МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

Кафедра інженерії програмного забезпечення

**КУРСОВА РОБОТА**

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

з дисципліни Web-технології Ч.2

на тему:

**«Форум для програмістів»**

студента ІI курсу групи ІПЗ-20-4

спеціальності 121 «Інженерія програмного забезпечення»

Гісса Дмитра Олексійовича

(прізвище, ім’я та по-батькові)

Керівник ктн, доц. А.В. Морозов

Дата захисту: " 14 " січня 2022 р.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_ М.С. Граф

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_ О.Г. Чижмотря

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_ О.В. Кузьменко

(підпис) (прізвище та ініціали)

Житомир – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

Факультет інформаційно-комп’ютерних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень: бакалавр

Спеціальність 121 «Інженерія програмного забезпечення»

«ЗАТВЕРДЖУЮ»

Зав. кафедри

\_\_\_\_\_\_\_\_\_\_\_\_ А.В.Морозов

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_2021р.

ЗАВДАННЯ

НА КУРСОВУ РОБОТУ СТУДЕНТУ

Гісса Дмитру Олексійовичу

1. Тема роботи: Форум для програмістів ,

керівник курсового проекту: к.т.н., доцент А.В. Морозов

1. Строк подання студентом: “ ” 2021р.
2. Вхідні дані до роботи: Розробити форум для допомоги програмістам
3. Зміст розрахунково-пояснювальної записки(перелік питань. Які підлягають розробці)
4. Постановка завдання;
5. Аналіз аналогічних розробок;
6. Алгоритми роботи програми;
7. Опис роботи програми;
8. Програмне дослідження.
9. Перелік графічного матеріалу(з точним зазначенням обов’язкових креслень)
   1. Презентація
   2. https://gitlab.com/2020-2024/ipz-20-4/gissa-dmytro/observers
10. Консультанти розділів проекту (роботи)

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посади консультанта | Підпис, дата | |
| завдання  видав | завдання прийняв |
| 1, 2, 3 | Чижмотря О. Г. старший викладач кафедри ІПЗ |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Дата видачі завдання “15 ” вересня 2021 р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів курсової роботи | Строк виконання етапів робоьт | Примітки |
| 1 | Постановка задачі | 15.09.21 - 19.10.21 | Виконано |
| 2 | Пошук, огляд та аналіз аналогічних розробок | 20.10.21 - 30.10.21 | Виконано |
| 3 | Формулювання технічного завдання | 31.10.21 - 14.11.21 | Виконано |
| 4 | Опрацювання літературних джерел | 15.11.21 - 27.11.21 | Виконано |
| 5 | Проектування структури | 28.11.21 - 7.12.21 | Виконано |
| 6 | Написання програмного коду | 08.12.21 - 07.01.22 | Виконано |
| 7 | Відлагодження | 08.01.22 - 10.01.22 | Виконано |
| 8 | Написання пояснювальної записки | 11.01.22 - 13.01.22 | Виконано |
| 9 | Захист | 14.01.22 |  |

**Студент** \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гісса Д. О

(підпис) (прізвище та ініціали)

**Керівник проекту** \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чижмотря О. Г.

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до курсової роботи на тему «Форум для програмістів» складається з переліку умовних скорочень, вступу, трьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на 47 сторінках друкованого тексту.

Пояснювальна записка має 9 сторінок додатків. Список використаних джерел містить 10 найменувань і займає 1 сторінку. В роботі наведено 37 рисунків. Загальний обсяг роботи – 59 сторінок.

У першому розділі було обґрунтовано створення сайту на тему «Форум для програмістів»

У другому розділі проведено проектування і розробку сайту.

У третьому розділі показано роботу і проведено тестування програмного продукту.

Висновок містить в собі результати виконаної роботи при створенні сайту на тему «Форум для програмістів».

У додатку представлений лістинг розробленого програмного продукту.

Ключові слова: PYTHON, POSTGRESQL, JAVASCRIPT, WEB-САЙТ, MVC, СЕРВЕР, АДМІН, АВТОРИЗАЦІЯ, ФОРУМ, ІНТЕРНЕТ, ЗАПИТАННЯ, СПІЛКУВАННЯ.

ЗМІСТ

[ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 6](#_Toc92980289)

[ВСТУП 7](#_Toc92980290)

[РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ 9](#_Toc92980291)

[**1.1.** **Аналіз задачі, засобів та методів її вирішення.** 9](#_Toc92980292)

[**1.2.** **Аналіз існуючих веб-сайтів за тематикою курсової роботи.** 11](#_Toc92980293)

[**1.3.** **Технічне завдання на курсову роботу.** 18](#_Toc92980294)

[**Висновик до 1 розділу** 21](#_Toc92980295)

[РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 22](#_Toc92980296)

[**2.1.** **Проектування загального алгоритму роботи веб-сайту** 22](#_Toc92980297)

[**Загальна схема роботи програми** 23](#_Toc92980298)

[**2.2.** **Розробка функціональних алгоритмів роботи програми** 24](#_Toc92980299)

[**Діаграма таблиць бази даних:** 25](#_Toc92980300)

[**2.3.** **Розробка програмного забезпечення** 26](#_Toc92980301)

[**Висновки до 2 розділу** 32](#_Toc92980302)

[РОЗДІЛ 3 ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ 33](#_Toc92980303)

[**3.1.** **Опис роботи з програмним додатком (Опис інтерфейсу)** 33](#_Toc92980304)

[**3.2.** **Тестування роботи програмного забезпечення** 43](#_Toc92980305)

[**Висновки до 3 розділу** 46](#_Toc92980306)

[**ВИСНОВКИ** 47](#_Toc92980307)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** 48](#_Toc92980308)

[**ДОДАТКИ** 49](#_Toc92980309)

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – База даних

КР - Курсова робота

MVC – Модель–вигляд–контролер (або Модель–представлення–контролер, Model-view-controller, MVC) — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

ПЗ – Програмне забезпечення

# ВСТУП

У цій курсовій роботі буде наведено процес створення сайту форуму для програмістів.

Форум – це місце для спілкування і суперечок людей на різноманітні теми. Також це форма спілкування на сайті, де кожне повідомлення має автора і зміст, а кожна ваша відповідь буде частиною відкритої дискусії на певну тему. Форуми дозволяють зберігати питання і відповідь до них протягом багатьох років.

Робота форуму полягає у створенні користувачами певних запитань, всередині яких відбувається їхнє обговорення. Запитання зазвичай містить у собі інформацію типу «автор-тема-дата-зміст».

Кожен учасник має свій акаунт, до якого він може увійти за допомогою логіну і паролю. Це допомагає ідентифікувати учасника дискусії і знати від кого написане певне повідомлення.

*Метою* курсової роботи є дослідження особливостей проектування і реалізації сайту та розробка форуму з використанням веб-технологій для створення клієнтської і серверної частини сайту.

*Завданням* на курсову роботу є:

* аналіз теоретичних засад проектування та реалізації форуму;
* аналіз існуючих форумів, визначення унікальності;
* визначення інформаційних потреб користувача;
* розробка адаптивного інтерфейсу веб-сайту за допомогою HTML5, CSS3, JavaScript;
* розробка алгоритмічної моделі функціонування системи на основі БД;
* проектування БД для форуму;
* написання серверної частини;
* написання клієнтської частини.

*Об’єктом дослідження* є робота з інформацією та даними користувачів, їхній пошук, відображення, редагування і зберігання.

*Предметом дослідження* є сайт форуму для програмістів для створення, редагування, перегляду, обговорення і видалення питань.

# РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

## **Аналіз задачі, засобів та методів її вирішення.**

Задача полягає в тому, що потрібно:

1. Визначити потреба адміністратора і звичайного користувача.
2. Спроектувати БД.
3. Написати серверну частину сайту.
4. Продумати дизайн сайту.
5. Написати клієнтську частину сайту.
6. Обмежити права звичайного користувача.
7. Продумати і реалізувати адаптивність сайту для зручності користування на телефоні.

Для розробки КР було обрано такі середовища розробки, як PyCharm Professional, WebStorm і Visual Studio Code. Вони були обрані, оскільки підтримують всі необхідні інструменти, які використовувалися в ході розробки.

Для написання серверної частини сайту було обрано мову програмування Python і фреймворк FastAPI, бо Python це легка мова програмування на якій можна швидко створювати проекти, а FastAPI використовує найновітніші практики і є зручним й асинхронним.

У якості БД було обрано PostgreSQL, оскільки вона є одною з найпопулярніших і найшвидших на сьогодні.

Для написання клієнтської частини сайту було обрано фреймворк React і мова програмування TypeScript. Такий вибір було здійснено, бо React дозволяє створювати сайти, які працюють повністю асинхронно і не потребують перезавантаження сторінки, а TypeScript дозволяє зробити відлагодження програмного коду швидшим і зручнішим.

Також, для стилізування сайту було обрано CSS-фреймворк Tailwind, оскільки він надає вже готові стилі і при цьому дозволяє повну їхню кастомізацію.

## **Аналіз існуючих веб-сайтів за тематикою курсової роботи.**

При аналізі вже існуючих веб-сайтів форумів було виявлено декілька проектів. Вони досить схожі один на одного за функціоналом, але відрізняються дизайном і мовою, на якій публікуються запитання.

Аналогічні веб-сайти за тематикою курсової роботи:

* + - 1. Stack Overflow

Посилання: <https://stackoverflow.com/>

Розглянемо веб-сайт Stack Overflow:

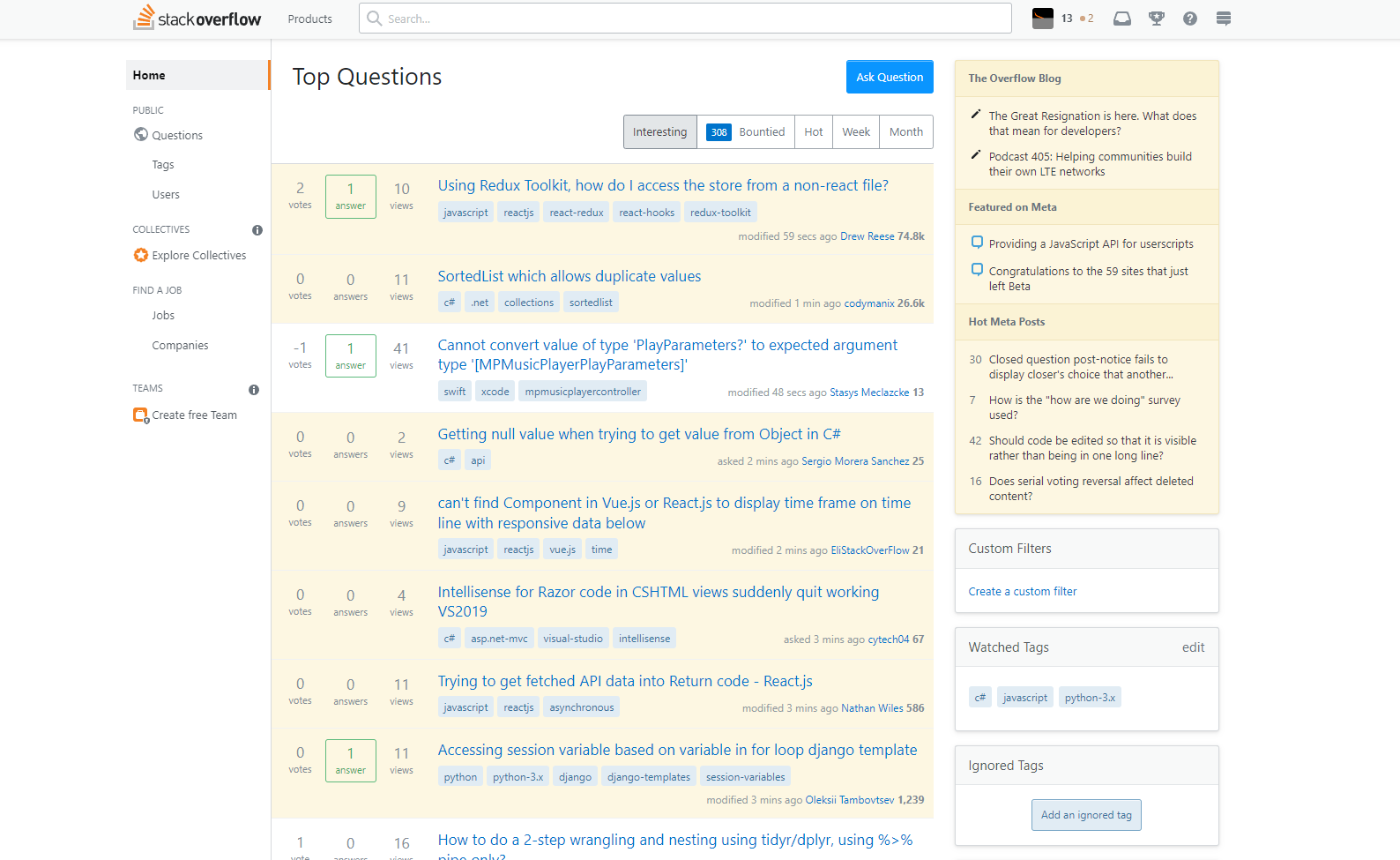


Рис. 1.1. Stack Overflow – домашня сторінка.

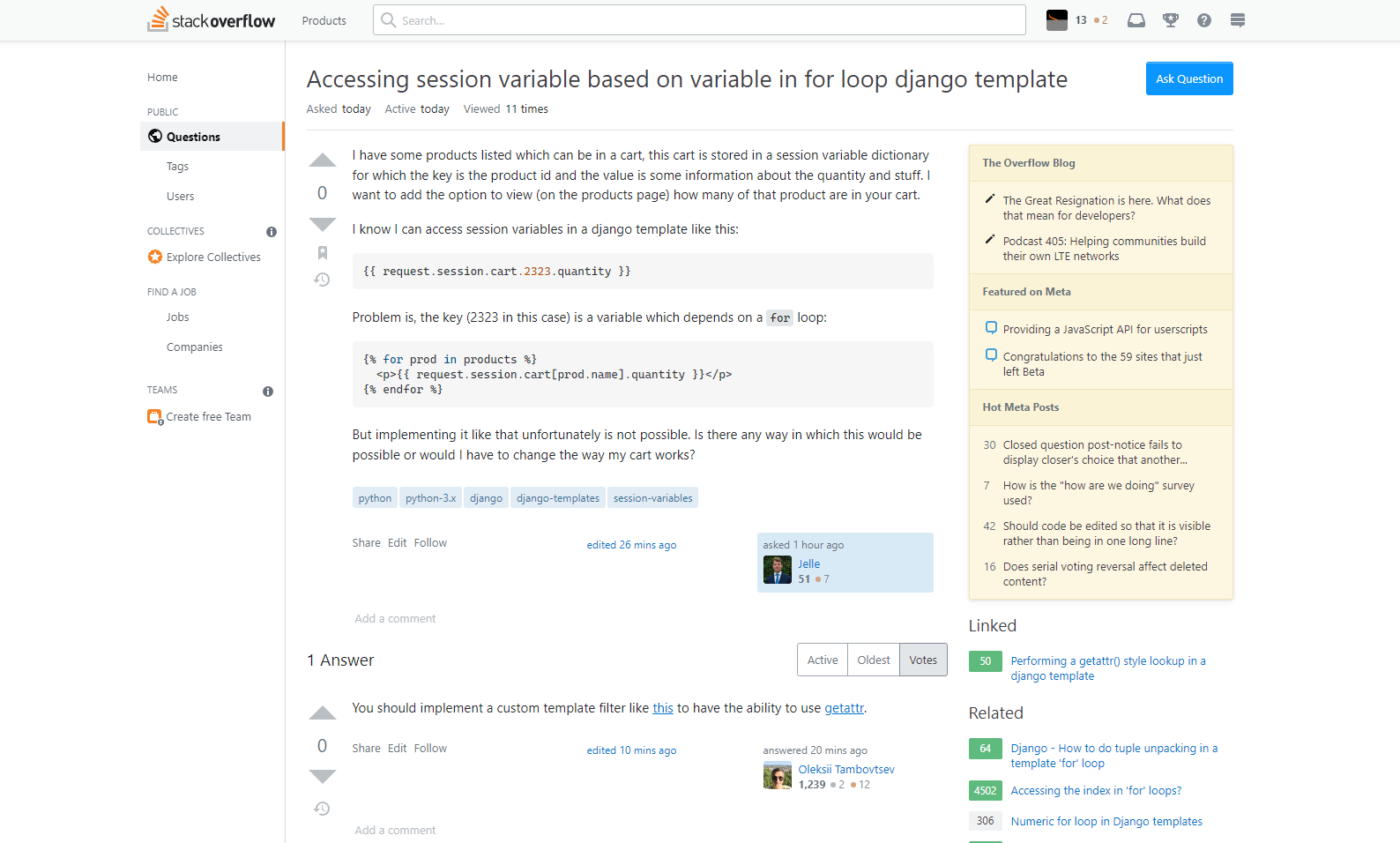


Рис. 1.2. Stack Overflow – приклад запитання.

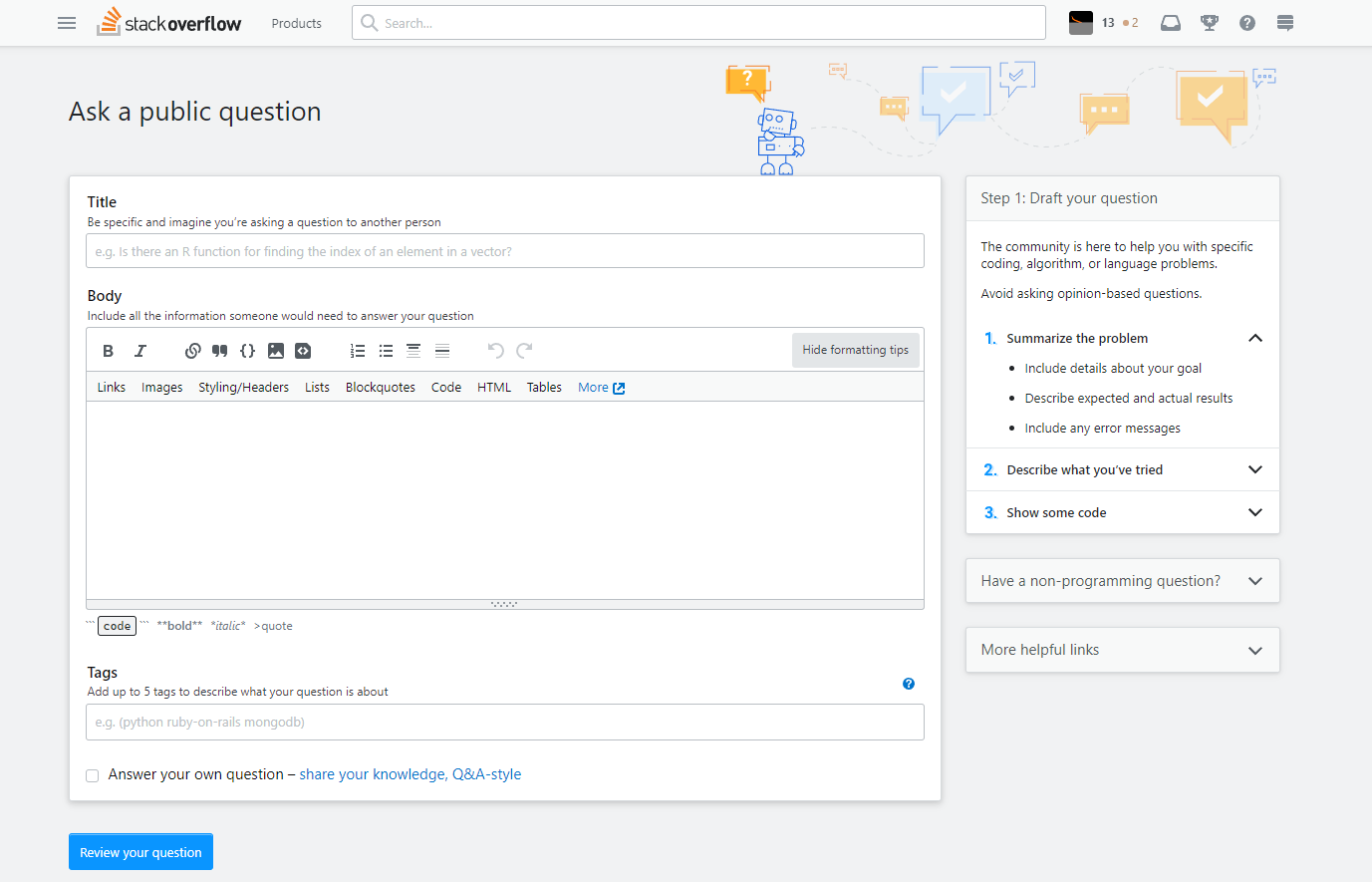


Рис. 1.3. Stack Overflow – форма для створення питання.

Форум Stack Overflow є найбільш популярним серед програмістів, оскільки на ньому користувачі використовують тільки англійську мову і це сприяє доступності матеріалу по усьому світу.

Основні переваги:

* Інтуїтивно-зрозумілий інтерфейс
* Можливість вибору теми оформлення сайту
* Присутній адаптив

Основні недоліки:

* Через те, що запитання ставляться лише англійською мовою, матеріал недоступний для користувачів, які її не знають (частково вирішено окремим сайтом Stack Exchange)
  + - 1. Habr Q&A

Посилання: <https://qna.habr.com/>

Розглянемо веб-сайт Habr Q&A:

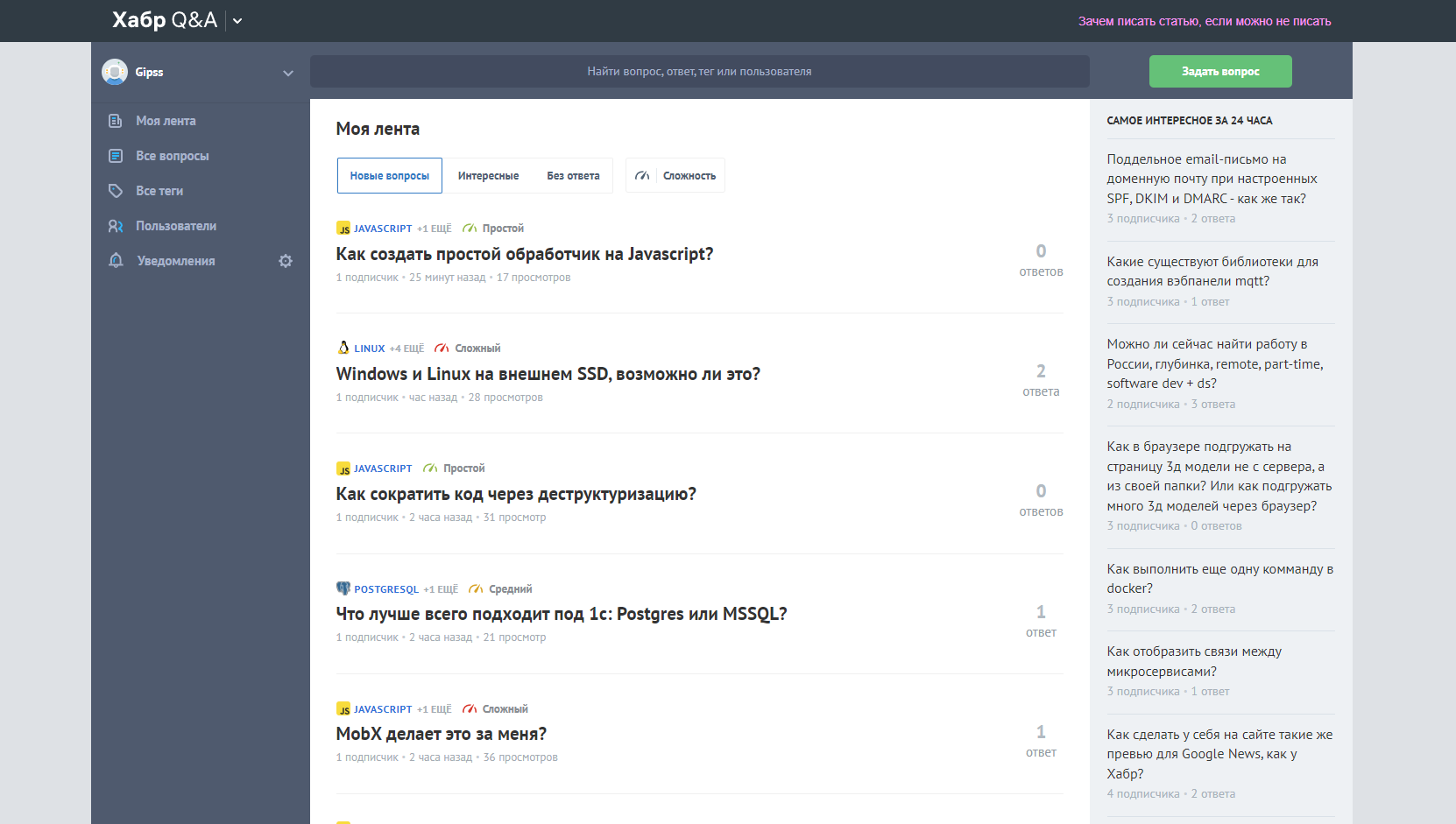


Рис. 1.4. Habr Q&A – домашня сторінка.

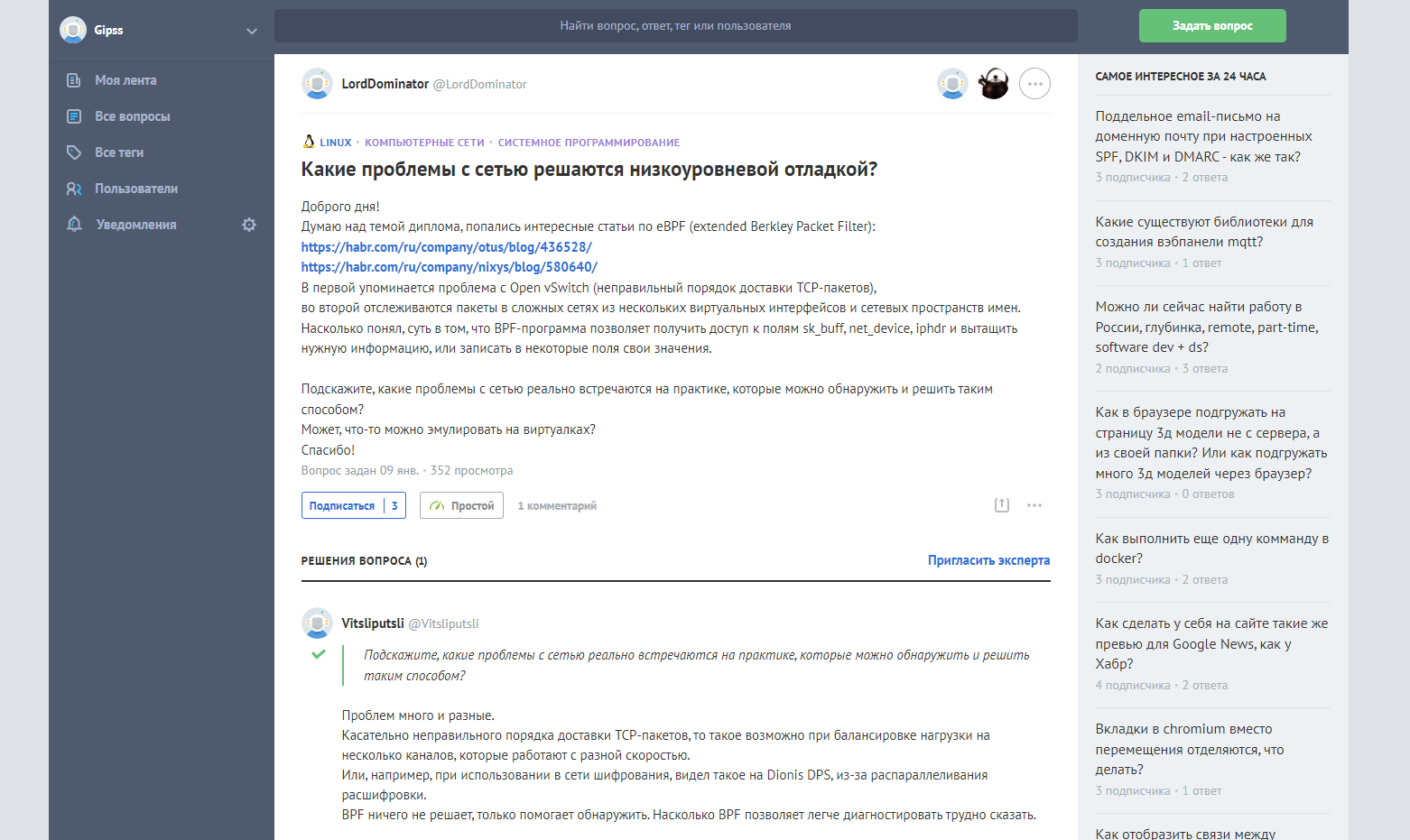


Рис. 1.5. Habr Q&A – приклад запитання.

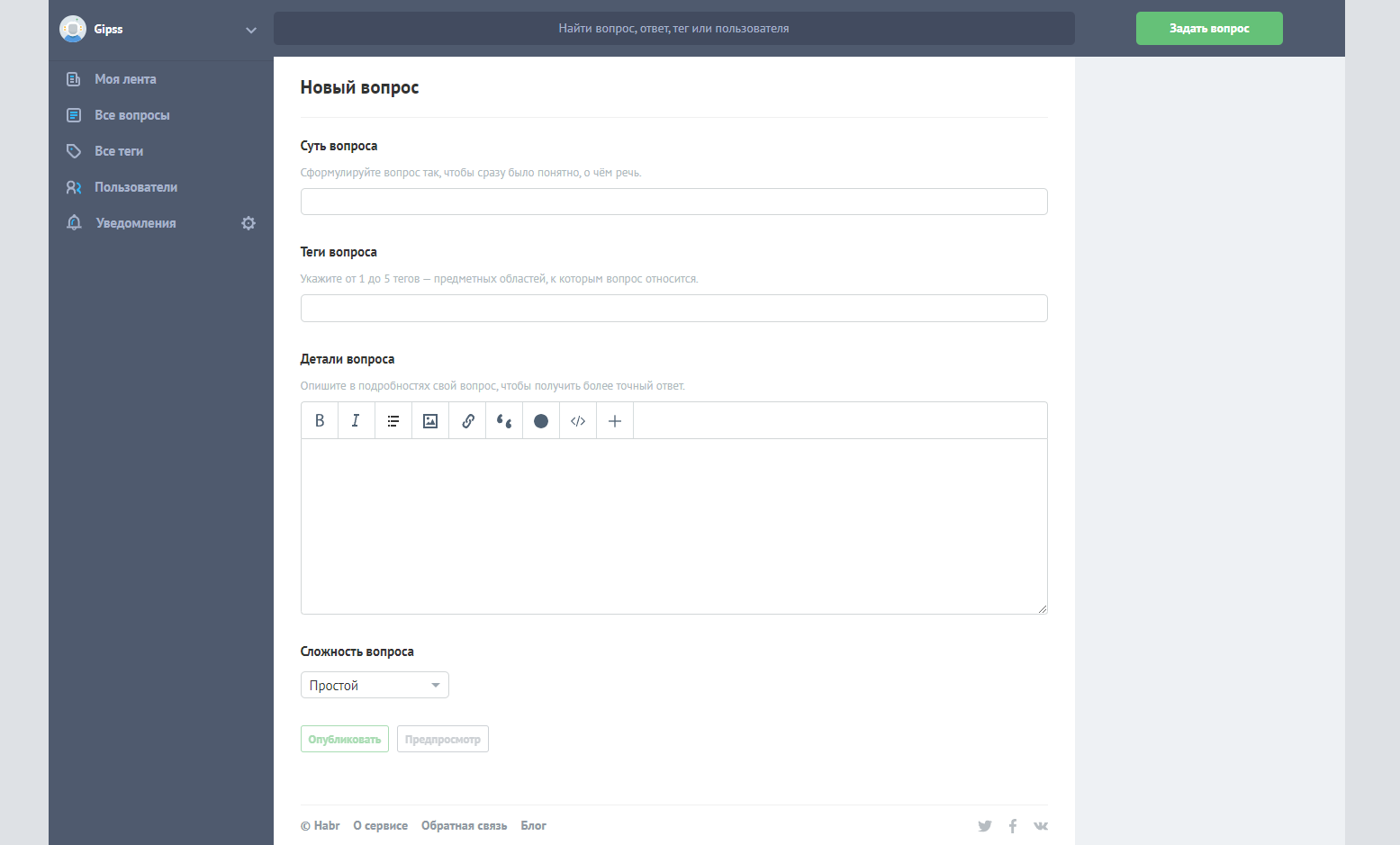


Рис. 1.6. Habr Q&A – форма створення питання.

Форум Habr Q&A популярний серед програмістів лише в країнах пострадянського простору. Особливістю сайту є те, що він одразу запитує про теми, які цікавлять користувача, і починає рекомендувати питання, зв’язані з ними.

Основні переваги:

* Інтуїтивно-зрозумілий інтерфейс
* Присутній адаптив

Основні недоліки:

* Неможливість вибору іншої теми оформлення сайту
* Запитання ставляться лише російської мовою, що робить інформацію недоступною для більшості країн світу
  + - 1. DOU

Посилання: <https://dou.ua/forums/>

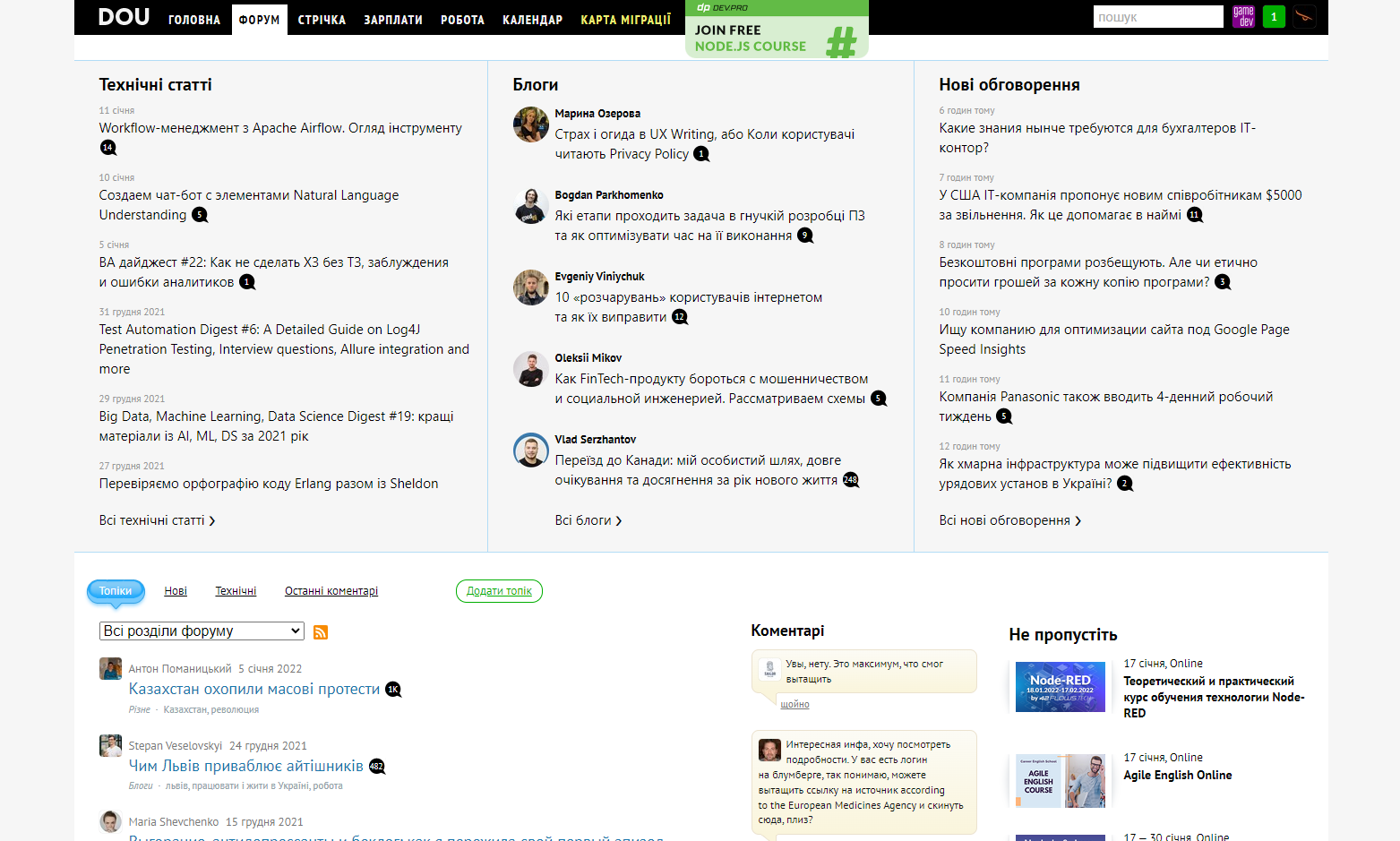


Рис. 1.7. DOU – домашня сторінка форуму.

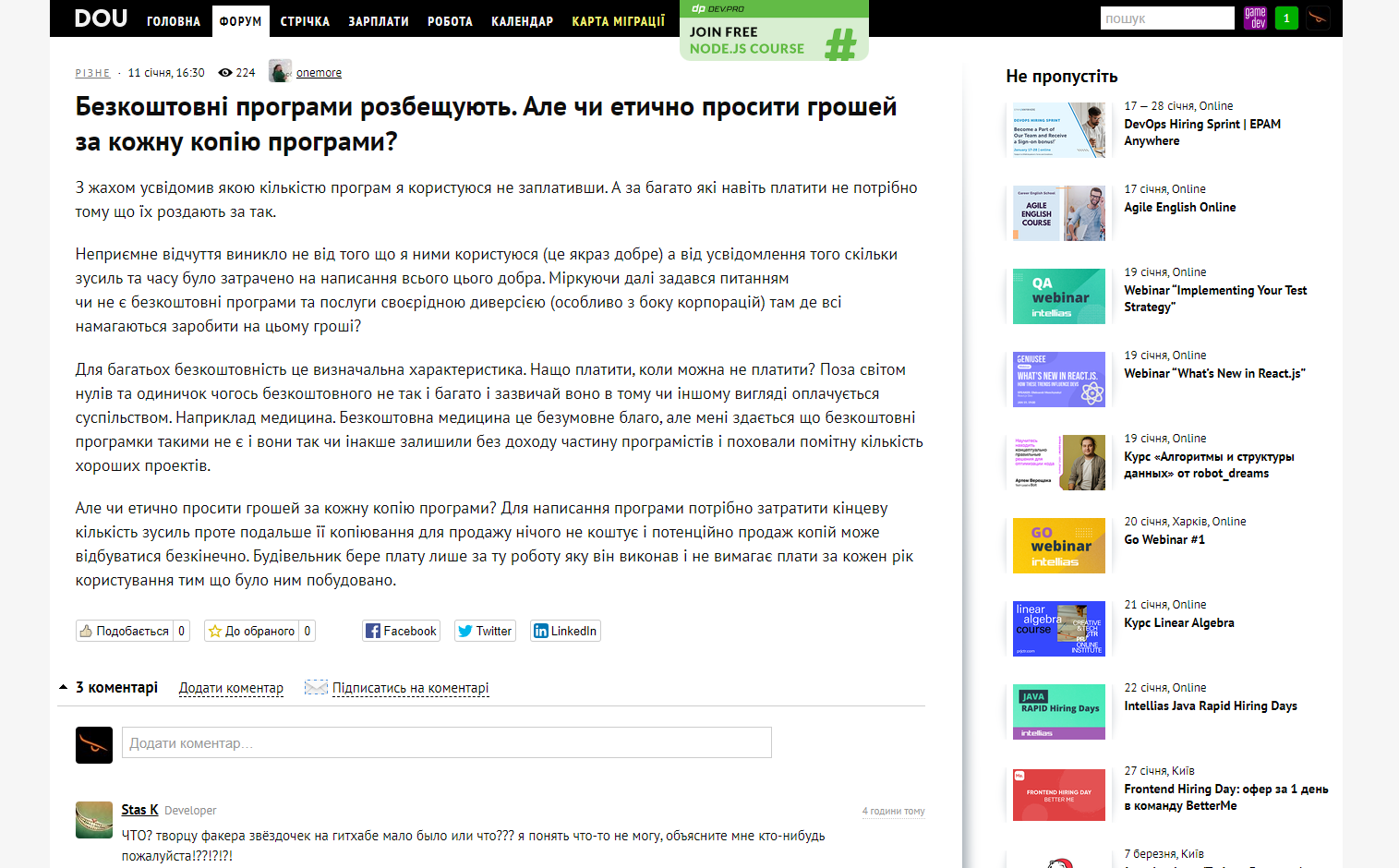


Рис. 1.8. DOU – приклад питання.

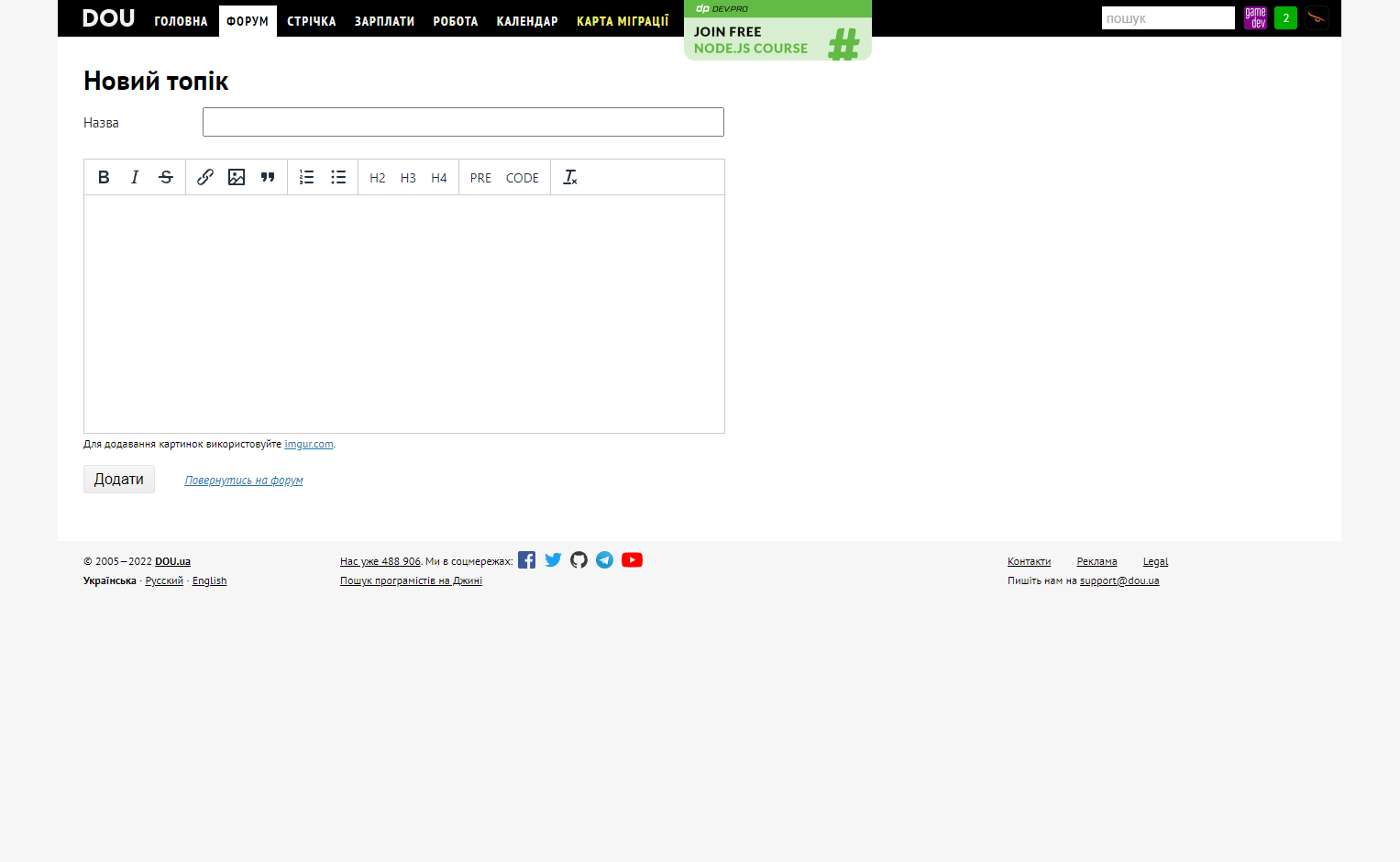


Рис. 1.9. DOU – форма створення топіку.

DOU – це сайт української спільноти програмістів. Він відрізняється від попередніх аналогів, оскільки він більше орієнтований на публікування і обговорення новин світу інформаційних технологій. Він містить як новини, так і технічні статті, які допомагають розібратися у певній технології.

Основні переваги:

* Інтуїтивно-зрозумілий інтерфейс
* Різнобічність сайту

Основні недоліки:

* Відсутність адаптиву
* Відсутність класичних запитань
* Матеріал публікується українською або російською мовами, що обмежує доступ до інформації іноземних користувачам

При створенні власного веб-сайту необхідно враховувати недоліки і переваги проаналізованих вище форумів. Це означає, що потрібно продумати інтерфейс, а також реалізувати адаптивність сайту. Також, потрібно обрати основною мовою сайту англійську, оскільки завдяки цьому доступність інформації буде вищою. Реалізувати на сайті як класичну модель запитання-відповідь, так і новини для поширення інформації про сайт і світ інформаційних технологій.

## **Технічне завдання на курсову роботу.**

**Загальне положення**

* 1. **Найменування програмного засобу**

Повне найменування програмної системи: “Розробка форуму для програмістів” (надалі “веб-додаток”). Коротка назва програмногої системи – “Форум”.

* 1. **Призначення розробки та область застосування**

Веб-сайт “Форум для програмістів” призначений для розміщення питань, їх пошуку та обговорення.

Веб-сайт “Форум для програмістів” дозволить швидко переглядати інформацію про запитання, дозволить користувачеві обговорювати ці запитання разом із іншими людьми, а також дозволить весь процес контролювати адміністратору, який зможе видаляти та редагувати запитання і коментарі.

* 1. **Найменування розробника та замовника**

Розробник даного продукту – студент групи ІПЗ-20-4 Гісса Дмитро Олексійович (надалі “розробник”).

Замовник програмного продукту – кафедра інженерія програмного забезпечення Державного університету “Житомирська політехніка” в межах виконання курсової з дисципліни «Web-технології» Чижмотря Олена Генадіївна, Морозов Андрій Васильович.

1. **Підстава для розробки**
   1. **Документ на підставі якого ведеться розробка**

Робота ведеться на підставі навчального плану за напрямом 121 «Інженерія Програмного Забезпечення».

1. **Вимоги до програми**
   1. **Вимоги до функціональних характеристик.**
      1. **Загальні вимоги**

Веб-додаток має забезпечувати:

* Можливість дистанційної роботи з робочих станцій локальної та глобальної мережі підприємства;
* Постійний доступ до веб-додатку;
* Створення запитання;
* Створення коментаря;
* Управління сайтом;
* Доступ до бази даних.
  + 1. **Склад виконуваних функцій**

Розробити форум для програмістів, що підтримує виконання наступних операцій:

1. Створення запитання з певною інформацією:

* Заголовок
* Теги
* Контент

1. Редагування або видалення запитання.
2. Пошук запитань.
3. Фільтрування запитань за параметрами:

* Час створення
* Перегляди
* Теги

1. Перегляд запитань.
2. Створення новин з певною інформацією:

* Заголовок
* Контент

1. Редагування або видалення новин.
2. Перегляд новин.
   * 1. **Організація вхідних і вихідних даних**

Вхідними даними є інформація про запитання (заголовок, теги, контент) або про новину (заголовок, контент).

* + 1. **Часові характеристики і розмір пам’яті, необхідної для роботи програми**
* Час реакції програми на дії користувача (маніпуляції з пристроями введеня даних) не повинен перевищувати 0.25 с.
* Час виконання команд меню не більше 0.5 с.
* Відображення масивів даних за запитами не більше 1 хвилини.
* Доступність БД – цілодобово.
* Операції з’єднання з БД не більше 0.5 хвилини.
* Обсяг оперативної пам’яті, необхідний для роботи програми не менше 1Гб.
* Дисковий простір, необхідний для збереження програми і файлів даних не більше 300 Мбат для робочої станції та 20 Гбайт.
* Інсталяційний пакет програми, що містить у складі БД не повинні перевищувати 100 Мбайт.
  1. **Вимоги до методів рішення і мов програмування**

Вибір методів рішення здійснюється розробником без узгодження з замовником.

* + 1. **Вимоги до системи програмних засобів**

PostgreSQL, Python 3.10, TypeScript, React, HTML5, CSS3.

* + 1. **Вимоги до програмного забезпечення робочої станції**

PgAdmin4, PyCharm Professional, WebStorm, Visual Studio Code.

## **Висновик до 1 розділу**

У ході виконання першого розділу було описано постановку завдання і те, як його можна вирішити. Також було обрано інструменти, які є необхідними для створення веб-сайту. Було проведено аналіз схожих форумів для програмістів. В результаті цього було виділено основні моменти, на які потрібно звернути увагу в ході розробки курсової роботи.

# РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## **Проектування загального алгоритму роботи веб-сайту**

На даному етапі розробки розглянемо загальний алгоритм роботи веб-сайту форуму.

Загальна схема роботи веб-сайту форуму приведена на рисунку 2.1.

Опис загального алгоритму веб-сайту наведено нижче.

При вході до сайту користувача зустрічає вітальна сторінка, з якої він може перейти у різні частини веб-сайту.

Якщо у користувача немає акаунту, він може зайти і зареєструватися. Після цього у нього з’являться можливість авторизуватися, що дає йому додаткові права на сайті, а сама створювати запитання, писати коментарі тощо.

Після авторизування користувач може зайти на свою сторінку, де має можливість редагувати інформацію про себе (пошта, фотографія, пароль).

Якщо користувач зайде на сторінку запитань, він побачить їхній перелік. Якщо користувач не авторизований, він може лише переглядати інформацію. В іншому разі, у нього з’являються додаткові можливості такі, як створення запитань і написання коментарів.

Також користувач має можливість зайти на сторінку новин, де зможе переглянути останні події з життя сайту й інформаційних технологій. Також, є можливість оцінити новину, поставивши «Лайк» або «Дизлайк», але вона доступна лише авторизованим користувачам.

Авторизовані користувачі можуть зайти на сторінку загального чату, де є інші відвідувачі. Там люди можуть спілкуватися у режимі реального часу, без потреби перезавантажувати сторінку.

Якщо користувач створив питання і хтось відповів йому, то у користувача з’являться сповіщення про цю подію і він може одразу перейти до певного запитання.

Якщо користувач це адміністратор, то у нього є можливість зайти на сторінку адміністратора, де він зможе побачити всі існуючі ролі на сайті, теги, користувачів, запитання, новини і коментарі. Також, він може редагувати деякі з цих моделей даних, наприклад, запитання, коментарі, новини.

### **Загальна схема роботи програми**

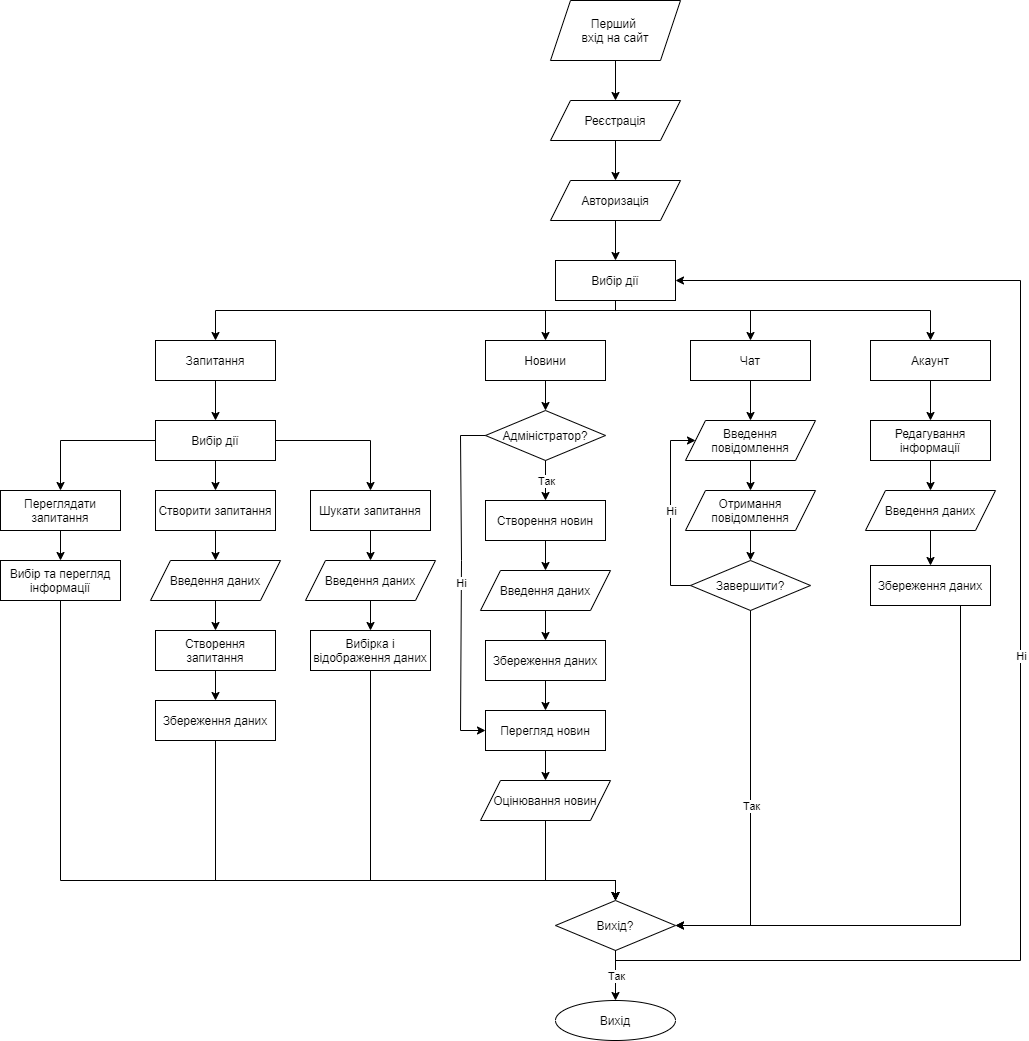


Рис. 2.1. Загальна схема роботи програми.

## **Розробка функціональних алгоритмів роботи програми**

Розглянемо алгоритми роботи основних функцій.

Основна задача форуму – це отримання інформації від користувача і її збереження протягом тривалого часу.

Вся інформація на сайті зберігається в таблицях бази даних. Для того, щоб додати певні дані, користувач повинен бути авторизованим і виконати певні дії (наприклад, ввести дані для створення запитання). Його дії викличуть певні функції, які будуть зв’язуватися із серверною частиною сайту і передавати необхідну інформацію у вигляді JSON, а сервер буде відповідати певними кодами статусу, які будуть означати певний результат дії (наприклад, коди, що починаються з 2 (200, 204) означають успіх, а коди, що починаються з 4 (404, 401) означають помилку на боці клієнта (користувача)).

Обов’язковим завданням, яке має вирішуватися у програмному коді є валідація даних, введених користувачем, оскільки він може помилитися або бути зловмисником. Для більшої надійності перевірка даних буде відбуватися, як на клієнтській частині сайту, так і на серверній. На боці сервера з валідацією даних частково допомагає фреймворк, але для специфічних місць проекту, дані будуть перевірятися власноруч. На стороні клієнту, дані в основному перевіряються за допомогою вже готового модуля, але в деяких місцях, де його функціонал є надлишковим, перевірка буде здійснена власноруч.

### **Діаграма таблиць бази даних:**

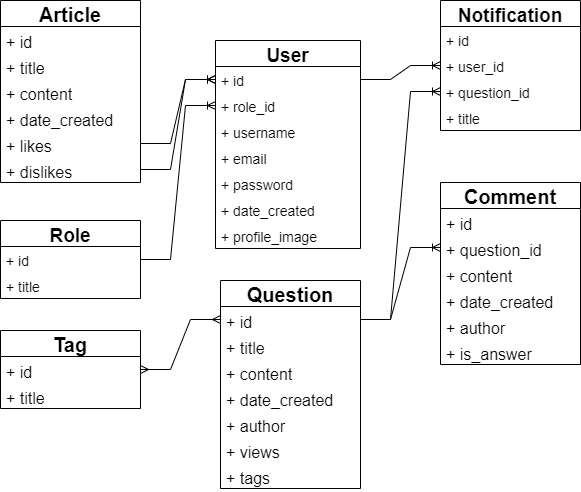


Рис. 2.2. Діаграма таблиць бази даних.

## **Розробка програмного забезпечення**

Для написання серверної частини веб-сайту було обрано фреймворк мови програмування Python FastAPI, а для написання клієнтської частини – фреймворк React разом з мовою програмування TypeScript.

Частини сайту було винесено в окремі папки для зручності і структурованості.

Структура серверної частини веб-сайту:

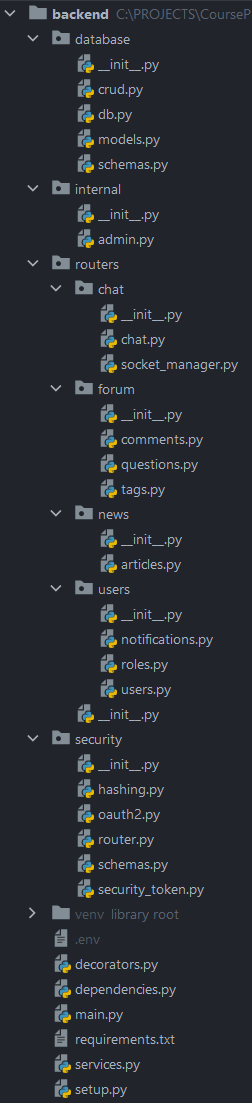


Рис. 2.3. Структура серверної частини веб-сайту.

Структура клієнтської частини веб-сайту:

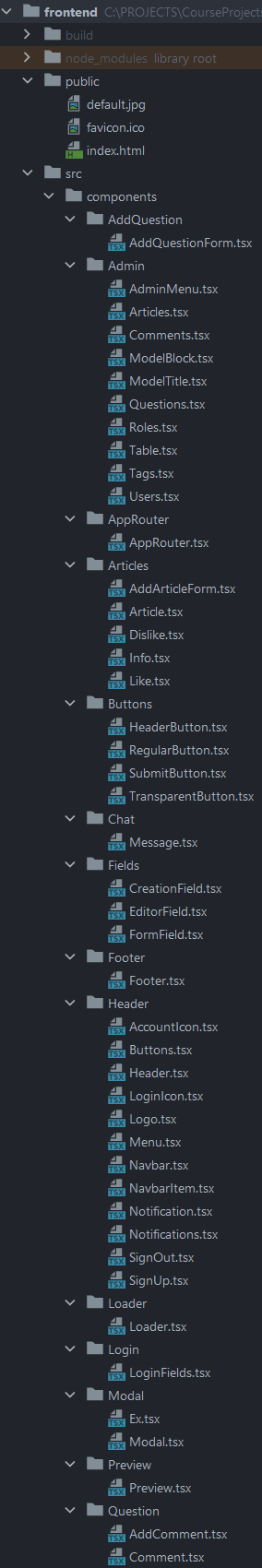
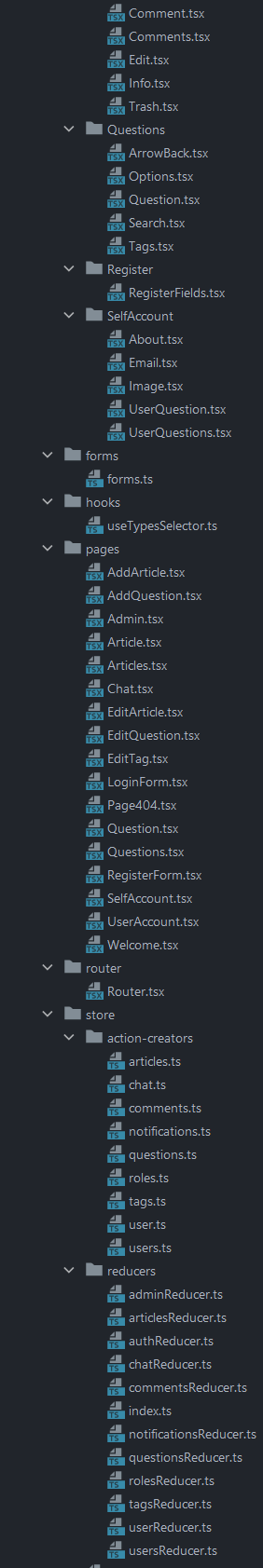
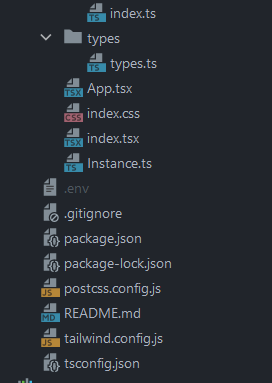
  

Рис. 2.4. Структура клієнтської частини веб-сайту.

Основна робота веб-сайту відбувається таким чином: користувач вводить дані, перевірка даних на клієнтській частині, передача даних на сервер, перевірка даних на сервері, звернення до бази даних і повернення певного результату. Тому, за приклад буде взято лише декілька участків коду, які є зразковими для всього проекту.

Спочатку розглянемо роботу на боці клієнта.

Для запитів на сервер використовувався модуль axios. Для спрощення і уникнення дублікування коду певні базові налаштування запитів для усього проекту було здійснено в одному файлі:

**import *axios* from "axios"**;  
  
**export const *url*** = **'192.168.0.102:8000/'  
  
export const *instance*** = ***axios***.create({  
 **baseURL**: **`http://**${***url***}**api/`**,  
});  
  
***instance***.**interceptors**.**request**.use((config) => {  
 **const** token = ***localStorage***.getItem(**'token'**);  
 **if**(config && config.**headers**)  
 config.headers.**Authorization** = token ? token : **''**;  
 **return** config;  
});

Збір даних і їх відправку було реалізовано в різних файлах, тому назви функцій відрізняються. Розглянемо функцію submit, яка збирає дані, частково перевіряє їх (повну перевірку зробив модуль Formik) і передає у функцію відправлення:

**const** submit = () => {  
 **const** title: **string** = formik.**values**.**title**;  
 **const** tags: **string**[] = ***Array***.from(  
 **new *Set***(formik.**values**.**tags** .replaceAll(/\s+/g, **" "**)  
 .split(**" "**)  
 .filter(element => element !== **''**))  
 );  
 **const** content: **string** = formik.**values**.**content**;  
  
 **let** body: **any** = {title, content};  
  
 **if**(tags.**length** > 0){  
 body.**tags** = tags;  
 }  
  
 onSubmit(body)  
}

Варто звернути увагу на перевірку тегів, оскільки вона виконується власноруч. В першу чергу, ми отримуємо значення поля, потім заміняємо повторювані пробіли одним, розділюємо рядки по пробілу в масив і видаляємо пусті рядки. Далі ми поміщуємо масив в Set для того, що видалити повторювані елементи, оскільки Set містить лише унікальні. В кінці ми перевіряємо чи було введено теги. Якщо так, то ми додаємо їх до тіла запиту і передаємо його у функцію відправлення.

Розглянемо функцію відправлення запиту на сервер:

**const** createQuestion = (body: **any**) => {  
 ***instance***.post<IQuestion>(**'forum/questions/'**, body)  
 .then(response => {  
 navigate(**'/questions'**);  
 });  
}

Спершу, функція отримує тіло запиту. Далі ми вказуємо тип запиту, в цьому випадку це POST, що означає, що ми будемо створювати нові дані. Потім ми шлях до контроллера на сервері для створення запитання і передаємо тіло запиту. Після цього ми вказуємо, що після повернення сервером відповіді, потрібно переадресувати користувача на сторінку запитань.

Переглянемо контроллер, який виконується під час вищевказаного запиту:

@router.post(**'/'**, response\_model=schemas.Question)  
**def** create\_question(  
 question: schemas.QuestionCreate,  
 db: Session = Depends(get\_db),  
 current\_user: schemas.User = Depends(get\_current\_user)  
) -> models.Question:**if** question.author\_id **is None**:  
 question.author\_id = current\_user.id  
  
 **return** crud.create\_question(db=db, question=question)

У декораторі функції вказано шлях, за яким виконується контроллер (насправді він відрізняється від того, що там вказано через певні налаштування в інших частинах програми), і схема (клас модуля Pydantic), за якою буде валідуватися відповідь сервера.

Цей контроллер має три аргумента. Перший з них, question, це і є тіло запиту, перетворене у Pydantic-модель. Наступні два аргументи реалізовані за допомогою залежностей, які виконуються завдяки фреймворку FastAPI. В результаті db – сесія підключення до бази даних, а current\_user – це Pydantic-модель поточного користувача.

В результат повертається результат функції create\_question, розглянемо її:

**def** create\_question(db: Session, question: schemas.QuestionCreate) -> models.Question:get\_object(cls=models.User, db=db, object\_id=question.author\_id)  
  
 question\_db = models.Question(  
 title=question.title,  
 content=question.content,  
 author\_id=question.author\_id  
 )  
  
 db.add(question\_db)  
  
 fill\_tags(db=db, tags=question.tags, question\_db=question\_db)  
  
 db.commit()  
 db.refresh(question\_db)  
 **return** question\_db

Ця функція перевіряє чи існує користувач, який вказаний автором запитання, створює об’єкт для додавання у БД, якщо потрібно створює і заповнює теги об’єкта і повертає його.

В результаті цей об’єкт перетворюється в Pydantic-модель, потім у JSON і відправляється на клієнтську частину сайту.

За таким алгоритмом працює більша частина сайту.

## **Висновки до 2 розділу**

У ході виконання другого розділу було спроектовано алгоритми роботи програми та діаграму таблиць бази даних. Також, було показано структуру проекта, пояснено основний алгоритм роботи сайту і проаналізовано його важливі ділянки коду.

# РОЗДІЛ 3 ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ

## **Опис роботи з програмним додатком (Опис інтерфейсу)**

Після входу на сайт користувач бачить вітальну сторінку (Рисунок 3.1.), на якій він може вибрати, що він хоче зробити. Наприклад, користувач може зареєструватися, редагувати акаунт, читати запитання, створити запитання, читати новини, спілкуватися з іншими учасниками.

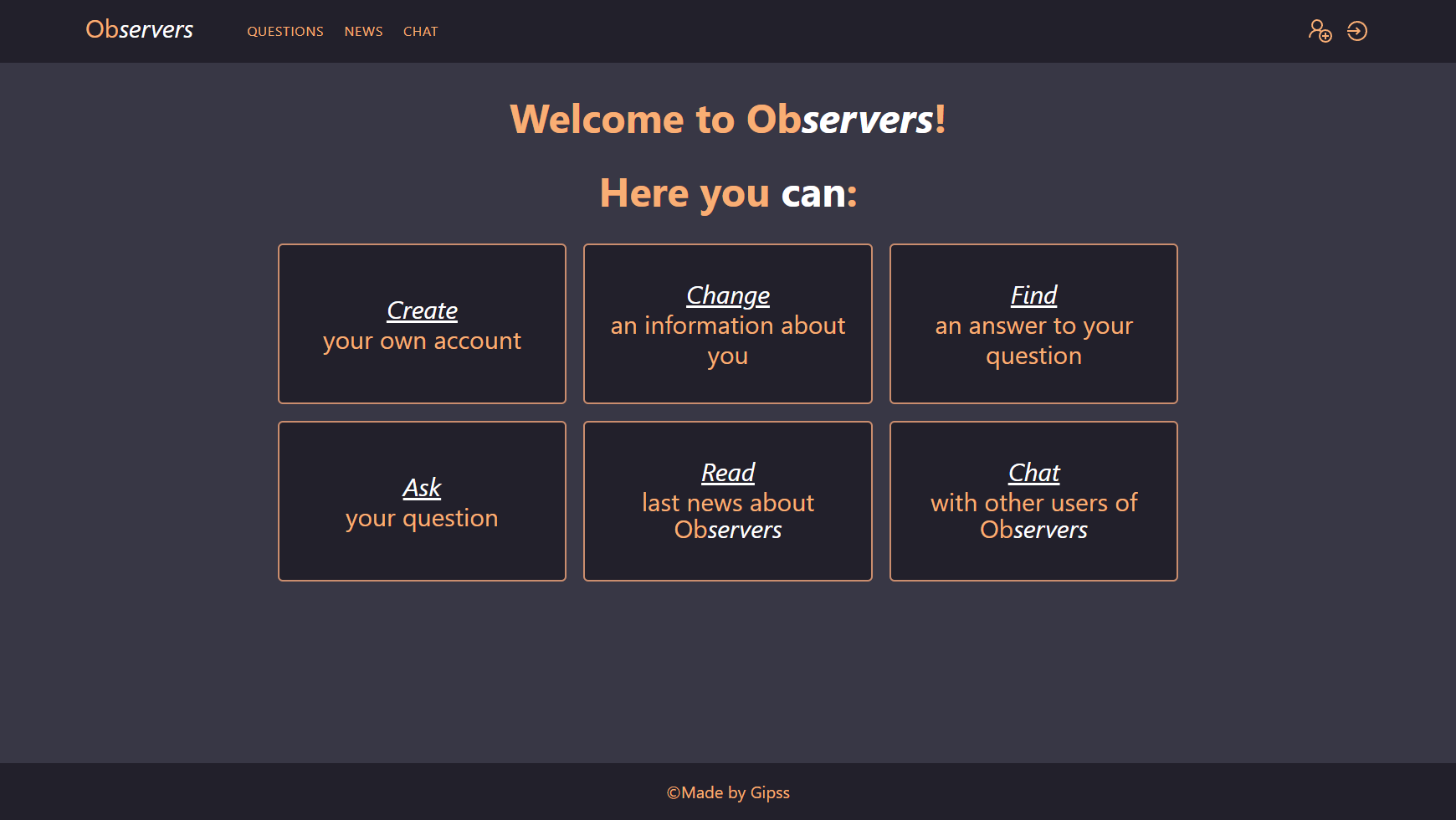


Рис. 3.1. Вітальна сторінка сайту.

Якщо обрати реєстрацію, то користувач перейде до сторінки форми реєстрації (Рисунок 3.2.), де він зможе ввести свої дані і створити акаунт.

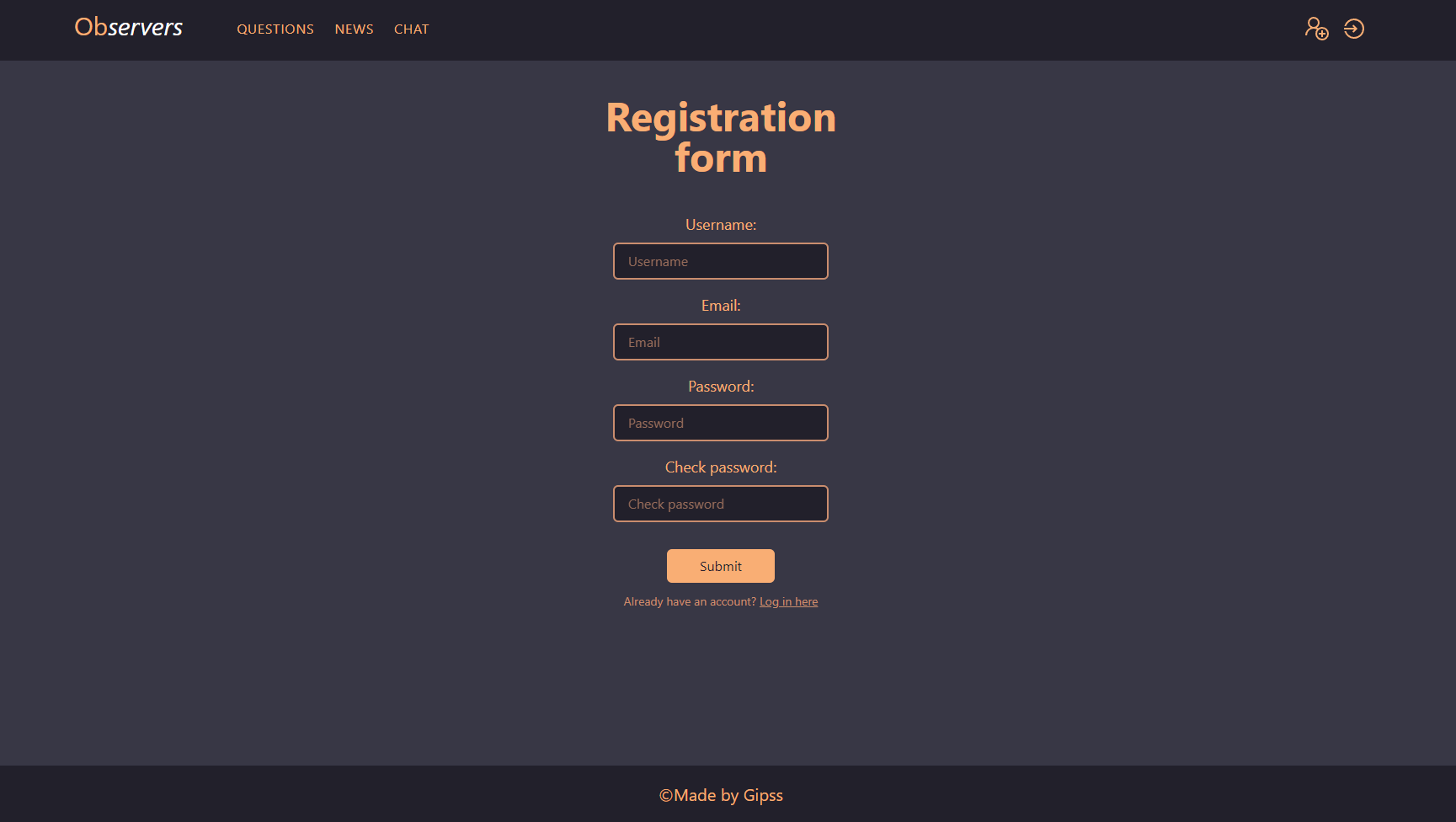


Рис. 3.2. Форма реєстрації.

Відправивши форму реєстрації користувача переміщує на форму авторизації (Рисунок 3.3.), де він може увійти до свого акаунту.

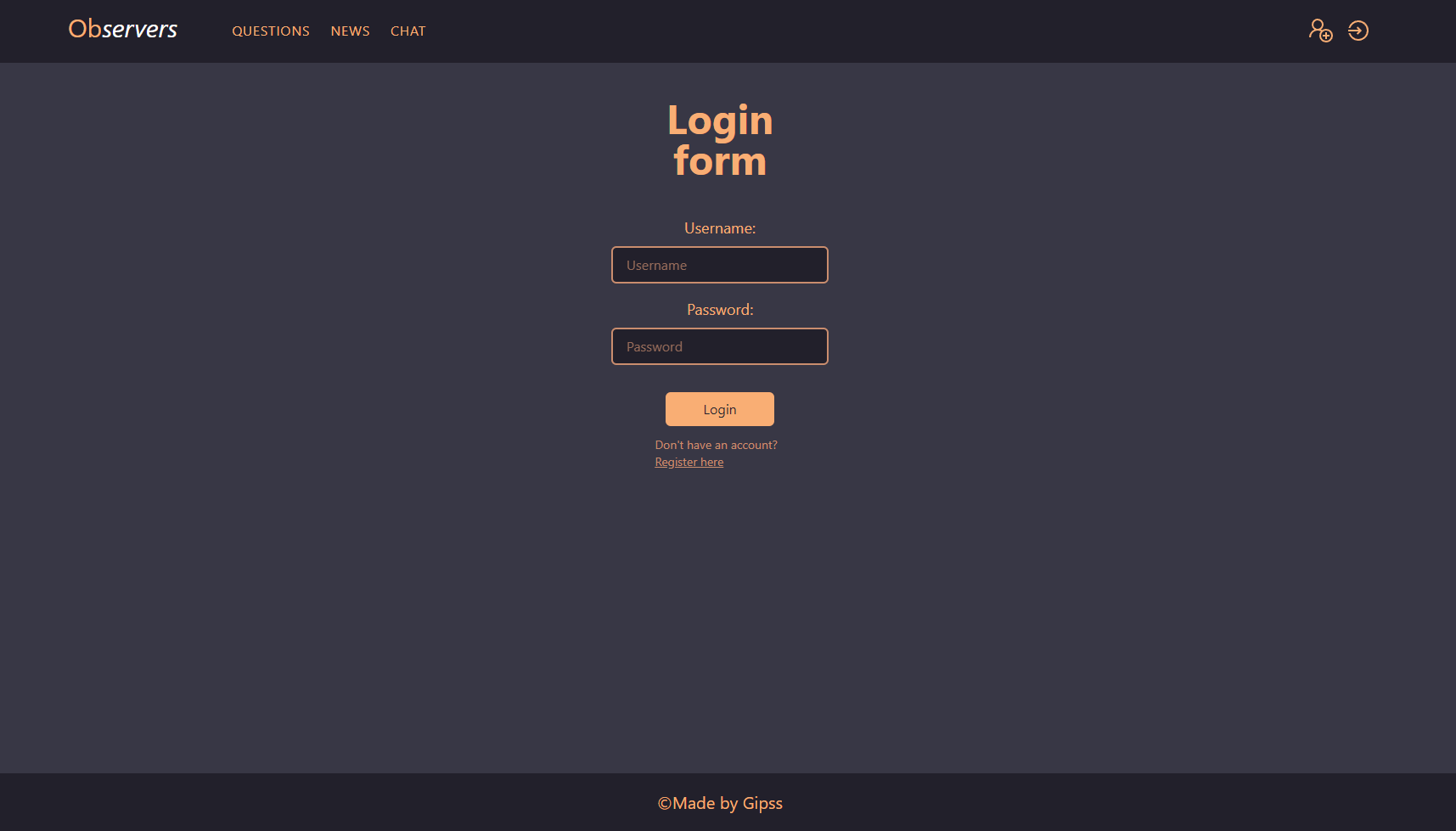


Рис. 3.3. Форма авторизації.

Після того, як користувач увійшов до свого акаунту, він може переглянути і відредагувати профіль (Рисунок 3.4.) за необхідності.

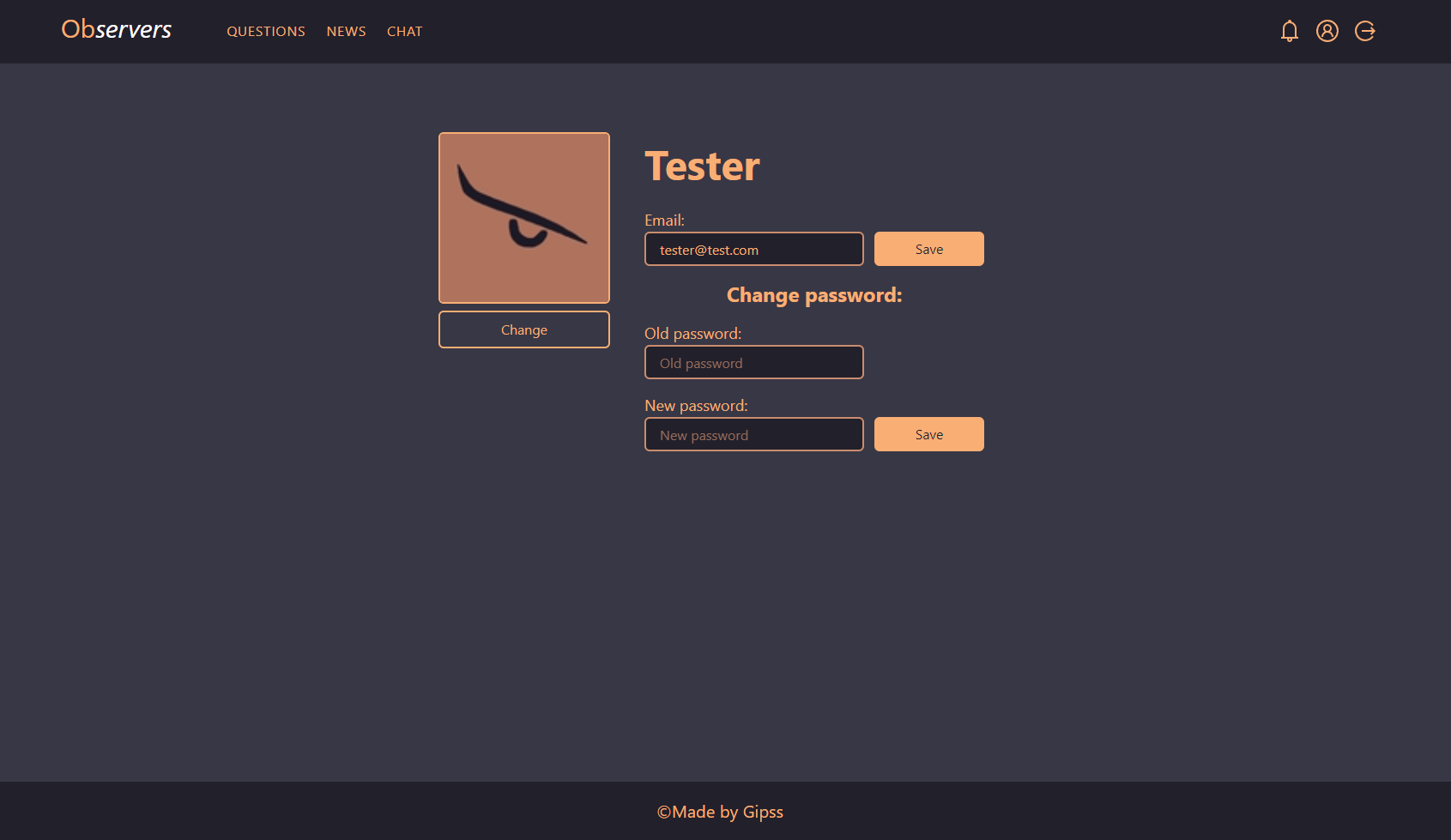


Рис. 3.4. Профіль користувача.

На сторінці профіля користувач може змінюватися фотографію профіля, пошту і пароль.

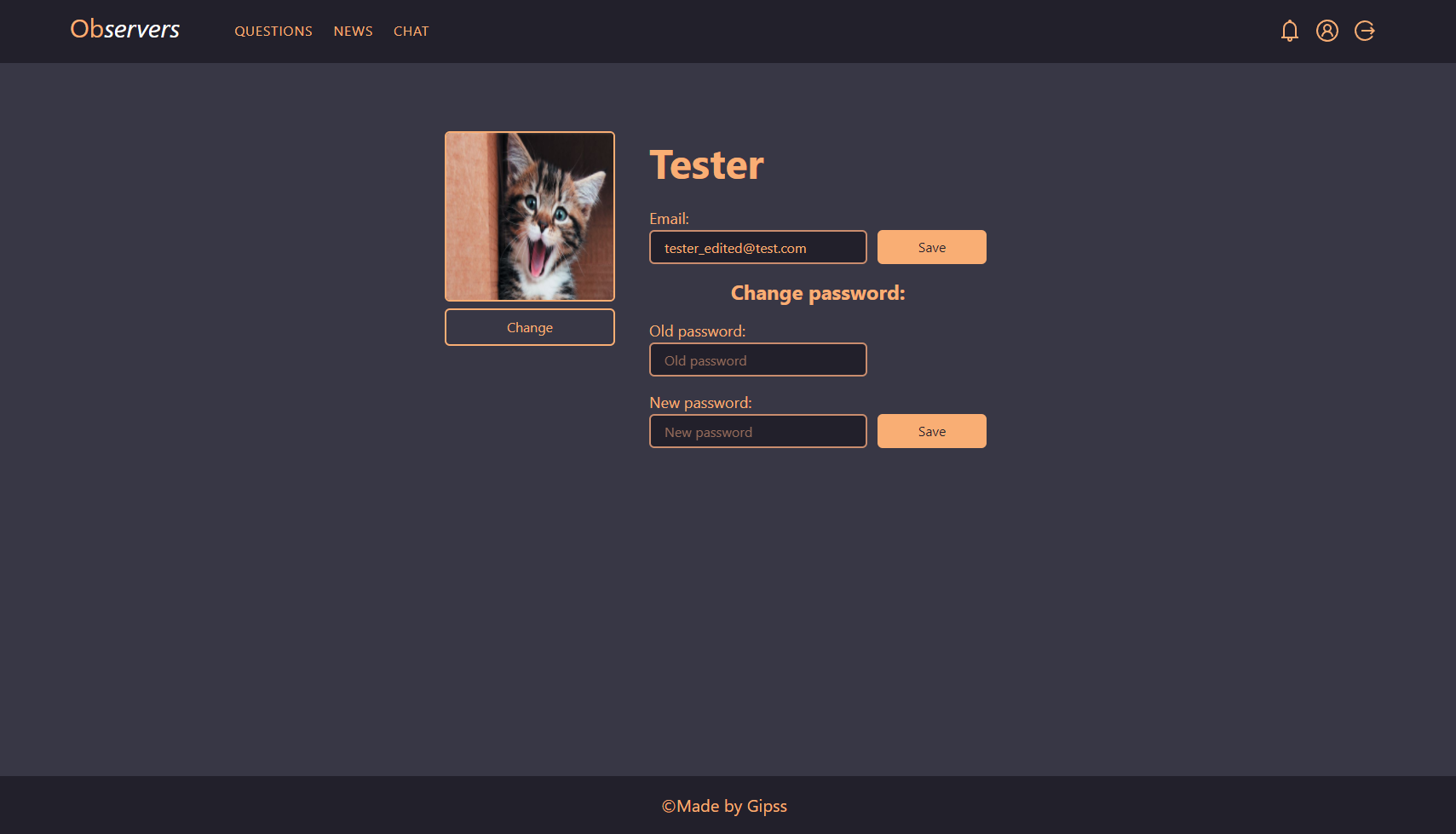


Рис. 3.5. Профіль користувача зі зміненою інформацією.

Якщо перейти на вкладку запитань, користувач побачить перелік усіх питань, які створювали відвідувачі сайту (Рисунок 3.6.).

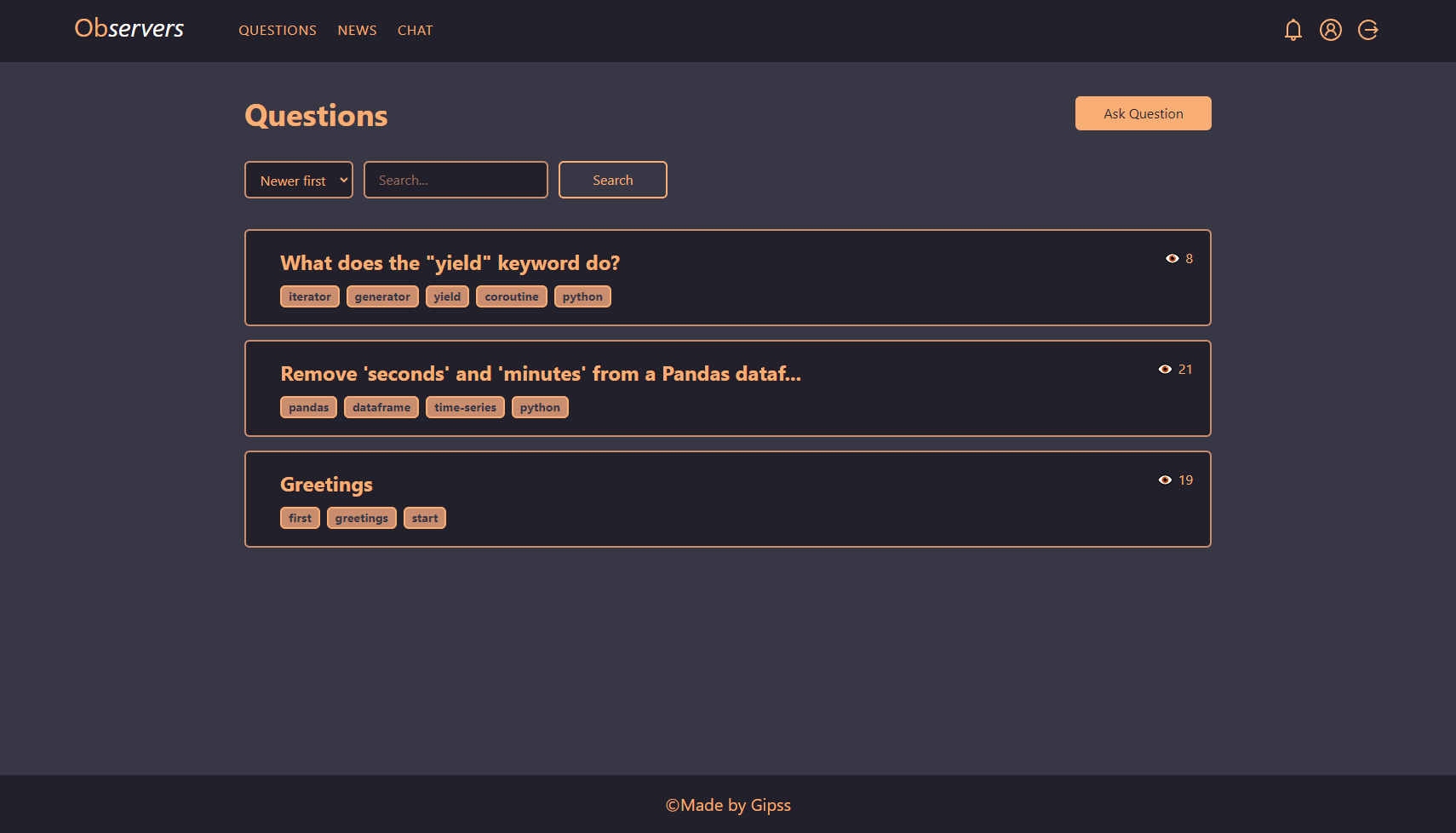


Рис. 3.6. Перелік запитань.

У авторизованого користувача є можливість створювати запитання. При натисканні на кнопку, відкривається форма створення (Рисунок 3.7.).

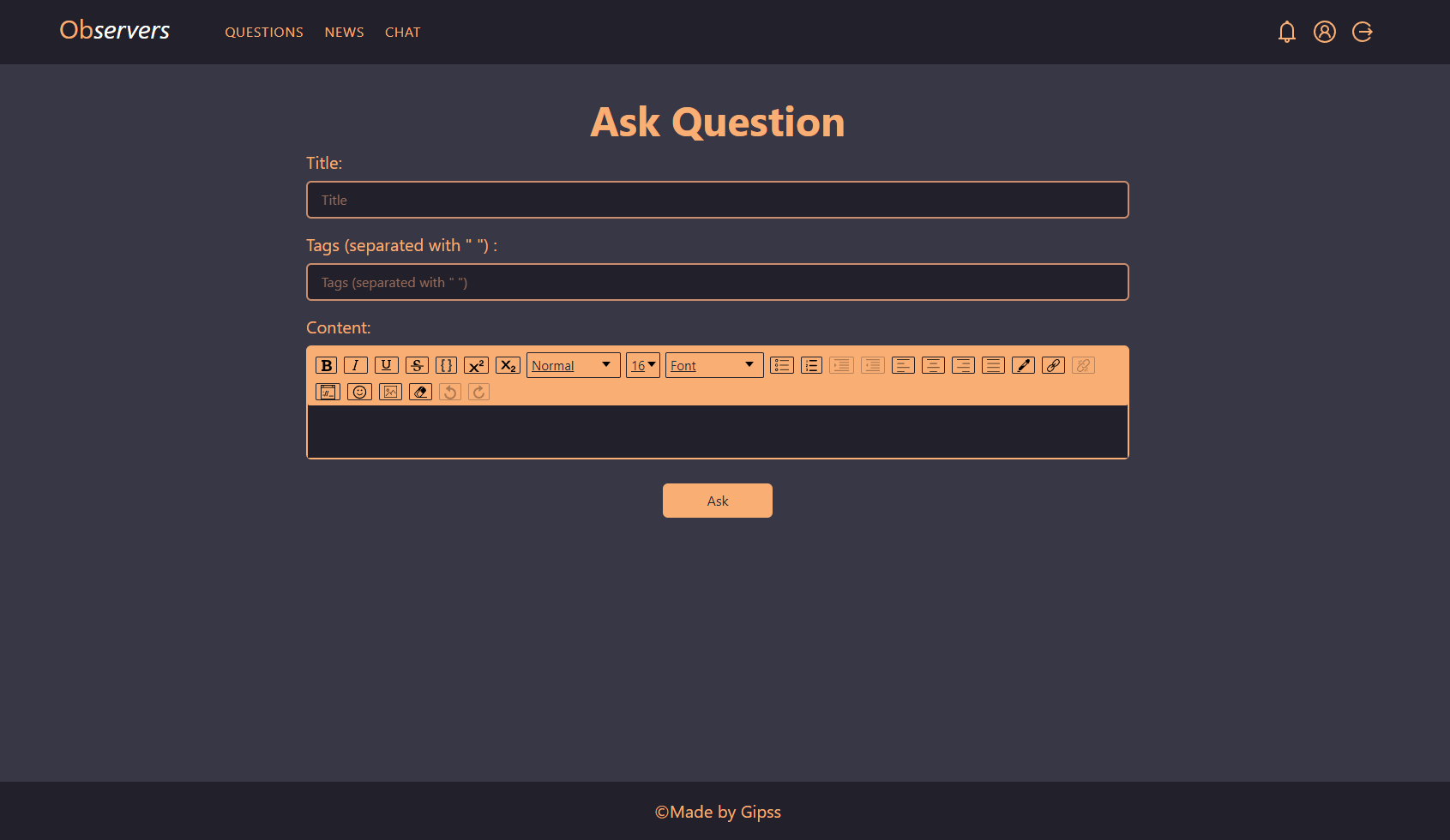


Рис. 3.7. Форма створення запитання.

Після створення запитання користувача переміщує на сторінку запитань, де вже з’явилося його запитання.

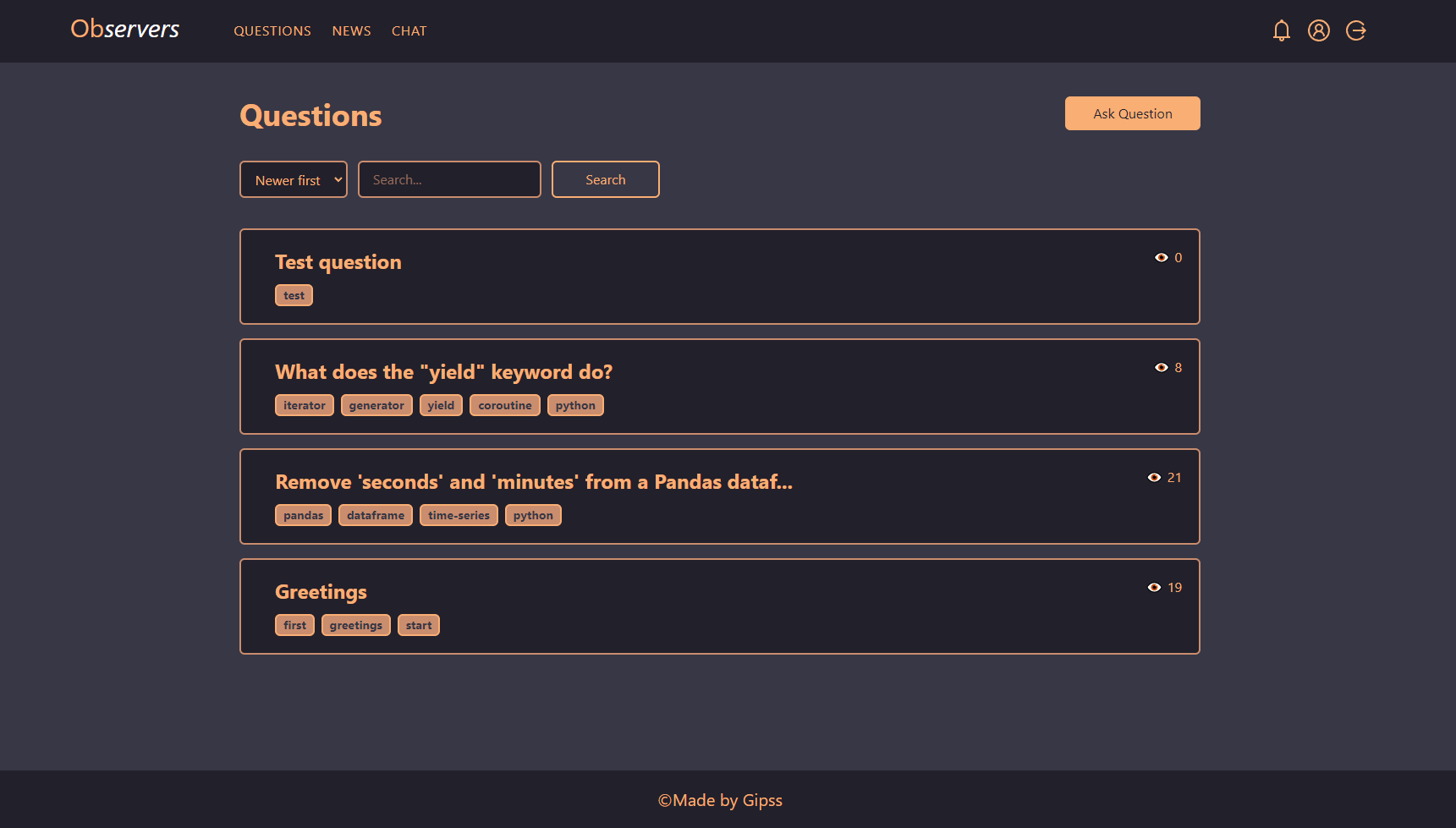


Рис. 3.8. Запитання додалося до переліку.

Якщо натиснути на запитання, відкриється його сторінка, де можна переглянути вміст запитання, коментарі, дату створення і кількість переглядів. Також, користувач зможе створити коментар і він відобразиться у списку коментарів.

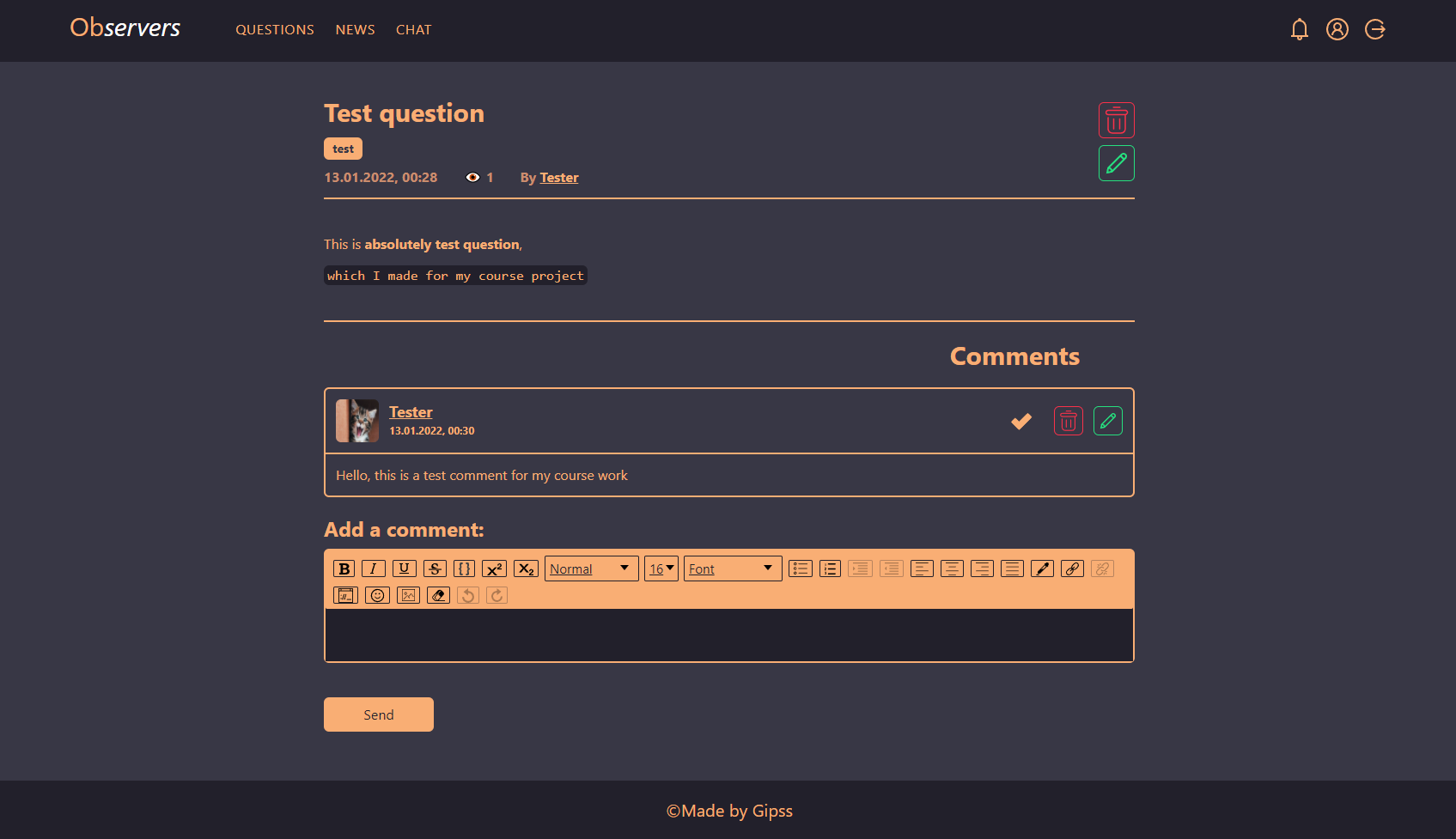


Рис. 3.9. Сторінка запитання.

У адміністратора і автора запитання є можливість видалити і редагувати питання. При натиснені на іконку видалення з’являється модальне вікно з підтвердженням дії (Рисунок 3.10.), а при натисненні на іконку редагування користувач переміщується на окрему сторінку, де він може редагувати запитання (Рисунки 3.11. та 3.12.).

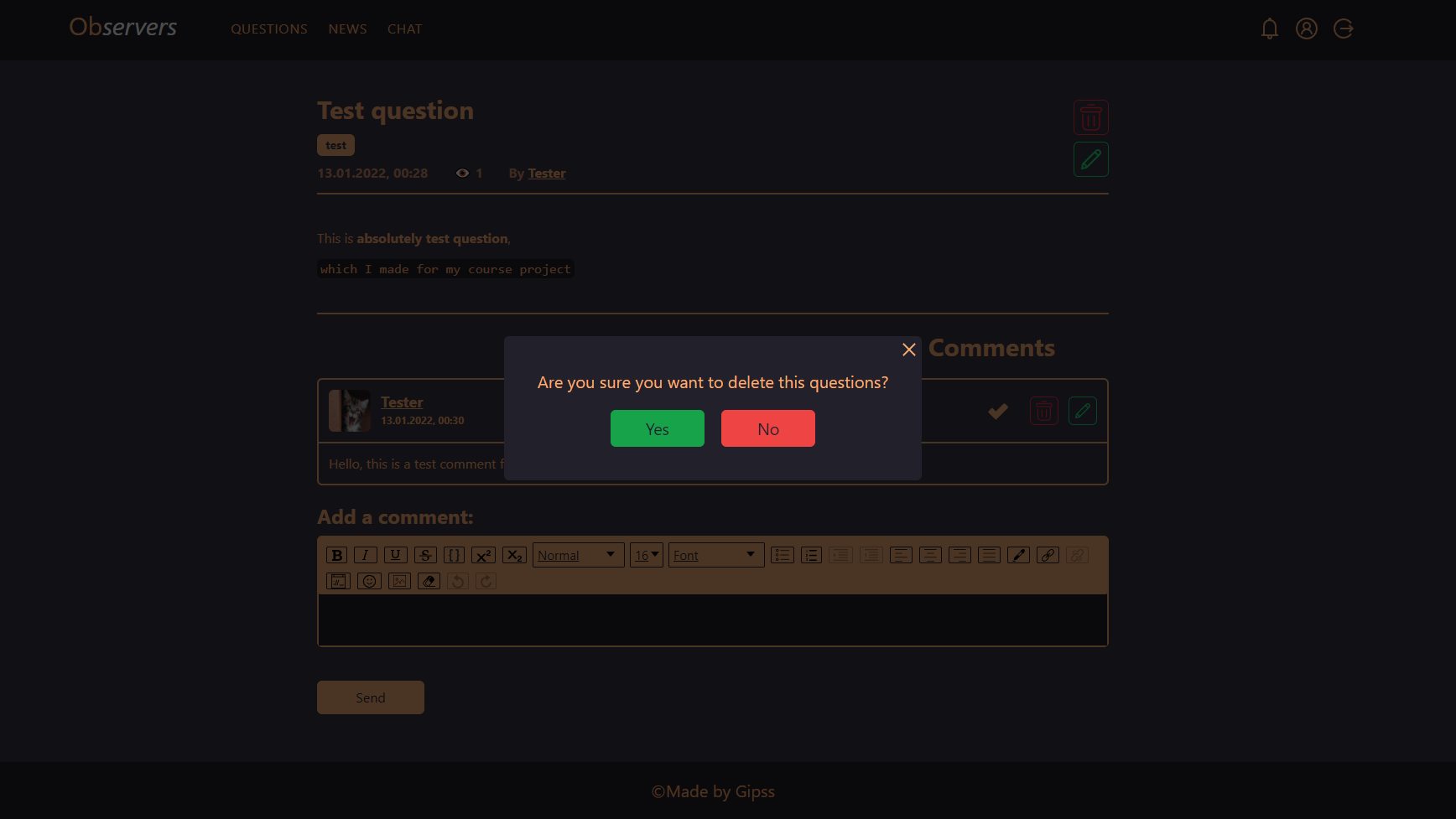


Рис. 3.10. Модально вікно підтвердження.

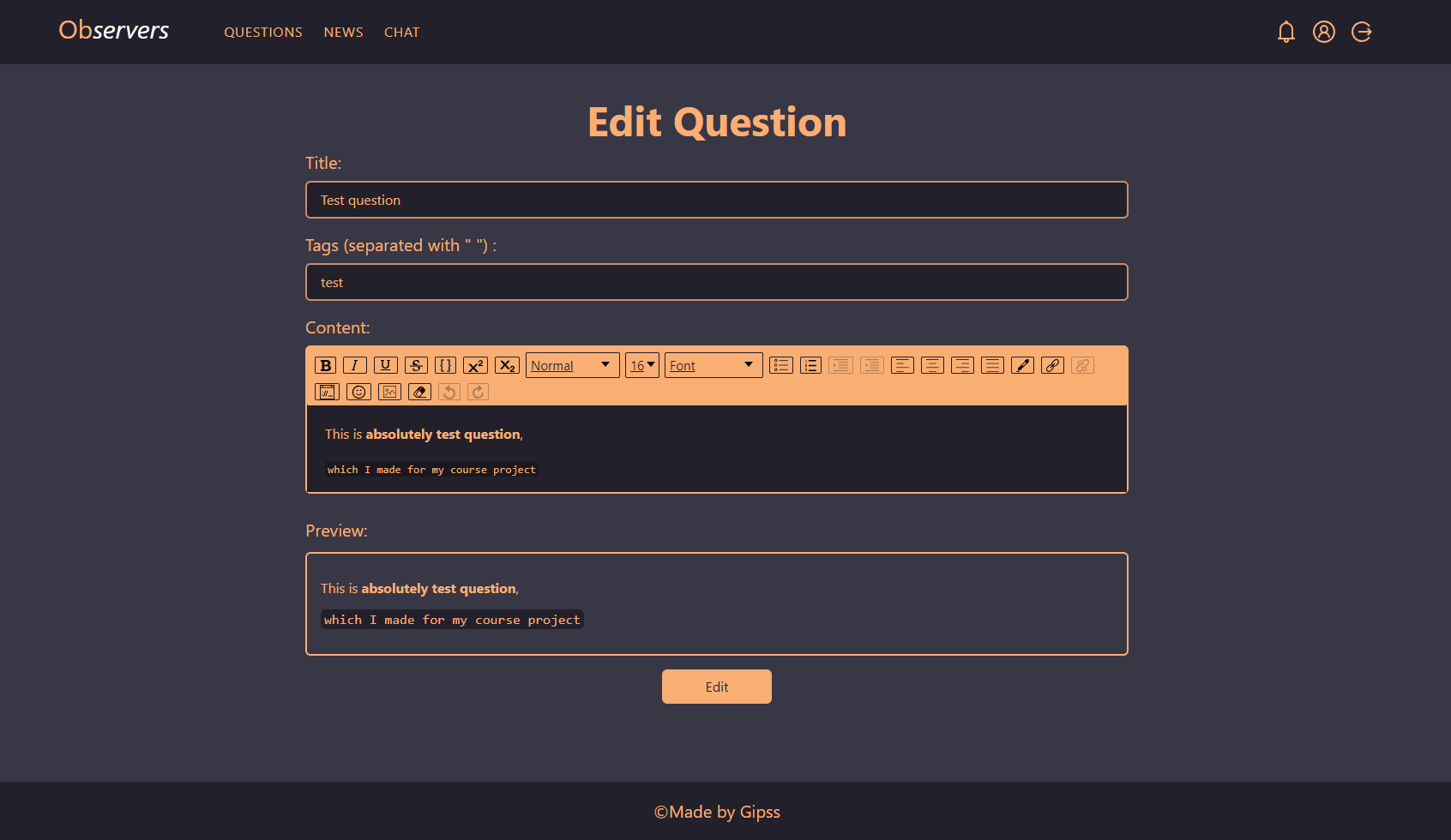


Рис. 3.11. Форма редагування запитання.

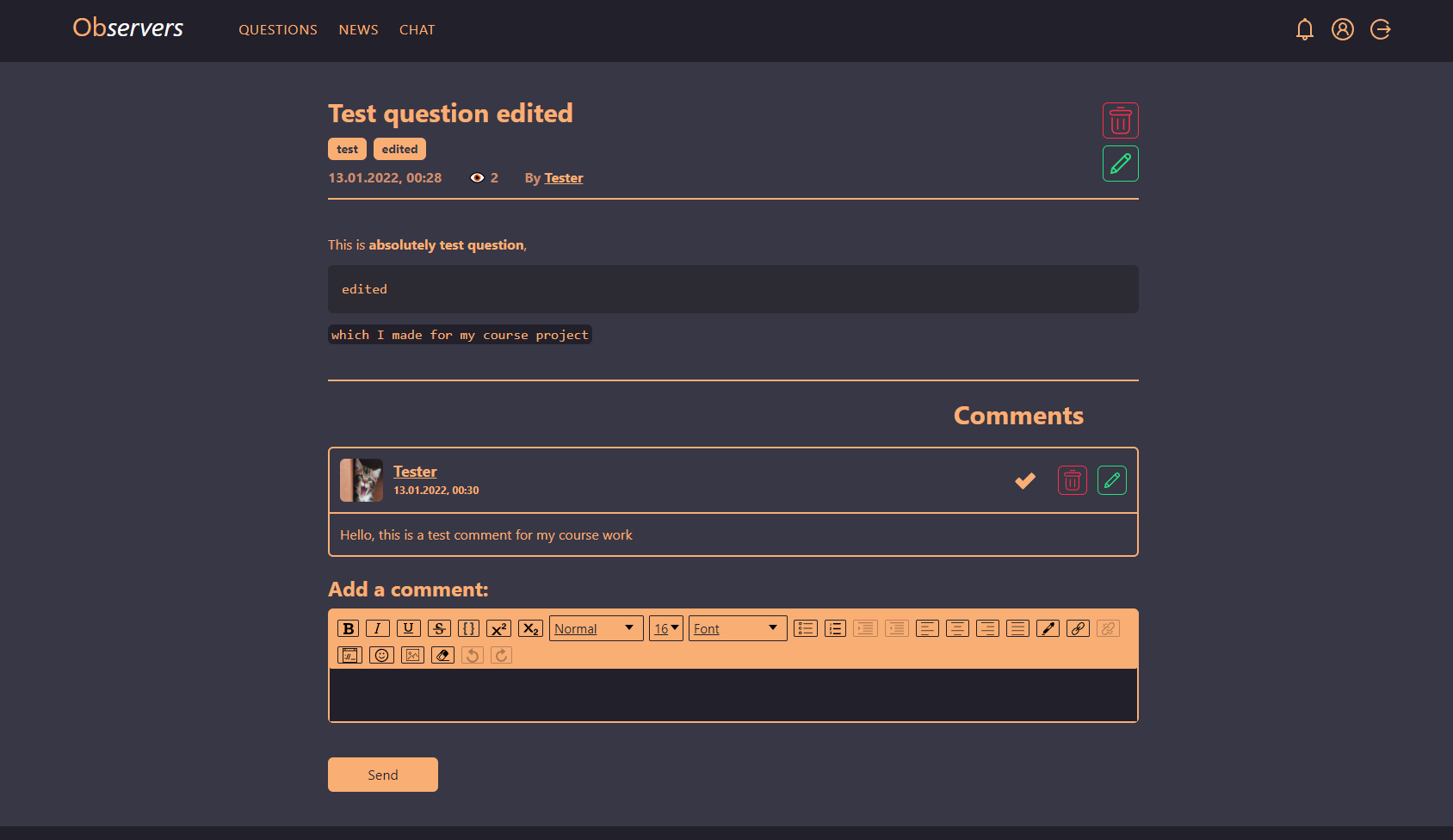


Рис. 3.12. Відредаговане запитання.

При переході на сторінку новин, перед користувачем з’являється список всіх опублікованих новин. Звичайний відвідувач може тільки переглядати і оцінювати новини (Рисунки 3.13. і 3.14.).



Рис. 3.13. Сторінка новин.

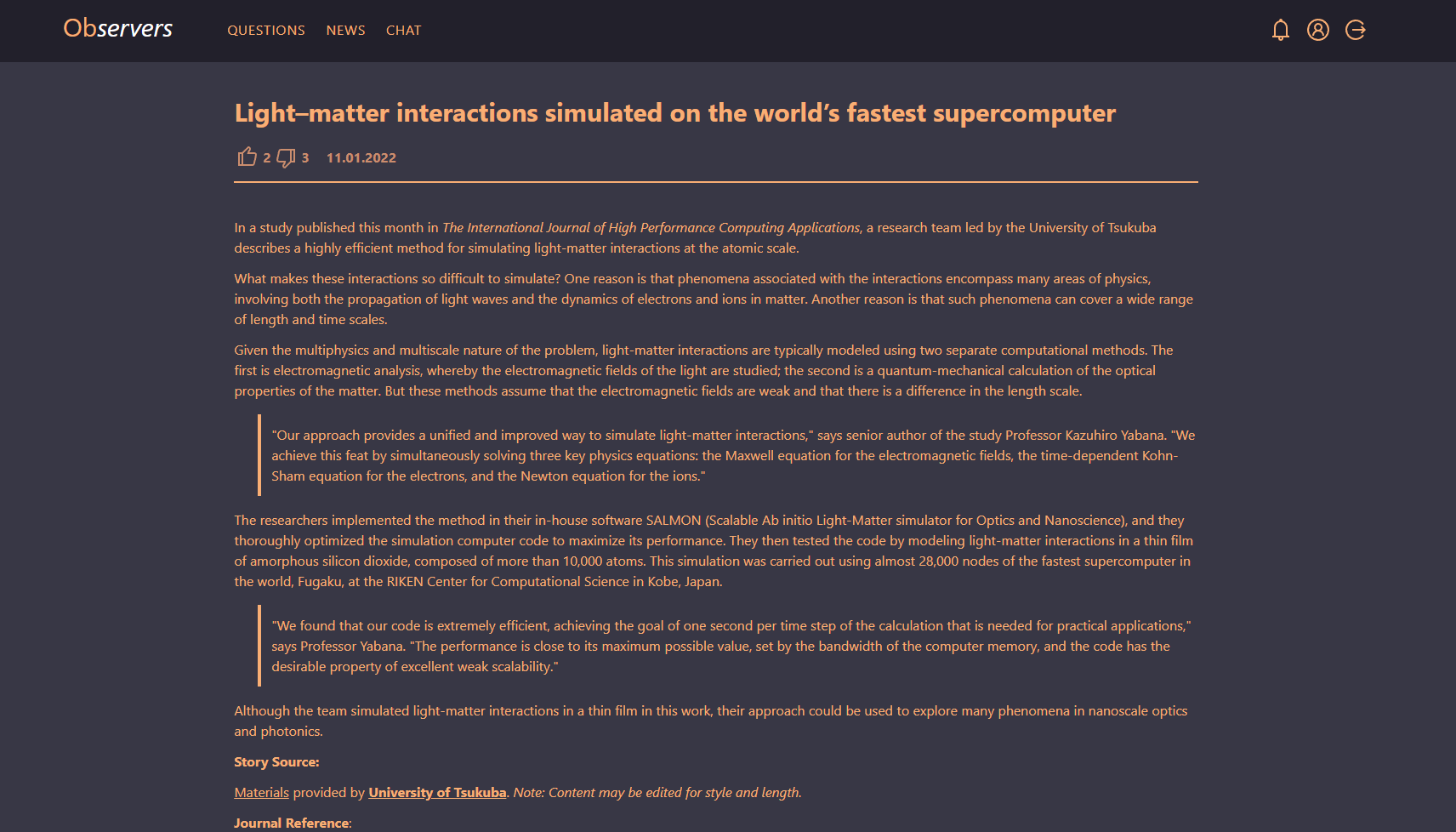


Рис. 3.14. Сторінка однієї новини.

При переході на сторінку чату, користувач може спілкуватися з іншими відвідувачами сайту. Також, при приєднуванні і від’єднуванні користувачів відображається відповідне повідомлення (Рисунки 3.15. і 3.16.).

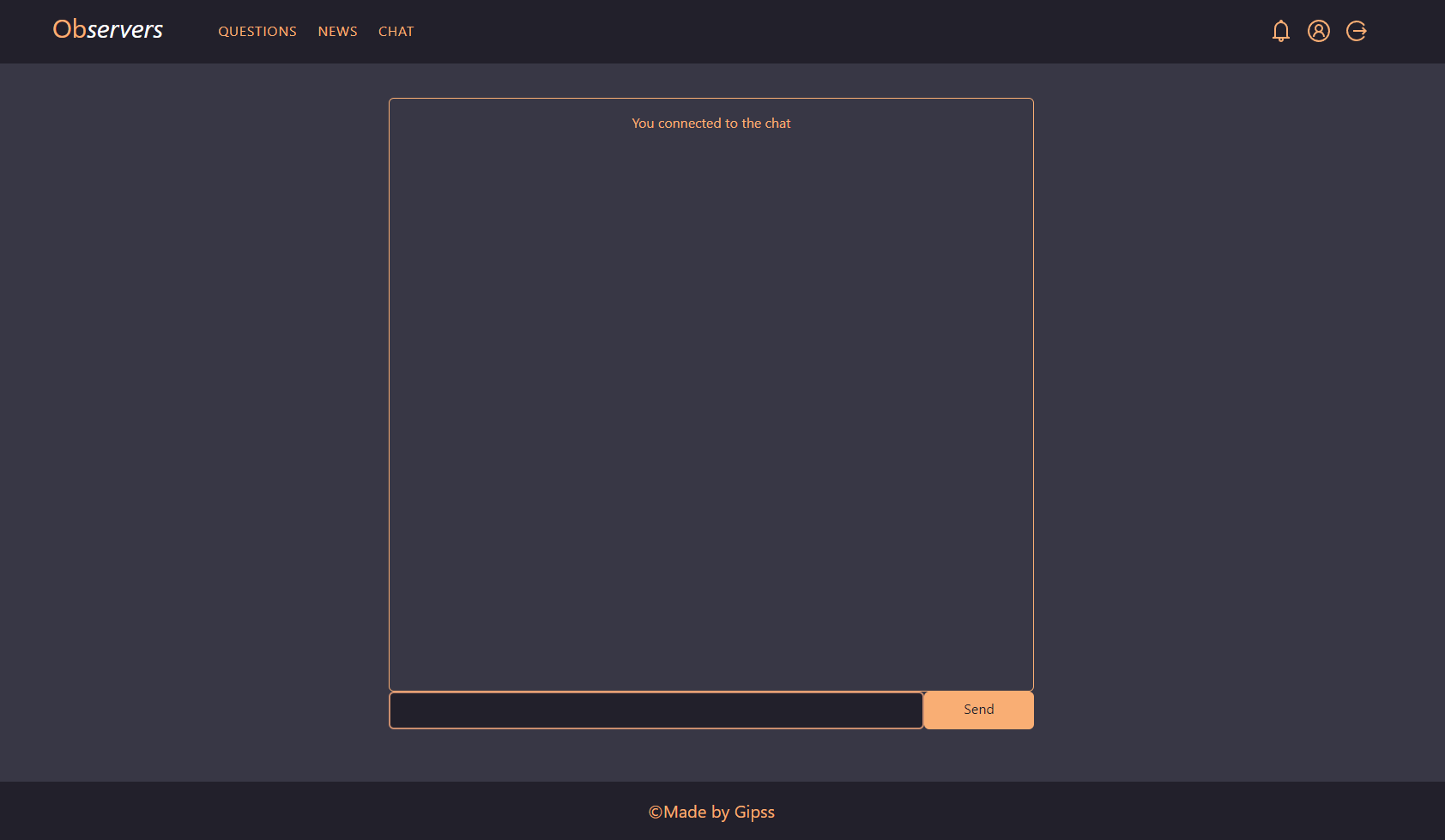


Рис. 3.15. Поточний користувач приєднався до чату.

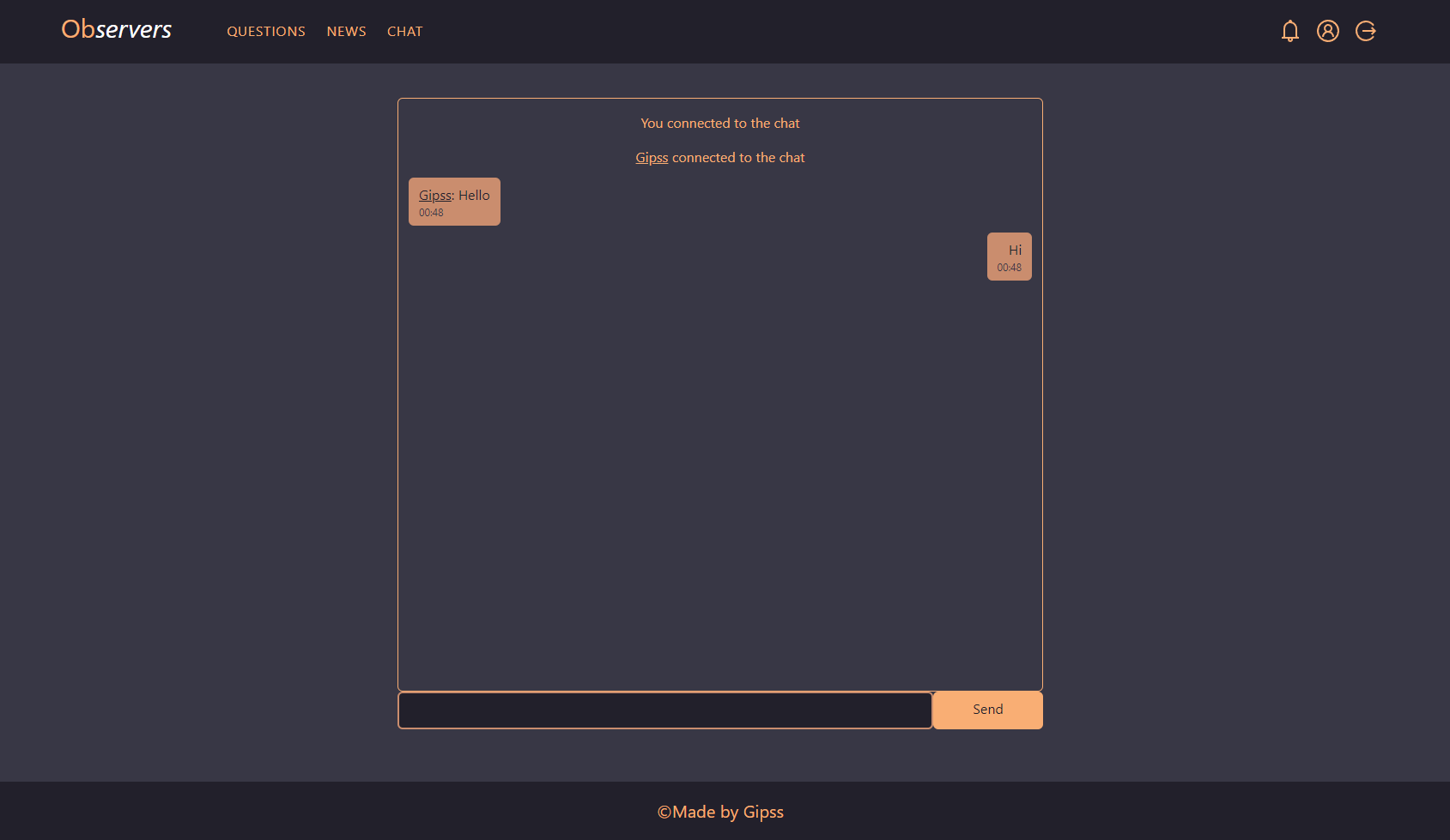


Рис. 3.16. Спілкування користувачів.

У адміністратора є можливість зайти в панель адміністратора, де він може редагувати та видаляти різні об’єкти сайту (наприклад, користувачів, теги, запитання, новини тощо).



Рис. 3.17. Панель адміністратора – користувачі.

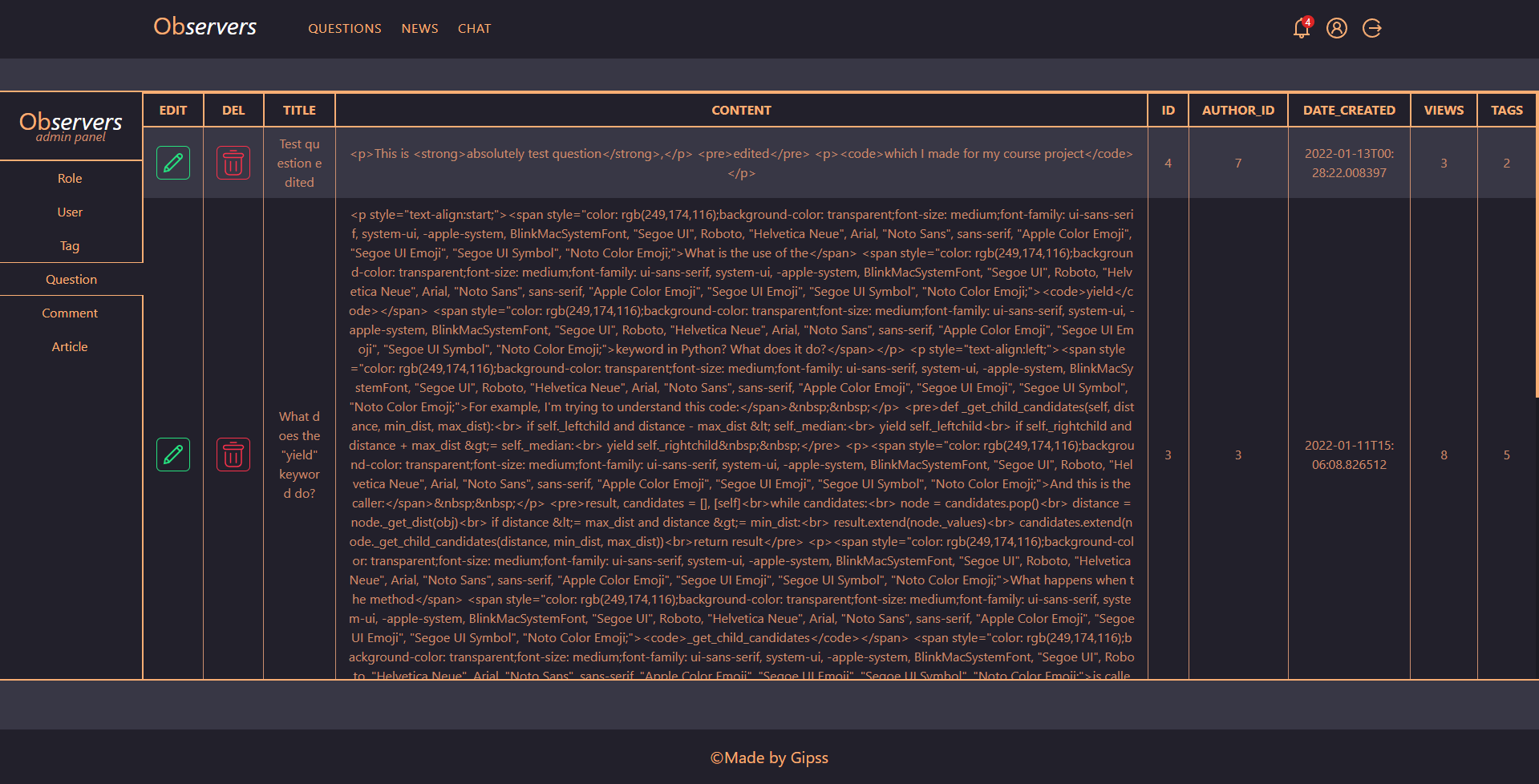


Рис. 3.18. Панель адміністратора – запитання.

## **Тестування роботи програмного забезпечення**

Веб-сайт передбачає помилки при введені даних користувачем і відображає їх на екрані. Це забезпечує коректну роботу веб-додатку і покращує досвід користування сайтом. Наприклад, у формі реєстрації відбувається ряд перевірок на валідність даних, а саме на те, чи присутні дані, чи вони відповідають необхідному формату.

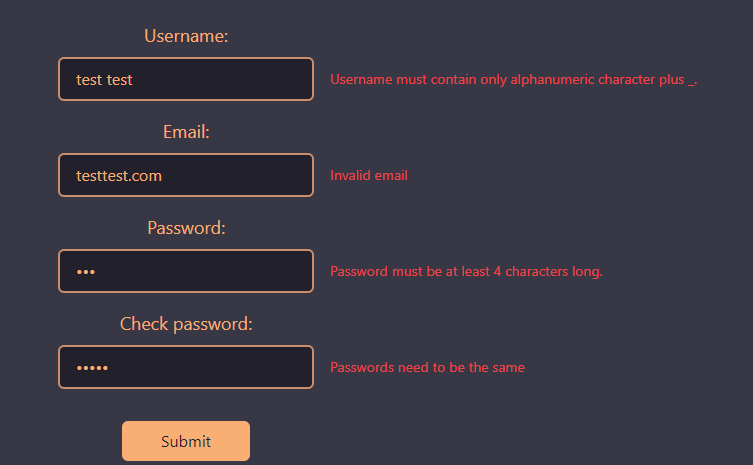


Рис. 3..19. Валідація форми реєстрації.

У випадку, якщо вказане ім’я користувача вже зайняте в системі, веб-сайт сповіщує про це.

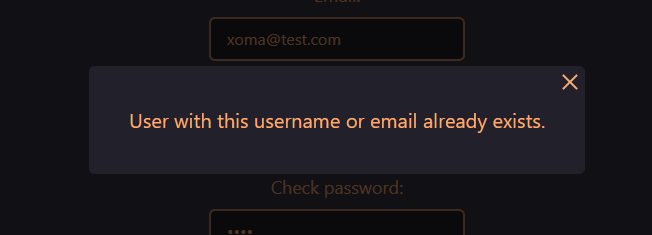


Рис. 3.20. Перевірка на унікальність логіну і пошти.

Якщо при авторизації ввести неправильні дані, сайт сповістить про це за допомогою модального вікна.

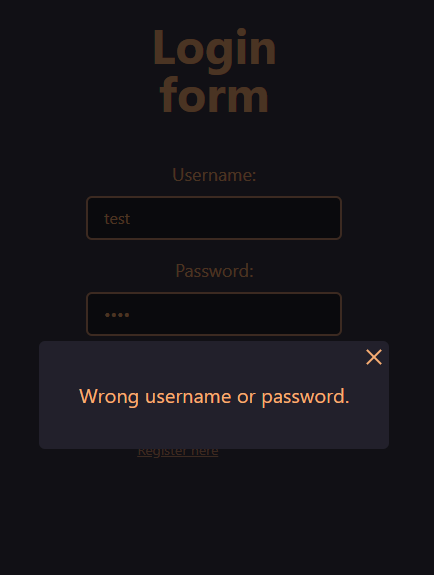


Рис. 3.21. Перевірка на правильність даних при авторизації.

При зміні пошти в акаунті також відбувається перевірка на унікальність.

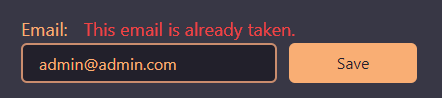


Рис. 3.22. Перевірка на пошти на унікальність.

При створені і редагуванні запитання відбувається перевірка на довжину введених даних для того, щоб частково уникнути публікування безкорисної інформації.

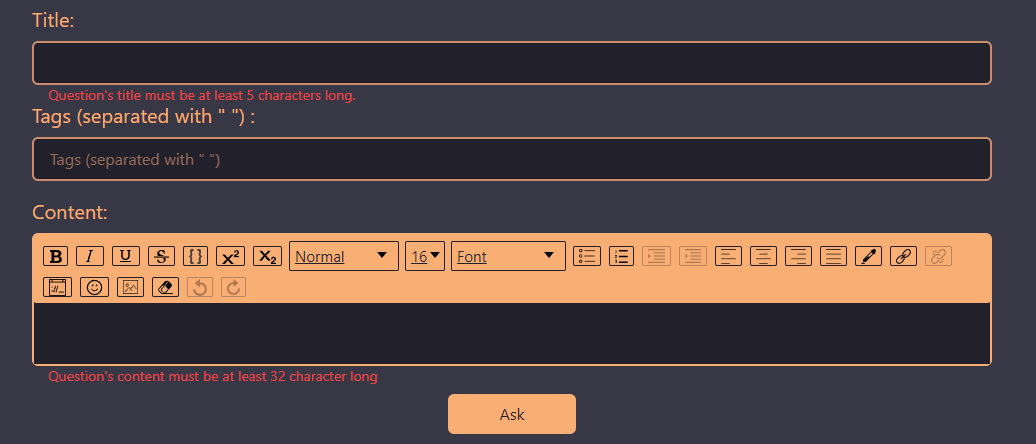


Рис. 3.23. Перевірка на довжину даних запитання.

При створені новини відбувається аналогічна перевірка.

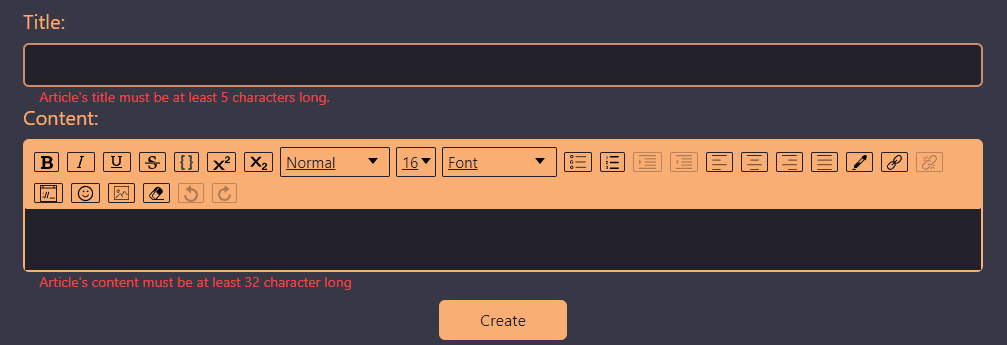


Рис. 3.24. Перевірка на довжину даних новини.

## **Висновки до 3 розділу**

У ході виконання третього розділу було повністю описано роботу веб-додатку, інтерфейс користувача та дії, які він може робити. Також, була описана перевірка інформації, яку вводить користувач, що сприяло коректній роботі програми і покращенню користувацького досвіду.

# **ВИСНОВКИ**

В першому розділі курсової роботи було описано постановку завдання, а також методи і способи його вирішення. Було обрано певні інструменти для створення веб-сайту. Також, був проведений аналіз аналогічних веб-додатків. Це дозволило виділити основні моменти, на які необхідно звернути увагу, і сформувати уявлення про майбутні функціонал і структуру сайту.

В другому розділі курсової роботи було спроектовано діаграму таблиць бази даних і основні алгоритми роботи веб-додатку. Також, було створено структуру проекту, розібрано основний принцип роботи сайту і його ключових ділянок коду.

В третьому розділі курсової роботи було детально описано роботу веб-сайту й інтерфейс користувач. Також, було розібрано які можливості має звичайний користувач та адміністратор. Було проведено ряд тестувань, для перевірки коректної роботи веб-додатку, оскільки це є важливою умовою для покращення досвіду користування сайтом.

В результаті виконання курсової роботи було створено форум для програмістів, який відповідає запланованому функціоналу, що був поставлений на початку проектування. Було використано найкращі практики і напрацювання з аналогічних сайтів, що зробило веб-додаток дуже зручним, зрозумілим і простим у користуванні.

# **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Документація FastAPI [Електронний ресурс] – Режим доступу: <https://fastapi.tiangolo.com/>
2. Документація ReactJS [Електронний ресурс] – Режим доступу: <https://ru.reactjs.org/>
3. Документація Tailwind [Електронний ресурс] – Режим доступу: <https://tailwindcss.com/docs>
4. Документація Redux [Електронний ресурс] – Режим доступу: <https://redux.js.org/>
5. Документація SQLAlchemy [Електронний ресурс] – Режим доступу: <https://docs.sqlalchemy.org/en/14/>
6. Л. Рамальо. Fluent Python, 2nd Edition, 2021. – 324 с.
7. Документація Pydantic [Електронний ресурс] – Режим доступу: <https://pydantic-docs.helpmanual.io/>
8. Канал на YouTube Ulbi TV [Електронний ресурс] – Режим доступу: <https://www.youtube.com/c/UlbiTV>
9. Ч. Сваруп. A bite of Python, 2020. – 164 с.
10. Документація Python [Електронний ресурс] – Режим доступу: <https://docs.python.org/3/>

# **ДОДАТКИ**

Додаток А

**Програмний код:**

main.py

**from** fastapi **import** FastAPI  
**from** fastapi.middleware.cors **import** CORSMiddleware  
**from** sqlalchemy.orm.session **import** Session  
  
**from** database **import** models  
**from** database.db **import** engine, SessionLocal  
**from** services **import** isemail  
**from** security **import** router **as** security\_router  
**from** security.hashing **import** get\_password\_hash  
**from** routers **import** router  
  
app = FastAPI()  
  
origins = [  
 **"\*"**]  
  
app.add\_middleware(  
 CORSMiddleware,  
 allow\_origins=origins,  
 allow\_credentials=**True**,  
 allow\_methods=[**"\*"**],  
 allow\_headers=[**"\*"**],  
)  
  
app.include\_router(router)  
app.include\_router(security\_router.router)  
  
  
@app.on\_event(**'startup'**)  
**def** startup():  
 models.Base.metadata.create\_all(bind=engine)  
 db: Session = SessionLocal()  
 roles = [role.title **for** role **in** db.query(models.Role).all()]  
 **if not** roles:  
 a = models.Role(title=**'Admin'**)  
 u = models.Role(title=**'User'**)  
 db.add\_all([a, u])  
  
 **if 'admin' not in** [user.username **for** user **in** db.query(models.User).all()]:  
 **while not** isemail(admin\_email := input(**'Choose admin\'s email: '**)):  
 print(**'Wrong email!'**)  
 admin\_password = get\_password\_hash(input(**'Choose admin\'s password: '**))  
 admin = models.User(  
 username=**'admin'**,  
 email=admin\_email,  
 password=admin\_password,  
 role\_id=1  
 )  
 db.add(admin)  
  
 db.commit()

routers.\_\_init\_\_.py

**from** fastapi **import** APIRouter  
  
**from** . **import** users, forum, news, chat  
  
router = APIRouter(prefix=**'/api'**)  
  
router.include\_router(users.router)  
router.include\_router(forum.router)  
router.include\_router(news.router)  
router.include\_router(chat.router)

forum.\_\_init\_\_.py

**from** fastapi **import** APIRouter  
  
**from** . **import** questions, tags, comments  
  
router = APIRouter(prefix=**'/forum'**)  
  
router.include\_router(questions.router)  
router.include\_router(tags.router)  
router.include\_router(comments.router)

question.py

**from** typing **import** Union  
**from** fastapi **import** APIRouter, Depends, Response, status, HTTPException  
**from** sqlalchemy.orm.session **import** Session  
  
**from** database **import** crud, models, schemas  
**from** dependencies **import** get\_db, get\_current\_user  
  
router = APIRouter(prefix=**'/questions'**, tags=[**'questions'**])  
  
  
@router.post(**'/'**, response\_model=schemas.Question)  
**def** create\_question(  
 question: schemas.QuestionCreate,  
 db: Session = Depends(get\_db),  
 current\_user: schemas.User = Depends(get\_current\_user)  
) -> models.Question:  
 *"""Creates a `Question` object with a given `QuestionCreate` schema.  
  
 Args:  
 `question` (schemas.QuestionCreate): A `schemas.QuestionCreate` object.  
 `db` (Session, optional): Database connection.  
  
 Returns:  
 `models.Question`: `Question` object.  
 """* **if** question.author\_id **is None**:  
 question.author\_id = current\_user.id  
  
 **return** crud.create\_question(db=db, question=question)  
  
  
@router.get(**'/'**, response\_model=list[schemas.Question])  
**def** get\_questions(  
 skip: int = 0,  
 limit: int = 100,  
 by\_title: str = **None**,  
 order\_by\_date: str = **None**,  
 order\_by\_views: bool = **None**,  
 db: Session = Depends(get\_db)  
) -> list[models.Question]:  
 *"""Gets all `Questions` from database in range [`skip`:`skip+limit`] and returns them to the client.  
  
 Args:  
 `skip` (Optional[int], optional): How many objects to skip. Defaults to 0.  
 `limit` (Optional[int], optional): Maximum amount of objects. Defaults to 100.  
 `by\_title` (Optional[str], optional): Title to get by. Defaults to None.  
 `order\_by\_date` (Optional[str], optional): Method of sorting by date. Defaults to None.  
 `order\_by\_views` (Optional[bool], optional): To sort by views or not. Defaults to None.  
 `db` (Session, optional): Database connection.  
  
 Returns:  
 `list[models.Question]`: A `list` of all `Question` objects.  
 """* **if** by\_title:  
 **return** crud.get\_questions\_by\_title(db=db, title=by\_title)  
  
 **if** order\_by\_date:  
 **if** order\_by\_date.lower() == **'desc'**:  
 **return** crud.get\_objects(  
 cls=models.Question,  
 db=db,  
 skip=skip,  
 limit=limit,  
 order\_by=models.Question.date\_created.desc()  
 )  
 **elif** order\_by\_date.lower() == **'asc'**:  
 **return** crud.get\_objects(  
 cls=models.Question,  
 db=db,  
 skip=skip,  
 limit=limit,  
 order\_by=models.Question.date\_created.asc()  
 )  
 **elif** order\_by\_views:  
 **return** crud.get\_objects(  
 cls=models.Question,  
 db=db,  
 skip=skip,  
 limit=limit,  
 order\_by=models.Question.views.desc()  
 )  
 **return** crud.get\_objects(cls=models.Question, db=db, skip=skip, limit=limit)  
  
  
@router.get(**'/{question\_title}/title/'**, response\_model=list[schemas.Question])  
**def** get\_questions\_by\_title(question\_title: str, db: Session = Depends(get\_db)) -> list[models.Question]:  
 **return** crud.get\_questions\_by\_title(db=db, title=question\_title)  
  
  
@router.get(**'/{question\_id}/'**, response\_model=schemas.Question)  
**def** get\_question(question\_id: int, db: Session = Depends(get\_db)) -> models.Question:  
 *"""Gets `Question` object by `question\_key`.  
  
 Args:  
 `question\_id` (int): `Question` object's id.  
 `db` (Session, optional): Database connection.  
  
 Returns:  
 `models.Question`: `Question` object.  
 """* **return** crud.get\_object(cls=models.Question, db=db, object\_id=question\_id)  
  
  
@router.get(**'/{user\_id}/user/'**, response\_model=list[schemas.Question])  
**def** get\_questions\_by\_user(user\_id: int, db: Session = Depends(get\_db)) -> list[models.Question]:  
 *"""Returns a list of user's questions by `user\_id`.  
  
 Args:  
 `user\_id` (int): User's id.  
 `db` (): Database connection.  
  
 Returns:  
 `list[models.Question]`: List of `Question` objects.  
 """* **return** crud.get\_objects\_by\_expression(  
 cls=models.Question,  
 db=db,  
 expression=(models.Question.author\_id == user\_id),  
 )  
  
  
@router.patch(**'/{question\_id}/'**, response\_model=schemas.Question)  
**def** update\_question(  
 question\_id: int,  
 question: schemas.QuestionUpdate,  
 db: Session = Depends(get\_db),  
 current\_user: schemas.User = Depends(get\_current\_user)  
) -> models.Question:  
 *"""Updates `Question` object by `question\_id`.  
  
 Args:  
 `question\_id` (int): `Question` object's id.  
 `question` (schemas.QuestionUpdate): `QuestionUpdate` schema.  
 `db` (Session, optional): Database connection.  
  
 Returns:  
 `models.Question`: `Question` object.  
 """* **if** current\_user.role.title != **'Admin' and** \  
 current\_user.id != crud.get\_object(cls=models.Question, db=db, object\_id=question\_id).author.id:  
 **raise** HTTPException(  
 status\_code=status.HTTP\_400\_BAD\_REQUEST,  
 detail=**'You did not give the views parameter.'** )  
  
 **return** crud.update\_question(db=db, question\_id=question\_id, question=question)  
  
  
@router.patch(**'/{question\_id}/views/'**)  
**def** update\_question\_views(question\_id: int, views: int, db: Session = Depends(get\_db)) -> models.Question:  
 *"""Updates question views by a given `question\_id`.  
  
 Args:  
 `question\_id` (int):  
 `views` (int):  
 `db` (Session, optional): Database connection.  
  
 Returns:  
 `models.Question`: `Question` object.  
 """* **return** crud.update\_question(  
 db=db, question\_id=question\_id, question=schemas.QuestionUpdate(views=views)  
 )  
  
  
@router.delete(**'/{question\_id}/'**)  
**def** delete\_question(  
 question\_id: int,  
 db: Session = Depends(get\_db),  
 current\_user: schemas.User = Depends(get\_current\_user)  
) -> Response:  
 *"""Deletes a question by a given `question\_id`.  
  
 Args:  
 `question\_id` (int): `Question`'s id.  
 `db` (Session, optional): Database connection.  
  
 Raises:  
 `HTTPException`: If an invalid `question\_id` was given.  
  
 Returns:  
 `Response`: No content response.  
 """* **if** current\_user.role.title != **'Admin' and**\  
 current\_user.id != crud.get\_object(cls=models.Question, db=db, object\_id=question\_id).author.id:  
 **raise** HTTPException(  
 status\_code=status.HTTP\_403\_FORBIDDEN,  
 detail=**'You are not the owner of the question.'** )  
  
 crud.delete\_object(cls=models.Question, db=db, object\_id=question\_id)  
 **return** Response(status\_code=status.HTTP\_204\_NO\_CONTENT)

security\_token.py

**import** os  
**import** secrets  
**from** datetime **import** datetime, timedelta  
**from** typing **import** Optional  
**from** dotenv **import** load\_dotenv  
  
**from** fastapi.security **import** OAuth2PasswordBearer  
**from** jose **import** jwt  
  
load\_dotenv()  
SECRET\_KEY = os.environ.get(**'SECRET\_KEY'**) **or** secrets.token\_hex(64)  
ALGORITHM = **"HS256"**ACCESS\_TOKEN\_EXPIRE\_MINUTES = 60  
  
oauth2\_scheme = OAuth2PasswordBearer(tokenUrl=**"api/token"**)  
  
  
**def** create\_access\_token(data: dict, expires\_delta: Optional[timedelta] = **None**) -> str:  
 *"""Creates access token with a given `data` and `expires\_delta`.  
  
 Args:  
 `data` (dict): A `dict` where exists a key "sub" with value of username.  
 `expires\_delta` (Optional[timedelta], optional): Time of token expiration. Defaults to None.  
  
 Returns:  
 `str`: A Json Web Token.  
 """* to\_encode = data.copy()  
 **if** expires\_delta:  
 expire = datetime.utcnow() + expires\_delta  
 **else**:  
 expire = datetime.utcnow() + timedelta(minutes=15)  
 to\_encode.update({**"exp"**: expire})  
 encoded\_jwt = jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)  
 **return** encoded\_jwt

decorators.py

**from** functools **import** wraps  
  
**from** fastapi **import** HTTPException, status  
  
  
**def** raise\_403\_if\_not\_admin(func):  
 @wraps(func)  
 **def** wrapper(\*args, \*\*kwargs):  
 **if** (user := kwargs.get(**'current\_user'**)) **and** user.role.title != **'Admin'**:  
 **raise** HTTPException(  
 status\_code=status.HTTP\_403\_FORBIDDEN,  
 detail=**'You are not an admin.'** )  
 **return** func(\*args, \*\*kwargs)  
  
 **return** wrapper  
  
  
**def** raise\_403\_if\_no\_access(func):  
 @wraps(func)  
 **def** wrapper(\*args, \*\*kwargs):  
 **if** (user := kwargs.get(**'current\_user'**)) \  
 **and** user.role.title != **'Admin'** \  
 **and** user.id != kwargs.get(**'user\_id'**):  
 **raise** HTTPException(  
 status\_code=status.HTTP\_403\_FORBIDDEN,  
 detail=**'You are not that user.'** )  
 **return** func(\*args, \*\*kwargs)  
  
 **return** wrapper

dependencies.py

**from** fastapi **import** Depends, HTTPException, status, WebSocket, Query  
**from** jose **import** jwt, JWTError  
**from** sqlalchemy.orm **import** Session  
  
**from** database.db **import** SessionLocal  
**from** database **import** models  
**from** security **import** security\_token, schemas  
**from** services **import** get\_user\_by\_username\_or\_email  
  
  
**def** get\_db():  
 *"""Returns a database `Session`.  
  
 Yields:  
 `Iterator[SessionLocal]`: A database connection.  
 """* db = SessionLocal()  
 **try**:  
 **yield** db  
 **finally**:  
 db.close()  
  
  
**def** get\_user(user\_token: str, db: Session):  
 *"""Returns the current user if he passes authentication.  
  
 Args:  
 `user\_token` (str, optional): User's token.  
 `db` (Session, optional): Database connection.  
  
 Raises:  
 `HTTPException`: If there's no key `sub` in a given token.  
 `HTTPException`: In case of JWTError.  
 `HTTPException`: If there's no user with this username.  
  
 Returns:  
 models.User: A current user.  
 """* credentials\_exception = HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail=**"Could not validate credentials"**,  
 headers={**"WWW-Authenticate"**: **"Bearer"**},  
 )  
 **try**:  
 payload = jwt.decode(user\_token, security\_token.SECRET\_KEY, algorithms=[security\_token.ALGORITHM])  
 username: str = payload.get(**"sub"**)  
 **if** username **is None**:  
 **raise** credentials\_exception  
 token\_data = schemas.TokenData(username=username)  
 **except** JWTError:  
 **raise** credentials\_exception  
  
 user = get\_user\_by\_username\_or\_email(db=db, username=token\_data.username)  
  
 **if** user **is None**:  
 **raise** credentials\_exception  
 **return** user  
  
  
**async def** get\_current\_user(  
 user\_token: str = Depends(security\_token.oauth2\_scheme),  
 db: Session = Depends(get\_db)  
) -> models.User:  
 **return** get\_user(user\_token, db)  
  
  
**async def** get\_token\_in\_query(  
 websocket: WebSocket,  
 token: str = Query(...),  
):  
 **return** token.replace(**'Bearer '**, **''**)  
  
  
**async def** get\_current\_user\_by\_query(  
 user\_token: str = Depends(get\_token\_in\_query),  
 db: Session = Depends(get\_db)  
) -> models.User:  
 **return** get\_user(user\_token, db)

App.tsx

**import** React, {FC, *useEffect*} **from 'react'**;  
**import** *Header* **from "./components/Header/Header"**;  
**import** {*BrowserRouter*} **from "react-router-dom"**;  
**import** *Footer* **from "./components/Footer/Footer"**;  
**import** *AppRouter* **from "./components/AppRouter/AppRouter"**;  
**import** *Loader* **from "./components/Loader/Loader"**;  
**import** {***useTypedSelector***} **from "./hooks/useTypesSelector"**;  
**import** {*useDispatch*} **from "react-redux"**;  
**import** {*fetchUser*} **from "./store/action-creators/user"**;  
**import** {*fetchNotifications*} **from "./store/action-creators/notifications"**;  
  
**const** *App*: FC = () => {  
 **const** {user, loading, error} = ***useTypedSelector***(state => state.user);  
 **const** dispatch = *useDispatch*();  
  
 *useEffect*(() => {  
 **if**(!user)  
 dispatch(*fetchUser*());  
 **if** (user)  
 dispatch(*fetchNotifications*(user.**id**));  
  
 **if**(error && ***localStorage***.getItem(**'token'**)){  
 ***localStorage***.removeItem(**'token'**);  
 }  
 }, [user]);  
  
 **return**(  
 <**BrowserRouter**>  
 <**div className="app-container"**>  
 {  
 loading  
 ? <**Loader** />  
 : <>  
 <**Header** />  
 <**AppRouter** />  
 <**Footer** />  
 </>  
 }  
  
 </**div**>  
 </**BrowserRouter**>  
 );  
};  
  
**export default** *App*;

AppRouter.tsx

**import** React, {FC} **from 'react'**;  
**import** {***router***} **from "../../router/Router"**;  
**import** {*Route*, *Routes*} **from "react-router-dom"**;  
  
**const** *AppRouter*: FC = () => {  
 **return** (  
 <**div className="content"**>  
 <**Routes**>  
 {***router***.map(route => <**Route path=**{route.**path**} **element=**{route.**component**} **key=**{route.**path**}/> )}  
 </**Routes**>  
 </**div**>  
 );  
};  
  
**export default** *AppRouter*;

reducers.index.ts

**import** {*combineReducers*} **from "redux"**;  
**import** {*userReducer*} **from "./userReducer"**;  
**import** {*authReducer*} **from "./authReducer"**;  
**import** {*questionsReducer*} **from "./questionsReducer"**;  
**import** {*notificationsReducer*} **from "./notificationsReducer"**;  
**import** {*articlesReducer*} **from "./articlesReducer"**;  
**import** {*chatReducer*} **from "./chatReducer"**;  
**import** {*adminReducer*} **from "./adminReducer"**;  
**import** {*rolesReducer*} **from "./rolesReducer"**;  
**import** {*usersReducer*} **from "./usersReducer"**;  
**import** {*tagsReducer*} **from "./tagsReducer"**;  
**import** {*commentsReducer*} **from "./commentsReducer"**;  
  
**export const *rootReducer*** = *combineReducers*({  
 user: *userReducer*,  
 users: *usersReducer*,  
 auth: *authReducer*,  
 questions: *questionsReducer*,  
 notifications: *notificationsReducer*,  
 articles: *articlesReducer*,  
 chat: *chatReducer*,  
 admin: *adminReducer*,  
 roles: *rolesReducer*,  
 tags: *tagsReducer*,  
 comments: *commentsReducer*,  
});  
  
**export type** RootState = ReturnType<**typeof *rootReducer***>;

action-creators.questions.ts

**import** {***instance***} **from "../../Instance"**;  
**import** {IQuestion, ITag, IUser, QuestionsAction, QuestionsActionTypes} **from "../../types/types"**;  
**import** {Dispatch} **from "redux"**;  
**import** React **from "react"**;  
  
**export const** *fetchQuestions* = (params: **string** | **undefined**, path: **boolean** = **false**) => {  
 **return async** (dispatch: Dispatch<QuestionsAction>) => {  
 dispatch({**type**: QuestionsActionTypes.*FETCH\_QUESTIONS*});  
 ***instance***.get<IQuestion[]>(**`forum/questions/**${params ? **`**${!path ? **'?'** : **''**}${params}**`** : **''**}**`**)  
 .then(response => {  
 dispatch({**type**: QuestionsActionTypes.*FETCH\_QUESTIONS\_SUCCESS*, **payload**: response.**data**});  
 })  
 .catch(error => {  
 dispatch({**type**: QuestionsActionTypes.*FETCH\_QUESTIONS\_ERROR*, **payload**: **'Error while loading questions'**});  
 });  
 }  
}  
  
**export const** *fetchQuestion* = (  
 id: **number**,  
 setAuthor: React.Dispatch<**any**>,  
 setLoadingAuthor: React.Dispatch<**any**>,  
 addView: **boolean** = **true**) => {  
 **return async** (dispatch: Dispatch<QuestionsAction>) => {  
 dispatch({**type**: QuestionsActionTypes.*FETCH\_QUESTION*});  
 ***instance***.get<IQuestion>(**`forum/questions/**${id}**/`**)  
 .then(response => {  
 dispatch({**type**: QuestionsActionTypes.*FETCH\_QUESTION\_SUCCESS*, **payload**: response.**data**});  
 **if**(addView)  
 ***instance***.patch<IQuestion>(**`forum/questions/**${id}**/views/?views=**${response.**data**.**views** + 1}**`**);  
  
 ***instance***.get<IUser>(**`accounts/users/**${response.**data**.**author\_id**}**/`**)  
 .then(response => {  
 setAuthor(response.**data**);  
 setLoadingAuthor(**false**);  
 })  
 })  
 .catch(error => {  
 dispatch({**type**: QuestionsActionTypes.*FETCH\_QUESTIONS\_ERROR*, **payload**: **'Error while loading question'**})  
 });  
  
 }  
}  
  
**export const** *fetchQuestionsByTag* = (tagName: **string**) => {  
 **return async** (dispatch: Dispatch<QuestionsAction>) => {  
 dispatch({**type**: QuestionsActionTypes.*FETCH\_QUESTIONS*});  
 ***instance***.get<ITag>(**`forum/tags/**${tagName}**`**)  
 .then(response =>{  
 **if**(response.**data**.**questions**){  
 dispatch({**type**: QuestionsActionTypes.*FETCH\_QUESTIONS\_SUCCESS*, **payload**: response.**data**.**questions**});  
 }  
  
 })  
 }  
}