Міністерство освіти та науки України Харківський національний університет радіоелектроніки

Кафедра ПІ

Звіт

З лабораторної роботи 2

Тема роботи: «РОЗРОБКА БАЗИ ДАНИХ ДЛЯ СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМНОЇ СИСТЕМИ ТА ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСУ (API)»

з дисципліни «Аналіз та рефакторинг коду»

Виконав: Перевірив:

ст. гр. ПЗПІ-22-10 ст. викладач Сокорчук І.П.

Клецов М.Д.

Харків 2024

**Мета роботи:** На лабораторній роботі №2 потрібно розробити базу даних для серверної частини програмної системи та прикладного програмного інтерфейсу.

**Хід роботи:**

Розробити будову програмної системи.

1. Створити UML діаграму прецедентів для серверної частини системи.

Діаграма прецедентів для системи моніторингу авторафіку відображає взаємодію між користувачами (адміністратором і користувачем) та функціональністю системи.

Изображение выглядит как текст, диаграмма, План, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.1 – UML діаграму прецедентів

1. Створити ER діаграму даних.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.1 – ER діаграма даних

1. Розробити базу даних (БД) програмної системи.

Ми створили чотири таблиці для бази даних, що описують транспортні засоби, користувачів, маршрути та сповіщення. Ось запити для створення кожної з них:

CREATE DATABASE TrafficMonitoring; -- створення БД з іменем TrafficMonitoring

CREATE USER Nikita WITH PASSWORD '123123123'; -- створення користувача Nikita, що буде мати доступ до БД

-- Створення таблиці Sensors

CREATE TABLE Sensors (

id SERIAL PRIMARY KEY,

location TEXT NOT NULL,

type VARCHAR(50) NOT NULL,

status VARCHAR(10) CHECK (status IN ('active', 'inactive')) DEFAULT 'active'

);

-- Створення таблиці TrafficJams

CREATE TABLE TrafficJams (

id SERIAL PRIMARY KEY,

location TEXT NOT NULL,

severity VARCHAR(10) CHECK (severity IN ('low', 'medium', 'high')) NOT NULL,

timestamp TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

sensor\_id INT,

FOREIGN KEY (sensor\_id) REFERENCES Sensors(id) ON DELETE SET NULL

);

-- Створення таблиці Users

CREATE TABLE Users (

id SERIAL PRIMARY KEY,

username VARCHAR(20) NOT NULL UNIQUE,

email VARCHAR(20) NOT NULL UNIQUE,

password VARCHAR(20) NOT NULL

);

-- Створення таблиці Notifications

CREATE TABLE Notifications (

id SERIAL PRIMARY KEY,

user\_id INT,

message TEXT NOT NULL,

timestamp TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES Users(id) ON DELETE CASCADE);Ці таблиці є основою для зберігання та організації даних про транспортні засоби, користувачів, маршрути та сповіщення в базі даних.

1. Створити діаграму структури БД.

Було сформовано ER-діаграму, яка візуально відображає сутності, їхні атрибути та зв'язки між ними, що допомагає краще зрозуміти структуру даних і забезпечує зручність при розробці та використанні бази даних.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4.1 – Діаграма структури БД

1. Розробити функції роботи з БД (ORM або CoRM тощо).

Для роботи з базою даних були розроблені функції, що реалізують ключові операції в системі, такі як авторизація, реєстрація користувачів, а також додавання нових автомобілів і маршрутів. Ці функції взаємодіють із базою даних PostgreSQL, виконуючи SQL-запити для збереження, оновлення та отримання необхідних даних. У Додатку А наведено код для демонстрації виконання завдання.

1. Розробити API (REST або GraphQL, gRPC тощо) для взаємодії серверної частини з клієнтами.

Для забезпечення ефективної взаємодії між серверною частиною і клієнтами було розроблено API на основі REST. API включає основні функціональні можливості для роботи з користувачами, автомобілями та маршрутами. Код для реалізації завдання наведено у Додатку Б.

1. Створити програмну реалізацію розробленого API та функцій роботи з БД.

7. Створення програмної реалізації розробленого API

Програмна реалізація розробленого API забезпечує інтерактивну взаємодію між серверною частиною та клієнтами. Це дозволяє клієнтам виконувати операції, такі як реєстрація, авторизація користувачів, а також додавання нових автомобілів і маршрутів у систему.

Изображение выглядит как текст, программное обеспечение, число, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 7.1 – Розроблений API

Посилання на відео-звіт: https://youtu.be/uHSYoW-O9PY

**Висновки:** У процесі розробки було створено API, яке забезпечує взаємодію між серверною частиною і клієнтами. API реалізує функції реєстрації та авторизації користувачів, а також додавання нових автомобілів і маршрутів. Використання фреймворка Express для обробки HTTP-запитів та PostgreSQL для збереження даних дозволило створити ефективну і масштабовану серверну частину.

**Додаток А – Функції роботи з БД**

const { Pool } = require("pg");

require("dotenv").config();

const pool = new Pool({

user: process.env.DB\_USER,

host: process.env.DB\_HOST,

database: process.env.DB\_NAME,

password: process.env.DB\_PASS,

port: process.env.DB\_PORT,

});

pool.connect()

.then(() => console.log("PostgreSQL успішно підключено"))

.catch(err => console.error("Помилка підключення до БД", err));

module.exports = pool;

**Додаток Б – API для взаємодії серверної частини з клієнтами**

const express = require("express");

const User = require("../models/userModel");

const router = express.Router();

// Отримати всіх користувачів

router.get("/", async (req, res) => {

try {

const users = await User.getAll();

res.json(users);

} catch (error) {

res.status(500).json({ message: error.message });

}

});

// Додати нового користувача

router.post("/", async (req, res) => {

try {

const { username, email, password } = req.body;

const newUser = await User.create(username, email, password);

res.status(201).json(newUser);

} catch (error) {

res.status(400).json({ message: error.message });

}

});

// Оновити інформацію про користувача

router.put("/:id", async (req, res) => {

try {

const { id } = req.params;

const { username, email, password } = req.body;

const updatedUser = await User.update(id, username, email, password);

res.json(updatedUser);

} catch (error) {

res.status(400).json({ message: error.message });

}

});

// Видалити користувача

router.delete("/:id", async (req, res) => {

try {

const { id } = req.params;

await User.delete(id);

res.json({ message: "Користувача успішно видалено" });

} catch (error) {

res.status(500).json({ message: error.message });

}

});

module.exports = router;