**ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**«ТОМСКИЙ ЭКОНОМИКО-ПРОМЫШЛЕННЫЙ КОЛЛЕДЖ»**

**ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА REST API СЕРВЕРА НА ФРЕЙМВОРКЕ LARAVEL И МОБИЛЬНОГО ПРИЛОЖЕНИЯ САМОКАТ ПРО ДЛЯ АДМИНИСТРАТОРОВ**

***КУРСОВОЙ ПРОЕКТ***

*СПЕЦИАЛЬНОСТЬ 09.02.07 ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ПРОГРАММИРОВАНИЕ*

Выполнил:

Студенты группы 1912с

\_\_\_\_\_\_\_\_\_\_\_\_ / Рюгин А.И.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

Проверил:

Преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_ / Уляхин В.А.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc190550054)

[ТЕХНОЛОГИИ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ 6](#_Toc190550055)

[Используемые языки программирования и фреймворки 6](#_Toc190550056)

[Используемое программное обеспечение 9](#_Toc190550057)

[ГЛАВА 1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ 10](#_Toc190550058)

[ИНФОРМАЦИОННОГО ПРОДУКТА. 10](#_Toc190550059)

[Описание проекта и задач 10](#_Toc190550060)

[Специфические требования к API 12](#_Toc190550061)

[Требования к мобильному приложению 63](#_Toc190550062)

[ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БАЗЫ ДАННЫХ 83](#_Toc190550063)

[Нормализация данных 83](#_Toc190550064)

[Концептуальная модель данных 85](#_Toc190550065)

[Логическая модель данных 86](#_Toc190550066)

[Физическая модель данных 87](#_Toc190550067)

[Диаграмма базы данных 88](#_Toc190550068)

[ГЛАВА 3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ФУНКЦИОНАЛА 89](#_Toc190550069)

[ИНФОРМАЦИОННОГО ПРОДУКТА 89](#_Toc190550070)

[Установка и настройка фреймворка Laravel11 89](#_Toc190550071)

[Создание миграций 90](#_Toc190550072)

[Создание исключений 92](#_Toc190550073)

[Создание моделей 93](#_Toc190550074)

[Создание контроллеров 96](#_Toc190550075)

[Валидация запроса формы 105](#_Toc190550076)

[Маршрутизация 111](#_Toc190550077)

[ГЛАВА 4. АВТОТЕСТИРОВАНИЕ API 112](#_Toc190550078)

[401 Unauthorized 112](#_Toc190550079)

[403 Forbidden 113](#_Toc190550080)

[404 Not Found 113](#_Toc190550081)

[200 Ok 114](#_Toc190550082)

[201 Created 115](#_Toc190550083)

[ГЛАВА 5. РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ 116](#_Toc190550084)

[Введение 116](#_Toc190550085)

[Структура проекта 117](#_Toc190550086)

[Конвертеры 118](#_Toc190550087)

[Модели 121](#_Toc190550088)

[Реализация страниц 127](#_Toc190550089)

# ВВЕДЕНИЕ

Актуальность проекта:

В условиях высокой конкуренции на рынке доставки и растущих ожиданий клиентов, эффективность работы сотрудников становится критически важной для успеха компании. Современные технологии позволяют оптимизировать внутренние процессы, улучшить взаимодействие между отделами и повысить производительность персонала.

Разработка специализированного приложения для сотрудников «Самокат Про» актуальна по следующим причинам:

* Удобство для сотрудников: Администраторы получают единый инструмент для управления своими задачами, что упрощает их работу и повышает удовлетворённость от трудового процесса.
* Повышение прозрачности и контроля: Приложение предоставляет возможность отслеживать статус заказов, анализировать производительность сотрудников и оперативно решать возникающие проблемы. Это способствует улучшению качества сервиса.
* Снижение нагрузки на менеджмент: Автоматизация процессов снижает нагрузку на управленческий состав, позволяя сосредоточиться на стратегических задачах, а не на ручном управлении курьерами.
* Адаптация к масштабированию: С ростом компании и увеличением числа заказов приложение становится необходимым инструментом для поддержания высокой скорости и качества работы. Оно позволяет легко масштабировать процессы без потери эффективности.
* Конкурентоспособность: Наличие собственного приложения для сотрудников демонстрирует технологическую продвинутость компании, что помогает привлекать и удерживать квалифицированный персонал.
* Соответствие современным трендам: Внедрение цифровых решений для управления персоналом соответствует общемировому тренду цифровизации бизнеса и повышает гибкость компании в условиях меняющегося рынка.

Целью разработки приложения «Самокат Про» для администраторов является оптимизация рабочих процессов сотрудников компании за счёт внедрения цифрового инструмента, который автоматизирует задачи, повышает прозрачность выполнения заказов и улучшает взаимодействие между отделами. API должно быть разработано с использованием современных технологий и соответствовать стандартам безопасности. Мобильное приложение должно быть разработано для платформ Android.

Задачи:

1. Разработка пользовательского интерфейса мобильного приложения, обеспечивающего простоту и интуитивность использования для администраторов.
2. Создание API, обеспечивающего взаимодействие мобильного приложения с серверной частью системы.
3. Разработка системы подтверждения доступностей администраторами.
4. Интеграция системы мониторинга и управления курьерскими доступностями.

# ТЕХНОЛОГИИ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

## Используемые языки программирования и фреймворки

PHP - это широко используемый язык программирования общего назначения, который применяется в основном для разработки веб-приложений и веб-серверных скриптов. PHP обычно встраивается непосредственно в HTML код, и его скрипты выполняются на сервере, что позволяет генерировать динамический контент на веб-страницах.

Основные характеристики PHP включают в себя:

* Простота использования: PHP имеет синтаксис, который легко читать и понимать, что делает его доступным для новичков.
* Гибкость: PHP поддерживает различные парадигмы программирования, включая процедурное и объектно-ориентированное программирование, что позволяет разработчикам выбирать подход, который лучше всего подходит для их задач.
* Широкая поддержка: PHP является одним из наиболее популярных языков программирования для веб-разработки и имеет обширное сообщество разработчиков, библиотек и фреймворков.
* Интеграция с веб-серверами: PHP легко интегрируется с различными веб-серверами, такими как Apache, Nginx и другие.
* Динамическое создание контента: PHP позволяет создавать динамические веб-страницы, которые могут меняться в зависимости от пользовательского ввода, данных из базы данных и других факторов.
* Базы данных: PHP обладает встроенной поддержкой для работы с различными системами управления базами данных, такими как MySQL, PostgreSQL, SQLite и другими.
* PHP используется для создания широкого спектра веб-приложений, включая блоги, интернет-магазины, корпоративные веб-сайты, социальные сети и многое другое. Он остается одним из наиболее популярных и распространенных языков программирования для веб-разработки благодаря своей простоте, гибкости и широкой поддержке сообщества.

SQL - это специализированный язык программирования, который используется для работы с реляционными базами данных. Он предоставляет набор команд и инструкций для управления данными в базах данных, таких как добавление, удаление, изменение и извлечение данных.

Основные характеристики SQL:

* Декларативный язык: SQL является декларативным языком программирования, что означает, что вы определяете, что вы хотите сделать с данными, а не как это делать. Вы описываете желаемый результат, а не последовательность действий.
* Манипуляция данными: SQL предоставляет команды для манипуляции данными в базе данных. Например, с помощью SQL вы можете создавать новые таблицы, добавлять новые записи, обновлять существующие записи и удалять записи.
* Язык запросов: для извлечения данных из базы данных. Вы можете создавать запросы, чтобы выбирать данные из одной или нескольких таблиц, фильтровать данные, сортировать их и группировать.

Laravel - это бесплатный и с открытым исходным кодом фреймворк для создания веб-приложений на языке программирования PHP. Он предоставляет разработчикам инструменты и структуру для быстрой и эффективной разработки веб-приложений, используя современные практики разработки и популярные шаблоны проектирования.

Основные характеристики Laravel:

* Элегантный синтаксис: Laravel предлагает простой и выразительный синтаксис, который делает код легким для чтения и понимания.
* Мощные функции: Laravel обладает богатым набором функций, включая маршрутизацию, контроллеры, валидацию данных, работу с базой данных, аутентификацию пользователей, кэширование, отправку электронной почты и многое другое.
* Быстрый старт: Laravel предлагает множество готовых компонентов и инструментов, которые позволяют быстро начать работу над проектом, не тратя время на настройку инфраструктуры.

C# - это объектно-ориентированный язык программирования, разработанный Microsoft. Предназначен для создания широкого спектра приложений, включая веб-приложения, мобильные приложения, десктопные приложения, игры и многое другое.

Основные характеристики C#:

* Объектно-ориентированный: C# поддерживает такие принципы ООП, как инкапсуляция, наследование и полиморфизм.
* Типизация: C# является строго типизированным языком, что означает, что типы данных должны быть явно объявлены и проверены во время компиляции.
* Многопоточность: C# поддерживает многопоточное программирование, что позволяет создавать приложения, которые могут выполнять несколько задач одновременно.
* Сборка мусора: C# автоматически управляет памятью, освобождая программиста от необходимости вручную управлять памятью.
* Интеграция с .NET: C# тесно интегрирован с платформой .NET, что позволяет использовать библиотеки классов .NET для создания приложений.
* LINQ: C# поддерживает Language Integrated Query (LINQ), что позволяет писать запросы к данным прямо в коде.
* Асинхронное программирование: C# поддерживает асинхронное программирование с использованием ключевых слов async и await, что упрощает написание отзывчивых приложений.
* Расширяемость: C# позволяет создавать расширения и библиотеки, которые могут быть использованы в других проектах.

## Используемое программное обеспечение

Draw.io — это бесплатный онлайн инструмент для создания диаграмм и схем. Он предоставляет широкий набор инструментов и элементов, которые позволяют пользователям создавать различные типы диаграмм, такие как организационные диаграммы, блок-схемы, ER-диаграммы, сетевые диаграммы, диаграммы потока процессов и многое другое.

Open Server Panel — это бесплатное программное обеспечение для Windows, которое предоставляет локальную среду разработки для веб-приложений. Он включает в себя веб-сервер Apache, базу данных MySQL, сервер приложений PHP и другие инструменты, необходимые для разработки и тестирования веб-приложений.

PhpStorm — это интегрированная среда разработки (IDE) для языка программирования PHP, разработанная компанией JetBrains. PhpStorm предоставляет разработчикам мощные инструменты для написания кода, отладки, тестирования и поддержки веб-приложений на PHP.

Postman — это популярный инструмент для разработки, тестирования и документирования API. Он предоставляет разработчикам удобный способ отправлять HTTP-запросы к веб-сервисам и анализировать ответы.

Visual Studio — это интегрированная среда разработки (IDE) от Microsoft, предназначенная для создания, отладки и развертывания приложений на различных платформах и языках программирования.

# ГЛАВА 1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ ИНФОРМАЦИОННОГО ПРОДУКТА.

## Описание проекта и задач

**ОБЩЕЕ ОПИСАНИЕ ПРОЕКТА**

Все пользователи системы подразделяются на две группы:

1. Администраторы.
2. Курьеры.

Курьеры получают доступ к функциям только после успешной авторизации.

Функционал не авторизированного пользователя:

1. Авторизация.

Функционал администратора:

1. Регистрация курьеров.
2. Добавление, редактирование, удаление бонусов.
3. Просмотр, редактирование, удаление профилей у курьеров.
4. Просмотр, подтверждение/отмена доступностей курьеров.
5. Просмотр, добавление, редактирование, удаление штрафов у курьеров.

Функционал курьера:

1. Просмотр, добавление, редактирование, удаление своих доступностей.
2. Просмотр своего штрафа.
3. Просмотр своего статуса.
4. Выставление доступности.

Функционал авторизированного пользователя:

Выход.

1. Просмотр и редактирование своего профиля.
2. Просмотр бонусов.

В примерах будет использоваться переменная {{host}} которая обозначает адрес

информационного продукта.

**ОБЩИЕ ТРЕБОВАНИЯ**

Сервер не может или не будет обрабатывать запрос из-за чего-то, что очевидно является ошибкой клиента:

|  |
| --- |
| Неверный формат запроса |
| "code": 400,  "message": "Bad Request" |

Для доступа к ресурсу требуется аутентификация, и клиент не предоставил действительные учетные данные:

|  |
| --- |
| Не авторизован |
| «code»: 401,  «message»: «Unauthorized» |

Сервер понял запрос, но отказывается его авторизовать. Это может быть связано с отсутствием необходимых прав доступа:

|  |
| --- |
| Доступ запрещен |
| «code»: 403,  «message»: «Forbidden» |

Сервер не может найти запрошенный ресурс. Это может быть связано с неправильным URL или удалением ресурса:

|  |
| --- |
| Ресурс не найден |
| «code»: 404,  «message»: «Not Found» |

В случае ошибок, связанных с валидацией данных во всех запросах необходимо возвращать следующее:

|  |
| --- |
| Тело ответа |
| “code”: <code>,  “message”: <message> |

Вместо <code> и <message> необходимо указывать соответствующее значение, определенное в описании ответа на соответствующий запрос.

## Специфические требования к API

**Авторизация**

Запрос для аутентификации пользователя в системе. При отправке запроса необходимо передать объект с логином и паролем. Если пользователь отправил корректные данные, то необходимо вернуть сгенерированный токен, а иначе сообщение об ошибке.

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: <http://courseproject>4/api/login |
| Заголовки |
| Content-Type: application/json  Accept: application/json |
| Тело запроса |
| {  password: QWEqwe123@  login: Alex  } |
| Ответ на успешную авторизацию (200 OK) |
| {  «token»: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»,  «user»: {  “id”: 1,  “surname”: “Рюгин”,  “name”: “Алексей”,  “patronymic”: “Иванович”,  “login”: “Alex”,  “role\_id”: 1,  “fine\_id”: 1,  “status\_id”: 1,  “created\_at”: “2025-01-13T08:43:24.000000Z”,  “updated\_at”: “2025-01-27T10:26:13.000000Z”  }  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Неверный логин/пароль»  } |

**Выход**

Запрос предназначен для очистки значение токена пользователя.

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: <http://courseproject>4/api/logout |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Тело запроса |
| {  token: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешный выход из системы (200 OK) |
| {  «message»: «Вы вышли из системы»  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован «  } |

**Регистрация**

Запрос для регистрации пользователя в системе. При отправке запроса необходимо передать объект с именем, фамилией, почтой, логином, паролем и номером телефона. Если пользователь отправил корректные данные, то необходимо вернуть сгенерированный токен, а иначе сообщение об ошибке.

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: <http://courseproject>4/api/register |
| Заголовки |
| Authorization: application/json  Accept: application/json |
| Тело запроса |
| {  name: Дмитрий  surname: Евсеев  patronymic: Евгеньевич  login: vseev  password: ZXCzxc123@  },  {  «token»: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешную регистрацию (200 OK) |
| {  «user»: {  «name»: «Дмитрий»,  «surname»: «Евсеев»,  “patronymic”: “Евгеньевич”,  “login”: “evseevm”,  “role\_id”: 2,  “fine\_id”: 1,  “status\_id”: 1,  “updated\_at”: “2025-01-27T11:12:23.000000Z”,  “created\_at”: “2025-01-27T11:12:23.000000Z”,  “id”: 14  },  «token»: «$2y$12$Ieutp71ss421EHR7GNCeOMiXS6nQiWrtbaV0bfi»  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {  «code»: 422,  «message»: «Ошибка валидации данных»  } |

**CRUD-операции по штрафу**

Get-запрос к /api/fine с токеном в теле запроса возвращает список штрафов (200 ok) или сообщение об ошибке авторизации (401 unauthorized) или о том, что штрафы не найдены (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: <http://courseproject>4/api/fine |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  token: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешный просмотр (200 OK) |
| {  “id”: 1,  “description”: “Без штрафов”,  “created\_at”: “2025-01-11T12:03:52.000000Z”,  “updated\_at”: “2025-01-11T12:03:52.000000Z”  },  {  “id”: 2,  “description”: “Отстранение”,  “created\_at”: “2025-01-11T12:03:52.000000Z”,  “updated\_at”: “2025-01-11T12:03:52.000000Z”  } |
| Ошибка авторизации |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка поиска (404 Not Found) |
| {  «code»: 404,  «message»: «Не найдено»  } |

Post-запрос к /api/fine с описанием штрафа в теле и токеном в теле запроса создаёт новый штраф (200 ok), либо возвращает ошибку авторизации (401 unauthorized), доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: <http://courseproject>4/api/fine |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  «description»: «Без штрафов»  },  {  «token»: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешное создание (200 OK) |
| {  “id”: 1,  “description”: “Без штрафов”,  “created\_at”: “2025-01-11T12:03:52.000000Z”,  “updated\_at”: “2025-01-11T12:03:52.000000Z”  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка доступа (403 Forbidden) |
| {  «code»: 403,  «message»: «Запрещено»  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {  «code»: 422,  «message»: «Ошибка валидации данных»  } |

Patch-запрос к /api/fine/{id} с обновленным описанием штрафа и токеном в теле запроса обновляет существующий штраф (200 ok) или возвращает ошибку авторизации (401 unauthorized), доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: patch  Url: <http://courseproject>4/api/fine/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  «description»: «Без всех штрафов»  },  {  «token»: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешное обновлено (200 OK) |
| {  “id”: 1,  “description”: “Без всех штрафов”,  “created\_at”: “2025-01-11T12:03:52.000000Z”,  “updated\_at”: “2025-01-11T12:03:52.000000Z”  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка доступа (403 Forbidden) |
| {  «code»: 403,  «message»: «Запрещено»  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {  «code»: 422,  «message»: «Ошибка валидации данных»  } |

Delete-запрос к /api/fine/{id} с токеном в теле запроса удаляет штраф (200 ok) или возвращает ошибку авторизации (401 unauthorized) или доступа (403 forbidden).

|  |
| --- |
| HTTP-метод и URL |
| Метод: delete  Url: <http://courseproject>4/api/fine/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Тело запроса |
| {  «token»: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешное удалено (200 OK) |
| {  «code»: 200,  «message»: «Штраф удален»  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка доступа (403 Forbidden) |
| {  «code»: 403,  «message»: «Запрещено»  } |

Get-запрос к /api/fine/{id} с токеном в теле запроса возвращает штраф по его id (200 ok) или сообщение об ошибке авторизации (401 unauthorized) или о том, что штраф не найден (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: <http://courseproject>4/api/fine/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  token: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешный просмотр (200 OK) |
| {  “id”: 1,  “description”: “Без штрафов”,  “created\_at”: “2025-01-11T12:03:52.000000Z”,  “updated\_at”: “2025-01-11T12:03:52.000000Z”  } |
| Ошибка авторизации |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка поиска (404 Not Found) |
| {  «code»: 404,  «message»: «Не найдено»  } |

**CRUD-операции по бонусу**

Get-запрос к /api/bonus с токеном в теле запроса возвращает список бонусов (200 ok) или сообщение об ошибке авторизации (401 unauthorized) или о том, что бонусы не найдены (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: <http://courseproject>4/api/bonus |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  token: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешный просмотр (200 OK) |
| [      {          «id»: 1,          «title»: «Приведи друга»,          «description»: «При пополнение нашей команды ты получаешь бонус.»,          “price”: “15000.00”,          “role\_id”: 1,          “created\_at”: “2025-01-13T08:28:16.000000Z”,          “updated\_at”: “2025-01-13T08:28:16.000000Z”,          “role”: {              “id”: 1,              “name”: “Администратор”,              “code”: “admin”,              “created\_at”: “2025-01-13T08:28:16.000000Z”,              “updated\_at”: “2025-01-13T08:28:16.000000Z”          }      },      {          «id»: 2,          «title»: «Доплата за плохую погоду»,          «description»: «Условия: Компания-партнёр доплатит до 15% от базовой ставки за доставленные заказы в периоды сложных погодных условий. Доплата начисляется к заказам, если осадки идут 4 часа и более.»,          “price”: “10.00”,          “role\_id”: 2,          “created\_at”: “2025-01-13T08:28:16.000000Z”,          “updated\_at”: “2025-01-13T08:28:16.000000Z”,          “role”: {              “id”: 2,              “name”: “Курьер”,              “code”: “сourier”,              “created\_at”: “2025-01-13T08:28:16.000000Z”,              “updated\_at”: “2025-01-13T08:28:16.000000Z”          }      },  . . .  ] |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка поиска (404 Not Found) |
| {  «code»: 404,  «message»: «Не найдено»  } |

Post-запрос к /api/bonus с описанием бонуса в теле и токеном в теле запроса создаёт новый бонус (200 ok), либо возвращает ошибку авторизации (401 unauthorized), доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: <http://courseproject>4/api/bonus |
| Заголовки |
|  |
| Тело запроса |
| {  «title»: «Приведи друга»,  «description»: «При пополнение нашей команды ты получаешь бонус.»,  «price»: «15000.00»  ,  },  {  «token»: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешное создание (200 OK) |
| {  «id»: 1,  «title»: «Приведи друга»,  «description»: «При пополнение нашей команды ты получаешь бонус.»,  “price”: “15000.00”,  “created\_at”: “2025-01-11T12:03:52.000000Z”,  “updated\_at”: “2025-01-11T12:03:52.000000Z”  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка доступа (403 Forbidden) |
| {  «code»: 403,  «message»: «Запрещено»  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {  «code»: 422,  «message»: «Ошибка валидации данных»  } |

Patch-запрос к /api/bonus/{id} с обновленным описанием бонуса и токеном в теле запроса обновляет существующий бонус (200 ok) или возвращает ошибку авторизации (401 unauthorized), доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: patch  Url: <http://courseproject>4/api/bonus/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  «title»: «Приведи друга»,  },  {  «token»: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешное обновление (200 OK) |
| {  «id»: 1,  «title»: «Приведи друга»,  «description»: «При пополнение нашей команды ты получаешь бонус.»,  “price”: 1500,  “created\_at”: “2025-01-11T12:03:52.000000Z”,  “updated\_at”: “2025-01-11T12:03:52.000000Z”  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка доступа (403 Forbidden) |
| {  «code»: 403,  «message»: «Запрещено»  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {  «code»: 422,  «message»: «Ошибка валидации данных»  } |

Delete-запрос к /api/bonus/{id} с токеном в теле запроса удаляет бонус (200 ok) или возвращает ошибку авторизации (401 unauthorized) или доступа (403 forbidden).

|  |
| --- |
| HTTP-метод и URL |
| Метод: delete  Url: <http://courseproject>4/api/bonus/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Тело запроса |
| {  «token»: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешное удаление (200 OK) |
| {  «code»: 200,  «message»: «Бонус удален»  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка доступа (403 Forbidden) |
| {  «code»: 403,  «message»: «Запрещено»  } |

Get-запрос к /api/bonus/{id} с токеном в теле запроса возвращает бонусов по его id (200 ok) или сообщение об ошибке авторизации (401 unauthorized) или о том, что бонусы не найдены (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: <http://courseproject>4/api/bonus/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  token: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешный просмотр (200 OK) |
| {      «id»: 1,      «title»: «Приведи друга»,      «description»: «При пополнение нашей команды ты получаешь бонус.»,      “price”: “15000.00”,      “role\_id”: 1,      “created\_at”: “2025-01-13T08:28:16.000000Z”,      “updated\_at”: “2025-01-13T08:28:16.000000Z”,      “role”: {          “id”: 1,          “name”: “Администратор”,          “code”: “admin”,          “created\_at”: “2025-01-13T08:28:16.000000Z”,          “updated\_at”: “2025-01-13T08:28:16.000000Z”      }  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка поиска (404 Not Found) |
| {  «code»: 404,  «message»: «Не найдено»  } |

**CRUD-операции по статусу**

Get-запрос к /api/status с токеном в теле запроса возвращает список статусов (200 ok) или сообщение об ошибке авторизации (401 unauthorized) или о том, что бонусы не найдены (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: <http://courseproject>4/api/status |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  token: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешный просмотр (200 OK) |
| {  “id”: 1,  “name”: “Активный”,  “code”: “active”,  “created\_at”: “2025-01-11T12:03:52.000000Z”,  “updated\_at”: “2025-01-11T12:03:52.000000Z”  },  {  “id”: 2,  “name”: “Неактивные”,  “code”: “inactive”,  “created\_at”: “2025-01-11T12:03:52.000000Z”,  “updated\_at”: “2025-01-11T12:03:52.000000Z”  },  {  “id”: 3,  “name”: “Отстранён”,  “code”: “suspended”,  “created\_at”: “2025-01-11T12:03:52.000000Z”,  “updated\_at”: “2025-01-11T12:03:52.000000Z”  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка поиска (404 Not Found) |
| {  «code»: 404,  «message»: «Не найдено»  } |

Post-запрос к /api/status с описанием статуса в теле и токеном в теле запроса создаёт новый статус (200 ok), либо возвращает ошибку авторизации (401 unauthorized), доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: <http://courseproject>4/api/status |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  «name»: «Неактивные»,  «code»: «inactive»,  },  {  «token»: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешное создание (200 OK) |
| {  “id”: 1,  “name”: “Неактивные”,  “code”: “inactive”,  “created\_at”: “2025-01-11T12:03:52.000000Z”,  “updated\_at”: “2025-01-11T12:03:52.000000Z”  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка доступа (403 Forbidden) |
| {  «code»: 403,  «message»: «Запрещено»  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {  «code»: 422,  «message»: «Ошибка валидации данных»  } |

Patch-запрос к /api/status/{id} с обновленным описанием статуса и токеном в теле запроса обновляет существующий статус (200 ok) или возвращает ошибку авторизации (401 unauthorized), доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: patch  Url: <http://courseproject>4/api/status/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  «name»: «Активный»,  },  {  «token»: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешное обновление (200 OK) |
| {  “id”: 1,  “name”: “Активный”,  “code”: “inactive”,  “created\_at”: “2025-01-11T12:03:52.000000Z”,  “updated\_at”: “2025-01-11T12:03:52.000000Z”  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка доступа (403 Forbidden) |
| {  «code»: 403,  «message»: «Запрещено»  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {  «code»: 422,  «message»: «Ошибка валидации данных»  } |

Delete-запрос к /api/status/{id} с токеном в теле запроса удаляет статус (200 ok) или возвращает ошибку авторизации (401 unauthorized) или доступа (403 forbidden).

|  |
| --- |
| HTTP-метод и URL |
| Метод: delete  Url: <http://courseproject>4/api/status/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Тело запроса |
| {  «token»: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешное удаление (200 OK) |
| {  «code»: 200,  «message»: «Статус удален»  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка доступа (403 Forbidden) |
| {  «code»: 403,  «message»: «Запрещено»  } |

Get-запрос к /api/status/{id} с токеном в теле запроса возвращает статус по его id (200 ok) или сообщение об ошибке авторизации (401 unauthorized) или о том, что статус не найден (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: <http://courseproject>4/api/status/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  token: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешный просмотр (200 OK) |
| {  “id”: 1,  “name”: “Активный”,  “code”: “active”,  “created\_at”: “2025-01-11T12:03:52.000000Z”,  “updated\_at”: “2025-01-11T12:03:52.000000Z”  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка поиска (404 Not Found) |
| {  «code»: 404,  «message»: «Не найдено»  } |

**CRUD-операции по профилю**

Get-запрос к /api/profile с токеном в теле запроса возвращает список пользователей (200 ok) или сообщение об ошибке авторизации (401 unauthorized) или о том, что пользователи не найдены (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: <http://courseproject>4/api/profile |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  token: «$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC»  } |
| Ответ на успешный просмотр (200 OK) |
| [      {          “id”: 1,          “surname”: “Рюгин”,          “name”: “Алексей”,          “patronymic”: “Иванович”,          “login”: “Alex”,          “role\_id”: 1,          “fine\_id”: 1,          “status\_id”: 1,          “created\_at”: “2025-01-13T08:28:17.000000Z”,          “updated\_at”: “2025-01-13T08:30:45.000000Z”      },      {          “id”: 2,          “surname”: “Ладяев”,          “name”: “Никита”,          “patronymic”: “Александрович”,          “login”: “Nikita”,          “role\_id”: 2,          “fine\_id”: 1,          “status\_id”: 1,          “created\_at”: “2025-01-13T08:28:17.000000Z”,          “updated\_at”: “2025-01-13T08:28:17.000000Z”      },  . . .  ] |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка поиска (404 Not Found) |
| {  «code»: 404,  «message»: «Не найдено»  } |

Patch-запрос к /api/profile/{id} с обновленным описанием пользователя и токеном в теле запроса обновляет существующего пользователя (200 ok) или возвращает ошибку авторизации (401 unauthorized), доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: patch  Url: <http://courseproject>4/api/profile/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {      «surname»: «Рюгин»,      «name»: «Алексей»,      «patronymic»: «Иванович»,      “login”: “Alex”,      “password”: “Lexa2019@”,  }  {  “token”: “$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC”  } |
| Ответ на успешное обновление (200 OK) |
| {      “id”: 1,      “surname”: “Рюгин”,      “name”: “Алексей”,      “patronymic”: “Иванович”,      “login”: “Alex”,      “role\_id”: 1,      “fine\_id”: 1,      “status\_id”: 1,      “created\_at”: “2025-01-13T08:28:17.000000Z”,      “updated\_at”: “2025-02-10T09:40:40.000000Z”  } |
| Ошибка авторизации (401 Unauthorized) |
| {  «code»: 401,  «message»: «Не авторизован»  } |
| Ошибка доступа (403 Forbidden) |
| {  «code»: 403,  «message»: «Запрещено»  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {  «code»: 422,  «message»: «Ошибка валидации данных»  } |

Delete-запрос к /api/profile/{id} с токеном в теле запроса удаляет пользователя (200 ok) или возвращает ошибку авторизации (401 unauthorized) или доступа (403 forbidden).

|  |
| --- |
| HTTP-метод и URL |
| Метод: delete  Url: http://courseproject4/api/profile/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Тело запроса |
| {  "token": "$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC"  } |
| Ответ на успешное удаление (200 OK) |
| {  "code": 200,  "message": "Пользователь удален"  } |
| Ошибка авторизации (401 Unauthorized) |
| {  "code": 401,  "message": "Не авторизован"  } |
| Ошибка доступа (403 Forbidden) |
| {  "code": 403,  "message": "Запрещено"  } |

Get-запрос к /api/profile/{id} с токеном в теле запроса возвращает пользователя по его id (200 ok) или сообщение об ошибке авторизации (401 unauthorized) или о том, что пользователь не найден (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: http://courseproject4/api/profile/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  token: "$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC"  } |
| Ответ на успешный просмотр (200 OK) |
| {      "id": 2,      "surname": "Ладяев",      "name": "Никита",      "patronymic": "Александрович",      "login": "Nikita",      "role\_id": 2,      "fine\_id": 1,      "status\_id": 1,      "created\_at": "2025-01-13T08:28:17.000000Z",      "updated\_at": "2025-01-13T08:28:17.000000Z"  } |
| Ошибка авторизации (401 Unauthorized) |
| {  "code": 401,  "message": "Не авторизован"  } |
| Ошибка поиска (404 Not Found) |
| {  "code": 404,  "message": "Не найдено"  } |

**CRUD-операции по доступности**

Get-запрос к /api/accesses с токеном в теле запроса возвращает список доступностей в пределах двух недель (200 ok) или сообщение об ошибке авторизации (401 unauthorized) или о том, что пользователи не найдены (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: http://courseproject4/api/accesses |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  token: "$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC"  } |
| Ответ на успешный просмотр (200 OK) |
| [      {          "id": 6,          "date": "2025-02-15",          "startChange": "08:00:00",          "endChange": "23:00:00",          "confirm": 0,          "user\_id": 2,          "created\_at": "2025-01-13T11:22:24.000000Z",          "updated\_at": "2025-01-13T11:22:24.000000Z"      },      {          "id": 7,          "date": "2025-02-13",          "startChange": "08:00:00",          "endChange": "23:00:00",          "confirm": 0,          "user\_id": 6,          "created\_at": **null**,          "updated\_at": **null**      },  . . .  ] |
| Ошибка авторизации (401 Unauthorized) |
| {  "code": 401,  "message": "Не авторизован"  } |
| Ошибка поиска (404 Not Found) |
| {  "code": 404,  "message": "Не найдено"  } |

Get-запрос к /api/accesses/{id} с токеном в теле запроса возвращает доступность по её id (200 ok) или сообщение об ошибке авторизации (401 unauthorized) или о том, что доступность не найдена (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: http://courseproject4/api/accesses/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  token: "$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC"  } |
| Ответ на успешный просмотр (200 OK) |
| {      "id": 6,      "date": "2025-02-15",      "startChange": "08:00:00",      "endChange": "23:00:00",      "confirm": 0,      "user\_id": 2,      "created\_at": "2025-01-13T11:22:24.000000Z",      "updated\_at": "2025-01-13T11:22:24.000000Z"  } |
| Ошибка авторизации (401 Unauthorized) |
| {  "code": 401,  "message": "Не авторизован"  } |
| Ошибка поиска (404 Not Found) |
| {  "code": 404,  "message": "Не найдено"  } |

**Подтверждение/отмена доступности**

Patch-запрос к /api/accesses-confirm/{id} с токеном в теле запроса обновляет статус подтверждения доступности на true (200 ok) или возвращает ошибку авторизации (401 unauthorized), доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: patch  Url: http://courseproject4/api/accesses-confirm/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  "token": "$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC"  } |
| Ответ на успешное обновление (200 OK) |
| {      "message": "Доступность успешно подтверждена",      "access": {          "id": 6,          "date": "2025-02-15",          "startChange": "08:00:00",          "endChange": "23:00:00",          "confirm": **true**,          "user\_id": 2,          "created\_at": "2025-01-13T11:22:24.000000Z",          "updated\_at": "2025-02-10T10:54:11.000000Z"      }  } |
| Ошибка авторизации (401 Unauthorized) |
| {  "code": 401,  "message": "Не авторизован"  } |
| Ошибка доступа (403 Forbidden) |
| {  "code": 403,  "message": "Запрещено"  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {  "code": 422,  "message": "Ошибка валидации данных"  } |

Patch-запрос к /api/accesses-cancel/{id} с токеном в теле запроса обновляет статус подтверждения доступности на false (200 ok) или возвращает ошибку авторизации (401 unauthorized), доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: patch  Url: http://courseproject4/api/accesses-cancel/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  "token": "$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC"  } |
| Ответ на успешное обновление (200 OK) |
| {      "message": "Доступность успешно отменена",      "access": {          "id": 6,          "date": "2025-02-15",          "startChange": "08:00:00",          "endChange": "23:00:00",          "confirm": **false**,          "user\_id": 2,          "created\_at": "2025-01-13T11:22:24.000000Z",          "updated\_at": "2025-02-10T10:54:11.000000Z"      }  } |
| Ошибка авторизации (401 Unauthorized) |
| {  "code": 401,  "message": "Не авторизован"  } |
| Ошибка доступа (403 Forbidden) |
| {  "code": 403,  "message": "Запрещено"  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {  "code": 422,  "message": "Ошибка валидации данных"  } |

Post-запрос к /api/accesses-partial-confirm с описанием времени подтверждения и токеном в теле запроса создаёт новую доступность с подтверждённым интервалом, создавая новые доступности с оставшимися не подтверждёнными участками интервала (200 ok), либо возвращает ошибку авторизации (401 unauthorized), доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: http://courseproject4/api/accesses-partial-confirm/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  "startChange": "10",  "endChange": "22",  },  {  "token": "$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC"  } |
| Ответ на успешное создание (200 OK) |
| {      "message": "Доступность успешно частично подтверждена",      "new\_access": {          "date": "2025-02-15",          "startChange": "10:00:00",          "endChange": "22:00:00",          "confirm": **true**,          "user\_id": 2,          "updated\_at": "2025-02-10T11:02:47.000000Z",          "created\_at": "2025-02-10T11:02:47.000000Z",          "id": 9      },      "updated\_access": {          "id": 6,          "date": "2025-02-15",          "startChange": "08:00:00",          "endChange": "23:00:00",          "confirm": 0,          "user\_id": 2,          "created\_at": "2025-01-13T11:22:24.000000Z",          "updated\_at": "2025-02-10T10:58:19.000000Z"      }  } |
| Ошибка авторизации (401 Unauthorized) |
| {  "code": 401,  "message": "Не авторизован"  } |
| Ошибка доступа (403 Forbidden) |
| {  "code": 403,  "message": "Запрещено"  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {  "code": 422,  "message": "Ошибка валидации данных"  } |

Post-запрос к /api/accesses-partial-cancel с описанием времени отмены и токеном в теле запроса создаёт новую доступность с отменённым интервалом, создавая новые доступности с оставшимися подтверждёнными участками интервала (200 ok), либо возвращает ошибку авторизации (401 unauthorized), доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: http://courseproject4/api/accesses-partial-cancel/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {  "startChange": "10",  "endChange": "22",  },  {  "token": "$2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC"  } |
| Ответ на успешное создание (200 OK) |
| {      "message": "Доступность успешно частично отменена",      "new\_access": {          "date": "2025-02-15",          "startChange": "08:00:00",          "endChange": "23:00:00",          "confirm": **false**,          "user\_id": 2,          "updated\_at": "2025-02-10T11:16:43.000000Z",          "created\_at": "2025-02-10T11:16:43.000000Z",          "id": 12      },      "updated\_access": {          "id": 9,          "date": "2025-02-15",          "startChange": "10:00:00",          "endChange": "22:00:00",          "confirm": 1,          "user\_id": 2,          "created\_at": "2025-02-10T11:02:47.000000Z",          "updated\_at": "2025-02-10T11:02:47.000000Z"      }  } |
| Ошибка авторизации (401 Unauthorized) |
| {  "code": 401,  "message": "Не авторизован"  } |
| Ошибка доступа (403 Forbidden) |
| {  "code": 403,  "message": "Запрещено"  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {  "code": 422,  "message": "Ошибка валидации данных"  } |

## **Требования к мобильному приложению**

***Авторизация***

Страница должна содержать логотип приложения и приветствие. Должен быть блок с полями для заполнения логина и пароля, а внизу кнопка «Войти», оформленное в соответствии (см. рисунок 1).

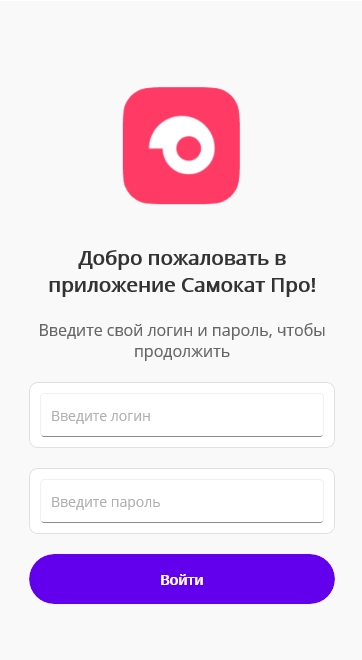


Рисунок 1 – Страница авторизации

**Главная**

Страница должна содержать метку с приветствием по имени пользователя в зависимости от времени суток. Должно быть три раздела: «Пользователь», «Функционал администратора», «Выход из системы», оформленное в соответствии (см. рисунок 2). В первом разделе должна быть рамка, по нажатию на которую происходит переход на страницу профиля с данными о пользователе. Во втором разделе должны быть рамки: «Доступности», «Курьеры», «Зарегистрировать», «Бонусы», «Штрафы». По нажатию на «Доступности» должна открываться страница с доступностями курьеров, при нажатии на «Курьеры» должна открываться страница со списком курьеров, по нажатию на «Зарегистрировать» должна открываться страница, на которой можно добавить нового пользователя в систему, при нажатии на «Бонусы» должна открываться страница со всеми существующими бонусами, по нажатию на «Штрафы» должна открываться страница со всеми штрафами из базы данных. В третьем разделе должна быть одна рамка с названием «Выход», по нажатию которой должен воспроизводиться выход из аккаунта и происходить переход на страницу авторизации.

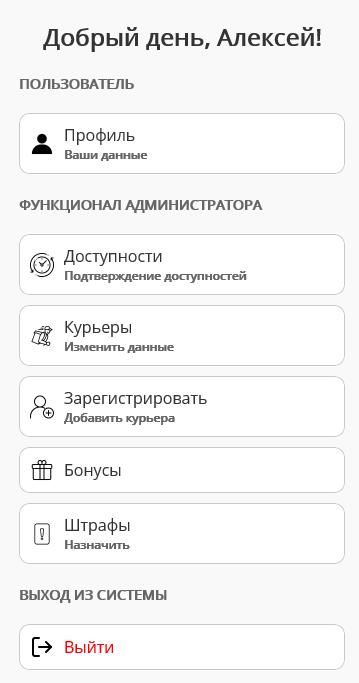


Рисунок 2 – Главная страница

***Профиль***

На данной странице (см. рисунок 3) должны выводиться данные о текущем пользователя, такие как: «Фамилия», «Имя», «Отчество» и «Логин». Также под его данными должна быть кнопка «Редактировать профиль», которая переносит на страницу для редактирования данных о пользователе.

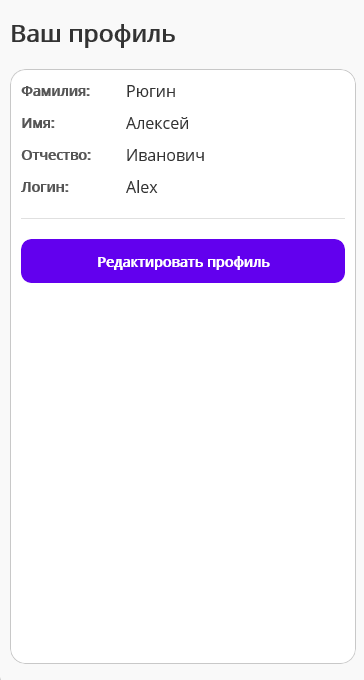


Рисунок 3 – Страница профиля

***Доступности***

На этой странице должны быть даты на которых существуют доступности в базе данных. Чтобы страница не была слишком длинной, сами доступности изначально спрятаны (см. рисунок 4), и чтобы их увидеть нужно раскрыть дату, после чего появляется список доступностей (см. рисунок 5) на выбранный день с информацией о курьере, что её выставил, интервал (начало и конец), а также статус подтверждения. По нажатию на рамку с доступностью, происходит переход на страницу с деталями доступности.

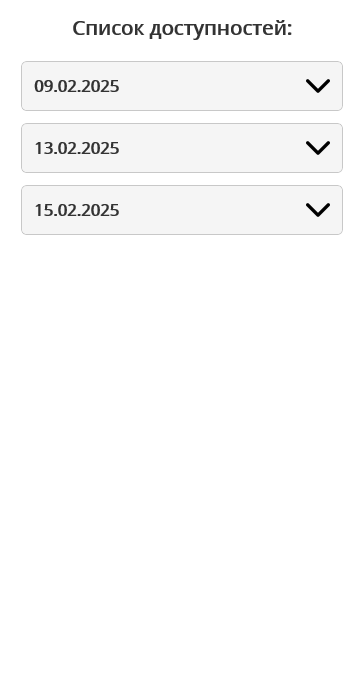


Рисунок 4 – Страница доступностей

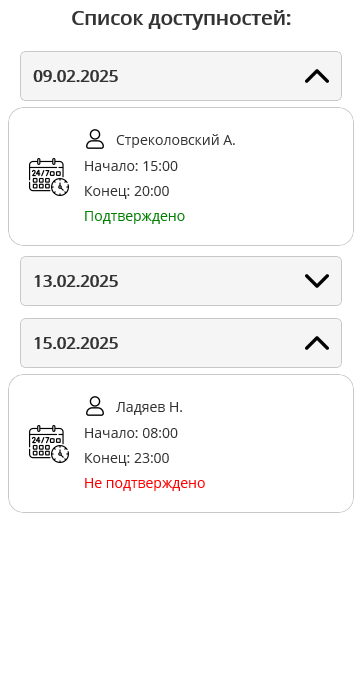


Рисунок 5 – Страница доступностей, показывающая доступности

***Курьеры***

На данной странице (см. рисунок 6) должен быть список из курьеров с помощью рамок, в которых содержится имя, фамилия и логин курьера, также по нажатию на них, должен осуществляться переход на страницу с полной информацией о курьере.

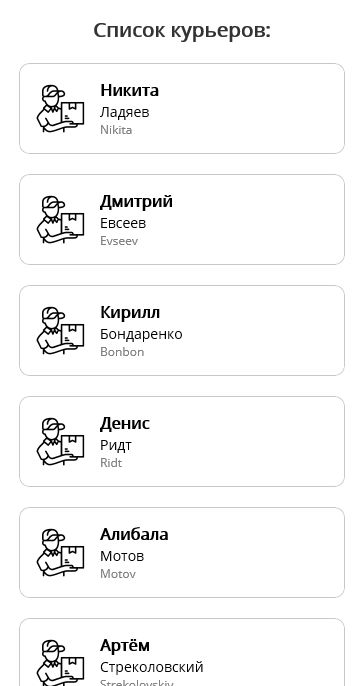


Рисунок 6 – Страница курьеров

**Регистрация**

Данная страница (см. рисунок 7, 8) должна содержать логотип приложения, метку предназначения страницы, также блок с полями для заполнения данных о новом пользователе таких как: фамилия, имя, отчество, логин, пароль. Должно быть поле для подтверждения пароля. После чего под блоком, должна иметься кнопка «Зарегистрировать» для создания нового пользователя.

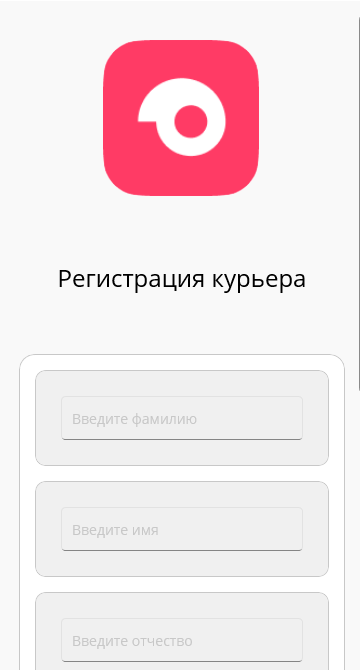


Рисунок 7 – Страница регистрации

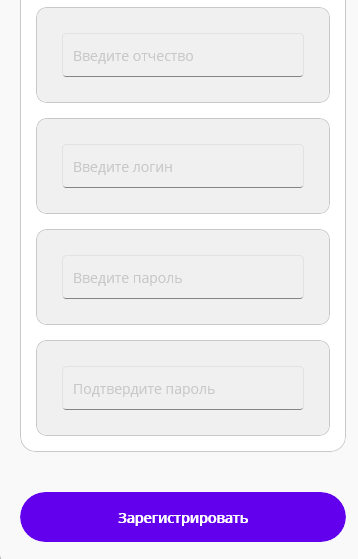


Рисунок 8 – Страница регистрации (2)

***Бонусы***

На этой странице (см. рисунок 9) должны выводиться рамки с бонусами (при нажатии которых происходит переход на страницу с подробной информацией о выбранном бонусе), в которых содержится информация, такая как: название бонуса, награда за выполнение, а также для каких пользователей работает данный бонус. Внизу экрана должна постоянно быть кнопка «Создать бонус», при нажатии которой происходит переход на страницу для создания бонуса.

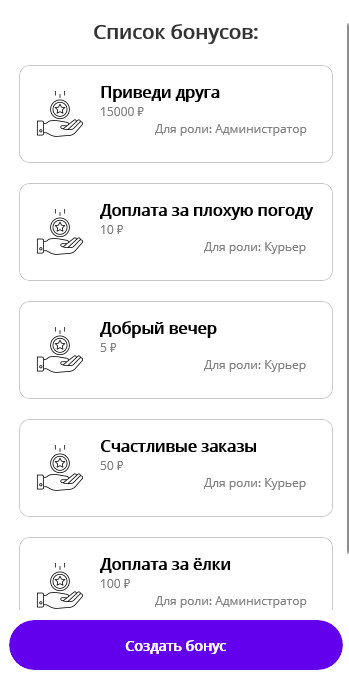


Рисунок 9 – Страница бонусов

***Штрафы***

На этой странице (см. рисунок 10) должны выводиться рамки с описаниями штрафов и при нажатии которых происходит переход на страницу с назначением выбранного штрафа курьеру. Также страница должна иметь кнопку «Создать штраф», при нажатии которой происходит переход на страницу для создания штрафа.

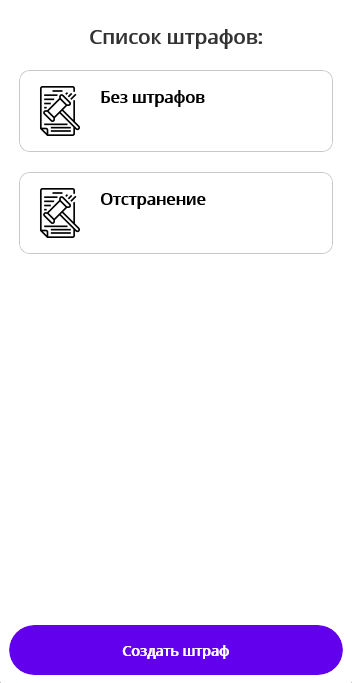


Рисунок 10 – Страница штрафов

**Редактирование профиля**

На этой странице (см. рисунок 11) должны быть блоки для изменения информации о текущем пользователе с уже заполненными полями из имеющихся данных. Также должен быть CheckBox, который изначально является активным, с текстом «Не изменять пароль» при деактивации которого должны появляться дополнительные поля для изменения пароля. Внизу должна быть кнопка «Сохранить», которая сохраняет изменения при нажатии.

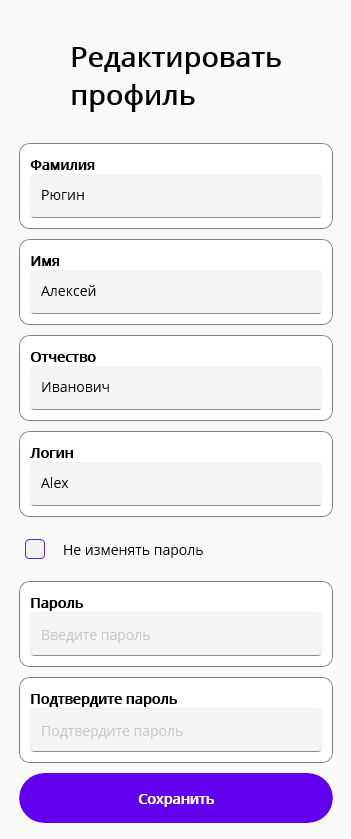


Рисунок 11 – Страница редактирования профиля

**Детали доступности**

На этой странице должны выводиться данные о доступности, такие как: дата доступности, фамилия и имя курьера, интервал, статус. Также продумано два варианта отображения страницы:

1. Если доступность стоит на день, который в прошлом, то должна быть только одна кнопка «Просмотреть курьера» (см. рисунок 12) по нажатию на которую происходит переход на страницу с полной информацией о курьере.
2. Если доступность стоит на сегодняшний день или на будущее, то должны быть три кнопки «Просмотреть курьера», «Подтвердить» или «Отменить» и «Подтвердить часть» или «Отменить часть» в зависимости от статуса (см. рисунок 13, 14). При нажатии на кнопки «Подтвердить» или «Отменить» доступность должна менять свой статус на противоположный, а при нажатии на «Подтвердить часть» или «Отменить часть» пользователя должно переносить на страницу частичного подтверждения или отмены доступности, в зависимости от нажатой кнопки.

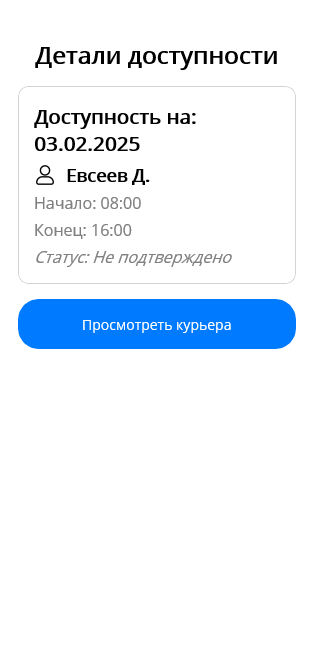


Рисунок 12 – Страница деталей неподтверждённой доступности

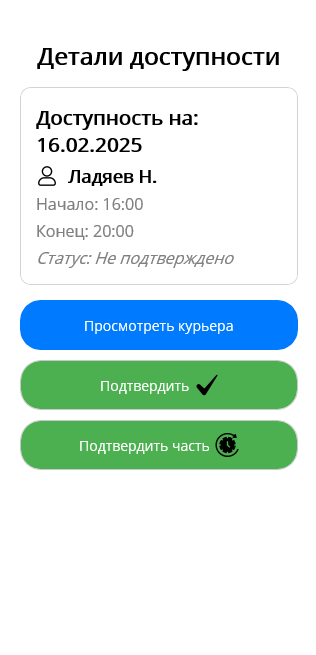


Рисунок 13 – Страница деталей неподтверждённой доступности

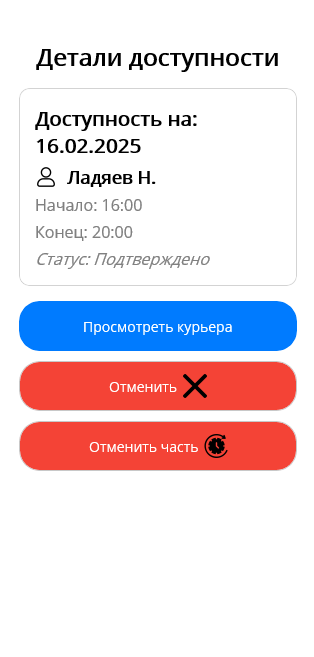


Рисунок 14 – Страница деталей подтверждённой доступности

**Частичное подтверждение/отмена доступности**

Эта страница также должна иметь два варианта (см. рисунок 15, 16) в зависимости от того, что пользователь будет делать с доступностью. На ней должен быть блок с информацией о доступности (дата доступности, дата, интервал), два Picker’а для того, чтобы выбрать с какого по какой час подтверждается или отменяется доступность и кнопка «Подтвердить выделенные часы» или «Отменить выделенные часы».

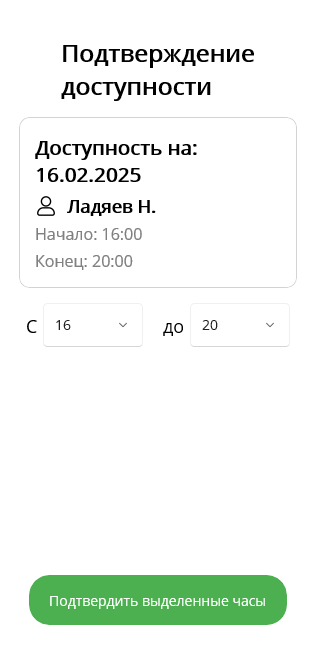


Рисунок 15 – Страница частичного подтверждения доступности

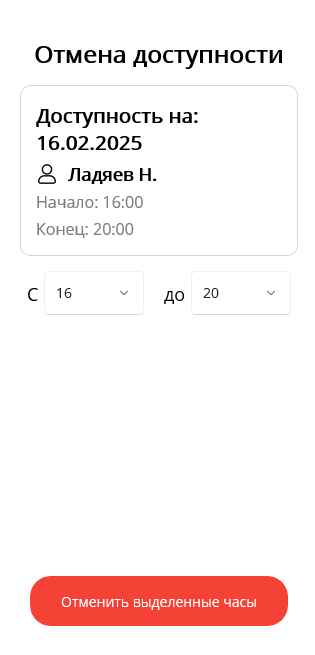


Рисунок 16 – Страница частичной отмены доступности

***Курьер***

На этой странице (см. рисунок 17) должно быть три блока: основная информация, штраф, статус. В первом блоке должна описываться информация о курьере, такая как: фамилия, имя, отчество и логин. Во втором должна быть информация о штрафе. В третьем информация о статусе курьера. После блоков должно быть две кнопки «Изменить профиль» и «Удалить курьера». По нажатию на первую должен осуществляться переход на страницу редактирования профиля курьера, а по нажатию на второю курьер должен удаляться из базы данных.

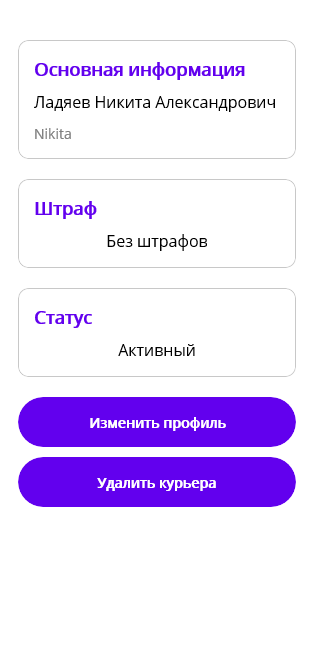


Рисунок 17 – Страница курьера

***Редактирование профиля курьера***

Данная страница (см. рисунок 18) должна содержать два раздела: основные данные и статус. Первый раздел должен содержать блоки с уже заполненными полями из имеющихся данных. Также должен быть CheckBox, который изначально является активным, с текстом «Не изменять пароль» при деактивации которого должны появляться дополнительные поля для изменения пароля. Во втором разделе должен содержаться блок с раскрывающимся списком из всех существующих статусов. Внизу должна быть кнопка «Сохранить», которая сохраняет изменения при нажатии.

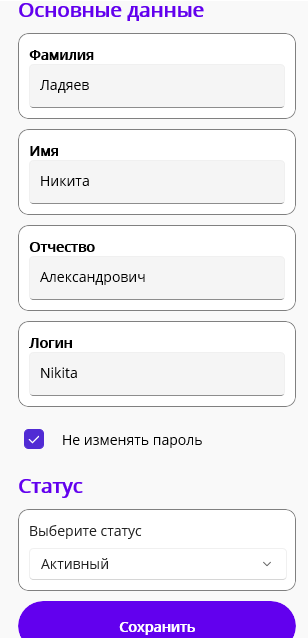


Рисунок 18 – Страница редактирования профиля курьера

**Бонус**

На этой странице (см. рисунок 19) должна отображаться подробная информация о бонусе, такая как: название, описание, награда и для какой роли предназначен. Ниже должно быть две кнопки «Изменить» и «Удалить». По нажатию на первую, должен происходить переход на страницу редактирования бонуса, а при нажатии на вторую бонус должен удаляться из базы данных.

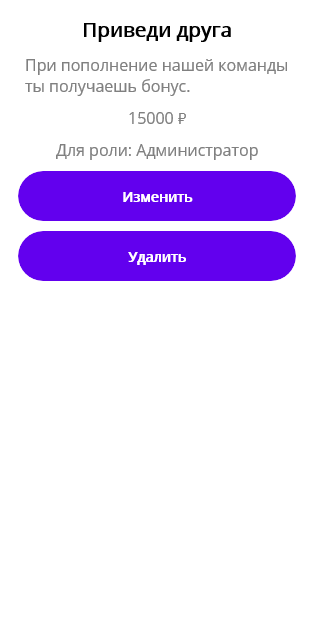


Рисунок 19 – Страница бонуса

**Редактирование бонуса**

Данная страница (см. рисунок 20) должна содержать блоки с полями для редактирования информации о бонусе, такие как: название, описание, награда и роль, для которой предназначен данный бонус. Роль должна выбираться через раскрывающийся список. Внизу должна быть кнопка «Сохранить» для сохранения изменений.

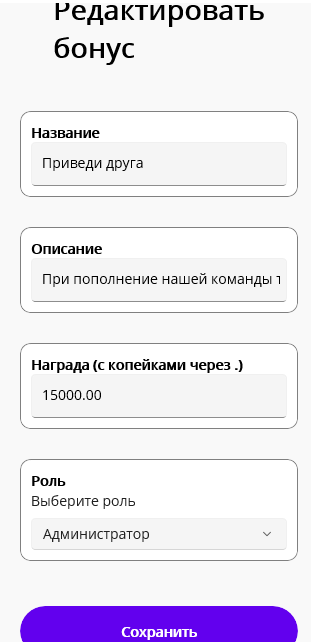


Рисунок 20 – Страница редактирования бонуса

**Создание бонуса**

На этой странице (см. рисунок 21, 22) должен присутствовать логотип приложения, содержать блоки с полями для написания информации о новом бонусе, такие как: название, описание, награда и роль, для которой предназначен данный бонус. Роль должна выбираться через раскрывающийся список. Внизу должна быть кнопка «Создать» для создания бонуса.

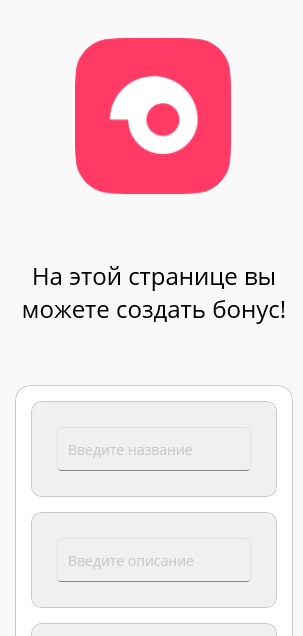


Рисунок 21 – Страница создания бонуса

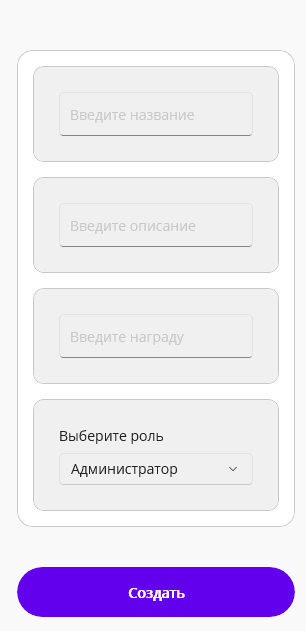


Рисунок 22 – Страница создания бонуса (2)

***Назначение штрафа***

Страница (см. рисунок 23, 24) должна содержать метку с описанием выбранного штрафа со страницы со списком штрафов. С помощью рамок должен выводиться список курьеров с информацией о них (фамилия, имя, логин), также в каждой рамке должна быть кнопка «Назначить», которая меняет штраф курьера на выбранный. В самом внизу должно быть две кнопки «Изменить» и «Удалить». По нажатию на первую должен осуществляться переход на страницу редактирования штрафа, а на вторую должно происходить удаление штрафа из базы данных.

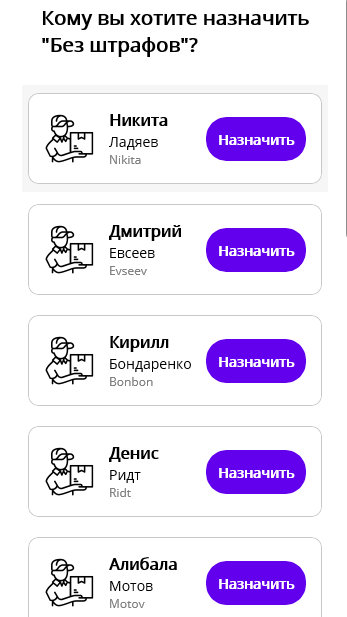


Рисунок 23 – Страница назначения штрафа

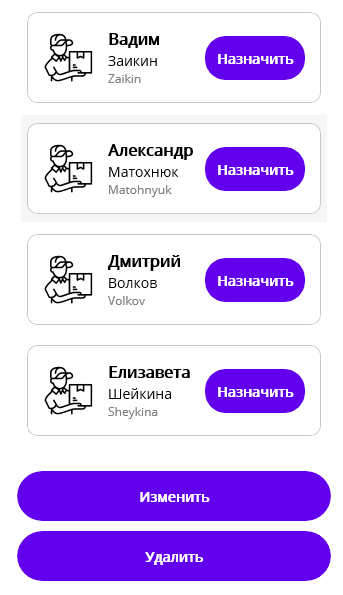


Рисунок 24 – Страница назначения штрафа (2)

***Редактирование штрафа***

Эта страница (см. рисунок 25) должна иметь блок для редактирования описания, а также кнопку «Сохранить» для сохранения изменений о штрафе.

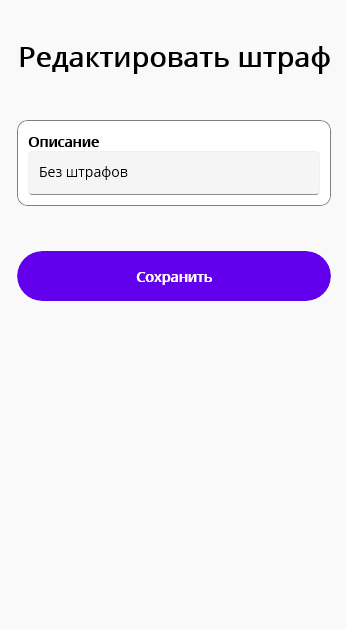


Рисунок 25 – Страница редактирования штрафа

***Создание штрафа***

На этой странице (см. рисунок 26) должен присутствовать логотип приложения, содержать блок с полем для написания описания нового штрафа. Внизу должна быть кнопка «Создать» для создания штрафа.

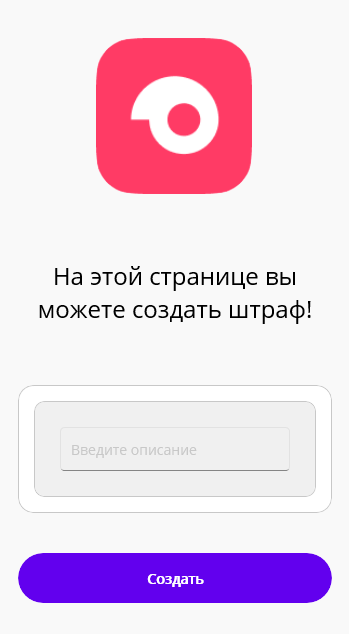


Рисунок 26 – Страница создания штрафа

# ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БАЗЫ ДАННЫХ

## Нормализация данных

Нормальная форма — свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных. Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение.

Процесс преобразования отношений базы данных к виду, отвечающему нормальным формам, называется нормализацией. Нормализация предназначена для приведения структуры БД к виду, обеспечивающему минимальную логическую избыточность, и не имеет целью уменьшение или увеличение производительности работы или же уменьшение или увеличение физического объёма базы данных. Конечной целью нормализации является уменьшение потенциальной противоречивости, хранимой в базе данных информации.

Устранение избыточности производится, как правило, за счёт декомпозиции отношений таким образом, чтобы в каждом отношении хранились только первичные факты.

Первая нормальная форма (1NF)

Переменная отношения находится в первой нормальной форме (1НФ) тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов.

Вторая нормальная форма (2NF)

Переменная отношения находится во второй нормальной форме тогда и только тогда, когда она находится в первой нормальной форме и каждый не ключевой атрибут неприводимо (функционально полно) зависит от её потенциального ключа. Функционально полная зависимость означает, что если потенциальный ключ является составным, то атрибут зависит от всего ключа и не зависит от его частей.

Третья нормальная форма (3NF)

Переменная отношения находится в третьей нормальной форме тогда и только тогда, когда она находится во второй нормальной форме, и отсутствуют транзитивные функциональные зависимости не ключевых атрибутов от ключевых. При проектировании БД третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается. Однако иногда полезно продолжить процесс нормализации.

## Концептуальная модель данных

Концептуальная модель – это отражение предметной области, для которой разрабатывается база данных. Не вдаваясь в теорию, отметим, что это некая диаграмма с принятыми обозначениями элементов. Все объекты, обозначающие вещи, обозначаются в виде прямоугольника. Атрибуты, характеризующие объект - в виде овала, а связи между объектами - ромбами. Мощность связи обозначаются стрелками (в направлении, где мощность равна многим - двойная стрелка, а со стороны, где она равна единице - одинарная).

Анализируя предметную область, можно выделить следующие сущности с их атрибутами (см. рисунок 27):

1. Сущность «Пользователи» с атрибутами «name», «surname», «patronymic», «password», «login», «token».
2. Сущность «Статусы» с атрибутами «name», «code».
3. Сущность «Штрафы» с атрибутом «description».
4. Сущность «Роль» с атрибутами «name», «code».
5. Сущность «Доступность» с атрибутами «date», «startChange», «endChange».
6. Сущность «Бонусы» с атрибутами «title», «description», «price».

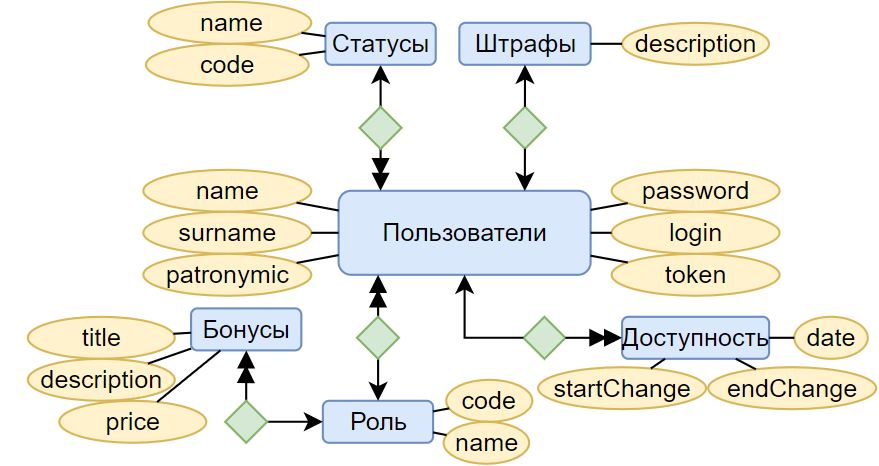


Рисунок 27 - Концептуальная модель данных

## Логическая модель данных

Логическая модель базы данных — схема базы данных, выраженная в понятиях модели данных. Этим отличается от концептуальной модели, описывающей семантику предметной области без указания технологии (конкретных методов реализации), и от физической модели, которая описывает конкретные физические механизмы, применяемые для хранения данных в накопителях.

Получившиеся логическая модель данных представлена на рисунке 28.

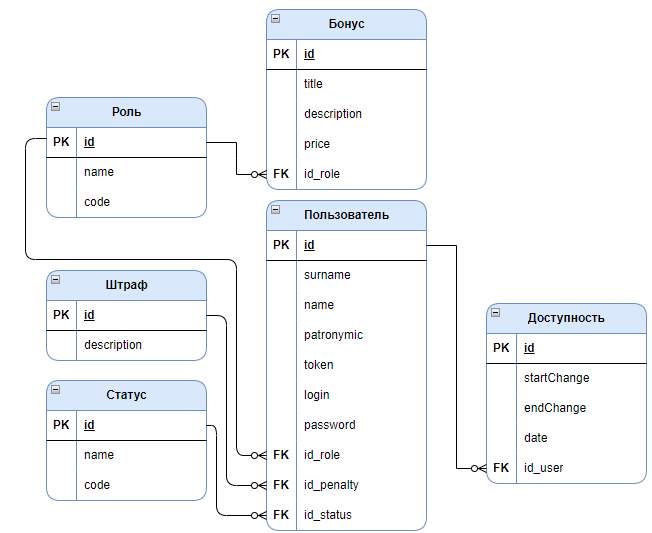


Рисунок 28 - Логическая модель данных

## Физическая модель данных

Физическая модель данных – это модель данных, описанная с помощью средств конкретной системы управления базами данных (СУБД). Физическая модель данных строится на базе даталогической путем добавления особенностей конкретной СУБД. К таким особенностям могут относиться поддерживаемые СУБД типы данных, соглашения о присвоении имен таблицам, атрибутам и т.д. Физическая модель данных фактически является готовым заданием на создание БД, имея которое можно реализовать БД в выбранной СУБД.

На рисунке 29 представленная физическая модель данных.

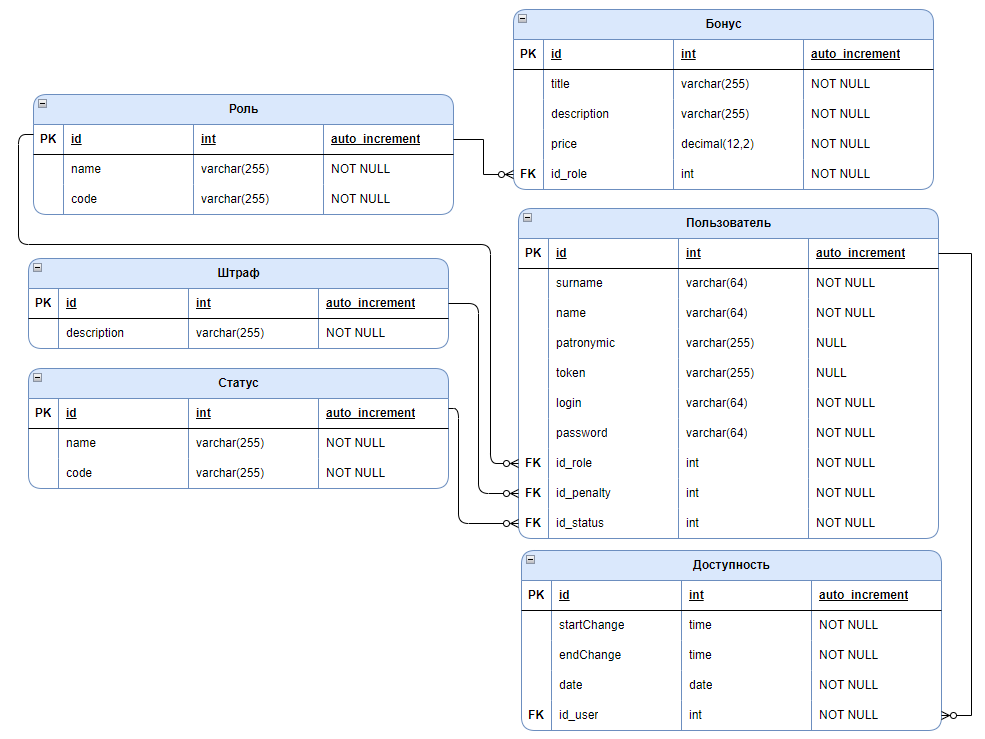


Рисунок 29 - Физическая модель данных

## Диаграмма базы данных

На рисунке 4 представлена итоговая диаграмма базы данных.

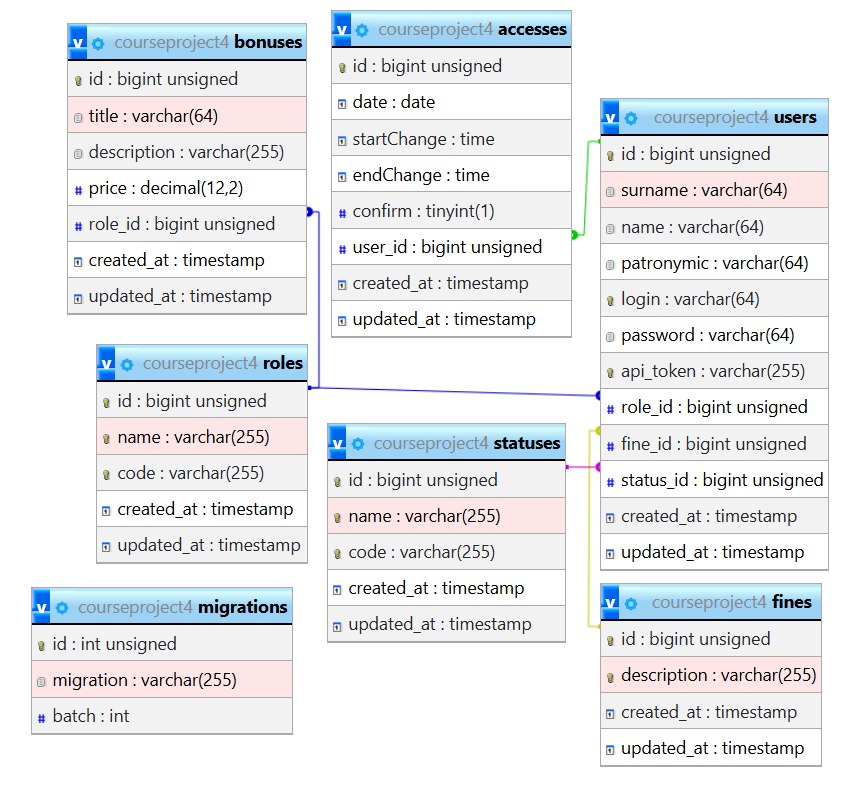


Рисунок 30 - Диаграмма базы данных

# ГЛАВА 3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ФУНКЦИОНАЛА ИНФОРМАЦИОННОГО ПРОДУКТА

## Установка и настройка фреймворка Laravel11

Для установки фреймворка требуется установка Composer: Laravel использует Composer для управления зависимостями. Можно скачать и установить Composer с официального сайта.

Установка Laravel с помощью Composer: после установки Composer перейдите в терминале в каталог, в который вы хотите установить Laravel, и выполните следующую команду: composer create-project laravel/laravel .

Где (.) – директория проекта. Composer создаст новый проект Laravel в указанном каталоге.

В корневой директории сайта создаем файл .htaccess и прописываем правила

перенаправления запросов в точку входа в приложение (в папку public).

RewriteEngine On  
RewriteRule ^(.\*)$ public/$1 [L]

Этот фрагмент кода предназначен для настройки правил перезаписи URL веб-сервера Apache.

RewriteEngine On: - Эта строка включает модуль перезаписи модуля mod\_rewrite в Apache.

RewriteRule ^(.\*)$ public/$1 [L] - Это правило перезаписи, которое перенаправляет все запросы на веб-сервере к корневому каталогу /public. Знак (.\*)? соответствует любой строке в URL-адресе, а $1 обозначает часть URL, совпавшую с выражением в скобках. [L] - Этот флаг означает “если это правило совпало, остановите обработку дальнейших правил перезаписи”. Это важно, чтобы избежать бесконечных циклов перезаписи. Таким образом, эта строка перенаправляет все запросы на сервере к файлам в каталоге /public.

Параметры в файле .env, можно увидеть на листинге 1, являются настройками подключения к базе данных MySQL для проекта Laravel:

***Листинг 1. Настройки подключения к базе данных***

DB\_CONNECTION=mysql - Этот параметр указывает, какой вы используете драйвер базы данных для соединения с базой данных.  
DB\_HOST=127.0.0.1 - Здесь указывается хост базы данных.  
DB\_PORT=3306 - Этот параметр указывает порт.  
DB\_DATABASE=samokat - Это имя базы данных.  
DB\_USERNAME=root - Это имя пользователя для доступа к базе данных MySQL.  
DB\_PASSWORD= - Это пароль для пользователя базы данных MySQL.

SESSION\_DRIVER=file - Это строка конфигурации, которая определяет, какой драйвер используется для хранения данных сессий в Laravel.

## Создание миграций

Миграции в Laravel - это способ создания и изменения структуры базы данных с использованием кода PHP вместо прямых SQL-запросов. Это помогает сохранить структуру базы данных в репозитории проекта и упростить совместную работу над базой данных.

Создаем миграции для таблиц fine, statuses, roles, bonuses, users, accesses для этого в корневой директории проекта выполняем следующие команды:

php artisan make: migration create\_fines\_table

php artisan make: migration create\_statuses\_table

php artisan make: migration create\_roles\_table

php artisan make: migration create\_bonuses\_table

php artisan make: migration create\_users\_table

php artisan make: migration create\_accesses\_table

Миграции похожи на контроль версий для вашей базы данных, позволяют вашей команде определять схемы базы данных приложения и совместно использовать их определение.

Фасад Schema обеспечивает независимую от базы данных поддержку для создания и управления таблицами во всех поддерживаемых Laravel системах баз данных. В обычной ситуации, этот фасад используется для создания и изменения таблиц / столбцов базы данных во время миграции. Итоговые миграции представлены в листинге 2.

**Листинг 2. Миграции базы данных**

Schema::create('fines', function (Blueprint $table) {  
 $table->id();  
 $table->string('description')->unique();  
 $table->timestamps();  
});

Schema::create('statuses', function (Blueprint $table) {  
 $table->id();  
 $table->string('name')->unique();  
 $table->string('code')->unique();  
 $table->timestamps();  
});

Schema::create('roles', function (Blueprint $table) {  
 $table->id();  
 $table->string('name')->unique();  
 $table->string('code')->unique();  
 $table->timestamps();  
});

Schema::create('bonuses', function (Blueprint $table) {  
 $table->id();  
 $table->string('title',64);  
 $table->string('description');  
 $table->decimal('price', 12,2);  
 $table->foreignId('role\_id')->constrained();  
 $table->timestamps();  
});

Schema::create('users', function (Blueprint $table) {  
 $table->id();  
 $table->string('surname',64);  
 $table->string('name',64);  
 $table->string('patronymic',64)->nullable();  
 $table->string('login',64)->unique();  
 $table->string('password',64);  
 $table->string('api\_token')->nullable()->unique();  
 $table->foreignId('role\_id')->constrained();  
 $table->foreignId('fine\_id')->constrained();  
 $table->foreignId('status\_id')->constrained();  
 $table->timestamps();  
});

Schema::create('accesses', function (Blueprint $table) {  
 $table->id();  
 $table->date('date');  
 $table->time('startChange');  
 $table->time('endChange');  
 $table->boolean('confirm')->default(false);  
 $table->foreignId('user\_id')->constrained()->onDelete('cascade');  
 $table->timestamps();  
});

Выполняем миграцию с помощью команды: php artisan migrate.

## Создание исключений

Во время выбрасывания исключений Laravel проверяет, есть ли в классе исключения метод render(), если да, то он использует метод этого исключения для отображения результата. Если вы не хотите полагаться на глобальную систему отлова исключений Laravel, то можете вернуть ответ в JSON напрямую из контроллера.

Illuminate\Http\Exceptions\HttpResponseException - это особое исключение, встроенное в Laravel. Особенность этого исключения заключается в том, что оно уже содержит шаблон ответа для клиента, поэтому Laravel просто возвращает ответ из этого исключения.

Созданный класс исключения ApiException расширяет базовый класс Exception. В конструкторе он принимает код и сообщение исключения, а также, при необходимости, массив ошибок. При создании объекта исключения он вызывает конструктор родительского класса Exception, передавая ему сообщение и код исключения.

Создадим особое исключение app\Exceptions\Api\ApiException.php, можно увидеть на листинге 3, которое будет определять шаблон выдаваемых пользователю ошибок (будь то ошибка авторизации или ошибка валидации входных данных, либо ошибка доступа к определенному функционалу приложения) со следующим содержимым:

Конструктор принимает три параметра:

$message: Сообщение об ошибке (строка).

$code: HTTP-статус код (по умолчанию 500).

$errors: Дополнительные ошибки (например, ошибки валидации).

Создается массив $response, который содержит:

code: Статус-код.

message: Сообщение об ошибке.

errors: Ошибки, если они переданы.

Затем вызывается конструктор родительского класса (HttpResponseException) с JSON-ответом, установленным на указанный статус-код.

***Листинг 3. Класс исключения ApiException***

class ApiException extends HttpResponseException  
{  
 public function \_\_construct(string $message, int $code = 500, $errors = [])  
 {  
 $response = [  
 'code' => $code,  
 'message' => $message,  
 ];  
 if (!empty($errors)) {  
 $response['errors'] = $errors;  
 }  
 parent::*\_\_construct*( response()->json($response)->setStatusCode($code));  
 }  
}

## Создание моделей

Модели в фреймворке Laravel играют ключевую роль в работе с данными вашего приложения, например:

* Работа с данными базы данных: Модели представляют собой объектно-ориентированный способ взаимодействия с таблицами базы данных. Каждая модель обычно связана с определенной таблицей в базе данных, и она предоставляет удобные методы для выполнения запросов к этой таблице, таких как создание, чтение, обновление и удаление записей.
* Организация бизнес-логики: Модели часто содержат бизнес-логику вашего приложения, связанную с данными. Например, методы модели могут включать валидацию данных, вычисления или другие операции, связанные с обработкой данных.
* Отношения между таблицами: В Laravel отношения между таблицами базы данных могут быть легко определены в моделях. Это позволяет вам работать с данными, которые связаны между собой, например, получать связанные записи или устанавливать связи между ними.
* Использование в контроллерах и представлениях: Модели обычно используются в контроллерах для выполнения операций с данными, а также в представлениях для отображения этих данных. Они обеспечивают удобный способ доступа к данным вашего приложения из любой его части.
* Тестирование: Использование моделей позволяет упростить процесс тестирования вашего приложения. Вы можете создавать экземпляры моделей в тестах и проверять их поведение в различных сценариях.

Чтобы сгенерировать новую модель Eloquent, необходимо использовать команду:

php artisan make:model.

Создадим модели ко всем таблицам нашей базы данных, для этого поочередно выполняем следующие команды:

php artisan make:model Access

php artisan make:model Bonus

php artisan make:model Fine

php artisan make:model Role

php artisan make:model Status

php artisan make:model User

Далее отредактируем наши модели, можно увидеть на листинге 4, указав связи:

***Листинг 4. Модели базы данных***

class Access extends Model  
{  
 protected $fillable = ['startChange', 'endChange','date','confirm','user\_id'];  
  
 public function user(){  
 return $this->belongsTo(User::class);  
 }  
}

class Bonus extends Model  
{  
 protected $fillable = ['title', 'description','price', 'role\_id'];  
 public function role()  
 {  
 return $this->belongsTo(Role::class);  
 }  
}

class Fine extends Model  
{  
 protected $fillable = ['description'];  
 public function users(){  
 return $this->hasMany(User::class);  
 }  
}

class Role extends Model  
{  
 protected $fillable = ['name', 'code'];  
 public function users(){  
 return $this->hasMany(User::class);  
 }  
 public function bonuses(){  
 return $this->hasMany(Bonus::class);  
 }  
}

class Status extends Model  
{  
 protected $fillable = ['name', 'code'];  
 public function users(){  
 return $this->hasMany(User::class);  
 }  
}

class User extends *Authenticatable*{  
 use HasApiTokens, HasFactory, Notifiable;  
  
 protected $fillable = [  
 'surname','name','patronymic','login','password','api\_token','role\_id','fine\_id','status\_id',  
 ];  
 protected $hidden = [  
 'password',  
 'api\_token',  
 ];  
 protected function casts(): array  
 {  
 return [  
 'password' => 'hashed',  
 ];  
 }  
 public function role()  
 {  
 return $this->belongsTo(Role::class);  
 }  
 public function fine()  
 {  
 return $this->belongsTo(Fine::class);  
 }  
 public function accesses()  
 {  
 return $this->hasMany(Access::class);  
 }  
 public function status()  
 {  
 return $this->belongsTo(Status::class);  
 }  
}

Класс User содержит поля $fillable и представляет собой модель пользователя в Laravel, которая использует трейты HasApiTokens, HasFactory и Notifiable и связана с моделями Role, Fine, Access, Status.

С помощью $hidden определяются, атрибуты, которые должны быть скрыты при сериализации модели в JSON.

С помощью метода casts определяются, атрибуты модели, которые должны быть приведены к определенным типам. В вашем случае поле password автоматически кэшируется при сохранении в базу данных.

## Создание контроллеров

В фреймворке Laravel контроллеры играют важную роль в организации логики приложения. Они обрабатывают HTTP-запросы и возвращают соответствующие HTTP-ответы.

Выполним генерация контроллеров, необходимых для решения технического задания, для этого выполним следующие команды:

php artisan make:controller AuthController

php artisan make:controller AccesseController

php artisan make:controller BonusController

php artisan make:controller FineController

php artisan make:controller StatusController

php artisan make:controller UserController

Далее отредактируем наши котроллеры, можно увидеть на листингах 5, 6, 7, 8, 9 и 10, написав необходимые методы:

***Листинг 5. Контроллер AccesseController для доступностей***

class AccesseController extends Controller  
{  
 // Метод для получения всех доступностей курьеров в пределах двух недель  
 public function index()  
 {  
 // Определяем диапазон дат  
 $startDate = Carbon::*now*()->subWeeks(2)->startOfDay(); // Две недели назад  
 $endDate = Carbon::*now*()->addWeeks(2)->endOfDay(); // Две недели вперед  
  
 // Фильтруем доступности по диапазону дат  
 $accesses = Access::*whereBetween*('date', [$startDate, $endDate])->get();  
  
 if ($accesses->isEmpty()) {  
 return response()->json(['message' => 'Доступности не найдены'], 404);  
 }  
  
 return response()->json($accesses)->setStatusCode(200);  
 }  
 // Метод для получения своих выставленных доступностей (используется курьерами)  
 public function indexCourier()  
 {  
 // Получаем ID текущего пользователя  
 $userId = Auth::id();  
  
 // Определяем диапазон дат  
 $startDate = Carbon::*now*()->subWeeks(2)->startOfDay(); // Две недели назад  
 $endDate = Carbon::*now*()->addWeeks(2)->endOfDay(); // Две недели вперед  
  
 // Фильтруем доступности текущего пользователя по диапазону дат  
 $accesses = Access::*where*('user\_id', $userId)  
 ->whereBetween('date', [$startDate, $endDate])  
 ->get();  
  
 return response()->json($accesses);  
 }  
 // Метод для просмотра конкретных доступностей  
 public function show($id)  
 {  
 $access = Access::*find*($id);  
  
 if ($access){  
 return response()->json($access)->setStatusCode(200, 'Успешно');  
 }else{  
 return response()->json('Доступность не найдена')->setStatusCode(404, 'Не найдено');  
 }  
 }  
 // Метод для создания доступности (используется курьерами)  
 public function store(CreateAccesseRequest $request){  
  
 if(Auth::user()->role->code != 'сourier'){  
  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
  
 $IdUser = Auth::user()->id;  
  
 $access = new Access([  
 'date' => $request->input('date'),  
 'startChange' => $request->input('startChange'),  
 'endChange' => $request->input('endChange'),  
 'user\_id' => $IdUser,  
 ]);  
  
 $access->save();  
 return response()->json($access)->setStatusCode(200);  
 }  
 // Метод для обновления доступности (используется курьерами)  
 public function update(UpdateAccesseRequest $request, $id)  
 {  
 if(Auth::user()->role->code != 'сourier'){  
  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
  
 $accesse = Access::*find*($id);  
  
 if ($accesse){  
 $accesse->update($request->all());  
 return response()->json($accesse)->setStatusCode(200, 'Успешно');  
 }else{  
 return response()->json('Доступность не найдена')->setStatusCode(404, 'Не найдено');  
 }  
 }  
 // Метод для удаления доступности (используется курьерами)  
 public function destroy($id)  
 {  
 if(Auth::user()->role->code != 'сourier'){  
  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
  
 $accesse = Access::*find*($id);  
 if (!$accesse){  
 throw new ApiException('Не найдено', 404);  
 }  
  
 if ($accesse->confirm == 1){  
 return response()->json(['message' => 'Нельзя удалить подтвержденную доступность'], 403);  
 }else{  
 $accesse->delete();  
 return response()->json('Доступность удалена')->setStatusCode(200);  
 }  
 }  
}

***Листинг 6. Контроллер AuthController для авторизации***

class AuthController extends Controller  
{  
 // Метод для регистрации курьеров  
 public function register(RegisterRequest $request)  
 {  
 if(Auth::user()->role->code != 'admin'){  
  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
  
 $role\_id = Role::*where*('code', 'сourier')->first()->id;  
 $fine\_id = Fine::*where*('description', 'Без штрафов')->first()->id;  
 $status\_id = Status::*where*('code', 'active')->first()->id;  
 $validated = $request->validated();  
  
 $user = User::*create*([  
 ...$validated,  
 'role\_id' => $role\_id,  
 'fine\_id' => $fine\_id,  
 'status\_id' => $status\_id  
 ]);  
  
 $user->api\_token = Hash::make(Str::*random*(60));  
 $user->save();  
  
 return response()->json(  
 [  
 'user' => $user,  
 'token' => $user->api\_token,  
 ]  
 )->setStatusCode(201);  
 }  
 // Метод для входа в систему  
 public function login(Request $request){  
  
 if (!Auth::attempt($request->only('login', 'password'))) {  
 throw new ApiException("Неверный логин/пароль",401);  
 }  
  
 $user = Auth::user();  
 $user->api\_token = Hash::make(Str::*random*(60));  
 $user->save();  
 return response()->json([  
 'token' => $user->api\_token,  
 'user' => $user,  
 ])->setStatusCode(200);  
 }  
 // Метод для выхода из системы  
 public function logout(){  
 $user = Auth::user();  
 $user->api\_token = null;  
 $user->save();  
 return response()->json(['message' => 'Вы вышли из системы'])->setStatusCode(200);  
 }  
}

***Листинг 7. Контроллер BonusController для бонусов***

class BonusController extends Controller  
{  
 // Метод для просмотра всех бонусов  
 public function index()  
 {  
 $bonuses = Bonus::*with*('role')->get();  
 if ($bonuses->isEmpty()) {  
 throw new ApiException( 'Не найдено', 404);  
 }  
 return response()->json($bonuses)->setStatusCode(200);  
 }  
 // Метод для создания бонусов  
 public function store(CreateBonusRequest $request)  
 {  
 if(Auth::user()->role->code != 'admin'){  
  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 $bonus = new Bonus($request->all());  
 $bonus->save();  
 return response()->json($bonus)->setStatusCode(201);  
 }  
 // Метод для просмотра конкретных бонусов  
 public function show($id)  
 {  
 $bonus = Bonus::*with*('role')->find($id);  
 if ($bonus) {  
 return response()->json($bonus)->setStatusCode(200, 'Успешно');  
 } else {  
 return response()->json('Бонус не найден')->setStatusCode(404, 'Не найдено');  
 }  
 }  
 // Метод для обновления бонусов  
 public function update(UpdateBonusRequest $request, $id)  
 {  
 if(Auth::user()->role->code != 'admin'){  
  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 $bonus = Bonus::*find*($id);  
 if ($bonus){  
 $bonus->update($request->all());  
 return response()->json($bonus)->setStatusCode(200, 'Успешно');  
 }else{  
 return response()->json('Бонус не найден')->setStatusCode(404, 'Не найдено');  
 }  
 }  
 // Метод для удаления бонусов  
 public function destroy($id)  
 {  
 if(Auth::user()->role->code != 'admin'){  
  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 $bonus = Bonus::*find*($id);  
 if (!$bonus){  
 throw new ApiException('Не найдено', 404);  
 }  
 $bonus->delete();  
 return response()->json('Бонус удален')->setStatusCode(200);  
 }  
}

***Листинг 8. Контроллер FineController для штрафов***

class FineController extends Controller  
{  
 // Метод для просмотра всех штрафов  
 public function index()  
 {  
 $fine = Fine::*all*();  
 if (!$fine){  
 throw new ApiException('Не найдено', 404);  
 }  
 return response()->json($fine)->setStatusCode(200);  
 }  
 // Метод для создания штрафа  
 public function store(CreateFineRequest $request)  
 {  
 if(Auth::user()->role->code != 'admin'){  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 $fine = new Fine($request->all());  
 $fine->save();  
 return response()->json($fine)->setStatusCode(201);  
 }  
 // Метод для просмотра конкретного штрафа  
 public function show($id)  
 {  
 $fine = Fine::*find*($id);  
 if ($fine){  
 return response()->json($fine)->setStatusCode(200, 'Успешно');  
 }else{  
 return response()->json('Штраф не найден')->setStatusCode(404, 'Не найдено');  
 }  
 }  
 // Метод для обновления штрафа  
 public function update(UpdateFineRequest $request, $id)  
 {  
 if(Auth::user()->role->code != 'admin'){  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 $fine = Fine::*find*($id);  
 if ($fine){  
 $fine->update($request->all());  
 return response()->json($fine)->setStatusCode(200, 'Успешно');  
 }else{  
 return response()->json('Штраф не найден')->setStatusCode(404, 'Не найдено');  
 }  
 }  
 // Метод для удаления штрафа  
 public function destroy($id)  
 {  
 if(Auth::user()->role->code != 'admin'){  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 $fine = Fine::*find*($id);  
 if (!$fine){  
 throw new ApiException('Не найдено', 404);  
 }  
 $fine->delete();  
 return response()->json('Штраф удален')->setStatusCode(200);  
 }  
}

***Листинг 9. Контроллер StatusController для статусов***

class StatusController extends Controller  
{  
 // Метод для просмотра всех статусов  
 public function index()  
 {  
 $status = Status::*all*();  
 if (!$status){  
 throw new ApiException('Не найдено', 404);  
 }  
 return response()->json($status)->setStatusCode(200);  
 }  
 // Метод для создания статусов  
 public function store(CreateStatusRequest $request)  
 {  
 if(Auth::user()->role->code != 'admin'){  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 $status = new Status($request->all());  
 $status->save();  
 return response()->json($status)->setStatusCode(201);  
 }  
 // Метод для просмотра конкретного статуса  
 public function show($id)  
 {  
 $status = Status::*find*($id);  
 if ($status){  
 return response()->json($status)->setStatusCode(200, 'Успешно');  
 }else{  
 return response()->json('Статус не найден')->setStatusCode(404, 'Не найдено');  
 }  
 }  
 // Метод для обновления статуса  
 public function update(UpdateStatusRequest $request, $id)  
 {  
 if(Auth::user()->role->code != 'admin'){  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 $status = Status::*find*($id);  
 if ($status){  
 $status->update($request->all());  
 return response()->json($status)->setStatusCode(200, 'Успешно');  
 }else{  
 return response()->json('Статус не найден')->setStatusCode(404, 'Не найдено');  
 }  
 }  
 // Метод для удаления статусов  
 public function destroy(string $id)  
 {  
 if(Auth::user()->role->code != 'admin'){  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 $status = Status::*find*($id);  
 if (!$status){  
 throw new ApiException('Не найдено', 404);  
 }  
 $status->delete();  
 return response()->json('Статус удален')->setStatusCode(200);  
 }  
}

***Листинг 10. Контроллер UserController для пользователей***

class UserController extends Controller  
{  
 // Метод для просмотра пользователей  
 public function index(){  
 $users = User::*all*();  
 return response()->json($users)->setStatusCode(200,'Ок');  
 }  
 // Метод для просмотра конкретного пользователя  
 public function show($id)  
 {  
 $user = User::*find*($id);  
 if ($user){  
 return response()->json($user)->setStatusCode(200, 'Успешно');  
 }else{  
 return response()->json('Пользователь не найден')->setStatusCode(404, 'Не найдено');  
 }  
 }  
 // Метод для обновления пользователя  
 public function update(UserUpdateRequest $request, $id, User $user){  
 $user = User::*find*($id);  
 if ($user){  
 $user->update($request->all());  
 return response()->json($user)->setStatusCode(200, 'Ok');  
 }else{  
 return response()->json('Пользователь не найден')->setStatusCode(404, 'Не найдено');  
 }  
 }  
 // Метод для удаления пользователя  
 public function destroy($id)  
 {  
 if(Auth::user()->role->code != 'admin'){  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 $user = User::*find*($id);  
 if (!$user){  
 throw new ApiException('Не найдено', 404);  
 }  
 $user->delete();  
 return response()->json('Пользователь удален')->setStatusCode(200);  
 }  
}

Для полного/частичного подтверждения/отмены доступностей был создан отдельный контроллер, можно увидеть на листинге 11:

***Листинг 11. Контроллер ConfirmController для подтверждения/отмен доступностей***

class ConfirmController extends Controller  
{  
 // Метод для полного подтверждения доступности  
 public function confirm(Request $request, $id)  
 {  
 // Проверка прав доступа  
 if (Auth::user()->role->code !== 'admin') {  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 // Поиск доступности по ID  
 $access = Access::*find*($id);  
 if (!$access) {  
 return response()->json(['message' => 'Доступность не найдена'], 404);  
 }  
 // Обновляем значение confirm на true  
 $access->update([  
 'confirm' => true,  
 ]);  
 // Объединяем соседние записи  
 $this->mergeAdjacentAccesses($access);  
 return response()->json([  
 'message' => 'Доступность успешно подтверждена',  
 'access' => $access,  
 ]);  
 }  
 // Метод для полной отмены доступности  
 public function cancel(Request $request, $id)  
 {  
 // Проверка роли пользователя  
 if (Auth::user()->role->code !== 'admin') {  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 // Поиск доступности по ID  
 $access = Access::*find*($id);  
 if (!$access) {  
 return response()->json(['message' => 'Доступность не найдена'], 404);  
 }  
 // Логика отмены (сбрасываем подтверждение)  
 $access->update([  
 'confirm' => false, // Отменяем подтверждение  
 ]);  
 // Объединяем соседние записи  
 $this->mergeAdjacentAccesses($access);  
 return response()->json([  
 'message' => 'Доступность успешно отменена',  
 'access' => $access,  
 ]);  
 }  
 // Метод для частичного подтверждения доступности  
 public function partialConfirm(Request $request, $id)  
 {  
 // Проверка прав доступа  
 if (Auth::user()->role->code !== 'admin') {  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 // Валидация входных данных  
 $validator = Validator::make($request->all(), [  
 'startChange' => 'required|date\_format:H', // Время в формате HH  
 'endChange' => 'required|date\_format:H|after:startChange', // Время в формате HH и после startChange  
 ]);  
 if ($validator->fails()) {  
 return response()->json(['message' => 'Ошибка валидации', 'errors' => $validator->errors()], 422);  
 }  
 // Поиск записи по ID  
 $access = Access::*find*($id);  
 if (!$access) {  
 return response()->json(['message' => 'Доступность не найдена'], 404);  
 }  
 // Проверяем, что запись ещё не подтверждена  
 if ($access->confirm) {  
 return response()->json(['message' => 'Нельзя подтвердить уже подтверждённую запись'], 422);  
 }  
 // Получаем временные интервалы из запроса  
 $startTime = $request->input('startChange') . ':00:00';  
 $endTime = $request->input('endChange') . ':00:00';  
 // Проверяем, что выбранный интервал находится внутри текущего интервала доступности  
 if ($startTime < $access->startChange || $endTime > $access->endChange) {  
 return response()->json(['message' => 'Выбранный интервал выходит за пределы текущей доступности'], 422);  
 }  
 // Создаем новую запись с подтверждённым статусом  
 $newAccess = $access->replicate();  
 $newAccess->startChange = $startTime;  
 $newAccess->endChange = $endTime;  
 $newAccess->confirm = true;  
 $newAccess->save();  
 // Обновляем исходную запись, чтобы отразить оставшуюся часть доступности  
 if ($startTime == $access->startChange && $endTime == $access->endChange) {  
 // Если выбран полный интервал, удаляем исходную запись  
 $access->delete();  
 } elseif ($startTime == $access->startChange) {  
 // Если начало совпадает, обновляем только конец  
 $access->startChange = $endTime;  
 $access->save();  
 } elseif ($endTime == $access->endChange) {  
 // Если конец совпадает, обновляем только начало  
 $access->endChange = $startTime;  
 $access->save();  
 } else {  
 // Если выбранный интервал находится в середине, создаем две новые записи  
 $firstPart = $access->replicate();  
 $firstPart->endChange = $startTime;  
 $firstPart->save();  
 $secondPart = $access->replicate();  
 $secondPart->startChange = $endTime;  
 $secondPart->save();  
 // Удаляем исходную запись  
 $access->delete();  
 }  
 // Объединяем соседние записи  
 $this->mergeAdjacentAccesses($newAccess);  
 return response()->json([  
 'message' => 'Доступность успешно частично подтверждена',  
 'new\_access' => $newAccess,  
 'updated\_access' => $access ?: null, // Возвращаем null, если исходная запись была удалена  
 ]);  
 }  
 // Метод для частичной отмены доступности  
 public function partialCancel(Request $request, $id)  
 {  
 // Проверка прав доступа  
 if (Auth::user()->role->code !== 'admin') {  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 // Валидация входных данных  
 $validator = Validator::make($request->all(), [  
 'startChange' => 'required|date\_format:H', // Время в формате H  
 'endChange' => 'required|date\_format:H|after:startChange', // Время в формате H и после startChange  
 ]);  
 if ($validator->fails()) {  
 return response()->json(['message' => 'Ошибка валидации', 'errors' => $validator->errors()], 422);  
 }  
 // Поиск доступности по ID  
 $access = Access::*find*($id);  
 if (!$access) {  
 return response()->json(['message' => 'Доступность не найдена'], 404);  
 }  
 // Проверяем, что запись ещё не отменена  
 if (!$access->confirm) {  
 return response()->json(['message' => 'Нельзя отменить уже неподтверждённую запись'], 422);  
 }  
 // Получаем временные интервалы из запроса  
 $startTime = $request->input('startChange') . ':00:00';  
 $endTime = $request->input('endChange') . ':00:00';  
 // Проверяем, что выбранный интервал находится внутри текущего интервала доступности  
 if ($startTime < $access->startChange || $endTime > $access->endChange) {  
 return response()->json(['message' => 'Выбранный интервал выходит за пределы текущей доступности'], 422);  
 }  
 // Создаем новую запись с не подтвержденным статусом  
 $newAccess = $access->replicate();  
 $newAccess->startChange = $startTime;  
 $newAccess->endChange = $endTime;  
 $newAccess->confirm = false;  
 $newAccess->save();  
 // Обновляем исходную запись, чтобы отразить оставшуюся часть доступности  
 if ($startTime == $access->startChange && $endTime == $access->endChange) {  
 // Если выбран полный интервал, удаляем исходную запись  
 $access->delete();  
 } elseif ($startTime == $access->startChange) {  
 // Если начало совпадает, обновляем только конец  
 $access->startChange = $endTime;  
 $access->save();  
 } elseif ($endTime == $access->endChange) {  
 // Если конец совпадает, обновляем только начало  
 $access->endChange = $startTime;  
 $access->save();  
 } else {  
 // Если выбранный интервал находится в середине, создаем две новые записи  
 $firstPart = $access->replicate();  
 $firstPart->endChange = $startTime;  
 $firstPart->save();  
 $secondPart = $access->replicate();  
 $secondPart->startChange = $endTime;  
 $secondPart->save();  
 // Удаляем исходную запись  
 $access->delete();  
 }  
 // Объединяем соседние записи  
 $this->mergeAdjacentAccesses($newAccess);  
 return response()->json([  
 'message' => 'Доступность успешно частично отменена',  
 'new\_access' => $newAccess,  
 'updated\_access' => $access ?: null, // Возвращаем null, если исходная запись была удалена  
 ]);  
 }  
 // Метод для объединения соседних доступностей по времени и пользователю  
 private function mergeAdjacentAccesses($access)  
 {  
 // Поиск соседних записей  
 $adjacentAccesses = Access::*where*('user\_id', $access->user\_id)  
 ->where('date', $access->date)  
 ->where('confirm', $access->confirm) // Учитываем только записи с одинаковым статусом  
 ->where(function ($query) use ($access) {  
 $query->where(function ($q) use ($access) {  
 $q->where('endChange', $access->startChange);  
 })->orWhere(function ($q) use ($access) {  
 $q->where('startChange', $access->endChange);  
 });  
 })  
 ->get();  
 // Объединяем записи  
 foreach ($adjacentAccesses as $adjacentAccess) {  
 // Обновляем интервал текущей записи  
 $access->startChange = min($access->startChange, $adjacentAccess->startChange);  
 $access->endChange = max($access->endChange, $adjacentAccess->endChange);  
 $access->save();  
 // Удаляем соседнюю запись  
 $adjacentAccess->delete();  
 }  
 // Рекурсивно вызываем метод, чтобы объединить все возможные соседние записи  
 if ($adjacentAccesses->count() > 0) {  
 $this->mergeAdjacentAccesses($access);  
 }  
 }  
}

## Валидация запроса формы

В Laravel валидация запроса формы обычно выполняется в контроллере, используя встроенные инструменты валидации.

Form Request - это класс где мы описываем правила валидации для входящих данных. Обычно класс содержит набор правил под запрос из клиента. Мы можем его декларировать в контроллере, и через контейнер в Laravel он автоматически проверит данные на соответствии нашим правилам и через внутренние механизмы фреймворка выдаст ответ клиенту.

Для более сложных сценариев валидации вы можете создать «запрос формы». Запрос формы – это ваш класс запроса, который инкапсулирует свою собственную логику валидации и авторизации. Сгенерируем родительский запрос формы ApiRequest c методами непрошедшей авторизации и непрошедшей валидации.

Класс ApiRequest, можно увидеть на листинге 12, расширяет FormRequest и переопределяет два метода: failedValidation и failedAuthorization.

***Листинг 12. Класс ApiRequest c методами непрошедшей авторизации и непрошедшей валидации***

class ApiRequest extends FormRequest  
{  
 //Вызов исключения при провале валидации данных  
 public function failedValidation(Validator $validator)  
 {  
 throw new ApiException('Ошибка валидации данных', 422, $validator->errors());  
 }  
 //Вызов исключения при провале авторизации  
 public function failedAuthorization()  
 {  
 throw new ApiException('Ошибка доступа пользователя', 403);  
 }  
}

Класс RegisterRequest, виден на листинге 13, расширяет ApiRequest и определяет правила валидации для регистрации пользователя.

***Листинг 13. Класс RegisterRequest с правилами валидации для регистрации пользователя***

class RegisterRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules(): array  
 {  
 return [  
 'name' => 'required|string|max:64|min:3',  
 'surname' => 'required|string|max:64|min:3',  
 'patronymic' => 'nullable|string|max:64|min:3',  
 'login' => 'required|string|max:64|unique:users,login,',  
 'password' => 'required|string|max:64|regex:/^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$/'  
 ];  
 }  
 public function messages(): array{  
 return [  
 'name.required' => 'Поле "Имя" обязательно для заполнения.',  
 'name.max' => 'Поле "Имя" не должно превышать 64 символа.',  
 'name.min' => 'Поле "Имя" не должно быть меньше 3 символов.',  
  
 'surname.required' => 'Поле "Фамилия" обязательно для заполнения.',  
 'surname.max' => 'Поле "Фамилия" не должно превышать 64 символа.',  
 'surname.min' => 'Поле "Фамилия" не должно быть меньше 3 символов.',  
  
 'patronymic.max' => 'Поле "Отчество" не должно превышать 64 символа.',  
 'patronymic.min' => 'Поле "Отчество" не должно быть меньше 3 символов.',  
  
 'login.required' => 'Поле "Логин" обязательно для заполнения.',  
 'login.max' => 'Поле "Логин" не должно превышать 64 символа.',  
 'login.unique' => 'Такой "Логин" уже существует.',  
  
 'password.required' => 'Поле "Пароль" обязательно для заполнения.',  
 'password.max' => 'Поле "Пароль" не должно превышать 64 символа.',  
 'password.regex' => 'Пароль должен содержать как минимум одну заглавную букву, одну строчную букву, одну цифру и один специальный символ (@$!%\*?&).',  
 ];  
 }  
}

Класс UserUpdateRequest, можно увидеть на листинге 14, расширяет ApiRequest и определяет правила валидации для редактирования пользователя.

***Листинг 14. Класс UserUpdateRequest с правилами валидации для редактирования пользователя***

class UserUpdateRequest extends FormRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules(): array  
 {  
 return [  
 'name' => 'string|max:64|min:3',  
 'surname' => 'string|max:64|min:3',  
 'patronymic' => 'nullable|string|max:64|min:3',  
 'login' => 'string|max:64|unique:users,login,' . $this->route('id'),  
 'password' => 'string|max:64|regex:/^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$/',  
 'fine\_id' => 'integer|exists:fines,id',  
 'status\_id' => 'integer|exists:statuses,id',  
 ];  
 }  
 public function messages(): array{  
 return [  
 'name.max' => 'Поле "Имя" не должно превышать 64 символа.',  
 'name.min' => 'Поле "Имя" не должно быть меньше 3 символов.',  
  
 'surname.max' => 'Поле "Фамилия" не должно превышать 64 символа.',  
 'surname.min' => 'Поле "Фамилия" не должно быть меньше 3 символов.',  
  
 'patronymic.max' => 'Поле "Отчество" не должно превышать 64 символа.',  
 'patronymic.min' => 'Поле "Отчество" не должно быть меньше 3 символов.',  
  
 'login.max' => 'Поле "Логин" не должно превышать 64 символа.',  
 'login.unique' => 'Такой "Логин" уже существует.',  
  
 'password.max' => 'Поле "Пароль" не должно превышать 64 символа.',  
 'password.regex' => 'Пароль должен содержать как минимум одну заглавную букву, одну строчную букву, одну цифру и один специальный символ (@$!%\*?&).',  
 ];  
 }  
}

Класс CreateAccesseRequest, видно на листинге 15, расширяет ApiRequest и определяет правила валидации для создания доступности.

***Листинг 15. Класс CreateAccesseRequest с правилами валидации для создания доступности***

class CreateAccesseRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules(): array  
 {  
 return [  
 'date' => 'required|date',  
 'startChange' => 'required|date\_format:H:i',  
 'endChange' => 'required|date\_format:H:i',  
 ];  
 }  
 public function messages(): array  
 {  
 return [  
 'date.required' => 'Пожалуйста, укажите дату.',  
 'date.date' => 'Дата должна быть в формате ГГГГ-ММ-ДД.',  
 'startChange.required' => 'Пожалуйста, укажите время начала смены.',  
 'endChange.required' => 'Пожалуйста, укажите время окончания смены.',  
 ];  
 }  
}

Класс UpdateAccesseRequest, виден на листинге 16, расширяет ApiRequest и определяет правила валидации для редактирования доступности.

***Листинг 16. Класс UpdateAccesseRequest с правилами валидации для редактирования доступности***

class UpdateAccesseRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules(): array  
 {  
 return [  
 'date' => 'date',  
 'startChange' => 'date\_format:H:i',  
 'endChange' => 'date\_format:H:i',  
 ];  
 }  
 public function messages(): array  
 {  
 return [  
 'date.date' => 'Дата должна быть в формате ГГГГ-ММ-ДД.',  
 'startChange.date\_format' => 'Время начала смены должно быть в формате ЧЧ.',  
 'endChange.date\_format' => 'Время окончания смены должно быть в формате ЧЧ.',  
  
 ];  
 }  
}

Класс CreateBonusRequest, можно увидеть на листинге 17, расширяет ApiRequest и определяет правила валидации для создания бонуса.

***Листинг 17. Класс CreateBonusRequest с правилами валидации для создания бонуса***

class CreateBonusRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules(): array  
 {  
 return [  
 'title' => 'required|string|max:64',  
 'description' => 'required|string',  
 'price' => 'required|numeric|min:0',  
 'role\_id' => 'required|integer|exists:roles,id',  
 ];  
 }  
 public function messages(): array {  
 return [  
 'title.required' => 'Заголовок бонуса не должен быть пустым',  
 'title.max' => 'Заголовок бонуса должен содержать максимум 64 символа',  
  
 'description.required' => 'Описание бонуса не должен быть пустым',  
  
 'price.required' => 'Цена бонуса не должна быть пустой',  
 'price.numeric' => 'Цена бонуса должна быть числом',  
 'price.min' => 'Вознаграждение от бонуса не должно быть 0',  
  
 'role\_id' => 'Роль должна быть выбрана'  
 ];  
 }  
}

Класс UpdateBonusRequest, можно увидеть на листинге 18, расширяет ApiRequest и определяет правила валидации для редактирования доступности.

***Листинг 18. Класс UpdateBonusRequest с правилами валидации для редактирования бонуса***

class UpdateBonusRequest extends ApiRequest  
{  
 public function rules(): array  
 {  
 return [  
 'title' => 'string|max:64',  
 'description' => 'string',  
 'price' => 'numeric|min:0',  
 'role\_id' => 'integer|exists:roles,id'  
 ];  
 }  
 public function messages(): array {  
 return [  
 'title.max' => 'Заголовок бонуса должен содержать максимум 64 символа',  
 'price.numeric' => 'Цена бонуса должна быть числом',  
 'price.min' => 'Вознаграждение от бонуса не должно быть 0',  
 'role\_id' => 'Роль должна быть выбрана',  
 ];  
 }  
}

Класс CreateFineRequest, как видно на листинге 19, расширяет ApiRequest и определяет правила валидации для создания штрафа.

***Листинг 19. Класс CreateFineRequest с правилами валидации для создания штрафа***

class CreateFineRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules(): array  
 {  
 return [  
 'description' => 'required|string|max:255',  
 ];  
 }  
 public function messages(): array{  
 return [  
 'description.required' => 'Поле "описание" обязательно для заполнения.',  
 'description.max' => 'Поле "описание" должно содержать максимум 255 символов',  
 ];  
 }  
}

Класс UpdateFineRequest, можно увидеть на листинге 20, расширяет ApiRequest и определяет правила валидации для редактирования штрафа.

***Листинг 20. Класс UpdateFineRequest с правилами валидации для редактирования штрафа***

class UpdateFineRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules(): array  
 {  
 return [  
 'description' => 'string|max:255',  
 ];  
 }  
 public function messages(): array{  
 return [  
 'description.max' => 'Поле "описание" должно содержать максимум 255 символов',  
 ];  
 }  
}

Класс CreateStatusRequest, видно на листинге 21, расширяет ApiRequest и определяет правила валидации для создания статуса.

***Листинг 21. Класс CreateStatusRequest с правилами валидации для создания статуса***

class CreateStatusRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules(): array  
 {  
 return [  
 'name' => 'required|string|max:64|unique:statuses,name',  
 'code' => 'required|string|max:64|unique:statuses,code',  
 ];  
 }  
 public function messages(): array  
 {  
 return [  
 'name.required' => 'Поле "Название" обязательно для заполнения.',  
 'name.max' => 'Поле "Название" не должно превышать 64 символа.',  
 'name.unique' => 'Такое "Название" уже существует.',  
  
 'code.required' => 'Поле "Код" обязательно для заполнения.',  
 'code.max' => 'Поле "Код" не должно превышать 64 символа.',  
 'code.unique' => 'Такой "Код" уже существует.',  
 ];  
 }  
}

Класс UpdateStatusRequest, можно увидеть на листинге 22, расширяет ApiRequest и определяет правила валидации для редактирования статуса.

***Листинг 22. Класс UpdateStatusRequest с правилами валидации для редактирования статуса***

class UpdateStatusRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules(): array  
 {  
 return [  
 'name' => 'max:64|min:5|unique:statuses,name',  
 'code' => 'max:64|min:5|unique:statuses,code',  
 ];  
 }  
 public function messages(): array  
 {  
 return [  
 'name.required' => 'Поле "Название" обязательно для заполнения.',  
 'name.max' => 'Поле "Название" не должно превышать 64 символа.',  
 'name.min' => 'Поле "Название" не должно быть меньше 5 символа.',  
 'name.unique' => 'Такое "Название" уже существует.',  
  
 'code.required' => 'Поле "Код" обязательно для заполнения.',  
 'code.max' => 'Поле "Код" не должно превышать 64 символа.',  
 'code.min' => 'Поле "Код" не должно быть меньше 5 символа.',  
 'code.unique' => 'Такой "Код" уже существует.',  
 ];  
 }  
}

## Маршрутизация

Маршрутизация в Laravel - это процесс определения того, как приложение должно отвечать на различные HTTP-запросы. В фреймворке Laravel маршруты определяются в файле routes/web.php для веб-маршрутов и routes/api.php для API-маршрутов.

Маршруты, определенные в файле routes/api.php, вложены в группу маршрутов в RouteServiceProvider. Внутри этой группы автоматически применяется префикс URI/api, поэтому не нужно вручную добавлять его к каждому маршруту в файле маршрутов.

Исходя из технического задания были разработаны маршруты, которые записаны в файле routes\api.php и можно увидеть на листинге 23:

***Листинг 23. Маршрутизация***

// Регистрация пользователя  
Route::post('/register', [AuthController::class, 'register'])->middleware('auth:api');  
// Авторизация  
Route::post('/login', [AuthController::class, 'login']);  
// Выход  
Route::post('/logout', [AuthController::class, 'logout'])->middleware('auth:api');  
  
// Профиль  
Route::middleware('auth:api')->apiResource('profile',UserController::class);  
// Штрафы  
Route::middleware('auth:api')->apiResource('fine',FineController::class);  
// Статусы  
Route::middleware('auth:api')->apiResource('status',StatusController::class);  
// Бонусы  
Route::middleware('auth:api')->apiResource('bonus', BonusController::class);  
// Доступность  
Route::middleware('auth:api')->get('/accesses/my', [AccesseController::class, 'indexCourier']);  
Route::middleware('auth:api')->apiResource('accesses', AccesseController::class);  
  
// Подтверждение доступности  
Route::middleware('auth:api')->patch('/accesses-confirm/{id}', [ConfirmController::class, 'confirm']);  
// Отмена доступности  
Route::middleware('auth:api')->patch('/accesses-cancel/{id}', [ConfirmController::class, 'cancel']);  
// Частичное подтверждение доступности  
Route::middleware('auth:api')->post('/accesses-partial-confirm/{id}', [ConfirmController::class, 'partialConfirm']);  
// Частичная отмена доступности  
Route::middleware('auth:api')->post('/accesses-partial-cancel/{id}', [ConfirmController::class, 'partialCancel']);

# ГЛАВА 4. АВТОТЕСТИРОВАНИЕ API

В рамках разработки системы было проведено комплексное автоматизированное тестирование API с применением Postman. Были разработаны тестовые сценарии, охватывающие основные эндпоинты и сценарии взаимодействия с сервером.

Код состояния HTTP 401 Unauthorized указывает на то, что запрос к API не может быть выполнен, поскольку отсутствует действующая аутентификация; это означает, что клиент должен предоставить учетные данные для доступа к защищенному ресурсу, например, предоставить верный токен в заголовке Authorization, либо у текущего пользователя недостаточно прав для доступа.

## 401 Unauthorized

pm.test ("Проверка кода ответа", function () {

pm.response.to.have.status(401);

});

pm.test ("Проверка сообщения", function () {

pm.expect(pm.response.json()).to.have.property("message");

});

|  |  |
| --- | --- |
| **URL** | **МЕТОД** |
| http://{path}/api/register | POST |
| http://{path}/api/login | POST |
| http://{path}/api/logout | POST |
| http://{path}/api/profile | GET |
| PATCH |
| DELETE |
| http://{path}/api/profile/{id} | GET |
| http://{path}/api/fine | GET |
| POST |
| PATCH |
| DELETE |
| http://{path}/api/fine/{id} | GET |
| http://{path}/api/status | GET |
| POST |
| PATCH |
| DELETE |
| http://{path}/api/status/{id} | GET |
| http://{path}/api/bonus | GET |
| POST |
| PATCH |
| DELETE |
| http://{path}/api/bonus/{id} | GET |
| http://{path}/api/accesses/my | GET |
| http://{path}/api/accesses | GET |
| http://{path}/api/accesses/{id} | GET |
| http://{path}/api/accesses-confirm/{id} | PATCH |
| http://{path}/api/accesses-cancel/{id} | PATCH |
| http://{path}/api/accesses-partial-confirm/{id} | POST |
| http://{path}/api/accesses-partial-cancel/{id} | POST |

## [403 Forbidden](https://ru.wikipedia.org/wiki/%D0%9E%D1%88%D0%B8%D0%B1%D0%BA%D0%B0_403)

Код состояния HTTP 403 Forbidden указывает на то, что сервер понимает запрос, но отказывается его выполнять, даже если клиент аутентифицирован; это означает, что у пользователя, даже после успешной аутентификации, недостаточно прав доступа к запрошенному ресурсу или эндпоинту, либо доступ к этому ресурсу запрещен по другим причинам, не связанным с аутентификацией.

pm.test ("Проверка кода ответа", function () {

pm.response.to.have.status(403);

});

pm.test ("Проверка сообщения", function () {

pm.expect(pm.response.json()).to.have.property("message");

});

|  |  |
| --- | --- |
| **URL** | **МЕТОД** |
| http://{path}/api/accesses | POST |
| PATCH |
| DELETE |

Код состояния HTTP 404 Not Found указывает на то, что сервер не смог найти ресурс, соответствующий запрошенному URL; это означает, что запрошенный эндпоинт или ресурс не существует на сервере, либо был удален, либо URL был введен с ошибкой, и сервер не может обработать запрос, так как целевой ресурс не идентифицирован.

## 404 Not Found

pm.test ("Проверка кода ответа", function () {

pm.response.to.have.status(404);

});

pm.test ("Проверка сообщения", function () {

pm.expect(pm.response.json()).to.have.property("message");

});

|  |  |
| --- | --- |
| **URL** | **МЕТОД** |
| http://{path}/api/profile | PATCH |
| DELETE |
| http://{path}/api/profile/{id} | GET |
| http://{path}/api/fine | GET |
| PATCH |
| DELETE |
| http://{path}/api/fine/{id} | GET |
| http://{path}/api/status | GET |
| PATCH |
| DELETE |
| http://{path}/api/status/{id} | GET |
| http://{path}/api/bonus | GET |
| PATCH |
| DELETE |
| http://{path}/api/bonus/{id} | GET |
| http://{path}/api/accesses | GET |
| http://{path}/api/accesses/{id} | GET |
| http://{path}/api/accesses-confirm/{id} | PATCH |
| http://{path}/api/accesses-cancel/{id} | PATCH |
| http://{path}/api/accesses-partial-confirm/{id} | DELETE |
| http://{path}/api/accesses-partial-cancel/{id} | DELETE |

## 200 OK

Код состояния HTTP 200 OK указывает на то, что запрос был успешно обработан сервером; это означает, что запрошенное действие было выполнено, и сервер возвращает запрошенные данные или подтверждение успешного выполнения операции в теле ответа.

pm.test ("Проверка кода ответа", function () {

pm.response.to.have.status(200);

});

pm.test ("Проверка сообщения", function () {

pm.expect(pm.response.json()).to.have.property("message");

});

|  |  |
| --- | --- |
| **URL** | **МЕТОД** |
| http://{path}/api/login | POST |
| http://{path}/api/logout | POST |
| http://{path}/api/profile | GET |
| PATCH |
| DELETE |
| http://{path}/api/profile/{id} | GET |
| http://{path}/api/fine | GET |
| PATCH |
| DELETE |
| http://{path}/api/fine/{id} | GET |
| http://{path}/api/status | GET |
| PATCH |
| DELETE |
| http://{path}/api/status/{id} | GET |
| http://{path}/api/bonus | GET |
| PATCH |
| DELETE |
| http://{path}/api/bonus/{id} | GET |
| http://{path}/api/accesses | GET |
| http://{path}/api/accesses/{id} | GET |

## 201 Created

Код состояния HTTP 201 Created указывает на то, что запрос был успешно обработан и в результате на сервере был создан новый ресурс; это означает, что сервер успешно создал новый элемент (например, запись в базе данных) на основе данных, предоставленных в запросе, и возвращает URI нового ресурса в заголовке Location.

pm.test ("Проверка кода ответа", function () {

pm.response.to.have.status(201);

});

pm.test ("Проверка сообщения", function () {

pm.expect(pm.response.json()).to.have.property("message");

});

|  |  |
| --- | --- |
| **URL** | **МЕТОД** |
| http://{path}/api/register | POST |
| http://{path}/api/fine | POST |
| http://{path}/api/status | POST |
| http://{path}/api/bonus | POST |

# ГЛАВА 5. РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ

Для разработки мобильного приложения использовалось приложение Visual Studio.

Visual Studio — это интегрированная среда разработки (IDE) от Microsoft, предназначенная для создания, отладки и развертывания приложений на различных платформах и языках программирования.

Для разработки использовался язык программирование C#.

C# - это объектно-ориентированный язык программирования, разработанный Microsoft. Предназначен для создания широкого спектра приложений, включая веб-приложения, мобильные приложения, десктопные приложения, игры и многое другое.

Для разработки на языке C#. использовался фреймворк .NET MAUI.

.NET Multi-Platform App UI (.NET MAUI) — это кроссплатформенная инфраструктура для создания нативных мобильных и классических приложений на C# и XAML. С помощью .NET MAUI можно разрабатывать приложения, которые могут работать на Android, iOS, macOS и Windows, используя единую общую кодовую базу.

## Введение

В современном динамичном мире доставки, эффективное управление курьерской службой является ключевым фактором успеха. Одним из важнейших аспектов такого управления является контроль смен.

В рамках данного курсового проекта была поставлена задача разработки кроссплатформенного мобильного приложения для автоматизации процесса подтверждения доступностей курьерам. Приложение, разработанное на платформе .NET MAUI, призвано предоставить удобный и интуитивно понятный интерфейс для администраторов, позволяя им эффективно контролировать и отслеживать доступности, а также обмениваться информацией в режиме реального времени.

Безопасное взаимодействие приложения с серверной частью, реализованной на Laravel, обеспечивается через REST API, где каждый запрос проходит строгую аутентификацию и авторизацию посредством токенов.

## Структура проекта

Структура на листинге 24 предоставляет хорошую отправную точку для организации MAUI-проекта. Адаптирующая под свои нужды, чтобы обеспечить понятность и удобство сопровождения кода.

***Листинг 24. Структура***

***AdminSamokat***

***├── Properties // настройки проекта***

***├── Зависимости***

***├── Models // классы, представляющие данные***

***└── Access.cs***

***└── AuthResponse.cs***

***└── Bonus.cs***

***└── Fine.cs***

***└── Role.cs***

***└── Status.cs***

***└── User.cs***

***└── ValidationErrorResponse.cs***

***├── Platforms // код, для каждой целевой платформы***

***├── Resources // общие ресурсы***

***├── Views // определяющие пользовательский интерфейс и логику приложения***

***├── Accesses***

***├── Converters // классы-конвертеры***

***└── ConfirmToTexttConverter.cs***

***└── ConfirmToVisibilityConverter.cs***

***└── AccessPage.xaml***

***└── AccessPage.xaml.cs***

***└── AllAccesses.xaml***

***└── AllAccesses.xaml,cs***

***└── PartialAccess.xaml***

***└── PartialAccess.xaml.cs***

***├── Auth***

***└── Login.xaml***

***└──Login.xaml.cs***

***└──Register.xaml***

***└──Register.xaml.cs***

***├── Bonuses***

***└──AllBonuses.xaml***

***└──AllBonuses.xaml.cs***

***└──BonusPage.xaml***

***└──BonusPage.xaml.cs***

***└──CreateBonusPage.xaml***

***└──CreateBonusPage.xaml.cs***

***└──EditBonus.xaml***

***└──EditBonus.xaml.cs***

***├── Couriers***

***└──AllCouriers.xaml***

***└──AllCouriers.xaml.cs***

***└──Courier.xaml***

***└──Courier.xaml.cs***

***└──EditProfileCourier.xaml***

***└──EditProfileCourier.xaml.cs***

***├── Fines***

***└──AllFines.xaml***

***└──AllFines.xaml.cs***

***└──CreateFinePage.xaml***

***└──CreateFinePage.xaml.cs***

***└──EditFine.xaml***

***└──EditFine.xaml.cs***

***└──FinePage.xaml***

***└──FinePage.xaml.cs***

***├── UserProfile***

***└──EditProfile.xaml***

***└──EditProfile.xaml.cs***

***└──Profile.xaml***

***└──Profile.xaml.cs***

***└──Home.xaml***

***└── Home.xaml.cs***

***├──App.xaml // ресурсы и стили, применяемые ко всему приложению***

***└──App.xaml.cs***

***├──AppShell.xaml // структуру навигации***

***└──AppShell.xaml.cs***

***└──MauiProgram.cs // точка входа в приложение и конфигурацию MAUI***

## Конвертеры

Конвертеры — это мощный инструмент для работы с привязкой данных в XAML-приложениях. Они позволяют гибко преобразовывать данные для отображения в пользовательском интерфейсе, обеспечивая разделение ответственности и переиспользование кода. Использование конвертеров помогает создавать более чистый, модульный и удобный в поддержке код.

На листинге 25 представлен конвертер ConfirmToTextConverter, который преобразует целочисленные значения (где 1 обычно означает “подтверждено”) в текстовые строки, такие как “Подтверждено” или “Не подтверждено”, для отображения в пользовательском интерфейсе. Это обеспечивает удобное представление логических состояний в понятной для пользователя форме, упрощая взаимодействие с данными.

***Листинг 25. Класс-конвертер ConfirmToTextConverter***

public class ConfirmToTextConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

// Проверка на null и приведение значения

if (value is int confirm)

{

return confirm == 1 ? "Подтверждено" : "Не подтверждено";

}

return "Нет";

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

На листинге 26 представлен конвертер ConfirmToVisibilityConverter, который определяет видимость элементов интерфейса на основе переданных данных. Конвертер принимает значение типа Tuple<int, DateTime>, где первый элемент (Item1) представляет статус подтверждения (1 — подтверждено, 0 — не подтверждено), а второй элемент (Item2) — дату.

Конвертер выполняет следующие действия:

1. Для параметра "Cancel" возвращает true, если статус подтверждения равен 1, показывая кнопку отмены.
2. Для параметра "PartialConfirm" возвращает true, если статус подтверждения равен 0, показывая кнопку частичного подтверждения.

* Для параметра "PartialCancel" возвращает true, если статус подтверждения равен 1, показывая кнопку частичной отмены.
* Если параметр отсутствует или не распознан, возвращает true для статуса подтверждения 0, показывая кнопку подтверждения.
* Если входное значение не соответствует типу Tuple<int, DateTime>, конвертер возвращает false, скрывая элемент интерфейса.
* Метод ConvertBack не реализован и выбрасывает исключение NotImplementedException, так как конвертер не поддерживает обратное преобразование.

Этот конвертер удобен для управления видимостью кнопок или других элементов интерфейса в зависимости от статуса подтверждения и актуальности даты.

***Листинг 26. Класс-конвертер ConfirmToVisibilityConverter***

public class ConfirmToVisibilityConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

if (value is Tuple<int, DateTime> confirmAndDate)

{

int confirm = confirmAndDate.Item1;

DateTime date = confirmAndDate.Item2;

// Если дата раньше сегодняшнего дня, ничего не показываем

if (date.Date < DateTime.Now.Date)

{

return false;

}

// Обрабатываем параметры

if (parameter is string action)

{

switch (action)

{

case "Cancel":

// Показываем кнопку отмены, если Confirm == 1

return confirm == 1;

case "PartialConfirm":

// Показываем кнопку частичного подтверждения, если Confirm == 0

return confirm == 0;

case "PartialCancel":

// Показываем кнопку частичной отмены, если Confirm == 1

return confirm == 1;

default:

// Показываем кнопку подтверждения, если Confirm == 0

return confirm == 0;

}

}

else

{

// Если параметр отсутствует, показываем кнопку подтверждения, если Confirm == 0

return confirm == 0;

}

}

return false;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

## Модели

В .NET MAUI модели представляют собой классы, определяющие структуру данных, используемых приложением, и служат основой для отображения и обработки информации в пользовательском интерфейсе. Эти классы, как правило, содержат свойства, соответствующие атрибутам данных, и могут включать в себя валидацию данных, а также логику, связанную с самими данными.

Атрибуты JsonPropertyName указывают, как свойства класса соответствуют полям в JSON, используемом при сериализации и десериализации данных, что важно при взаимодействии с API.

На листинге 27 представлена модель Access, который описывает модель данных для таблицы accesses. Этот класс используется для хранения информации о доступностях и их атрибутах, включая уникальный идентификатор (Id), дату доступности (Date), время начала и окончания (StartChange и EndChange), статус подтверждения (Confirm), идентификатор пользователя (UserId), а также вычисляемое свойство ConfirmAndDate, объединяющее статус подтверждения и дату в один объект Tuple. Дополнительные свойства, такие как UserFullName, StartTime, EndTime и флаг IsVisible, обеспечивают удобство работы с данными, включая отображение в пользовательском интерфейсе и управление видимостью.

***Листинг 27. Модель Access***

public class Access

{

// Класс, описывающий таблицу Accesses

[JsonPropertyName("id")]

public ulong Id { get; set; } // Уникальный идентификатор записи

[JsonPropertyName("date")]

public DateTime Date { get; set; } // Дата доступности

[JsonPropertyName("startChange")]

public TimeSpan StartChange { get; set; }

[JsonPropertyName("endChange")]

public TimeSpan EndChange { get; set; }

[JsonPropertyName("confirm")]

public int Confirm { get; set; } // Подтверждение доступности

[JsonPropertyName("user\_id")]

public ulong UserId { get; set; } // ID пользователя, связанного с доступностью

public Tuple<int, DateTime> ConfirmAndDate => new Tuple<int, DateTime>(Confirm, Date); // Свойство для комбинирования Confirm и Date

public string UserFullName { get; set; }

public string StartTime { get; set; }

public string EndTime { get; set; }

public bool IsVisible { get; set; } = false; // По умолчанию доступности скрыты

}

На листинге 28 представлена модель AuthResponse, который описывает ответ модуля аутентификации. Класс содержит два свойства: User, которое представляет объект пользователя типа User, и Token, хранящий строку с токеном аутентификации. Этот класс используется для передачи данных о пользователе и его токене после успешной аутентификации, обеспечивая удобство работы с API.

***Листинг 28. Модель AuthResponse***

public class AuthResponse

{

// Класс, описывающий ответ модуля Auth

[JsonPropertyName("user")]

public User User { get; set; }

[JsonPropertyName("token")]

public string Token { get; set; }

}

На листинге 29 представлена модель Bonus, который описывает таблицу bonuses. Класс содержит свойства для хранения уникального идентификатора (Id), названия (Title), описания (Description), цены (Price), идентификатора роли (RoleId), даты создания (CreatedAt) и обновления (UpdatedAt), а также связанного объекта роли (Role). Дополнительно реализовано вычисляемое свойство FormattedPrice, которое форматирует цену, разделяя рубли и копейки и добавляя символ валюты (₽). Если копейки равны "00", отображаются только рубли, что делает отображение цены более удобным для пользователя.

***Листинг 29. Модель Bonus***

public class Bonus

{

// Класс, описывающий таблицу bonuses

[JsonPropertyName("id")]

public ulong Id { get; set; }

[JsonPropertyName("title")]

public string Title { get; set; }

[JsonPropertyName("description")]

public string? Description { get; set; }

[JsonPropertyName("price")]

public string Price { get; set; }

[JsonPropertyName("role\_id")]

public ulong RoleId { get; set; }

[JsonPropertyName("created\_at")]

public DateTime CreatedAt { get; set; }

[JsonPropertyName("updated\_at")]

public DateTime UpdatedAt { get; set; }

// Свойство для связанной роли

[JsonPropertyName("role")]

public Role Role { get; set; }

// Свойство для форматированной цены

public string FormattedPrice

{

get

{

if (string.IsNullOrEmpty(Price))

return "0 ₽";

// Разделяем рубли и копейки

var parts = Price.Split('.');

var rubles = parts[0];

var kopecks = parts.Length > 1 ? parts[1] : "00";

// Если копейки "00", отображаем только рубли

if (kopecks == "00")

return $"{rubles} ₽";

// Иначе отображаем рубли и копейки

return $"{rubles}.{kopecks} ₽";

}

}

}

На листинге 30 представлена модель Fine, который описывает таблицу fines. Класс содержит свойства для хранения уникального идентификатора (Id), описания (Description), даты создания (CreatedAt) и обновления (UpdatedAt). Эти свойства позволяют хранить информацию о штрафах, включая их описание и временные метки.

***Листинг 30. Модель Fine***

public class Fine

{

// Класс, описывающий таблицу fines

[JsonPropertyName("id")]

public ulong Id { get; set; }

[JsonPropertyName("description")]

public string Description { get; set; }

[JsonPropertyName("created\_at")]

public DateTime CreatedAt { get; set; }

[JsonPropertyName("updated\_at")]

public DateTime UpdatedAt { get; set; }

}

На листинге 31 представлена модель Role, которая описывает таблицу roles. Класс содержит свойства для хранения уникального идентификатора (Id), названия (Name), кодового обозначения (Code), даты создания (CreatedAt) и обновления (UpdatedAt). Эти свойства позволяют хранить информацию о ролях пользователей, включая их наименование, код и временные метки.

***Листинг 31. Модель Role***

public class Role

{

// Класс, описывающий таблицу roles

[JsonPropertyName("id")]

public ulong Id { get; set; }

[JsonPropertyName("name")]

public string Name { get; set; }

[JsonPropertyName("code")]

public string Code { get; set; }

[JsonPropertyName("created\_at")]

public DateTime CreatedAt { get; set; }

[JsonPropertyName("updated\_at")]

public DateTime UpdatedAt { get; set; }

}

На листинге 32 представлена модель Status, которая описывает таблицу statuses. Класс содержит свойства для хранения уникального идентификатора (Id), названия (Name), кодового обозначения (Code), даты создания (CreatedAt) и обновления (UpdatedAt). Эти свойства позволяют хранить информацию о статусах, включая их наименование, код и временные метки.

***Листинг 32. Модель Status***

public class Status

{

// Класс, описывающий таблицу statuses

[JsonPropertyName("id")]

public ulong Id { get; set; }

[JsonPropertyName("name")]

public string Name { get; set; }

[JsonPropertyName("code")]

public string Code { get; set; }

[JsonPropertyName("created\_at")]

public DateTime CreatedAt { get; set; }

[JsonPropertyName("updated\_at")]

public DateTime UpdatedAt { get; set; }

}

На листинге 33 представлена модель User, которая описывает таблицу users. Класс содержит свойства для хранения уникального идентификатора (Id), фамилии (Surname), имени (Name), отчества (Patronymic), логина (Login), пароля (Password), API-токена (ApiToken), а также внешних ключей для связи с таблицами ролей (RoleId), штрафов (FineId) и статусов (StatusId). Дополнительно в модели предусмотрены временные метки (CreatedAt и UpdatedAt), фиксирующие дату создания и обновления записи. Эти свойства позволяют хранить полные сведения о пользователях, включая их учетные данные и статусные характеристики.

***Листинг 33. Модель User***

public class User

{

// Класс, описывающий таблицу users

[JsonPropertyName("id")]

public ulong Id { get; set; }

[JsonPropertyName("surname")]

public string Surname { get; set; }

[JsonPropertyName("name")]

public string Name { get; set; }

[JsonPropertyName("patronymic")]

public string? Patronymic { get; set; }

[JsonPropertyName("login")]

public string Login { get; set; }

[JsonPropertyName("password")]

public string? Password { get; set; } = null;

[JsonPropertyName("api\_token")]

public string? ApiToken { get; set; }

[JsonPropertyName("role\_id")]

public ulong RoleId { get; set; }

[JsonPropertyName("fine\_id")]

public ulong FineId { get; set; }

[JsonPropertyName("status\_id")]

public ulong StatusId { get; set; }

[JsonPropertyName("created\_at")]

public DateTime CreatedAt { get; set; }

[JsonPropertyName("updated\_at")]

public DateTime UpdatedAt { get; set; }

}

На листинге 34 представлена модель ValidationErrorResponse, которая используется для представления ответа об ошибке валидации. Класс содержит свойства для хранения кода ошибки (Code), общего сообщения об ошибке (Message), а также детализированной информации о полях, в которых произошли ошибки (Errors). Последнее представлено в виде словаря, где ключи — это названия полей, а значения — списки сообщений об ошибках. Эта модель позволяет структурированно передавать информацию о нарушениях валидации данных.

***Листинг 34. Модель ValidationErrorResponse***

public class ValidationErrorResponse

{

public int Code { get; set; } // Код ошибки

public string Message { get; set; } // Общее сообщение об ошибке

public Dictionary<string, List<string>> Errors { get; set; } // Детализированные ошибки

}

## Реализация страниц

**Страница определения общих ресурсов**

На листинге 35 представлено XAML-описание приложения AdminSamokat, основанное на .NET MAUI. В файле App.xaml определены общие ресурсы приложения, включая стили и цветовые схемы, которые подключаются через ResourceDictionary. Главным классом приложения является AdminSamokat.App, который устанавливает AppShell в качестве главной страницы.

***Листинг 35. Страница App.xaml***

<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:local="clr-namespace:AdminSamokat"

x:Class="AdminSamokat.App">

<Application.Resources>

<ResourceDictionary>

<ResourceDictionary.MergedDictionaries>

<ResourceDictionary Source="Resources/Styles/Colors.xaml" />

<ResourceDictionary Source="Resources/Styles/Styles.xaml" />

</ResourceDictionary.MergedDictionaries>

</ResourceDictionary>

</Application.Resources>

</Application>

На листинге 36 представлен код App.xaml.cs, который содержит логику инициализации приложения. В методе OnStart() происходит проверка сохраненных пользовательских данных, таких как токен, идентификатор, логин и ФИО. Если userId успешно преобразуется в ulong, создается объект пользователя и выполняется переход на главную страницу Home. В противном случае приложение направляет пользователя на страницу авторизации Login.

***Листинг 36. C# код страницы App.xaml***

public partial class App : Application

{

public App()

{

InitializeComponent();

MainPage = new AppShell();

}

protected override void OnStart()

{

string userToken = Preferences.Get("UserToken", null);

string userId = Preferences.Get("UserId", null);

string userSurname = Preferences.Get("UserSurname", null);

string userName = Preferences.Get("UserName", null);

string userPatronymic = Preferences.Get("UserPatronymic", null);

string userLogin = Preferences.Get("UserLogin", null);

string userPassword = Preferences.Get("UserPassword", null);

// Проверяем и преобразуем userId в ulong

if (ulong.TryParse(userId, out ulong parsedUserId))

{

// Создаём объект User, если id успешно преобразован

var user = new Models.User

{

Id = parsedUserId, // Преобразованный ID

Surname = userSurname,

Name = userName,

Patronymic = userPatronymic,

Login = userLogin,

Password = userPassword

};

// Переход на главную страницу

MainPage = new NavigationPage(new Home(user, userToken));

}

else

{

// Если userId некорректный, отправляем пользователя на страницу авторизации

MainPage = new NavigationPage(new Views.Auth.Login());

}

}

}

***Страница авторизации***

На листинге 37 представлена разметка XAML для страницы авторизации Login в приложении AdminSamokat. Интерфейс включает логотип, приветственное сообщение, поля ввода логина и пароля, а также кнопку входа. Для удобства пользователей реализован индикатор загрузки, который отображается во время обработки запроса.

***Листинг 37. Страница авторизации Login.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Auth.Login"

Shell.NavBarIsVisible="False"

BackgroundColor="#F9F9F9">

<ScrollView>

<!-- Основной контейнер -->

<Grid>

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center"

VerticalOptions="Center"

Color="Gray" />

<!-- Контейнер для интерфейса авторизации -->

<VerticalStackLayout x:Name="LoginForm" Padding="30" Spacing="20" VerticalOptions="Center">

<!-- Логотип -->

<Image Source="logo.png"

WidthRequest="150"

HeightRequest="150"

HorizontalOptions="Center" />

<!-- Приветствие -->

<Label Text="Добро пожаловать в приложение Самокат Про!"

FontSize="20"

FontAttributes="Bold"

TextColor="#333333"

HorizontalTextAlignment="Center" />

<!-- Подзаголовок -->

<Label Text="Введите свой логин и пароль, чтобы продолжить"

FontSize="16"

TextColor="#666666"

HorizontalTextAlignment="Center" />

<!-- Поля ввода -->

<Frame Padding="10"

CornerRadius="8"

BorderColor="#E0E0E0"

BackgroundColor="White"

HasShadow="False">

<Entry x:Name="LoginEntry"

Placeholder="Введите логин"

FontSize="14"

TextColor="#333333"

PlaceholderColor="#B0B0B0" />

</Frame>

<Frame Padding="10"

CornerRadius="8"

BorderColor="#E0E0E0"

BackgroundColor="White"

HasShadow="False">

<Entry x:Name="PasswordEntry"

Placeholder="Введите пароль"

IsPassword="True"

FontSize="14"

TextColor="#333333"

PlaceholderColor="#B0B0B0" />

</Frame>

<!-- Кнопка -->

<Button Text="Войти"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"

Clicked="OnLoginButtonClicked" />

</VerticalStackLayout>

</Grid>

</ScrollView>

</ContentPage>

На листинге 38 представлен код C# для Login.xaml.cs, содержащий логику аутентификации пользователя. При нажатии кнопки входа вводимые данные проверяются, после чего отправляется HTTP-запрос к API. В случае успешной авторизации данные пользователя сохраняются в Preferences, и осуществляется переход на главную страницу Home. Если авторизация не удалась, отображается соответствующее сообщение об ошибке.

***Листинг 38. C# код страницы Login.xaml***

public partial class Login : ContentPage

{

private readonly HttpClient \_httpClient = new HttpClient();

public Login()

{

InitializeComponent();

}

private async void OnLoginButtonClicked(object sender, EventArgs e)

{

string login = LoginEntry.Text;

string password = PasswordEntry.Text;

if (string.IsNullOrEmpty(login) || string.IsNullOrEmpty(password))

{

await DisplayAlert("Ошибка", "Введите логин и пароль", "ОК");

return;

}

try

{

// Скрываем форму и показываем индикатор загрузки

LoginForm.IsVisible = false;

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

var loginResponse = await AuthenticateUserAsync(login, password);

if (loginResponse != null)

{

if (loginResponse.User.RoleId != 1)

{

await DisplayAlert("Ошибка", "Доступ разрешён только администраторам", "ОК");

return;

}

Preferences.Set("UserToken", loginResponse.Token);

Preferences.Set("UserId", loginResponse.User.Id.ToString());

Preferences.Set("UserSurname", loginResponse.User.Surname);

Preferences.Set("UserName", loginResponse.User.Name);

Preferences.Set("UserPatronymic", loginResponse.User.Patronymic);

Preferences.Set("UserLogin", loginResponse.User.Login);

Preferences.Set("UserPassword", loginResponse.User.Password);

// Переход на главную страницу

await Navigation.PushAsync(new Home(loginResponse.User, loginResponse.Token));

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

finally

{

// Восстанавливаем интерфейс, если авторизация не удалась

LoginForm.IsVisible = true;

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

}

}

private async Task<AuthResponse> AuthenticateUserAsync(string login, string password)

{

var loginData = new { login, password };

var jsonContent = new StringContent(JsonSerializer.Serialize(loginData), Encoding.UTF8, "application/json");

HttpResponseMessage response = await \_httpClient.PostAsync("http://courseproject4/api/login", jsonContent);

if (response.StatusCode == System.Net.HttpStatusCode.OK)

{

var content = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<AuthResponse>(content);

}

else if (response.StatusCode == System.Net.HttpStatusCode.Unauthorized)

{

await DisplayAlert("Ошибка входа", "Неправильный логин или пароль", "ОК");

}

else

{

await DisplayAlert("Ошибка", "Произошла ошибка на сервере", "ОК");

}

return null;

}

}

***Главная страница***

На листинге 39 представлена разметка XAML для страницы Home в приложении AdminSamokat. Интерфейс страницы состоит из нескольких разделов, включая приветствие, информацию о пользователе, доступный функционал для администратора и возможность выхода из системы. Каждый раздел включает карточки с иконками и краткими описаниями, на которые можно кликнуть для перехода к соответствующим действиям. Страница использует ScrollView для прокрутки содержимого и Frame для оформления элементов интерфейса. Для каждого элемента предусмотрены обработчики событий, которые выполняют навигацию к другим страницам приложения.

***Листинг 39. Главная страница Home.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Home"

Shell.NavBarIsVisible="False"

BackgroundColor="#F9F9F9">

<Grid>

<!-- Контейнер для всего содержимого страницы -->

<Grid x:Name="MainContent" RowDefinitions="\*,Auto">

<ScrollView>

<VerticalStackLayout Padding="20" Spacing="20">

<!-- Приветствие -->

<Label x:Name="nameLabel"

FontSize="24"

FontAttributes="Bold"

Text="Добро пожаловать!"

TextColor="#333333"

HorizontalOptions="Center" />

<!-- Раздел: Пользователь -->

<Label Text="ПОЛЬЗОВАТЕЛЬ"

FontSize="14"

FontAttributes="Bold"

TextColor="#666666"

HorizontalOptions="Start" />

<VerticalStackLayout Spacing="10">

<!-- Профиль -->

<Frame CornerRadius="10" Padding="10" BackgroundColor="#FFFFFF" HasShadow="False">

<HorizontalStackLayout>

<Image Source="profile.png" WidthRequest="24" HeightRequest="24" Margin="0,0,10,0" />

<VerticalStackLayout>

<Label Text="Профиль"

FontSize="16"

VerticalOptions="Center"

TextColor="#333333" />

<Label Text="Ваши данные"

FontSize="12"

VerticalOptions="Center"

TextColor="#666666"

FontAttributes="Bold"/>

</VerticalStackLayout>

</HorizontalStackLayout>

<Frame.GestureRecognizers>

<TapGestureRecognizer Tapped="OnProfileButtonClicked" />

</Frame.GestureRecognizers>

</Frame>

</VerticalStackLayout>

<!-- Раздел: Функционал администратора -->

<Label Text="ФУНКЦИОНАЛ АДМИНИСТРАТОРА"

FontSize="14"

FontAttributes="Bold"

TextColor="#666666"

HorizontalOptions="Start" />

<VerticalStackLayout Spacing="10">

<!-- Доступности -->

<Frame CornerRadius="10" Padding="10" BackgroundColor="#FFFFFF" HasShadow="False">

<HorizontalStackLayout>

<Image Source="time.png" WidthRequest="24" HeightRequest="24" Margin="0,0,10,0" />

<VerticalStackLayout>

<Label Text="Доступности"

FontSize="16"

VerticalOptions="Center"

TextColor="#333333" />

<Label Text="Подтверждение доступностей"

FontSize="12"

VerticalOptions="Center"

TextColor="#666666"

FontAttributes="Bold"/>

</VerticalStackLayout>

</HorizontalStackLayout>

<Frame.GestureRecognizers>

<TapGestureRecognizer Tapped="OnAccessesClicked" />

</Frame.GestureRecognizers>

</Frame>

<!-- Курьеры -->

<Frame CornerRadius="10" Padding="10" BackgroundColor="#FFFFFF" HasShadow="False">

<HorizontalStackLayout>

<Image Source="couriers.png" WidthRequest="24" HeightRequest="24" Margin="0,0,10,0" />

<VerticalStackLayout>

<Label Text="Курьеры"

FontSize="16"

VerticalOptions="Center"

TextColor="#333333" />

<Label Text="Изменить данные"

FontSize="12"

VerticalOptions="Center"

TextColor="#666666"

FontAttributes="Bold"/>

</VerticalStackLayout>

</HorizontalStackLayout>

<Frame.GestureRecognizers>

<TapGestureRecognizer Tapped="OnCouriersClicked" />

</Frame.GestureRecognizers>

</Frame>

<!-- Зарегистрировать -->

<Frame CornerRadius="10" Padding="10" BackgroundColor="#FFFFFF" HasShadow="False">

<HorizontalStackLayout>

<Image Source="register.png" WidthRequest="24" HeightRequest="24" Margin="0,0,10,0" />

<VerticalStackLayout>

<Label Text="Зарегистрировать"

FontSize="16"

VerticalOptions="Center"

TextColor="#333333" />

<Label Text="Добавить курьера"

FontSize="12"

VerticalOptions="Center"

TextColor="#666666"

FontAttributes="Bold"/>

</VerticalStackLayout>

</HorizontalStackLayout>

<Frame.GestureRecognizers>

<TapGestureRecognizer Tapped="OnRegisterCourierClicked" />

</Frame.GestureRecognizers>

</Frame>

<!-- Бонусы -->

<Frame CornerRadius="10" Padding="10" BackgroundColor="#FFFFFF" HasShadow="False">

<HorizontalStackLayout>

<Image Source="bonuses.png" WidthRequest="24" HeightRequest="24" Margin="0,0,10,0" />

<Label Text="Бонусы"

FontSize="16"

VerticalOptions="Center"

TextColor="#333333" />

</HorizontalStackLayout>

<Frame.GestureRecognizers>

<TapGestureRecognizer Tapped="OnBonusesClicked" />

</Frame.GestureRecognizers>

</Frame>

<!-- Штрафы -->

<Frame CornerRadius="10" Padding="10" BackgroundColor="#FFFFFF" HasShadow="False">

<HorizontalStackLayout>

<Image Source="penalties.png" WidthRequest="24" HeightRequest="24" Margin="0,0,10,0" />

<VerticalStackLayout>

<Label Text="Штрафы"

FontSize="16"

VerticalOptions="Center"

TextColor="#333333" />

<Label Text="Назначить"

FontSize="12"

VerticalOptions="Center"

TextColor="#666666"

FontAttributes="Bold"/>

</VerticalStackLayout>

</HorizontalStackLayout>

<Frame.GestureRecognizers>

<TapGestureRecognizer Tapped="OnPenaltiesClicked" />

</Frame.GestureRecognizers>

</Frame>

</VerticalStackLayout>

<!-- Раздел: Выход из системы -->

<Label Text="ВЫХОД ИЗ СИСТЕМЫ"

FontSize="14"

FontAttributes="Bold"

TextColor="#666666"

HorizontalOptions="Start" />

<!-- Раздел: Выход -->

<Frame CornerRadius="10" Padding="10" BackgroundColor="#FFFFFF" HasShadow="False">

<HorizontalStackLayout>

<Image Source="logout.png" WidthRequest="24" HeightRequest="24" Margin="0,0,10,0" />

<Label Text="Выйти"

FontSize="16"

VerticalOptions="Center"

TextColor="#FF0000" />

</HorizontalStackLayout>

<Frame.GestureRecognizers>

<TapGestureRecognizer Tapped="OnLogoutButtonClicked" />

</Frame.GestureRecognizers>

</Frame>

</VerticalStackLayout>

</ScrollView>

</Grid>

<!-- Слой с индикатором загрузки -->

<Grid x:Name="Overlay" BackgroundColor="#FFFFFF" Opacity="0.8" IsVisible="False">

<ActivityIndicator x:Name="LoadingIndicator"

IsRunning="True"

Color="#333333"

VerticalOptions="Center"

HorizontalOptions="Center"

WidthRequest="50"

HeightRequest="50" />

</Grid>

</Grid>

</ContentPage>

На листинге 40 представлен код C# для страницы Home.xaml.cs, содержащий логику отображения приветственного сообщения и функционала страницы. В зависимости от времени суток приветственное сообщение меняется, отображая утреннее, дневное, вечернее или ночное приветствие. Код также реализует логику выхода из системы с подтверждением действия, а также отображение индикатора загрузки в процессе выполнения HTTP-запроса на сервер. В случае успешного выхода из системы пользователь перенаправляется на страницу авторизации, а токен удаляется из настроек. Кроме того, страницы функционала администратора, такие как доступности, курьеры, бонусы и штрафы, открываются при нажатии соответствующих элементов интерфейса.

***Листинг 40. C# код страницы Home.xaml***

public partial class Home : ContentPage

{

// Инициализация HTTP клиента

private readonly HttpClient \_httpClient = new HttpClient();

private User \_user;

private string \_token;

public Home(User user, string token)

{

InitializeComponent();

\_user = user;

\_token = token;

int hour = DateTime.Now.Hour;

if (hour >= 5 && hour < 12)

{

nameLabel.Text = "Доброе утро, " + user.Name + "!";

}

else if (hour >= 12 && hour < 18)

{

nameLabel.Text = "Добрый день, " + user.Name + "!";

}

else if (hour >= 18 && hour < 22)

{

nameLabel.Text = "Добрый вечер, " + user.Name + "!";

}

else

{

nameLabel.Text = "Доброй ночи, " + user.Name + "! Что вы здесь забыли в это время, ночью зрение портится, ай-ай-ай";

}

}

private async void OnLogoutButtonClicked(object sender, EventArgs e)

{

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите выйти из аккаунта?",

"Да",

"Нет"

);

if (!isConfirmed)

{

return;

}

// Показываем индикатор загрузки

Overlay.IsVisible = true;

string userToken = Preferences.Get("UserToken", string.Empty);

if (!string.IsNullOrEmpty(userToken))

{

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", userToken);

try

{

HttpResponseMessage response = await \_httpClient.PostAsync("http://courseproject4/api/logout", null);

// Скрываем индикатор загрузки ПЕРЕД уведомлением

Overlay.IsVisible = false;

if (response.IsSuccessStatusCode)

{

await DisplayAlert("Успех", "Вы успешно вышли из системы.", "ОК");

}

else

{

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Не удалось завершить сессию. Ответ сервера: {errorContent}", "ОК");

}

}

catch (Exception ex)

{

// Скрываем индикатор загрузки при исключении

Overlay.IsVisible = false;

await DisplayAlert("Ошибка", $"Не удалось выполнить запрос: {ex.Message}", "ОК");

}

}

else

{

// Скрываем индикатор загрузки, если токен отсутствует

Overlay.IsVisible = false;

}

Preferences.Clear();

// Возвращаемся на страницу входа

await Navigation.PushAsync(new Views.Auth.Login());

Navigation.RemovePage(this);

}

protected override void OnAppearing()

{

base.OnAppearing();

// Скрыть системную кнопку "Назад"

NavigationPage.SetHasBackButton(this, false);

}

private async void OnAccessesClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new Accesses.AllAccesses(\_user, \_token));

}

private async void OnCouriersClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new Couries.AllCouriers(\_user, \_token));

}

private async void OnRegisterCourierClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new Auth.Register(\_user, \_token));

}

private async void OnBonusesClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new Bonuses.AllBonuses(\_user, \_token));

}

private async void OnPenaltiesClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new Fines.AllFines(\_user, \_token));

}

private async void OnProfileButtonClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new Profile(\_user, \_token));

}

}

***Страница с деталями доступности***

На листинге 41 представлена разметка XAML для страницы AccessPage в приложении AdminSamokat. Интерфейс состоит из нескольких секций: заголовок страницы, информация о доступности пользователя и кнопки действий. Раздел с информацией включает такие данные, как дата доступности, имя пользователя, время начала и окончания доступности, а также статус подтверждения. Для отображения информации используется привязка данных. Также на странице предусмотрены кнопки для просмотра данных пользователя, подтверждения или отмены доступности, а также частичного подтверждения или отмены. Каждая кнопка имеет соответствующую иконку, текст и индикатор загрузки, который появляется во время обработки запросов.

***Листинг 41. Страница с деталями AccessPage.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:local="clr-namespace:AdminSamokat.Views.Accesses.Converters"

x:Class="AdminSamokat.Views.Accesses.AccessPage"

Padding="20">

<!-- Ресурсы страницы -->

<ContentPage.Resources>

<ResourceDictionary>

<local:ConfirmToVisibilityConverter x:Key="ConfirmToVisibilityConverter" />

</ResourceDictionary>

</ContentPage.Resources>

<ScrollView>

<VerticalStackLayout Spacing="15">

<!-- Заголовок страницы -->

<Label Text="Детали доступности"

FontSize="24"

FontAttributes="Bold"

HorizontalOptions="Center"

VerticalOptions="Start" />

<!-- Информация о пользователе -->

<Frame CornerRadius="10" Padding="15" BorderColor="LightGray">

<VerticalStackLayout Spacing="5">

<!-- Добавляем Label для отображения даты -->

<Label x:Name="DateLabel"

Text="Доступность на: "

FontSize="20"

FontAttributes="Bold"

TextColor="Black" />

<StackLayout HorizontalOptions="FillAndExpand" Orientation="Horizontal" Spacing="10">

<Image Source="user\_icon.png"

HeightRequest="22"

WidthRequest="22"

VerticalOptions="Center" />

<Label x:Name="UserFullNameLabel"

Text="Полное имя пользователя"

FontSize="18"

FontAttributes="Bold"

TextColor="Black" />

</StackLayout>

<Label x:Name="StartTimeLabel"

Text="Начало: "

FontSize="16"

TextColor="Gray" />

<Label x:Name="EndTimeLabel"

Text="Конец: "

FontSize="16"

TextColor="Gray" />

<Label x:Name="ConfirmStatusLabel"

Text="Статус: "

FontSize="16"

FontAttributes="Italic"

TextColor="Gray" />

</VerticalStackLayout>

</Frame>

<!-- Кнопки действий -->

<VerticalStackLayout Spacing="10">

<!-- Кнопка просмотра пользователя -->

<Button x:Name="ViewUserButton"

Text="Просмотреть курьера"

Clicked="OnViewUserButtonClicked"

TextColor="White"

CornerRadius="20"

BackgroundColor="#007BFF"

HorizontalOptions="Fill"

HeightRequest="50" />

<!-- Фиксированная высота кнопки -->

<!-- Кнопка подтверждения доступности -->

<Frame x:Name="ConfirmButtonFrame"

CornerRadius="20"

Padding="0"

HasShadow="False"

HorizontalOptions="Fill"

VerticalOptions="Center"

BackgroundColor="#4CAF50"

IsVisible="{Binding ConfirmAndDate, Converter={StaticResource ConfirmToVisibilityConverter}}"

HeightRequest="50">

<Grid HorizontalOptions="Center" VerticalOptions="Center" Padding="10">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Label x:Name="ConfirmButtonText"

Text="Подтвердить"

TextColor="White"

VerticalOptions="Center"

HorizontalOptions="Center"

Grid.Row="0"

Grid.Column="0"

VerticalTextAlignment="Center"

Padding="0,0,5,0"/>

<Image x:Name="ConfirmButtonIcon"

Source="confirm\_icon.png"

HeightRequest="24"

VerticalOptions="Center"

HorizontalOptions="Center"

Grid.Row="0"

Grid.Column="1"/>

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingConfirmIndicator"

IsRunning="False"

IsVisible="False"

Color="White"

VerticalOptions="Center"

HorizontalOptions="Center"

Grid.RowSpan="1"

Grid.ColumnSpan="2"/>

</Grid>

<Frame.GestureRecognizers>

<TapGestureRecognizer Tapped="OnConfirmButtonClicked" />

</Frame.GestureRecognizers>

</Frame>

<!-- Кнопка отмены подтверждения -->

<Frame x:Name="CancelButtonFrame"

CornerRadius="20"

Padding="0"

HasShadow="False"

HorizontalOptions="Fill"

VerticalOptions="Center"

BackgroundColor="#F44336"

IsVisible="{Binding ConfirmAndDate, Converter={StaticResource ConfirmToVisibilityConverter}, ConverterParameter=Cancel}"

HeightRequest="50">

<Grid HorizontalOptions="Center" VerticalOptions="Center" Padding="10">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<!-- Текст кнопки -->

<Label x:Name="CancelButtonText"

Text="Отменить"

TextColor="White"

VerticalOptions="Center"

HorizontalOptions="Center"

Grid.Row="0"

Grid.Column="0"

VerticalTextAlignment="Center"

Padding="0,0,5,0"/>

<!-- Иконка отмены -->

<Image x:Name="CancelButtonIcon"

Source="cancel\_icon.png"

HeightRequest="24"

VerticalOptions="Center"

HorizontalOptions="Center"

Grid.Row="0"

Grid.Column="1"/>

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingCancelIndicator"

IsRunning="False"

IsVisible="False"

Color="White"

VerticalOptions="Center"

HorizontalOptions="Center"

Grid.RowSpan="1"

Grid.ColumnSpan="2"/>

</Grid>

<Frame.GestureRecognizers>

<TapGestureRecognizer Tapped="OnCancelButtonClicked" />

</Frame.GestureRecognizers>

</Frame>

<Frame x:Name="PartialConfirmButtonFrame"

CornerRadius="20"

Padding="0"

HasShadow="False"

HorizontalOptions="Fill"

VerticalOptions="Center"

BackgroundColor="#4CAF50"

IsVisible="{Binding ConfirmAndDate, Converter={StaticResource ConfirmToVisibilityConverter}, ConverterParameter=PartialConfirm}"

HeightRequest="50">

<Grid HorizontalOptions="Center" VerticalOptions="Center" Padding="10">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Label x:Name="PartialConfirmButtonText"

Text="Подтвердить часть"

TextColor="White"

VerticalOptions="Center"

HorizontalOptions="Center"

Grid.Row="0"

Grid.Column="0"

VerticalTextAlignment="Center"

Padding="0,0,5,0"/>

<Image x:Name="PartialConfirmButtonIcon"

Source="partial\_icon.png"

HeightRequest="24"

VerticalOptions="Center"

HorizontalOptions="Center"

Grid.Row="0"

Grid.Column="1"/>

</Grid>

<Frame.GestureRecognizers>

<TapGestureRecognizer Tapped="OnPartialConfirmButtonClicked" />

</Frame.GestureRecognizers>

</Frame>

<Frame x:Name="PartialCancelButtonFrame"

CornerRadius="20"

Padding="0"

HasShadow="False"

HorizontalOptions="Fill"

VerticalOptions="Center"

BackgroundColor="#F44336"

IsVisible="{Binding ConfirmAndDate, Converter={StaticResource ConfirmToVisibilityConverter}, ConverterParameter=PartialCancel}"

HeightRequest="50">

<Grid HorizontalOptions="Center" VerticalOptions="Center" Padding="10">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Label x:Name="PartialCancelButtonText"

Text="Отменить часть"

TextColor="White"

VerticalOptions="Center"

HorizontalOptions="Center"

Grid.Row="0"

Grid.Column="0"

VerticalTextAlignment="Center"

Padding="0,0,5,0"/>

<Image x:Name="PartialCancelButtonIcon"

Source="partial\_icon.png"

HeightRequest="24"

VerticalOptions="Center"

HorizontalOptions="Center"

Grid.Row="0"

Grid.Column="1"/>

</Grid>

<Frame.GestureRecognizers>

<TapGestureRecognizer Tapped="OnPartialCancelButtonClicked" />

</Frame.GestureRecognizers>

</Frame>

</VerticalStackLayout>

</VerticalStackLayout>

</ScrollView>

</ContentPage>

На листинге 42 представлен код C# для **AccessPage.xaml.cs**, реализующий логику страницы. В конструкторе страницы заполняется информация о доступности пользователя, которая отображается на UI. Когда пользователь нажимает кнопку для просмотра профиля, отправляется HTTP-запрос для получения данных пользователя. Если запрос успешен, происходит переход на страницу с детальной информацией о пользователе. Для подтверждения или отмены доступности используется механизм запросов PATCH. В случае успешного выполнения запросов обновляется UI, скрываются или показываются соответствующие кнопки, и отображается статус. При возникновении ошибок или сбоев в процессе отправки запроса отображается сообщение об ошибке.

***Листинг 42. C# код страницы AccessPage.xaml***

public partial class AccessPage : ContentPage

{

private User \_user;

private string \_token;

private Access \_access;

private readonly HttpClient \_httpClient = new HttpClient();

public AccessPage(Access access, User user, string token)

{

InitializeComponent();

\_access = access;

\_user = user;

\_token = token;

// Устанавливаем BindingContext

BindingContext = \_access;

FillAccessDetails();

}

// Метод для заполнения информации о доступности

private void FillAccessDetails()

{

// Отображаем имя пользователя

UserFullNameLabel.Text = \_access.UserFullName;

// Отображаем время начала и конца

StartTimeLabel.Text = $"Начало: {\_access.StartChange.ToString(@"hh\:mm")}";

EndTimeLabel.Text = $"Конец: {\_access.EndChange.ToString(@"hh\:mm")}";

// Отображаем дату доступности

DateLabel.Text = $"Доступность на: {\_access.Date.ToString("dd.MM.yyyy")}";

// Отображаем статус подтверждения

ConfirmStatusLabel.Text = \_access.Confirm == 1 ? "Статус: Подтверждено" : "Статус: Не подтверждено";

}

// Обработчик нажатия на кнопку для просмотра пользователя

private async void OnViewUserButtonClicked(object sender, EventArgs e)

{

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_token);

var userResponse = await \_httpClient.GetAsync($"http://courseproject4/api/profile/{\_access.UserId}");

if (userResponse.IsSuccessStatusCode)

{

var userContent = await userResponse.Content.ReadAsStringAsync();

var user = JsonSerializer.Deserialize<User>(userContent);

// Переход на страницу курьера

await Navigation.PushAsync(new Couries.Courier(user, \_user, \_token));

}

else

{

await DisplayAlert("Ошибка", "Не удалось загрузить информацию о пользователе.", "ОК");

}

}

// Обработчик нажатия на кнопку для подтверждения доступности

private async void OnConfirmButtonClicked(object sender, EventArgs e)

{

// Подтверждение перед подтверждением

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите подтвердить доступность?",

"Да",

"Нет"

);

if (!isConfirmed)

{

// Если пользователь выбрал "Нет", создание отменяется

return;

}

// Показываем индикатор загрузки и скрываем текст и иконку

LoadingConfirmIndicator.IsRunning = true;

LoadingConfirmIndicator.IsVisible = true;

ConfirmButtonText.IsVisible = false;

ConfirmButtonIcon.IsVisible = false;

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_token);

var content = new FormUrlEncodedContent(new[]

{

new KeyValuePair<string, string>("confirm", "1")

});

try

{

var response = await \_httpClient.PatchAsync(

$"http://courseproject4/api/accesses-confirm/{\_access.Id}", content);

if (response.IsSuccessStatusCode)

{

// Останавливаем индикатор загрузки перед вызовом DisplayAlert

LoadingConfirmIndicator.IsRunning = false;

LoadingConfirmIndicator.IsVisible = false;

ConfirmButtonText.IsVisible = true;

ConfirmButtonIcon.IsVisible = true;

// Не показываем кнопку подтверждения

ConfirmButtonFrame.IsVisible = false;

// Показываем кнопку отмены

CancelButtonFrame.IsVisible = true;

PartialConfirmButtonFrame.IsVisible = false;

PartialCancelButtonFrame.IsVisible = true;

ConfirmStatusLabel.Text = "Статус: Подтверждено";

await DisplayAlert("Успех", "Доступность подтверждена.", "ОК");

// Обновляем свойство Confirm

\_access.Confirm = 1;

// Сообщаем об изменении свойств для привязки

OnPropertyChanged(nameof(\_access.Confirm));

await Navigation.PushAsync(new Home(\_user, \_token));

}

else

{

// Останавливаем индикатор загрузки перед вызовом DisplayAlert

LoadingConfirmIndicator.IsRunning = false;

LoadingConfirmIndicator.IsVisible = false;

ConfirmButtonText.IsVisible = true;

ConfirmButtonIcon.IsVisible = true;

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Не удалось подтвердить доступность. Ответ: {errorContent}", "ОК");

}

}

catch (Exception ex)

{

// Останавливаем индикатор загрузки перед вызовом DisplayAlert

LoadingConfirmIndicator.IsRunning = false;

LoadingConfirmIndicator.IsVisible = false;

ConfirmButtonText.IsVisible = true;

ConfirmButtonIcon.IsVisible = true;

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

}

private async void OnCancelButtonClicked(object sender, EventArgs e)

{

// Подтверждение перед отменой

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите отменить доступность?",

"Да",

"Нет"

);

if (!isConfirmed)

{

return;

}

// Показываем индикатор загрузки и скрываем текст и иконку

LoadingCancelIndicator.IsRunning = true;

LoadingCancelIndicator.IsVisible = true;

CancelButtonText.IsVisible = false;

CancelButtonIcon.IsVisible = false;

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_token);

var content = new FormUrlEncodedContent(new[]

{

new KeyValuePair<string, string>("confirm", "0")

});

try

{

var response = await \_httpClient.PatchAsync(

$"http://courseproject4/api/accesses-cancel/{\_access.Id}", content);

if (response.IsSuccessStatusCode)

{

// Останавливаем индикатор загрузки

LoadingCancelIndicator.IsRunning = false;

LoadingCancelIndicator.IsVisible = false;

CancelButtonText.IsVisible = true;

CancelButtonIcon.IsVisible = true;

// Показываем кнопку подтверждения

ConfirmButtonFrame.IsVisible = true;

// Не показываем кнопку отмены

CancelButtonFrame.IsVisible = false;

PartialConfirmButtonFrame.IsVisible = true;

PartialCancelButtonFrame.IsVisible = false;

// Обновляем статус

ConfirmStatusLabel.Text = "Статус: Не подтверждено";

await DisplayAlert("Успех", "Доступность отменена.", "ОК");

// Обновляем свойство Confirm

\_access.Confirm = 0;

// Сообщаем об изменении свойств для привязки

OnPropertyChanged(nameof(\_access.Confirm));

await Navigation.PushAsync(new Home(\_user, \_token));

}

else

{

// Останавливаем индикатор загрузки

LoadingCancelIndicator.IsRunning = false;

LoadingCancelIndicator.IsVisible = false;

CancelButtonText.IsVisible = true;

CancelButtonIcon.IsVisible = true;

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Не удалось отменить доступность. Ответ: {errorContent}", "ОК");

}

}

catch (Exception ex)

{

// Останавливаем индикатор загрузки

LoadingCancelIndicator.IsRunning = false;

LoadingCancelIndicator.IsVisible = false;

CancelButtonText.IsVisible = true;

CancelButtonIcon.IsVisible = true;

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

}

private async void OnPartialConfirmButtonClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new PartialAccessPage(\_access, \_user, \_token));

}

private async void OnPartialCancelButtonClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new PartialAccessPage(\_access, \_user, \_token));

}

}

***Страница доступностей***

На листинге 43 представлена разметка XAML для страницы отображения доступностей AllAccesses в приложении AdminSamokat. Интерфейс состоит из заголовка, индикатора загрузки, сообщения об отсутствии данных, а также коллекции доступностей, сгруппированных по датам. Каждая группа доступностей включает дату и иконку для раскрытия/сокрытия доступностей. Также предусмотрены элементы для отображения информации о доступности, включая имя пользователя, время начала и окончания, а также статус подтверждения доступности, который отображается с помощью динамически меняющегося цвета текста.

***Листинг 43. Страница доступностей AllAccesses.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Accesses.AllAccesses"

xmlns:converters="clr-namespace:AdminSamokat.Views.Accesses.Converters">

<!-- Ресурсы страницы -->

<ContentPage.Resources>

<converters:ConfirmToTextConverter x:Key="ConfirmConverter" />

</ContentPage.Resources>

<Grid BackgroundColor="#FFFFFF"

RowDefinitions="Auto, \*">

<Label Text="Список доступностей:"

FontSize="20"

FontAttributes="Bold"

TextColor="#333333"

Margin="2"

HorizontalOptions="Center"

VerticalOptions="Center"

Grid.Row="0" />

<Grid Grid.Row="1">

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center"

VerticalOptions="Center"

Color="Gray" />

<!-- Сообщение об отсутствии данных -->

<Label x:Name="EmptyMessageLabel"

Text="Доступности отсутствуют на ближайшие две недели"

FontSize="18"

TextColor="Gray"

HorizontalOptions="Center"

VerticalOptions="Center"

IsVisible="False" />

<!-- CollectionView для отображения доступностей -->

<CollectionView x:Name="AccessesCollectionView"

IsGrouped="True"

SelectionMode="None"

Margin="10"

ItemsLayout="VerticalList">

<!-- Шаблон группы (даты) -->

<CollectionView.GroupHeaderTemplate>

<DataTemplate>

<Frame BackgroundColor="#F5F5F5" Padding="12" CornerRadius="5" Margin="0,5,0,0">

<Grid RowDefinitions="Auto" ColumnDefinitions="\*, Auto">

<!-- Дата -->

<Label Text="{Binding Key}"

FontAttributes="Bold"

FontSize="16"

TextColor="#333333"

VerticalOptions="Center"

Grid.Column="0" />

<!-- Иконка для раскрытия/скрытия доступностей -->

<Image x:Name="ToggleIcon"

Source="{Binding IconSource}"

HeightRequest="24"

WidthRequest="24"

VerticalOptions="Center"

HorizontalOptions="End"

Grid.Column="1" />

</Grid>

<!-- Обработчик нажатия на Frame -->

<Frame.GestureRecognizers>

<TapGestureRecognizer Tapped="OnDateFrameTapped" CommandParameter="{Binding Key}" />

</Frame.GestureRecognizers>

</Frame>

</DataTemplate>

</CollectionView.GroupHeaderTemplate>

<!-- Шаблон элемента доступности -->

<CollectionView.ItemTemplate>

<DataTemplate>

<VerticalStackLayout Spacing="15">

<!-- Обернули Frame для скругленных углов и тени -->

<Frame Padding="20"

Margin="0,2"

CornerRadius="15"

HasShadow="True"

BackgroundColor="#FFFFFF"

IsVisible="{Binding IsVisible}">

<Frame.GestureRecognizers>

<TapGestureRecognizer Tapped="OnAccessTapped" CommandParameter="{Binding .}" />

</Frame.GestureRecognizers>

<!-- Используем IsVisible -->

<Grid RowDefinitions="Auto, Auto, Auto, Auto"

ColumnDefinitions="Auto, \*"

ColumnSpacing="15">

<!-- Иконка слева -->

<Image Source="access\_icon.png"

HeightRequest="40"

WidthRequest="40"

Grid.RowSpan="4"

VerticalOptions="Center" />

<!-- Данные о доступности справа -->

<StackLayout Grid.Column="1" Spacing="5" VerticalOptions="Center">

<StackLayout HorizontalOptions="FillAndExpand" Orientation="Horizontal" Spacing="10">

<Image Source="user\_icon.png"

HeightRequest="22"

WidthRequest="22"

VerticalOptions="Center" />

<Label Text="{Binding UserFullName}"

FontSize="14"

VerticalTextAlignment="Center"

TextColor="#333333" />

</StackLayout>

<Label Text="{Binding StartTime}"

FontSize="14"

VerticalTextAlignment="Center"

TextColor="#333333" />

<Label Text="{Binding EndTime}"

FontSize="14"

VerticalTextAlignment="Center"

TextColor="#333333" />

<!-- Подтверждено -->

<Label Text="{Binding Confirm, Converter={StaticResource ConfirmConverter}}"

FontSize="14"

VerticalTextAlignment="Center">

<Label.Triggers>

<!-- Если Confirm равен true (или вашему значению), цвет становится зелёным -->

<DataTrigger TargetType="Label" Binding="{Binding Confirm}" Value="1">

<Setter Property="TextColor" Value="Green" />

</DataTrigger>

<!-- Если Confirm равен false, цвет становится красным -->

<DataTrigger TargetType="Label" Binding="{Binding Confirm}" Value="0">

<Setter Property="TextColor" Value="Red" />

</DataTrigger>

</Label.Triggers>

</Label>

</StackLayout>

</Grid>

</Frame>

</VerticalStackLayout>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

</Grid>

</Grid>

</ContentPage>

На листинге 44 представлен код C# для AllAccesses.xaml.cs, который включает логику загрузки доступностей. При инициализации страницы выполняется запрос к API для получения списка доступностей. Если запрос успешен, данные обрабатываются и группируются по датам, после чего они передаются в CollectionView для отображения. В случае ошибки авторизации пользователю показывается сообщение, и его перенаправляют на страницу авторизации. Также предусмотрено скрытие и отображение доступностей при нажатии на заголовки групп, а также обработка нажатий на конкретные доступности, что приводит к переходу на страницу с подробной информацией о доступности.

***Листинг 44. C# код страницы AllAccesses.xaml***

public partial class AllAccesses : ContentPage

{

private User \_user;

private string \_token;

private readonly HttpClient \_httpClient = new HttpClient();

private List<Grouping<string, Access>> \_groupedAccesses;

public AllAccesses(User user, string token)

{

InitializeComponent();

\_user = user;

\_token = token;

LoadAccesses();

}

private async void LoadAccesses()

{

try

{

LoadingIndicator.IsVisible = true;

LoadingIndicator.IsRunning = true;

var token = Preferences.Get("UserToken", string.Empty);

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", token);

var response = await \_httpClient.GetAsync("http://courseproject4/api/accesses");

if (response.IsSuccessStatusCode)

{

var content = await response.Content.ReadAsStringAsync();

var accesses = JsonSerializer.Deserialize<List<Access>>(content);

// Загружаем пользователей для каждой доступности

foreach (var access in accesses)

{

var userResponse = await \_httpClient.GetAsync($"http://courseproject4/api/profile/{access.UserId}");

if (userResponse.IsSuccessStatusCode)

{

var userContent = await userResponse.Content.ReadAsStringAsync();

var user = JsonSerializer.Deserialize<User>(userContent);

access.UserFullName = $"{user.Surname} {user.Name[0]}.";

access.StartTime = $"Начало: {access.StartChange.ToString(@"hh\:mm")}";

access.EndTime = $"Конец: {access.EndChange.ToString(@"hh\:mm")}";

}

}

\_groupedAccesses = accesses

.GroupBy(a => a.Date.ToShortDateString())

.OrderBy(group => DateTime.Parse(group.Key)) // Сортируем группы по дате

.Select(group => new Grouping<string, Access>(group.Key, group.OrderBy(a => a.StartChange))) // Сортируем элементы внутри группы

.ToList();

// Устанавливаем начальные значения для иконок

foreach (var group in \_groupedAccesses)

{

group.IsExpanded = false; // По умолчанию группы скрыты

foreach (var access in group)

{

access.IsVisible = false; // Скрываем доступности по умолчанию

}

}

AccessesCollectionView.ItemsSource = \_groupedAccesses;

EmptyMessageLabel.IsVisible = !\_groupedAccesses.Any();

}

else if (response.StatusCode == System.Net.HttpStatusCode.Unauthorized)

{

// Ошибка авторизации

await DisplayAlert("Ошибка", "Ваш токен не актуален. Авторизуйтесь заново!", "ОК");

await Navigation.PushAsync(new Views.Auth.Login());

}

else if (response.StatusCode == System.Net.HttpStatusCode.NotFound)

{

EmptyMessageLabel.IsVisible = true;

}

else

{

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Код: {response.StatusCode}, Ответ: {errorContent}", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

finally

{

LoadingIndicator.IsVisible = false;

LoadingIndicator.IsRunning = false;

}

}

public class Grouping<K, T> : ObservableCollection<T>

{

public K Key { get; }

public bool IsExpanded { get; set; } // Состояние раскрытия группы

public string IconSource => IsExpanded ? "up\_icon.png" : "down\_icon.png"; // Источник иконки

public Grouping(K key, IEnumerable<T> items) : base(items)

{

Key = key;

}

public void Toggle()

{

IsExpanded = !IsExpanded;

OnPropertyChanged(new System.ComponentModel.PropertyChangedEventArgs(nameof(IconSource))); // Уведомление об изменении

}

}

private void OnDateFrameTapped(object sender, EventArgs e)

{

var frame = ((sender as Element)?.Parent as Frame) ?? (sender as Frame);

if (frame != null)

{

var selectedDate = frame.BindingContext as Grouping<string, Access>;

if (selectedDate != null)

{

// Переключаем состояние группы и уведомляем об изменении

selectedDate.Toggle();

// Переключаем видимость доступностей

foreach (var access in selectedDate)

{

access.IsVisible = selectedDate.IsExpanded;

}

// Обновляем CollectionView

AccessesCollectionView.ItemsSource = null;

AccessesCollectionView.ItemsSource = \_groupedAccesses;

}

}

}

private async void OnAccessTapped(object sender, EventArgs e)

{

if (sender is Element element && element.BindingContext is Access selectedAccess)

{

await Navigation.PushAsync(new Accesses.AccessPage(selectedAccess, \_user, \_token));

}

}

}

***Страница частичного подтверждения/отмены доступности***

На листинге 45 представлена разметка XAML для страницы PartialAccessPage в приложении AdminSamokat. Интерфейс включает информацию о доступности пользователя, такую как дата, имя, время начала и окончания доступности. Страница также содержит два выпадающих списка (Picker) для выбора времени начала и окончания, а также кнопки для подтверждения или отмены выбранного времени. Для удобства отображается индикатор загрузки, который появляется во время обработки запроса.

***Листинг 45. Страница частичного подтверждения/отмены доступности PartialAccessPage.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Accesses.PartialAccessPage"

Padding="20">

<Grid>

<!-- Определяем строки для Grid -->

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

<!-- Контент (информация, выбор времени) -->

<RowDefinition Height="Auto" />

<!-- Кнопки -->

</Grid.RowDefinitions>

<!-- Контент -->

<ScrollView Grid.Row="0">

<VerticalStackLayout Spacing="15">

<!-- Заголовок страницы -->

<Label x:Name="Access"

FontSize="24"

FontAttributes="Bold"

HorizontalOptions="Center"

VerticalOptions="Start" />

<!-- Информация о пользователе и дате -->

<Frame CornerRadius="10" Padding="15" BorderColor="LightGray">

<VerticalStackLayout Spacing="5">

<!-- Добавляем Label для отображения даты -->

<Label x:Name="DateLabel"

Text="Доступность на: "

FontSize="20"

FontAttributes="Bold"

TextColor="Black" />

<!-- Добавляем Label для отображения имени пользователя -->

<StackLayout HorizontalOptions="FillAndExpand" Orientation="Horizontal" Spacing="10">

<Image Source="user\_icon.png"

HeightRequest="22"

WidthRequest="22"

VerticalOptions="Center" />

<Label x:Name="UserFullNameLabel"

Text="Полное имя пользователя"

FontSize="18"

FontAttributes="Bold"

TextColor="Black" />

</StackLayout>

<!-- Информация о доступности -->

<Label x:Name="StartTimeLabel" Text="Начало: " FontSize="16" TextColor="Gray" />

<Label x:Name="EndTimeLabel" Text="Конец: " FontSize="16" TextColor="Gray" />

</VerticalStackLayout>

</Frame>

<!-- Выбор времени начала и конца по горизонтали -->

<HorizontalStackLayout Spacing="20" HorizontalOptions="Center">

<!-- Выбор времени начала -->

<HorizontalStackLayout Spacing="5">

<Label Text="С" FontSize="18" HorizontalOptions="Center" VerticalOptions="Center"/>

<Picker x:Name="StartTimePicker"

HorizontalOptions="Center"

WidthRequest="100" />

</HorizontalStackLayout>

<!-- Выбор времени окончания -->

<HorizontalStackLayout Spacing="5">

<Label Text="до" FontSize="18" HorizontalOptions="Center" VerticalOptions="Center"/>

<Picker x:Name="EndTimePicker"

HorizontalOptions="Center"

WidthRequest="100" />

</HorizontalStackLayout>

</HorizontalStackLayout>

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center"

VerticalOptions="Center"

Color="Gray" />

</VerticalStackLayout>

</ScrollView>

<!-- Кнопки внизу экрана -->

<StackLayout Grid.Row="1" Spacing="10" HorizontalOptions="FillAndExpand" Padding="10">

<Button x:Name="PartialConfirmButton"

Text="Подтвердить выделенные часы"

Clicked="OnPartialConfirmButtonClicked"

BackgroundColor="#4CAF50"

TextColor="White"

CornerRadius="20"

HeightRequest="50"

HorizontalOptions="FillAndExpand"

IsVisible="False" />

<Button x:Name="PartialCancelButton"

Text="Отменить выделенные часы"

Clicked="OnPartialCancelButtonClicked"

BackgroundColor="#F44336"

TextColor="White"

CornerRadius="20"

HeightRequest="50"

HorizontalOptions="FillAndExpand"

IsVisible="False" />

</StackLayout>

</Grid>

</ContentPage>

На листинге 46 представлен код C# для PartialAccessPage.xaml.cs, содержащий логику работы с доступностью пользователя. В конструкторе страницы инициализируются значения времени начала и окончания на основе данных, полученных из модели \_access. В методах OnPartialConfirmButtonClicked и OnPartialCancelButtonClicked реализованы обработчики для кнопок подтверждения и отмены доступности. При нажатии на кнопку, выбираются значения времени, проверяется их корректность, после чего отправляется HTTP-запрос к API для подтверждения или отмены доступности. Если запрос успешен, отображается сообщение об успехе, и происходит переход на главную страницу. В случае ошибки показывается соответствующее сообщение.

***Листинг 46. C# код страницы PartialAccessPage.xaml***

public partial class PartialAccessPage : ContentPage

{

private Access \_access;

private User \_user;

private string \_token;

private readonly HttpClient \_httpClient = new HttpClient();

public PartialAccessPage(Access access, User user, string token)

{

InitializeComponent();

\_access = access;

\_user = user;

\_token = token;

// Инициализация Picker с 24-часовым форматом и ограничением по доступности

InitializeTimePickers();

// Отображаем информацию о доступности

StartTimeLabel.Text = $"Начало: {\_access.StartChange.ToString(@"hh\:mm")}";

EndTimeLabel.Text = $"Конец: {\_access.EndChange.ToString(@"hh\:mm")}";

// Отображаем дату доступности

DateLabel.Text = $"Доступность на: {\_access.Date.ToString("dd.MM.yyyy")}";

// Отображаем имя пользователя

UserFullNameLabel.Text = \_access.UserFullName;

// Устанавливаем текст для Label "Access" в зависимости от значения \_access.Confirm

if (\_access.Confirm == 0)

{

Access.Text = "Подтверждение доступности";

PartialConfirmButton.IsVisible = true; // Показываем кнопку "Подтвердить выделенное время"

PartialCancelButton.IsVisible = false; // Скрываем кнопку "Отменить выделенное время"

}

else

{

Access.Text = "Отмена доступности";

PartialConfirmButton.IsVisible = false; // Скрываем кнопку "Подтвердить выделенное время"

PartialCancelButton.IsVisible = true; // Показываем кнопку "Отменить выделенное время"

}

}

// Инициализация Picker с 24-часовым форматом и ограничением по доступности

private void InitializeTimePickers()

{

// Очищаем Picker

StartTimePicker.Items.Clear();

EndTimePicker.Items.Clear();

// Заполняем Picker для начала времени (только доступные часы, исключая конец интервала)

for (int hour = \_access.StartChange.Hours; hour < \_access.EndChange.Hours; hour++)

{

StartTimePicker.Items.Add(hour.ToString("00"));

}

// Заполняем Picker для окончания времени (только доступные часы, исключая начало интервала)

for (int hour = \_access.StartChange.Hours + 1; hour <= \_access.EndChange.Hours; hour++)

{

EndTimePicker.Items.Add(hour.ToString("00"));

}

// Устанавливаем начальное значение для Picker

StartTimePicker.SelectedIndex = 0; // Начало интервала

EndTimePicker.SelectedIndex = EndTimePicker.Items.Count - 1; // Конец интервала

}

// Обработчик для кнопки частичного подтверждения

private async void OnPartialConfirmButtonClicked(object sender, EventArgs e)

{

// Получаем выбранные значения времени

var startTime = StartTimePicker.Items[StartTimePicker.SelectedIndex];

var endTime = EndTimePicker.Items[EndTimePicker.SelectedIndex];

// Проверка, что выбраны корректные временные интервалы

if (int.Parse(startTime) >= int.Parse(endTime))

{

await DisplayAlert("Ошибка", "Время начала должно быть меньше времени окончания.", "ОК");

return;

}

// Подтверждение действия

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите подтвердить выбранную доступность?",

"Да",

"Нет"

);

if (!isConfirmed)

{

return;

}

// Показываем индикатор загрузки

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

PartialConfirmButton.IsEnabled = false;

try

{

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_token);

// Отправляем запрос на сервер для частичного подтверждения

var response = await \_httpClient.PostAsync(

$"http://courseproject4/api/accesses-partial-confirm/{\_access.Id}",

new StringContent(JsonSerializer.Serialize(new

{

startChange = startTime, // Отправляем значение

endChange = endTime, // Отправляем значение

Confirm = 1

}), Encoding.UTF8, "application/json")

);

if (response.IsSuccessStatusCode)

{

// Останавливаем индикатор загрузки

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

PartialConfirmButton.IsEnabled = true;

// Обновляем интерфейс

await DisplayAlert("Успех", "Выбранная доступность подтверждена.", "ОК");

// Возвращаемся на главную страницу

await Navigation.PushAsync(new Home(\_user, \_token));

}

else

{

// Останавливаем индикатор загрузки

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

PartialConfirmButton.IsEnabled = true;

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Не удалось подтвердить выбранную доступность. Ответ: {errorContent}", "ОК");

}

}

catch (Exception ex)

{

// Останавливаем индикатор загрузки

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

PartialConfirmButton.IsEnabled = true;

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

}

// Обработчик для кнопки частичной отмены

private async void OnPartialCancelButtonClicked(object sender, EventArgs e)

{

// Получаем выбранные значения времени

var startTime = StartTimePicker.Items[StartTimePicker.SelectedIndex];

var endTime = EndTimePicker.Items[EndTimePicker.SelectedIndex];

// Проверка, что выбраны корректные временные интервалы

if (int.Parse(startTime) >= int.Parse(endTime))

{

await DisplayAlert("Ошибка", "Время начала должно быть меньше времени окончания.", "ОК");

return;

}

// Подтверждение действия

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите отменить выбранную доступность?",

"Да",

"Нет"

);

if (!isConfirmed)

{

return;

}

// Показываем индикатор загрузки

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

PartialCancelButton.IsEnabled = false;

try

{

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_token);

// Отправляем запрос на сервер для частичной отмены

var response = await \_httpClient.PostAsync(

$"http://courseproject4/api/accesses-partial-cancel/{\_access.Id}",

new StringContent(JsonSerializer.Serialize(new

{

startChange = startTime, // Отправляем значение

endChange = endTime, // Отправляем значение

Confirm = 0

}), Encoding.UTF8, "application/json")

);

if (response.IsSuccessStatusCode)

{

// Останавливаем индикатор загрузки

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

PartialCancelButton.IsEnabled = true;

// Обновляем интерфейс

await DisplayAlert("Успех", "Выбранная доступность отменена.", "ОК");

// Возвращаемся на главную страницу

await Navigation.PushAsync(new Home(\_user, \_token));

}

else

{

// Останавливаем индикатор загрузки

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

PartialCancelButton.IsEnabled = true;

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Не удалось отменить выбранную доступность. Ответ: {errorContent}", "ОК");

}

}

catch (Exception ex)

{

// Останавливаем индикатор загрузки

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

PartialCancelButton.IsEnabled = true;

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

}

}

***Страница регистрации пользователя***

На листинге 47 представлена разметка XAML для страницы регистрации Register в приложении AdminSamokat. Интерфейс включает в себя логотип, заголовок, форму регистрации с полями для ввода фамилии, имени, отчества, логина и пароля. Также присутствуют два поля для подтверждения пароля и кнопка для отправки данных на сервер. Для улучшения пользовательского опыта добавлен индикатор загрузки, который отображается во время отправки данных.

***Листинг 47. Страница регистрации пользователя Register.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Auth.Register"

Shell.NavBarIsVisible="False"

BackgroundColor="#F9F9F9">

<!-- Основной контейнер -->

<Grid>

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center"

VerticalOptions="Center"

Color="Gray" />

<!-- Основная форма -->

<ScrollView x:Name="RegistrationForm" IsVisible="True">

<StackLayout Padding="20" Spacing="20">

<!-- Логотип -->

<Image Source="logo.png" HorizontalOptions="Center" WidthRequest="200" HeightRequest="200"/>

<!-- Текстовое описание -->

<Label Text="Регистрация курьера"

FontSize="24"

HorizontalTextAlignment="Center"

VerticalOptions="Center"

Margin="0,20,0,20"/>

<!-- Форма регистрации -->

<Frame Padding="15" CornerRadius="15" BackgroundColor="White" Margin="0,20">

<StackLayout Spacing="15">

<Frame BackgroundColor="#F0F0F0" CornerRadius="10">

<Entry x:Name="SurnameEntry"

Placeholder="Введите фамилию"

Margin="5"/>

</Frame>

<Frame BackgroundColor="#F0F0F0" CornerRadius="10">

<Entry x:Name="NameEntry"

Placeholder="Введите имя"

Margin="5"/>

</Frame>

<Frame BackgroundColor="#F0F0F0" CornerRadius="10">

<Entry x:Name="PatronymicEntry"

Placeholder="Введите отчество"

Margin="5"/>

</Frame>

<Frame BackgroundColor="#F0F0F0" CornerRadius="10">

<Entry x:Name="LoginEntry"

Placeholder="Введите логин"

Margin="5"/>

</Frame>

<Frame BackgroundColor="#F0F0F0" CornerRadius="10">

<Entry x:Name="PasswordEntry"

Placeholder="Введите пароль"

IsPassword="True"

Margin="5"/>

</Frame>

<Frame BackgroundColor="#F0F0F0" CornerRadius="10">

<Entry x:Name="ConfirmPasswordEntry"

Placeholder="Подтвердите пароль"

IsPassword="True"

Margin="5"/>

</Frame>

</StackLayout>

</Frame>

<!-- Кнопка регистрации -->

<Button Text="Зарегистрировать"

Clicked="OnRegisterButtonClicked"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"/>

</StackLayout>

</ScrollView>

</Grid>

</ContentPage>

На листинге 48 представлен код C# для страницы Register.xaml.cs, реализующий логику регистрации нового пользователя. При нажатии на кнопку регистрации проверяется корректность введённых данных (поля не должны быть пустыми, пароли должны совпадать). Затем пользователь получает запрос на подтверждение действия. После подтверждения выполняется сбор данных из формы и отправка их на сервер с помощью HTTP-запроса. В случае успешной регистрации пользователю показывается сообщение об успешном добавлении, иначе — ошибки, если есть проблемы с авторизацией или валидацией. В случае ошибки авторизации приложение перенаправляет пользователя на страницу входа.

***Листинг 48. C# код страницы Register.xaml***

public partial class Register : ContentPage

{

private readonly HttpClient \_httpClient = new HttpClient();

private User \_user;

private string \_token;

public Register(User user, string token)

{

InitializeComponent();

\_user = user;

\_token = token;

}

private async void OnRegisterButtonClicked(object sender, EventArgs e)

{

// Проверка на пустые поля

if (string.IsNullOrWhiteSpace(SurnameEntry.Text) ||

string.IsNullOrWhiteSpace(NameEntry.Text) ||

string.IsNullOrWhiteSpace(LoginEntry.Text) ||

string.IsNullOrWhiteSpace(PasswordEntry.Text))

{

await DisplayAlert("Ошибка", "Все обязательные поля должны быть заполнены", "ОК");

return;

}

if (PasswordEntry.Text != ConfirmPasswordEntry.Text)

{

await DisplayAlert("Ошибка", "Пароли не совпадают", "ОК");

return;

}

// Подтверждение перед регистрацией

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите добавить нового пользователя?",

"Да",

"Нет"

);

if (!isConfirmed)

{

// Если пользователь выбрал "Нет", регистрация отменяется

return;

}

// Сбор данных из формы

string surname = SurnameEntry.Text;

string name = NameEntry.Text;

string? patronymic = PatronymicEntry.Text;

string login = LoginEntry.Text;

string password = PasswordEntry.Text;

var registerData = new MultipartFormDataContent

{

{ new StringContent(surname), "surname" },

{ new StringContent(name), "name" },

{ new StringContent(patronymic ?? string.Empty), "patronymic" },

{ new StringContent(login), "login" },

{ new StringContent(password), "password" },

};

try

{

// Отображаем индикатор загрузки и скрываем форму

RegistrationForm.IsVisible = false;

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

\_httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", \_token);

HttpResponseMessage response = await \_httpClient.PostAsync("http://courseproject4/api/register", registerData);

if (response.IsSuccessStatusCode)

{

await DisplayAlert("Успешная регистрация", "Пользователь успешно добавлен", "ОК");

}

else if (response.StatusCode == System.Net.HttpStatusCode.Unauthorized)

{

// Ошибка авторизации

await DisplayAlert("Ошибка", "Ваш токен не актуален. Авторизуйтесь заново!", "ОК");

await Navigation.PushAsync(new Views.Auth.Login());

}

else if (response.StatusCode == System.Net.HttpStatusCode.UnprocessableEntity)

{

var errorContent = await response.Content.ReadAsStringAsync();

try

{

// Парсим JSON как JsonDocument

using var document = JsonDocument.Parse(errorContent);

var root = document.RootElement;

var message = root.GetProperty("message").GetString();

if (root.TryGetProperty("errors", out var errors))

{

var errorMessages = new List<string>();

// Проходим по всем ошибкам

foreach (var error in errors.EnumerateObject())

{

foreach (var msg in error.Value.EnumerateArray())

{

errorMessages.Add(System.Text.RegularExpressions.Regex.Unescape(msg.GetString()));

}

}

var combinedErrors = string.Join("\n", errorMessages);

await DisplayAlert("Ошибка валидации", combinedErrors, "ОК");

}

else

{

await DisplayAlert("Ошибка", message ?? "Произошла ошибка валидации.", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Не удалось обработать ответ: {ex.Message}\n\nОтвет: {errorContent}", "ОК");

}

}

else

{

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Код ошибки: {(int)response.StatusCode}\n{errorContent}", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка сети", ex.Message, "ОК");

}

finally

{

// Восстанавливаем форму и скрываем индикатор загрузки

RegistrationForm.IsVisible = true;

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

}

}

}

***Страница бонусов***

На листинге 49 представлена разметка XAML для страницы отображения всех бонусов AllBonuses в приложении AdminSamokat. Интерфейс включает индикатор загрузки, который появляется при загрузке данных, и сообщение о том, что бонусы отсутствуют, если данные пусты. В странице также содержится заголовок, список бонусов (с использованием CollectionView), отображающий информацию о каждом бонусе, и кнопка для создания нового бонуса.

***Листинг 49. Страница бонусов AllBonuses.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Bonuses.AllBonuses">

<Grid>

<!-- Определяем строки для Grid -->

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

<!-- Контент (бонусы, индикатор загрузки, сообщение) -->

<RowDefinition Height="Auto" />

<!-- Кнопка -->

</Grid.RowDefinitions>

<!-- Индикатор загрузки по центру экрана -->

<ActivityIndicator x:Name="LoadingIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center"

VerticalOptions="Center"

Color="Gray"

Grid.Row="0" />

<!-- Сообщение об отсутствии бонусов -->

<Label x:Name="EmptyMessageLabel"

Text="Бонусы отсутствуют"

FontSize="18"

TextColor="Gray"

HorizontalOptions="Center"

VerticalOptions="Center"

IsVisible="False"

Grid.Row="0" />

<!-- ScrollView для прокрутки содержимого -->

<ScrollView Grid.Row="0">

<VerticalStackLayout Spacing="10" Padding="10">

<!-- Заголовок страницы -->

<Label Text="Список бонусов:"

FontSize="20"

FontAttributes="Bold"

TextColor="#333333"

HorizontalOptions="Center"

VerticalOptions="Center" />

<!-- CollectionView для отображения бонусов -->

<CollectionView x:Name="BonusesCollectionView"

SelectionMode="Single"

SelectionChanged="OnBonusSelected"

IsVisible="False">

<CollectionView.ItemTemplate>

<DataTemplate>

<!-- Оборачиваем в StackLayout с отступами -->

<StackLayout Padding="10" Spacing="10">

<Frame Padding="15" CornerRadius="10" HasShadow="True" BackgroundColor="White">

<Grid RowDefinitions="Auto,Auto,Auto,Auto" ColumnDefinitions="Auto,\*">

<Image Source="bonus\_icon.png"

HeightRequest="50"

WidthRequest="50"

Aspect="AspectFill"

Grid.RowSpan="4"

Margin="0,0,15,0"/>

<Label Text="{Binding Title}"

FontAttributes="Bold"

FontSize="16"

Grid.Column="1" />

<Label Text="{Binding FormattedPrice}"

FontSize="12"

TextColor="Gray"

Grid.Row="2"

Grid.Column="1" />

<!-- Добавляем Label для отображения роли справа внизу -->

<Label Text="{Binding Role.Name, StringFormat='Для роли: {0}'}"

FontSize="12"

TextColor="Gray"

Grid.Row="3"

Grid.Column="1"

HorizontalOptions="End"

VerticalOptions="End"

Margin="0,0,10,10" />

</Grid>

</Frame>

</StackLayout>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

</VerticalStackLayout>

</ScrollView>

<!-- Кнопка для создания бонуса -->

<Button Grid.Row="1"

Text="Создать бонус"

FontAttributes="Bold"

BackgroundColor="#6200EE"

TextColor="White"

CornerRadius="25"

Padding="15"

HorizontalOptions="FillAndExpand"

VerticalOptions="End"

Margin="10"

Clicked="OnCreateBonusClicked" />

</Grid>

</ContentPage>

На листинге 50 представлен код C# для страницы AllBonuses.xaml.cs, реализующий логику загрузки бонусов с сервера. При инициализации страницы создается HTTP-запрос для получения списка бонусов, который затем отображается в CollectionView. В случае ошибки или отсутствия бонусов показывается соответствующее сообщение. Также реализована обработка событий: при выборе бонуса осуществляется переход на страницу с подробной информацией о бонусе, а при нажатии кнопки "Создать бонус" происходит переход на страницу для создания нового бонуса.

***Листинг 50. C# код страницы AllBonuses.xaml***

public partial class AllBonuses : ContentPage

{

private User \_user;

private string \_token;

private readonly HttpClient \_httpClient = new HttpClient();

public ObservableCollection<Bonus> Bonuses { get; set; } = new ObservableCollection<Bonus>();

public AllBonuses(User user, string token)

{

InitializeComponent();

\_user = user;

\_token = token;

BonusesCollectionView.ItemsSource = Bonuses;

LoadBonuses();

}

private async void LoadBonuses()

{

try

{

// Показываем индикатор загрузки

LoadingIndicator.IsVisible = true;

LoadingIndicator.IsRunning = true;

var token = Preferences.Get("UserToken", string.Empty);

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", token);

var response = await \_httpClient.GetAsync("http://courseproject4/api/bonus");

if (response.IsSuccessStatusCode)

{

var content = await response.Content.ReadAsStringAsync();

var bonuses = JsonSerializer.Deserialize<List<Bonus>>(content);

Bonuses.Clear();

foreach (var bonus in bonuses)

{

Bonuses.Add(bonus);

}

// Управляем видимостью элементов в зависимости от количества бонусов

BonusesCollectionView.IsVisible = Bonuses.Count > 0;

EmptyMessageLabel.IsVisible = Bonuses.Count == 0;

}

else if (response.StatusCode == System.Net.HttpStatusCode.Unauthorized)

{

// Ошибка авторизации

await DisplayAlert("Ошибка", "Ваш токен не актуален. Авторизуйтесь заново!", "ОК");

await Navigation.PushAsync(new Views.Auth.Login());

}

else

{

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Код: {response.StatusCode}, Ответ: {errorContent}", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

finally

{

// Скрываем индикатор загрузки после завершения загрузки данных

LoadingIndicator.IsVisible = false;

LoadingIndicator.IsRunning = false;

}

}

private async void OnBonusSelected(object sender, SelectionChangedEventArgs e)

{

// Получаем выбранного пользователя

if (e.CurrentSelection.FirstOrDefault() is Bonus selectedBonus)

{

// Переход на страницу информации о бонусе

await Navigation.PushAsync(new BonusPage(selectedBonus, \_user, \_token));

}

// Сбрасываем выбор

((CollectionView)sender).SelectedItem = null;

}

private async void OnCreateBonusClicked(object sender, EventArgs e)

{

// Переход на страницу создания бонуса

await Navigation.PushAsync(new CreateBonusPage(\_user, \_token));

}

}

***Страница бонуса***

На листинге 51 представлена разметка XAML для страницы информации о бонусе BonusPage в приложении AdminSamokat. Интерфейс включает данные о бонусе, такие как название, описание, цена и роль, для которой предоставлен бонус. Также присутствуют две кнопки: "Изменить" для перехода на страницу редактирования бонуса и "Удалить" для удаления бонуса. Внизу страницы находится индикатор загрузки, который отображается во время обработки запроса.

***Листинг 51. Страница бонуса BonusPage.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Bonuses.BonusPage">

<ScrollView>

<Grid>

<!-- Основное содержимое страницы -->

<Grid x:Name="MainContent" RowDefinitions="\*,Auto" Padding="20">

<!-- Основная информация о бонусе -->

<VerticalStackLayout Grid.Row="0" Spacing="10">

<Label x:Name="BonusNameLabel"

FontSize="20"

FontAttributes="Bold"

HorizontalOptions="Center" />

<Label x:Name="BonusDescriptionLabel"

FontSize="16"

TextColor="Gray"

HorizontalOptions="Center" />

<Label x:Name="BonusPriceLabel"

FontSize="16"

TextColor="Gray"

HorizontalOptions="Center" />

<!-- Добавляем Label для отображения роли -->

<Label x:Name="BonusRoleLabel"

FontSize="16"

TextColor="Gray"

HorizontalOptions="Center" />

<Button Text="Изменить"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"

Clicked="OnEditButtonClicked" />

<Button Text="Удалить"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"

Clicked="OnDeleteButtonClicked" />

</VerticalStackLayout>

</Grid>

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center"

VerticalOptions="Center" />

</Grid>

</ScrollView>

</ContentPage>

На листинге 52 представлен код C# для страницы BonusPage.xaml.cs, реализующий логику отображения данных о бонусе и обработку событий для кнопок. При инициализации страницы отображаются сведения о выбранном бонусе, включая роль, если она указана. Кнопка "Изменить" вызывает переход на страницу редактирования бонуса, а кнопка "Удалить" инициирует процесс удаления бонуса с подтверждением. При успешном удалении бонуса отображается сообщение об успехе, и осуществляется переход на главную страницу. В случае ошибок показывается сообщение об ошибке.

***Листинг 52. C# код страницы BonusPage.xaml***

public partial class BonusPage : ContentPage

{

private User \_user;

private string \_token;

private Bonus \_bonus;

private readonly HttpClient \_httpClient = new HttpClient();

public BonusPage(Bonus bonus, User user, string token)

{

InitializeComponent();

\_bonus = bonus;

\_user = user;

\_token = token;

// Отобразить данные бонуса

BonusNameLabel.Text = \_bonus.Title;

BonusDescriptionLabel.Text = \_bonus.Description;

BonusPriceLabel.Text = \_bonus.FormattedPrice;

// Отобразить роль

if (\_bonus.Role != null)

{

BonusRoleLabel.Text = $"Для роли: {\_bonus.Role.Name}";

}

else

{

BonusRoleLabel.Text = "Роль не указана";

}

}

private async void OnEditButtonClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new EditBonus(\_bonus, \_user, \_token));

}

private async void OnDeleteButtonClicked(object sender, EventArgs e)

{

// Подтверждение перед удалением

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите удалить этот бонус?",

"Да",

"Нет"

);

if (!isConfirmed)

{

// Если пользователь выбрал "Нет", удаление отменяется

return;

}

try

{

// Скрываем основное содержимое и показываем индикатор

MainContent.IsVisible = false;

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

// Настраиваем заголовки и токен

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_token);

// Отправляем DELETE-запрос на сервер

var response = await \_httpClient.DeleteAsync($"http://courseproject4/api/bonus/{\_bonus.Id}");

if (response.IsSuccessStatusCode)

{

await DisplayAlert("Успех", "Бонус успешно удалён.", "ОК");

await Navigation.PushAsync(new Home(\_user, \_token));

}

else

{

var responseContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Не удалось удалить бонус: {response.StatusCode} - {responseContent}", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

finally

{

// Возвращаем основное содержимое и скрываем индикатор

MainContent.IsVisible = true;

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

}

}

}

***Страница создания бонуса***

На листинге 53 представлена разметка XAML для страницы создания бонуса CreateBonusPage в приложении AdminSamokat. Интерфейс включает форму для ввода данных о бонусе, таких как название, описание, награда и роль. В верхней части страницы отображается логотип и текстовое описание. Также предусмотрена кнопка для создания бонуса. В процессе создания отображается индикатор загрузки, который активируется во время обработки запроса.

***Листинг 53. Страница создания бонуса CreateBonusPage.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Bonuses.CreateBonusPage"

Shell.NavBarIsVisible="False"

BackgroundColor="#F9F9F9">

<!-- Основной контейнер -->

<Grid>

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center"

VerticalOptions="Center"

Color="Gray" />

<!-- Основная форма -->

<ScrollView x:Name="CreateForm" IsVisible="True">

<StackLayout Padding="20" Spacing="20">

<!-- Логотип -->

<Image Source="logo.png" HorizontalOptions="Center" WidthRequest="200" HeightRequest="200"/>

<!-- Текстовое описание -->

<Label Text="На этой странице вы можете создать бонус!"

FontSize="24"

HorizontalTextAlignment="Center"

VerticalOptions="Center"

Margin="0,20,0,20"/>

<!-- Форма создания -->

<Frame Padding="15" CornerRadius="15" BackgroundColor="White" Margin="0,20">

<StackLayout Spacing="15">

<Frame BackgroundColor="#F0F0F0" CornerRadius="10">

<Entry x:Name="titleEntry"

Placeholder="Введите название"

Margin="5"/>

</Frame>

<Frame BackgroundColor="#F0F0F0" CornerRadius="10">

<Entry x:Name="descriptionEntry"

Placeholder="Введите описание"

Margin="5"/>

</Frame>

<Frame BackgroundColor="#F0F0F0" CornerRadius="10">

<Entry x:Name="priceEntry"

Placeholder="Введите награду"

Margin="5"/>

</Frame>

<Frame BackgroundColor="#F0F0F0" CornerRadius="10">

<Picker x:Name="rolePicker"

Title="Выберите роль"

ItemDisplayBinding="{Binding Name}"

BackgroundColor="Transparent"

Margin="5"/>

</Frame>

</StackLayout>

</Frame>

<!-- Кнопка создания -->

<Button Text="Создать"

Clicked="OnCreateButtonClicked"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"/>

</StackLayout>

</ScrollView>

</Grid>

</ContentPage>

На листинге 54 представлен код C# для страницы CreateBonusPage.xaml.cs, реализующий логику создания нового бонуса. При нажатии кнопки создания бонуса проверяется, заполнены ли все обязательные поля, после чего отправляется запрос на сервер. В случае успешного добавления бонуса показывается сообщение об успехе, и пользователь перенаправляется на главную страницу. В случае ошибок, таких как ошибки валидации или сетевые ошибки, отображается соответствующее сообщение.

***Листинг 54. C# код страницы CreateBonusPage.xaml***

public partial class CreateBonusPage : ContentPage

{

private User \_user;

private string \_token;

private readonly HttpClient \_httpClient = new HttpClient();

public CreateBonusPage(User user, string token)

{

InitializeComponent();

\_user = user;

\_token = token;

// Устанавливаем список ролей в Picker

rolePicker.ItemsSource = Roles;

// Устанавливаем выбранную роль по умолчанию

if (Roles.Count > 0)

{

rolePicker.SelectedItem = Roles[0];

}

}

private static readonly List<Role> Roles = new List<Role>

{

new Role { Id = 1, Name = "Администратор", Code = "admin" },

new Role { Id = 2, Name = "Курьер", Code = "courier" }

};

private async void OnCreateButtonClicked(object sender, EventArgs e)

{

// Проверка на пустые поля

if (string.IsNullOrWhiteSpace(titleEntry.Text) ||

string.IsNullOrWhiteSpace(descriptionEntry.Text) ||

string.IsNullOrWhiteSpace(priceEntry.Text) ||

rolePicker.SelectedItem == null)

{

await DisplayAlert("Ошибка", "Все обязательные поля должны быть заполнены", "ОК");

return;

}

// Подтверждение перед созданием

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите добавить новый бонус?",

"Да",

"Нет"

);

if (!isConfirmed)

{

// Если пользователь выбрал "Нет", создание отменяется

return;

}

// Сбор данных из формы

string title = titleEntry.Text;

string description = descriptionEntry.Text;

string price = priceEntry.Text;

var selectedRole = (Role)rolePicker.SelectedItem;

var createData = new MultipartFormDataContent

{

{ new StringContent(title), "title" },

{ new StringContent(description), "description" },

{ new StringContent(price), "price" },

{ new StringContent(selectedRole.Id.ToString()), "role\_id" } // Добавляем role\_id

};

try

{

// Отображаем индикатор загрузки и скрываем форму

CreateForm.IsVisible = false;

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

\_httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", \_token);

HttpResponseMessage response = await \_httpClient.PostAsync("http://courseproject4/api/bonus", createData);

if (response.IsSuccessStatusCode)

{

await DisplayAlert("Успех", "Бонус успешно добавлен", "ОК");

await Navigation.PushAsync(new Views.Home(\_user, \_token));

}

else if (response.StatusCode == System.Net.HttpStatusCode.UnprocessableEntity)

{

var errorContent = await response.Content.ReadAsStringAsync();

try

{

// Парсим JSON как JsonDocument

using var document = JsonDocument.Parse(errorContent);

var root = document.RootElement;

var message = root.GetProperty("message").GetString();

if (root.TryGetProperty("errors", out var errors))

{

var errorMessages = new List<string>();

// Проходим по всем ошибкам

foreach (var error in errors.EnumerateObject())

{

foreach (var msg in error.Value.EnumerateArray())

{

errorMessages.Add(System.Text.RegularExpressions.Regex.Unescape(msg.GetString()));

}

}

var combinedErrors = string.Join("\n", errorMessages);

await DisplayAlert("Ошибка валидации", combinedErrors, "ОК");

}

else

{

await DisplayAlert("Ошибка", message ?? "Произошла ошибка валидации.", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Не удалось обработать ответ: {ex.Message}\n\nОтвет: {errorContent}", "ОК");

}

}

else

{

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Код ошибки: {(int)response.StatusCode}\n{errorContent}", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка сети", ex.Message, "ОК");

}

finally

{

// Восстанавливаем форму и скрываем индикатор загрузки

CreateForm.IsVisible = true;

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

}

}

}

***Страница редактирования бонуса***

На листинге 55 представлена разметка XAML для страницы редактирования бонуса EditBonus в приложении AdminSamokat. Интерфейс включает заголовок, поля ввода названия, описания и суммы награды, а также выпадающий список для выбора роли. Для удобства пользователей реализована кнопка сохранения и индикатор загрузки, который отображается во время обработки запроса.

***Листинг 55. Страница редактирования бонуса EditBonus.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Bonuses.EditBonus"

Shell.NavBarIsVisible="False"

BackgroundColor="#F9F9F9">

<Grid>

<!-- Основной контент с прокруткой -->

<ScrollView x:Name="MainContent" IsVisible="True">

<StackLayout Padding="20" Spacing="20">

<!-- Заголовок -->

<Label Text="Редактировать бонус"

FontSize="28"

FontAttributes="Bold"

HorizontalOptions="Center"

VerticalOptions="Center"

Margin="0,20,0,20"/>

<!-- Поля для ввода -->

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray" Margin="0,5">

<StackLayout>

<Label Text="Название" FontAttributes="Bold" />

<Entry x:Name="titleLabel" Placeholder="Введите название" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray" Margin="0,5">

<StackLayout>

<Label Text="Описание" FontAttributes="Bold" />

<Entry x:Name="descriptionLabel" Placeholder="Введите описание" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray" Margin="0,5">

<StackLayout>

<Label Text="Награда (с копейками через .)" FontAttributes="Bold" />

<Entry x:Name="priceLabel" Keyboard="Numeric" Placeholder="Введите награду" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray" Margin="0,5">

<StackLayout>

<Label Text="Роль" FontAttributes="Bold" />

<Picker x:Name="rolePicker"

Title="Выберите роль"

ItemDisplayBinding="{Binding Name}"

BackgroundColor="#F5F5F5" />

</StackLayout>

</Frame>

<!-- Кнопка сохранения -->

<Button Text="Сохранить"

Clicked="OnSaveButtonClicked"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"

Margin="0,20,0,0"/>

</StackLayout>

</ScrollView>

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingIndicator"

IsVisible="False"

IsRunning="False"

Color="Gray"

HorizontalOptions="Center"

VerticalOptions="Center"/>

</Grid>

</ContentPage>

На листинге 56 представлен код C# для EditBonus.xaml.cs, содержащий логику обновления бонуса. При загрузке страницы поля заполняются текущими значениями бонуса. При нажатии кнопки сохранения выполняется валидация данных, и, если ошибок нет, формируется HTTP-запрос к API для обновления информации. В случае успешного обновления данные обновляются в интерфейсе, и пользователь перенаправляется на главную страницу. Если сервер возвращает ошибки валидации, они отображаются в виде сообщений пользователю.

***Листинг 56. C# код страницы EditBonus.xaml***

public partial class EditBonus : ContentPage

{

// Инициализация HTTP клиента

private readonly HttpClient \_httpClient = new HttpClient();

private Bonus \_bonus;

private User \_user;

private string \_token;

public EditBonus(Bonus bonus, User user, string token)

{

InitializeComponent();

\_user = user;

\_token = token;

\_bonus = bonus;

titleLabel.Text = bonus.Title;

descriptionLabel.Text = bonus.Description;

priceLabel.Text = bonus.Price;

// Устанавливаем список ролей в Picker

rolePicker.ItemsSource = Roles;

// Устанавливаем выбранную роль

if (\_bonus.RoleId > 0)

{

var selectedRole = Roles.FirstOrDefault(r => r.Id == \_bonus.RoleId);

if (selectedRole != null)

{

rolePicker.SelectedItem = selectedRole;

}

}

}

private static readonly List<Role> Roles = new List<Role>

{

new Role { Id = 1, Name = "Администратор", Code = "admin" },

new Role { Id = 2, Name = "Курьер", Code = "courier" }

};

private async void OnSaveButtonClicked(object sender, EventArgs e)

{

// Проверка на пустые поля

if (string.IsNullOrWhiteSpace(titleLabel.Text) ||

string.IsNullOrWhiteSpace(descriptionLabel.Text) ||

string.IsNullOrWhiteSpace(priceLabel.Text))

{

await DisplayAlert("Ошибка", "Все обязательные поля должны быть заполнены", "ОК");

return;

}

// Проверка, что роль выбрана

if (rolePicker.SelectedItem == null)

{

await DisplayAlert("Ошибка", "Выберите роль", "ОК");

return;

}

// Подтверждение перед сохранением изменений

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите сохранить изменения?",

"Да",

"Нет"

);

if (!isConfirmed)

{

// Если пользователь выбрал "Нет", сохранение отменяется

return;

}

// Получаем выбранную роль

var selectedRole = (Role)rolePicker.SelectedItem;

var updatedBonus = new Bonus

{

Title = titleLabel.Text,

Description = descriptionLabel.Text,

Price = priceLabel.Text,

RoleId = selectedRole.Id

};

var options = new JsonSerializerOptions

{

PropertyNamingPolicy = JsonNamingPolicy.CamelCase // Преобразует ключи в camelCase

};

var jsonContent = new StringContent(JsonSerializer.Serialize(updatedBonus, options), Encoding.UTF8, "application/json");

// Запрос серверу

try

{

// Скрываем форму и показываем индикатор загрузки

MainContent.IsVisible = false;

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

// Отправляем запрос и записываем ответ в response

var token = Preferences.Get("UserToken", string.Empty);

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", token);

HttpResponseMessage response = await \_httpClient.PutAsync($"http://courseproject4/api/bonus/{\_bonus.Id}", jsonContent);

if (response.StatusCode == System.Net.HttpStatusCode.OK)

{

\_bonus.Title = updatedBonus.Title;

\_bonus.Description = updatedBonus.Description;

\_bonus.Price = updatedBonus.Price;

\_bonus.RoleId = updatedBonus.RoleId; // Обновляем RoleId

// Обновляем UI

titleLabel.Text = updatedBonus.Title;

descriptionLabel.Text = updatedBonus.Description;

priceLabel.Text = updatedBonus.Price;

await DisplayAlert("Успех", "Бонус успешно обновлён!", "Вернуться на главную");

await Navigation.PushAsync(new Views.Home(\_user, token));

}

else if (response.StatusCode == System.Net.HttpStatusCode.UnprocessableEntity)

{

var errorContent = await response.Content.ReadAsStringAsync();

try

{

// Парсим JSON как JsonDocument

using var document = JsonDocument.Parse(errorContent);

var root = document.RootElement;

var message = root.GetProperty("message").GetString();

if (root.TryGetProperty("errors", out var errors))

{

var errorMessages = new List<string>();

// Проходим по всем ошибкам

foreach (var error in errors.EnumerateObject())

{

foreach (var msg in error.Value.EnumerateArray())

{

errorMessages.Add(System.Text.RegularExpressions.Regex.Unescape(msg.GetString()));

}

}

var combinedErrors = string.Join("\n", errorMessages);

await DisplayAlert("Ошибка валидации", combinedErrors, "ОК");

}

else

{

await DisplayAlert("Ошибка", message ?? "Произошла ошибка валидации.", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Не удалось обработать ответ: {ex.Message}\n\nОтвет: {errorContent}", "ОК");

}

}

else

{

var responseContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Не удалось обновить бонус: {responseContent}", "OK");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "OK");

}

finally

{

// Восстанавливаем интерфейс, если авторизация не удалась

MainContent.IsVisible = true;

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

}

}

}

***Страница курьеров***

На листинге 57 представлена разметка XAML для страницы AllCouriers в приложении AdminSamokat. Интерфейс включает заголовок, отображающий список курьеров, а также индикатор загрузки и сообщение об отсутствии данных. Курьеры представлены в виде карточек внутри CollectionView, содержащих имя, фамилию и логин. Если список пуст, пользователю выводится соответствующее сообщение.

***Листинг 57. Страница курьеров AllCouriers.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Couries.AllCouriers">

<ScrollView>

<Grid>

<!-- Индикатор загрузки по центру экрана -->

<ActivityIndicator x:Name="LoadingIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center"

VerticalOptions="Center"

Color="Gray" />

<!-- Сообщение об отсутствии курьеров -->

<Label x:Name="EmptyMessageLabel"

Text="Курьеры отсутствуют"

FontSize="18"

TextColor="Gray"

HorizontalOptions="Center"

VerticalOptions="Center"

IsVisible="False" />

<!-- Вертикальный стек с заголовком и списком -->

<VerticalStackLayout Spacing="10" Padding="10">

<!-- Заголовок страницы -->

<Label Text="Список курьеров:"

FontSize="20"

FontAttributes="Bold"

TextColor="#333333"

HorizontalOptions="Center"

VerticalOptions="Center" />

<!-- CollectionView для отображения курьеров -->

<CollectionView x:Name="UsersCollectionView"

SelectionMode="Single"

SelectionChanged="OnCourierSelected"

IsVisible="False">

<CollectionView.ItemTemplate>

<DataTemplate>

<!-- Оборачиваем в StackLayout с отступами -->

<StackLayout Padding="10" Spacing="10">

<Frame Padding="15" CornerRadius="10" HasShadow="True" BackgroundColor="White">

<Grid RowDefinitions="Auto,Auto,Auto" ColumnDefinitions="Auto,\*">

<Image Source="courier\_icon.png"

HeightRequest="50"

WidthRequest="50"

Aspect="AspectFill"

Grid.RowSpan="3"

Margin="0,0,15,0"/>

<Label Text="{Binding Name}"

FontAttributes="Bold"

FontSize="16"

Grid.Column="1" />

<Label Text="{Binding Surname}"

FontSize="14"

Grid.Row="1"

Grid.Column="1" />

<Label Text="{Binding Login}"

FontSize="12"

TextColor="Gray"

Grid.Row="2"

Grid.Column="1" />

</Grid>

</Frame>

</StackLayout>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

</VerticalStackLayout>

</Grid>

</ScrollView>

</ContentPage>

На листинге 58 представлен код C# для AllCouriers.xaml.cs, который управляет загрузкой и отображением списка курьеров. При инициализации страницы выполняется запрос к API для получения списка пользователей, отфильтрованных по роли курьера (RoleId = 2). Если запрос успешен, данные обновляются в ObservableCollection, а в случае отсутствия курьеров отображается сообщение. При выборе курьера происходит переход на страницу его подробной информации. В коде также реализована обработка ошибок, включая неактуальный токен, при котором пользователь перенаправляется на страницу авторизации.

***Листинг 58. C# код страницы AllCouriers.xaml***

public partial class AllCouriers : ContentPage

{

private User \_user;

private string \_token;

private readonly HttpClient \_httpClient = new HttpClient();

public ObservableCollection<User> Users { get; set; } = new ObservableCollection<User>();

public AllCouriers(User user, string token)

{

InitializeComponent();

\_user = user;

\_token = token;

UsersCollectionView.ItemsSource = Users;

LoadUsers();

}

private async void LoadUsers()

{

try

{

// Показываем индикатор загрузки

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

var token = Preferences.Get("UserToken", string.Empty);

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", token);

var response = await \_httpClient.GetAsync("http://courseproject4/api/profile");

if (response.IsSuccessStatusCode)

{

var content = await response.Content.ReadAsStringAsync();

var users = JsonSerializer.Deserialize<List<User>>(content);

Users.Clear();

// Фильтруем пользователей с ролью 2

var couriers =

users.Where(user => user.RoleId == 2); // Предполагается, что RoleId указывает роль пользователя

foreach (var courier in couriers)

{

Users.Add(courier);

}

// Управляем видимостью элементов

UsersCollectionView.IsVisible = Users.Count > 0;

EmptyMessageLabel.IsVisible = Users.Count == 0;

}

else if (response.StatusCode == System.Net.HttpStatusCode.Unauthorized)

{

// Ошибка авторизации

await DisplayAlert("Ошибка", "Ваш токен не актуален. Авторизуйтесь заново!", "ОК");

await Navigation.PushAsync(new Views.Auth.Login());

}

else

{

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Код: {response.StatusCode}, Ответ: {errorContent}", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

finally

{

// Скрываем индикатор загрузки

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

}

}

private async void OnCourierSelected(object sender, SelectionChangedEventArgs e)

{

// Получаем выбранного курьера

if (e.CurrentSelection.FirstOrDefault() is User selectedCourier)

{

// Переход на страницу информации о курьере

await Navigation.PushAsync(new Courier(selectedCourier, \_user, \_token));

}

// Сбрасываем выбор

((CollectionView)sender).SelectedItem = null;

}

}

***Страница курьера***

На листинге 59 представлена разметка XAML для страницы Courier в приложении AdminSamokat. Интерфейс включает карточки с основной информацией о курьере, его штрафах и статусе, а также кнопки для редактирования профиля и удаления курьера. Для удобства пользователей предусмотрены индикаторы загрузки, отображаемые при получении данных с сервера.

***Листинг 59. Страница курьера Courier.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Couries.Courier">

<Grid>

<!-- Основное содержимое -->

<Grid x:Name="MainContent" RowDefinitions="\*,Auto" Padding="20">

<!-- Содержимое страницы -->

<ScrollView Grid.Row="0">

<VerticalStackLayout Spacing="20">

<!-- Карточка: Основная информация -->

<Frame CornerRadius="10" Padding="15" HasShadow="True" BackgroundColor="White">

<VerticalStackLayout Spacing="10">

<Label Text="Основная информация"

FontSize="18"

FontAttributes="Bold"

TextColor="#6200EE" />

<Label x:Name="CourierFullNameLabel"

FontSize="16"

TextColor="Black" />

<Label x:Name="CourierLoginLabel"

FontSize="14"

TextColor="Gray" />

</VerticalStackLayout>

</Frame>

<!-- Карточка: Штрафы -->

<Frame CornerRadius="10" Padding="15" HasShadow="True" BackgroundColor="White">

<VerticalStackLayout Spacing="10">

<Label Text="Штраф"

FontSize="18"

FontAttributes="Bold"

TextColor="#6200EE" />

<ActivityIndicator x:Name="LoadingFineIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center" />

<Label x:Name="FineLabel"

FontSize="16"

TextColor="Black"

HorizontalOptions="Center"

HorizontalTextAlignment="Center" />

</VerticalStackLayout>

</Frame>

<!-- Карточка: Статус -->

<Frame CornerRadius="10" Padding="15" HasShadow="True" BackgroundColor="White">

<VerticalStackLayout Spacing="10">

<Label Text="Статус"

FontSize="18"

FontAttributes="Bold"

TextColor="#6200EE" />

<ActivityIndicator x:Name="LoadingStatusIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center" />

<Label x:Name="StatusLabel"

FontSize="16"

TextColor="Black"

HorizontalOptions="Center"

HorizontalTextAlignment="Center" />

</VerticalStackLayout>

</Frame>

<!-- Кнопки внизу -->

<VerticalStackLayout Grid.Row="1" Spacing="10">

<Button Text="Изменить профиль"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"

Clicked="OnEditButtonClicked" />

<Button Text="Удалить курьера"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"

Clicked="OnDeleteButtonClicked" />

</VerticalStackLayout>

</VerticalStackLayout>

</ScrollView>

</Grid>

<!-- Индикатор загрузки удаления -->

<ActivityIndicator x:Name="DeleteLoadingIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center"

VerticalOptions="Center" />

</Grid>

</ContentPage>

На листинге 60 представлен код C# для Courier.xaml.cs, содержащий логику загрузки данных и взаимодействия с API. При инициализации страницы загружается информация о курьере, включая его ФИО, логин, штрафы и статус. Для получения данных отправляются HTTP-запросы, при этом в случае ошибки выводится соответствующее сообщение. Также реализованы обработчики событий для кнопок: при нажатии «Изменить профиль» происходит переход на страницу редактирования, а при нажатии «Удалить курьера» отображается диалог подтверждения, после чего отправляется запрос на удаление курьера.

***Листинг 60. C# код страницы Courier.xaml***

public partial class Courier : ContentPage

{

private User \_courier;

private User \_user;

private string \_token;

private readonly HttpClient \_httpClient = new HttpClient();

public Courier(User courier, User user, string token)

{

InitializeComponent();

\_user = user;

\_token = token;

\_courier = courier;

// Отобразить данные курьера

CourierFullNameLabel.Text = $"{\_courier.Surname} {\_courier.Name} {\_courier.Patronymic}";

CourierLoginLabel.Text = \_courier.Login;

// Загружаем штраф

LoadFineDetails();

// Загружаем статус

LoadStatusDetails();

}

private async void LoadFineDetails()

{

var token = Preferences.Get("UserToken", string.Empty);

\_httpClient.DefaultRequestHeaders.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", token);

try

{

// Включаем индикатор загрузки у штрафа

LoadingFineIndicator.IsRunning = true;

LoadingFineIndicator.IsVisible = true;

// Загружаем информацию о штрафах

var responseFine = await \_httpClient.GetAsync($"http://courseproject4/api/fine/{\_courier.FineId}");

var contentFine = await responseFine.Content.ReadAsStringAsync();

if (responseFine.IsSuccessStatusCode)

{

var fine = JsonSerializer.Deserialize<Fine>(contentFine);

FineLabel.Text = fine.Description;

}

else

{

FineLabel.Text = "Не удалось загрузить штраф";

await DisplayAlert("Ошибка", $"Код: {responseFine.StatusCode}, Ответ: {contentFine}", "OK");

}

}

catch (Exception ex)

{

FineLabel.Text = "Ошибка при загрузке штрафа";

await DisplayAlert("Ошибка", ex.Message, "OK");

}

finally

{

// Выключаем индикатор загрузки

LoadingFineIndicator.IsRunning = false;

LoadingFineIndicator.IsVisible = false;

}

}

private async void LoadStatusDetails()

{

var token = Preferences.Get("UserToken", string.Empty);

\_httpClient.DefaultRequestHeaders.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", token);

try

{

// Включаем индикатор загрузки у статуса

LoadingStatusIndicator.IsRunning = true;

LoadingStatusIndicator.IsVisible = true;

// Загружаем информацию о статусе

var responseStatus = await \_httpClient.GetAsync($"http://courseproject4/api/status/{\_courier.StatusId}");

var contentStatus = await responseStatus.Content.ReadAsStringAsync();

if (responseStatus.IsSuccessStatusCode)

{

var status = JsonSerializer.Deserialize<Status>(contentStatus);

StatusLabel.Text = status.Name;

}

else

{

FineLabel.Text = "Не удалось загрузить статус";

await DisplayAlert("Ошибка", $"Код: {responseStatus.StatusCode}, Ответ: {contentStatus}", "OK");

}

}

catch (Exception ex)

{

FineLabel.Text = "Ошибка при загрузке статуса";

await DisplayAlert("Ошибка", ex.Message, "OK");

}

finally

{

// Выключаем индикатор загрузки

LoadingStatusIndicator.IsRunning = false;

LoadingStatusIndicator.IsVisible = false;

}

}

private async void OnEditButtonClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new EditProfileCourier(\_courier, \_user, \_token));

}

private async void OnDeleteButtonClicked(object sender, EventArgs e)

{

// Подтверждение перед удалением

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите удалить этого курьера?",

"Да",

"Нет"

);

if (!isConfirmed)

{

// Если пользователь выбрал "Нет", удаление отменяется

return;

}

try

{

// Скрываем основное содержимое и показываем индикатор

MainContent.IsVisible = false;

DeleteLoadingIndicator.IsRunning = true;

DeleteLoadingIndicator.IsVisible = true;

// Настраиваем заголовки и токен

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_token);

// Отправляем DELETE-запрос на сервер

var response = await \_httpClient.DeleteAsync($"http://courseproject4/api/profile/{\_courier.Id}");

if (response.IsSuccessStatusCode)

{

await DisplayAlert("Успех", "Курьер успешно удалён.", "ОК");

await Navigation.PushAsync(new Home(\_user, \_token));

}

else

{

var responseContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Не удалось удалить курьера: {response.StatusCode} - {responseContent}", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

finally

{

// Возвращаем основное содержимое и скрываем индикатор

MainContent.IsVisible = true;

DeleteLoadingIndicator.IsRunning = false;

DeleteLoadingIndicator.IsVisible = false;

}

}

}

***Страница редактирования профиля курьера***

На листинге 61 представлена разметка XAML для страницы редактирования профиля курьера EditProfileCourier в приложении AdminSamokat. Интерфейс включает заголовок, поля ввода для фамилии, имени, отчества, логина, а также возможность изменения пароля. Реализован выпадающий список для выбора статуса курьера. Для удобства пользователей предусмотрен индикатор загрузки, который отображается во время обновления данных.

***Листинг 61. Страница редактирования профиля курьера EditProfileCourier.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Couries.EditProfileCourier"

Shell.NavBarIsVisible="False"

BackgroundColor="#F9F9F9">

<Grid>

<!-- Основной контент с прокруткой -->

<ScrollView x:Name="MainContent" IsVisible="True">

<StackLayout Padding="20" Spacing="10">

<!-- Заголовок -->

<Label Text="Редактировать профиль"

FontSize="28"

FontAttributes="Bold"

HorizontalOptions="Center"

VerticalOptions="Center"

Margin="0,20,0,20"/>

<!-- Раздел: Основные данные -->

<Label Text="Основные данные" FontSize="20" FontAttributes="Bold" TextColor="#6200EE"/>

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray">

<StackLayout>

<Label Text="Фамилия" FontAttributes="Bold" />

<Entry x:Name="surnameLabel" Placeholder="Введите фамилию" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray">

<StackLayout>

<Label Text="Имя" FontAttributes="Bold" />

<Entry x:Name="nameLabel" Placeholder="Введите имя" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray">

<StackLayout>

<Label Text="Отчество" FontAttributes="Bold" />

<Entry x:Name="patronymicLabel" Placeholder="Введите отчество" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray">

<StackLayout>

<Label Text="Логин" FontAttributes="Bold" />

<Entry x:Name="loginLabel" Placeholder="Введите логин" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<StackLayout Orientation="Horizontal" VerticalOptions="Center">

<CheckBox x:Name="checkPassword" IsChecked="True" CheckedChanged="OnCheckPasswordChanged" VerticalOptions="Center" />

<Label Text="Не изменять пароль" VerticalOptions="Center" />

</StackLayout>

<Frame x:Name="password" IsVisible="False" Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray">

<StackLayout>

<Label Text="Пароль" FontAttributes="Bold" />

<Entry x:Name="passwordLabel" Placeholder="Введите пароль" IsPassword="True" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<Frame IsVisible="False" x:Name="confirmPassword" Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray">

<StackLayout>

<Label Text="Подтвердите пароль" FontAttributes="Bold" />

<Entry x:Name="confirmPasswordLabel" Placeholder="Подтвердите пароль" IsPassword="True" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<!-- Раздел: Статус -->

<Label Text="Статус" FontSize="20" FontAttributes="Bold" TextColor="#6200EE"/>

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray">

<StackLayout>

<Picker x:Name="StatusPicker" Title="Выберите статус" ItemsSource="{Binding Statuses}" ItemDisplayBinding="{Binding Name}" />

</StackLayout>

</Frame>

<!-- Кнопка сохранения -->

<Button Text="Сохранить"

Clicked="OnSaveButtonClicked"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"/>

</StackLayout>

</ScrollView>

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingIndicator"

IsVisible="False"

IsRunning="False"

Color="Gray"

HorizontalOptions="Center"

VerticalOptions="Center"/>

</Grid>

</ContentPage>

На листинге 62 представлен код C# для EditProfileCourier.xaml.cs, содержащий логику редактирования профиля курьера. При загрузке страницы заполняются поля данными текущего пользователя, а также загружается список доступных статусов. При нажатии кнопки «Сохранить» выполняется валидация введённых данных и отправляется HTTP-запрос на сервер для обновления профиля. В случае успешного обновления отображается уведомление и выполняется переход на главную страницу. Если сервер вернёт ошибку валидации, пользователю отобразится детализированное сообщение. Также реализована возможность изменения пароля, которая активируется при снятии соответствующего флажка.

***Листинг 62. C# код страницы EditProfileCourier.xaml***

public partial class EditProfileCourier : ContentPage

{

// Инициализация HTTP клиента

private readonly HttpClient \_httpClient = new HttpClient();

private User \_courier;

private User \_user;

private string \_token;

public EditProfileCourier(User courier, User user, string token)

{

InitializeComponent();

\_user = user;

\_token = token;

\_courier = courier;

surnameLabel.Text = courier.Surname;

nameLabel.Text = courier.Name;

if (string.IsNullOrEmpty(courier.Patronymic))

{

patronymicLabel.Text = "";

}

else

{

patronymicLabel.Text = courier.Patronymic;

}

loginLabel.Text = courier.Login;

LoadStatuses();

}

private async void OnSaveButtonClicked(object sender, EventArgs e)

{

// Проверка на пустые поля

if (string.IsNullOrWhiteSpace(surnameLabel.Text) ||

string.IsNullOrWhiteSpace(nameLabel.Text) ||

string.IsNullOrWhiteSpace(loginLabel.Text))

{

await DisplayAlert("Ошибка", "Все обязательные поля должны быть заполнены", "ОК");

return;

}

if (!checkPassword.IsChecked && passwordLabel.Text != confirmPasswordLabel.Text)

{

await DisplayAlert("Ошибка", "Пароли не совпадают", "ОК");

return;

}

// Подтверждение перед сохранением изменений

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите сохранить изменения?",

"Да",

"Нет"

);

if (!isConfirmed)

{

// Если пользователь выбрал "Нет", сохранение отменяется

return;

}

var selectedStatus = StatusPicker.SelectedItem as Status;

// Формируем данные для обновления

var updatedUser = new Dictionary<string, object>

{

{ "surname", surnameLabel.Text },

{ "name", nameLabel.Text },

{ "patronymic", string.IsNullOrEmpty(patronymicLabel.Text) ? null : patronymicLabel.Text },

// Добавляем выбранные ID статуса

{ "status\_id", selectedStatus.Id }

};

// Добавляем поле "login", только если оно изменилось

if (loginLabel.Text != \_courier.Login)

{

updatedUser.Add("login", loginLabel.Text);

}

// Добавляем поле "password", только если галочка снята

if (!checkPassword.IsChecked)

{

updatedUser.Add("password", passwordLabel.Text);

}

// Настраиваем сериализацию для преобразования ключей в нижний регистр

var options = new JsonSerializerOptions

{

PropertyNamingPolicy = JsonNamingPolicy.CamelCase // Преобразует ключи в camelCase

};

var jsonContent = new StringContent(JsonSerializer.Serialize(updatedUser, options), Encoding.UTF8, "application/json");

// Запрос серверу

try

{

// Скрываем форму и показываем индикатор загрузки

MainContent.IsVisible = false;

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

// Отправляем запрос и записываем ответ в response

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_token);

HttpResponseMessage response = await \_httpClient.PutAsync($"http://courseproject4/api/profile/{\_courier.Id}", jsonContent);

if (response.StatusCode == System.Net.HttpStatusCode.OK)

{

// Обновляем локальную копию данных пользователя

\_courier.Surname = surnameLabel.Text;

\_courier.Name = nameLabel.Text;

\_courier.Patronymic = string.IsNullOrEmpty(patronymicLabel.Text) ? null : patronymicLabel.Text;

\_courier.StatusId = selectedStatus.Id; // Сохраняем выбранный статус

if (loginLabel.Text != \_courier.Login)

{

\_courier.Login = loginLabel.Text;

}

// Обновляем UI

surnameLabel.Text = \_courier.Surname;

nameLabel.Text = \_courier.Name;

patronymicLabel.Text = \_courier.Patronymic ?? "";

loginLabel.Text = \_courier.Login;

await DisplayAlert("Успех", "Профиль успешно обновлён!", "Вернуться на главную");

await Navigation.PushAsync(new Views.Home(\_user, \_token));

Navigation.RemovePage(this); // Убираем текущую страницу из стека

}

else if (response.StatusCode == System.Net.HttpStatusCode.UnprocessableEntity)

{

var errorContent = await response.Content.ReadAsStringAsync();

try

{

// Парсим JSON как JsonDocument

using var document = JsonDocument.Parse(errorContent);

var root = document.RootElement;

var message = root.GetProperty("message").GetString();

if (root.TryGetProperty("errors", out var errors))

{

var errorMessages = new List<string>();

// Проходим по всем ошибкам

foreach (var error in errors.EnumerateObject())

{

foreach (var msg in error.Value.EnumerateArray())

{

errorMessages.Add(System.Text.RegularExpressions.Regex.Unescape(msg.GetString()));

}

}

var combinedErrors = string.Join("\n", errorMessages);

await DisplayAlert("Ошибка валидации", combinedErrors, "ОК");

}

else

{

await DisplayAlert("Ошибка", message ?? "Произошла ошибка валидации.", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Не удалось обработать ответ: {ex.Message}\n\nОтвет: {errorContent}", "ОК");

}

}

else

{

var responseContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Не удалось обновить профиль: {responseContent}", "OK");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "OK");

}

finally

{

// Восстанавливаем интерфейс, если изменить не удалось

MainContent.IsVisible = true;

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

}

}

private async void LoadStatuses()

{

try

{

LoadingIndicator.IsVisible = true;

LoadingIndicator.IsRunning = true;

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_token);

// Загружаем список статусов

var statusesResponse = await \_httpClient.GetAsync("http://courseproject4/api/status");

if (statusesResponse.IsSuccessStatusCode)

{

var statusesContent = await statusesResponse.Content.ReadAsStringAsync();

var statuses = JsonSerializer.Deserialize<List<Status>>(statusesContent);

// Привязываем список статусов к Picker

StatusPicker.ItemsSource = statuses;

// Устанавливаем выбранное значение, соответствующее текущему статусу пользователя

StatusPicker.SelectedItem = statuses.FirstOrDefault(s => s.Id == \_courier.StatusId);

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка при загрузке данных: {ex.Message}", "OK");

}

finally

{

LoadingIndicator.IsVisible = false;

LoadingIndicator.IsRunning = false;

}

}

private void OnCheckPasswordChanged(object sender, CheckedChangedEventArgs e)

{

bool isChecked = e.Value;

// Если галочка снята, показываем поля для ввода пароля

password.IsVisible = !isChecked;

confirmPassword.IsVisible = !isChecked;

}

}

***Страница штрафов***

На листинге 63 представлена разметка XAML для страницы отображения списка штрафов AllFines в приложении AdminSamokat. Интерфейс включает индикатор загрузки, отображаемый во время загрузки данных, сообщение о пустом списке штрафов, а также коллекцию штрафов, представленных в виде карточек с описанием. В нижней части экрана расположена кнопка для создания нового штрафа.

***Листинг 63. Страница штрафов AllFines.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Fines.AllFines">

<Grid RowDefinitions="\*,Auto">

<!-- Верхняя часть для содержимого -->

<Grid Grid.Row="0">

<!-- Индикатор загрузки по центру экрана -->

<ActivityIndicator x:Name="LoadingIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center"

VerticalOptions="Center"

Color="Gray" />

<!-- Сообщение об отсутствии штрафов -->

<Label x:Name="EmptyMessageLabel"

Text="Штрафы отсутствуют"

FontSize="18"

TextColor="Gray"

HorizontalOptions="Center"

VerticalOptions="Center"

IsVisible="False" />

<!-- ScrollView для прокрутки содержимого -->

<ScrollView>

<VerticalStackLayout Spacing="10" Padding="10">

<!-- Заголовок страницы -->

<Label Text="Список штрафов:"

FontSize="20"

FontAttributes="Bold"

TextColor="#333333"

HorizontalOptions="Center"

VerticalOptions="Center" />

<!-- CollectionView для отображения штрафов -->

<CollectionView x:Name="FinesCollectionView"

SelectionMode="Single"

SelectionChanged="OnFineSelected"

IsVisible="False">

<CollectionView.ItemTemplate>

<DataTemplate>

<!-- Оборачиваем в StackLayout с отступами -->

<StackLayout Padding="10" Spacing="10">

<Frame Padding="15" CornerRadius="10" HasShadow="True" BackgroundColor="White">

<Grid RowDefinitions="Auto,Auto,Auto" ColumnDefinitions="Auto,\*">

<Image Source="fine\_icon.png"

HeightRequest="50"

WidthRequest="50"

Aspect="AspectFill"

Grid.RowSpan="3"

Margin="0,0,15,0"/>

<Label Text="{Binding Description}"

FontAttributes="Bold"

FontSize="16"

Grid.Column="1" />

</Grid>

</Frame>

</StackLayout>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

</VerticalStackLayout>

</ScrollView>

</Grid>

<!-- Нижняя часть для кнопки -->

<Button Grid.Row="1"

Text="Создать штраф"

FontAttributes="Bold"

BackgroundColor="#6200EE"

TextColor="White"

CornerRadius="25"

Padding="15"

HorizontalOptions="FillAndExpand"

Margin="10"

Clicked="OnCreateFineClicked" />

</Grid>

</ContentPage>

На листинге 64 представлен код C# для AllFines.xaml.cs, содержащий логику загрузки и отображения штрафов. При инициализации страницы выполняется HTTP-запрос к API для получения списка штрафов. В зависимости от результата запроса список обновляется, а также управляется видимость элементов (индикатор загрузки, сообщение об отсутствии данных). Реализована обработка выбора штрафа, при котором выполняется переход на страницу деталей штрафа, а также переход на страницу создания нового штрафа по нажатию соответствующей кнопки.

***Листинг 64. C# код страницы AllFines.xaml***

public partial class AllFines : ContentPage

{

private User \_user;

private string \_token;

private readonly HttpClient \_httpClient = new HttpClient();

public ObservableCollection<Fine> Fines { get; set; } = new ObservableCollection<Fine>();

public AllFines(User user, string token)

{

InitializeComponent();

\_user = user;

\_token = token;

FinesCollectionView.ItemsSource = Fines;

LoadFines();

}

private async void LoadFines()

{

try

{

// Показываем индикатор загрузки

LoadingIndicator.IsVisible = true;

LoadingIndicator.IsRunning = true;

var token = Preferences.Get("UserToken", string.Empty);

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", token);

var response = await \_httpClient.GetAsync("http://courseproject4/api/fine");

if (response.IsSuccessStatusCode)

{

var content = await response.Content.ReadAsStringAsync();

var fines = JsonSerializer.Deserialize<List<Fine>>(content);

Fines.Clear();

foreach (var fine in fines)

{

Fines.Add(fine);

}

// Управляем видимостью элементов в зависимости от количества бонусов

FinesCollectionView.IsVisible = Fines.Count > 0;

EmptyMessageLabel.IsVisible = Fines.Count == 0;

}

else if (response.StatusCode == System.Net.HttpStatusCode.Unauthorized)

{

// Ошибка авторизации

await DisplayAlert("Ошибка", "Ваш токен не актуален. Авторизуйтесь заново!", "ОК");

await Navigation.PushAsync(new Views.Auth.Login());

}

else

{

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Код: {response.StatusCode}, Ответ: {errorContent}", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

finally

{

// Скрываем индикатор загрузки после завершения загрузки данных

LoadingIndicator.IsVisible = false;

LoadingIndicator.IsRunning = false;

}

}

private async void OnFineSelected(object sender, SelectionChangedEventArgs e)

{

// Получаем выбранного пользователя

if (e.CurrentSelection.FirstOrDefault() is Fine selectedFine)

{

// Переход на страницу информации о бонусе

await Navigation.PushAsync(new FinePage(selectedFine, \_user, \_token));

}

// Сбрасываем выбор

((CollectionView)sender).SelectedItem = null;

}

private async void OnCreateFineClicked(object sender, EventArgs e)

{

// Переход на страницу создания штрафа

await Navigation.PushAsync(new CreateFinePage(\_user, \_token));

}

}

***Страница создания штрафа***

На листинге 65 представлена разметка XAML для страницы создания штрафа CreateFinePage в приложении AdminSamokat. Интерфейс включает логотип, текстовое описание, поле ввода для описания штрафа и кнопку создания. Для удобства пользователей реализован индикатор загрузки, который отображается во время обработки запроса, а также механизм скрытия основной формы на время отправки данных.

***Листинг 65. Страница создания штрафа CreateFinePage.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Fines.CreateFinePage"

Shell.NavBarIsVisible="False"

BackgroundColor="#F9F9F9">

<!-- Основной контейнер -->

<Grid>

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center"

VerticalOptions="Center"

Color="Gray" />

<!-- Основная форма -->

<ScrollView x:Name="CreateForm" IsVisible="True">

<StackLayout Padding="20" Spacing="20">

<!-- Логотип -->

<Image Source="logo.png" HorizontalOptions="Center" WidthRequest="200" HeightRequest="200"/>

<!-- Текстовое описание -->

<Label Text="На этой странице вы можете создать штраф!"

FontSize="24"

HorizontalTextAlignment="Center"

VerticalOptions="Center"

Margin="0,20,0,20"/>

<!-- Форма создания -->

<Frame Padding="15" CornerRadius="15" BackgroundColor="White" Margin="0,20">

<StackLayout Spacing="15">

<Frame BackgroundColor="#F0F0F0" CornerRadius="10">

<Entry x:Name="descriptionEntry"

Placeholder="Введите описание"

Margin="5"/>

</Frame>

</StackLayout>

</Frame>

<!-- Кнопка создания -->

<Button Text="Создать"

Clicked="OnCreateButtonClicked"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"/>

</StackLayout>

</ScrollView>

</Grid>

</ContentPage>

На листинге 66 представлен код C# для CreateFinePage.xaml.cs, содержащий логику создания нового штрафа. При нажатии кнопки "Создать" проверяется заполненность обязательных полей, после чего пользователь подтверждает свое действие. Далее формируется HTTP-запрос с данными штрафа, отправляется POST-запрос к API, и в случае успешного ответа выполняется переход на главную страницу. Если сервер вернул ошибки валидации, они обрабатываются и отображаются пользователю. В случае сетевых ошибок также выводится соответствующее сообщение.

***Листинг 66. C# код страницы CreateFinePage.xaml***

public partial class CreateFinePage : ContentPage

{

private User \_user;

private string \_token;

private readonly HttpClient \_httpClient = new HttpClient();

public CreateFinePage(User user, string token)

{

InitializeComponent();

\_user = user;

\_token = token;

}

private async void OnCreateButtonClicked(object sender, EventArgs e)

{

// Проверка на пустые поля

if (

string.IsNullOrWhiteSpace(descriptionEntry.Text))

{

await DisplayAlert("Ошибка", "Все обязательные поля должны быть заполнены", "ОК");

return;

}

// Подтверждение перед созданием

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите добавить новый штраф?",

"Да",

"Нет"

);

if (!isConfirmed)

{

// Если пользователь выбрал "Нет", создание отменяется

return;

}

// Сбор данных из формы

string description = descriptionEntry.Text;

var createData = new MultipartFormDataContent

{

{ new StringContent(description), "description" },

};

try

{

// Отображаем индикатор загрузки и скрываем форму

CreateForm.IsVisible = false;

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

\_httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", \_token);

HttpResponseMessage response = await \_httpClient.PostAsync("http://courseproject4/api/fine", createData);

if (response.IsSuccessStatusCode)

{

await DisplayAlert("Успех", "Штраф успешно добавлен", "ОК");

await Navigation.PushAsync(new Views.Home(\_user, \_token));

}

else if (response.StatusCode == System.Net.HttpStatusCode.UnprocessableEntity)

{

var errorContent = await response.Content.ReadAsStringAsync();

try

{

// Парсим JSON как JsonDocument

using var document = JsonDocument.Parse(errorContent);

var root = document.RootElement;

var message = root.GetProperty("message").GetString();

if (root.TryGetProperty("errors", out var errors))

{

var errorMessages = new List<string>();

// Проходим по всем ошибкам

foreach (var error in errors.EnumerateObject())

{

foreach (var msg in error.Value.EnumerateArray())

{

errorMessages.Add(System.Text.RegularExpressions.Regex.Unescape(msg.GetString()));

}

}

var combinedErrors = string.Join("\n", errorMessages);

await DisplayAlert("Ошибка валидации", combinedErrors, "ОК");

}

else

{

await DisplayAlert("Ошибка", message ?? "Произошла ошибка валидации.", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Не удалось обработать ответ: {ex.Message}\n\nОтвет: {errorContent}", "ОК");

}

}

else

{

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Код ошибки: {(int)response.StatusCode}\n{errorContent}", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка сети", ex.Message, "ОК");

}

finally

{

// Восстанавливаем форму и скрываем индикатор загрузки

CreateForm.IsVisible = true;

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

}

}

}

***Страница редактирования штрафа***

На листинге 67 представлена разметка XAML для страницы редактирования штрафа EditFine в приложении AdminSamokat. Интерфейс включает заголовок, поле ввода для изменения описания штрафа и кнопку сохранения. Для удобства пользователей реализован индикатор загрузки, который отображается во время выполнения запроса к серверу.

***Листинг 67. Страница редактирования штрафа EditFine.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Fines.EditFine"

Shell.NavBarIsVisible="False"

BackgroundColor="#F9F9F9">

<Grid>

<!-- Основной контент с прокруткой -->

<ScrollView x:Name="MainContent" IsVisible="True">

<StackLayout Padding="20" Spacing="20">

<!-- Заголовок -->

<Label Text="Редактировать штраф"

FontSize="28"

FontAttributes="Bold"

HorizontalOptions="Center"

VerticalOptions="Center"

Margin="0,20,0,20"/>

<!-- Поля для ввода -->

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray" Margin="0,5">

<StackLayout>

<Label Text="Описание" FontAttributes="Bold" />

<Entry x:Name="descriptionLabel" Placeholder="Введите описание" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<!-- Кнопка сохранения -->

<Button Text="Сохранить"

Clicked="OnSaveButtonClicked"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"

Margin="0,20,0,0"/>

</StackLayout>

</ScrollView>

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingIndicator"

IsVisible="False"

IsRunning="False"

Color="Gray"

HorizontalOptions="Center"

VerticalOptions="Center"/>

</Grid>

</ContentPage>

На листинге 68 представлен код C# для EditFine.xaml.cs, содержащий логику редактирования штрафа. При открытии страницы отображается текущее описание штрафа, полученное из переданных данных. После изменения и нажатия кнопки "Сохранить" выполняется проверка заполненности поля, затем пользователь подтверждает действие. Данные отправляются на сервер через HTTP-запрос методом PUT. В случае успешного обновления штрафа выполняется переход на главную страницу. Если сервер возвращает ошибки валидации, они обрабатываются и отображаются пользователю.

***Листинг 68. C# код страницы EditFine.xaml***

public partial class EditFine : ContentPage

{

// Инициализация HTTP клиента

private readonly HttpClient \_httpClient = new HttpClient();

private Fine \_fine;

private User \_user;

private string \_token;

public EditFine(Fine fine, User user, string token)

{

InitializeComponent();

\_fine = fine;

\_user = user;

\_token = token;

descriptionLabel.Text = fine.Description;

}

private async void OnSaveButtonClicked(object sender, EventArgs e)

{

// Проверка на пустые поля

if (string.IsNullOrWhiteSpace(descriptionLabel.Text))

{

await DisplayAlert("Ошибка", "Все обязательные поля должны быть заполнены", "ОК");

return;

}

// Подтверждение перед сохранением изменений

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите сохранить изменения?",

"Да",

"Нет"

);

if (!isConfirmed)

{

// Если пользователь выбрал "Нет", сохранение отменяется

return;

}

// Формируем данные для обновления

var updatedFine = new

{

Description = descriptionLabel.Text,

};

// Настраиваем сериализацию для преобразования ключей в нижний регистр

var options = new JsonSerializerOptions

{

PropertyNamingPolicy = JsonNamingPolicy.CamelCase // Преобразует ключи в camelCase

};

var jsonContent = new StringContent(JsonSerializer.Serialize(updatedFine, options), Encoding.UTF8, "application/json");

// Запрос серверу

try

{

// Скрываем форму и показываем индикатор загрузки

MainContent.IsVisible = false;

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

// Отправляем запрос и записываем ответ в response

var token = Preferences.Get("UserToken", string.Empty);

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", token);

HttpResponseMessage response = await \_httpClient.PutAsync($"http://courseproject4/api/fine/{\_fine.Id}", jsonContent);

if (response.StatusCode == System.Net.HttpStatusCode.OK)

{

\_fine.Description = updatedFine.Description;

// Обновляем UI

descriptionLabel.Text = updatedFine.Description;

await DisplayAlert("Успех", "Штраф успешно обновлён!", "Вернуться на главную");

await Navigation.PushAsync(new Views.Home(\_user, token));

}

else if (response.StatusCode == System.Net.HttpStatusCode.UnprocessableEntity)

{

var errorContent = await response.Content.ReadAsStringAsync();

try

{

// Парсим JSON как JsonDocument

using var document = JsonDocument.Parse(errorContent);

var root = document.RootElement;

var message = root.GetProperty("message").GetString();

if (root.TryGetProperty("errors", out var errors))

{

var errorMessages = new List<string>();

// Проходим по всем ошибкам

foreach (var error in errors.EnumerateObject())

{

foreach (var msg in error.Value.EnumerateArray())

{

errorMessages.Add(System.Text.RegularExpressions.Regex.Unescape(msg.GetString()));

}

}

var combinedErrors = string.Join("\n", errorMessages);

await DisplayAlert("Ошибка валидации", combinedErrors, "ОК");

}

else

{

await DisplayAlert("Ошибка", message ?? "Произошла ошибка валидации.", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Не удалось обработать ответ: {ex.Message}\n\nОтвет: {errorContent}", "ОК");

}

}

else

{

var responseContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Не удалось обновить штраф: {responseContent}", "OK");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "OK");

}

finally

{

// Восстанавливаем интерфейс, если авторизация не удалась

MainContent.IsVisible = true;

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

}

}

}

***Страница назначения штрафа***

На листинге 69 представлена разметка XAML для страницы назначения штрафа FinePage в приложении AdminSamokat. Интерфейс включает описание штрафа, список доступных курьеров, а также кнопки для редактирования, удаления штрафа и его назначения курьеру. Если курьеры отсутствуют, отображается соответствующее сообщение. Для удобства пользователей реализован индикатор загрузки, который показывается во время выполнения сетевых запросов.

***Листинг 69. Страница назначения штрафа FinePage.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Fines.FinePage">

<ScrollView x:Name="MainContent">

<VerticalStackLayout Padding="20" Spacing="15">

<!-- Основная информация о штрафе -->

<Label x:Name="FineDescriptionLabel"

FontSize="20"

FontAttributes="Bold"

HorizontalOptions="Center" />

<!-- Сообщение об отсутствии курьеров -->

<Label x:Name="EmptyMessageLabel"

Text="Курьеры отсутствуют"

FontSize="18"

TextColor="Gray"

HorizontalOptions="Center"

VerticalOptions="Center"

IsVisible="False" />

<!-- CollectionView для отображения курьеров -->

<CollectionView x:Name="UsersCollectionView"

SelectionMode="Single"

SelectionChanged="OnCourierSelected"

IsVisible="True"

Margin="0,10">

<CollectionView.ItemTemplate>

<DataTemplate>

<StackLayout Padding="10" Spacing="10">

<Frame Padding="15" CornerRadius="10" HasShadow="True" BackgroundColor="White">

<Grid RowDefinitions="Auto,Auto,Auto" ColumnDefinitions="Auto,\*,Auto">

<Image Source="courier\_icon.png"

HeightRequest="50"

WidthRequest="50"

Aspect="AspectFill"

Grid.RowSpan="3"

Margin="0,0,15,0"/>

<Label Text="{Binding Name}"

FontAttributes="Bold"

FontSize="16"

Grid.Column="1" />

<Label Text="{Binding Surname}"

FontSize="14"

Grid.Row="1"

Grid.Column="1" />

<Label Text="{Binding Login}"

FontSize="12"

TextColor="Gray"

Grid.Row="2"

Grid.Column="1" />

<!-- Кнопка "Назначить" -->

<Button Text="Назначить"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="20"

HeightRequest="35"

WidthRequest="100"

Padding="0"

VerticalOptions="Center"

HorizontalOptions="Center"

Clicked="OnAssignButtonClicked"

BindingContext="{Binding}"

Grid.Row="0"

Grid.RowSpan="3"

Grid.Column="2" />

</Grid>

</Frame>

</StackLayout>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

<!-- Кнопки после коллекции -->

<VerticalStackLayout Spacing="10">

<Button Text="Изменить"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"

Clicked="OnEditButtonClicked" />

<Button Text="Удалить"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"

Clicked="OnDeleteButtonClicked" />

</VerticalStackLayout>

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingIndicator"

IsRunning="False"

IsVisible="False"

HorizontalOptions="Center"

VerticalOptions="Center" />

</VerticalStackLayout>

</ScrollView>

</ContentPage>

На листинге 70 представлен код C# для FinePage.xaml.cs, содержащий логику загрузки данных и взаимодействия с API. При открытии страницы выполняется запрос к серверу для получения списка курьеров. Пользователь может выбрать курьера и назначить ему штраф, отправив соответствующий HTTP-запрос. Также реализованы функции редактирования и удаления штрафа, с предварительным подтверждением от пользователя. В случае ошибки выводится сообщение с деталями проблемы.

***Листинг 70. C# код страницы FinePage.xaml***

public partial class FinePage : ContentPage

{

private User \_user;

private string \_token;

private Fine \_fine;

private readonly HttpClient \_httpClient = new HttpClient();

public ObservableCollection<User> Users { get; set; } = new ObservableCollection<User>();

public FinePage(Fine fine, User user, string token)

{

InitializeComponent();

\_fine = fine;

\_token = token;

\_user = user;

UsersCollectionView.ItemsSource = Users;

// Отобразить данные штрафа

FineDescriptionLabel.Text = "Кому вы хотите назначить \"" + \_fine.Description + "\"?";

LoadUsers();

}

private async void LoadUsers()

{

try

{

// Показываем индикатор загрузки

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

var token = Preferences.Get("UserToken", string.Empty);

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", token);

var response = await \_httpClient.GetAsync("http://courseproject4/api/profile");

if (response.IsSuccessStatusCode)

{

var content = await response.Content.ReadAsStringAsync();

var users = JsonSerializer.Deserialize<List<User>>(content);

Users.Clear();

// Фильтруем пользователей с ролью 2

var couriers =

users.Where(user => user.RoleId == 2); // Предполагается, что RoleId указывает роль пользователя

foreach (var courier in couriers)

{

Users.Add(courier);

}

// Управляем видимостью элементов

UsersCollectionView.IsVisible = Users.Count > 0;

EmptyMessageLabel.IsVisible = Users.Count == 0;

}

else

{

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Код: {response.StatusCode}, Ответ: {errorContent}", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

finally

{

// Скрываем индикатор загрузки

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

}

}

private async void OnCourierSelected(object sender, SelectionChangedEventArgs e)

{

// Получаем выбранного курьера

if (e.CurrentSelection.FirstOrDefault() is User selectedCourier)

{

// Переход на страницу информации о курьере

await Navigation.PushAsync(new Couries.Courier(selectedCourier, \_user, \_token));

}

// Сбрасываем выбор

((CollectionView)sender).SelectedItem = null;

}

private async void OnEditButtonClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new EditFine(\_fine, \_user, \_token));

}

private async void OnDeleteButtonClicked(object sender, EventArgs e)

{

// Подтверждение перед удалением

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите удалить этот штраф?",

"Да",

"Нет"

);

if (!isConfirmed)

{

// Если пользователь выбрал "Нет", удаление отменяется

return;

}

try

{

// Скрываем основное содержимое и показываем индикатор

MainContent.IsVisible = false;

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

// Настраиваем заголовки и токен

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_token);

// Отправляем DELETE-запрос на сервер

var response = await \_httpClient.DeleteAsync($"http://courseproject4/api/fine/{\_fine.Id}");

if (response.IsSuccessStatusCode)

{

await DisplayAlert("Успех", "Штраф успешно удалён.", "ОК");

await Navigation.PushAsync(new Home(\_user, \_token));

}

else

{

var responseContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Не удалось удалить штраф: {response.StatusCode} - {responseContent}", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

finally

{

// Возвращаем основное содержимое и скрываем индикатор

MainContent.IsVisible = true;

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

}

}

private async void OnAssignButtonClicked(object sender, EventArgs e)

{

// Получаем курьера, связанного с кнопкой

var selectedCourier = ((Button)sender).BindingContext as User;

if (selectedCourier != null)

{

// Проверяем, назначен ли уже этот штраф

if (selectedCourier.FineId == \_fine.Id)

{

await DisplayAlert("Внимание", "Пользователю уже назначен этот штраф.", "ОК");

return; // Прекращаем выполнение, если штраф уже назначен

}

// Подтверждение перед назначением штрафа

bool isConfirmed = await DisplayAlert(

"Подтверждение",

$"Вы точно хотите назначить штраф \"{\_fine.Description}\" курьеру {selectedCourier.Name}?",

"Да",

"Нет"

);

if (!isConfirmed)

{

return; // Прекращаем выполнение, если пользователь отменил действие

}

try

{

// Настраиваем запрос

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_token);

// Формируем данные для обновления

var updateData = new Dictionary<string, object>

{

{ "fine\_id", \_fine.Id } // Устанавливаем ID текущего штрафа

};

var jsonContent = new StringContent(

JsonSerializer.Serialize(updateData),

Encoding.UTF8,

"application/json"

);

// Отправляем запрос на сервер

var response = await \_httpClient.PutAsync($"http://courseproject4/api/profile/{selectedCourier.Id}", jsonContent);

if (response.IsSuccessStatusCode)

{

// Обновляем локально данные курьера

selectedCourier.FineId = \_fine.Id;

await DisplayAlert("Успех", "Штраф \"" + \_fine.Description + $"\" успешно назначен курьеру {selectedCourier.Name}!", "ОК");

var courierToUpdate = Users.FirstOrDefault(u => u.Id == selectedCourier.Id);

if (courierToUpdate != null)

{

courierToUpdate.FineId = \_fine.Id;

}

}

else

{

var errorContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Не удалось назначить штраф: {response.StatusCode} - {errorContent}", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "ОК");

}

}

}

}

***Страница редактирования профиля***

На листинге 71 представлена разметка XAML для страницы редактирования профиля EditProfile в приложении AdminSamokat. Интерфейс содержит заголовок, текстовые поля для изменения фамилии, имени, отчества и логина пользователя, а также опциональные поля для изменения пароля. Для улучшения взаимодействия с пользователем реализован чекбокс, позволяющий скрывать или отображать поля смены пароля. Также предусмотрен индикатор загрузки, который активируется во время обновления данных.

***Листинг 71. Страница редактирования профиля EditProfile.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.EditProfile"

Shell.NavBarIsVisible="False"

BackgroundColor="#F9F9F9">

<Grid>

<!-- Основной контент с прокруткой -->

<ScrollView x:Name="MainContent" IsVisible="True">

<StackLayout Padding="20" Spacing="10">

<!-- Заголовок -->

<Label Text="Редактировать профиль"

FontSize="28"

FontAttributes="Bold"

HorizontalOptions="Center"

VerticalOptions="Center"

Margin="0,20,0,20"/>

<!-- Поля для ввода -->

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray" >

<StackLayout>

<Label Text="Фамилия" FontAttributes="Bold" />

<Entry x:Name="surnameLabel" Placeholder="Введите фамилию" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray">

<StackLayout>

<Label Text="Имя" FontAttributes="Bold" />

<Entry x:Name="nameLabel" Placeholder="Введите имя" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray">

<StackLayout>

<Label Text="Отчество" FontAttributes="Bold" />

<Entry x:Name="patronymicLabel" Placeholder="Введите отчество" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<Frame Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray">

<StackLayout>

<Label Text="Логин" FontAttributes="Bold" />

<Entry x:Name="loginLabel" Placeholder="Введите логин" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<StackLayout Orientation="Horizontal" VerticalOptions="Center">

<CheckBox x:Name="checkPassword" IsChecked="True" CheckedChanged="OnCheckPasswordChanged" VerticalOptions="Center" />

<Label Text="Не изменять пароль" VerticalOptions="Center" />

</StackLayout>

<Frame x:Name="password" IsVisible="False" Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray">

<StackLayout>

<Label Text="Пароль" FontAttributes="Bold" />

<Entry x:Name="passwordLabel" Placeholder="Введите пароль" IsPassword="True" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<Frame IsVisible="False" x:Name="confirmPassword" Padding="10" CornerRadius="10" BackgroundColor="White" BorderColor="Gray">

<StackLayout>

<Label Text="Подтвердите пароль" FontAttributes="Bold" />

<Entry x:Name="confirmPasswordLabel" Placeholder="Подтвердите пароль" IsPassword="True" BackgroundColor="#F5F5F5"/>

</StackLayout>

</Frame>

<!-- Кнопка сохранения -->

<Button Text="Сохранить"

Clicked="OnSaveButtonClicked"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="25"

Padding="15"/>

</StackLayout>

</ScrollView>

<!-- Индикатор загрузки -->

<ActivityIndicator x:Name="LoadingIndicator"

IsVisible="False"

IsRunning="False"

Color="Gray"

HorizontalOptions="Center"

VerticalOptions="Center"/>

</Grid>

</ContentPage>

На листинге 72 представлен код C# для EditProfile.xaml.cs, отвечающий за обработку изменений профиля. При нажатии кнопки сохранения проверяется корректность введённых данных, после чего формируется HTTP-запрос к API для обновления профиля пользователя. В случае успешного запроса обновлённые данные сохраняются локально в Preferences, а пользователь перенаправляется на главную страницу Home. Если возникают ошибки, например, несоответствие паролей или невалидные данные, отображаются соответствующие уведомления. Также реализована обработка ошибок сети и авторизации, включая сценарии, когда токен пользователя становится недействительным.

***Листинг 72. C# код страницы EditProfile.xaml***

public partial class EditProfile : ContentPage

{

// Инициализация HTTP клиента

private readonly HttpClient \_httpClient = new HttpClient();

private User \_user;

private string \_token;

public EditProfile(User user, string token)

{

InitializeComponent();

\_user = user;

\_token = token;

surnameLabel.Text = user.Surname;

nameLabel.Text = user.Name;

if (string.IsNullOrEmpty(user.Patronymic))

{

patronymicLabel.Text = "";

}

else

{

patronymicLabel.Text = user.Patronymic;

}

loginLabel.Text = user.Login;

}

private async void OnSaveButtonClicked(object sender, EventArgs e)

{

// Проверка на пустые поля

if (string.IsNullOrWhiteSpace(surnameLabel.Text) ||

string.IsNullOrWhiteSpace(nameLabel.Text))

{

await DisplayAlert("Ошибка", "Все обязательные поля должны быть заполнены", "ОК");

return;

}

if (!checkPassword.IsChecked && passwordLabel.Text != confirmPasswordLabel.Text)

{

await DisplayAlert("Ошибка", "Пароли не совпадают", "ОК");

return;

}

// Подтверждение перед сохранением изменений

bool isConfirmed = await DisplayAlert(

"Подтверждение",

"Вы уверены, что хотите сохранить изменения?",

"Да",

"Нет"

);

if (!isConfirmed)

{

// Если пользователь выбрал "Нет", сохранение отменяется

return;

}

// Формируем данные для обновления

var updatedUser = new Dictionary<string, object>

{

{ "surname", surnameLabel.Text },

{ "name", nameLabel.Text },

{ "patronymic", string.IsNullOrEmpty(patronymicLabel.Text) ? null : patronymicLabel.Text },

};

// Добавляем поле "login", только если оно изменилось

if (loginLabel.Text != \_user.Login)

{

updatedUser.Add("login", loginLabel.Text);

}

// Добавляем поле "password", только если галочка снята

if (!checkPassword.IsChecked)

{

updatedUser.Add("password", passwordLabel.Text);

}

// Настраиваем сериализацию для преобразования ключей в нижний регистр

var options = new JsonSerializerOptions

{

PropertyNamingPolicy = JsonNamingPolicy.CamelCase // Преобразует ключи в camelCase

};

var jsonContent = new StringContent(JsonSerializer.Serialize(updatedUser, options), Encoding.UTF8, "application/json");

// Запрос серверу

try

{

// Скрываем форму и показываем индикатор загрузки

MainContent.IsVisible = false;

LoadingIndicator.IsRunning = true;

LoadingIndicator.IsVisible = true;

// Отправляем запрос и записываем ответ в response

\_httpClient.DefaultRequestHeaders.Authorization =

new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", \_token);

HttpResponseMessage response = await \_httpClient.PutAsync($"http://courseproject4/api/profile/{\_user.Id}", jsonContent);

if (response.StatusCode == System.Net.HttpStatusCode.OK)

{

// Обновляем локальную копию данных пользователя

\_user.Surname = surnameLabel.Text;

\_user.Name = nameLabel.Text;

\_user.Patronymic = string.IsNullOrEmpty(patronymicLabel.Text) ? null : patronymicLabel.Text;

if (loginLabel.Text != \_user.Login)

{

\_user.Login = loginLabel.Text;

}

Preferences.Set("UserSurname", \_user.Surname);

Preferences.Set("UserName", \_user.Name);

Preferences.Set("UserPatronymic", \_user.Patronymic ?? "");

Preferences.Set("UserLogin", \_user.Login);

// Обновляем UI

surnameLabel.Text = \_user.Surname;

nameLabel.Text = \_user.Name;

patronymicLabel.Text = \_user.Patronymic ?? "";

loginLabel.Text = \_user.Login;

await DisplayAlert("Успех", "Профиль успешно обновлён!", "Вернуться на главную");

await Navigation.PushAsync(new Views.Home(\_user, \_token));

Navigation.RemovePage(this); // Убираем текущую страницу из стека

}

else if (response.StatusCode == System.Net.HttpStatusCode.Unauthorized)

{

// Ошибка авторизации

await DisplayAlert("Ошибка", "Ваш токен не актуален. Авторизуйтесь заново!", "ОК");

await Navigation.PushAsync(new Views.Auth.Login());

}

else if (response.StatusCode == System.Net.HttpStatusCode.UnprocessableEntity)

{

var errorContent = await response.Content.ReadAsStringAsync();

try

{

// Парсим JSON как JsonDocument

using var document = JsonDocument.Parse(errorContent);

var root = document.RootElement;

var message = root.GetProperty("message").GetString();

if (root.TryGetProperty("errors", out var errors))

{

var errorMessages = new List<string>();

// Проходим по всем ошибкам

foreach (var error in errors.EnumerateObject())

{

foreach (var msg in error.Value.EnumerateArray())

{

errorMessages.Add(System.Text.RegularExpressions.Regex.Unescape(msg.GetString()));

}

}

var combinedErrors = string.Join("\n", errorMessages);

await DisplayAlert("Ошибка валидации", combinedErrors, "ОК");

}

else

{

await DisplayAlert("Ошибка", message ?? "Произошла ошибка валидации.", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Не удалось обработать ответ: {ex.Message}\n\nОтвет: {errorContent}", "ОК");

}

}

else

{

var responseContent = await response.Content.ReadAsStringAsync();

await DisplayAlert("Ошибка", $"Не удалось обновить профиль: {responseContent}", "OK");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Произошла ошибка: {ex.Message}", "OK");

}

finally

{

// Восстанавливаем интерфейс, если авторизация не удалась

MainContent.IsVisible = true;

LoadingIndicator.IsRunning = false;

LoadingIndicator.IsVisible = false;

}

}

private void OnCheckPasswordChanged(object sender, CheckedChangedEventArgs e)

{

bool isChecked = e.Value;

// Если галочка снята, показываем поля для ввода пароля

password.IsVisible = !isChecked;

confirmPassword.IsVisible = !isChecked;

}

}

***Страница профиля***

На листинге 73 представлена разметка XAML для страницы профиля Profile в приложении AdminSamokat. Интерфейс включает заголовок, отображение информации о пользователе (фамилия, имя, отчество, логин) и кнопку редактирования профиля. Данные представлены в виде стилизованного фрейма с тенями, а для удобства пользователей предусмотрен разделитель между информацией и кнопкой редактирования.

***Листинг 73. Страница профиля Profile.xaml***

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="AdminSamokat.Views.Profile"

Shell.NavBarIsVisible="False"

BackgroundColor="#F9F9F9">

<ScrollView>

<Grid Padding="10">

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<!-- Заголовок -->

<Label Grid.Row="0" Text="Ваш профиль"

FontSize="24"

FontAttributes="Bold"

TextColor="#333333"

HorizontalOptions="Start"

Margin="0,10,0,10" />

<!-- Информация о пользователе -->

<Frame Grid.Row="1" Padding="10" CornerRadius="15" BackgroundColor="White" Margin="0,10">

<Frame.Shadow>

<Shadow Brush="#E0E0E0" Opacity="0.5" Offset="3,3" />

</Frame.Shadow>

<StackLayout Spacing="10">

<!-- Данные пользователя с горизонтальным выравниванием -->

<StackLayout Orientation="Horizontal" Spacing="15">

<Label Text="Фамилия:" FontAttributes="Bold" TextColor="#555555" WidthRequest="90" />

<Label x:Name="surnameLabel" FontSize="16" TextColor="#333333" VerticalOptions="Center" />

</StackLayout>

<StackLayout Orientation="Horizontal" Spacing="15">

<Label Text="Имя:" FontAttributes="Bold" TextColor="#555555" WidthRequest="90" />

<Label x:Name="nameLabel" FontSize="16" TextColor="#333333" VerticalOptions="Center" />

</StackLayout>

<StackLayout Orientation="Horizontal" Spacing="15">

<Label Text="Отчество:" FontAttributes="Bold" TextColor="#555555" WidthRequest="90" />

<Label x:Name="patronymicLabel" FontSize="16" TextColor="#333333" VerticalOptions="Center" />

</StackLayout>

<StackLayout Orientation="Horizontal" Spacing="15">

<Label Text="Логин:" FontAttributes="Bold" TextColor="#555555" WidthRequest="90" />

<Label x:Name="loginLabel" FontSize="16" TextColor="#333333" VerticalOptions="Center" />

</StackLayout>

<!-- Разделитель -->

<BoxView HeightRequest="1" BackgroundColor="#E0E0E0" Margin="0,10" />

<!-- Кнопка редактирования -->

<Button Text="Редактировать профиль"

Clicked="OnEditProfileButtonClicked"

FontAttributes="Bold"

TextColor="White"

BackgroundColor="#6200EE"

CornerRadius="10"

Padding="10" />

</StackLayout>

</Frame>

</Grid>

</ScrollView>

</ContentPage>

На листинге 74 представлен код C# для Profile.xaml.cs, который отвечает за отображение данных пользователя и обработку нажатия кнопки редактирования профиля. При создании страницы передается объект пользователя и токен аутентификации, после чего данные заполняются в соответствующие элементы интерфейса. В случае отсутствия отчества, вместо него отображается сообщение «Отчество отсутствует». При нажатии на кнопку "Редактировать профиль" выполняется переход на страницу редактирования EditProfile.

***Листинг 74. C# код страницы Profile.xaml***

public partial class Profile : ContentPage

{

private User \_user;

private string \_token;

public Profile(User user, string token)

{

InitializeComponent();

\_user = user;

\_token = token;

surnameLabel.Text = user.Surname;

nameLabel.Text = user.Name;

if (string.IsNullOrEmpty(user.Patronymic))

{

patronymicLabel.Text = "Отчество отсутствует";

}

else

{

patronymicLabel.Text = user.Patronymic;

}

loginLabel.Text = user.Login;

}

private async void OnEditProfileButtonClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new EditProfile(\_user, \_token));

}

}

# ЗАКЛЮЧЕНИЕ

В рамках проекта по разработке сервиса управления сменами «Самокат Про», предназначенного для синхронизации и оптимизации работы курьеров, были успешно достигнуты поставленные цели и задачи. Разработанное решение, построенное на базе C# MAUI для мобильного приложения и Laravel для серверной части, позволяет эффективно управлять доступностью. «Самокат Про» предоставляет курьерам удобный инструмент для выставления доступности и управления своими сменами.

Разработка сервиса «Самокат Про» продемонстрировала эффективность использования C# MAUI и Laravel для создания масштабируемого и удобного решения для управления сменами курьеров. MAUI позволила создать кроссплатформенное мобильное приложение с единой кодовой базой, значительно сократив время разработки и упростив поддержку. Laravel, с его мощными возможностями и удобством разработки API, обеспечил надежную и масштабируемую серверную часть.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

***Используемые интернет-источники:***

* Руководство по C#.

Режим доступа: <https://metanit.com/sharp/>открытый.

* Руководство по PHP.

Режим доступа: <https://metanit.com/php/>открытый.

* Руководство по.NET MAUI.

Режим доступа: <https://metanit.com/sharp/maui/> открытый.

* Руководство по Laravel.

Режим доступа: <https://laravel.com/> открытый.

* Руководство по MySQL.

Режим доступа: <https://metanit.com/sql/mysql/> открытый.

* Список кодов состояния HTTP.

Режим доступа: <https://en.wikipedia.org/wiki/List_of_HTTP_status_codes> открытый.