**СОДЕРЖАНИЕ**

Введение…………………………………………………………………….7

1.Описание предметной области……………………………………. …....8

1.1 Назначение и область применения программного продукта…………8

1.2. Источники входной информации……………………………………...8

1.3. Выходная информация………………………………………………...9

1.4. Требования к программному продукту……………………………….9

2. Описание маппинга…………………………………………………….11

3. Описание точек доступа………………………………………………..17

3.1 Описание точек доступа для управления маршрутами……………..17

3.1.1 Создание маршрута………………………………………………….17

3.1.2 Получение списка всех заданий…………………………………….17

3.1.3 Получение информации о задании по его идентификатору………17

3.1.4 Обновление информации о задании по его идентификатору……..17

3.1.5 Удаление задания по его идентификатору…………………………18

3.1.6 Получение детальной информации о задании по его идентификатору………………………………………………………………….18

3.2 Описание точек доступа для управления автобусами……………...18

3.2.1 Создание автобуса…………………………………………………..18

3.2.2 Получение списка всех автобусов…………………………………19

3.2.3 Получение информации об автобусе по его идентификатору…...19

3.2.4 Обновление информации об автобусе по его идентификатору….19

3.2.5 Удаление автобуса по его идентификатору……………………….20

3.3 Описание точек доступа для управления водителями……………..20

3.3.1 Создание водителя………………………………………………….20

3.3.2 Получение списка всех водителей………………………………...20

3.3.3 Получение информации о водителе по его идентификатору……20

3.3.4 Обновление информации о водителе по его идентификатору…..21

3.3.5 Удаление водителя по его идентификатору……………………….21

3.3.6 Получение списка автобусов, связанных с водителем……………21

3.4 Описание точек доступа для управления маршрутами…………….21

3.4.1 Создание маршрута……………………………………………........21

3.4.2 Получение списка всех маршрутов…………………………………22

3.4.3 Получение информации о маршруте по его идентификатору……22

3.4.4 Обновление информации о маршруте по его идентификатору…..22

3.4.5 Удаление маршрута по его идентификатору………………………23

4. Разработка REST-сервиса……………………………………………...24

4.1 Структура проекта…………………………………………………….24

4.2 Запуск проекта…………………………………………………………25

* 1. Создание объектов базы данных……………………………………...25

Заключение………………………………………………………………...27

Список использованных источников…..………………………………...28

**ВВЕДЕНИЕ**

Целью данного курсового проекта является разработка REST-сервиса "Автобусный парк" с использованием передовых технологий Node.js (Express) и PostgreSQL. Наша цель - предоставить удобный и эффективный инструмент для управления автобусами, водителями и операциями над ними.

В наше время автобусные парки играют важную роль в обеспечении городского транспорта. Однако, управление автобусами и водителями, а также выполнение различных операций с ними, требует современного подхода. REST-сервис будет обеспечивать гибкое и масштабируемое решение для эффективного управления автобусными ресурсами.

Node.js был выбран для создания серверной части приложения из-за его высокой производительности и широких возможностей. Express.js, фреймворк для Node.js, предоставляет удобные инструменты для создания веб-приложений и API. PostgreSQL, надежная реляционная база данных, была выбрана для хранения данных из-за своей надежности и расширенных возможностей запросов.

Мы ожидаем, что результатом нашего проекта будет полноценный REST-сервис, способный управлять автобусами и водителями, обеспечивая операции добавления, удаления, редактирования и поиска данных.

Разработка REST-сервиса "Автобусный парк" предоставит нам практический опыт работы с передовыми технологиями Node.js и PostgreSQL, а также научит создавать эффективные и масштабируемые веб-сервисы.

**1. Описание предметной области**

1.1 Назначение и область применения программного продукта

Автобусные парки играют важную роль в современной транспортной инфраструктуре, обеспечивая пассажирские перевозки между различными точками города и за его пределами. В наше время автобусные парки стали неотъемлемой частью городской жизни, обеспечивая комфортное и доступное транспортное обслуживание.

В цифровую эпоху автобусные парки стали центральным элементом многих транспортных компаний и государственных организаций, обеспечивая эффективное управление автобусами, контроль маршрутов и планирование графиков движения. Разработка системы управления автобусным парком является важной задачей для обеспечения плавной и безопасной работы общественного транспорта.

Необходимо разработать REST-сервис "Автобусный парк", который будет предоставлять API для управления автобусами, водителями и операциями над ними. Использование современных технологий, таких как Node.js (Express) и PostgreSQL, позволит создать гибкую и масштабируемую систему управления автобусными ресурсами.

1.2. Источники входной информации

Для разработки REST-сервиса автобусного парка были использованы следующие источники входной информации:

1. Справочная литература: Учебники и учебные пособия по транспортной логистике и управлению автобусными парками.
2. Анализ существующих решений и конкурентов: Проведен обзор существующих систем и аналогичных REST-сервисов для выявления особенностей функциональности и лучших практик. Это помогло определить ключевые особенности, которые следует учесть при разработке нашего сервиса.

Эти источники информации были ключевыми для определения функциональных требований и обеспечения соответствия сервиса потребностям пользователей и бизнес-целям проекта.

1.3. Выходная информация

Отчеты и статистика:

* Статистические данные о количестве автобусов в парке.
* Отчеты о выполненных рейсах и перевезенных пассажирах.
* Аналитические отчеты о доходах и расходах автобусного парка.

Информация о рейсах и расписаниях:

* Расписание движения автобусов по маршрутам.
* Информация о времени отправления и прибытия на конечные остановки.

Данные о водителях и автобусах:

* Список водителей с указанием их личных данных.

Информация о каждом автобусе, включая марку, модель, год выпуска и техническое состояние.

Управление транспортным парком:

* Операции по добавлению новых автобусов в парк.
* Возможность удаления автобусов из парка при списании или реализации.
* Возможность изменения информации о водителях и автобусах.

Выходная информация представляет собой результат работы системы управления автобусным парком и предназначена для использования администрацией парка, водителями, а также другими заинтересованными сторонами, такими как пассажиры и городские органы управления транспортом.

1.4. Требования к программному продукту

Управление автобусами:

* Возможность добавления, редактирования и удаления информации о каждом автобусе в парке, включая марку, модель, год выпуска и техническое состояние.

Управление водителями:

* Ведение базы данных водителей с указанием их личных данных, категорий водительских прав.
* Распределение водителей по автобусам и маршрутам, в соответствии с графиком движения.

Требования к программному продукту "Автобусный парк" направлены на создание системы управления, которая будет обеспечивать эффективное и безопасное функционирование автобусного парка, удовлетворяя потребности как администрации, так и пользователей общественного транспорта.

**2. Описание маппинга**

Отображение объектов базы данных на объекты приложения.

Сущности базы данных, такие как "Автобус" и "Водитель", должны быть отображены на соответствующие классы или объекты в приложении. Например, таблица "Автобусы" может быть отображена на класс Bus, а таблица "Водители" - на класс Driver.

Установление связей между сущностями.Определение связей между различными сущностями базы данных, такими как связь "один к одному", "один ко многим" или "многие ко многим". Например, у каждого автобуса может быть только один водитель, что соответствует связи "один к одному".

Определение атрибутов и их типов.Сопоставление атрибутов каждой сущности базы данных с полями или свойствами объектов приложения. Например, атрибуты "марка" и "модель" автобуса должны быть отображены на соответствующие свойства класса Bus.

Управление изменениями и согласованностью данных. Реализация механизмов для автоматического согласования данных между объектами приложения и базой данных при внесении изменений в одном из них. Например, при изменении данных о водителе в приложении должны автоматически обновляться соответствующие записи в таблице "Водители" базы данных.

Использование ORM-библиотеки. В данном проекте использовалась объектно-реляционных отображений (ORM) для упрощения маппинга данных между объектами приложения и базой данных - Sequelize для Node.js.

Ниже представлены модели используемые в проекте:

Модель "Assignment" представляет собой информацию о назначении автобусов на определенные маршруты с определенными водителями.

const { DataTypes } = require('sequelize');

module.exports = (sequelize) => {

const Assignment = sequelize.define('Assignment', {

assignment\_id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true,

},

bus\_id: {

type: DataTypes.INTEGER,

allowNull: false,

},

driver\_id: {

type: DataTypes.INTEGER,

allowNull: false,

},

route\_id: {

type: DataTypes.INTEGER,

allowNull: false,

},

date\_assigned: {

type: DataTypes.DATE,

allowNull: false,

},

}, {

tableName: 'assignments',

timestamps: false,

});

Assignment.associate = (models) => {

Assignment.belongsTo(models.Bus, { foreignKey: 'bus\_id' });

Assignment.belongsTo(models.Driver, { foreignKey: 'driver\_id' });

Assignment.belongsTo(models.Route, { foreignKey: 'route\_id' });

};

return Assignment;

};

Пример модели:

{

"assignment\_id": 1,

"bus\_id": 1,

"driver\_id": 1,

"route\_id": 1,

"date\_assigned": "2024-04-15T10:00:00Z"

}

Модель "Bus" содержит информацию о каждом автобусе в автобусном парке, включая модель, год выпуска и регистрационный номер.

const { DataTypes } = require('sequelize');

module.exports = (sequelize) => {

const Bus = sequelize.define('Bus', {

bus\_id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true,

},

model: {

type: DataTypes.STRING,

allowNull: false,

},

year: {

type: DataTypes.INTEGER,

allowNull: false,

},

registration\_number: {

type: DataTypes.STRING,

allowNull: false,

},

capacity: {

type: DataTypes.INTEGER,

allowNull: false,

},

}, {

tableName: 'buses',

timestamps: false,

});

return Bus;

};

Пример модели:

{

"bus\_id": 1,

"model": "Mercedes-Benz Sprinter",

"year": 2020,

"registration\_number": "AB123CD",

"capacity": 30

}

Модель "Driver" содержит информацию о водителях, работающих в автобусном парке, включая их имена, возраст, стаж и номер водительского удостоверения.

const { DataTypes } = require('sequelize');

module.exports = (sequelize) => {

const Driver = sequelize.define('Driver', {

driver\_id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true,

},

first\_name: {

type: DataTypes.STRING,

allowNull: false,

},

last\_name: {

type: DataTypes.STRING,

allowNull: false,

},

age: {

type: DataTypes.INTEGER,

allowNull: false,

},

experience: {

type: DataTypes.INTEGER,

allowNull: false,

},

license\_number: {

type: DataTypes.STRING,

allowNull: false,

unique: true,

},

}, {

tableName: 'drivers',

timestamps: false,

});

return Driver;

};

Пример модели:

{

"driver\_id": 1,

"first\_name": "John",

"last\_name": "Doe",

"age": 35,

"experience": 10,

"license\_number": "ABC123456"

}

Модель "Route" содержит информацию о маршрутах, которые обслуживаются автобусами, включая начальную и конечную остановки, продолжительность, расстояние и частоту движения.

Пример кода:

const { DataTypes } = require('sequelize');

module.exports = (sequelize) => {

const Route = sequelize.define('Route', {

route\_id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true,

},

start\_stop: {

type: DataTypes.STRING,

allowNull: false,

},

end\_stop: {

type: DataTypes.STRING,

allowNull: false,

},

duration: {

type: DataTypes.INTEGER,

allowNull: false,

},

distance: {

type: DataTypes.DECIMAL,

allowNull: false,

},

frequency: {

type: DataTypes.INTEGER,

allowNull: false,

},

}, {

tableName: 'routes',

timestamps: false,

});

return Route;

};

Пример модели:

{

"route\_id": 1,

"start\_stop": "Central Station",

"end\_stop": "Business District",

"duration": 30,

"distance": 10.5,

"frequency": 15

}

**3. Описание точек доступа**

3.1 Описание точек доступа для управления маршрутами

3.1.1 Создание маршрута

Метод: POST

Путь: /assignments

Описание: Создает новое задание в автобусном парке.

Пример запроса:

{

"bus\_id": 1,

"driver\_id": 1,

"route\_id": 1,

"date\_assigned": "2024-04-14T08:00:00"

}

3.1.2 Получение списка всех заданий:

Метод: GET

Путь: /assignments

Описание: Возвращает список всех заданий в автобусном парке.

3.1.3 Получение информации о задании по его идентификатору:

Метод: GET

Путь: /assignments/:id

Описание: Возвращает информацию о задании с указанным идентификатором.

Пример запроса: /assignments/1

3.1.4 Обновление информации о задании по его идентификатору:

Метод: PUT

Путь: /assignments/:id

Описание: Обновляет информацию о задании с указанным идентификатором.

Пример запроса: /assignments/1

Пример тела запроса:

{

"bus\_id": 2,

"driver\_id": 2,

"route\_id": 2,

"date\_assigned": "2024-04-15T08:00:00"

}

3.1.5 Удаление задания по его идентификатору:

Метод: DELETE

Путь: /assignments/:id

Описание: Удаляет задание с указанным идентификатором из автобусного парка.

3.1.6 Получение детальной информации о задании по его идентификатору:

Метод: GET

Путь: /assignments-detail/:assignmentId

Описание: Возвращает детальную информацию о задании с указанным идентификатором, включая информацию о связанных автобусе, маршруте и водителе.

Пример запроса: /assignments-detail/1

3.2 Описание точек доступа для управления автобусами

3.2.1 Создание автобуса:

Метод: POST

Путь: /buses

Описание: Создает новый автобус.

Пример запроса:

{

"model": "Volvo B7R",

"year": 2018,

"registration\_number": "ABC123",

"capacity": 40

}

3.2.2 Получение списка всех автобусов:

Метод: GET

Путь: /buses

Описание: Возвращает список всех автобусов.

3.2.3 Получение информации об автобусе по его идентификатору:

Метод: GET

Путь: /buses/:id

Описание: Возвращает информацию об автобусе с указанным идентификатором.

Пример запроса: /buses/1

3.2.4 Обновление информации об автобусе по его идентификатору:

Метод: PUT

Путь: /buses/:id

Описание: Обновляет информацию об автобусе с указанным идентификатором.

Пример запроса: /buses/1

Пример тела запроса:

{

"model": "Mercedes-Benz Sprinter",

"year": 2020,

"registration\_number": "XYZ456",

"capacity": 30

}

3.2.5 Удаление автобуса по его идентификатору:

Метод: DELETE

Путь: /buses/:id

Описание: Удаляет автобус с указанным идентификатором.

3.3 Описание точек доступа для управления водителями

3.3.1 Создание водителя:

Метод: POST

Путь: /drivers

Описание: Создает нового водителя.

Пример запроса:

{

"first\_name": "John",

"last\_name": "Doe",

"age": 35,

"experience": 10,

"license\_number": "ABC123456"

}

3.3.2 Получение списка всех водителей:

Метод: GET

Путь: /drivers

Описание: Возвращает список всех водителей.

3.3.3 Получение информации о водителе по его идентификатору:

Метод: GET

Путь: /drivers/:id

Описание: Возвращает информацию о водителе с указанным идентификатором.

Пример запроса: /drivers/1

3.3.4 Обновление информации о водителе по его идентификатору:

Метод: PUT

Путь: /drivers/:id

Описание: Обновляет информацию о водителе с указанным идентификатором.

Пример запроса: /drivers/1

Пример тела запроса:

{

"first\_name": "Jane",

"last\_name": "Smith",

"age": 40,

"experience": 15,

"license\_number": "XYZ987654"

}

3.3.5 Удаление водителя по его идентификатору:

Метод: DELETE

Путь: /drivers/:id

Описание: Удаляет водителя с указанным идентификатором.

3.3.6 Получение списка автобусов, связанных с водителем:

Метод: GET

Путь: /driver/:driverId/buses

Описание: Возвращает список всех автобусов, связанных с указанным водителем.

Пример запроса: /driver/1/buses

3.4 Описание точек доступа для управления маршрутами

3.4.1 Создание маршрута:

Метод: POST

Путь: /routes

Описание: Создает новый маршрут.

Пример запроса:

{

"start\_stop": "A",

"end\_stop": "B",

"duration": 60,

"distance": 10.5,

"frequency": 30

}

3.4.2 Получение списка всех маршрутов:

Метод: GET

Путь: /routes

Описание: Возвращает список всех маршрутов.

3.4.3 Получение информации о маршруте по его идентификатору:

Метод: GET

Путь: /routes/:id

Описание: Возвращает информацию о маршруте с указанным идентификатором.

Пример запроса: /routes/1

3.4.4 Обновление информации о маршруте по его идентификатору:

Метод: PUT

Путь: /routes/:id

Описание: Обновляет информацию о маршруте с указанным идентификатором.

Пример запроса: /routes/1

Пример тела запроса:

{

"start\_stop": "C",

"end\_stop": "D",

"duration": 45,

"distance": 8.2,

"frequency": 45

}

3.4.5 Удаление маршрута по его идентификатору:

Метод: DELETE

Путь: /routes/:id

Описание: Удаляет маршрут с указанным идентификатором.

**4. Разработка REST-сервиса**

4.1 Структура проекта

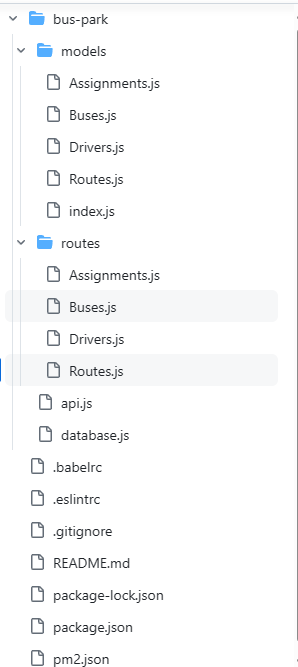


Рисунок 4.1 Структура проекта

В папке routes находятся обработчики запросов API, которые принимают запросы от клиентов, обрабатывают их и взаимодействуют с базой данных для выполнения соответствующих операций. Эти обработчики реализуют операции создания, чтения, обновления и удаления данных, а затем отправляют обратно клиентам необходимые ответы.

Модели базы данных, находящиеся в папке models, описывают структуру данных, хранящихся в базе. Они определяют таблицы и связи между ними, а также методы для выполнения операций с данными, таких как поиск, создание, обновление и удаление записей.

Файл api.js является главной точкой входа в приложение, где настраивается и запускается сервер Express, а также определяются основные маршруты API, которые будут доступны клиентам.

Файл database.js содержит конфигурацию для подключения к базе данных, устанавливая соединение и настройки для взаимодействия с базой данных.

Файлы package.json и package-lock.json содержат информацию о зависимостях и настройках проекта, включая используемые библиотеки и их версии.

4.2 Запуск проекта

Запуск проекта обычно осуществляется через команду запуска, указанную в скриптах файла package.json. В проекте скрипт для запуска называется start.

Чтобы запустить проект, вам нужно перейти в корневую директорию проекта в командной строке и выполнить следующую команду: npm start

Это запустит скрипт start, который в свою очередь запустит файл api.js, который является точкой входа в приложение Express. После этого приложение будет доступно по указанному порту.

После успешного запуска проекта вы увидите сообщение в консоли, которое указывает на то, что ваш сервер Express запущен и слушает указанный порт.

Если в процессе запуска возникают ошибки, убедитесь, что все зависимости установлены правильно с помощью команды npm install, и проверьте конфигурацию вашего проекта и код в файлах api.js и package.json.

4.3 Создание объектов базы данных

-- Create table "Buses"

CREATE TABLE Buses (

bus\_id SERIAL PRIMARY KEY,

model VARCHAR(100),

year INTEGER,

registration\_number VARCHAR(20),

capacity INTEGER

);

-- Create table "Drivers"

CREATE TABLE Drivers (

driver\_id SERIAL PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

age INTEGER,

experience INTEGER,

license\_number VARCHAR(20)

);

-- Create table "Routes"

CREATE TABLE Routes (

route\_id SERIAL PRIMARY KEY,

start\_stop VARCHAR(100),

end\_stop VARCHAR(100),

duration INTEGER,

distance NUMERIC,

frequency INTEGER

);

-- Create table "Assignments"

CREATE TABLE Assignments (

assignment\_id SERIAL PRIMARY KEY,

bus\_id INTEGER REFERENCES Buses(bus\_id),

route\_id INTEGER REFERENCES Routes(route\_id),

driver\_id INTEGER REFERENCES Drivers(driver\_id),

date\_assigned DATE NOT NULL

);

**ЗАКЛЮЧЕНИЕ**

В заключении данной курсовой работы о создании автобусного парка можно подчеркнуть важность разработки такого проекта с использованием современных технологий. Автобусные парки играют важную роль в общественном транспорте, обеспечивая перевозку людей по городу и за его пределами. Создание программного продукта для управления автобусным парком с использованием технологий Node.js, Express и PostgreSQL предоставляет возможность эффективного управления транспортными средствами, водителями, маршрутами и заданиями.

Использование Node.js для серверной разработки обеспечивает высокую производительность и масштабируемость приложения. Фреймворк Express предоставляет удобные инструменты для создания веб-приложений и API, что значительно упрощает процесс разработки. PostgreSQL выбран в качестве базы данных благодаря своей надежности, возможностям транзакций и широким функциональным возможностям.

В результате выполнения данной работы был разработан полноценный REST-сервис для управления автобусным парком, который позволяет осуществлять операции добавления, удаления, редактирования данных о транспортных средствах, водителях, маршрутах и заданиях. Этот проект дает практический опыт работы с современными технологиями и позволяет создать эффективное и масштабируемое веб-приложение для управления автобусным парком.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Документация Node.js: <https://nodejs.org/en/docs/>
2. Документация Express.js: <https://expressjs.com/>
3. Документация Sequelize): <https://sequelize.org/>
4. Документация PostgreSQL: <https://www.postgresql.org/docs/>
5. "PostgreSQL: Введение и руководство по администрированию" (Simon Riggs, Hannu Krosing)