

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»
Інститут прикладного системного аналізу
Кафедра системного проєктування

Лабораторна робота №4

з курсу *«Проектування інформаційних систем»*

на тему: «Розробка тестового мобільного додатку»

Виконав:
студент групи ДА-71
Стефура Олег

Мета: розробити тестовий мобільний додаток за темою індивідуального завдання.

Задача: вивчити принципи побудови мобільних додатків на прикладі системи Андроїд. Побудувати інтерфейс користувача та функціонал (частково) інформаційної системи за обраною індивідуальною темою. Частково реалізувати функціонал додатку (CRUD). Використовувати мобільну нативну платформу.

Завдання:

1. Створити інтерфейс користувача мобільного додатку інформаційної системи.
2. Розробити основний функціонал мобільного додатку CRUD.
3. Провести тестування мобільного додатку відповідно до SRS з л.р. 2.

Хід роботи

1. Графічний інтерфейс користувача

Наведемо вигляд основних вікон мобільного додатку:

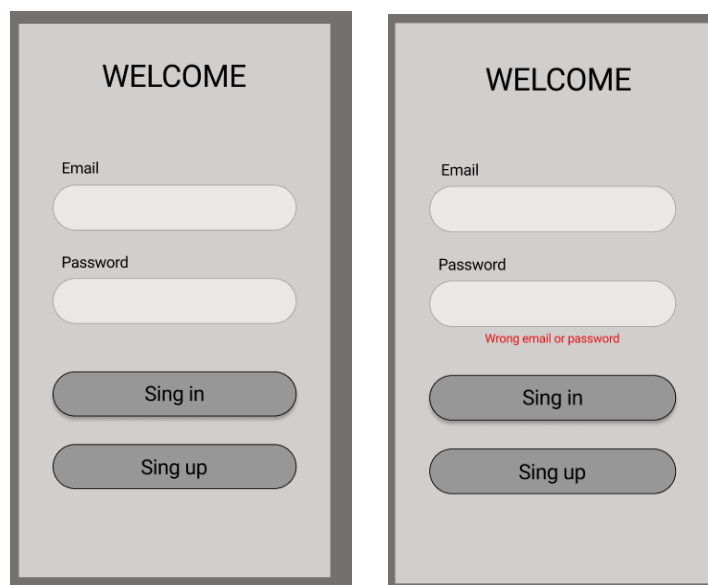


Рисунок 4.1 – Стартове вікно додатку

На рисунку 4.1 можна побачити, що бачить користувач при вході в додаток. Передбачувано, це вікно авторизації. Справа на рисунку – результат введення неправильних даних.

WELCOME

Email

Password

First Name

Last Name

Nickname

Sing up

Рисунок 4.2 – Вікно реєстрації нового користувача

На рисунку 4.2 зображено вікно реєстрації. До нього можна потрапити натиснувши кнопку «Sing up» на стартовому вікні.

Після процедури авторизації\реєстрації користувач попадає на головне вікно додатку (рис.4.3)

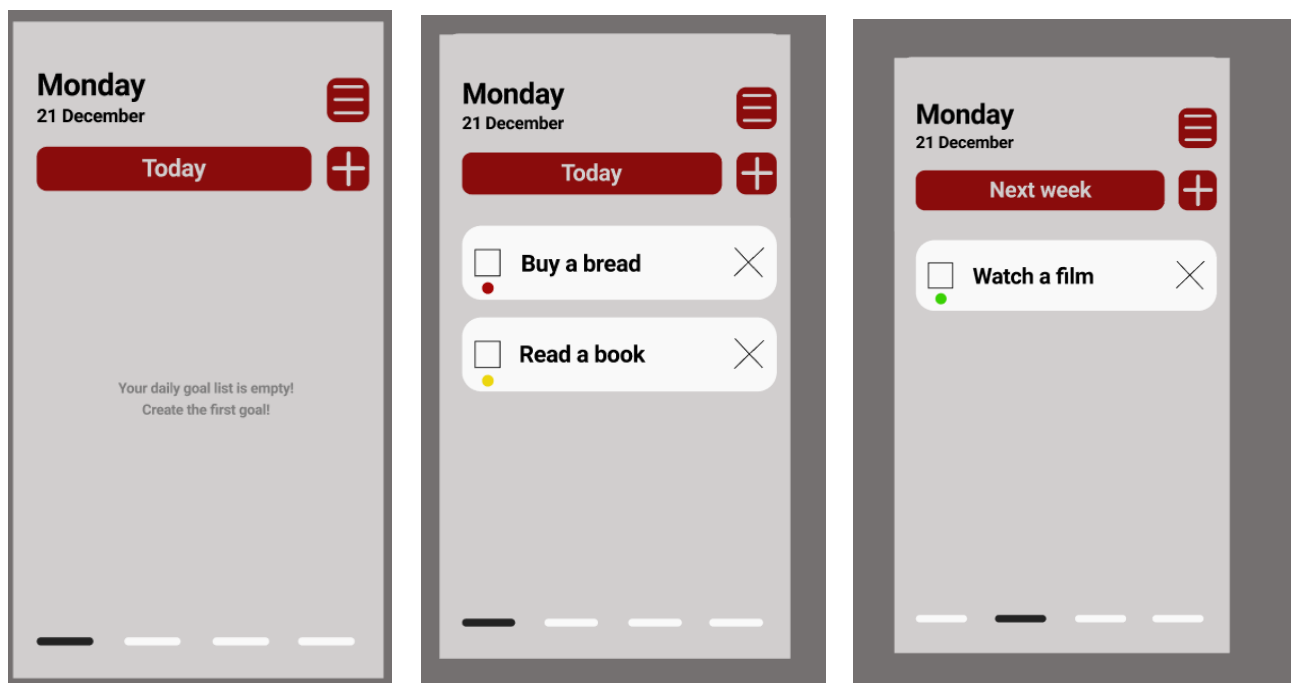


Рисунок 4.3 – Головне вікно

В залежності від того, чи є в користувача попередньо створені цілі, він побачить або порожній список, або список із наявними в ньому елементами.

Кожна ціль представлена блоком в списку із назвою, кнопками для видалення цілі, та відмітки її як виконану, а також із індикатором важливості (червоний, жовтий або зелений індикатор в залежності від того, чи ціль термінова, середньої важливості або не критична відповідно).

Всього існує чотири блоки цілей («сьогодні», «наступний тиждень», «наступний місяць», «без конкретної дати»). Перемикання між ними здійснюється свайпами на основі слайдера.

Створити нову ціль можна натиснувши кнопку «+» поряд із заголовком блоку (рис.4.4).

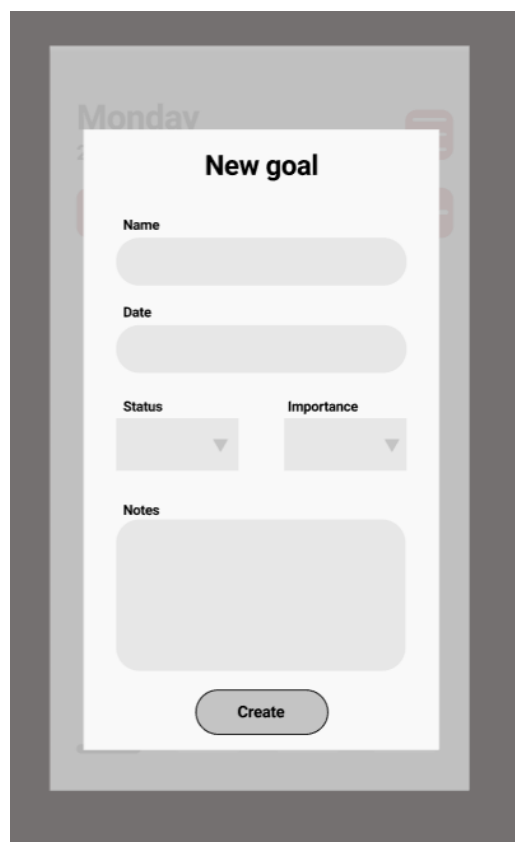
The image shows a mobile application interface for creating a new goal. A white modal form titled "New goal" is centered on the screen. It contains several input fields: "Name" (a rounded rectangle), "Date" (a rounded rectangle), "Status" (a dropdown menu with a downward arrow), "Importance" (a dropdown menu with a downward arrow), and "Notes" (a larger rounded rectangle). At the bottom of the form is a rounded button labeled "Create". The background is a blurred view of a calendar with the word "Monday" visible at the top.

Рисунок 4.4 – Створення нової цілі

При цьому необхідно вказати назву, дату, короткий опис (не обов'язково), та вибрати із відповідних списків значення статусу (приватна чи публічна), та важливості.

Натиснувши на одне із вікон цілей в списках, можна отримати більше інформації про них (рис.4.5).

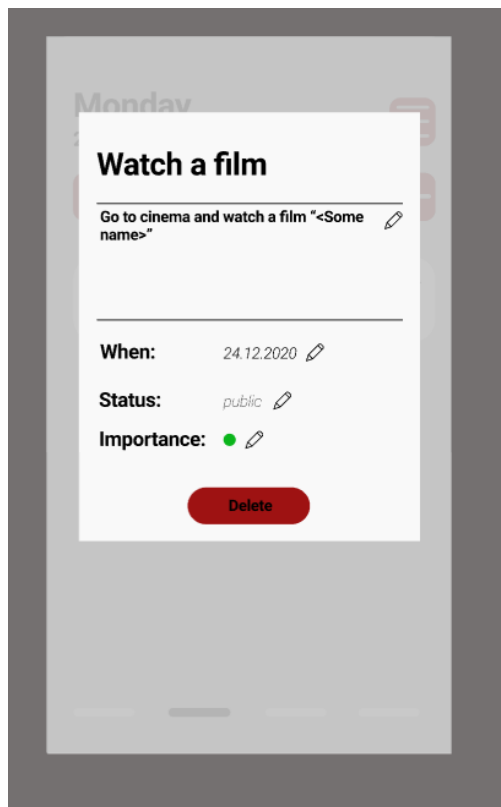


Рисунок 4.5 – Вікно цілі

В цьому вікні, помимо перегляду детальної інформації про ціль, можна змінити деякі її параметри (натиснувши на значок олівця).

Над кнопкою створення нової цілі на оловному вікні ще є кнопка меню, натиснувши яку з'являється список додаткових сторінок (рис.4.6).

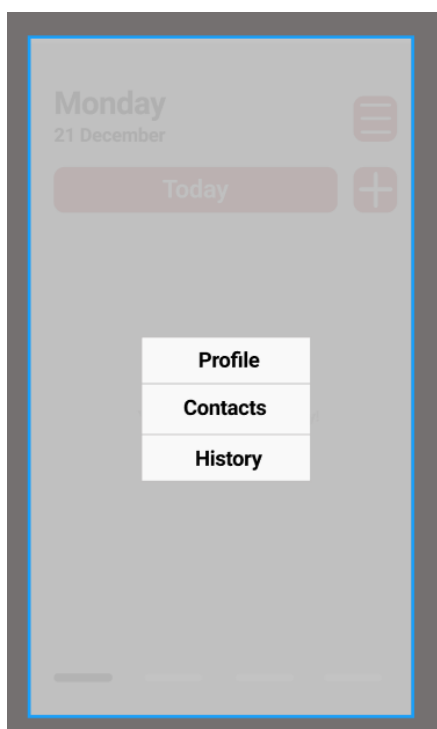


Рисунок 4.6 – Додаткове вікно навігації

Вибравши в ньому, наприклад, пункт «Profile», можна потрапити на вікно профіля користувача (рис.4.7).

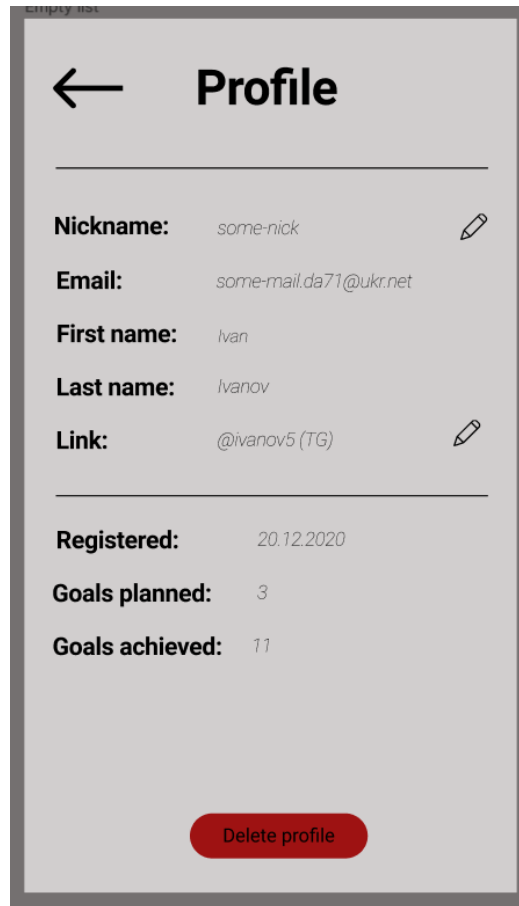


Рисунок 4.7 – Профіль користувача

Тут же можна змінити деякі дані, натиснувши на відповідні кнопки. За бажанням можна видалити профіль.

2. Реалізація функціоналу

З метою зробити систему універсальною (із можливістю її використання на різних видах пристроїв), основний функціонал реалізується у вигляді сервісу із REST API. Це API буде використовуватися різними «адаптерами» (веб-сайтами, мобільними додатками, планшетними додатками). Архітектура системи приведена на рисунку 4.8.

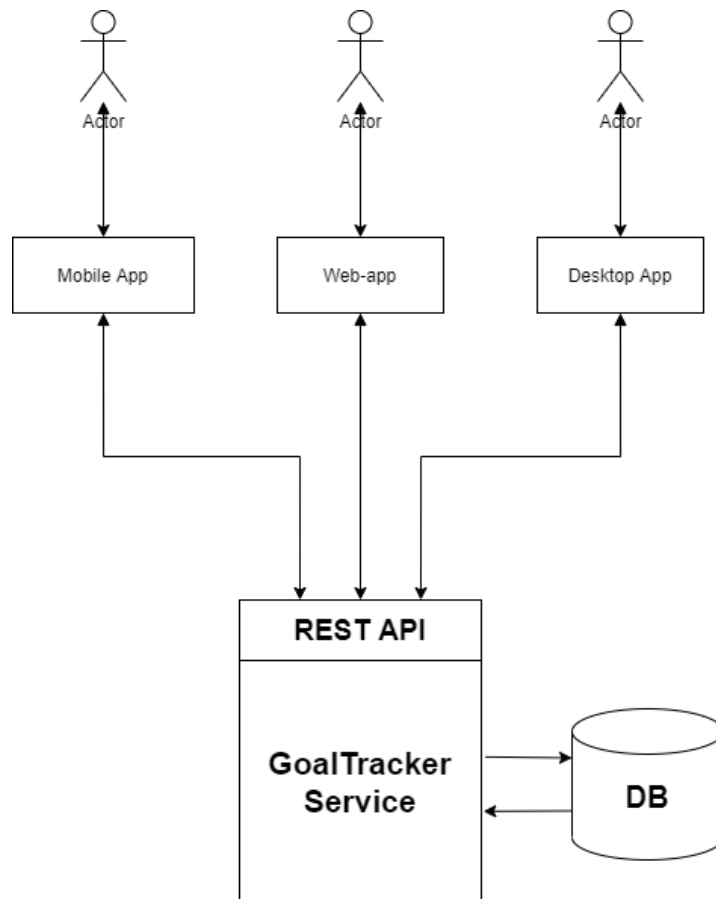


Рисунок 4.8 – Архітектура системи

Опишемо архітектуру, власне, сервісу.

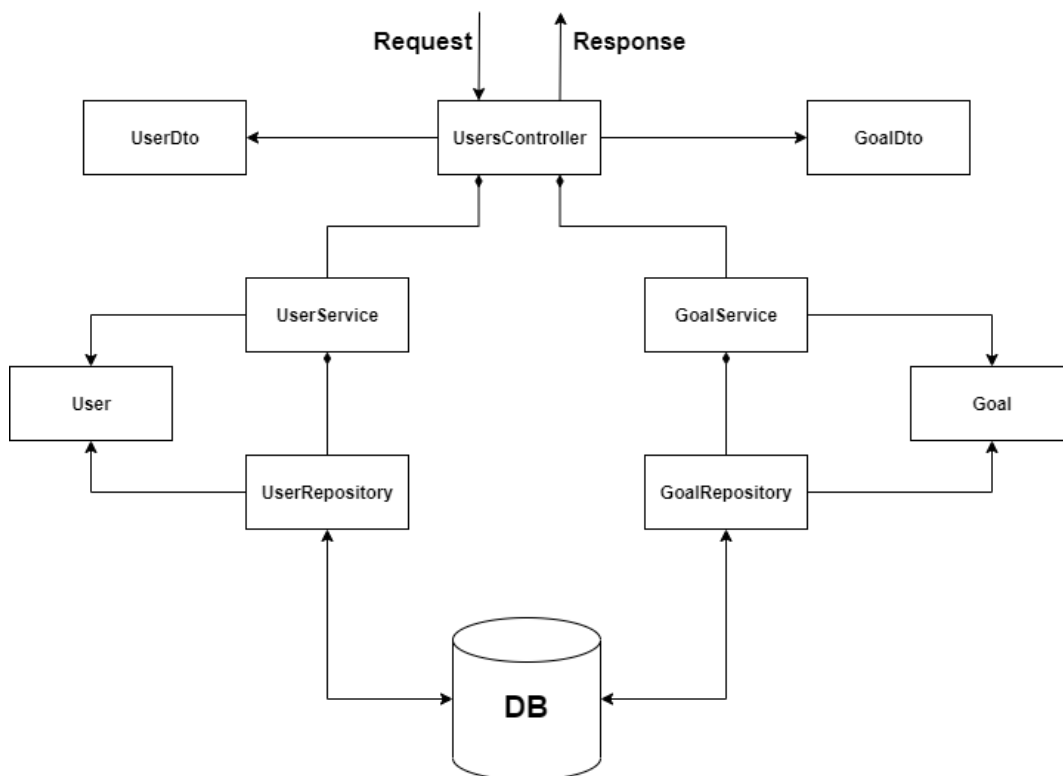


Рисунок 4.9 – Архітктура сервісу

Сервіс написано на технології Java (Spring Boot, Spring Data).

Тут:

- User/Goal – моделі даних із БД
- UserDto/GoalDto – DTO для обміну даними із зовнішніми застосунками
- UserRepository/GoalRepository – DAO-класи для маніпуляціями із моделями даних
- UserService/GoalService – класи, в яких зосереджена головна бізнес-логіка
- UsersController – клас, що відповідає за обробку запитів за URL REST API

Вигляд класу UsersController приведено нижче.

```
@RestController
@RequestMapping("/v1/users")
public class UsersController {
    @Autowired
    private UserService userService;

    @Autowired
    private GoalService goalService;

    @GetMapping
    @ResponseBody
    public List<UserDto> getAllUsers() {
        return
userService.getAllUsers().stream().map(this::mapUserToDto).collect(Collectors.to
List());
    }

    @GetMapping("/{id}")
    @ResponseBody
    public UserDto getUserById(@PathVariable Integer id) {
        return mapUserToDto(userService.getUserById(id));
    }

    @PostMapping("/{id}")
    @ResponseBody
    public UserDto createUser(@RequestBody UserDto userToCreate) {
        User createdUser = mapDtoToUser(userToCreate);
        createdUser = userService.addUser(createdUser);

        return mapUserToDto(createdUser);
    }

    // Other methods
}
```

Лістинг 4.1 – UsersController.java

Далі приведені моделі системи:


```

@Entity
@Table(name="users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "email", nullable = false, unique = true)
    private String email;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "nickname", nullable = false, unique = true)
    private String nickname;

    @Column(name = "link", unique = true)
    private String link;

    @OneToMany(mappedBy = "user")
    private Set<Goal> goals = new HashSet<>();

    // Getters and Setters
}

```

Лістинг 4.2 – Моедль «Користувач»

```

@Entity
@Table(name="goals")
public class Goal {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "name", nullable = false)
    private String name;

    @Column(name = "date", nullable = false)
    private String date;

    @Column(name = "status", nullable = false)
    private String status;

    @Column(name = "importance", nullable = false)
    private String importance;

    @Column(name = "notes")
    private String notes;

    @Column(name = "finihsed")
    private Boolean finished;

    public Boolean getFinished() {
        return finished;
    }

    public void setFinished(Boolean finished) {
        this.finished = finished;
    }

    @ManyToOne
    @JoinColumn(name = "user_id")

```

```
private User user;  
  
// Getters and Setters  
}
```

Лістинг 4.3 – Модель «Ціль»

Увесь код сервісу можна дослідити за посиланням:

<https://github.com/olegSt4/goal-tracker>

Він реалізує наступне API:

GET [200] /users – список всіх користувачів в системі

GET [200, 404] /users/{id} – користувач з конкретним ІД

POST [200] /users + {body} – створення нового користувача

PUT [200] /users/{id} – оновлення даних конкретного користувача

DELETE [200] /users/{id} – видалення конкретного користувача

GET [200, 404] /users/{u_id}/goals – увесь список цілей конкретного користувача

GET [200, 404] /users/{u_id}/goals/{g_id} – конкретна ціль конкретного користувача

POST [200] users/{u_id}/goals – створення нової цілі

PUT [200] users/{u_id}/goals/{g_id} – оновлення конкретної цілі

DELETE [200, 404] /users/{u_id}/goals/{g_id} – видалення конкретної цілі

3. Тестування

Протестуємо створений сервіс згідно теблиці тестування, приведений в л.р.2. за допомогою інструменту Postman:

1) Реєстрація в системі

Ця дія еквівалентна створенню нового користувача в системі. Сформуємо та виконаємо запит:

POST localhost:8085/v1/users

Params Authorization Headers **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded **raw** binary GraphQL JSON

```

1 {
2   "email": "some.user2010@ukr.net",
3   "firstName": "Some",
4   "lastName": "User",
5   "nickname": "some_user1123"
6 }

```

Body Cookies Headers (5) Test Results Status: 200 OK 1

Pretty Raw Preview Visualize JSON

```

1 {
2   "id": 3,
3   "email": "some.user2010@ukr.net",
4   "firstName": "Some",
5   "lastName": "User",
6   "nickname": "some_user1123",
7   "link": null
8 }

```

Рисунок 4.10 - Створення нового користувача

2) Авторизація в системі

Для авторизації необхідно переконатися, що користувач із вказаним email в системі існує. Здійнимо запит:

GET localhost:8085/v1/users

Params Authorization Headers **Body** Pre-request Script Tests Settings

none **form-data** x-www-form-urlencoded raw binary GraphQL

	KEY	VALUE
<input checked="" type="checkbox"/>	email	some.user2010@ukr.net

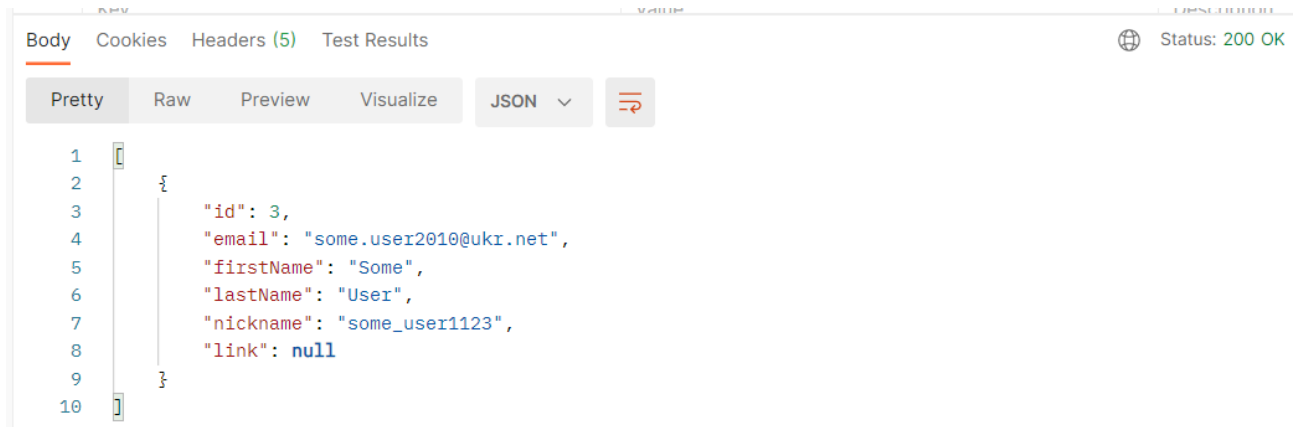


Рисунок 4.11 – Пошук користувача за його email

3) Видалення акаунта

Видалення акаунта рівнозначне видаленню користувача із системи.

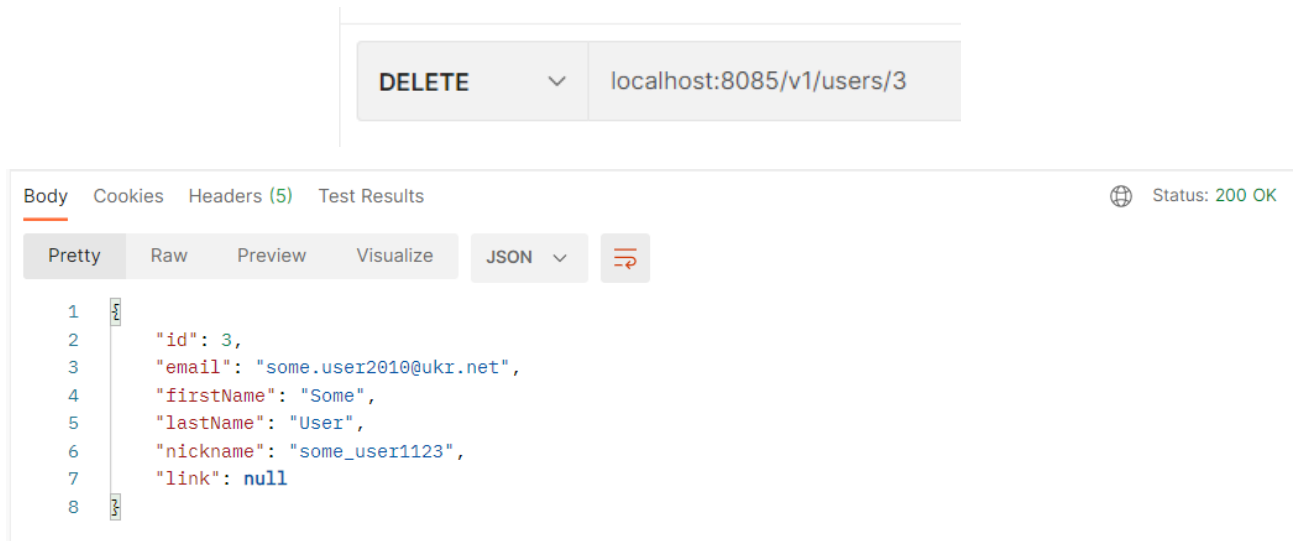


Рисунок 4.13 – Видалення користувача

4) Зміна нікнейму

Зміна нікнейму (чи іншого параметра профілю) – це оновлення даних про користувача в системі.

Нехай є наступний користувач:

```

{
  "id": 5,
  "email": "joe.doe.2020@gmail.com",
  "firstName": "Joe",
  "lastName": "Doe",
  "nickname": "joe.d.1100",
  "link": null
}

```

Рисунок 4.16 Існуючий користувач

Змінимо нікнейм:

The screenshot shows a REST client interface with a PUT request to `localhost:8085/v1/users/5`. The request body is a JSON object with the following fields: `"id": 4`, `"email": "joe.doe.2020@gmail.com"`, `"firstName": "Joe"`, `"lastName": "Doe"`, and `"nickname": "NewNick"`. The response status is 200 OK. The response body is displayed in a 'Pretty' JSON format, showing the updated user object with `"id": 5` and `"nickname": "NewNick"`.

```

1 {
2   "id": 4,
3   "email": "joe.doe.2020@gmail.com",
4   "firstName": "Joe",
5   "lastName": "Doe",
6   "nickname": "NewNick"
7 }

```

```

1 {
2   "id": 5,
3   "email": "joe.doe.2020@gmail.com",
4   "firstName": "Joe",
5   "lastName": "Doe",
6   "nickname": "NewNick",
7   "link": null
8 }

```

Рисунок 4.17 Зміна нікнейму

5) Додавання цілі

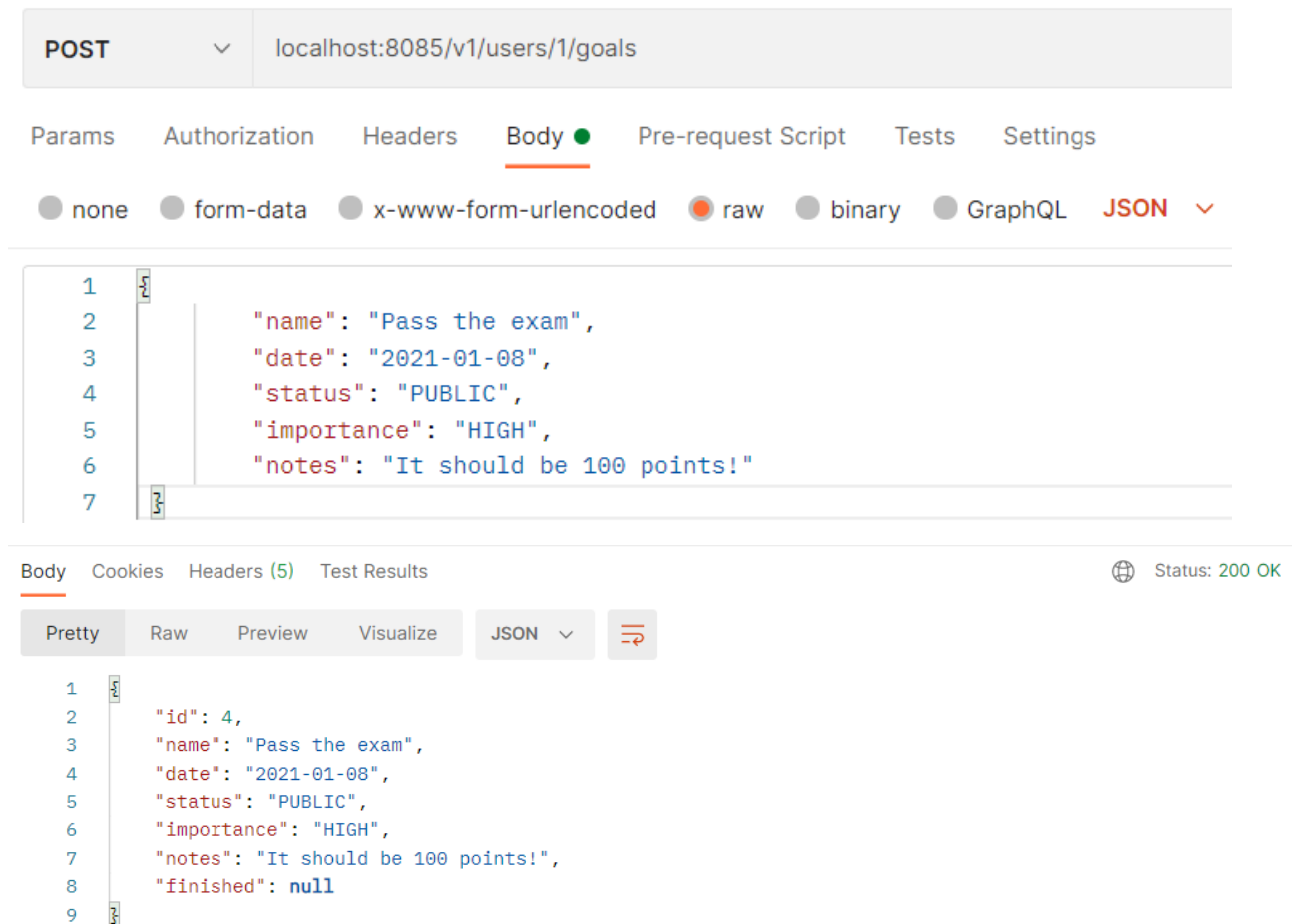


Рисунок 4.18 – Додавання цілі

б) Зміна «дедлайну» цілі

Зміна дедлайну, як і зміна будь якого іншого поля – це оновлення вже існуючого об’єкту в системі.

Нехай є ціль:

```
{
  "id": 1,
  "name": "Watch a film",
  "date": "2020-12-28",
  "status": "PUBLIC",
  "importance": "LOW",
  "notes": "Watch a movie in the cinema",
  "finished": false
},
```

Рисунок 4.19 – Існуюча ціль

PUT localhost:8085/v1/users/1/goals/1

Params Authorization Headers **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded **raw** binary GraphQL **JSON**

```
1 {
2   "id": 1,
3   "name": "Watch a film",
4   "date": "2020-12-28",
5   "status": "PUBLIC",
6   "importance": "LOW",
7   "notes": "Watch a movie in the cinema"
8 }
```

Body Cookies Headers (5) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "name": "Watch a film",
4   "date": "2020-12-30",
5   "status": "PUBLIC",
6   "importance": "LOW",
7   "notes": "Watch a movie in the cinema",
8   "finished": false
9 }
```

Рисунок 4.20 – Редагування цілі

7) Видалення цілі

DELETE localhost:8085/v1/users/1/goals/1

Body Cookies Headers (5) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "name": "Watch a film",
4   "date": "2020-12-28",
5   "status": "PUBLIC",
6   "importance": "LOW",
7   "notes": "Watch a movie in the cinema",
8   "finished": false
9 }
```

Рисунок 4.21 - Видалення цілі

Підсумуємо, що всі тести завершилися успішно.

Висновки: в роботі продемонстровано інтерфейс користувача, який задовольняє основним вимогам, вказаним в SRS (а саме забезпечує необхідний функціонал). Важливо відмітити, що в цілому інтерфейс є інтуїтивно-зрозумілим та не містить в собі багато непотрібних функцій, що могли б заплутати користувача. Це також одна із вимог до додатку, вказана в SRS. Приведено архітектуру додатку та системи в цілому. Було прийнято рішення зупинитись на розробці загального REST API, що дає змогу зробити систему кросплатформеною.

REST API реалізовано за допомогою інструментів Java (Spring Boot, Spring Data) та успішно протестовано згідно таблиці тестування SRS.