Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем

Розробка закритої соціальної мережі з неблокуючим вводом-виводом(nio)

Текстова частина до курсової роботи
за спеціальністю «Прикладна математика»

Керівник курсової роботи
ст. викладач Борозенний С. О.

_____
(підпис)
“____” _____ 2021 р.
Виконав студент 3 р. н.
Тихий Р. В.
“____” _____ 2022 р.

Київ – 2021

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем

доцент, к.ф.-м.н.

О. П. Жежерун

_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Тихому Роману Вікторовичу 3-го курсу факультету інформатики

ТЕМА Розробка закритої соціальної мережі з неблокуючим вводом-виводом(nio)

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

1 Аналіз предметної області

2 Проектування архітектури додатку

3 Огляд фреймворків та бібліотек

4 Дослідження реактивних потоків на базі бібліотеки Project Reactor для розробки веб застосунку

5 Розробка додатку

Висновки

Список літератури

Дата видачі „___” _____ 2021 р. Керівник _____

Завдання отримав _____

Тема: Розробка закритої соціальної мережі з неблокуючим вводом-виводом(nio)

Календарний план виконання роботи:

№ п/п	Назва етапу	Термін виконання	Примітка
1.	Отримання завдання на курсову роботу.	14.10.2021	
2.	Аналіз предметної області.	Листопад 2021	
3.	Проектування архітектури додатку	Грудень 2021	
3.	Огляд фреймворків та бібліотек	Лютий 2022	
4.	Розробка додатку	Березень 2022	
5.	Написання текстової частини роботи.	Квітень 2022	
6.	Здача роботи на перевірку на плагіат	07.06.2022	

Студент Тихий Р. В.

Керівник Борозенний О. С.

“_____”_____

Зміст

Анотація

Курсова робота присвячена створенню соціальної мережі із закритим доступом із елементами реактивного програмування. Для його розробки були використані мови програмування Java, JavaScript, інтегроване середовище розробки IntelliJ IDEA, система автоматичного збирання Gradle(Java), менеджер залежностей Node Package Manager, фреймворк Spring Boot, мова запитів і

маніпуляції даними GraphQl, document-oriented database MongoDB, фреймворк React, бібліотека React компонентів Material UI та багато іншого.

Основні функції системи: аунтефікація та авторизація за допомогою корпоративної пошти Outlook або за допомогою ім’я та пароля, створення постів із текстовим наповненням, медіа наповненням та посиланнями на інші ресурси, перегляд найбільш популярних постів, вподобання постів та можливість залишати під ними коментарі, отримання моментальних сповіщень про вподобання ваших постів або при нових коментарях та інше.

Ключові слова: Spring Boot, Spring WebFlux, Reactor Netty, Project Reactor, React, Service, Controller.

Вступ

Мало хто може уявити свій день без хоч одного заходу в соціальні мережі на день. Деякі з нас можуть скролити їх годинами. Але що якщо система починає повільно працювати та не задовольняє потреби користувача у швидкості обробки та подачі інформації? Або користувач незадоволений користувацьким інтерфейсом? Або користувач витрачає забагато свого часу на реєстрацію або вхід до свого акаунту?

Метою курсової роботи є створення сучасної соціальної мережі у браузерному середовищі. Вона повинна бути максимально доступна для всіх людей, мати сучасний, приємний, інтуїтивно зрозумілий графічний інтерфейс із можливістю змінювати користувацьку тему, оптимізувати мережеві запити та виконувати поставлені перед нею задачі. Основними завданнями даного дослідження є:

- аналіз предметної області та створення функціональних вимог;
- вибір архітектури та її реалізація в самому додатку;
- огляд фреймворків та обґрунтування їх виборів;
- дослідження реактивних потоків на базі бібліотеки Project Reactor для розробки веб застосунку;
- створення застосунку.

Об’єктом дослідження є процес розробки застосунку з використанням реактивних потоків.

Предметом дослідження є сам застосунок та реактивні потоки.

У цій роботі висвітлюється тематика соціальних мереж. Застосунок створений згідно функціональних вимог і відповідає критеріям сучасного застосунку, а також використовує правильні архітектурні рішення під час його проектування.

Робота складається з п’яти розділів.

Перший розділ призначений для аналізу предметної області і визначення основних функціональних вимог, згідно яких буде будуватися функціонал застосунку.

Другий розділ присвячений дослідженню основних методів проектування, вибору архітектурного шаблону та його детального опису.

У третьому розділі обґрунтовується вибір фреймворків та інших засобів.

Четвертий розділ присвячений опису можливостей реактивних потоків в області веб розробки клієнт-серверних застосунків.

В останньому розділі наводиться реалізація графічного інтерфейсу користувача, опис структури класів та їх взаємодія один з одним на рівні сервісів та контролерів. Також приділяється увага використаним при розробці додатковим бібліотекам.

У результаті виконання дослідження створено програмний продукт, який призначений для реалізації соціальних потреб людини онлайн, тобто гарне проводження часу у всесвітній мережі.

1 Аналіз предметної області

1.1 Характеристика предметної області

Обраною темою є створення сучасного застосунку у веб середовищі(браузері) за допомогою фреймворку з неблокуючим вводом-виводом Spring WebFlux. Додаток складається з із серверної(Backend) та клієнтської(Frontend) частин: серверна частина обробляє мережеві запити надіслані з клієнтської частини, керує веб сесіями та безпекою, підключається до інших мікросервісів, зберігає, оновлює та видаляє записи із віддаленої бази даних; клієнтська частина надає графічний інтерфейс користувачу та керує локальним станом застосунку, а також займається оптимізацією даних(кешуванням). Аунтефікація та авторизація користувача можлива двома способами. Перший: за допомогою корпоративної пошти Outlook(oAuth 2.0); другий: за допомогою логіну(ім’я користувача) та паролю. В обох випадках користувачу повертається JWT – JSON Web Token, який містить в собі основні данні про чинного користувача та зберігається на стороні клієнта. Якщо аунтефікація відбувається за допомогою корпоративної пошти вперше, то акаунт користувача буде створено автоматично та збережено у базу даних, за згоди користувача до доступу певних даних його Microsoft акаунту. Клієнтська частина застосунку складається з React роутера який займається перенаправленням за посиланнями, та містить в собі такі шляхи:

```
<Routes>
    <Route path="/login" element={<SignInSide/>}/>
    <Route path="/" element={<Layout unread={unreadNotifications}/>}>
        <Route path='/notifications'
               element={<Notifications notifications={notifications}
                                       wipeUnread={setUnreadNotifications}/>} />
        <Route path='/top' element={<TopPosts/>}/>
        <Route path="/user">
            <Route path=':id' element={<UserPage/>}/>
        </Route>
        <Route path="/post">
            <Route path=":id" element={<PostComments />} />
        </Route>
        <Route path="/" element={<Main/>}/>
        <Route path="*" element={<NotFound/>}/>
    </Route>
</Routes>
```

Рис. 1.1 Дерево клієнтського роутера

1.2 Функціональні вимоги

Беручи за основу предметну область і мету розробки проекту, зазначаю функціональні вимоги, поставлені переді мною перед розробкою функціоналу соціальної мережі:

- Можливість створювати пости із простим текстом, медіа-файлами та посиланнями які містять в собі інформацію.
- Можливість аунтефікації та авторизації користувача двома способами: за допомогою корпоративної пошти Outlook(oAuth 2.0); за допомогою логіну(ім’я користувача) та паролю.
- Можливість вподобати пост .
- Можливість прокоментувати пост.
- Можливість змінити користувацьку тему.
- Можливість переглядати пости впорядковані за часом або кількістю вподобань.
- Можливість переглядати свій профіль та профілі інших користувачів

- Адаптація під різні розміри екрану.

2 Аналіз предметної області

2.1 Характеристика предметної області

Архітектура дає змогу розділити проект на 4 шари, а саме:

- Шар конфігурації (англ. Configuration layer).

  Цей шар відповідальний за створення Java Beans, підключення до баз даних, налаштування конфігурація модуль Spring Boot, налаштуванням безпеки та інше.

- Шар домену (англ. Domain layer).

  Шар, в якому зосереджена та представлена бізнес логіка додатку, є проміжним рівнем між шаром даних і шаром представлення.

- Шар представлення або шар контролера(англ. Presentation layer or Controller layer).

  Шар який займається розпізнаванням запитів, як HTTP так і Websocket, та скеруванням їх до шару сервісу.

- Шар сервісу(англ. Service layer).

  Шар який займається діями пов’язаними із сутностями та їх зміною.

  Наприклад запит до бази даних або до сервісу іншого домену, запит до іншого застосунку, генерація сповіщень та інше.

Розділення проекту на чотири шари використовувалось при розробці мого додатку. Це дає можливість розбити класи по різним шарам, що допомагає краще орієнтуватись, підтримувати, тестувати та повторно використовувати написаний код.

3 Вибір фреймворку та його обґрунтування

3.1 Обґрунтування вибору фреймворку Spring Boot

Мною було обрано фреймворк Spring Boot за його простоту, модульність, швидкість розробки, перспективність, підтримку оновлень та виправлень багів і головним чином за мій досвід роботи з ним, а також за можливість попрацювати із реактивним потоками на прикладі бібліотеки Project Reactor, яка є основою HTTP серверу Netty, який є сервером за замовчуванням в конфігурації Spring WebFlux.

3.2 Переваги та недоліки фреймворку Spring Boot

Серед переваг можна виділити велику кількість Spring фреймворків які можна використовувати, адже Spring насправді є фреймворком фреймворків, тобто кожен з них може використовуватись і окремо, але разом вони полегшують швидкість розробки, що без сумнівів є великим плюсом. Spring Boot не новий фреймворк, до того ж популярний, тому з його вивченням не буде проблем, враховуючи що він дуже добре задокументований. Реактивні потоки не зроблять роботу вашої програми швидше, але вони зроблять так, щоб ресурси вашої машини використовувались максимально ефективно, тому Spring WebFlux чудово підходить для написання високонавантажених систем, завдяки потоковій обробці із неблокованим зворотним тиском.

Серед недоліків саме Spring WebFlux є його складність. Реактивні потоки не є інтуїтивно зрозумілі, адже це перехід від синхронного програмування до асинхронного. Я б описав свій досвід використання цього фреймворку так: «Прості речі стають складними, а складні легкими». Для кращого розуміння варто знати Java Stream API, адже саме на ньому і базується Project Reactor. Мінімальна версія Java для використання реактивних потоків 8.

3.3 Обґрунтування вибору фреймворку React

React — відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими стикаються в розробці односторінкових застосунків.

Завдяки відкритому коду React зміг стати один із найпопулярніших Frontend фреймворків, який постійно підтримується розробниками та спільнотою. Фреймворк є нескладним у вивченні завдяки хорошій документації та підтримці спільноти.

3.4 Переваги та недоліки фреймворку React

Серед переваг є однозначно популярність та простота. Багато сторонніх бібліотек пишуть свої React компоненти та хуки, що дає великий вибір та гнучкість. Можливість відлагодження прямо у браузері за допомогою розширення суттєво покращує досвід та швидкість розробки. Також існує велика кількість бібліотек React компонентів, наприклад Material UI або Bootstrap React.

Серед недоліків я не міг би виділити нічого, що мені спало б на думку під час написання практичної частини дослідження.

4. Дослідження реактивних потоків на базі бібліотеки Project Reactor для розробки веб застосунку

4.1 Як реактивна парадигма програмування може покращити утилізацію використання ресурсів

Є два способи покращити ефективність програми:

1. Збільшувати кількість потоків та кількість ресурсів(«заліза»)
2. Використовувати максимально ефективно наявну кількість потоків та ресурсів

У звичайних Java застосунках код є блокуючим, тобто коли наприклад програма звертається до бази даних або іншого не синхронного джерела, то тред очікує відповіді, тобто простоює тим самим витрачаючи час і не використовуючи усі можливі ресурси програми.

Тому паралелізація не є панацею від усіх бід, і тут на виручку приходить реактивне програмування. Пишучи асинхронний неблокуючий код, ми дозволяємо потоку перемикатись на інші процеси задіяючи ті самі ресурси

пізніше перемикаючись до первинного процесу, коли закінчиться асинхронна обробка.

Reactor представляє реактивні типи, які можна компонувати та які реалізують Publisher, а також великий словник реактивних операторів до них: Flux і Mono. Об'єкт Flux представляє реактивну послідовність з 0..N елементів, тоді як об'єкт Mono представляє результат з одним значенням або порожнім (0..1).

4.2 Використання Processor та Sink у веб застосунках

Процесор — це особливий вид Publisher, який також є Subscriber. Спочатку вони були задумані як можливе уявлення проміжного кроку, який потім можна було б спільно використовувати між реалізаціями Reactive Streams. Однак у Reactor такі кроки скоріше представлені операторами, які є Publisher.

У Reactor Sink — це клас, який дозволяє безпечно запускати сигнали вручну. Він може бути пов’язаний з підпискою (зсередини оператора) або повністю автономний.

Одним із найкращих варіантів використання Processor та Sink є реалізація сповіщень.

```java
@Component
public class SubscriptionsSinks {


    @Bean
    public Sinks.Many<PostNUser> postCreatedSink(){
        return Sinks.many().multicast().onBackpressureBuffer(Integer.MAX_VALUE, autoCancel: false);
    }

    ConcurrentHashMap<String, Sinks.Many<SubscriptionEvent>> concurrentHashMap = new ConcurrentHashMap<>();

    @Bean
    public ConcurrentHashMap<String, Sinks.Many<SubscriptionEvent>> sinksHashMap() { return concurrentHashMap; }

    @Bean
    RouterFunction<ServerResponse> staticResourceRouter(){
        return RouterFunctions.resources( pattern: "/**", new ClassPathResource("static/"));
    }

}
```

Рис. 4.2.1 Створень Bean-ів з Sink-ами

```java
    @SubscriptionMapping
    public Flux<PostNUser> postCreated() { return postCreatedSink.asFlux().share(); }

    @SubscriptionMapping
    public Flux<SubscriptionEvent> personalSubscription(Authentication authentication) {
        User user = (User) authentication.getPrincipal();
        sinksHashMap.putIfAbsent(user.getId(), personalSink());
        return sinksHashMap.get(user.getId()).asFlux();
    }
```

Рис. 4.2.2 Кінцеві точки підписок GraphQl за протоколом Websocket

Для точки postCreated() Sink є спільним для усіх підписників за допомогою оператора .share(), що дозволяє ще краще використовувати ресурси.

Для точки personalSubscription() Sink є персональним і для кожного користувача є приватним. Зберігається він у потокобезпечній колекції

```
ConcurrentHashMap<String, Sinks.Many<SubscriptionEvent>>
```

```java
@Service
public class SubscriptionService {

    private final Sinks.Many<PostNUser> postCreatedSink;
    private final ConcurrentHashMap<String, Sinks.Many<SubscriptionEvent>> sinksHashMap;

    public SubscriptionService(Sinks.Many<PostNUser> postCreatedSink, ConcurrentHashMap<String,
            Sinks.Many<SubscriptionEvent>> sinksHashMap) {
        this.postCreatedSink = postCreatedSink;
        this.sinksHashMap = sinksHashMap;
    }

    public void postCreated(User user, Post post) { postCreatedSink.tryEmitNext(new PostNUser(user, post)); }

    public void commentAdded(User user, Post post, String id) {
        sinksHashMap.get(id)
                .tryEmitNext(new SubscriptionEvent(EventTypes.COMMENT_ADDED, new PostNUser(user, post)));
    }

    public void postLiked(User user, Post post) {
        sinksHashMap.get(post.getAuthorId())
                .tryEmitNext(new SubscriptionEvent(EventTypes.POST_LIKED, new PostNUser(user, post)));
    }
}
```

Рис 4.2.3 Сервіс підписок який «штовхає» в Sink подію

5. Структура додатку

5.1 Графічний інтерфейс користувача

Усі шляхи як на сервері так і на клієнті доступні лише аунтефікованим користувачам, виняток – “/login”. При спробі зайти на захищену сторінку користувач буде автоматично перенаправлений на сторінку входу.

За допомогою Sass міксинів адаптація вебсторінки під різні розміри вікна та користувацькі теми стає набагато простішою.

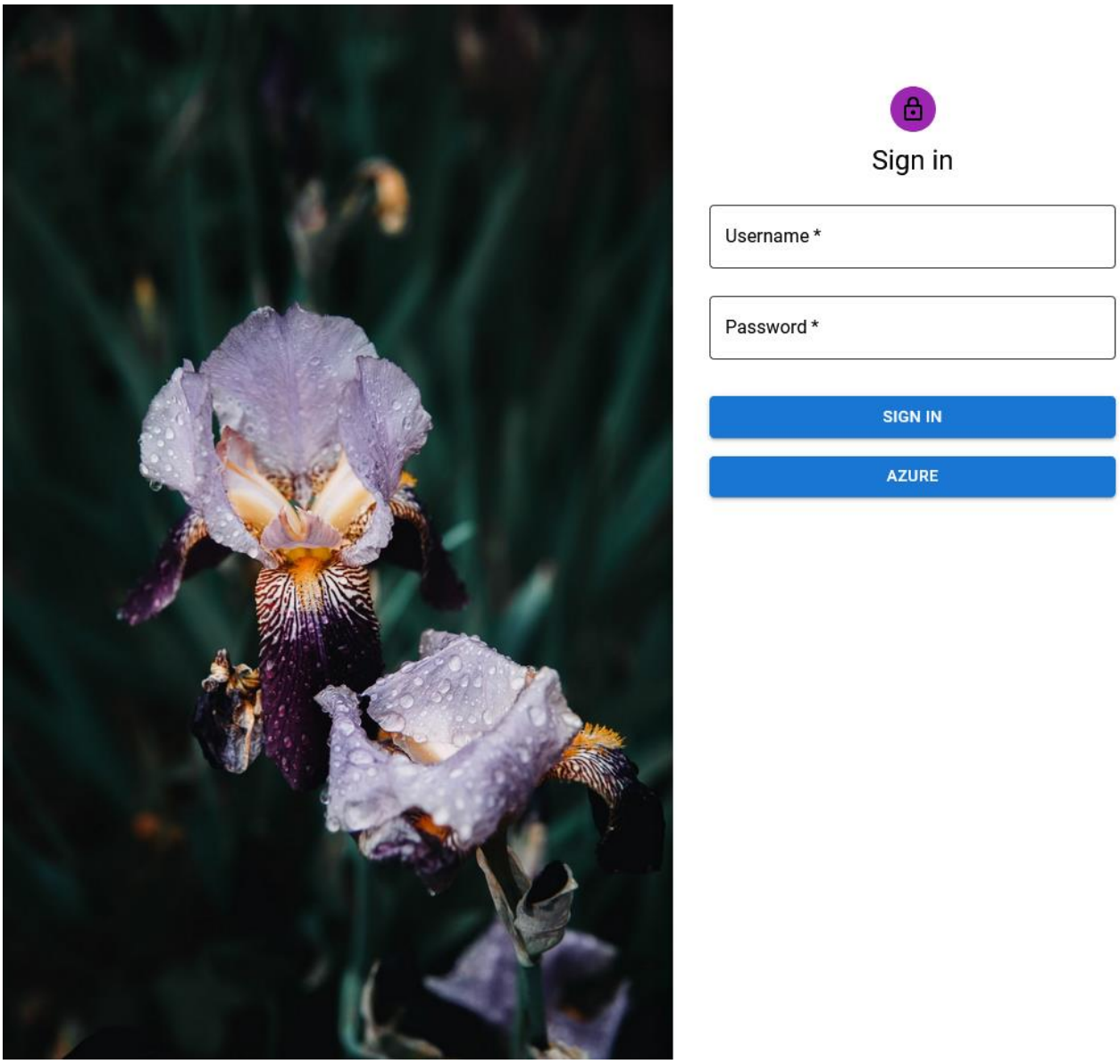

Рис. 5.1.1 Вигляд сторінки входу на екранах більші за середні

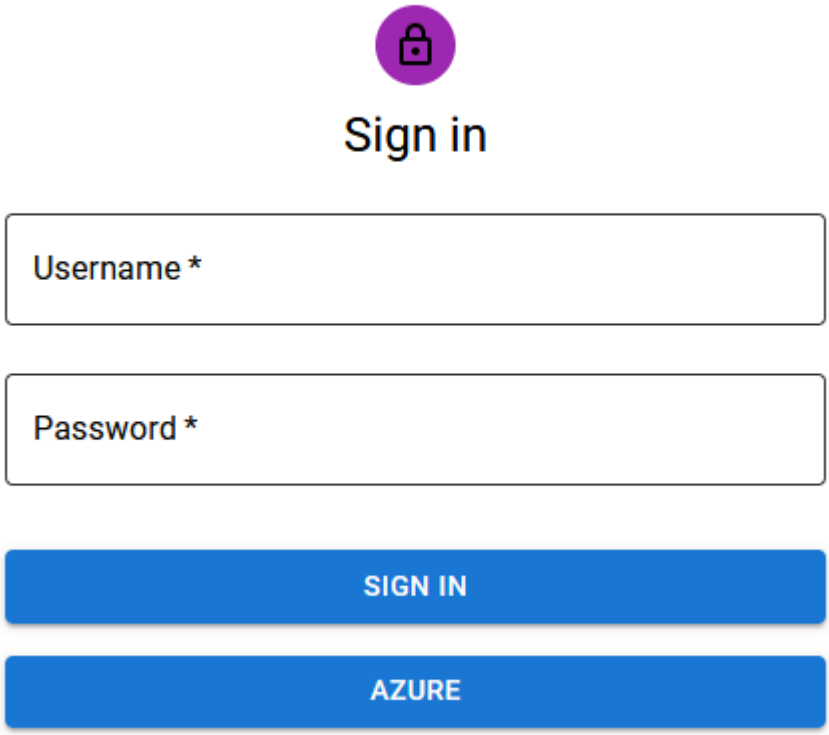

Рис. 5.1.2 Вигляд сторінки входу на малих екранах

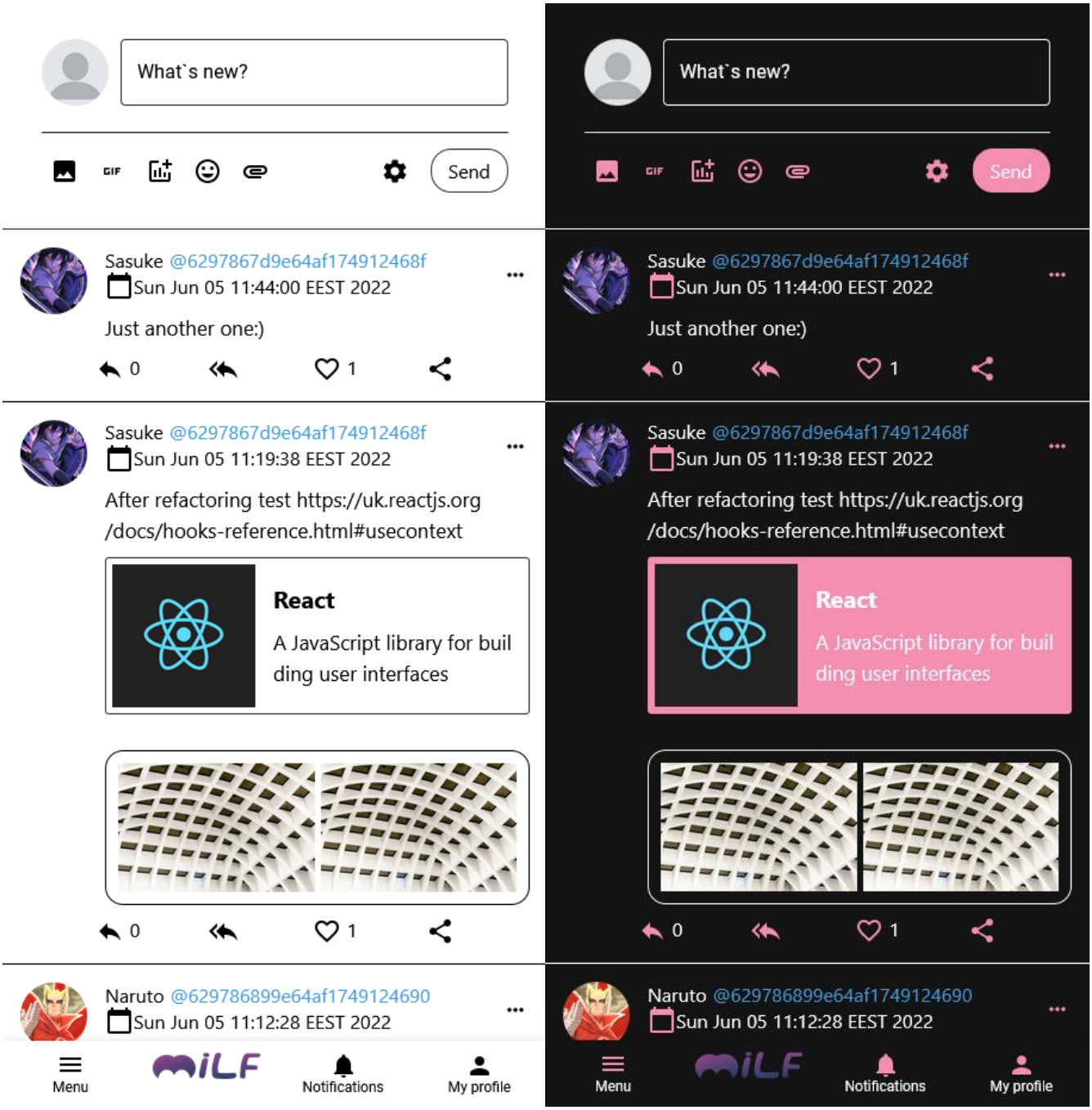

Рис. 5.1.3 Вигляд головних сторінок(світла та темні темни) на малих екранах(менші за 480 пікселів)

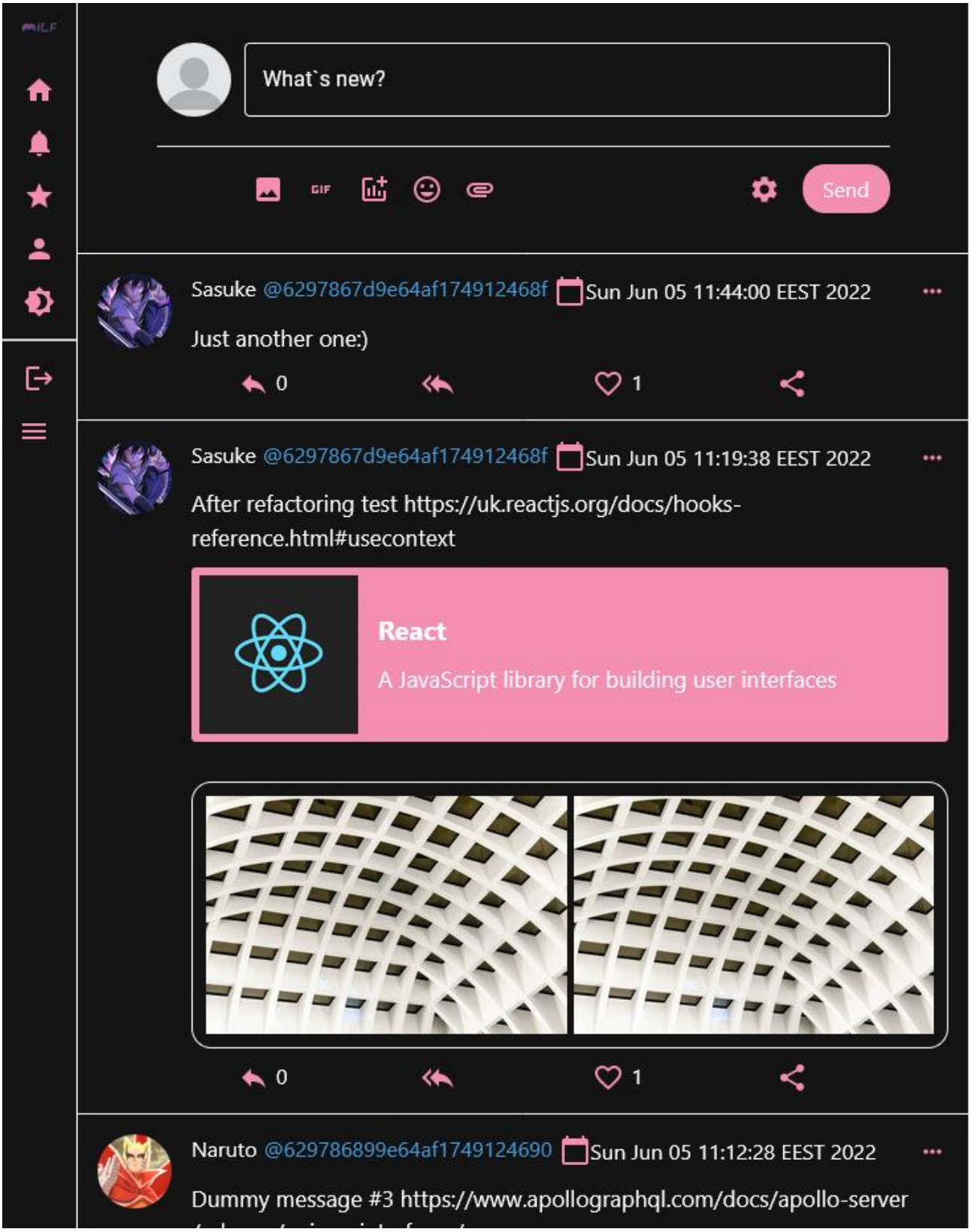

Рис. 5.1.4 Вигляд головної сторінки середніх екранах(від 480 до 780 пікселів)

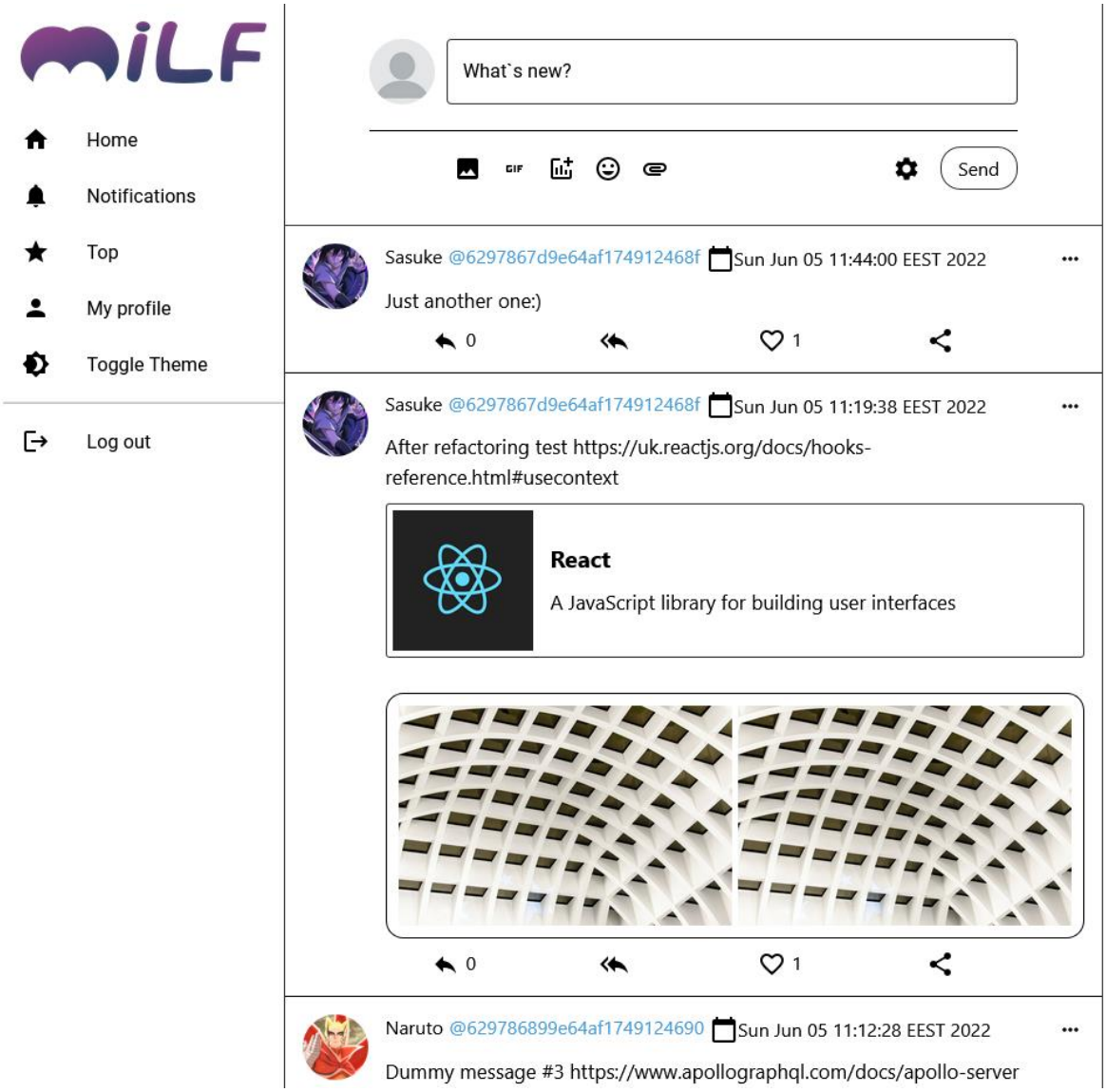

Рис. 5.1.5 Вигляд головної сторінки на великих екранах(понад 780 пікселів)

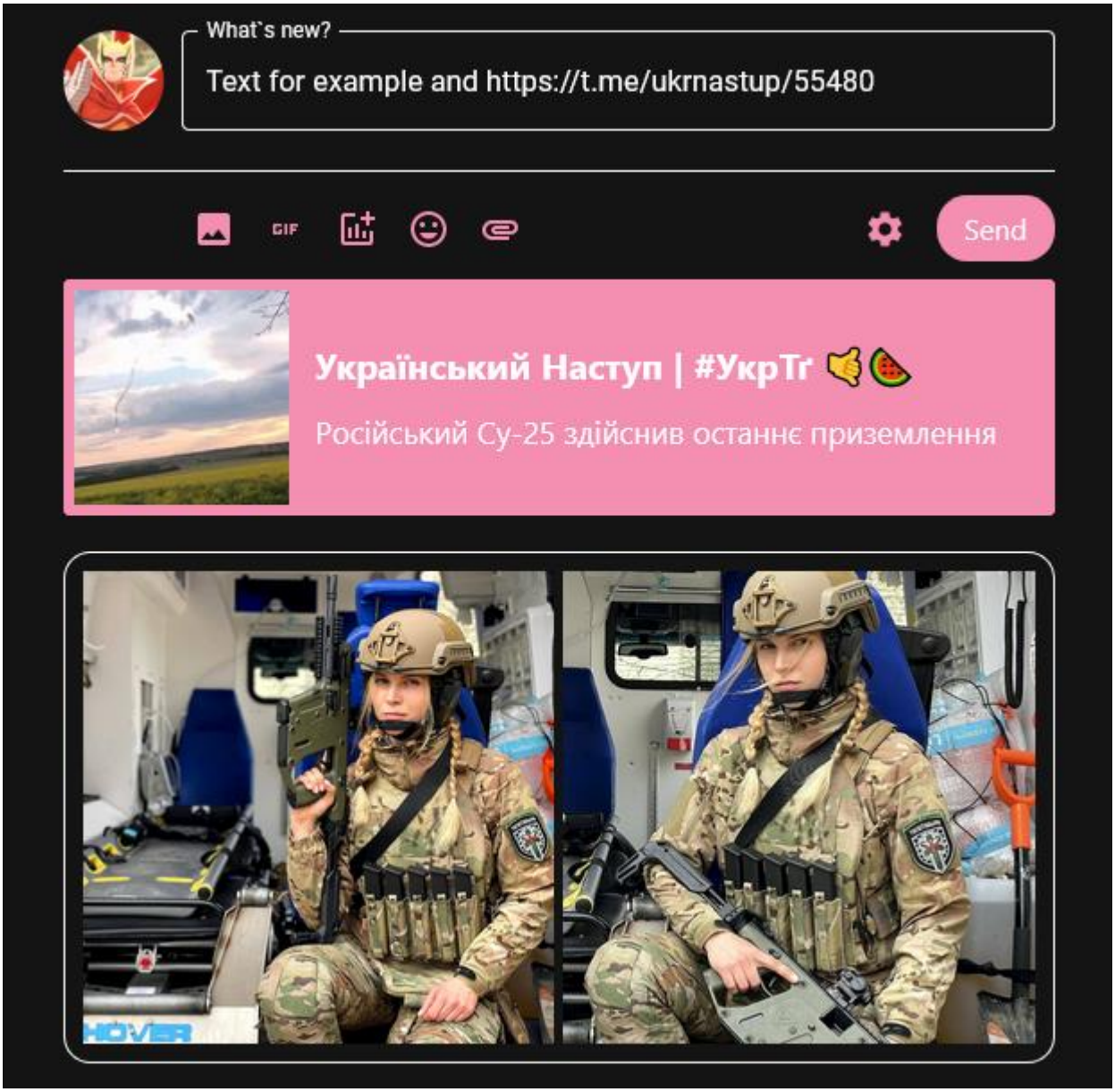

Рис. 5.1.6 Вигляд заповненої форми нового посту із використанням OpenGraph Protocol

Рис 5.1.7 Вигляд поля сповіщень коли вони є так коли відсутні

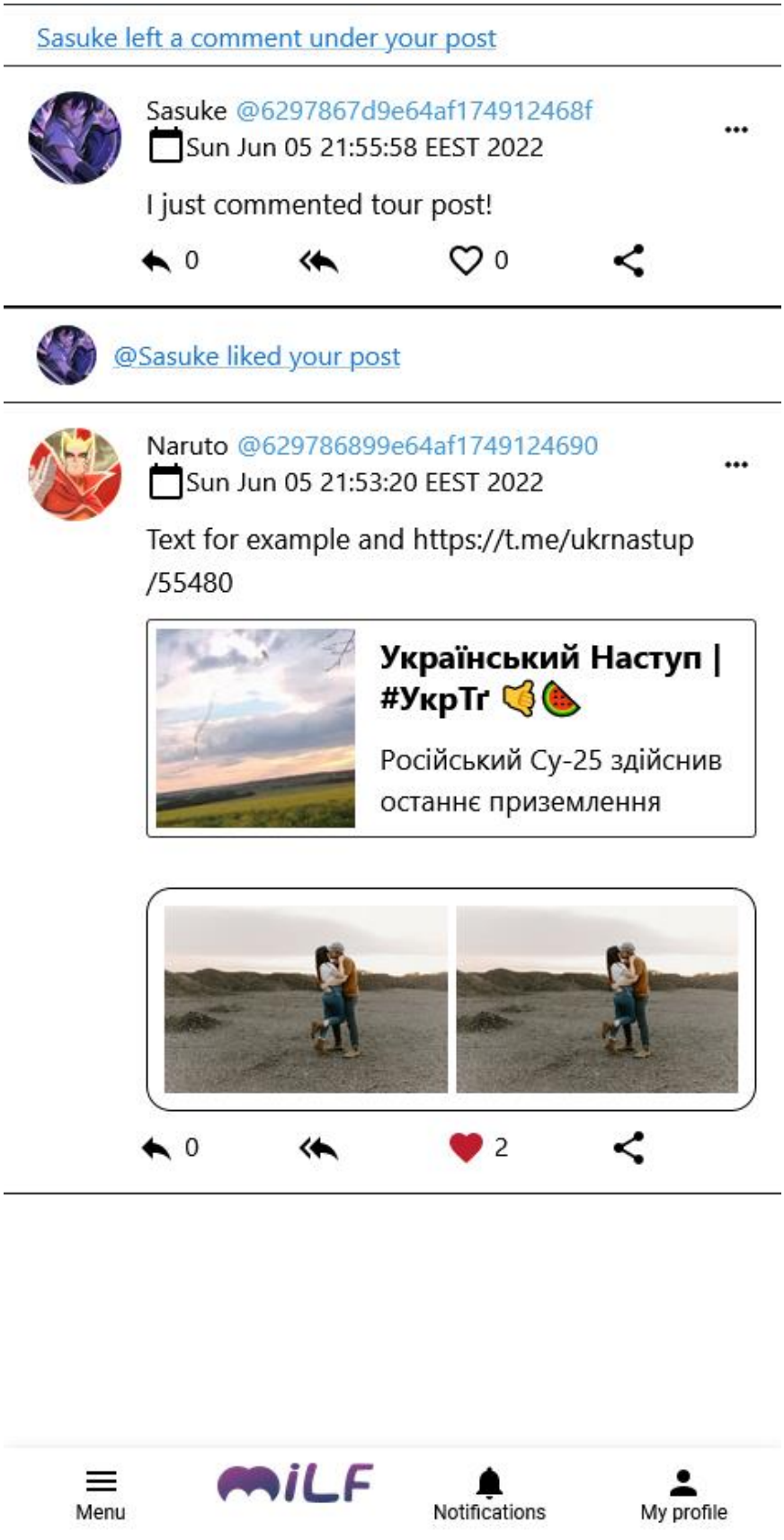

Рис. 5.1.8 Вигляд сторінки сповіщень

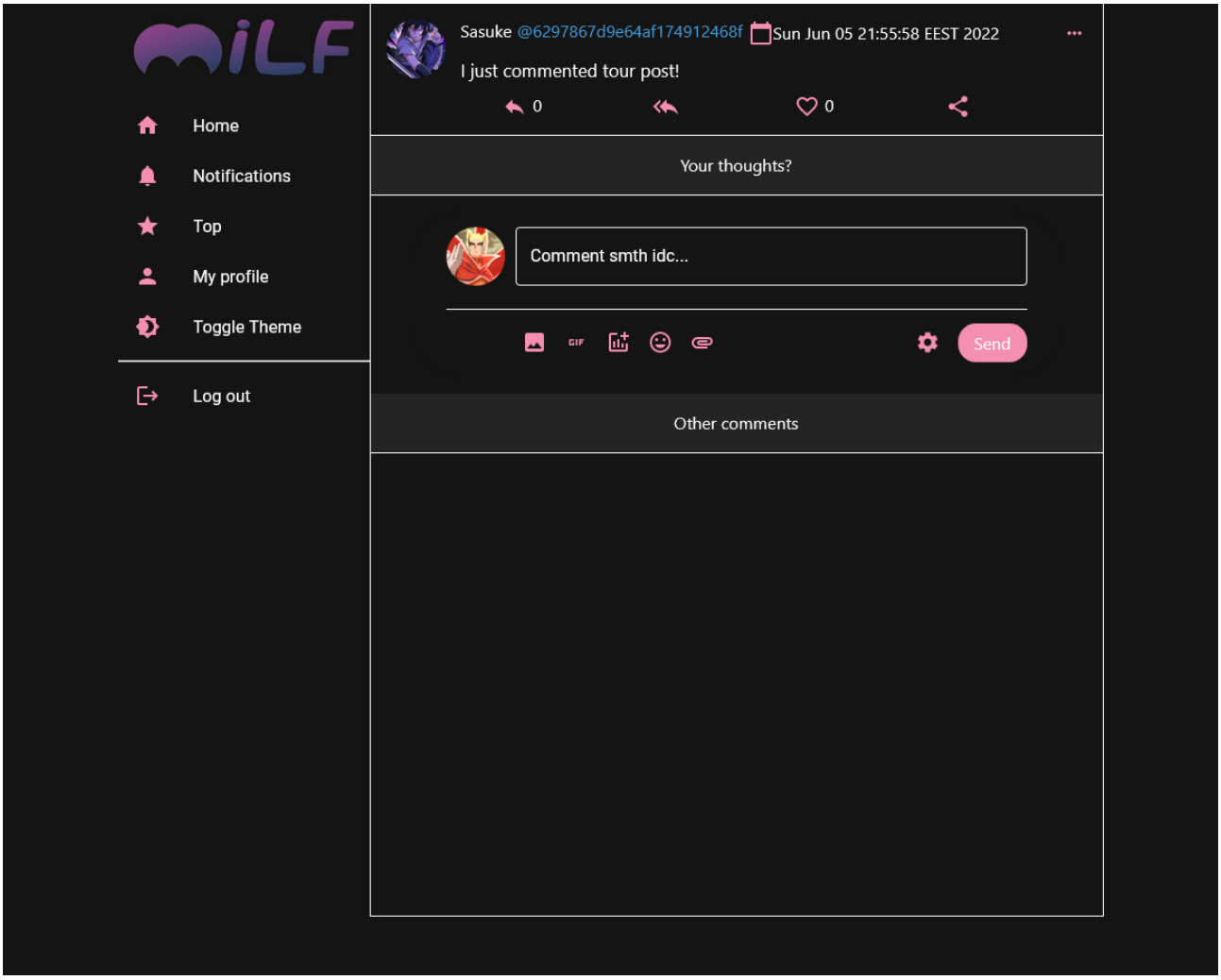

Рис. 5.1.9 Вигляд сторінки поста із можливістю прокоментувати його

5.2 Шар домену

Для уникнення шаблонного коду використовується Lombok.

1. Post.class

```java
@Data
@AllArgsConstructor
@Document(collection = "post")
@EqualsAndHashCode(of = "id")
public class Post {

    @Id
    private String id;
    private String text;
    @Min(0)
    private Integer likesCount;
    @Min(0)
    private Integer commentsCount;
    private Set<PostLike> likes;
    private String isCommentTo;
    private Set<String> commentIds;
    private Set<String> fileLocations;
    private String authorId;
    private MetaDto metaDto;
    private Date date;

    public Post() {
        this.id = null;
        this.likesCount = 0;
        this.commentsCount = 0;
        this.likes = new HashSet<>();
        this.commentIds = new HashSet<>();
        this.fileLocations = new HashSet<>();
        this.isCommentTo = null;
        this.date = new Date();
    }

}
```

2. PostLike.class

```java
@EqualsAndHashCode(of = "userId")
public class PostLike {
    private String userId;
    private Date creationTime;

    public PostLike() {
    }

    public String getUserId() { return userId; }

    public void setUserId(String userId) { this.userId = userId; }

    public Date getCreationTime() { return creationTime; }

    public void setCreationTime(Date creationTime) { this.creationTime = creationTime; }

    public PostLike(String userId) {
        this.userId = userId;
        this.creationTime = new Date();
    }
}
```

3. User.class

```java
@Document(collection = "usr")
@Data
@AllArgsConstructor
public class User implements UserDetails {

    @Id
    private String id;

    private String username;

    @JsonIgnore
    private String password;

    private String avatar;

    private String description;

    private Set<Role> roles;

    public User(){

    }

    public User(String id, String username){
        this.id = id;
        this.username = username;
        this.avatar = null;
        this.description = null;
        this.password = null;
        this.roles = Set.of(Role.USER);
    }

    @JsonIgnore
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return this.roles.stream().map(authority ->
                new SimpleGrantedAuthority(authority.name()))
                .collect(Collectors.toList());
    }

    @JsonIgnore
    @Override
    public boolean isAccountNonExpired() { return true; }

    @JsonIgnore
    @Override
    public boolean isAccountNonLocked() { return true; }

    @JsonIgnore
    @Override
    public boolean isCredentialsNonExpired() { return true; }

    @JsonIgnore
    @Override
    public boolean isEnabled() { return true; }
}
```

4. Role.enum

```
public enum Role {
    USER, ADMIN
}
```

5. DTO класи – Data transfer object один із шаблонів проєктування, який використовують для передачі даних між підсистемами програми.
    1. MetaDto.class – використовується для передачі даних отриманих із мікросервісу який парсить HTML meta теги

```
@Data
@AllArgsConstructor
@RequiredArgsConstructor
public class MetaDto {
    private String title;
    private String url;
    private String description;
    private String cover;
}
```

    2. PostNUser.class – містить в собі User та Post

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class PostNUser {
    private User user;
    private Post post;
}
```

5.3 Сервіси застосунку

Всього у застосунку присутні 6 сервісів:

```
1. CustomOidcUserService extends
  OidcReactiveOAuth2UserService
2. PostService
3. SubscriptionService
4. UserDetailsService implements
  ReactiveUserDetailsService
5. UserService
6. WebClientService
```

1. CustomOidcUserService – клас який розширює OidcReactiveOAuth2UserService.

```
@Override
public Mono<OidcUser> loadUser(OidcUserRequest userRequest) throws OAuth2AuthenticationException {
    Mono<OidcUser> oidcUser = super.loadUser(userRequest);
    log.info(oidcUser.toString());
    return oidcUser.flatMap(this::updateUser);
}

private Mono<OidcUser> updateUser(OidcUser oidcUser) {
    User user = new User(oidcUser.getClaim("oid"), oidcUser.getFullName());
    return userService.getOneById(user.getId())
            .switchIfEmpty(Mono.defer(() -> userService.saveUser(user)))
            .flatMap(user1 -> Mono.just(oidcUser));
}
```

Містить один @Override метод loadUser – метод який використовується для завантаження OidcUser. В методі updateUser створюється потрібний нам User та зберігається в базу даних якщо його там ще немає.

2. PostService – сервіс для взаємодії із Post

Методи цього сервісу:

1. getTopPostsWithUsers() – повертає послідовність PostNUser відсортованих за кількістю вподобань
2. findAllPostWithUsers() - повертає послідовність PostNUser
3. findAllPosts() – повертає послідовність Post

4. getPostWithUser() – повертає PostNUser
5. getCommentsToPostWithUsers() – повертає послідовність PostNUser які є коментарями до певного посту
6. likeOrDislikePost() – метод з анотацією @Transactional, що означає що всі запити до бази даних будуть виконані в одній сесії, що гарантує узгодженість(англ. Consistency). В залежності від того чи вже вподобаний пост чинним користувачем буде додане нове вподобання або видалене старе. Повертає Post.
7. saveOne() - зберігає в базі даних Post. Повертає Post.
8. saveOneTransactional() – зберігає в базі даних Post, але при цьому якщо є коментарем до іншого посту то потребує бути в транзакції для здійснення додаткових запитів до бази даних для надсилання сповіщення автору посту. Повертає Post.
9. validateAndSaveMessage() – перевіряє кількість файлів, їх тип, та у разі якщо все проходить валідацію зберігає файли на жорсткий диск. Проводить валідацію тексту посту, посилання якщо воно є, та здійснює запит до webClientService для отримання мета-даних сайту. Викликає saveOneTransactional() у кінці ланцюжка. Повертає Post.
10. getAllByAuthorId() – повертає послідовність постів автором яких є змінна передана в метод
11. getOne() – повертає пост за змінною в параметрах методу

3. SubscriptionService – сервіс який «штовхає» сповіщення до потрібного Sink-a

```
public void postCreated(User user, Post post) { postCreatedSink.tryEmitNext(new PostNUser(user, post)); }

public void commentAdded(User user, Post post, String id) {
    sinksHashMap.get(id)
            .tryEmitNext(new SubscriptionEvent(EventTypes.COMMENT_ADDED, new PostNUser(user, post)));
}

public void postLiked(User user, Post post) {
    sinksHashMap.get(post.getAuthorId())
            .tryEmitNext(new SubscriptionEvent(EventTypes.POST_LIKED, new PostNUser(user, post)));
}
```

4. UserDetailsService implements ReactiveUserDetailsService – сервіс який необхідний для реалізації власного менеджера аунтефікації.

```
@Override
public Mono<UserDetails> findByUsername(String username) {
    return userRepository.findByUsername(username).cast(UserDetails.class);
}
```

5. UserService – сервіс для взаємодії із User
    1. getAllUsersWithIds() – повертає послідовність User у яких співпадає айді переданий у параметрах. Використовується у пост сервісі.
    2. getOneById() – повертає User із вказаним айді.
    3. saveUser() – зберігає User у базі даних. Повертає User.
6. WebClientService – сервіс для обміну даних з іншими застосунками за допомогою HTTP протоколу.

```
WebClient client = WebClient.create("http://localhost:8081");


public Mono<MetaDto> getMetaDtoMono(String url) {
    return client.get() WebClientRequestHeadersUriSpec<...>
            .uri( uri: "/parse-html?url=" + url) capture of ?
            .accept(MediaType.APPLICATION_JSON)
            .retrieve() WebClientResponseSpec
            .bodyToMono(MetaDto.class);
}
```

Метод повертає метадані із віддаленого ресурсу.

5.4 Шар конфігурації

5.4.1 SecurityConfiguration – конфігуруємо безпеку у застосунку.

Реалізуємо власний механізм аунтефікації та авторизації за допомогою JWT. Для цього створюємо AuthenticationManager та

SecurityContextRepository. За створення, валідацію та декодинг JWT відповідає JwtUtil.

Для того щоб надсилати JWT у разі успіху аунтефікації з використанням стандарту oAuth 2.0 та встановлювати потрібний нам редірект використовуємо власний RedirectServerAuthenticationSuccessHandler:

```
@Override
public Mono<Void> onAuthenticationSuccess(WebFilterExchange webFilterExchange, Authentication authentication) {

    ServerHttpResponse response = webFilterExchange.getExchange().getResponse();

    if (response.isCommitted()) {
        return Mono.empty();
    }

    DefaultOidcUser oidcUser = (DefaultOidcUser) authentication.getPrincipal();
    User user = new User(oidcUser.getClaim("oid"), oidcUser.getFullName());
    response.getHeaders().setLocation(URI.create("http://localhost:3000/login"));
    response.setRawStatusCode(302);


    response.addCookie(ResponseCookie.from( name: "token", jwtUtil.generateToken(user))
            .path("/")
            .maxAge(86400)
            .httpOnly(true).build());
```

Створюємо власний AuthenticationWebFilter:

```
private AuthenticationWebFilter jwtAuthenticationFilter() {
    AuthenticationWebFilter jwtAuthenticationFilter = new AuthenticationWebFilter(authenticationManager);
    jwtAuthenticationFilter.setSecurityContextRepository(securityContextRepository);
    jwtAuthenticationFilter.setServerAuthenticationConverter(authConverter);
    return jwtAuthenticationFilter;
}
```

Конфігуруємо SecurityWebFilterChain:

```
@Bean
public SecurityWebFilterChain securityWebFilterChain(ServerHttpSecurity httpSecurity) {

    return httpSecurity
            .csrf().disable() ServerHttpSecurity
            .formLogin().disable()
            .httpBasic().disable()
            .logout() ServerHttpSecurity.LogoutSpec
            .logoutUrl("/log-out")
            .logoutSuccessHandler((exchange, authentication) -> {
                ServerHttpResponse response = exchange.getExchange().getResponse();
                response.setRawStatusCode(200);
                response.getHeaders().setLocation(URI.create("http://localhost:3000/login"));
                response.getCookies().remove( key: "SESSION");
                response.getHeaders().setAccessControlAllowOrigin("http://localhost:3000");
                response.getHeaders().setAccessControlAllowCredentials(true);
                response.addCookie(ResponseCookie.from( name: "token",  value: null)
                        .path("/")
                        .httpOnly(true)
                        .maxAge(0)
                        .build());
                return exchange.getExchange().getSession()
                        .flatMap(WebSession::invalidate);
            })
            .and() ServerHttpSecurity
            .addFilterAt(jwtAuthenticationFilter(), SecurityWebFiltersOrder.AUTHENTICATION)
            .oauth2Login(oauth2 -> oauth2.authenticationSuccessHandler(oAuthSuccessRedirectHandler))
            .authorizeExchange() ServerHttpSecurity.AuthorizeExchangeSpec
            .pathMatchers( ...antPatterns: "/login").permitAll()
            .pathMatchers(HttpMethod.OPTIONS).permitAll()
            .anyExchange().authenticated()
            .and() ServerHttpSecurity
            .build();
}
```

5.4.2 MongoConfig – налаштовуємо підключення до бази даних та менеджер транзакцій

```java
@Configuration
public class MongoConfig {

    public @Bean
    MongoClient reactiveMongoClient() {
        return MongoClients.create("mongodb+srv:@cluster0.gv6ch.mongodb.net/Milf?retryWrites=true&w=majority");
    }

    public @Bean
    ReactiveMongoTemplate reactiveMongoTemplate() {
        return new ReactiveMongoTemplate(reactiveMongoClient(), databaseName: "Milf");
    }

    public @Bean
    ReactiveMongoDatabaseFactory reactiveMongoDatabaseFactory() {
        return new SimpleReactiveMongoDatabaseFactory(MongoClients.create(), databaseName: "Milf");
    }

    public @Bean
    ReactiveMongoTransactionManager transactionManager(ReactiveMongoDatabaseFactory reactiveMongoDatabaseFactory) {
        return new ReactiveMongoTransactionManager(reactiveMongoDatabaseFactory);
    }

}
```

5.4.3 Налаштовуємо HTTPS(TLSv1.3) та HTTPv2:

```
@Bean
public WebServerFactoryCustomizer<NettyReactiveWebServerFactory> customizer() {
    return factory -> {
        Http2 http2 = new Http2();
        http2.setEnabled(true);
        factory.setHttp2(http2);

        Ssl ssl = new Ssl();
        ssl.setEnabled(true);
        ssl.setKeyStore("D:\\jks\\new\\keystore.jks");
        ssl.setKeyPassword("password");
        ssl.setKeyAlias("selfsigned");
        ssl.setCiphers(new String[]{
                /* openssl = ECDHE-ECDSA-AES128-GCM-SHA256 */
                "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256",

                /* REQUIRED BY HTTP/2 SPEC */
                /* openssl = ECDHE-RSA-AES128-GCM-SHA256 */
                "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256",
                /* REQUIRED BY HTTP/2 SPEC */

                /* openssl = ECDHE-ECDSA-AES256-GCM-SHA384 */
                "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384",
                /* openssl = ECDHE-RSA-AES256-GCM-SHA384 */
                "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384",
                /* openssl = ECDHE-ECDSA-CHACHA20-POLY1305 */
                //   "TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256",
                /* openssl = ECDHE-RSA-CHACHA20-POLY1305 */
                //   "TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256",

                /* TLS 1.3 ciphers */
                //   "TLS_AES_128_GCM_SHA256",
                "TLS_AES_256_GCM_SHA384",
                //   "TLS_CHACHA20_POLY1305_SHA256"
        });
        ssl.setProtocol("Tls");
        ssl.setEnabledProtocols(new String[]{"TLSv1.3", "TLSv1.2"});
        factory.setSsl(ssl);
    };
}
```

Висновки:

Результатом виконання цієї роботи стало створення самостійного програмного продукту, який дозволяє користувачеві в повному обсязі використовувати всі його технічні можливості і виконувати поставлені перед ним задачі. Застосунок прийдеться до смаку людям які люблять гаяти свій час у соцмережах.

В ході розробки та проектування були досліджені і проаналізовані основні методи та засоби розробки програмного забезпечення у браузерному середовищі з використанням протоколів HTTP, Websocket. Також був розроблений зручний і інтуїтивно зрозумілий графічний інтерфейс з адаптацією під різні розміри екрану та двома користувацькими темами, що дозволяє користуватися цим застосунком на будь-якому пристрої який може використовувати браузер.

Всі функціональні вимоги, які ставилися переді мною, були успішно виконані. Даний програмний продукт був розроблений за допомогою мов програмування Java та JavaScript і інтегрованого середовища розробки IntelliJ IDEA, яке дозволяє в повному обсязі використовувати усі необхідні засоби та інструменти для успішної розробки клієнт-серверних застосунків.

Список літератури

1. Project Reactor [Електронний ресурс] – Режим доступу до ресурсу: https://projectreactor.io/docs/core/release/reference/
2. Spring WebFlux [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html
3. Spring Security [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-security/reference/index.html
4. Spring MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/
5. Spring for GraphQl [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-graphql/docs/current-SNAPSHOT/reference/html/
6. React [Електронний ресурс] – Режим доступу до ресурсу: https://uk.reactjs.org/docs/hooks-reference.html
7. Apollo client [Електронний ресурс] – Режим доступу до ресурсу: https://www.apollographql.com/docs/
8. Sass [Електронний ресурс] – Режим доступу до ресурсу: https://sass-lang.com/documentation/
9. Material UI [Електронний ресурс] – Режим доступу до ресурсу: https://mui.com/material-ui
10. React Router [Електронний ресурс] – Режим доступу до ресурсу: https://reactrouter.com/docs/en/v6
11. MDN [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/

Додаток А (обов’язковий). Лістинг класу PostController

```
private final PostService postService;
private final WebClientService webClientService;
private final Sinks.Many<PostNUser> postCreatedSink;
private final ConcurrentHashMap<String, Sinks.Many<SubscriptionEvent>> sinksHashMap;

public Sinks.Many<SubscriptionEvent> personalSink() {
    return Sinks.many().multicast().onBackpressureBuffer(Integer.MAX_VALUE,  autoCancel: false);
}

public PostController(PostService postService,
                      WebClientService webClientService,
                      Sinks.Many<PostNUser> postCreatedSink,
                      ConcurrentHashMap<String, Sinks.Many<SubscriptionEvent>> sinksHashMap) {
    this.postService = postService;
    this.webClientService = webClientService;
    this.postCreatedSink = postCreatedSink;
    this.sinksHashMap = sinksHashMap;
}
```

```
@SubscriptionMapping
public Flux<PostNUser> postCreated() { return postCreatedSink.asFlux().share(); }

@SubscriptionMapping
public Flux<SubscriptionEvent> personalSubscription(Authentication authentication) {
    User user = (User) authentication.getPrincipal();
    sinksHashMap.putIfAbsent(user.getId(), personalSink());
    return sinksHashMap.get(user.getId()).asFlux();
}

@GetMapping(value = "/parse-html", produces = MediaType.APPLICATION_JSON_VALUE)
Mono<MetaDto> result(@RequestParam(required = true) String url) { return webClientService.getMetaDtoMono(url); }


@QueryMapping
Flux<PostNUser> getGlobalFeed() { return postService.findAllPostWithUsers(); }

@QueryMapping
Flux<Post> getAllPosts() { return postService.findAllPosts(); }

@QueryMapping
Mono<Post> getOnePost(@Argument String id) { return postService.getOne(id); }
```

```
@QueryMapping
Flux<Post> getAllPostsByAuthorId(@Argument String authorId) { return postService.getAllByAuthorId(authorId); }

@QueryMapping
Mono<PostNUser> getPostWithUser(@Argument String postId) { return postService.getPostWithUser(postId); }

@QueryMapping
Flux<PostNUser> getTopPostsWithUsers() { return postService.getTopPostsWithUsers(); }

@QueryMapping
Flux<PostNUser> getCommentsToPost(@Argument String postId) {
    return postService.getCommentsToPostWithUsers(postId);
}

@MutationMapping
Mono<Post> likeOrDislikePost(@Argument String postId, Authentication authentication) {
    User user = (User) authentication.getPrincipal();
    return postService.likeOrDislikePost(postId, user.getId());
}
```

```
    @PostMapping(value = "/upload-post")
    public Mono<Post> uploadPost(
            @RequestPart(value = "files", required = false) Flux<FilePart> fileParts,
            @RequestPart(value = "text", required = false) String text,
            @RequestPart(value = "link", required = false) String url,
            @RequestPart(value = "isCommentTo", required = false) String isCommentTo,
            Authentication authentication
    ) {

        User user = (User) authentication.getPrincipal();

        File uploadDir = new File( pathname: "D:\\files2");
        if (!uploadDir.exists()) {
            uploadDir.mkdir();
        }

        CleanerProperties props = new CleanerProperties();
        String cleanedText = new HtmlCleaner(props).clean(text).getText().toString();

        GenericExtFilter genericExtFilter = new GenericExtFilter( ...exts: "png", "jpeg", "jpg", "gif", "webp");

        Post post = new Post();
        post.setText(cleanedText);
        post.setAuthorId(user.getId());

        if (isCommentTo != null)
            post.setIsCommentTo(isCommentTo);
        if (url == null)
            url = "";

        return postService.validateAndSaveMessage(fileParts, text, url, genericExtFilter, post);
    }
```

Додаток Б(обов’язковий). Лістинг класу UserController

```
    private final UserRepository userRepository;
    private final UserService userService;
    private final JwtUtil jwtUtil;

    public UserController(UserRepository userRepository, UserService userService, JwtUtil jwtUtil) {
        this.userRepository = userRepository;
        this.userService = userService;
        this.jwtUtil = jwtUtil;
    }


    @PostMapping("/log-out")
    public Mono<ResponseEntity> logOut(ServerWebExchange swe, Authentication authentication) {

        ResponseCookie responseCookie = ResponseCookie.from( name: "token",  value: null)
                .path("/")
                .httpOnly(true)
                .maxAge(0)
                .build();

        return Mono.just(ResponseEntity
                .ok()
                .header(HttpHeaders.SET_COOKIE, responseCookie.toString())
                .build());
    }
```

```
@PostMapping("/login")
public Mono<ResponseEntity> login(ServerWebExchange swe) {
    return swe.getFormData().flatMap(credentials ->
            userRepository.findByUsername(credentials.getFirst( key: "username"))
                    .cast(User.class)
                    .map(userDetails -> {
                                if (Objects.equals(credentials.getFirst( key: "password"), userDetails.getPassword())) {
                                    swe.getResponse()
                                            .addCookie(
                                                    ResponseCookie.from( name: "token", jwtUtil.generateToken(userDetails))
                                                            .path("/")
                                                            .httpOnly(true).build());
                                    return ResponseEntity.ok(userDetails);
                                } else
                                    return UNAUTHORIZED;
                            }
                    )
                    .defaultIfEmpty(UNAUTHORIZED)
    );
}


@QueryMapping
public Mono<User> getMyProfile(Authentication authentication) {
    User user = (User) authentication.getPrincipal();
    return Mono.just(user);
}

@QueryMapping
public Mono<User> getUserById(@Argument String id) { return userService.getOneById(id); }
```