Міністерство освіти і науки України Харківський національний університет радіоелектроніки

Кафедра програмної інженерії

Звіт

З лабораторної роботи №3

З дисципліни «Аналіз та рефакторинг коду»

на тему: «РОЗРОБКА БІЗНЕС-ЛОГІКИ ТА ФУНКЦІЙ АДМІНІСТРУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМНОЇ СИСТЕМИ »

Виконавець:

ст. гр. ПЗПІ-22-4 Попов Б. С.

Перевірив:

ст. викладач кафедри ПІ Сокорчук І. П.

Харків 2024

# МЕТА РОБОТИ

Метою роботи є розробка бізнес-логіки та функцій адміністрування для серверної частини програмної системи.

# ХІД РОБОТИ

У процесі виконання лабораторної роботи було реалізовано функціонал, що охоплює ключові аспекти адміністрування, безпеки та обробки даних. Основні інженерні рішення включають:

Управління користувачами

* Отримання списку всіх користувачів із бази даних.
* Видалення користувачів за унікальним ідентифікатором (ID).

Управління сповіщеннями

* Створення нових сповіщень та їх збереження у базу даних.
* Отримання списку всіх сповіщень.
* Видалення конкретного сповіщення за ID.

Робота з даними транспортних засобів

* Додавання нових даних про стан автомобілів (температура двигуна, тиск у шинах тощо).
* Отримання цих даних для аналітики та моніторингу.

Оновлення токенів

* Реалізація механізму оновлення токенів (refresh tokens) для підтримання активності сесії користувача.

Фрагменти програмного коду

Функції роботи зі сповіщеннями

1 router.post("/notification", adminController.addNotification);

2 router.get("/notifications", adminController.getNotifications);

3 router.delete("/notifications/:id", adminController.deleteNotification);

(Код маршрутів наведено у [додатку В](#_ДОДАТОК_В).)

Функції роботи з користувачами

Асинхронна функція для отримання користувачів:

1 async getUsers(req, res, next) {

2 try {

3 const users = await userService.getAllUsers();

4 return res.json(users);

5 } catch (e) {

6 next(e);

7 }

8 }

Асинхронна функція для видалення користувача:

1 async deleteUser(req, res, next) {

2 try {

3 const { id } = req.params;

4 await User.findByIdAndDelete(id);

5 res.json({ message: "User deleted successfully" });

6 } catch (e) {

7 next(e);

8 }

9 }

Функції роботи з даними транспортних засобів

1 router.get("/vehicle/data", adminController.getVehicleData);

2 router.post("/vehicle/data", adminController.addVehicleData);

(Код маршрутів наведено у [додатку Г](#_ДОДАТОК_Г).)

Реалізований функціонал дозволяє ефективно управляти даними системи AutoTrack, забезпечувати їхню актуальність і виконувати необхідні адміністративні операції.

# ВИСНОВКИ

Під час виконання лабораторної роботи для проекту AutoTrack було реалізовано низку функцій, які охоплюють основні аспекти адміністрування, безпеки та аналізу даних автомобіля. Зокрема, були розроблені рішення для управління користувачами, сповіщеннями та даними про стан транспортного засобу, що значно підвищує ефективність системи та зручність її використання.

Управління користувачами включає можливість отримання списку всіх користувачів з бази даних та видалення їх за унікальним ідентифікатором (ID). Це дає змогу адміністраторам системи ефективно керувати обліковими записами, забезпечуючи належний рівень безпеки та контролю доступу.

Управління сповіщеннями реалізовано через функціонал для створення, збереження та отримання сповіщень з бази даних, а також їх видалення за ID. Це дозволяє своєчасно інформувати користувачів про важливі події або зміни в системі, що підвищує ефективність взаємодії та комунікації.

Робота з даними про стан транспортного засобу включає додавання інформації про температуру, тиск у шинах, стан двигуна та GPS-координати. Це дозволяє проводити аналіз технічного стану автомобіля, надаючи корисну інформацію для ефективного обслуговування.

Оновлення токенів здійснюється через механізм рефрешу токенів для підтримки сесії користувача, що гарантує безперервність авторизації та захист даних користувачів.

Ці функції сприяють підвищенню надійності системи, покращенню користувацького досвіду та забезпечують ефективне адміністрування та обробку даних.

Посилання на ютуб - <https://youtu.be/Kf00ZbRub58>

# ДОДАТОК А

UML діаграма отримання всіх користувачів

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

Рисунок А.1 – UML отримання всіх користувачів

Діаграма описує процес отримання списку користувачів, починаючи від клієнта та закінчуючи поверненням результату з бази даних.

# ДОДАТОК Б

UML діаграми активності

Рисунок Б.1 – UML діаграма створення сповіщення

Діаграма на рис. Б.1 активності демонструє логіку перевірки та створення нового сповіщення в системі.

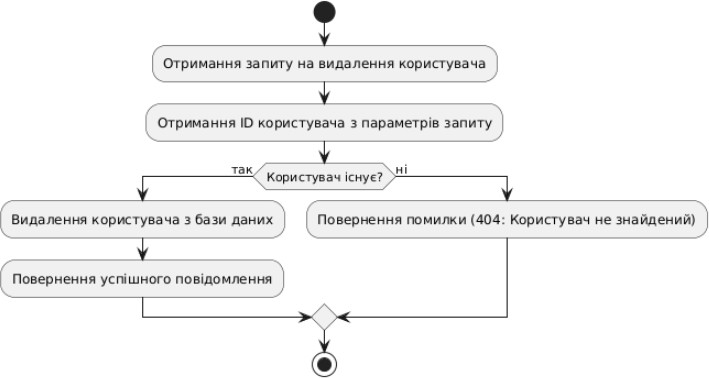


Рисунок Б.2 – UML діаграма видалення користувача

Діаграма на рис. Б.2 описує, як система опрацьовує запит на видалення користувача.

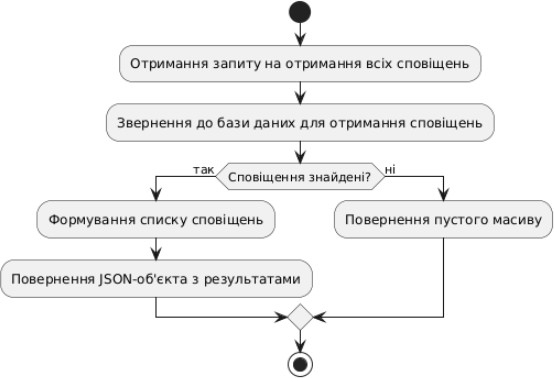


Рисунок Б.3 – UML діаграма отримання сповіщень

Діаграма на рис. Б.3 демонструє процес отримання всіх сповіщень для адміністратора.

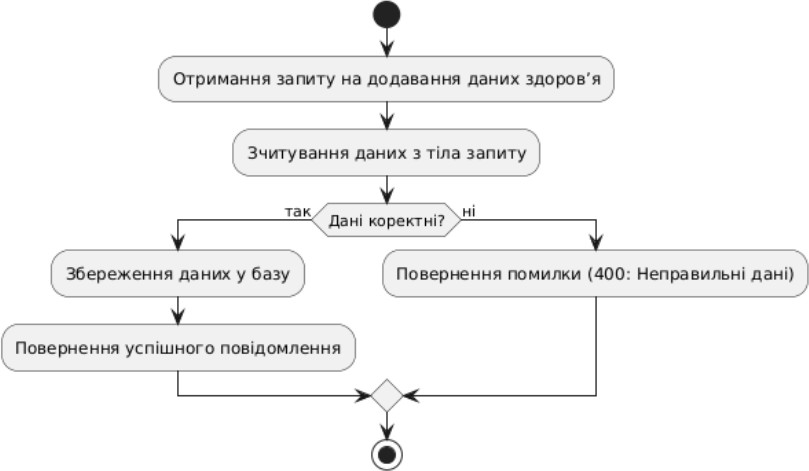


Рисунок Б.4 – UML діаграма додавання даних здоров’я користувача

Діаграма на рис. Б.4 показує, як система додає інформацію про здоров’я користувача у базу даних після валідації вхідних даних.



Рисунок Б.5 – UML діаграма видалення сповіщення

Діаграма на рис. Б.5 ілюструє, як система обробляє запит на видалення сповіщення.

# ДОДАТОК В

Код з програмної реалізації маршрутів для сповіщень (із серверної частини)

1. async createotification(req, res, next) {
2. try {
3. const errors = validationResult(req);
4. if (!errors.isEmpty()) {
5. return next(ApiError.BadRequest("Помка пр ваідації",

errors.array()));

6. }

1. const { email, password, role } = req.body;
2. res.status(201).json({ message: "otification created successfully." });
3. } catch (e) {
4. next(e);

11. }

12. }

1. async deleteotification(req, res, next) {
2. try {
3. const { id } = req.params;
4. await otification.findByIdAndDelete(id);
5. res.json({ message: "otification deleted successfully" });
6. } catch (e) {
7. res.status(500).json({ message: "Error deleting notification" });

20. }

21. }

# ДОДАТОК Г

Код з програмної реалізації маршрутів для роботи з даними про здоров’я (із серверної частини)

1. async addUserealthData(req, res, next) {
2. try {
3. const { userId, pulse, activityLevel, stressLevel, sleepours } = req.body;
4. if (!userId || !pulse || !activityLevel || !stressLevel || !sleepours)

{

1. return res.status(400).json({ error: "Invalid input data" });

6. }

1. res.status(201).json({ message: "Data successfully added" });
2. } catch (e) {
3. next(e);

10. }

11. }

1. async getUserelthData(req, res, next) {
2. const { userId } = req.query;
3. if (!userId) {
4. return res.status(400).json({ error: "userId is required" });

16. }

1. const data = {
2. userId,
3. pulse: 70,
4. activityLevel: 3,
5. stressLevel: 2,
6. sleepours: 7,

23. };

24. res.status(200).json(data);

25. }