**СОДЕРЖАНИЕ**

Введение…………………………………………………………………….6

1.Описание предметной области……………………………………. …....7

1.1 Назначение программного продукта…………………………………..7

1.2. Задачи программного продукта ..……………………………………...7

1.3. Источники входной информации……………………………………...7

1.4. Выходная информация………………………………………………...8

2. Описание маппинга…………………………………………………….10

2.1 Сущности и атрибуты…………………………………………………10

2.2 Таблицы базы данных…………………………………………………11

2.3 Связи между сущностями …………………………………………….12

3. Описание точек доступа………………………………………………..14

4. Разработка REST-сервиса……………………………………………...19

4.1 Структура проекта…………………………………………………….19

4.2 Запуск проекта…………………………………………………………20

* 1. Модель данных………………………………………………………...20
  2. Маршрутизация и контроллеры………………………………………22

Заключение………………………………………………………………...24

Список использованных источников…..………………………………...25

**ВВЕДЕНИЕ**

В современном мире развитие информационных технологий значительно влияет на различные сферы бизнеса. Одной из таких сфер является общественное питание, где автоматизация процессов становится необходимостью для увеличения эффективности и улучшения качества обслуживания. В этом контексте разработка REST-приложения для кофейни на основе Node.js и с использованием PostgreSQL представляет собой актуальную задачу.

Цель данного курсового проекта заключается в создании программного продукта, который позволит кофейне эффективно управлять заказами. Использование технологии REST обеспечивает гибкость и масштабируемость приложения, а выбор Node.js в качестве основы обеспечивает высокую производительность и возможность асинхронного программирования.

Интеграция с PostgreSQL как реляционной базой данных позволит надежно хранить информацию о клиентах, заказах, меню и других аспектах работы кофейни. Это обеспечит не только эффективное управление данными, но и возможность анализа данных для принятия стратегических решений.

В рамках данного проекта будут реализованы основные функциональные возможности кофейни, такие как обработка заказов, управление меню и взаимодействие с клиентами.

**1. Описание предметной области**

1.1 Назначение программного продукта

Целью разработки программного продукта является создание инновационного и эффективного инструмента для автоматизации основных бизнес-процессов кофейни. Приложение должно быть направлено на улучшение работы кофейни, обеспечивая эффективное управление заказами, составление меню и взаимодействие с клиентами.

1.2 Задачи программного продукта

В рамках данного проекта необходимо разработать важные функции для работы приложения. Были выделены основные из них:

* Управление заказами. Приложение позволит клиентам удобно размещать заказы на продукцию кофейни через мобильное приложение или веб-интерфейс, а также обеспечит оперативную обработку заказов внутри кофейни.
* Анализ данных: Приложение будет обеспечивать сбор и анализ данных о заказах, предпочтениях клиентов и эффективности работы кофейни для принятия обоснованных решений и улучшения бизнес-процессов.
* Меню кофейни. Приложение сможет содержать большое количество товаров и блюд, которые могут реализовываться в кофейне.

Программное решение будет способствовать повышению эффективности работы кофейни, улучшению обслуживания клиентов и расширению возможностей для развития бизнеса.

1.3. Источники входной информации

Для разработки информационной системы кофейни были использованы следующие источники входной информации:

* Справочная литература: В процессе подготовки курсовой работы были изучены учебники и учебные пособия по управлению кофейными заведениями, а также литература по технологиям разработки программного обеспечения и базам данных. Эти источники позволили получить представление о требованиях к функциональности и управлению кофейней.
* Анализ конкурентов и существующих решений. Был проведен анализ конкурентов и изучены существующие информационные системы для кофеен. Это позволило выявить особенности и лучшие практики в управлении заказами, составление меню и взаимодействием с клиентами.

Эти источники информации стали основой для определения функциональных требований к разрабатываемой информационной системе и обеспечили соответствие сервиса потребностям пользователей и бизнес-целям проекта.

1.4. Выходная информация

Информационная система кофейни предоставляет разнообразную выходную информацию, необходимую для эффективного управления и принятия стратегических решений. Вот основные категории выходной информации:

* Отчеты и статистика:

Статистические данные о продажах: отчеты о количестве заказов, обороте и популярности определенных продуктов.

* Информация о заказах и меню:

Расписание и состав заказов: информация о заказах клиентов, включая состав и время исполнения.

Меню кофейни: перечень доступных продуктов с описанием и ценами, включая специальные предложения и акции.

* Управление кофейней:

Операции по управлению меню: возможность добавления, изменения и удаления продуктов из меню.

Управление персоналом: возможность добавления новых сотрудников, изменения их ролей и доступа к системе.

Управление инвентарем: операции по учету и пополнению запасов ингредиентов и оборудования.

Эта выходная информация предоставляется для администрации кофейни, баристам, а также может быть доступна для клиентов через мобильное приложение или веб-интерфейс. Она помогает в принятии решений по улучшению работы кофейни и удовлетворению потребностей клиентов.

**2. Описание маппинга**

Приложение для кофейни разрабатывается с использованием технологии Node.js для серверной части и PostgreSQL в качестве системы управления базами данных. Целью приложения является автоматизация управления заказами, инвентаризацией и взаимодействием с клиентами, обеспечивая эффективное функционирование кофейни и повышая удовлетворенность клиентов.

2.1 Сущности и атрибуты

При проектировании базы данных для информационной системы кофейни необходимо определить основные сущности и их атрибуты, которые будут храниться в базе данных. Ниже приведено описание каждой сущности и их атрибутов:

1. Продукты:

* product\_id - уникальный идентификатор продукта.
* product\_name - название продукта.
* price - цена продукта.
* description - описание продукта.

1. Заказы:

* order\_id - уникальный идентификатор заказа.
* order\_datetime - дата и время размещения заказа.
* order\_status - статус заказа.
* client\_id - идентификатор клиента, разместившего заказ.

1. Клиенты:

* client\_id - уникальный идентификатор клиента.
* client\_name - имя клиента.
* contact\_info - контактная информация клиента.

1. Содержимое заказа:

* item\_id - уникальный идентификатор элемента заказа.
* product\_id - идентификатор продукта, включенного в заказ.
* order\_id - идентификатор заказа.
* quantity - количество продукта в заказе.

Эти сущности и атрибуты обеспечивают основу для хранения информации о продуктах, заказах и клиентах в базе данных кофейни. Каждая сущность имеет свои уникальные атрибуты, которые позволяют эффективно управлять данными и обеспечивать работу информационной системы.

2.2 Таблицы базы данных

При разработке информационной системы кофейни были созданы следующие таблицы в базе данных для хранения данных о продуктах, заказах, клиентах и содержимом заказов:

Таблица "Продукты":

product\_id (тип: SERIAL, ограничение: PRIMARY KEY) - уникальный идентификатор продукта.

product\_name (тип: VARCHAR(255), ограничение: NOT NULL) - название продукта.

price (тип: NUMERIC(10, 2), ограничение: NOT NULL) - цена продукта.

description (тип: TEXT) - описание продукта.

Таблица "Заказы":

order\_id (тип: SERIAL, ограничение: PRIMARY KEY) - уникальный идентификатор заказа.

order\_datetime (тип: TIMESTAMP, ограничение: NOT NULL, значение по умолчанию: CURRENT\_TIMESTAMP) - дата и время размещения заказа.

order\_status (тип: VARCHAR(50), ограничение: NOT NULL) - статус заказа.

client\_id (тип: INTEGER, ограничение: FOREIGN KEY REFERENCES Clients(client\_id)) - идентификатор клиента, разместившего заказ.

Таблица "Клиенты":

client\_id (тип: SERIAL, ограничение: PRIMARY KEY) - уникальный идентификатор клиента.

client\_name (тип: VARCHAR(100), ограничение: NOT NULL) - имя клиента.

contact\_info (тип: TEXT) - контактная информация клиента.

Таблица "Содержимое заказа":

item\_id (тип: SERIAL, ограничение: PRIMARY KEY) - уникальный идентификатор элемента заказа.

product\_id (тип: INTEGER, ограничение: FOREIGN KEY REFERENCES Products(product\_id)) - идентификатор продукта, включенного в заказ.

order\_id (тип: INTEGER, ограничение: FOREIGN KEY REFERENCES Orders(order\_id)) - идентификатор заказа.

quantity (тип: INTEGER, ограничение: NOT NULL) - количество продукта в заказе.

Эти таблицы представляют собой основу для хранения информации о продуктах, заказах и клиентах, а также связи между ними в базе данных кофейни. Каждая таблица имеет уникальные поля, определенные для эффективного хранения и обработки данных в рамках информационной системы.

2.3 Связи между сущностями

При проектировании базы данных для информационной системы кофейни необходимо определить связи между различными сущностями, чтобы эффективно организовать хранение и обработку данных. В данном разделе описываются связи между основными сущностями: продуктами, заказами и клиентами.

* Связь "Один ко многим" между таблицей "Продукты" и "Содержимое заказа":

Поле product\_id в таблице "Содержимое заказа" является внешним ключом, ссылается на поле product\_id в таблице "Продукты".

Эта связь позволяет одному продукту быть включенным в несколько различных заказов.

* Связь "Один ко многим" между таблицей "Заказы" и "Содержимое заказа":

Поле order\_id в таблице "Содержимое заказа" является внешним ключом, ссылается на поле order\_id в таблице "Заказы".

Эта связь позволяет одному заказу содержать несколько элементов заказа, каждый из которых связан с определенным продуктом.

* Связь "Один ко многим" между таблицей "Клиенты" и "Заказы":

Поле client\_id в таблице "Заказы" является внешним ключом, ссылается на поле client\_id в таблице "Клиенты".

Эта связь позволяет одному клиенту размещать несколько заказов.

Эти связи определяют взаимосвязь между различными сущностями в базе данных и обеспечивают целостность данных при их хранении и обработке. Каждая связь имеет свою роль в организации данных и функционировании информационной системы кофейни.

2.4 Описание моделей Sequelize

В ходе разработки программного средства для сервиса кофейни выявилась проблема сопоставления полей данных между классами, отвечающими за различные функциональные задачи. Для решения этой проблемы была использована библиотека Sequelize в Node.js.

Sequelize предоставляет удобные инструменты для работы с базой данных PostgreSQL и автоматический маппинг объектов JavaScript на таблицы базы данных.

Рассмотрим одну из моделей:

module.exports = (sequelize, DataTypes) => {

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

product\_id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true,

},

product\_name: {

type: DataTypes.STRING,

allowNull: false,

},

price: {

type: DataTypes.NUMERIC(10, 2),

allowNull: false,

},

description: {

type: DataTypes.TEXT,

allowNull: true,

},

}, {

tableName: 'products',

timestamps: false,

});

Products.associate = (models) => {

Products.hasMany(models.Order\_Items, { foreignKey: 'product\_id' });

};

return Products;

};

Модель "Products" представляет собой сущность товаров. Ниже представлены описание полей модели:

Поля таблицы:

product\_id: Первичный ключ, автоинкрементируемый идентификатор продукта.

product\_name: Название продукта, обязательное поле.

price: Цена продукта, обязательное поле с двумя десятичными знаками.

description: Описание продукта, необязательное поле.

Ассоциации:

Products.hasMany(models.Order\_Items, { foreignKey: 'product\_id' }): Продукт может быть связан с множеством элементов заказа через внешний ключ product\_id.

С помощью этой модели мы можем генерировать новые записи в базе данных, изменять, удалять, а также выводить выборки из бд.

**3. Описание точек доступа**

Описанные ниже точки доступа предоставляют API для управления продуктами, клиентами, заказами и их содержимым в информационной системе кофейни. Каждая точка доступа имеет определенный метод, путь, параметры запроса (при необходимости) и формат тела запроса для выполнения определенной операции.

1. Создание продукта (POST):

Метод: POST

Путь: localhost:3333/create/product

Тело запроса: JSON

Поля тела запроса:

* product\_name (строка) - название продукта
* price (число) - цена продукта
* description (строка) - описание продукта

1. Получение всех товаров (GET):

Метод: GET

Путь: localhost:3333/getAll/product

1. Поиск продукта по id (GET):

Метод: GET

Путь: localhost:3333/get/product?id=1

Параметры запроса:

* id (число) - идентификатор продукта

1. Обновление продукта (PUT):

Метод: PUT

Путь: localhost:3333/update/product?id=1

Параметры запроса:

* id (число) - идентификатор продукта

Тело запроса: JSON

Поля тела запроса:

* product\_name (строка) - новое название продукта
* price (число) - новая цена продукта
* description (строка) - новое описание продукта

1. Удаление продукта (DELETE):

Метод: DELETE

Путь: localhost:3333/delete/product?id=1

Параметры запроса:

* id (число) - идентификатор продукта

1. Добавить клиента (POST):

Метод: POST

Путь: localhost:3333/create/client

Тело запроса: JSON

Поля тела запроса:

* client\_name (строка) - имя клиента
* contact\_info (строка) - контактная информация клиента

1. Получить всех клиентов (GET):

Метод: GET

Путь: localhost:3333/getAll/client

8. Получить клиента по id (GET):

Метод: GET

Путь: localhost:3333/get/client?id=1

Параметры запроса:

* id (число) - идентификатор клиента

9. Обновление клиента (PUT):

Метод: PUT

Путь: localhost:3333/update/client?id=1

Параметры запроса:

* id (число) - идентификатор клиента

Тело запроса: JSON

Поля тела запроса:

* client\_name (строка) - новое имя клиента

10. Удаление клиента (DELETE):

Метод: DELETE

Путь: localhost:3333/delete/client?id=1

Параметры запроса:

* id (число) - идентификатор клиента

11. Создать заказ (POST):

Метод: POST

Путь: localhost:3333/create/order

Тело запроса: JSON

Поля тела запроса:

* order\_status (строка) - статус заказа
* client\_id (число) - идентификатор клиента, для которого создается заказ

12. Получить все заказы (GET):

Метод: GET

Путь: localhost:3333/getAll/order

13. Получить заказ по id (GET):

Метод: GET

Путь: localhost:3333/get/order?id=1

Параметры запроса:

* id (число) - идентификатор заказа

14. Обновление заказа (PUT):

Метод: PUT

Путь: localhost:3333/update/order?id=1

Параметры запроса:

* id (число) - идентификатор заказа

Тело запроса: JSON

Поля тела запроса:

* order\_datetime (строка) - новая дата и время заказа
* order\_status (строка) - новый статус заказа
* client\_id (число) - идентификатор клиента

15. Удалить заказ (DELETE):

Метод: DELETE

Путь: localhost:3333/delete/order?id=1

Параметры запроса:

* id (число) - идентификатор заказа

16. Создать наполнение заказа (POST):

Метод: POST

Путь: localhost:3333/create/order-item

Тело запроса: JSON

Поля тела запроса:

* order\_id (число) - идентификатор заказа
* product\_id (число) - идентификатор продукта
* quantity (число) - количество продукта в заказе

17. Получить все наполнения заказов (GET):

Метод: GET

Путь: localhost:3333/getAll/order-item

18. Получить наполнение по id (GET):

Метод: GET

Путь: localhost:3333/get/order-item?id=1

Параметры запроса:

* id (число) - идентификатор элемента заказа

19. Обновить наполнение заказа (PUT):

Метод: PUT

Путь: localhost:3333/update/order-item?id=1

Параметры запроса:

* id (число) - идентификатор элемента заказа

Тело запроса: JSON

Поля тела запроса:

* product\_id (число) - новый идентификатор продукта
* order\_id (число) - новый идентификатор заказа
* quantity (число) - новое количество продукта в заказе

20. Удалить наполнение заказа (DELETE):

Метод: DELETE

Путь: localhost:3333/delete/order-item?id=1

Параметры запроса:

* id (число) - идентификатор элемента заказа

21. Получение количества заказов за сегодня (GET):

Метод: GET

Путь: localhost:3333/get/count-order-today

22. Получение полной информации о заказе (GET):

Метод: GET

Путь: localhost:3333/get/order-info?id=2

Параметры запроса:

* id (число) - идентификатор заказа.

**4. Разработка REST-сервиса**

4.1 Структура проекта

Проект состоит из нескольких основных компонентов, организованных в соответствии с модульной архитектурой:

* Конфигурация (config/):

Директория содержит файлы конфигурации проекта, такие как настройки порта, базы данных и маршрутов API.

Файлы конфигурации структурированы для удобства управления параметрами приложения в различных окружениях (development, production).

* Сервисы (services/):

Директория содержит сервисы, отвечающие за взаимодействие с внешними ресурсами, такими как база данных.

Каждый сервис организован в отдельный модуль для обеспечения четкой структуры и разделения ответственности.

* Контроллеры (api/controllers/):

Директория содержит контроллеры, обрабатывающие HTTP-запросы и возвращающие соответствующие HTTP-ответы.

Контроллеры используются для взаимодействия с бизнес-логикой и обработки данных, полученных от клиентов.

* Маршрутизация (api/routes/):

Директория содержит файлы маршрутов, определяющие маршруты API и их соответствующие обработчики контроллеров.

Каждый маршрут определяет метод запроса, путь и соответствующий контроллер для обработки запросов.

* Статические ресурсы (public/):

Директория содержит статические ресурсы, такие как изображения, стили CSS, клиентский JavaScript.

* Файлы приложения (app.js, server.js):

Файл app.js является точкой входа приложения, где создается экземпляр Express и настраиваются его middleware.

Файл server.js запускает HTTP-сервер и привязывает Express-приложение к определенному порту.

Также, в проекте используются сторонние библиотеки для облегчения разработки и расширения функциональности, такие как body-parser, helmet, cors.

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.

* 1. Модель данных

Модель данных — это абстрактное представление о данных в информационной системе, которое определяет их структуру, типы, связи и правила хранения. Она служит основой для организации данных в базе данных и обеспечивает их удобное и эффективное использование.

Модель данных описывает, каким образом данные будут представлены и организованы в базе данных, что позволяет разработчикам и администраторам системы лучше понимать структуру данных и обеспечивать их корректное использование.

Рассмотрим следующую модель данных для сущности "Клиенты":

module.exports = (sequelize, DataTypes) => {

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

client\_id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true,

},

client\_name: {

type: DataTypes.STRING(100),

allowNull: false,

},

contact\_info: {

type: DataTypes.TEXT,

allowNull: true,

},

}, {

tableName: 'clients',

timestamps: false,

});

Clients.associate = (models) => {

Clients.hasMany(models.Orders, { foreignKey: 'client\_id' });

};

return Clients;

};

Описание модели:

* client\_id: Тип: INTEGER Определение: Уникальный идентификатор клиента Свойства: Первичный ключ (primaryKey), автоинкремент (autoIncrement)
* client\_name: Тип: STRING(100) Определение: Имя клиента Свойства: Обязательное поле (allowNull: false)
* contact\_info: Тип: TEXT Определение: Контактная информация клиента Свойства: Поле может быть пустым (allowNull: true)

Дополнительные свойства модели:

* tableName: Название таблицы в базе данных, в которой хранятся данные о клиентах.
* timestamps: Указывает, нужно ли добавлять поля createdAt и updatedAt в записи о клиентах. Значение false означает, что эти поля не будут созданы.

Ассоциация с другими моделями:

Модель Clients ассоциирована с моделью Orders по отношению "один ко многим", где один клиент может иметь несколько заказов. Это отображено в методе Clients.associate, где указано, что у клиента может быть много заказов, связь осуществляется через внешний ключ client\_id в таблице Orders.

Эта модель данных используется для хранения информации о клиентах кофейни в базе данных. Каждая запись о клиенте содержит его уникальный идентификатор, имя и контактную информацию. Также эта модель связана с моделью заказов для отслеживания заказов, сделанных каждым клиентом.

4.4 Маршрутизация и контроллеры

Рассмотрим метод getAllClients из контроллера ClientsController, который отвечает за получение списка всех клиентов. Рассмотрим этот метод от получения запроса до отправления ответа:

async getAllClients(req, res) {

try {

// Получение всех клиентов из базы данных

const allClients = await Clients.findAll();

// Отправка ответа с массивом всех клиентов

res.json(allClients);

} catch (error) {

// Обработка ошибки, если что-то пошло не так

console.error('Ошибка при получении всех клиентов:', error);

res.status(500).json({ error: 'Ошибка при получении всех клиентов' });

}

}

Этот метод выполняет следующие действия:

* Получение запроса:Метод getAllClients ожидает получить HTTP-запрос от клиента. В данном случае, не ожидается передачи каких-либо данных в теле запроса или параметрах.
* Обработка запроса: В блоке try, метод использует модель Clients для выполнения запроса к базе данных и получения списка всех клиентов. Метод findAll() используется для извлечения всех записей из таблицы клиентов.
* Отправка ответа: Если запрос выполнен успешно и клиенты успешно получены из базы данных, метод отправляет ответ обратно клиенту. Используется метод res.json(), чтобы отправить JSON-представление массива всех клиентов в ответе.
* Обработка ошибок: Если в процессе получения клиентов возникает ошибка, контроллер перехватывает исключение в блоке catch. Ошибка обрабатывается путем логирования сообщения об ошибке и отправки соответствующего HTTP-статуса и сообщения об ошибке в ответе.

Таким образом, метод getAllClients обеспечивает получение списка всех клиентов из базы данных и отправляет его в ответ клиенту в формате JSON.

**ЗАКЛЮЧЕНИЕ**

В заключении данной курсовой работы о разработке приложения для кофейни можно подчеркнуть значимость создания такого проекта с применением современных технологий. Кофейни являются неотъемлемой частью нашей культуры, обеспечивая не только напитки, но и место для встреч, работы и отдыха. Разработка программного продукта для управления кофейней с использованием Node.js, Express и PostgreSQL предоставляет возможность эффективного управления всеми аспектами бизнеса, начиная от управления меню и заказами, и заканчивая аналитикой и отчетностью.

Использование Node.js для серверной разработки обеспечивает высокую производительность и гибкость приложения. Фреймворк Express предоставляет интуитивно понятные средства для создания веб-приложений, а PostgreSQL, как надежная реляционная база данных, обеспечивает сохранность и целостность данных.

В результате выполнения данной работы было создано полнофункциональное приложение для управления кофейней, которое позволяет управлять меню, заказами, клиентами и отслеживать различные аспекты работы кофейни. Этот проект дает практический опыт работы с современными технологиями и предоставляет возможность создать эффективное и масштабируемое веб-приложение для управления кофейней.

**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)