МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Введение в нереляционные базы данных»

Тема: Реализация сервиса билетов и контроллеров общественного транспорта.

Студент гр. 2384 _____ Исмаилов М.В.

Студент гр. 2384 _____ Гребенников Д.А.

Студент гр. 2384 _____ Березин Д.А.

Студент гр. 2384 _____ Гизатуллин А.Р.

Студент гр. 2300 _____ Быков Н.А.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2025

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Исмаилов М.В.

Студент Гребенников Д.А.

Студент Березин Д.А.

Студент Гизатуллин А.Р.

Студент Быков Н.А.

Тема работы: Реализация сервиса билетов и контроллеров общественного транспорта.

Исходные данные:

Задача – сделать информационную систему для оплаты общественного транспорта в Санкт-Петербурге. Нужны варианты как для пассажиров (кто, когда и за что платил, чем платил, балансы, счета), так и для контроллеров (у кого, когда смена, кого и когда поймал какой выписал штраф, какие штрафы оплачены и какие нет).

Используемая база данных – Neo4j.

Содержание пояснительной записки:

«Введение», «Сценарии использования», «Модель данных», «Разработанное приложение», «Выводы», «Приложения», «Литература»

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 05.02.2025

Дата сдачи реферата: 1.06.2025

Дата защиты реферата: 1.06.2025

Студент гр. 2384	_____	Исмаилов М.В.
Студент гр. 2384	_____	Гребенников Д.А.
Студент гр. 2384	_____	Березин Д.А.
Студент гр. 2384	_____	Гизатуллин А.Р.
Студент гр. 2300	_____	Быков Н.А.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках курса «Введение в нереляционные базы данных» в команде был разработан сервис «Система билетов и контроллеров общественного транспорта». Основные свойства проекта: единый и интуитивно понятный интерфейс для всех типов пользователей, а также высокая производительность приложения. В качестве базы данных в приложении используется графовая база данных Neo4j.

SUMMARY

As part of the course "Introduction to Non-relational Databases", the team developed the service "Public Transport Ticket and Controller System". The main properties of the project are: a single and intuitive interface for all types of users, as well as high application performance. The application uses the Neo4j graph database as a database.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – создание высокопроизводительного сервиса для управления билетами и контролёрами общественного транспорта с единым и очевидным интерфейсом для всех типов пользователей.

Разработан сайт, на котором контролёры смогут отмечать проверки билетов, а пассажиры смогут просматривать информацию о своих поездках и билетах. Пользователи будут иметь возможность оставлять отзывы о контролёрах, а также комментировать инциденты, связанные с проверками.

1. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

1.1. Макет UI

Перед началом создания самого приложения, требуется определиться, как будет выглядеть интерфейс приложения. Главное, чтобы он был интуитивно понятен и не различался для разных типов пользователей. В результате был разработан макет пользовательского интерфейса (см. рис. 1.1-1.11).

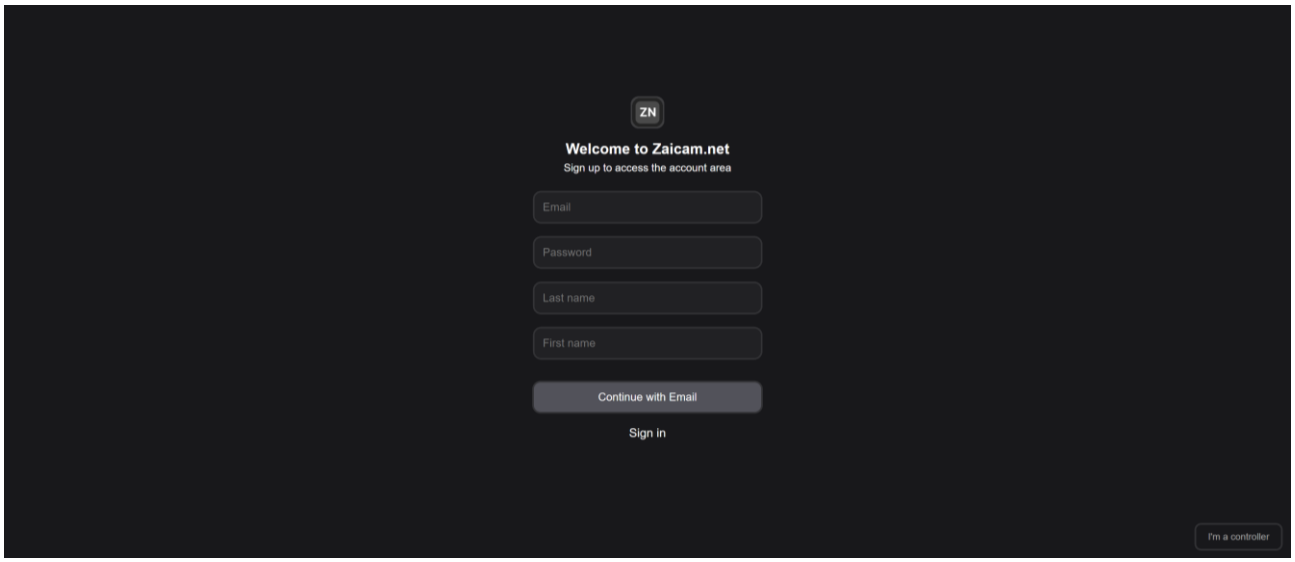

Рисунок 1.1 – Страница регистрации нового пользователя

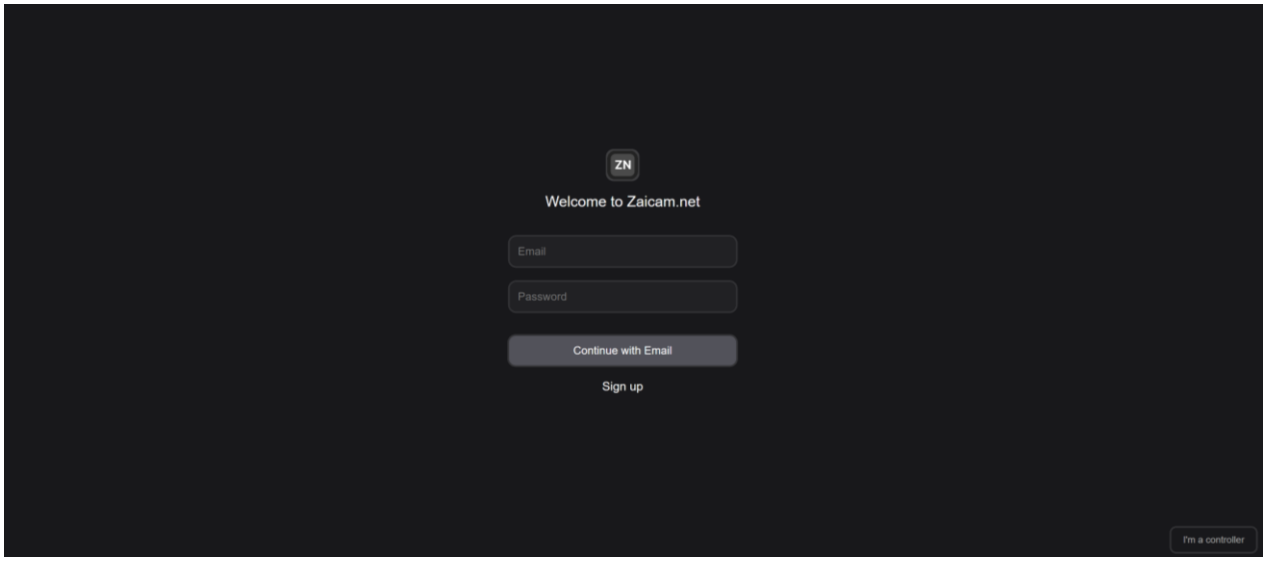

Рисунок 1.2 – Страница авторизации пользователя

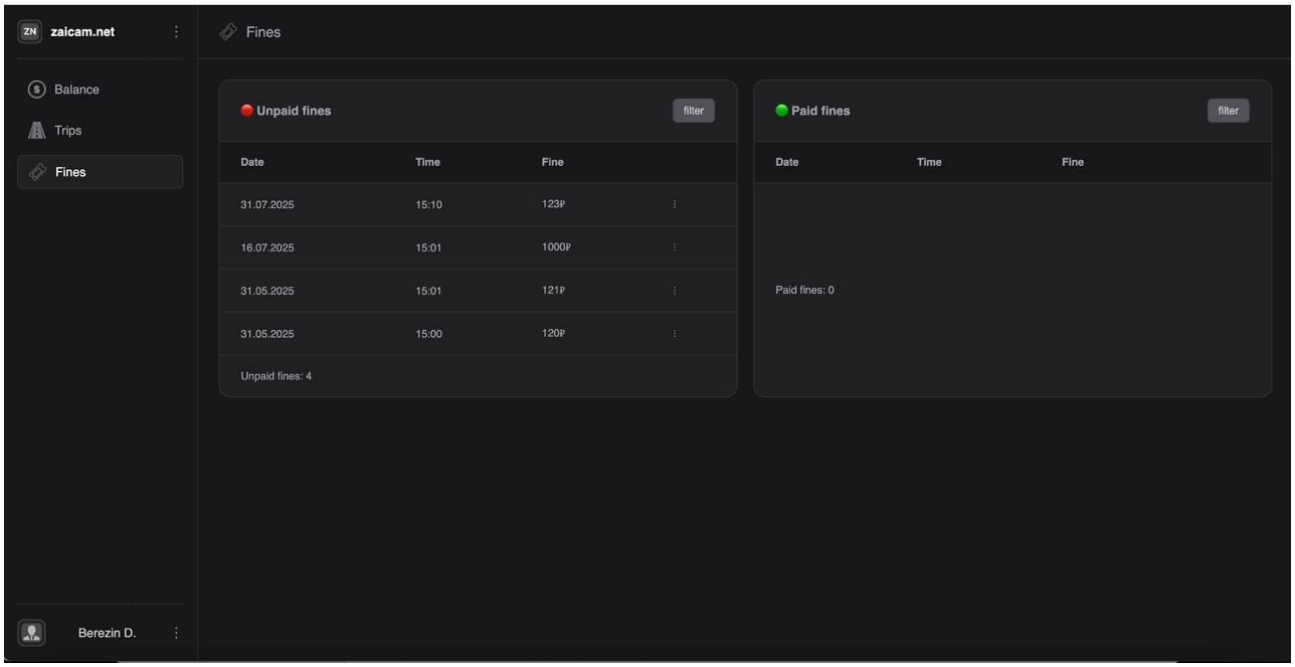

Рисунок 1.3 – Страница со штрафами

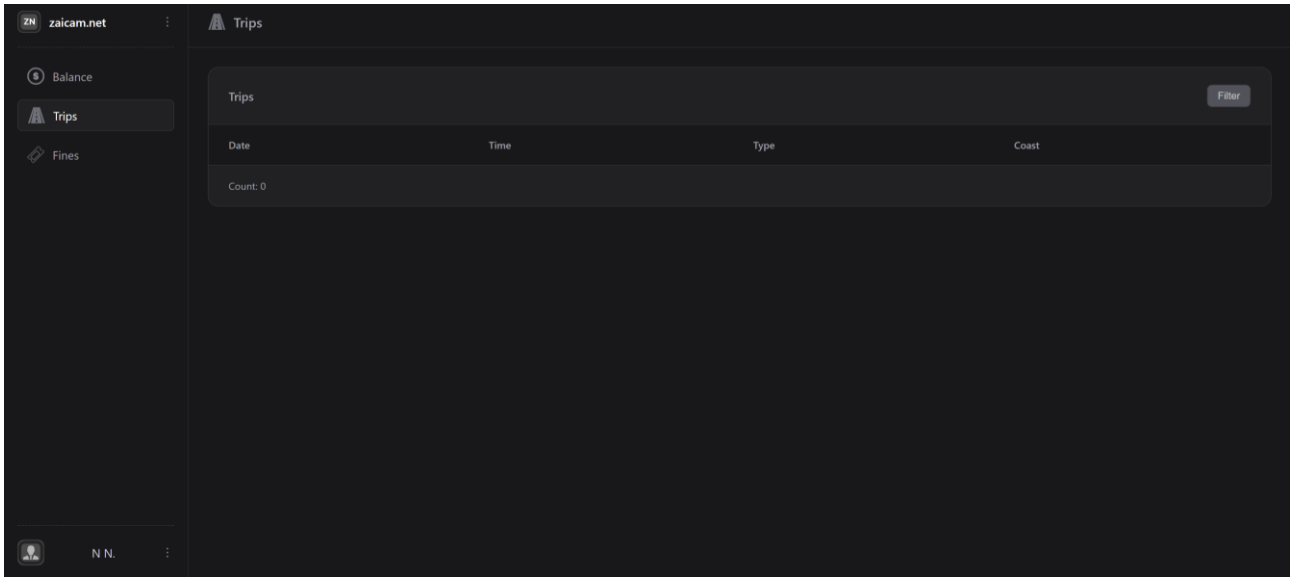

Рисунок 1.4 – Страница с поездками

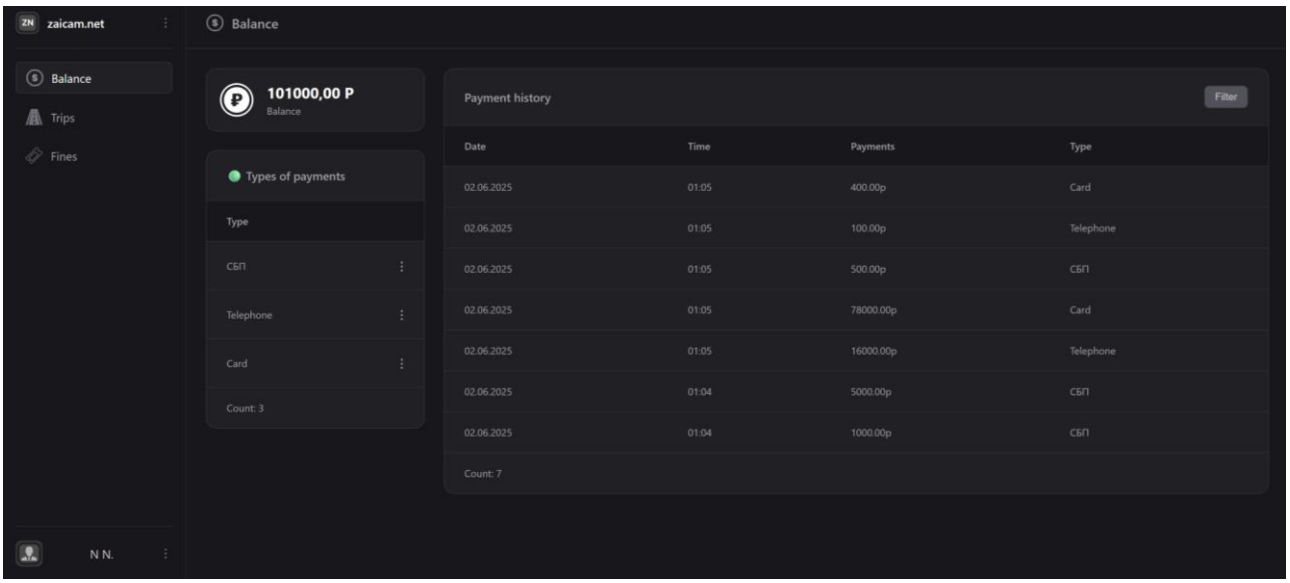

Рисунок 1.5 – Страница с балансом

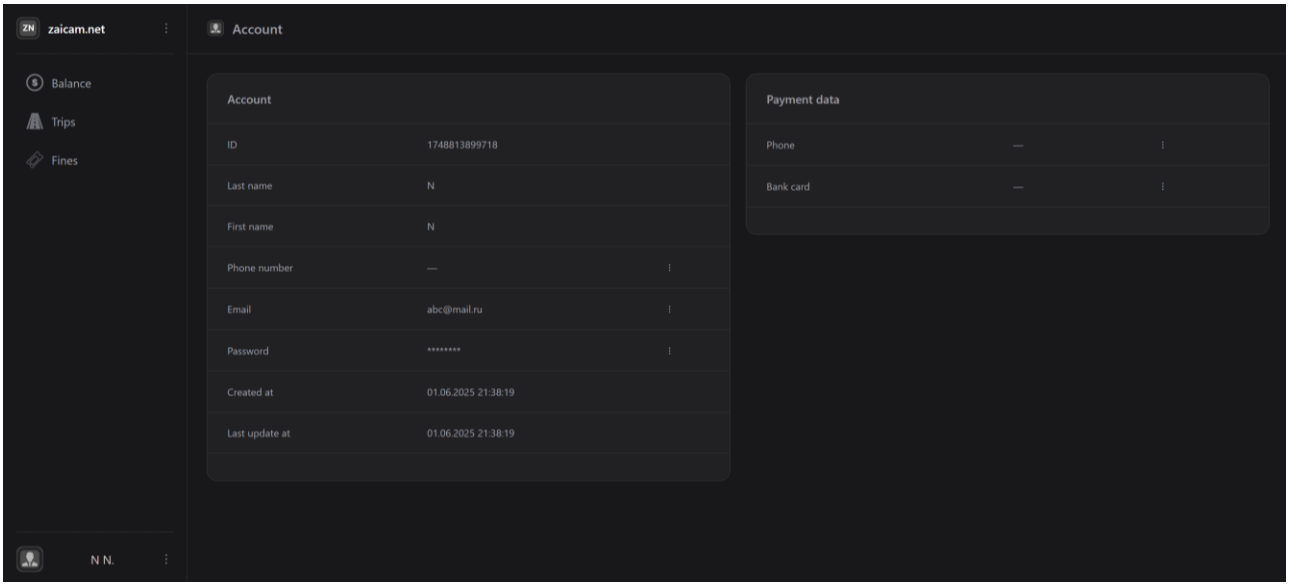

Рисунок 1.6 – Личный кабинет пассажира

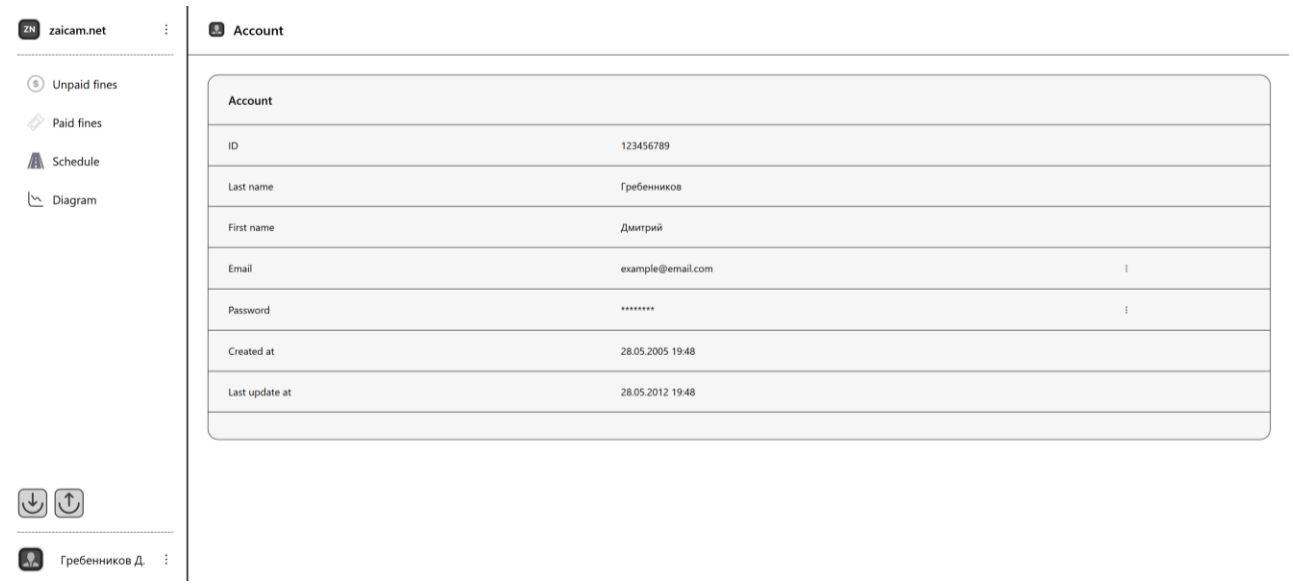

Рисунок 1.7 – Личный кабинет контроллера

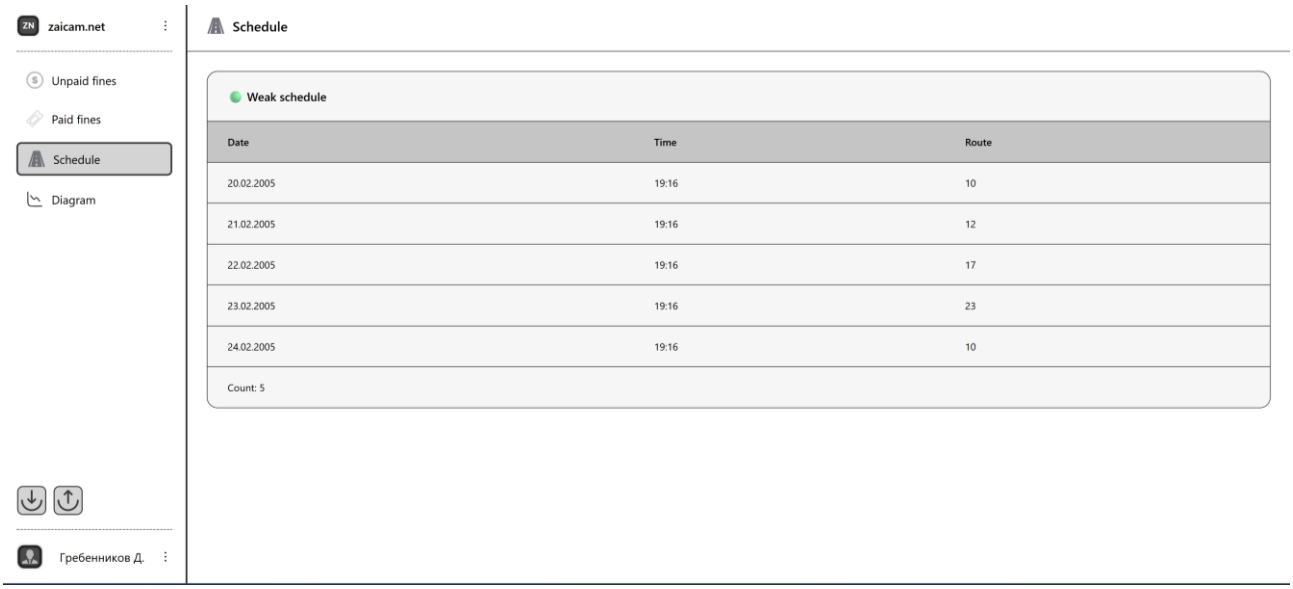

Рисунок 1.8 – Страничка контроллера с расписанием.

Рисунок 1.9 – Страница со штрафами

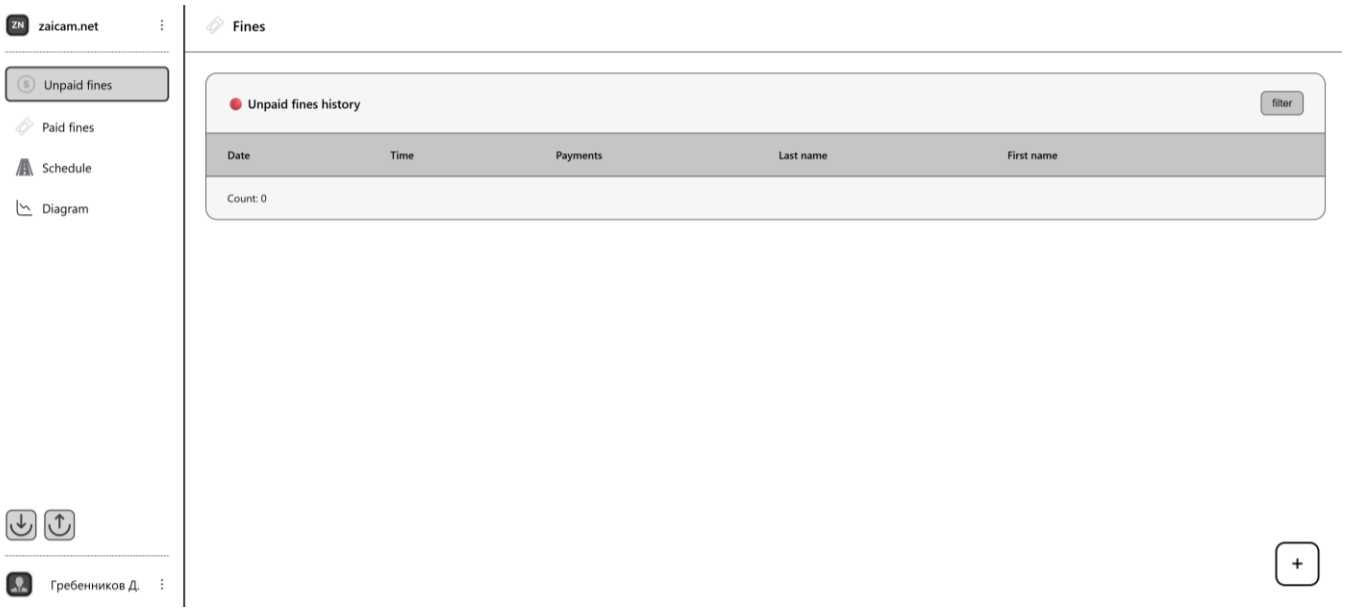

Рисунок 1.10 – Страница с неоплаченными штрафами.

Рисунок 1.11 – Страница статистики для контроллера

1.2. Сценарии использования

В ходе подготовки к разработке приложения были также разработаны сценарии его использования, которые соответствуют интерфейсу приложения. В них учтён каждый тип пользователя.

Сценарий использования: Фильтрация и сортировка данных

Цель: Отфильтровать и отсортировать данные по заданным критериям.
Действующее лицо: Пользователь (пассажир или контролер)

Основной сценарий:

1. Пользователь открывает таблицу с информацией (поездки, штрафы и т.д.).
2. Нажимает на кнопку «Filter», появляется модальное окно фильтрации.
3. В модальном окне отображаются текущие фильтры.
4. Пользователь нажимает «+», чтобы добавить новый фильтр.
5. Выбирает нужный параметр фильтрации.
6. Вводит значения (диапазон чисел, категории и т.п.).

7. Подтверждает, нажав «Confirm».
8. При необходимости добавляет дополнительные фильтры.
9. Закрывает окно кликом вне области.
10. Для сортировки кликает по заголовкам нужных столбцов.

Альтернативные сценарии:

- Пользователь удаляет все фильтры нажатием одной кнопки и возвращается к исходному виду таблицы.

Результат: Информация отфильтрована и отсортирована по нужным критериям.

Сценарий использования: Вход пользователя

Цель: Войти в систему с помощью своего аккаунта.
Действующее лицо: Пользователь (пассажир)

Основной сценарий:

1. Пользователь вводит email.
2. Вводит пароль.
3. Нажимает «Continue with email».

Альтернативные сценарии:

- Переход на страницу регистрации.
- Переход на страницу входа для контролеров.

Результат: Пользователь вошел в систему или перешел на другую нужную страницу.

Сценарий использования: Регистрация пользователя

Цель: Зарегистрировать новый аккаунт.

Действующее лицо: Пользователь

Основной сценарий:

1. Вводит email, пароль, фамилию, имя.
2. Нажимает «Register».

Альтернативные сценарии:

- Переход к авторизации.
- Переход на страницу входа для контролеров.

Результат: Пользователь успешно зарегистрирован.

Сценарий использования: Пополнение баланса

Цель: Узнать баланс и пополнить счет.

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь видит текущий баланс и историю пополнений.
2. Фильтрует историю по датам/суммам.
3. Нажимает кнопку пополнения.
4. В модальном окне вводит сумму.
5. Нажимает «Confirm payment».
6. При отмене — кликает вне модального окна.

Результат: Баланс пополнен, история отображается корректно.

Сценарий использования: Оплата штрафов

Цель: Оплатить штраф и просмотреть историю.

Действующее лицо: Пользователь

Основной сценарий:

1. Видит таблицы оплаченных и неоплаченных штрафов.
2. Фильтрует и сортирует данные.
3. Оплачивает штраф в модальном окне, нажимает «Confirm».

Альтернативный сценарий:

- Отменяет операцию, кликнув вне окна.

Результат: Штраф оплачен, данные обновлены.

Сценарий использования: История поездок

Цель: Посмотреть и проанализировать поездки.

Действующее лицо: Пользователь

Основной сценарий:

1. Открывает таблицу с поездками.
2. Применяет фильтры и сортировку.

Результат: Пользователь получил доступ к истории своих поездок.

Сценарий использования: Изменение данных аккаунта

Цель: Обновить личную информацию.

Действующее лицо: Пользователь

Основной сценарий:

1. Переходит на страницу профиля.
2. Нажимает на поле для редактирования.
3. Вводит новые данные в модальном окне и подтверждает.

Альтернативный сценарий:

- Отменяет операцию кликом вне окна.

Результат: Личные данные обновлены.

Сценарий использования: Вход контролера

Цель: Авторизоваться как контролер.

Действующее лицо: Контролер

Основной сценарий:

1. Вводит email и пароль.
2. Нажимает «Continue with email».

Альтернативный сценарий:

- Переход к входу как пользователь.

Результат: Контролер вошел в систему.

Сценарий использования: Выписка штрафа

Цель: Оформить новый штраф.

Действующее лицо: Контролер

Основной сценарий:

1. Нажимает кнопку «Add Fine».
2. Вводит дату, время, сумму, ID пользователя.
3. Подтверждает штраф нажатием «Confirm».

Альтернативный сценарий:

- Отменяет, кликнув вне модального окна.

Результат: Штраф оформлен и сохранён.

Сценарий использования: История штрафов

Цель: Посмотреть список выданных штрафов.

Действующее лицо: Контролер

Основной сценарий:

1. Видит таблицу (оплаченные или неоплаченные штрафы).
2. Применяет фильтры и сортировку.

Результат: Контролер получил нужную информацию.

Сценарий использования: Просмотр расписания

Цель: Узнать график работы.

Действующее лицо: Контролер

Основной сценарий:

1. Переходит к расписанию.
2. Видит таблицу с датами, маршрутами.
3. Применяет фильтры и сортировку.

Результат: Контролер узнал своё расписание.

Сценарий использования: Изменение данных аккаунта контролера

Цель: Обновить данные профиля.

Действующее лицо: Контролер

Основной сценарий:

1. Заходит в профиль.
2. Выбирает поле и вводит новые данные в модальном окне.
3. Подтверждает изменения нажатием «Confirm».

Альтернативный сценарий:

- Отменяет операцию.

Результат: Профиль обновлен.

Сценарий использования: Создание диаграммы

Цель: Построение визуальной статистики по данным.

Действующее лицо: Контролер

Основной сценарий:

1. Выбирает параметры фильтрации:
    - Имя, Фамилия, Email
    - Кол-во штрафов, оплаченных и нет
    - Суммы, поездки, даты и т.д.
2. Настраивает оси графика:
    - X: категориальные (ФИО, Email, Дата)
    - Y: числовые (кол-во штрафов, сумма и пр.)
3. Построенная диаграмма отображается на экране.

Результат: Получен наглядный график на основе выбранных параметров.

1.3. Вывод о преобладающем типе операций

В проекте преобладают операции чтения данных, так как большинство сценариев подразумевают частые запросы на получение информации:

- Фильтрация и сортировка списков (штрафы, поездки, расписания)
- Просмотр истории операций (баланс, оплаты)
- Загрузка профилей пользователей и контролеров
- Формирование статистики и диаграмм

Операции записи выполняются значительно реже и связаны с:

- Регистрацией и редким изменением данных аккаунта
- Единичными действиями (пополнение баланса, оплата штрафа)
- Выпиской новых штрафов (для контролеров)

Причина дисбаланса:

- Интерфейс требует постоянного обновления таблиц/графиков (чтение).
- Изменения данных происходят точечно (запись).

2. МОДЕЛЬ ДАННЫХ

2.1. Нереляционная модель

Графическое представление нереляционной модели см. на рис. 2.1.

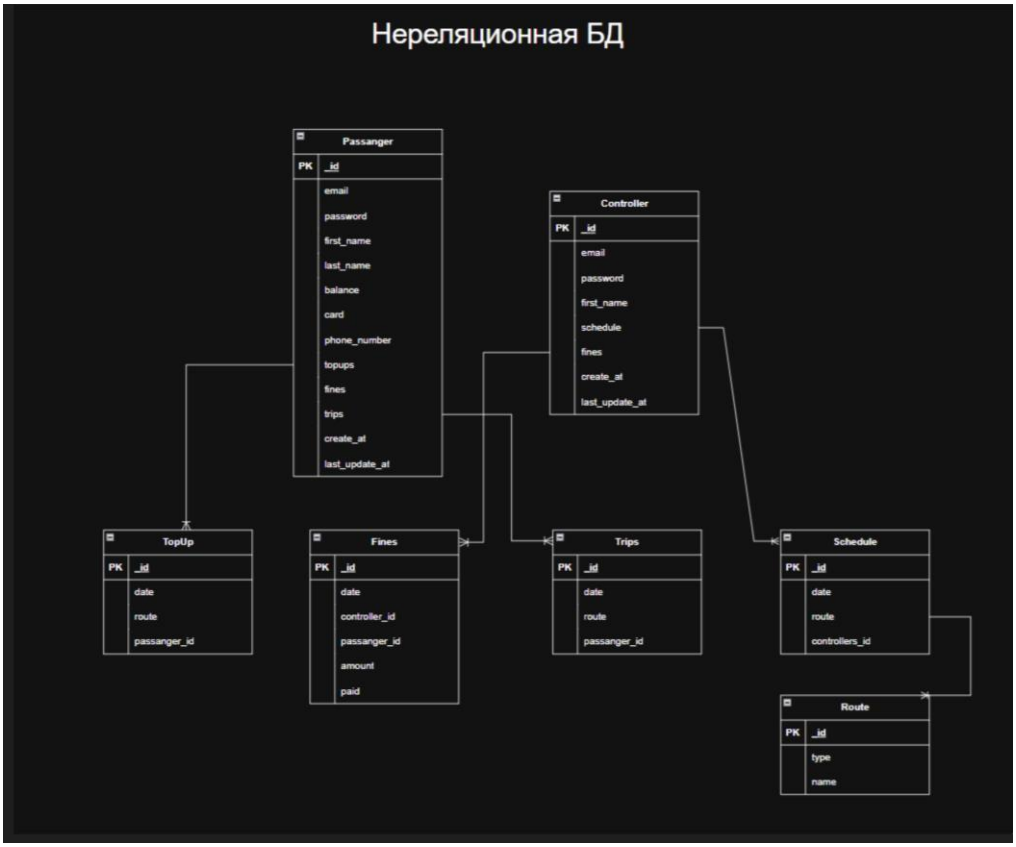

Рисунок 2.1 – Графическое представление нереляционной модели

В данной модели присутствуют следующие сущности с полями и связями:

Passenger – пользователь (Пассажир)

Название	Тип данных	Описание
PK_id	UUID	Уникальный ID пассажира
email	String	Электронная почта
password	String	Хеш пароля
first_name	String	Имя
last_name	String	Фамилия
balance	Float	Баланс счёта
card	String	Данные карты
phone_number	String	Номер телефона

topups	Array Ref	Ссылки на пополнения
lines	Array Ref	Ссылки на маршруты
trips	Array Ref	Ссылки на поездки
create_at	DateTime	Дата создания
last_update_at	DateTime	Дата обновления

Controller– контролер

Название	Тип данных	Описание
PK_id	UUID	Уникальный ID контролера
email	String	Электронная почта
password	String	Хеш пароля
first_name	String	Имя
schedule	String	Фамилия
lines	Float	Баланс счёта
create_at	DateTime	Дата создания
last_update_at	DateTime	Дата обновления

TopUP – поплнения

Название	Тип данных	Описание
PK_id	UUID	Уникальный ID пополнения
date	DateTime	Дата операции
route	Ref	Ссылка на маршрут
Passanger_id	Ref	Ссылка на пассажира

Fines - штрафы

PK_id	UUID	Уникальный ID штрафа
Date	DateTime	Дата выписки
Controller_id	Ref	Кто выписал
Passanger_Id	Ref	Кому выписали
Amount	Float	Сумма штрафа
paid	bool	Оплачен ли

Trips – поездки

PK_id	UUID	Уникальный ID поездки
Date	DateTime	Дата поездки
Route	Ref	Маршрут
Passanger_Id	Ref	Пассажир

Schedule – записи

PK_id	UUID	Уникальный ID записи
Date	DateTime	Дата записи
Route	Ref	Маршрут
Controler_Id	Ref	Контролер

Route – маршрут

PK_id	UUID	Уникальный ID маршрута
Type	String	Тип транспорта
Name	String	Название транспорта

Связи модели со свойствами:

Родительский узел	Связь	Дочерний узел	Описание
Passanger	→ topups	TopUp	Пассажир совершил пополнение
Passanger	→ trips	Trips	Пассажир совершил поездку
Passanger	→ fines	Fines	Пассажиру выписан штраф
Controller	→ fines	Fines	Контролёр выписал штраф

2. Расчёт общего объёма данных

P = Количество пассажиров (базовая единица)

C = Количество контролёров (C = P / 10)

U = Среднее число пополнений на пассажира (U = 5)

F = Среднее число штрафов на пассажира (F = 2)

T = Среднее число поездок на пассажира (T = 20)

S = Среднее число записей расписания на контролёра (S = 30)

R = Количество маршрутов (фиксировано, R = 50)

Формула:

Bclean = P*235+C×174+P*U*56+P*F*73+P*T*56+C*S*56+R*56

Подстановка:

Bclean = 235P+17.4P+280P+146P+1120P+168P+2800=1966.4P+2800

3. Учёт избыточности (Neo4j)

Дополнительные затраты:

Узлы: 32b на каждую сущность

Связи: 32b на каждую связь

Основные связи:

Пассажир → Пополнения (P × U связей)

Пассажир → Поездки (P × T связей)

Пассажир → Штрафы (P × F связей)

Контролёр → Штрафы (C × F связей)

Контролёр → Расписание (C × S связей)

Маршрут → Поездки (~R × 2 связей)

Расчёт:

B actual=B clean +
32*(P+C+P*U+P*F+P*T+C*S+R)+32*(P*U+P*F+P*T+C*F+C*S+R*2)

Упрощение:

B actual = 1966.4P+32*(1.1P+5P+2P+20P+3P)
+2800=1966.4P+995.2P+2800=2961.6P+2800

4. Фактор избыточности

Fct= B clean/B actual= 1966.4P+2800/2961.6P+2800 = 1.5(при больших P)

Вывод:

Объём данных линейно зависит от числа пассажиров (~3 КБ на пассажира).

Избыточность составляет ~1.5x из-за метаданных Neo4j.

Оптимизация:

Сжатие строковых полей (email, имена).

Уменьшение связей (например, хранение trips внутри Passanger как вложенный массив).

Использование индексов для ускорения запросов.

Примеры запросов к модели:

Сценарий использования - "Пассажир"

Сценарий: Просмотр всех поездок

MATCH (p:Passanger {email: "user@example.com"})-[:HAS_TRIP]->(t:Trips)

RETURN t.date, t.route, t.PK_id

ORDER BY t.date DESC

LIMIT 10

Задействованные сущности: Passanger, Trips

Сценарий: Просмотр штрафов

MATCH (p:Passanger {email: "user@example.com"})-[:HAS_FINE]->(f:Fines)

WHERE f.paid = false

RETURN f.date, f.amount, f.PK_id

Задействованные сущности: Passanger, Fines

Сценарий: Пополнение баланса

```
MATCH (p:Passanger {email: "user@example.com"})
CREATE (t:TopUp {
   PK_id: apoc.create.uuid(),
   date: datetime(),
   amount: 500,
   passanger_id: p.PK_id
})
SET p.balance = p.balance + 500
RETURN t
```

Задействованные сущности: Passanger, TopUp

Сценарий использования - "Контролёр"

Сценарий: Выписать штраф

MATCH (c:Controller {email: "controller@example.com"}),

(p:Passanger {email: "user@example.com"})

CREATE (f:Fines {

PK_id: apoc.create.uuid(),

date: datetime(),

amount: 1000,

paid: false,

controller_id: c.PK_id,

passanger_id: p.PK_id

})

RETURN f

Задействованные сущности: Controller, Passanger, Fines

Сценарий: Просмотр расписания

MATCH (c:Controller {email: "controller@example.com"})-[:HAS_SCHEDULE]->(s:Schedule)

RETURN s.date, s.route

ORDER BY s.date

Задействованные сущности: Controller, Schedule

Сценарий: Добавление нового маршрута

CREATE (r:Route {

PK_id: apoc.create.uuid(),

type: "Автобус",

name: "№ 25 (Центр-Вокзал)"

})

RETURN r

Задействованные сущности: Route

Сценарий: Блокировка пользователя

MATCH (p:Passanger {email: "user@example.com"})

SET p.status = "blocked"

RETURN p

Задействованные сущности: Passanger

Сценарий использования - "Регистрация и авторизация"

Сценарий: Регистрация пассажира

```
CREATE (p:Passanger {
   PK_id: apoc.create.uuid(),
   email: "newuser@example.com",
   password: "$2b$10$hashedpassword",
   first_name: "Иван",
   last_name: "Иванов",
   balance: 0,
   phone_number: "+79111234567",
   create_at: datetime(),
   last_update_at: datetime()
})
RETURN p
```

Задействованные сущности: Passanger

Сценарий: Авторизация

```
MATCH (u {email: "user@example.com", password:
"$2b$10$hashedpassword"})
RETURN u
```

Задействованные сущности: Passanger/Controller

Итоговая таблица запросов.

Сценарий	**Количество запросов**	**Задействованные сущности**
Просмотр поездок	1	Passanger, Trips
Просмотр штрафов	1	Passanger, Fines
Пополнение баланса	1	Passanger, TopUp
Выписать штраф	1	Controller, Passanger, Fines
Просмотр расписания	1	Controller, Schedule
Добавление маршрута	1	Route
Блокировка пользователя	1	Passanger
Регистрация пассажира	1	Passanger
Авторизация	1	Passanger/Controller

Все запросы оптимизированы для работы с транспортной системой и используют актуальные сущности из вашей базы данных.

2.2. Реляционная модель

Графическое представление реляционной модели см. на рис. 2.2.

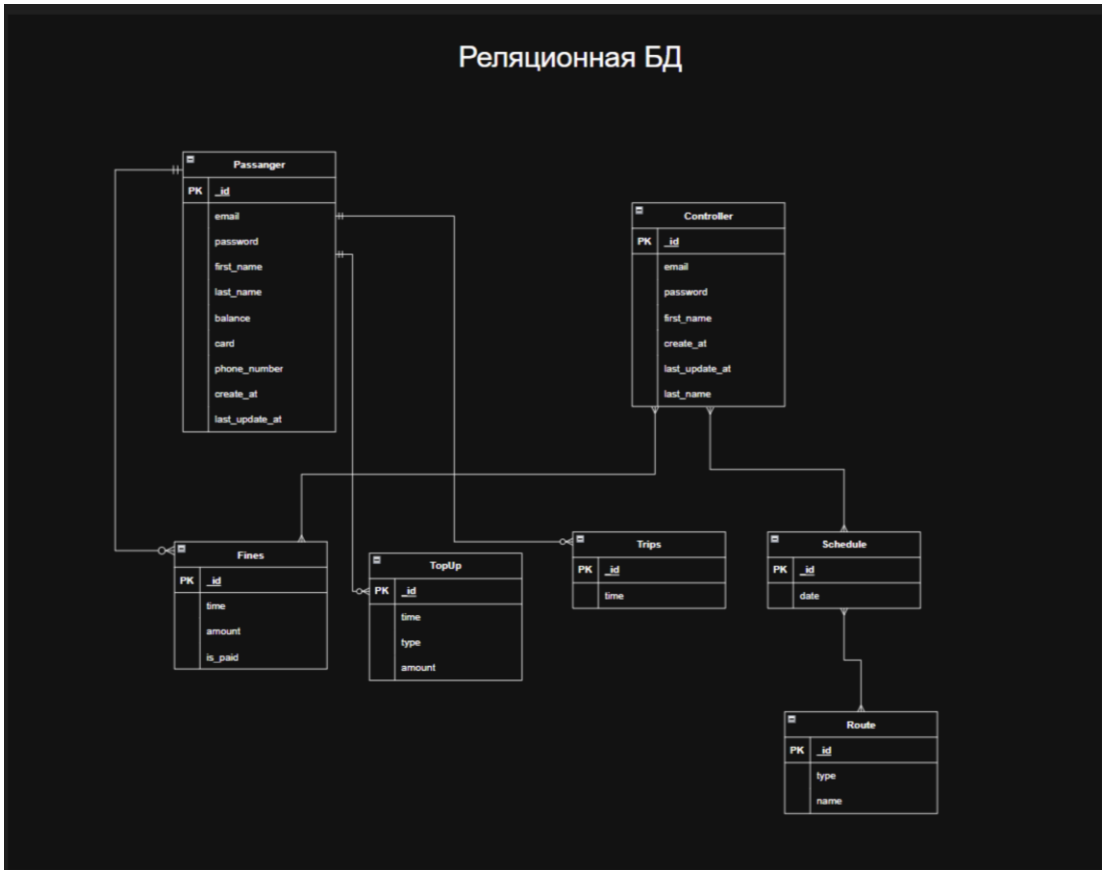

Рисунок 2.2 – Графическое представление реляционной модели

В данной модели присутствуют следующие сущности с полями и связями:

Таблица Passanger – Пассажиры

Название	Тип данных	Описание
Id	UUID	Уникальный идентификатор пассажира
Rmail	VARCHAR	Электронная почта
Password	VARCHAR	Хешированный пароль
First_name	VARCHAR	Имя
Last_name	VARCHAR	Фамилия
Balance	DECIMAL(10,2)	Баланс
Card	VARCHAR	Карта
Phone_number	VARCHAR	Телефон
Create_at	TIMESTAMP	Дата создания аккаунта
Last_update_at	TIMESTAMP	Дата обновления аккаунта

Controller– контролер

Название	Тип данных	Описание
id	UUID	Уникальный ID контролера
email	VARCHAR	Электронная почта
password	VARCHAR	Хеш пароля
first_name	VARCHAR	Имя
last_name	VARCHAR	Фамилия
create_at	TIMESTAMP	Дата создания
last_update_at	TIMESTAMP	Дата обновления

TopUP – поплнения

Название	Тип данных	Описание
id	UUID	Уникальный ID пополнения
Passanger_id	UUID	Ссылка на пассажира
time	TIMESTAMP	Дата и время пополнения

type	VARCHAR	Ссылка на маршрут
amount	DECIMAL(10,2)	Сумма пополнения

Fines - штрафы

PK_id	UUID	Уникальный ID штрафа
Date	TIMESTAMP	Дата выписки
Controller_id	UUID	Кто выписал
Passanger_Id	UUID	Кому выписали
Amount	DECIMAL(10,2)	Сумма штрафа
paid	BOOLEAN	Оплачен ли

Trips – поездки

PK_id	UUID	Уникальный ID поездки
Date	TIMESTAMP	Дата поездки
Route	UUID	Маршрут
Passanger_Id	UUID	Пассажир

Schedule – записи

PK_id	UUID	Уникальный ID записи
Date	TIMESTAMP	Дата записи
Route	UUID	Маршрут
Controler_Id	UUID	Контролер

Route – маршрут

PK_id	UUID	Уникальный ID маршрута
Type	VARCHAR	Тип транспорта
Name	VARCHAR	Название транспорта

Оценка объёма информации и избыточности

Обозначения:

T — количество билетов (единиц поездок).

T // 5 — количество пользователей (в среднем 5 поездок на пользователя).

T // 10 — количество контролёров (в среднем 10 проверок на контролёра).

C — количество контролей (пусть равно T // 2 — проверяется половина билетов).

Средний объём данных:

1. Пользователь (users):

10b login + 10b password + 50b full_name + 1b gender +

100b email + 4b created_at + 4b updated_at = 179b

+ 4b id + 4b status_id = 187b

U' = 187b

2. Билет (tickets):

4b id + 4b user_id + 4b created_at + 4b updated_at + 10b transport_type + 4b price = 30b

T' = 30b

3. Контролёр (controllers):

10b login + 10b password + 50b full_name + 100b email +

4b created_at + 4b updated_at = 178b + 4b id = 182b

Ctrl' = 182b

4. Проверка (checks):

4b id + 4b ticket_id + 4b controller_id + 4b checked_at + 5b status = 21b

Check' = 21b

5. Статус (statuses) — 3 фиксированных, по 15b = 45b

6. Метаданные — 200b на 1 билет

Формула объёма:

B_clean(T) = T * 30b + (T//5) * 179b + (T//10) * 178b + (T//2) * 19b + 45b

= T * (30 + 35.8 + 17.8 + 9.5) + 45

= T * 93.1 + 45

B_actual(T) = T * 30b + (T//5) * 187b + (T//10) * 182b + (T//2) * 21b + 200b * T + 45b

= T * (30 + 37.4 + 18.2 + 10.5 + 200) + 45

= T * 296.1 + 45

Фактор избыточности:

Fct(T) = B_actual(T) / B_clean(T) ≈ 296.1 / 93.1 = 3.18

Примеры запросов к модели:

Сценарий: Пользователь

Просмотр своих билетов

*SELECT * FROM tickets WHERE user_id = 3;*

Кол-во запросов: 1

Таблицы: tickets

Просмотр истории проверок

SELECT c.checked_at, c.status, ctr.full_name AS controller_name
FROM checks c
JOIN controllers ctr ON c.controller_id = ctr.id
JOIN tickets t ON t.id = c.ticket_id
WHERE t.user_id = 3;

Кол-во запросов: 2

Таблицы: checks, tickets, controllers

Сценарий: Контролёр

Проверка билета

INSERT INTO checks (ticket_id, controller_id, checked_at, status)

VALUES (123, 7, NOW(), 'valid');

Кол-во запросов: 1

Таблицы: checks

История всех проверок

SELECT t.id AS ticket_id, u.full_name AS passenger, c.checked_at, c.status

FROM checks c

JOIN tickets t ON t.id = c.ticket_id

JOIN users u ON u.id = t.user_id

WHERE c.controller_id = 7

ORDER BY c.checked_at DESC;

Кол-во запросов: 2

Таблицы: checks, tickets, users

Список всех пользователей

*SELECT * FROM users;*

Кол-во запросов: 1

Таблицы: users

Редактирование профиля пользователя

UPDATE users

SET full_name = 'Иван Сергеевич Петров', email = 'petrov@example.com', updated_at = NOW()

WHERE login = 'ivan_petrov';

Кол-во запросов: 1

Таблицы: users

Добавление контролёра

INSERT INTO controllers (login, password, full_name, email, created_at, updated_at)
VALUES ('ctrl123', 'pass321', 'Сидоров А.А.', 'sid@ctrl.com', NOW(), NOW());
Кол-во запросов: 1
Таблицы: controllers

Экспорт всех билетов
*COPY (SELECT row_to_json(t) FROM (SELECT * FROM tickets) t) TO 'tickets.json';*
Кол-во запросов: 1
Таблицы: tickets

Сценарий: Регистрация и вход
Регистрация пользователя

INSERT INTO users (
login, password, full_name, gender, email, created_at, updated_at, status_id
)
VALUES ('user123', 'mypassword', 'Алексей Котов', 'M', 'alex@example.com', NOW(), NOW(), 1);
Кол-во запросов: 1
Таблицы: users

Авторизация
*SELECT * FROM users WHERE login = 'user123' AND password = 'mypassword';*
Кол-во запросов: 1
Таблицы: users

2.3. Сравнение моделей

Удельный объём информации

NoSQL: объём данных, вычисленный ранее, оценивается по формуле B(N) = 2601·N байт. Однако в документе хранятся избыточные данные, что

увеличивает коэффициент избыточности до Fct(N) = 1.27.
SQL: объём хранимых данных оценивается по формуле B(N) = 2563·N байт. Однако и здесь присутствует избыточная информация (индексы, метаданные), что приводит к коэффициенту избыточности Fct(N) = 1.12.

Запросы по отдельным юзкейсам

NoSQL: количество запросов для реализации сценариев использования меньше за счёт более лаконичной структуры самой базы данных. Она состоит из меньшего количества различных коллекций, отсутствует операция JOIN как таковая.
SQL: для большинства сценариев требуется два и более запросов, поскольку необходимо объединение различных таблиц для получения полной информации об объектах. В результате реляционные базы данных уступают по числу запросов на реализацию типовых операций.

Вывод

NoSQL оптимален для проектов, требующих гибкой схемы данных, быстрого прототипирования и горизонтального масштабирования, что особенно полезно при работе с разнородной и динамически изменяющейся информацией. Несмотря на то, что SQL обеспечивает строгую нормализацию данных, уменьшает избыточность и поддерживает выполнение сложных запросов с высокой степенью целостности, его жёсткая схема ограничивает гибкость разработки. В рассматриваемом случае, где структура данных может меняться, а ключевыми являются взаимодействия между объектами (например, пассажирами и их поездками, штрафами, пополнениями), использование NoSQL предоставляет больше удобства, адаптивности и производительности.

3. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

3.1. Краткое описание

Для взаимодействаия с базой данных были реализованы следующие модули:

В проекте используется драйвер Neo4j для JavaScript:

- Подключение к базе данных осуществляется напрямую через драйвер Neo4j, настроенный на работу с сервером Neo4j.
- Логика работы с базой (создание бэкапов, запись и чтение данных) реализована с помощью функций и методов, которые взаимодействуют с драйвером Neo4j.
- Для резервного копирования данных реализованы функции, которые сохраняют данные в структуру JavaScript (например, объекты) и записывают их в JSON-файлы.
- Восстановление данных из бэкапа происходит через функции, которые читают JSON-файлы или объекты и записывают данные обратно в базу с использованием драйвера.
- Сценарии использования и бизнес-логика реализованы через набор функций, которые оперируют с базой данных через драйвер Neo4j.
- Вспомогательные функции вынесены в отдельный модуль utils.js.

```
const driver = neo4j.driver(
  'bolt://db:7687',
  neo4j.auth.basic('neo4j', 'strongpassword123'),
  { encrypted: 'ENCRYPTION_OFF' }
);
```

Модуль app.js — представляет собой REST API сервер, написанный с использованием Node.js и фреймворка Express. Сервер реализует все сценарии

использования приложения: получает запросы с фронтенда на получение, запись, обновление и удаление данных, обрабатывает их с помощью соответствующих маршрутов и возвращает ответы в зависимости от результата выполнения и корректности переданных данных.

Сам интерфейс приложения реализован с использованием js и html

В результате, в приложении используется три сервиса:

• db — сервис Neo4j

• backend — REST API сервис, реализованный на Node.js с Express

• frontend — сервис фронтенда, реализованный с использованием html + js

Пароль для базы данных задаётся через переменные окружения, прописанные напрямую в docker-compose.

Для запуска приложения необходимо склонировать репозиторий, перейти в папку проекта и развернуть приложение при помощи docker.

3.2. Использованные технологии

Нереляционная база данных: Neo4j

Backend: Node.js, Express

Frontend: Js, html

3.3. Схема экранов приложения

Схему экранов приложения см. на рис. 3.1.

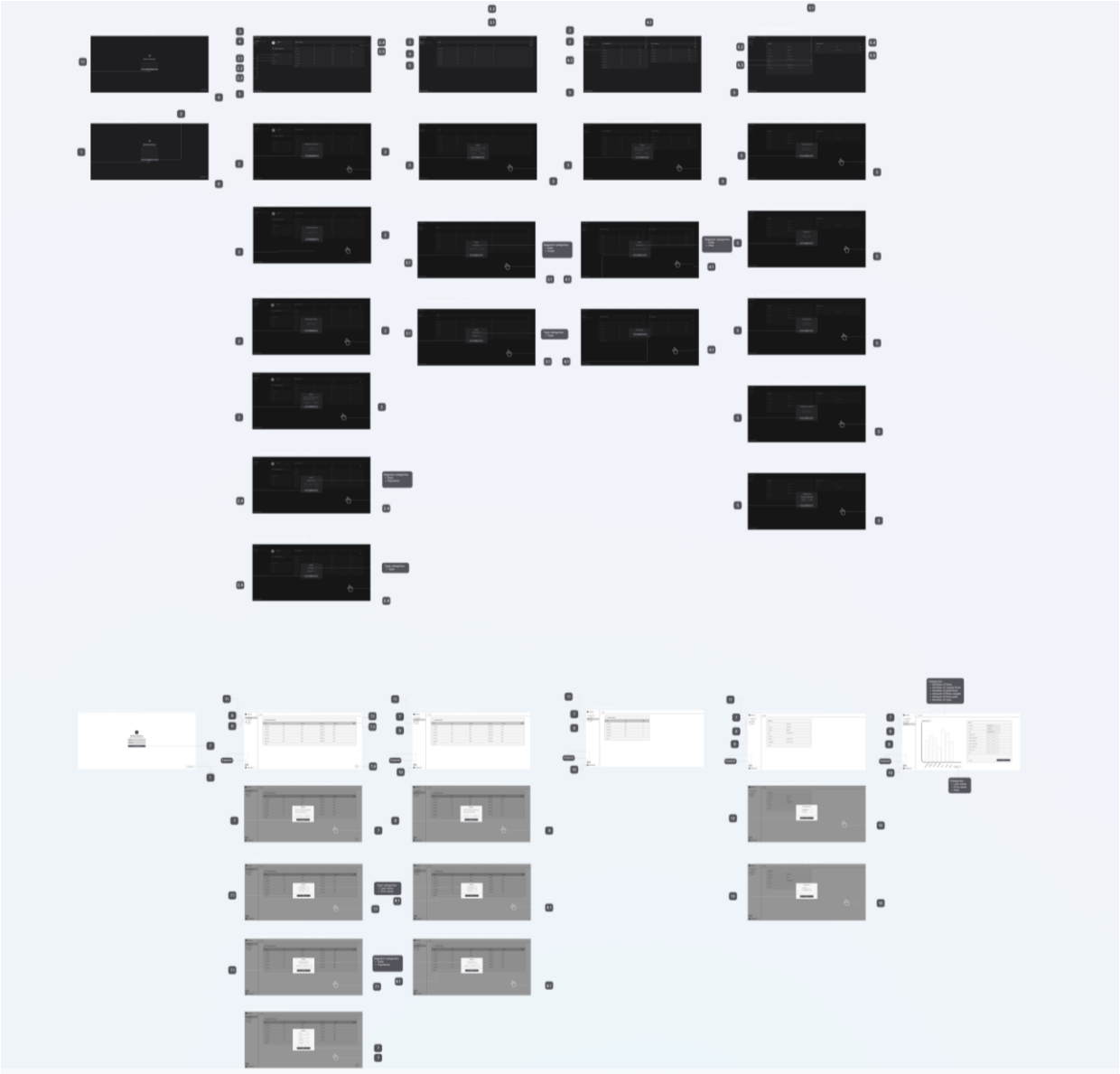

Рисунок 3.1 – Схема экранов приложения

4. ВЫВОДЫ

4.1. Достигнутые результаты

В ходе выполнения работы было разработано приложение-сервис для автоматизации системы билетов и работы контролёров общественного транспорта в Санкт-Петербурге. Сервис позволяет пассажирам просматривать историю поездок, пополнять баланс, оплачивать штрафы, а также управлять своими аккаунтами. Контролёры имеют доступ к расписанию смен, могут оформлять штрафы и просматривать их статус. Администратор системы может управлять пользователями, маршрутам и расписанием, а также осуществлять резервное копирование базы данных и анализировать статистику.

4.2. Недостатки и пути для улучшения полученного решения

На текущем этапе не реализованы функции фильтрации и сортировки пользователей для администратора, а также возможность редактирования профилей пассажиров и контролёров. Кроме того, не предусмотрен интерфейс для анализа статистики поездок и штрафов в разрезе маршрутов и времени.

4.3. Будущее развитие решения

Планируется реализация функций, не включённых в текущую версию, а также развитие кроссплатформенных клиентских приложений для пассажиров и контролёров. В дальнейшем возможна интеграция с внешними платёжными системами и использование технологий машинного обучения для анализа данных и прогнозирования нагрузки на маршруты.

ЗАКЛЮЧЕНИЕ

Реализовано приложение с тремя ролями пользователей: пассажир, контролёр и администратор. Каждая из ролей обладает своим набором прав и доступов. Приложение покрывает ключевые сценарии использования и может применяться уже на текущем этапе разработки. В качестве хранилища данных используется графовая база данных Neo4j, что обеспечивает гибкость и наглядность модели связей между сущностями. Развёртывание выполнено с использованием контейнеризации через Docker, что упрощает переносимость и масштабирование системы.

СПИСОК ЛИТЕРАТУРЫ

1. Ссылка на репозиторий с исходным кодом проекта: https://github.com/moevm/nosql1h25-ticketctrl
2. Документация Neo4j // Neo4j.rb. URL: https://neo4jrb.readthedocs.io/en/stable/
3. Официальная документация Neo4j // Neo4j. URL: neo4j.com/docs

ПРИЛОЖЕНИЕ А
ИНСТРУКЦИЯ ПО РАЗВЁРТЫВАНИЮ

Запуск приложения:

```
docker-compose up -d ; docker-compose logs app
```

Отладочные пользователи:

Controller
Email: *controller003@example.com*
Password: *1234567890*

User
Email: *email*
Password: *password*