ к базе данных IndexedDB, используемой в веб-технологиях, таких как JavaScript. IndexedDB является NoSQL базой данных, используемой для хранения объектов JavaScript, и ISAR предоставляет аналогичную функциональность для приложений Flutter.

Таким образом, ISAR связан с NoSQL базами данных в том смысле, что он предоставляет возможность использовать IndexedDB в приложениях Flutter. Вместо использования SQL и реляционной модели данных, ISAR позволяет вам работать с NoSQL базой данных, что может быть полезно в различных сценариях, например, когда вам нужна гибкая схема данных или вы работаете с большими объемами данных.

Сущности связанные с базой в приложении представлены на рисунке 3.1

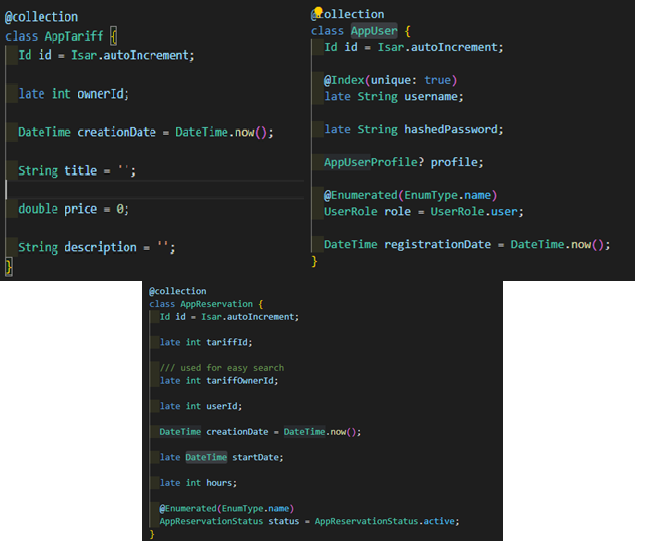


Рисунок 3.1 – Сущности связанные с базой

## **Разработка серверной части проекта**

Для управления состоянием использовался паттерн Bloc. Структура и элементы проекта связанные с данным пятерном представлены в приложении и указаны на рисунке 3.2.

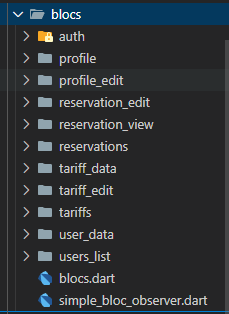


Рисунок 3.2 – Структура паттерна Bloc

Для обработки ошибок используется специальный роутер обработчик, который представлен на рисунке 3.3

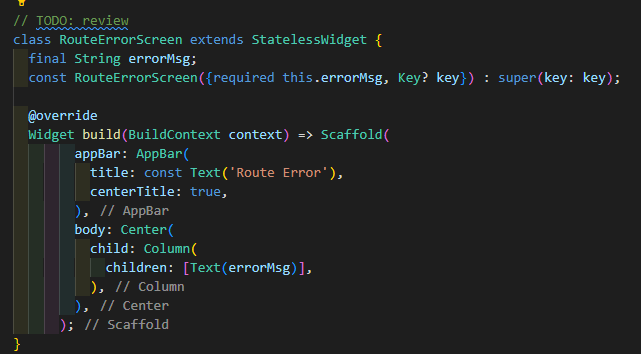


Рисунок 3.3 – Роутер обработчик

Все что связано с пользовательским интерфейсом представлено при помощи виджетов. Структура виджетов в проекте указана на рисунке 3.4

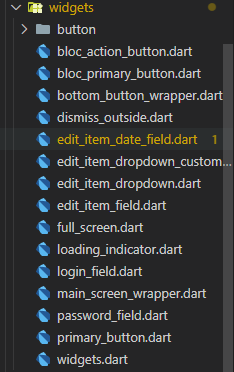


Рисунок 3.4 – Структура виджетов

Роутеры, которые являются связующим звеном между обработчиками и виджетами представлены на рисунке 3.5.

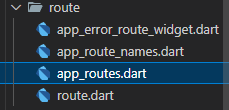


Рисунок 3.5 – Роутеры

Для смены темы в приложении приложения так же были созданы специальные конфиги, которые указаны на рисунке 3.6.

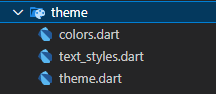


Рисунок 3.6 – Конфиги тем

Навигация в приложениях Flutter позволяет пользователям переходить между различными экранами или виджетами. Вот несколько основных методов навигации в Flutter:

Навигация на основе маршрутов (Route-based Navigation): Это классический способ навигации, когда каждый экран представлен своим собственным маршрутом. Flutter предоставляет класс Navigator, который управляет стеком маршрутов. Для перехода на новый экран вы можете использовать метод Navigator.push(), а для возврата на предыдущий экран - Navigator.pop().

Bottom Navigation Bar: Это виджет, расположенный в нижней части экрана, который позволяет переключаться между различными разделами вашего приложения. Вы можете использовать виджет BottomNavigationBar вместе с методом навигации на основе маршрутов для переключения между экранами.

TabBar и TabBarView: Эти виджеты позволяют пользователю переключаться между несколькими вкладками, каждая из которых содержит свое собственное содержимое. Вы можете использовать TabBar для отображения вкладок и TabBarView для отображения содержимого каждой вкладки. Каждая вкладка может быть связана с отдельным экраном приложения.

Drawer Navigation: Это выдвижное меню, которое позволяет пользователю получить доступ к различным разделам приложения. Для реализации выдвижного меню вы можете использовать виджет Drawer, который можно открыть с помощью жеста свайпа или кнопки меню.

Named Routes (Именованные маршруты): Этот подход позволяет определить имена для каждого маршрута в приложении. Это упрощает навигацию и облегчает понимание структуры приложения. Для использования именованных маршрутов вы можете определить их внутри объекта MaterialApp с помощью параметра routes.

Hero Animation: Это анимация, которая позволяет плавно перемещать объекты между разными экранами. Она обычно используется для создания впечатляющих эффектов при переходе между страницами с изображениями или виджетами.

Эти методы можно комбинировать и адаптировать под конкретные потребности вашего приложения. Например, вы можете использовать Bottom Navigation Bar для основной навигации между разделами приложения и Drawer для доступа к дополнительным функциям или настройкам.

В приложении используется навигация на основе маршрутов а файл навигации представлен на рисунке 3.7.

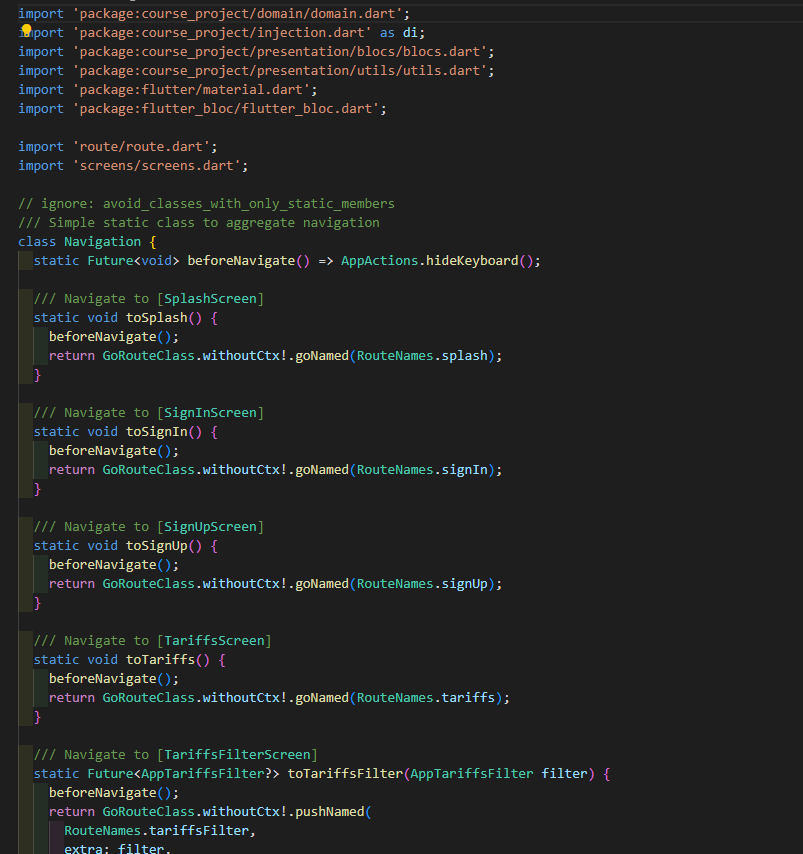


Рисунок 3.7 – Фаил навигации

Рисунок 3.8 представляет из себя репозитории для обработки данных связных с сущностями в проекте. Полный листинг кода навигатора представлен в приложении В.

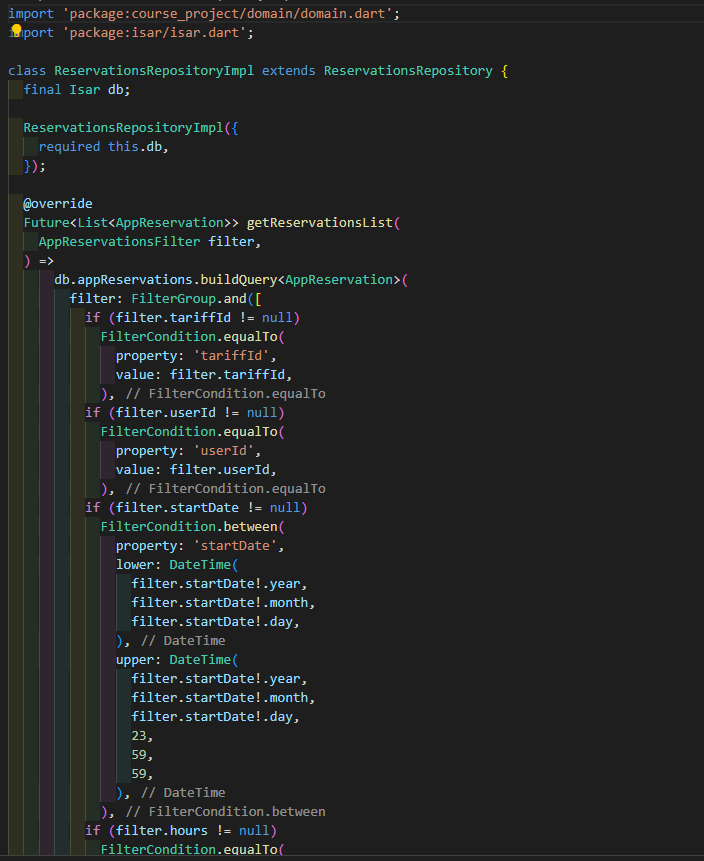


Рисунок 3.8 – Фаил репозитория

Данные репозитории являются связующим звеном для использования бизнес логии приложения. Листинг кода файлов репозитория указан в приложении Б.

# Тестирование приложения

В случае создания дополнительного пользователя с похожим именем (рисунок 4.1) должно происходить проверка. При которой выведется исключение что пользователь с таким именем уже присутствует.

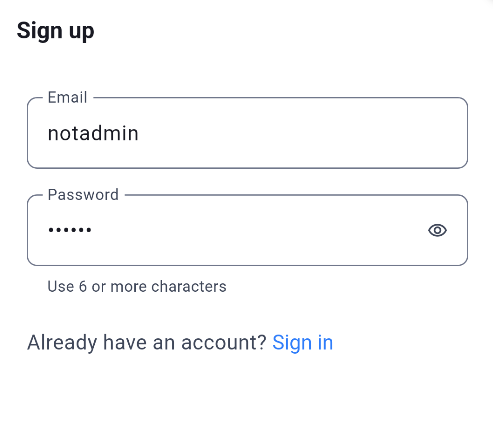


Рисунок 4.1 – Создание пользователя

Ра рисунке 4.2 мы как раз и видим уведомление о об данной ошибке.

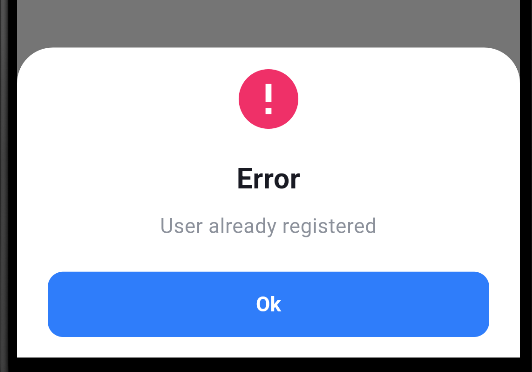


Рисунок 4.2 – Уведомление об ошибке

На рисунке 4.3 представлен интерфейс для бронирования. На нем должна корректно отображаться информация, связанная со временем резервации и количеством часов брони.

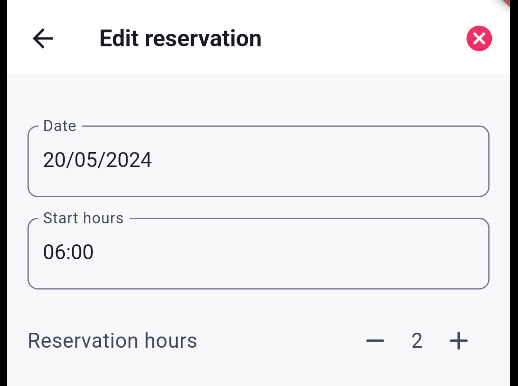


Рисунок 4.3 – Интерфейс бронирования

В случае создания нового тарифа и введении некорректной информации в поля должно выдаваться предупреждение о некорректно введенных данных (рисунок 4.4).

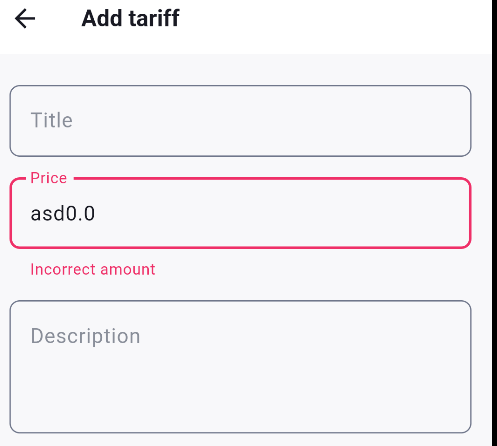


Рисунок 4.4 – Неверно введённые данные о цене

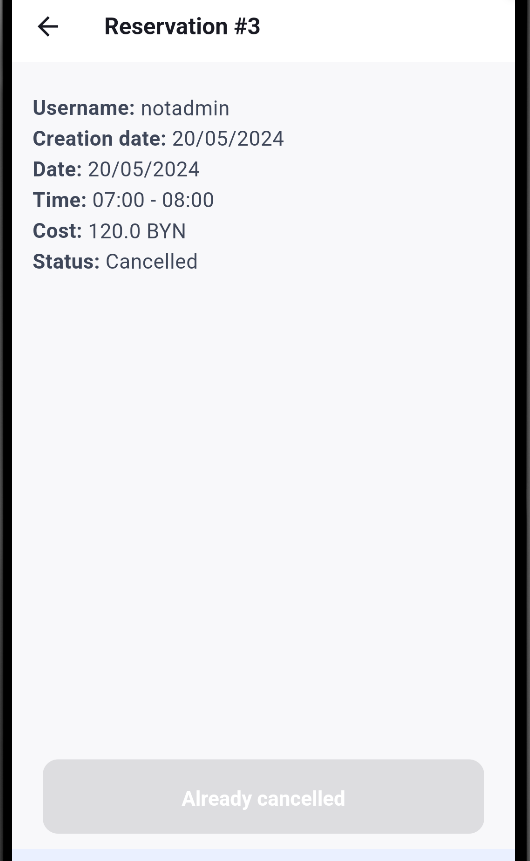


Рисунок 4.5 – Создание пользователя

На рисунке 4.5 представлено верное завышение или отмена резервации для пользователя администратором.

# Руководство пользователя

На рисунке 5.1 представлена минималистичная панель для выбора окон для взаимодействия с приложением. Представлены окна тарифов, бронирования, и пользовательских настроек.

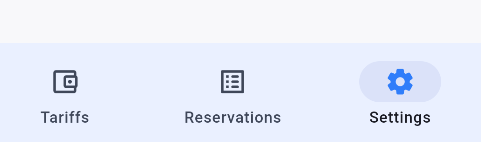


Рисунок 5.1 – Панель перехода между окнами

На рисунке 5.2 представлено окно для смены темы приложения.

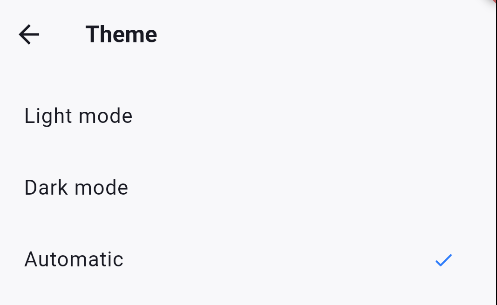


Рисунок 5.2 – Смена темы

На рисунке 5.3 представлены пользовательские настройки, которые позволят дополнить информацию о пользователе, выйти из системы и настроить тему приложения.

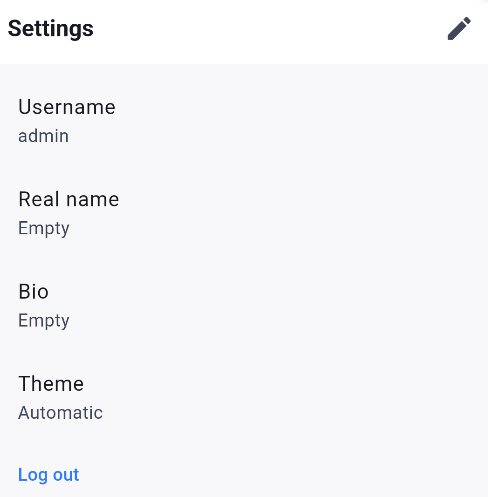


Рисунок 5.3 – Редактирование пользовательского интерфейса

На рисунке 5.4 представлена страница для выбора тарифа для бронирования.

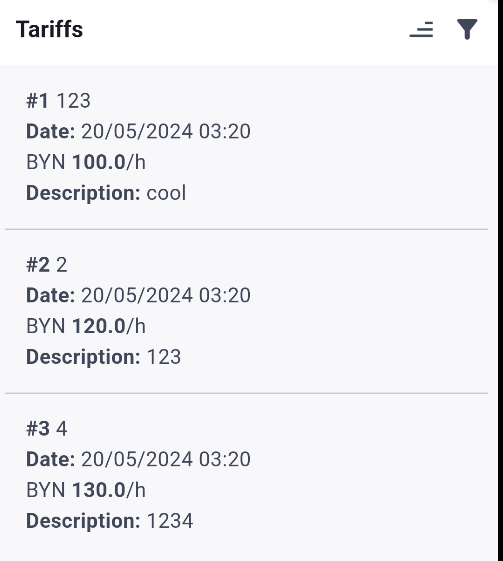


Рисунок 5.4 – Странница тарифов

На рисунке 5.5 приставлен фильтр для подбора наилучшего тарифа.

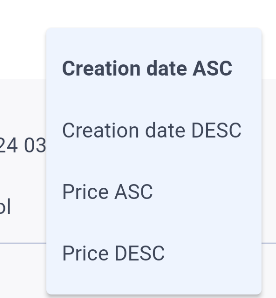


Рисунок 5.5 – Фильтр тарифов

На рисунке 5.6 представлено окно для создания пользовательского фильтра, который позволит наилучшим образом подобрать тариф пользователю.

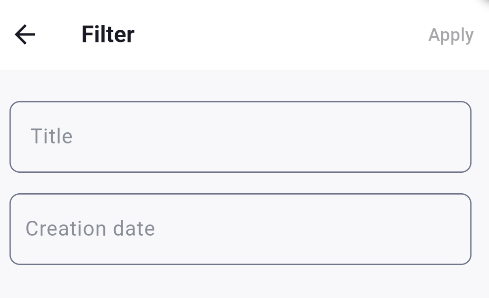


Рисунок 5.6 – Создание пользовательского фильтра

На рисунке 5.7 представлен вызов окна для создания нового тарифа. Данное окно доступно только пользователю с ролью администратора.

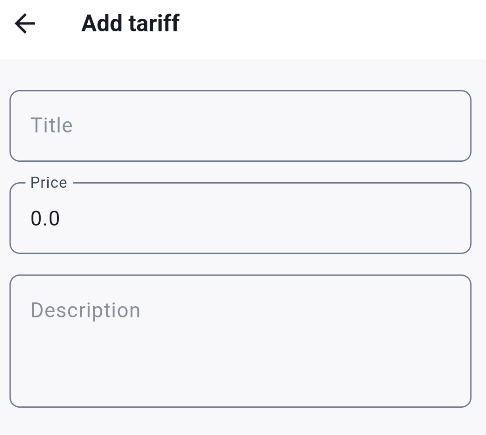


Рисунок 5.7 – Виджет создания тарифов

На рисунке 5.8 представлено окно с элементами бронирования. Данные элементы доступны пользователю, который совершил бронировании или админу для просмотра всех бронь которые присутствуют в базе.

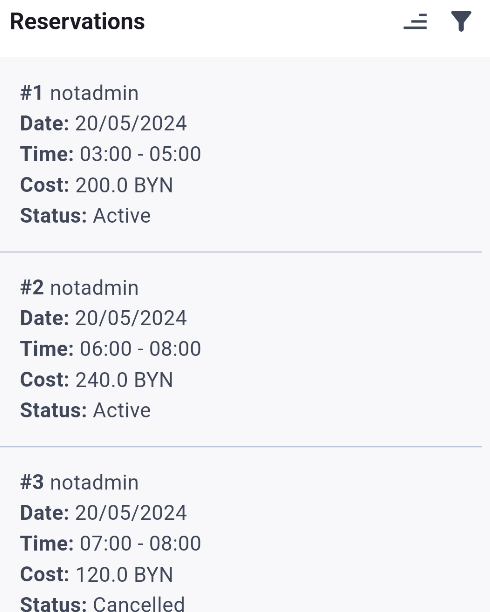


Рисунок 5.8 – Страница бронирования

На рисунке 5.9 представлено окно для завершения брони пользователя ли его отмены как самим пользователем, так и обычным юзером который регистрировал данную бронь.

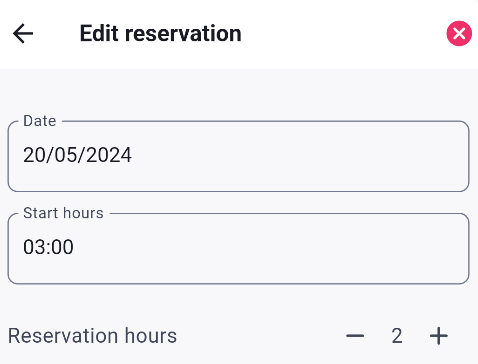


Рисунок 5.9 – Изменение бронирования или ее отмена

Для выбора даты был использован встроенный календарь, который позволяет улучшить пользовательский опыт (рисунок 5.10).

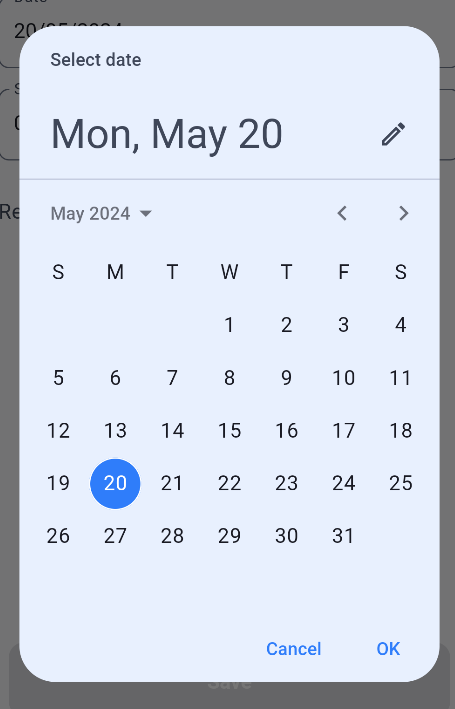


Рисунок 5.10 – Окно выбора даты бронирования

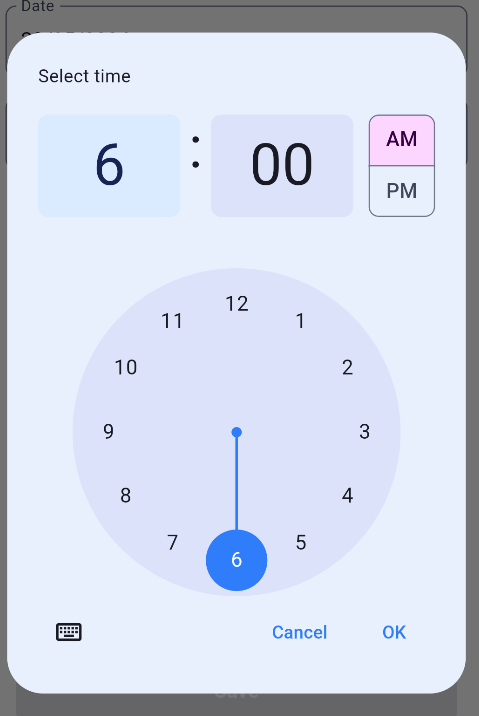


Рисунок 5.11 – Окно выбора времени бронирования

На рисунке 5.1 представлено окно выбора времени бронирования. Оно так же как и на рисунке 5.10 использует встроенный виджет часов для того чтобы не создавать пользовательские виджеты и улучшить качество приложения.

# Заключение

Перед началом разработки был произведен аналитический обзор прототипов приложений подобной тематики и определение функциональных возможностей разрабатываемого приложения.

В результате написания курсового проекта, было разработано мобильное приложение, использующее Flutter, как основной фреймворк для создания кроссплатформенных мобильных приложений. Flutter предоставляет все необходимое для разработки нативных приложений для мобильных устройств, веба и настольных платформ из единого кода. Dart, как язык программирования, используемый с Flutter. Dart оптимизирован для создания пользовательских интерфейсов и предлагает функции, такие как горячая перезагрузка, что ускоряет процесс разработки. Bloc: этот пакет используется для управления состоянием в приложении Flutter. Паттерн BLoC (Business Logic Component) помогает отделить бизнес-логику от пользовательского интерфейса, что делает приложение более поддерживаемым и тестируемым. В проекте используются зависимости, такие как bloc и bloc\_concurrency, которые помогают эффективно управлять состоянием и обработкой событий.

Isar: В проекте используется Isar, NoSQL база данных, разработанная для Flutter, для управления локальным хранилищем данных. Isar известна своей высокой производительностью и эффективностью, что подходит для мобильных приложений, обеспечивая быстрые операции чтения и записи. Также было написано руководство пользователя для веб-приложения.

При разработке выполнены следующие задачи:

* регистрация и автоматизация пользователя;
* добавление тарифа;
* изменение личной информации пользователя;
* сортировка, фильтрация и поиск тарифов.

Представлено описание каждого этапа проектирования и реализации мобильного приложения, которое позволяет упростить работу администраторов сетей компьютерных клубов, а также помочь пользователям корректно подобрать доступный для них тариф и время бронирования.

В пояснительной записке к курсовому проекту приведено описание руководства пользователя разработанного мобильного приложения.

В ходе тестирования приложения были выявлены и исправлены ошибки, что позволяет приложению работать бесперебойно.

# Список использованных источников

1. Страница мобильного прилежания Bookme [Электронный ресурс] – Режим доступа: [https://play.google.com/store/apps/details?id=com.macropaks. bookme&](https://play.google.com/store/apps/details?id=com.macropaks.%20bookme&)hl=en\_GB&pli=1/ – Дата доступа: 20.04.2024.
2. Главная страница сайта placebook [Электронный ресурс] – Режим доступа: https://placebook.hk/en/list-a-place/ – Дата доступа: 20.04.2024.
3. Главная страница сайта SG Club [Электронный ресурс] – Режим доступа: https://www.tablecheck.com/en/shops/the-sg-club/reserve/ – Дата доступа: 20.04.2024.
4. Документация Flutter [Электронный ресурс] – Режим доступа: https://docs.flutter.dev// – Дата доступа: 01.05.2024.
5. Dart [Электронный ресурс] – Режим доступа: https://dart.dev/guides/ – Дата доступа: 27.04.2024.
6. Документация Isar [Электронный ресурс] – Режим доступа: https://github.com/isar/isar– Дата доступа: 27.04.2024.
7. Документация паттерна Bloc [Электронный ресурс] – Режим доступа: https://bloclibrary.dev/ / – Дата доступа: 05.05.2024.
8. Код приложения [Электронный ресурс] – Режим доступа: / – Дата доступа: 20.05.2024.

# Приложение А

Листинг сущностей для Isac

import 'package:isar/isar.dart';

part 'app\_user.g.dart';

@collection

class AppUser {

  Id id = Isar.autoIncrement;

  @Index(unique: true)

  late String username;

  late String hashedPassword;

  AppUserProfile? profile;

  @Enumerated(EnumType.name)

  UserRole role = UserRole.user;

  DateTime registrationDate = DateTime.now();

}

/// available messengers

enum UserRole {

  admin,

  user,

}

@embedded

class AppUserProfile {

  String? realName;

  String? bio;

}

import 'package:freezed\_annotation/freezed\_annotation.dart';

import 'package:isar/isar.dart';

part 'app\_tariff.freezed.dart';

part 'app\_tariff.g.dart';

@collection

class AppTariff {

  Id id = Isar.autoIncrement;

  late int ownerId;

  DateTime creationDate = DateTime.now();

  String title = '';

  double price = 0;

  String description = '';

}

@freezed

class AppTariffEntity with \_$AppTariffEntity {

  const factory AppTariffEntity({

    required int id,

    required int ownerId,

    required DateTime creationDate,

    required String title,

    required double price,

    required String description,

  }) = \_AppTariffEntity;

  factory AppTariffEntity.empty({

    required int ownerId,

  }) =>

      AppTariffEntity(

        id: -1,

        ownerId: ownerId,

        creationDate: DateTime.now(),

        title: '',

        price: 0,

        description: '',

      );

  factory AppTariffEntity.fromJson(Map<String, dynamic> json) =>

      \_$AppTariffEntityFromJson(json);

}

extension AppTariffMapperExtension on AppTariff {

  AppTariffEntity get entity => AppTariffEntity(

        id: id,

        ownerId: ownerId,

        creationDate: creationDate,

        title: title,

        price: price,

        description: description,

      );

}

extension AppTariffEntityMapperExtension on AppTariffEntity {

  AppTariff get dto => AppTariff()

    ..id = id == -1 ? Isar.autoIncrement : id

    ..creationDate = creationDate

    ..ownerId = ownerId

    ..title = title

    ..price = price

    ..description = description;

}

import 'package:freezed\_annotation/freezed\_annotation.dart';

import 'package:isar/isar.dart';

part 'app\_reservation.freezed.dart';

part 'app\_reservation.g.dart';

@collection

class AppReservation {

  Id id = Isar.autoIncrement;

  late int tariffId;

  /// used for easy search

  late int tariffOwnerId;

  late int userId;

  DateTime creationDate = DateTime.now();

  late DateTime startDate;

  late int hours;

  @Enumerated(EnumType.name)

  AppReservationStatus status = AppReservationStatus.active;

}

enum AppReservationStatus {

  active,

  cancelled,

}

@freezed

class AppReservationEntity with \_$AppReservationEntity {

  const factory AppReservationEntity({

    required int id,

    required int tariffId,

    required int tariffOwnerId,

    required int userId,

    required DateTime creationDate,

    required DateTime startDate,

    required int hours,

    required AppReservationStatus status,

  }) = \_AppReservationEntity;

  factory AppReservationEntity.empty({

    required int tariffId,

    required int tariffOwnerId,

    required int userId,

  }) {

    final now = DateTime.now();

    return AppReservationEntity(

      id: -1,

      tariffId: tariffId,

      tariffOwnerId: tariffOwnerId,

      userId: userId,

      creationDate: DateTime.now(),

      startDate: DateTime(now.year, now.month, now.day, now.hour),

      hours: 0,

      status: AppReservationStatus.active,

    );

  }

  factory AppReservationEntity.fromJson(Map<String, dynamic> json) =>

      \_$AppReservationEntityFromJson(json);

}

extension AppReservationMapperExtension on AppReservation {

  AppReservationEntity get entity => AppReservationEntity(

        id: id,

        tariffId: tariffId,

        tariffOwnerId: tariffOwnerId,

        userId: userId,

        creationDate: creationDate,

        startDate: startDate,

        hours: hours,

        status: status,

      );

}

extension AppReservationEntityMapperExtension on AppReservationEntity {

  AppReservation get dto => AppReservation()

    ..id = id == -1 ? Isar.autoIncrement : id

    ..tariffId = tariffId

    ..tariffOwnerId = tariffOwnerId

    ..userId = userId

    ..creationDate = creationDate

    ..startDate = startDate

    ..hours = hours

    ..status = status;

}

# Приложение Б

Листинг файлов репозиториев для бизнес-логики

import 'package:freezed\_annotation/freezed\_annotation.dart';

import 'package:isar/isar.dart';

part 'app\_reservation.freezed.dart';

part 'app\_reservation.g.dart';

@collection

class AppReservation {

Id id = Isar.autoIncrement;

late int tariffId;

/// used for easy search

late int tariffOwnerId;

late int userId;

DateTime creationDate = DateTime.now();

late DateTime startDate;

late int hours;

@Enumerated(EnumType.name)

AppReservationStatus status = AppReservationStatus.active;

}

enum AppReservationStatus {

active,

cancelled,

}

@freezed

class AppReservationEntity with \_$AppReservationEntity {

const factory AppReservationEntity({

required int id,

required int tariffId,

required int tariffOwnerId,

required int userId,

required DateTime creationDate,

required DateTime startDate,

required int hours,

required AppReservationStatus status,

}) = \_AppReservationEntity;

factory AppReservationEntity.empty({

required int tariffId,

required int tariffOwnerId,

required int userId,

}) {

final now = DateTime.now();

return AppReservationEntity(

id: -1,

tariffId: tariffId,

tariffOwnerId: tariffOwnerId,

userId: userId,

creationDate: DateTime.now(),

startDate: DateTime(now.year, now.month, now.day, now.hour),

hours: 0,

status: AppReservationStatus.active,

);

}

factory AppReservationEntity.fromJson(Map<String, dynamic> json) =>

\_$AppReservationEntityFromJson(json);

}

extension AppReservationMapperExtension on AppReservation {

AppReservationEntity get entity => AppReservationEntity(

id: id,

tariffId: tariffId,

tariffOwnerId: tariffOwnerId,

userId: userId,

creationDate: creationDate,

startDate: startDate,

hours: hours,

status: status,

);

}

extension AppReservationEntityMapperExtension on AppReservationEntity {

AppReservation get dto => AppReservation()

..id = id == -1 ? Isar.autoIncrement : id

..tariffId = tariffId

..tariffOwnerId = tariffOwnerId

..userId = userId

..creationDate = creationDate

..startDate = startDate

..hours = hours

..status = status;

}

import 'package:course\_project/domain/domain.dart';

import 'package:dartx/dartx.dart';

import 'package:isar/isar.dart';

class TariffsRepositoryImpl extends TariffsRepository {

final Isar db;

TariffsRepositoryImpl({

required this.db,

});

@override

Future<List<AppTariff>> getTariffsList(AppTariffsFilter filter) =>

db.appTariffs.buildQuery<AppTariff>(

filter: FilterGroup.and([

if (filter.ownerId != null)

FilterCondition.equalTo(

property: 'ownerId',

value: filter.ownerId,

),

if (filter.titleQuery.isNotNullOrBlank)

FilterCondition.contains(

property: 'title',

value: filter.titleQuery!,

caseSensitive: true,

),

if (filter.creationDate != null)

FilterCondition.between(

property: 'creationDate',

lower: DateTime(

filter.creationDate!.year,

filter.creationDate!.month,

filter.creationDate!.day,

),

upper: DateTime(

filter.creationDate!.year,

filter.creationDate!.month,

filter.creationDate!.day,

23,

59,

59,

),

),

]),

sortBy: [

SortProperty(

property: filter.sort.name,

sort: filter.asc ? Sort.asc : Sort.desc,

)

],

).findAll();

@override

Future<AppTariff?> getTariff(int id) =>

db.appTariffs.filter().idEqualTo(id).findFirst();

@override

Future<void> putTariff(AppTariff data) async {

await db.writeTxn(() async {

await db.appTariffs.put(data);

});

return;

}

@override

Future<void> deleteTariff(int id) async {

await db.writeTxn(() async {

await db.appTariffs.delete(id);

});

return;

}

}

import 'package:course\_project/domain/domain.dart';

import 'package:dartx/dartx.dart';

import 'package:either\_dart/either.dart';

import 'package:isar/isar.dart';

class UsersRepositoryImpl extends UsersRepository {

final Isar db;

UsersRepositoryImpl({

required this.db,

});

/// return user entity if success or failure on error

@override

Future<Either<SignInFailure, AppUser>> signIn({

required String username,

required String hashedPassword,

}) async {

final user =

await db.appUsers.where().usernameEqualTo(username).build().findFirst();

if (user == null) {

return Left(SignInFailure.userNotFound());

}

if (user.hashedPassword != hashedPassword) {

return Left(SignInFailure.wrongPassword());

}

return Right(user);

}

/// return user entity if success or failure on error

@override

Future<Either<SignUpFailure, AppUser>> signUp({

required String username,

required String hashedPassword,

}) async {

final user =

await db.appUsers.where().usernameEqualTo(username).build().findFirst();

if (user != null) {

return Left(SignUpFailure.userAlreadyRegistered());

}

final newUser = AppUser()

..username = username

..hashedPassword = hashedPassword;

await db.writeTxn(() async {

await db.appUsers.put(newUser);

});

return Right(newUser);

}

@override

Future<List<AppUser>> getUsersList(AppUsersFilter filter) =>

db.appUsers.buildQuery<AppUser>(

filter: FilterGroup.and([

if (filter.usernameQuery.isNotNullOrBlank)

FilterCondition.contains(

property: 'username',

value: filter.usernameQuery!,

caseSensitive: false,

),

if (filter.registrationDate != null)

FilterCondition.equalTo(

property: 'registrationDate',

value: filter.registrationDate,

),

const FilterCondition.equalTo(

property: 'role',

value: UserRole.user,

),

]),

sortBy: [

SortProperty(

property: filter.sort.name,

sort: filter.asc ? Sort.asc : Sort.desc,

)

],

).findAll();

@override

Future<AppUser?> getUser(int id) =>

db.appUsers.filter().idEqualTo(id).findFirst();

@override

Future<AppUser?> getUserByName(String username) =>

db.appUsers.filter().usernameEqualTo(username).findFirst();

@override

Future<void> updateProfile({

required int userId,

required AppUserProfile profile,

}) async {

final user = await getUser(userId);

if (user == null) return;

await db.writeTxn(() async {

await db.appUsers.put(user..profile = profile);

});

return;

}

}

# Приложение В

Листинг навигатора

import 'package:course\_project/domain/domain.dart';

import 'package:course\_project/injection.dart' as di;

import 'package:course\_project/presentation/blocs/blocs.dart';

import 'package:course\_project/presentation/utils/utils.dart';

import 'package:flutter/material.dart';

import 'package:flutter\_bloc/flutter\_bloc.dart';

import 'route/route.dart';

import 'screens/screens.dart';

// ignore: avoid\_classes\_with\_only\_static\_members

/// Simple static class to aggregate navigation

class Navigation {

static Future<void> beforeNavigate() => AppActions.hideKeyboard();

/// Navigate to [SplashScreen]

static void toSplash() {

beforeNavigate();

return GoRouteClass.withoutCtx!.goNamed(RouteNames.splash);

}

/// Navigate to [SignInScreen]

static void toSignIn() {

beforeNavigate();

return GoRouteClass.withoutCtx!.goNamed(RouteNames.signIn);

}

/// Navigate to [SignUpScreen]

static void toSignUp() {

beforeNavigate();

return GoRouteClass.withoutCtx!.goNamed(RouteNames.signUp);

}

/// Navigate to [TariffsScreen]

static void toTariffs() {

beforeNavigate();

return GoRouteClass.withoutCtx!.goNamed(RouteNames.tariffs);

}

/// Navigate to [TariffsFilterScreen]

static Future<AppTariffsFilter?> toTariffsFilter(AppTariffsFilter filter) {

beforeNavigate();

return GoRouteClass.withoutCtx!.pushNamed(

RouteNames.tariffsFilter,

extra: filter,

);

}

/// Navigate to [TariffEditScreen]

static void toTariffEdit({AppTariff? data}) {

beforeNavigate();

return GoRouteClass.withoutCtx!.goNamed(

RouteNames.tariffEdit,

extra: data?.entity,

);

}

/// Navigate to [ReservationsScreen]

static void toReservations() {

beforeNavigate();

return GoRouteClass.withoutCtx!.goNamed(RouteNames.reservations);

}

/// Navigate to [ReservationsFilterScreen]

static Future<AppReservationsFilter?> toReservationsFilter(

AppReservationsFilter filter) {

beforeNavigate();

return GoRouteClass.withoutCtx!.pushNamed(

RouteNames.reservationsFilter,

extra: filter,

);

}

/// Navigate to [ReservationEditScreen]

static Future<void> toReservationEdit({

AppReservation? data,

int? tariffId,

int? tariffOwnerId,

int? userId,

}) {

beforeNavigate();

return GoRouteClass.withoutCtx!.pushNamed(

RouteNames.reservationEdit,

extra: data?.entity,

queryParameters: {

'tariffId': tariffId?.toString(),

'tariffOwnerId': tariffOwnerId?.toString(),

if (userId != null) 'userId': userId.toString(),

},

);

}

/// Navigate to [ReservationViewScreen]

static void toReservationView(AppReservation data) {

beforeNavigate();

return GoRouteClass.withoutCtx!.goNamed(

RouteNames.reservationView,

extra: data.entity,

);

}

/// Navigate to [SettingsScreen]

static void toSettings() {

beforeNavigate();

return GoRouteClass.withoutCtx!.goNamed(RouteNames.settings);

}

/// Navigate to [ProfileEditScreen]

static Future<void> toProfileEdit() {

beforeNavigate();

return GoRouteClass.withoutCtx!.pushNamed(RouteNames.profileEdit);

}

/// Navigate to [ThemeSelectScreen]

static Future<void> toThemeSelect() {

beforeNavigate();

return Navigator.of(GoRouteClass.withoutCtx!).push(

MaterialPageRoute(

builder: (context) => ThemeSelectScreen(

settingsController: di.locator(),

),

),

);

}

/// Navigate to [UserSelectScreen]

static Future<AppUser?> toUserSelect({

int? selectedId,

}) {

beforeNavigate();

return Navigator.of(GoRouteClass.withoutCtx!).push(

MaterialPageRoute(

builder: (context) => BlocProvider<UsersListCubit>(

create: (context) => UsersListCubit(

usersRepository: di.locator(),

),

child: UserSelectScreen(

selectedId: selectedId,

),

),

),

);

}

}