МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра «Програмної інженерії»

ЗВІТ

З Лабораторної роботи № 2 з дисципліни

«Аналіз та рефакторинг коду»

|  |  |
| --- | --- |
| Виконав  ст. гр. ПЗПІ-22-4  Попов Богдан | Перевірив  ст. викладач каф. ПІ Сокорчук І. П. |

Харків 2024

# Вступ

* 1. Тема

Розробка бази даних для серверної частини програмної системи та прикладного програмного інтерфейсу (АРІ).

* 1. Мета

Розробити базу даних для серверної частини програмної системи та прикладного програмного інтерфейсу.

# Розробка діаграм

* 1. Розробка UML діаграми прецедентів для серверної частини системи

Для UML-діаграми прецедентів було обрано акторів: Власник автомобіля (Car Owner) та Адміністратор (Admin). Прецеденти включають такі функції:

* Реєстрація користувача;
* Авторизація користувача;
* Перегляд історії місцезнаходження та технічного стану автомобіля;
* Отримання рекомендацій щодо технічного обслуговування;
* Отримання сповіщень про технічний стан та події;
* CRUD-операції для управління даними про транспортні засоби, користувачів та рекомендації (для адміністратора).

На основі зазначених акторів і прецедентів розроблено UML-діаграму, представлену на рисунку 2.1.

Изображение выглядит как текст, диаграмма, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 2.1 – UML діаграма прецедентів

* 1. Розробка ER діаграми даних

User (Користувач) — інформація про власників автомобілів; Vehicle (Автомобіль) — дані про зареєстровані транспортні засоби; LocationHistory (Історія місцезнаходження) — GPS-координати транспортного засобу; VehicleStatus (Стан автомобіля) — технічні показники, такі як температура двигуна, тиск у шинах тощо; Recommendation (Рекомендація) — персоналізовані поради для користувачів щодо обслуговування; Notification (Сповіщення) — повідомлення про стан автомобіля або події; Admin (Адміністратор) — дані про адміністраторів системи.

На рисунку 2.2 представлено ER-діаграму даних, що відображає взаємозв’язки між таблицями.

Изображение выглядит как снимок экрана, текст, 3D-моделирование, Графическое программное обеспечение

Автоматически созданное описание

Рис. 2.2 – ER діаграма даних

# Розробка бази даних

* + - 1. Розроблення бази даних програмної системи Код наведено у [додатку А](#_bookmark0)

Код наведено у додатку А.

Розробка бази даних для програмної системи AutoTrack здійснюється з урахуванням вимог щодо зберігання та обробки даних про користувачів, транспортні засоби, історію місцезнаходження, технічний стан автомобілів та інші важливі аспекти системи. Для цієї мети вибрано рішення на основі MongoDB — документно-орієнтованої бази даних, яка забезпечує високу гнучкість та масштабованість, особливо в контексті роботи з даними IoT-пристроїв і складними взаємозв'язками між різними сутностями.

Структура бази даних складається з наступних основних колекцій:

3.1.1 Колекція users

Колекція містить дані про користувачів системи. Кожен документ цієї колекції представляє інформацію про користувача, таку як ім’я, електронна пошта, пароль, телефон та роль (користувач або адміністратор).

3.1.2 Колекція vehicles

Ця колекція зберігає дані про транспортні засоби. Документи містять інформацію про марку, модель, рік випуску, VIN-код та посилання на користувача-власника.

3.1.3 Колекція location\_history

Колекція містить GPS-координати транспортного засобу. Кожен документ містить ідентифікатор транспортного засобу, широту, довготу, а також дату й час запису.

3.1.4 Колекція vehicle\_status

Ця колекція зберігає технічний стан автомобіля. Документи включають такі параметри, як температура двигуна, тиск у шинах, рівень пального та дата запису.

3.1.5 Колекція recommendations

Колекція містить персоналізовані рекомендації для користувачів на основі аналізу даних про транспортний засіб. Документи включають текст рекомендації, дату створення та ідентифікатор транспортного засобу.

3.1.6 Колекція notifications

Ця колекція зберігає сповіщення для користувачів, такі як нагадування про технічне обслуговування або попередження про проблеми. Документи включають текст повідомлення, статус прочитання та дату створення.

Зв’язки між колекціями

Колекції пов’язані між собою через поля типу ObjectId, що дозволяє підтримувати зв’язки між документами:

* users — основна колекція, що містить інформацію про користувачів.
* vehicles пов’язана з users через поле user\_id.
* location\_history та vehicle\_status пов’язані з vehicles через поле vehicle\_id.
* recommendations і notifications можуть бути пов’язані як з vehicles, так і з users.

Використання Mongoose для роботи з MongoDB

Для роботи з базою даних використовується Mongoose, що спрощує створення моделей, валідацію даних та взаємодію з MongoDB. Mongoose підтримує визначення схем даних, методи маніпуляції документами та колекціями, а також функціональність для зв’язків між документами через ref.

Таке проектування бази даних забезпечує ефективне зберігання й обробку даних та дозволяє масштабувати систему в майбутньому.

* + - 1. Розробка діаграми структури бази даних

Після створення бази даних, була розроблена діаграма структури бази даних (див. рис. 3.1).

Изображение выглядит как текст, снимок экрана, число, дизайн

Автоматически созданное описание

Рис. 3.1 – діаграма структури бази даних

# 4 Розробка серверної частини

* 1. Розробка АРІ для взаємодії серверної частини з клієнтами

# Для реалізації серверної частини системи AutoTrack було розроблено маршрути та контролери, що забезпечують функціонал для реєстрації, авторизації, управління транспортними засобами, перегляду історії місцезнаходження та стану автомобіля.

# 4.2.1 Маршрути:

# POST /registration – маршрут для реєстрації нового користувача.

# POST /login – маршрут для авторизації користувача.

# POST /logout – маршрут для виходу користувача.

# GET /activate/:link – маршрут для активації акаунта через посилання активації.

# GET /refresh – маршрут для оновлення сесії за допомогою refresh токену.

# GET /users – маршрут для отримання списку всіх користувачів (доступний лише адміністраторам).

# GET /vehicles – маршрут для отримання списку транспортних засобів користувача.

# POST /vehicles – маршрут для додавання нового транспортного засобу.

# GET /vehicles/:id/location-history – маршрут для отримання історії місцезнаходження автомобіля.

# GET /vehicles/:id/status – маршрут для перегляду технічного стану автомобіля.

# 4.2.2 Контролери:

# UserController:

# registration: Реєстрація користувача з валідацією даних. При успішній реєстрації генерується refresh токен, який зберігається у cookie.

# login: Авторизація користувача з перевіркою введених даних на відповідність збереженим у базі даних. При успішному вході генерується refresh токен.

# logout: Вихід користувача через видалення refresh токену з cookies.

# activate: Активація користувача за допомогою активаційного посилання, яке надсилається на email після реєстрації.

# refresh: Оновлення токенів для підтримання сесії без повторної авторизації.

# getUsers: Отримання списку всіх користувачів (доступно лише адміністраторам).

# VehicleController:

# getVehicles: Отримання списку транспортних засобів, пов’язаних із користувачем.

# addVehicle: Додавання нового автомобіля до системи.

# getLocationHistory: Отримання історії GPS-координат для конкретного автомобіля.

# getVehicleStatus: Отримання технічного стану автомобіля (температура двигуна, тиск у шинах тощо).

# 4.2.3 Код реалізації:

# Наведено у [додатку Б.](#_Додаток_Б)

# Цей набір маршрутів і контролерів забезпечує базову функціональність системи AutoTrack для реєстрації та авторизації користувачів, управління транспортними засобами, а також перегляду ключових даних про автомобілі.

# На рисунку 4.1 представлено програмну реалізацію API.

# Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение Автоматически созданное описание

# Рис. 4.1 – програмна реалізація для АРІ

# 5 Висновок

У процесі виконання лабораторної роботи було розроблено UML-діаграму прецедентів для серверної частини системи AutoTrack та ER-діаграму даних. На основі цих діаграм була створена база даних для програмної системи. Потім було сформовано діаграму структури бази даних. Для роботи з цією базою даних розроблено функції, а також створено API для забезпечення взаємодії серверної частини з клієнтами. Завершальним етапом стала програмна реалізація розробленого API.

Посилання на відео:

# Додаток А

1 const express = require('express');

2 const mongoose = require('mongoose');

3 const bodyParser = require('body-parser');

4

5 // Підключення до MongoDB

6 mongoose.connect('mongodb://localhost:27017/your\_project', {

7 useNewUrlParser: true,

8 useUnifiedTopology: true,

9 });

10

11 // Створення схем та моделей

12

13 // Схема користувача

14 const userSchema = new mongoose.Schema({

15 first\_name: { type: String, required: true },

16 last\_name: { type: String, required: true },

17 email: { type: String, required: true, unique: true },

18 password\_hash: { type: String, required: true },

19 phone: String,

20 role: { type: String, enum: ['User', 'Admin'], required: true },

21 created\_at: { type: Date, default: Date.now },

22 updated\_at: { type: Date, default: Date.now },

23 });

24

25 const User = mongoose.model('User', userSchema);

26

27 // Схема записів про активності

28 const activitySchema = new mongoose.Schema({

29 user\_id: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },

30 type: { type: String, required: true },

31 duration: Number,

32 date: { type: Date, default: Date.now },

33 created\_at: { type: Date, default: Date.now },

34 updated\_at: { type: Date, default: Date.now },

35 });

36

37 const Activity = mongoose.model('Activity', activitySchema);

38

39 // Схема для сповіщень

40 const notificationSchema = new mongoose.Schema({

41 user\_id: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },

42 message: { type: String, required: true },

43 type: { type: String, enum: ['Reminder', 'Alert', 'Info'], required: true },

44 status: { type: String, enum: ['Sent', 'Pending', 'Failed'], default: 'Pending' },

45 created\_at: { type: Date, default: Date.now },

46 updated\_at: { type: Date, default: Date.now },

47 });

48

49 const Notification = mongoose.model('Notification', notificationSchema);

50

51 // Ініціалізація Express

52 const app = express();

53 app.use(bodyParser.json());

54

55 // Маршрут для створення користувача

56 app.post('/users', async (req, res) => {

57 try {

58 const user = new User(req.body);

59 await user.save();

60 res.status(201).json(user);

61 } catch (error) {

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

63 }

64 });

65

66 // Маршрут для отримання всіх користувачів

67 app.get('/users', async (req, res) => {

68 try {

69 const users = await User.find();

70 res.status(200).json(users);

71 } catch (error) {

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

73 }

74 });

75

76 // Маршрут для створення активності

77 app.post('/activities', async (req, res) => {

78 try {

79 const activity = new Activity(req.body);

80 await activity.save();

81 res.status(201).json(activity);

82 } catch (error) {

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

84 }

85 });

86

87 // Маршрут для отримання всіх активностей

88 app.get('/activities', async (req, res) => {

89 try {

90 const activities = await Activity.find().populate('user\_id', 'first\_name last\_name email');

91 res.status(200).json(activities);

92 } catch (error) {

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

94 }

95 });

96

97 // Запуск сервера

98 const PORT = 3000;

99 app.listen(PORT, () => {

100 console.log(`Server is running on port ${PORT}`);

101 });

# Додаток Б

1 const UserModel = require("../models/user-model");

2 const bcrypt = require("bcrypt");

3 const uuid = require("uuid");

4

5 const tokenService = require("./token-service");

6 const UserDto = require("../dtos/user-dto");

7 const ApiError = require("../exceptions/api-error");

8

9 class UserService {

10 async registration(email, password) {

11 const candidate = await UserModel.findOne({ email });

12 if (candidate) {

13 throw ApiError.BadRequest(

14 `User with email address ${email} already exists`

15 );

16 }

17 const hashPassword = await bcrypt.hash(password, 3);

18 const activationLink = uuid.v4(); // v34fa-asfasf-142saf-sa-asf

19

20 const user = await UserModel.create({

21 email,

22 password: hashPassword,

23 activationLink,

24 });

25

26 const userDto = new UserDto(user); // id, email, isActivated

27 const tokens = tokenService.generateTokens({ ...userDto });

28 await tokenService.saveToken(userDto.id, tokens.refreshToken);

29

30 return { ...tokens, user: userDto };

31 }

32

33 async activate(activationLink) {

34 const user = await UserModel.findOne({ activationLink });

35 if (!user) {

36 throw ApiError.BadRequest("Incorrect activation link");

37 }

38 user.isActivated = true;

39 await user.save();

40 }

41

42 async login(email, password) {

43 const user = await UserModel.findOne({ email });

44 if (!user) {

45 throw ApiError.BadRequest("User with this email was not found");

46 }

47 const isPassEquals = await bcrypt.compare(password, user.password);

48 if (!isPassEquals) {

49 throw ApiError.BadRequest("Invalid password");

50 }

51 const userDto = new UserDto(user);

52 const tokens = tokenService.generateTokens({ ...userDto });

53

54 await tokenService.saveToken(userDto.id, tokens.refreshToken);

55 return { ...tokens, user: userDto };

56 }

57

58 async logout(refreshToken) {

59 const token = await tokenService.removeToken(refreshToken);

60 return token;

61 }

62

63 async refresh(refreshToken) {

64 if (!refreshToken) {

65 throw ApiError.UnauthorizedError();

66 }

67 const userData = tokenService.validateRefreshToken(refreshToken);

68 const tokenFromDb = await tokenService.findToken(refreshToken);

69 if (!userData || !tokenFromDb) {

70 throw ApiError.UnauthorizedError();

71 }

72 const user = await UserModel.findById(userData.id);

73 const userDto = new UserDto(user);

74 const tokens = tokenService.generateTokens({ ...userDto });

75

76 await tokenService.saveToken(userDto.id, tokens.refreshToken);

77 return { ...tokens, user: userDto };

78 }

79

80 async getAllUsers() {

81 const users = await UserModel.find();

82 return users;

83 }

84 }

85

86 module.exports = new UserService();