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](#_Toc190266669)

[ТЕХНОЛОГИИ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ 6](#_Toc190266670)

[Используемые языка программирования и фреймворки 6](#_Toc190266671)

[Используемое программное обеспечение 8](#_Toc190266672)

[ГЛАВА 1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ 9](#_Toc190266673)

[ИНФОРМАЦИОННОГО ПРОДУКТА. 9](#_Toc190266674)

[Описание проекта и задач 9](#_Toc190266675)

[Специфические требования к API 11](#_Toc190266676)

[Авторизация 11](#_Toc190266677)

[Выход 12](#_Toc190266678)

[Регистрация 13](#_Toc190266679)

[CRUD-операции по штрафу 14](#_Toc190266680)

[CRUD-операции по бонусу 19](#_Toc190266681)

[CRUD-операции по статусу 25](#_Toc190266682)

[CRUD-операции по профилю 30](#_Toc190266683)

[CRUD-операции по доступности 35](#_Toc190266684)

[Подтверждение/отмена доступности 37](#_Toc190266685)

[ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БАЗЫ ДАННЫХ 43](#_Toc190266686)

[Нормализация данных 43](#_Toc190266687)

[Концептуальная модель данных 44](#_Toc190266688)

[Логическая модель данных 45](#_Toc190266689)

[Физическая модель данных 46](#_Toc190266690)

[Диаграмма базы данных 47](#_Toc190266691)

[ГЛАВА 3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ФУНКЦИОНАЛА 48](#_Toc190266692)

[ИНФОРМАЦИОННОГО ПРОДУКТА 48](#_Toc190266693)

[Установка и настройка фреймворка Laravel11 48](#_Toc190266694)

[Создание миграций 49](#_Toc190266695)

[Создание исключений 51](#_Toc190266696)

[Создание моделей 52](#_Toc190266697)

[Создание контроллеров 55](#_Toc190266698)

[Валидация запроса формы 65](#_Toc190266699)

[Маршрутизация 72](#_Toc190266700)

[ГЛАВА 4. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ НА MAUI 73](#_Toc190266701)

# 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:

1. Выход.
2. Просмотр и редактирование своего профиля.
3. 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://courseproject4/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://courseproject4/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://courseproject4/api/register |
| Заголовки |
| Authorization: application/json  Accept: application/json |
| Тело запроса |
| {  name: Дмитрий  surname: Евсеев  patronymic: Евгеньевич  login: evseev  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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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://courseproject4/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***

Окно должно 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).

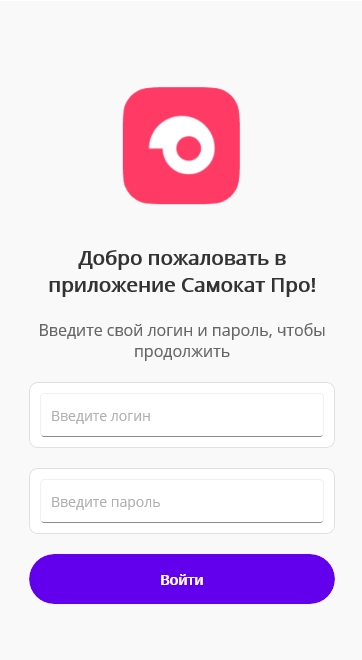


Рисунок 1 – Страница авторизации

**Главная**

Окно должно 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.

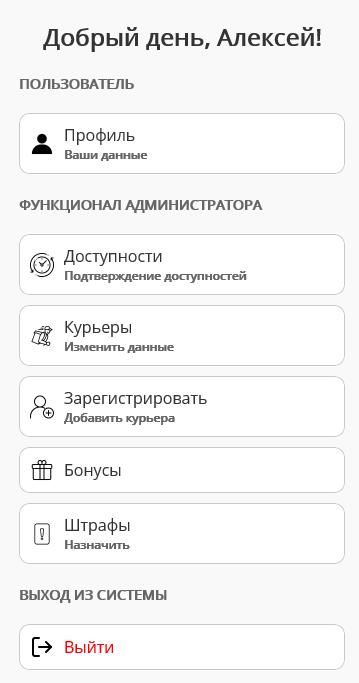


Рисунок 2 – Главная страница

***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.

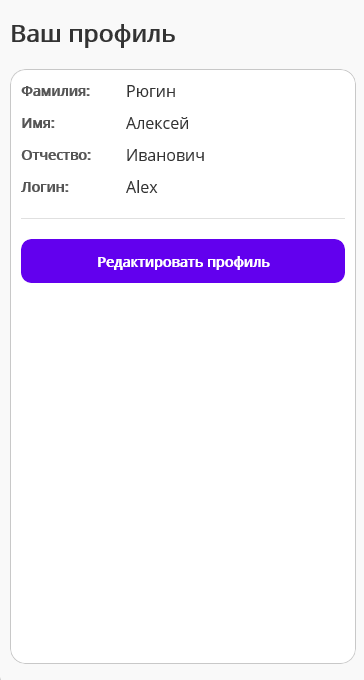


Рисунок 3 – Страница профиля

***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.

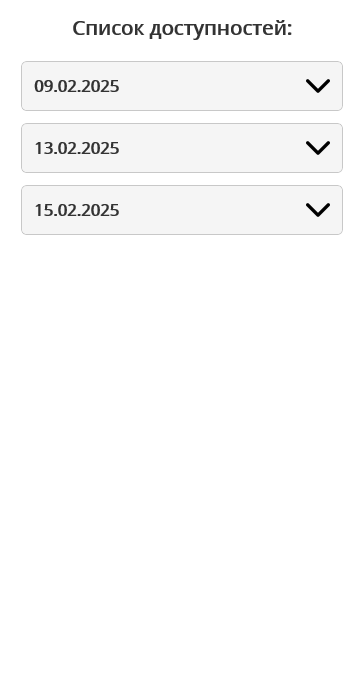


Рисунок 4 – Страница доступностей

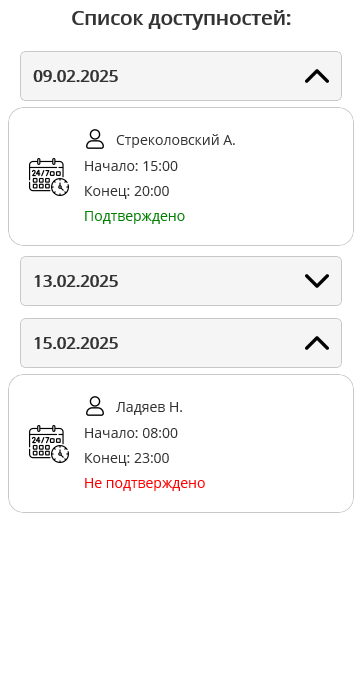


Рисунок 5 – Страница доступностей, показывающая доступности

***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, пользователя должен осуществляться переход на страницу с полной информацией о курьере.

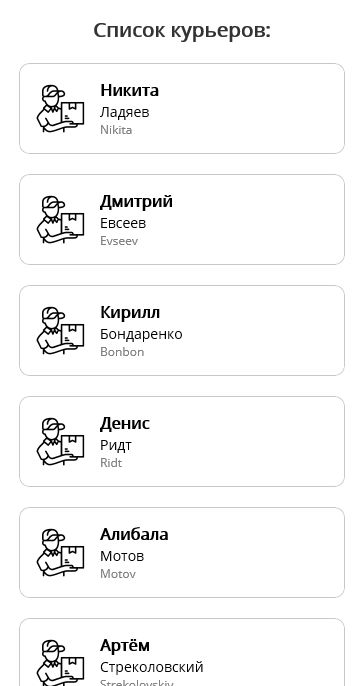


Рисунок 6 – Страница курьеров

**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.

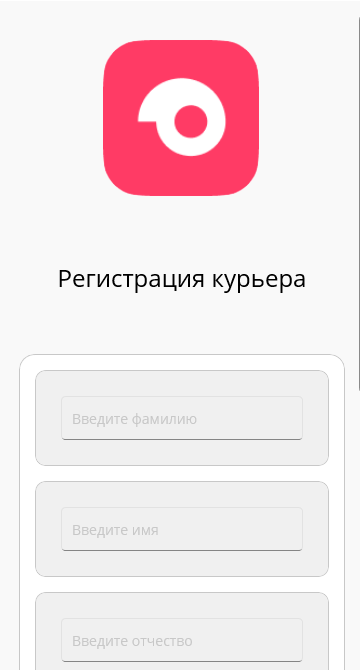


Рисунок 7 – Страница регистрации

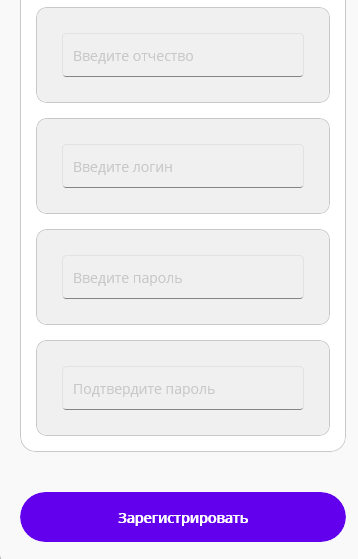


Рисунок 8 – Страница регистрации (2)

***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.

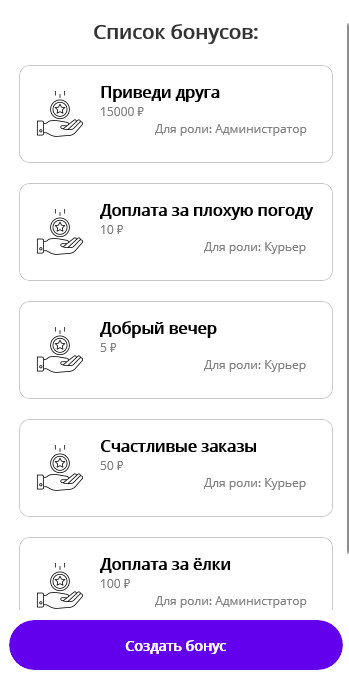


Рисунок 9 – Страница бонусов

***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.

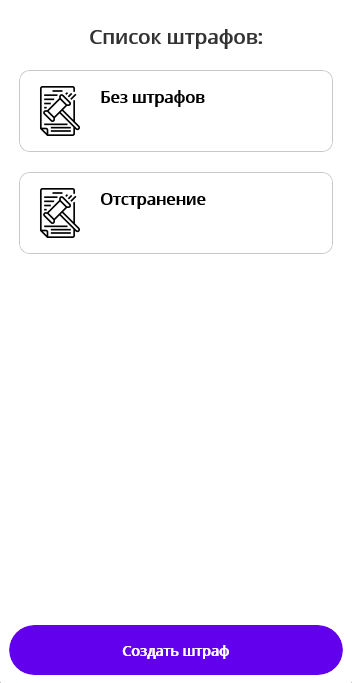


Рисунок 10 – Страница штрафов

**Редактирование профиля**

На этой странице 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.

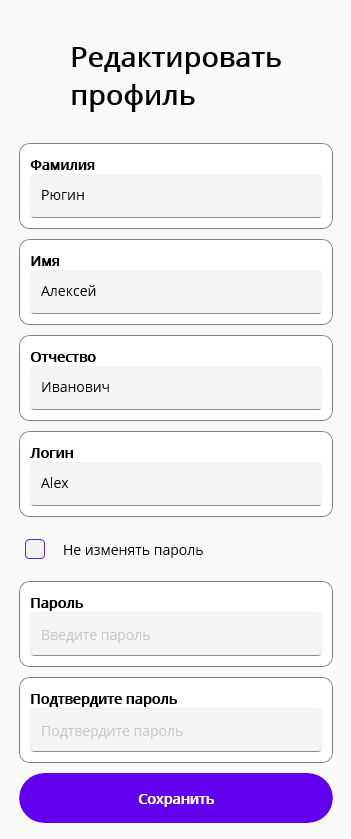


Рисунок 10 – Страница редактирования профиля

**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 11) 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 12). 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.

# 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 1):

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».

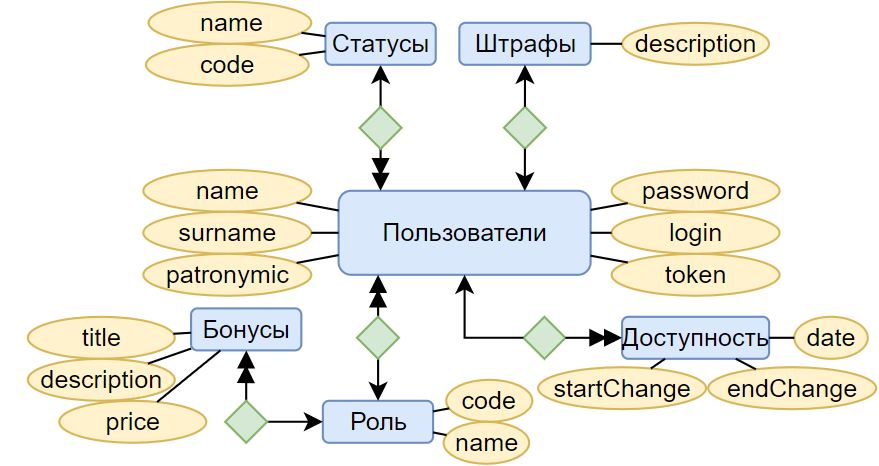


Рисунок - Концептуальная модель данных

## **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 2.

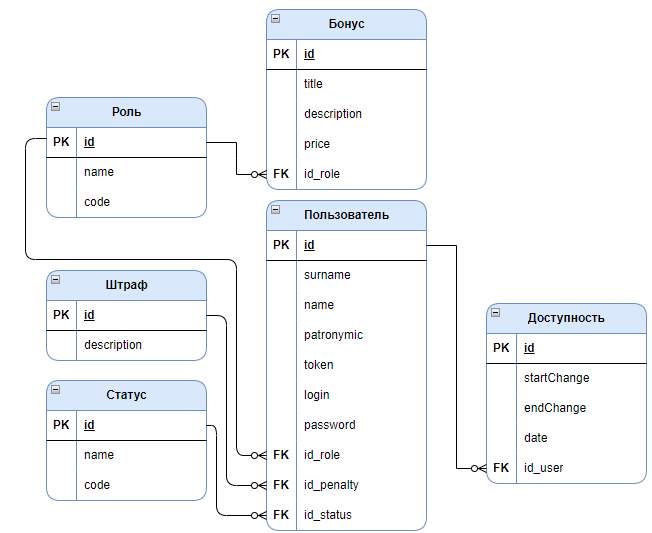


Рисунок - Логическая модель данных

## **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 3 представленная физическая модель данных.

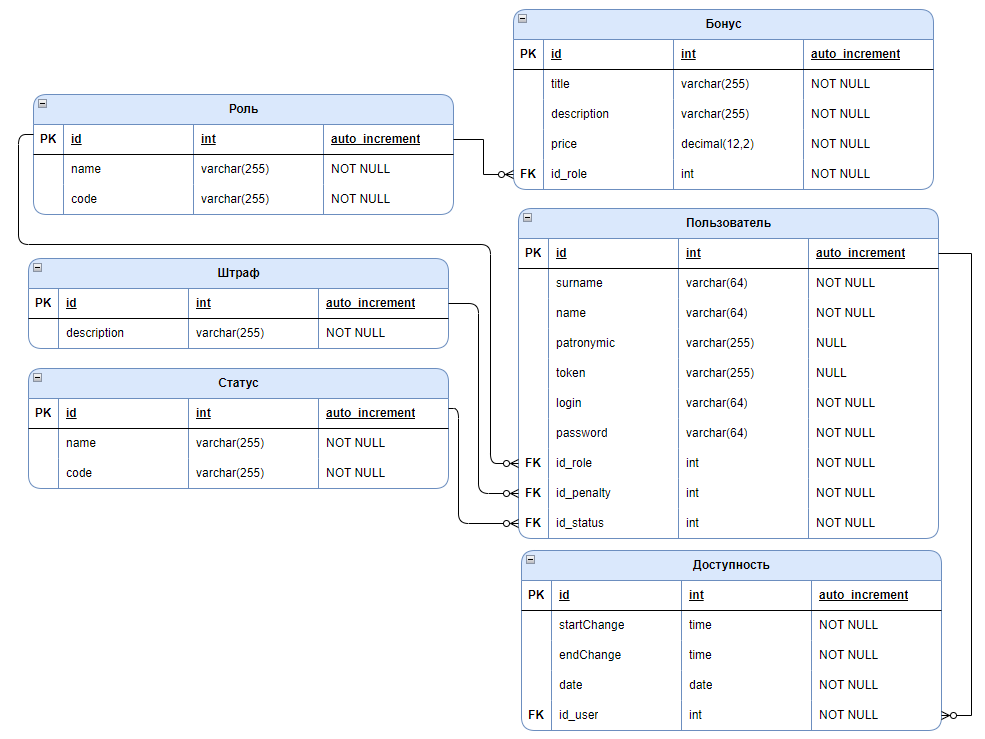


Рисунок - Физическая модель данных

## **Диаграмма базы данных**

На рисунке 4 представлена итоговая диаграмма базы данных.

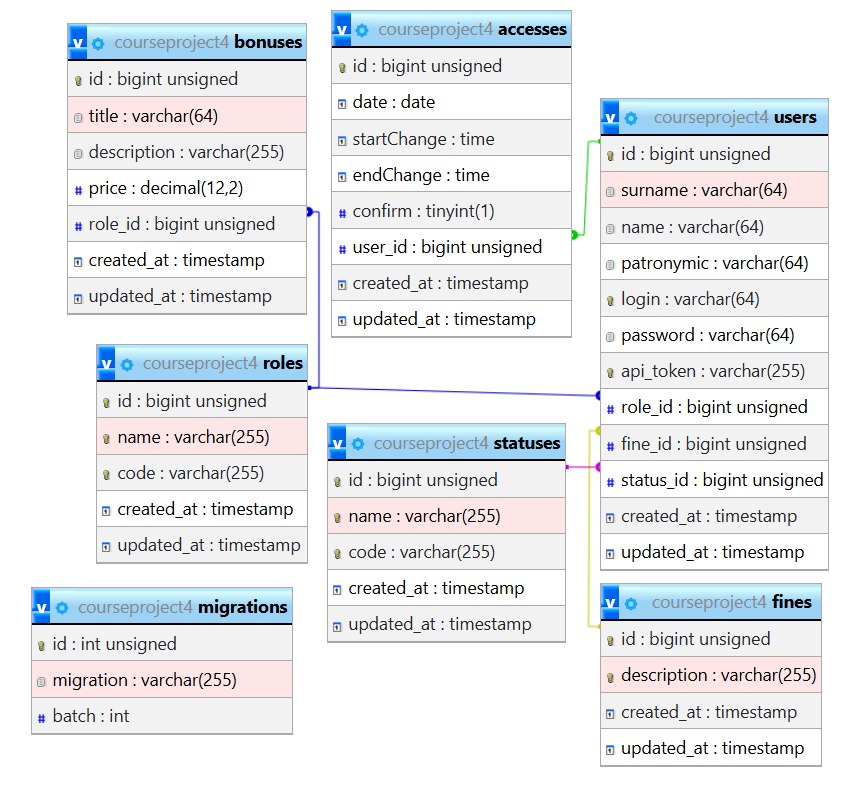


Рисунок - Диаграмма базы данных

# 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. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ НА MAUI