Прикарпатський національний університет імені Василя Стефаника
Фізико-технічний факультет
Кафедра комп'ютерної інженерії та електроніки

Стопчицький Юрій Юрійович
Yurii Stopchytskyi

УДК 004:42

Спеціальність 123 «Комп'ютерна інженерія»

Кваліфікаційна робота
на здобуття освітнього ступеня бакалавра

Розробка web-додатку для обміну мовами між носіями на основі фреймворку Django

Development of a web application for language exchange between native speakers based on the Django framework

Науковий керівник:
Кандитат т.н., доцент
комп'ютерної інженерії та
електроніки, Володимир ГРИГА
Рецензент:
Доктор ф.-м.н, професор
кафедри матеріалознавства та
новітніх технологій, Іван ЯРЕМІЙ

Івано-Франківськ
2024

Формат	Поз.	Позначення	Найменування	К-ть	Прим.
А4			Пояснювальна записка	59	
А4			Додаток А	1	
А4			Додаток Б	1	

Змн.	Арк.	№ докум.	Підпис	Дата	123.КІ-41.20		
Розробив		Стопчицький Ю.Ю.			Специфікація	Літ.	Арк. 2 / Аркушів 1
Перевірив		Грига В.М.					
Н. Контр.							
Затвердив							

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці веб-додатку для обміну мовами між носіями на основі фреймворку Django. Вона розглядає процес створення додатку для полегшення спілкування та практики іноземних мов у режимі реального часу між користувачами, які є носіями різних мов.

У роботі було проаналізовано схожі існуючі веб-додатки, проведено вибір технологій для реалізації, описано структуру та архітектуру додатку, наведено програмну реалізацію та продемонстровано функціонал розробленого додатку.

Новизна роботи полягає у створенні зручного та функціонального веб-додатку для обміну мовами між носіями з використанням актуальних технологій, таких як Django, WebSocket, Channels, PostgreSQL та інтеграцією з хмарним сховищем AWS S3.

Метою роботи є розробка веб-сайту для зручного пошуку користувачів та обміну мовами між ними з можливістю спілкування в режимі реального часу в індивідуальних та групових чатах.

Актуальність роботи зумовлена зростаючою потребою у вивченні іноземних мов та практиці спілкування з носіями в сучасному глобалізованому світі.

У результаті виконання кваліфікаційної роботи було розроблено веб-додаток для обміну мовами між носіями на основі фреймворку Django з використанням Python. Додаток забезпечує зручну платформу для пошуку партнерів, створення індивідуальних та групових чатів, спілкування в режимі реального часу, обміну файлами та мультимедіа, керування профілями, безпеку даних та захист від небажаних користувачів за допомогою функціоналу блокування.

Платформу можна використовувати для вивчення та практики іноземних мов, міжкультурної комунікації, пошуку мовних партнерів та організації мовних клубів. Його використання сприятиме покращенню мовних навичок користувачів, розширенню кругозору та налагодженню зв'язків між представниками різних національностей та культур.

ABSTRACT

The qualification work is devoted to the development of a web application for language exchange between native speakers based on the Django framework. It considers the process of creating an application to facilitate real-time communication and practice of foreign languages between users who are native speakers of different languages.

The paper analyzes similar existing web applications, selects technologies for implementation, describes the structure and architecture of the application, provides software implementation, and demonstrates the functionality of the developed application.

The novelty of the work lies in the creation of a convenient and functional web application for language exchange between native speakers using up-to-date technologies such as Django, WebSocket, Channels, PostgreSQL, and integration with the AWS S3 cloud storage.

The purpose of the work is to develop a website for convenient search of users and language exchange between them with the possibility of real-time communication in individual and group chats.

The relevance of the work is determined by the growing need to learn foreign languages and practice communication with native speakers in the modern globalized world.

As a result of the qualification work, a web application for language exchange between native speakers based on the Django framework using Python was developed. The application provides a convenient platform for finding partners, creating individual and group chats, real-time communication, file and multimedia sharing, profile management, data security, and protection from unwanted users through blocking functionality.

The platform can be used for learning and practicing foreign languages, intercultural communication, finding language partners, and organizing language clubs. Its use will contribute to improving users' language skills, broaden their horizons, and establish connections between representatives of different nationalities and cultures.

ПЕРЕЛІК ОСНОВНИХ СКОРОЧЕНЬ

ОС – операційна система

ORM (Object-relational Mapping) – Об'єктно-реляційне відображення

ASGI (Asynchronous Server Gateway Interface) – Інтерфейс підключення асинхронного сервера

DOM (Document Object Model) – Об'єктна модель документа

CSS (Cascading Style Sheets) – Каскадні таблиці стилів

HTML (HyperText Markup Language) – Мова розмітки гіпертексту

JS – JavaScript

Django – фреймворк для мови програмування Python

IDE (Integrated development environment) – Інтегроване середовище розробки

API – програмній інтерфейс додатку

Git – розподілена система керування версіями файлів та спільної роботи

Django – фреймворк мови програмування Python

React – бібліотека мови програмування Javscript

СКБД – Система керування базами даних)

Коміт – знімок локальних файлів, записаний в локальний репозиторій

Репозито́рій – сервер, на якому зберігається і з якого можна завантажити програмне забезпечення.

Скрипт – невелика програма, яка послідовно виконує список однотипних завдань.

Пояснювальна записка

до кваліфікаційної роботи

на тему:

«Розробка web-додатку для обміну мовами між носіями на основі фреймворку Django»

ЗМІСТ

ВСТУП

У сучасному глобалізованому світі володіння іноземними мовами стає все більш важливим і необхідним. Знання кількох мов відкриває нові можливості для спілкування, міжкультурного обміну та кар'єрного зростання. Однак процес вивчення нових мов може бути складним і потребувати значних зусиль та ресурсів. Одним з ефективних способів вдосконалення мовних навичок є практика з носіями мови, що дозволяє занурити в реальне мовне середовище та отримати цінний досвід спілкування.

У цьому контексті розробка веб-додатку для обміну мовами між носіями стає актуальним завданням. Такий додаток забезпечить платформу, яка об'єднає людей, які бажають вивчати нову мову, з тими, хто є її носіями, для взаємного мовного обміну. Користувачі зможуть практикувати свої мовні навички в реальному часі, обмінюватись культурним досвідом та знаннями, що сприятиме більш ефективному засвоєнню мови.

Для реалізації такого веб-додатку було обрано потужний фреймворк Django, який базується на мові програмування Python. Django відомий своєю зрозумілою структурою, високою швидкістю розробки та широким спектром можливостей для створення складних веб-додатків. Використання Django дозволить побудувати надійну, масштабовану та безпечну платформу для обміну мовами між носіями.

Далі в роботі буде детально розглянуто процес розробки веб-додатку для обміну мовами на основі фреймворку Django. Будуть висвітлені основні етапи проєктування, архітектура та структура додатку, а також ключові технології та бібліотеки, що використовувалися для реалізації необхідного функціоналу.

РОЗДІЛ 1. АНАЛІЗ АНАЛОГІЧНИХ ВЕБ-ДОДАТКІВ НА РИНКУ

1.1 Огляд існуючих додатків для обміну мовами

В сучасному глобалізованому світі вивчення іноземних мов набуває все більшої популярності. Люди прагнуть вдосконалювати свої мовні навички для кращого спілкування, відкриття нових можливостей у кар'єрі та подорожуванні. Однак, традиційні методи вивчення мови, такі як курси або приватні уроки, можуть бути дорогими та обмеженими в плані практики живого спілкування. Саме тому все більше людей звертаються до онлайн-платформ для обміну мовами, які дозволяють практикувати розмовні навички з носіями мови в зручний час і за доступною ціною.

1.1.1 Платформа для обміну мовами “Speaky”

Speaky є однією з провідних платформ для обміну мовами, яка була заснована в 2014 році. Вона пропонує користувачам можливість практикувати розмовні навички з носіями мови шляхом відеодзвінків та текстового спілкування. Платформа доступна як у вигляді веб-додатку, так і мобільного застосунку.

Speaky була розроблена з використанням популярних технологій веб-розробки, таких як React.js для клієнтської частини та Node.js для серверної частини. Для зберігання даних використовується MongoDB, а для обробки потокових даних та відеодзвінків - WebRTC.

Для забезпечення високої продуктивності та масштабованості, Speaky використовує мікросервісну архітектуру. Основні компоненти системи, такі як аутентифікація користувачів, пошук партнерів, планування занять та обробка відео, реалізовані як окремі мікросервіси, які взаємодіють між собою через REST API або повідомлення в черзі. Ця архітектура дозволяє легко масштабувати окремі компоненти залежно від навантаження та підвищує загальну стійкість системи.

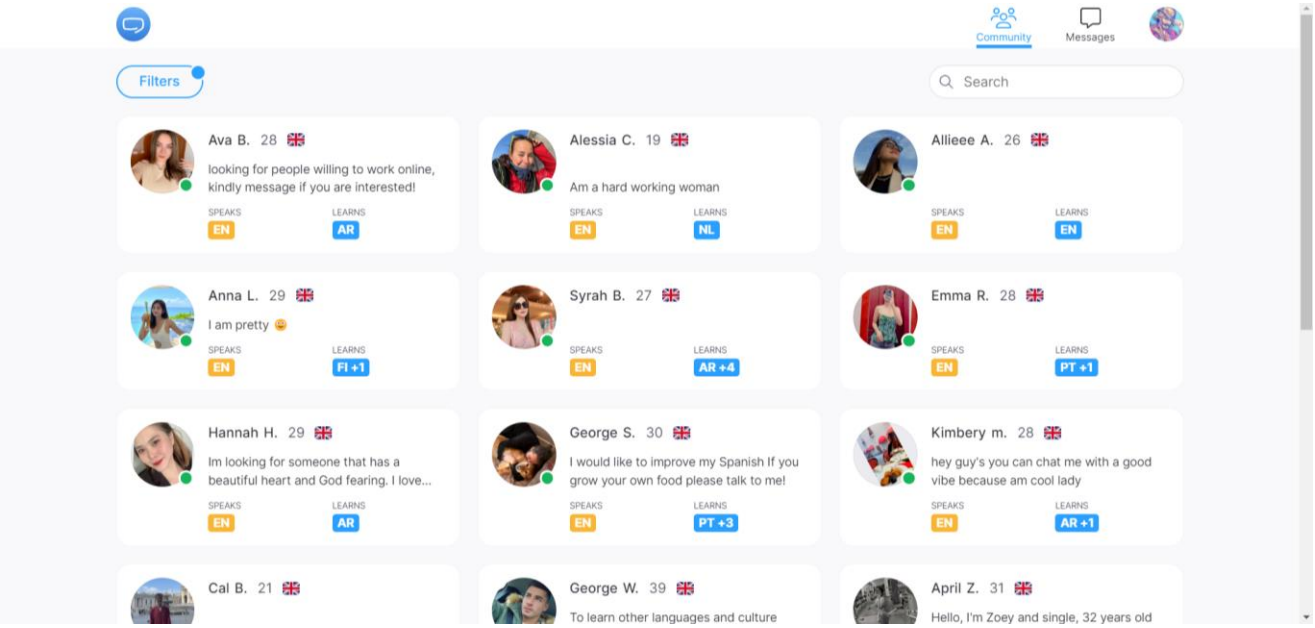

Рисунок 1.1 – Вигляд головної сторінки “Speaky”

Головною перевагою Speaky є її зручний та інтуїтивно зрозумілий інтерфейс, який дозволяє легко знаходити партнерів для спілкування та планувати заняття. Крім того, платформа пропонує різноманітні інструменти для вивчення мови, такі як словники, граматичні вправи та матеріали для читання.

Однак, деякі користувачі відзначають, що процес пошуку партнерів може бути дещо складним, особливо для менш поширених мов. Також існують певні побоювання щодо безпеки та конфіденційності під час спілкування з незнайомими людьми.

1.1.2 Платформа для обміну мовами “Tandem”

Tandem є іншою популярною платформою для обміну мовами, яка була заснована в 2015 році. Вона пропонує користувачам можливість практикувати розмовні навички шляхом текстового та голосового спілкування, а також обміну повідомленнями та файлами.

Tandem був розроблений з використанням фреймворку Flutter для крос-платформеної розробки мобільних додатків, а для веб-версії використовувався React.js. На стороні сервера використовується Node.js, а для зберігання даних - MongoDB.

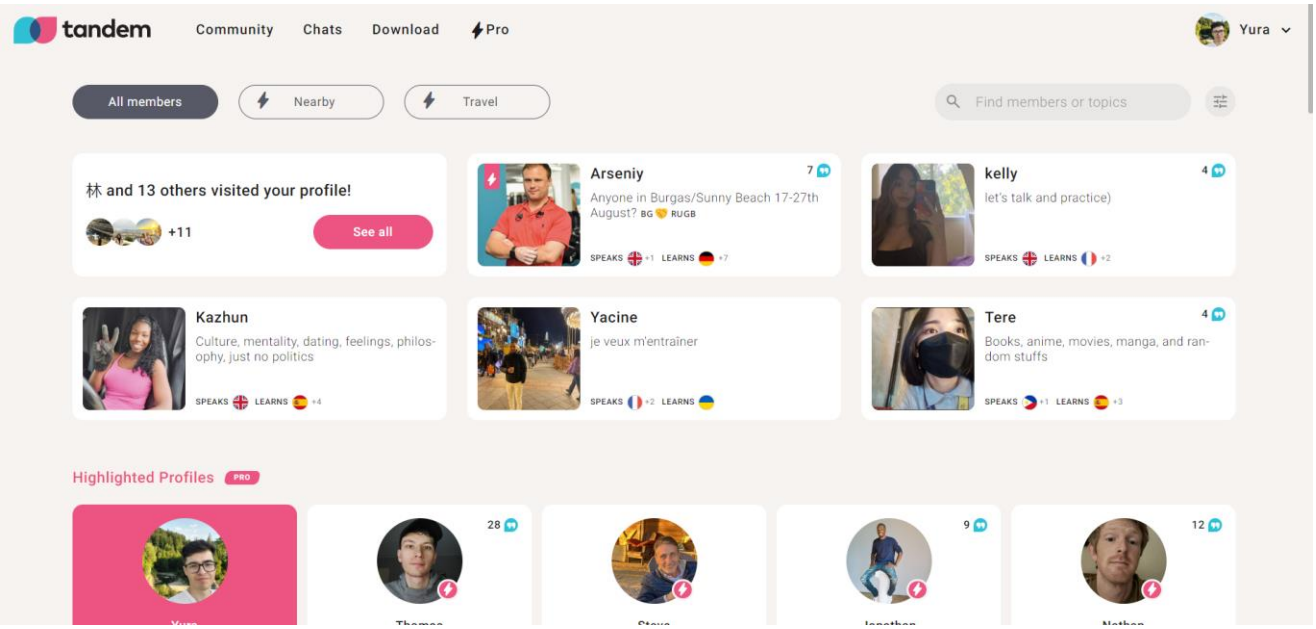

Рисунок 1.2 – Вигляд головної сторінки “Tandem”

Головною перевагою Tandem є її потужний алгоритм пошуку партнерів, який враховує не лише мовні вподобання користувачів, але й їхні інтереси, вік та місцезнаходження. Це дозволяє знаходити більш сумісних партнерів для спілкування та покращує якість вивчення мови.

Однак, деякі користувачі відзначають, що інтерфейс Tandem може бути дещо перевантаженим та складним для новачків. Крім того, існують певні проблеми з якістю звуку під час голосового спілкування.

1.1.3 Платформа для обміну мовами “HelloTalk”

HelloTalk є однією з перших та найпопулярніших платформ для обміну мовами, яка була заснована у 2012 році. Вона пропонує користувачам можливість

практикувати розмовні навички шляхом текстового та голосового спілкування, а також обміну повідомленнями, зображеннями та аудіо-файлами.

HelloTalk був розроблений з використанням React Native для крос-платформеної розробки мобільних додатків, а для веб-версії використовувався React.js. На стороні сервера використовується Node.js, а для зберігання даних - MongoDB.

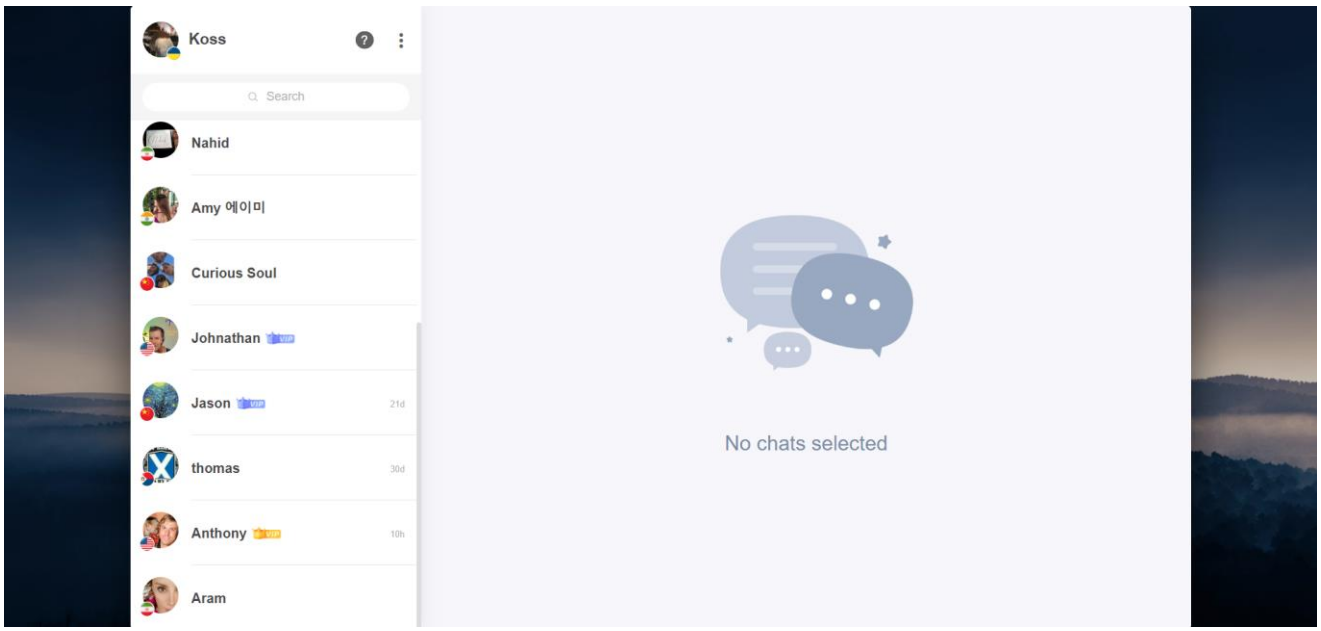

Рисунок 1.3 – Вигляд головної сторінки “HelloTalk”

Головною перевагою HelloTalk є її велика база користувачів з усього світу, що значно полегшує пошук партнерів для спілкування. Крім того, платформа пропонує зручну систему корекції помилок, де носії мови можуть виправляти помилки учнів та надавати корисні поради.

Однак, деякі користувачі відзначають, що на платформі трапляються випадки недоброчесної поведінки, такі як спам або неналежне використання сервісу. Також існують певні проблеми з якістю звуку та затримками під час голосового спілкування.

1.1.4 Платформа для обміну мовами “Preply”

Preply є відносно новою платформою для обміну мовами, яка була заснована у 2017 році. Вона пропонує користувачам можливість практикувати розмовні навички шляхом відеодзвінків та текстового спілкування з професійними викладачами та носіями мови.

Preply був розроблений з використанням Angular для клієнтської частини та Node.js для серверної частини. Для зберігання даних використовується PostgreSQL, а для обробки відеодзвінків - WebRTC.

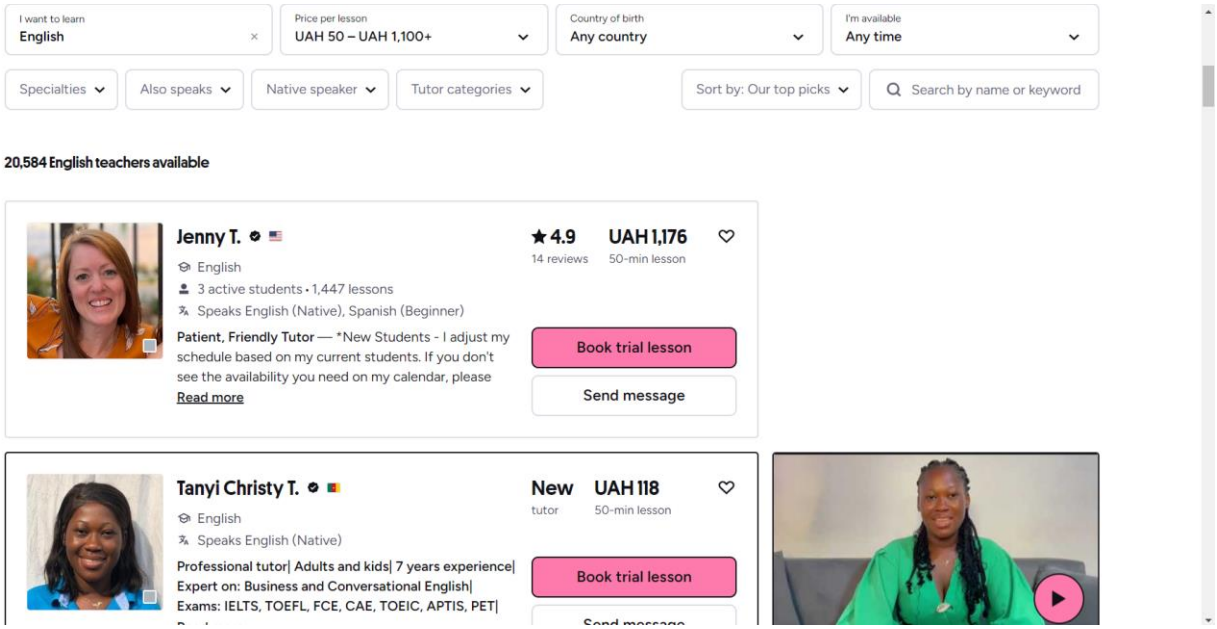

Рисунок 1.3 – Вигляд головної сторінки “Preply”

Головною перевагою Preply є її фокус на якості викладання та професійності викладачів. Платформа ретельно відбирає викладачів та перевіряє їхню кваліфікацію, щоб забезпечити високу якість навчання.

Однак, деякі користувачі відзначають, що ціни на уроки на Preply можуть бути дещо вищими, ніж на інших платформах. Крім того, існують певні проблеми з доступністю викладачів для менш поширених мов.

1.2 Аналіз фреймворків для розробки додатку для обміну мовами

При розробці додатку для обміну мовами між носіями, вибір правильного фреймворку є ключовим етапом, який визначає не лише швидкість та ефективність розробки, але й успіх проєкту в цілому. Розглянемо деякі найпопулярніші фреймворки та їх особливості з точки зору реалізації даного проєкту: Django, Flask, React та Ruby on Rails.

Django:

Django — це повнофункціональний фреймворк для веб-розробки, який базується на мові програмування Python. Його особливість полягає в тому, що він надає високорівневі засоби для швидкої та ефективної розробки веб-додатків. Основними перевагами Django є:

1. **Зручна структура проєкту:** Django надає готову структуру проєкту, що дозволяє розробникам швидко орієнтуватися та почати роботу над продуктом.
2. **Багатий набір готових рішень:** Фреймворк містить в собі багато готових модулів та бібліотек для аутентифікації, авторизації, управління користувачами та багато іншого, що значно спрощує розробку.
3. **Активна спільнота розробників:** Django має велику та активну спільноту розробників, що забезпечує постійну підтримку, оновлення та розвиток фреймворку.
4. **Безпека:** Фреймворк дотримується принципу "безпека з коробки", що означає, що він захищений від багатьох типових вразливостей веб-додатків, таких як міжсайтовий скриптинг (XSS), міжсайтове підроблення запитів (CSRF) та SQL-ін'єкції. Це забезпечується такими інструментами, як, наприклад, {% csrf_token %}.

5. **Розширений функціонал шаблоні:** Також він дає можливість використовувати django template language (шаблонні теги Django). Вони дозволяють передавати Python-подібні об’єкти в HTML. Таким чином розробка динамічних сторінок стає зручнішою та простішою.
6. **Масштабованість:** Django добре підходить для створення масштабованих веб-додатків, оскільки він підтримує кешування, балансування навантаження та шардинг бази даних.
7. **Адміністративна панель:** Django має вбудовану адміністративну панель, яка дозволяє легко керувати даними та користувачами веб-додатку.

Додатково, факт того, що одна з найпопулярніших соціальних мереж — Instagram, була розроблена на базі Django, свідчить про його потужність та надійність. Django ідеально підходить для проєктів, які потребують великої кількості користувачів, складної бізнес-логіки та високої надійності.

Flask:

Flask - це мінімалістичний мікрофреймворк для веб-розробки, також написаний на Python. Він пропонує більш гнучкий підхід до розробки, проте зазвичай вимагає більше ручних лалаштувань та розробки певних функцій “з нуля”. Flask частіше використовується для невеликих проєктів або додатків зі специфічними вимогами, які не вписуються в стандартну структуру Django. Основними перевагами Flask є:

1. **Легкість:** Flask є легким та компактним фреймворком, що робить його простим у вивченні та налаштуванні.
2. **Гнучкість:** На відміну від Django, Flask не нав'язує певну структуру чи архітектуру проєкту, дозволяючи розробникам будувати додатки так, як їм зручніше.

3. **Розширюваність:** Flask підтримує велику кількість розширень, які дозволяють додавати функціональність за потреби, такі як аутентифікація користувачів, кешування, завантаження файлів тощо.

4. **Швидкість розробки:** Завдяки своїй простоті та гнучкості, Flask часто забезпечує більш швидку розробку для невеликих проєктів.

Проте, для великих проєктів, таких як соціальна мережа обміну мовами, Flask може виявитися менш зручним, ніж Django, через відсутність вбудованих функцій та необхідність розробляти більше функціональності вручну.

React:

React - це бібліотека JavaScript для розробки інтерфейсів користувача. Вона використовується для створення динамічних та інтерактивних веб-інтерфейсів, які реагують на зміни даних в реальному часі. Основними перевагами React є:

1. **Компонентний підхід:** React дозволяє розбивати інтерфейс на окремі компоненти, які можна багаторазово використовувати та легко управляти.
2. **Віртуальний DOM:** React використовує віртуальний DOM для оптимізації продуктивності, оновлюючи лише ті частини інтерфейсу, які змінились.
3. **Великі можливості розширення:** React має велику екосистему бібліотек та інструментів, які дозволяють легко додавати нову функціональність до додатку.
4. **Потужна спільнота:** React має велику та активну спільноту розробників, що забезпечує постійну підтримку та розвиток бібліотеки.

Хоча React відмінно підходить для створення динамічних та інтерактивних інтерфейсів, для розробки повноцінного веб-додатку для обміну мовами потрібно додатково використовувати серверну частину, наприклад, Django або Flask.

Ruby on Rails:

Ruby on Rails - це фреймворк для веб-розробки, який базується на мові програмування Ruby. Його основною перевагою є швидкість розробки та конвенція над конфігурацією. Деякі ключові особливості Ruby on Rails включають:

1. **Конвенція над конфігурацією:** Ruby on Rails дотримується філософії "конвенція над конфігурацією", що дозволяє розробникам швидко розгортати веб-додатки, без необхідності налаштовувати багато конфігурацій вручну.
2. **Підтримка MVC:** Ruby on Rails підтримує архітектурний шаблон Model-View-Controller (MVC), що забезпечує чітке розділення логіки додатку.
3. **Швидкість розробки:** Завдяки своїм конвенціям та вбудованим інструментам, Ruby on Rails дозволяє швидко створювати прототипи та розробляти веб-додатки.
4. **Активна спільнота:** Ruby on Rails має велику та активну спільноту розробників, що забезпечує підтримку, інструменти та бібліотеки для розширення функціональності.

Проте, у порівнянні з Django, Ruby on Rails може бути менш гнучким та вимагати більше часу для оволодіння його конвенціями

У таблиці 1.1 продемонстровано рейтинг кожного з фреймворків відповідно до їх технічних характеристик.

Таблиця 1.1 – Порівняння відібраних фреймворків

Характеристика	**Фреймворк**			
	Django	Flask	React	Ruby on Rails
Функціональність	9	6	7	8
Розширюваність	9	9	9	7
Швидкодія	8	9	9	8
Компонентний підхід	10	6	10	8
Стабільність та підтримка	9	7	9	8
Безпека та надійність	9	7	7	8
Спільнота	9	7	10	8
Всього	63	51	61	55

Загалом, React має великий сумарний бал завдяки своїй популярності та зосередженості на розробці інтерфейсів користувача, а Django є найкращим вибором для створення повнофункціональних веб-додатків із серверною частиною.

У порівнянні з іншими фреймворками, Django виділяється своєю багатофункціональністю, зручністю у використанні та великою кількістю готових рішень для розробки веб-додатків. Django має потужну документацію та активну спільноту розробників, що забезпечує доступ до багатьох розширень, сторонніх бібліотек та рішень для спрощення процесу розробки веб-додатків, дозволяючи використовувати перевірені практики та підходи для вирішення специфічних задач. Для проєкту обміну мовами на основі соціальної мережі, Django підходить завдяки своїй потужності, швидкості розробки, безпеки та можливостям для розширення функціональності. Саме тому для роботи було використано даний фреймворк.

1.3 Постановка завдання

Метою проєкту є розробка веб-додатку для обміну мовами між носіями на основі фреймворка Django з використанням Python.

Для досягнення мети, потрібно виконати наступні завдання:

1. Встановити та налаштувати необхідні технології для подальшої розробки, а саме Python, Django, PostgreSQL, Poetry; Ruff, git.
2. Створити новий проєкт за допомогою команди django-admin startproject language_exchange_app;
3. Створити 3 окремі застосунки під різні цілі проєкту (авторизація, функціонал чату та головний застосунок);
4. Визначитися з архітектурою веб-додатку;
5. Розробити загальну схему бази даних та реалізувати її за допомогою Django ORM;
6. Створити сторінку профілю та редагування профілю користувача з відповідними полями для заповнення даних:
    - фото профілю,
    - ім’я,
    - вік,
    - стать,
    - національність,
    - список мов,
    - рівень володіння мовами,
    - опис профілю користувача;
7. Реалізувати функцію авторизації за допомогою Google account та email;
8. Розробити головну сторінку з відображенням інших користувачів;
9. Реалізувати функцію пошуку користувачів;
10. Розробити алгоритм підбору користувачів у стрічці на головній сторінці;
11. Додати систему блокування небажаних користувачів;

12.Імплементувати функціонал чату:

- 1-на-1 та групові чати,
- відправлення повідомлень за допомогою WebSocket,
- пересилання фото та файлів та збереження їх до AWS S3,
- система запрошень, права учасників групи та їх передачі власником,
- видалення чату;

13.Розробити дизайн веб-сайту;

14.Зробити адаптацію для різних пристроїв та діагоналей дисплею;

15.Провести тестування адаптивності під різні види екранів.

Для реалізації веб-додатку потрібно створити окремі сторінки, які будуть взаємодіяти між собою, а саме:

1. Black_List;
2. Contact;
3. Filter;
4. Index;
5. Message;
6. Profile;
7. Register;
8. Access_Denied;
9. Base.

РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБ-ДОДАТКУ

2.1 Розробка структури веб-сайту

Розробка структури веб-додатку для обміну мовами між носіями відіграє ключову роль у забезпеченні зручного та ефективного досвіду для користувачів. Правильно спроектована структура допомагає користувачам легко орієнтуватися в додатку, знаходити потрібний функціонал та взаємодіяти з іншими.

Насамперед, важливо визначити основні компоненти додатку та їх взаємозв'язки. У контексті веб-додатку для обміну мовами, ключовими компонентами є:

1. **Профілі користувачів:** Кожен користувач повинен мати власний профіль, де буде зберігатися інформація про нього, такі як ім'я, фото, опис до профілю, національність, а також перелік мов, якими він володіє, та рівень їх знання.

2. **Чати:** Ядром веб-додатку буде система чатів, яка дозволить користувачам взаємодіяти один з одним у режимі реального часу. Ця система повинна підтримувати як індивідуальні чати (1-на-1), так і групові чати для спілкування з кількома учасниками.

3. **Система управління чатами та запрошеннями:** Для забезпечення зручного керування чатами, необхідно розробити систему, що дозволить користувачам створювати нові чати, запрошувати та видаляти інших учасників, приєднуватися до існуючих чатів та залишати уже створені. Також потрібно реалізувати роль та права адміністратора (власника) чату.

4. **Пошук користувачів:** Для полегшення пошуку потенційних партнерів для спілкування та мовного обміну, додаток повинен мати функцію пошуку

користувачів за різними критеріями, такими як мова, її рівень володіння, країна, вік, стать.

5. **Зберігання файлів:** Оскільки користувачі можуть обмінюватися файлами під час спілкування, необхідно інтегрувати хмарне сховище даних, таке як AWS S3, для безпечного та ефективного зберігання цих файлів.

6. **Аутентифікація користувачів:** Для забезпечення безпеки та контролю доступу, необхідно реалізувати систему аутентифікації користувачів, яка дозволить їм реєструватися та входити в систему за допомогою облікових записів Google або електронної пошти.

Враховуючи ці основні компоненти, структура веб-додатку на Django може бути побудована з використанням патерну Model-View-Template (MVT).

Models (моделі Django) будуть представляти дані, пов'язані з профілями користувачів, чатами, повідомленнями та іншими сутностями додатку. Вони також забезпечать взаємодію з базою даних для зберігання та отримання цих даних.

Views (представлення Django) відповідають за обробку HTTP-запитів від клієнтів, виконують необхідну логіку та передають дані шаблонам для відображення. Наприклад, можуть бути створені окремі представлення для обробки запитів, пов'язаних з профілями користувачів, чатами, пошуком користувачів тощо.

Templates (шаблони Django) відповідають за відображення даних у HTML-форматі для користувачів. Вони отримують дані з Views та рендерять їх у зручному для користувача вигляді. Наприклад, можуть бути створені окремі шаблони для відображення профілю користувача, списку чатів, інтерфейсу чату, головної сторінки тощо.

Для інтеграції з хмарним сховищем AWS S3 можна використати сторонню Python-бібліотеку, наприклад, boto3. Ця бібліотека дозволить взаємодіяти з API AWS S3 для завантаження, збереження та отримання файлів з хмарного сховища.

Для реалізації чату в режимі реального часу можна використати технологію WebSocket. Django має вбудовану підтримку WebSocket через пакет Channels. Цей пакет дозволяє створювати асинхронні веб-сокети та обробляти повідомлення, що надходять від клієнтів. Він також підтримує інтеграцію з різними серверами веб-сокетів, такими як Daphne або Uvicorn.

Для аутентифікації користувачів можна використати вбудовану систему аутентифікації Django. Проте для безпеки та зручності користування краще інтегрувати сторонню бібліотеку, таку як django-allauth, яка підтримує аутентифікацію через облікові записи Google, електронну пошту та інші провайдери.

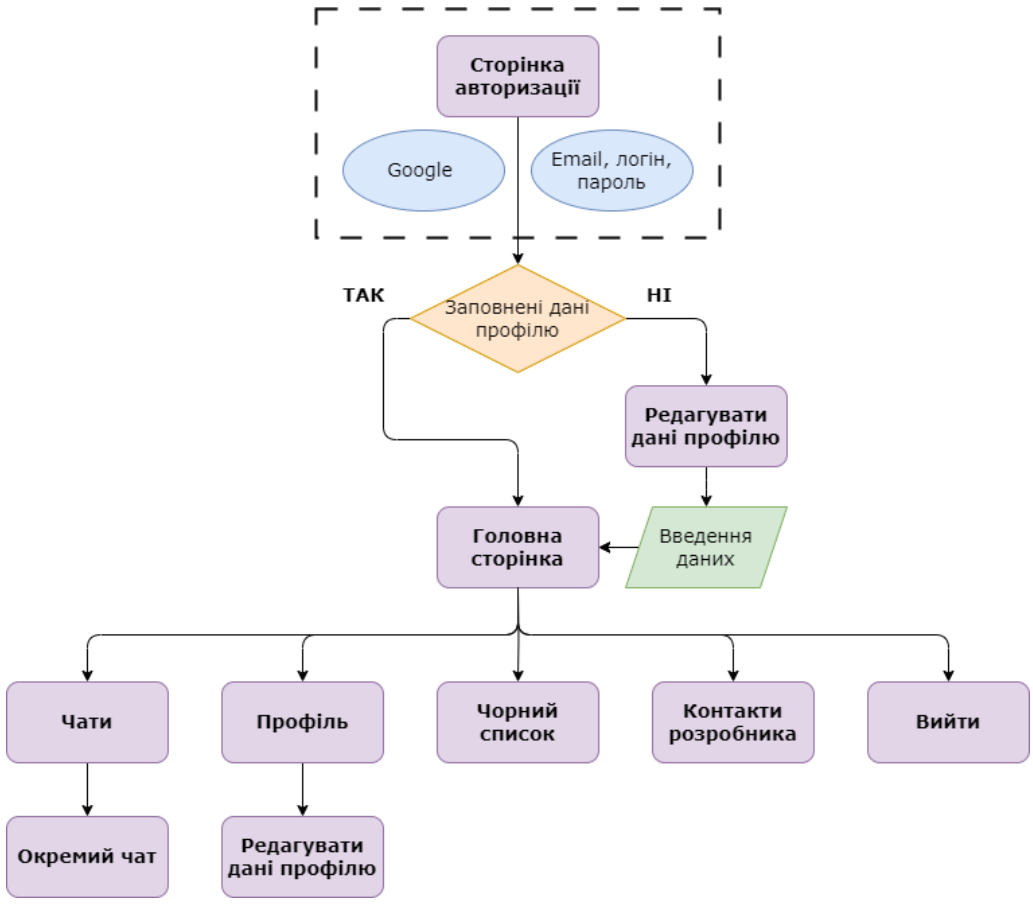

Рисунок 2.1 – Структура веб-додатку обміну мовами між носіями

Прийнявши всі вищезгадані вимоги до уваги, було розроблено схему веб-додатку, де зазначено, з яких елементів буде складатися проєкт (Рис. 2.1).

Загалом, розробка структури веб-додатку для обміну мовами на основі Django вимагає ретельного планування та врахування різних компонентів і функціональних вимог. Використання патерну MVT, інтеграція з хмарним сховищем даних, реалізація чату в режимі реального часу та надійна система аутентифікації користувачів - все це є ключовими аспектами, які слід враховувати під час проєктування структури.

2.2 Налаштування середовища розробки веб-додатку

Основним інструментом для написання коду та редагування файлів проєкту став Visual Studio Code (VSCode) – потужний та гнучкий редактор коду від компанії Microsoft. Завдяки інтуїтивному інтерфейсу, багатій екосистемі розширень та широкій підтримці різних мов програмування, включаючи Python, VSCode забезпечив зручне середовище для розробки веб-додатку на Django. Розширення, такі як Python, Pylance та Pylint, надали додаткові можливості для покращення продуктивності роботи з Python, включаючи автодоповнення коду, перевірку синтаксису та статичний аналіз.

Для управління залежностями проєкту та забезпечення ізоляції середовища був використаний менеджер пакетів Poetry. На відміну від традиційного підходу з pip та venv, Poetry пропонує більш структурований та зручний спосіб керування залежностями, автоматично вирішуючи конфлікти версій бібліотек. Завдяки Poetry було створено віртуальне середовище Python всередині папки проєкту, що дозволило уникнути конфліктів з глобальними залежностями та забезпечило портативність розгортання додатку.

Для забезпечення високої якості коду та дотримання кращих практик його написання були впроваджені інструменти статичного аналізу та перевірки стилю

коду. Зокрема, Ruff – потужний інструмент, який дозволяв виявляти потенційні помилки, неефективний код та порушення стандартів написання коду відповідно до різних конфігурацій. Крім того, були налаштовані pre-commit hooks – спеціальні скрипти, які автоматично запускалися перед кожним комітом коду у систему контролю версій Git. Ці скрипти забезпечували автоматизовану перевірку коду, форматування відповідно до встановлених правил та виявлення можливих проблем ще до внесення змін у репозиторій.

Система контролю версій Git відігравала ключову роль у збереженні файлів, відокремлення головних блоків та додаванні нового функціоналу. Git дозволяв відстежувати зміни в коді, працювати над різними гілками розробки та ефективно керувати версіями веб-додатку.

Під час локального розгортання та тестування додатку був використаний ASGI-сервер (Asynchronous Server Gateway Interface) – Daphne. На відміну від традиційних WSGI-серверів, ASGI забезпечує підтримку асинхронних веб-сокетів, що є критично важливим для реалізації функціоналу чату в режимі реального часу. Використання ASGI-сервера дозволило налагодити функціонал чату безпосередньо в локальному середовищі, забезпечуючи відповідність вимогам проєкту.

Загалом, ретельно підібране та налаштоване середовище розробки відіграло ключову роль у забезпеченні ефективності, якості та успішності реалізації веб-додатку для мовного обміну. Використання потужних інструментів, таких як VSCode, Poetry, Ruff, pre-commit hooks, Git та ASGI-сервер, дозволило зосередитися на функціональності додатку та дотримуватися кращих практик розробки.

На рисунку 2.2 наведений зразок вигляду інтерфейсу IDE VSCode та наведено структуру головного проєкту та його застосунків.

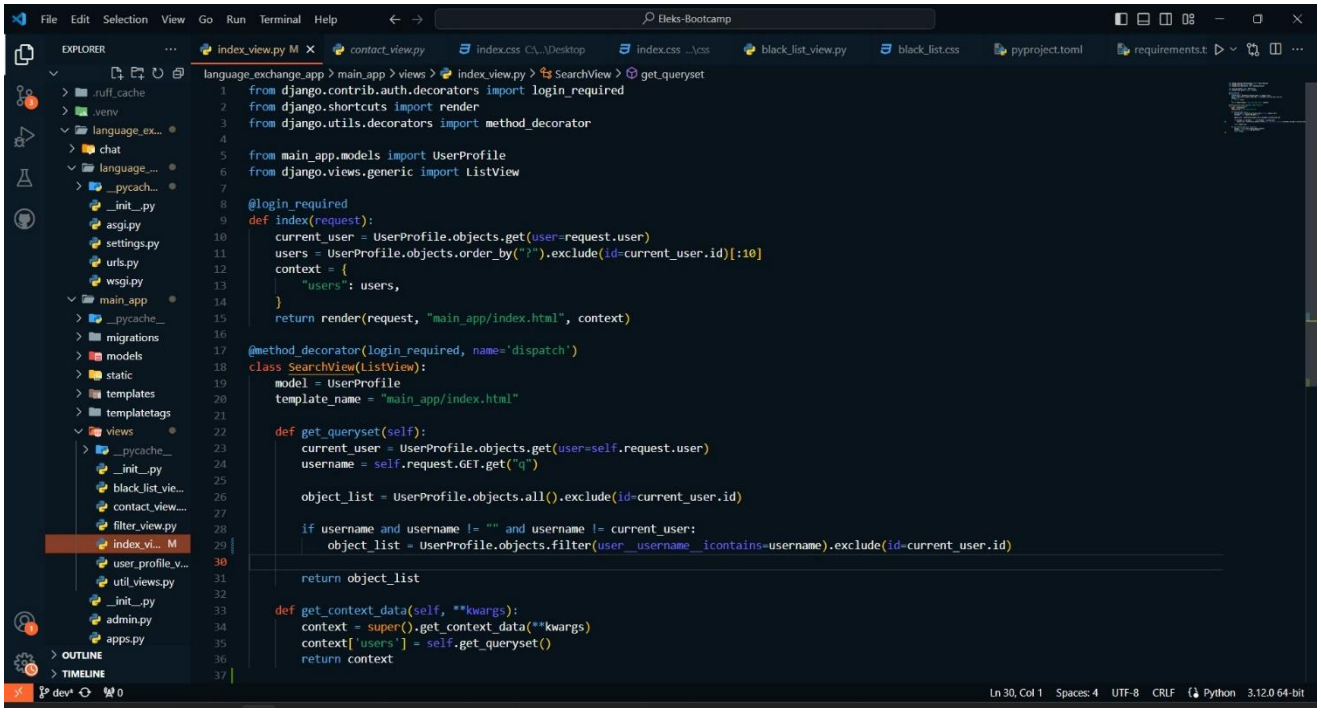

Рисунок 2.2 – Приклад розробки в середовищі VSCode

2.3 Мова програмування Python

Python - це інтерпретована, об'єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Її зрозумілий синтаксис разом із інтерпретованою природою роблять Python ідеальною для швидкої розробки додатків (Rapid Application Development). Python широко використовується у багатьох галузях, включаючи веб-розробку, аналіз даних, машинне навчання, автоматизацію скриптів тощо.

З точки зору веб-розробки, Python є надзвичайно потужним інструментом. Багато веб-фреймворків та бібліотек, таких як Django, Flask, Pyramid, дозволяють створювати як прості, так і комплексні веб-додатки. Обраний фреймворк Django надає структуровану основу з безліччю вбудованих можливостей для розробки веб-додатків на Python.

Ключові переваги Python для веб-розробки:

1. **Простий та легкий для читання синтаксис:** Python відомий своїм чистим, лаконічним кодом, який легко читати та підтримувати, що скорочує час розробки.
2. **Велика кількість бібліотек та розширень:** Величезна екосистема бібліотек та розширень Python робить цю мову надзвичайно гнучкою. Від обробки зображень до роботи з базами даних, машинного навчання та реалізації веб-скрепінгу.
3. **Швидкість розробки:** Завдяки простому синтаксису та великому набору внутрішніх методів та функцій, можна збільшити швидкість розробки веб-додатків на Python.
4. **Обробка помилок та легкість налагодження:** Зрозумілі повідомлення про помилки та відмінні інструменти налагодження роблять розробку на Python більш ефективною.
5. **Безпека та надійність:** Python відомий своєю безпекою та надійністю на рівні коду з чіткими інструкціями з написання безпечного коду.

Для створення веб-додатку для обміну мовами, ці особливості Python є визначальними. Мінімалістична, але функціональна архітектура Django дозволяє швидко будувати складні веб-додатки з безліччю вбудованих можливостей для аутентифікації, URL-маршрутизації, адміністративної панелі тощо.

Бібліотеки Python для веб-розробки:

1. Django REST framework - для створення RESTful API;
2. Channels та Daphne - для інтеграції WebSocket та функціоналу чату;
3. Django-allauth - для розширеної аутентифікації;
4. Pillow - для обробки зображень;
5. Psycopg2 - для взаємодії з PostgreSQL.

Додаткові можливості Python:

1. Потужні інструменти для роботи з файлами, рядками, JSON, дата, час тощо;
2. Великий вибір бібліотек для зв'язку з різними інтернет-протоколами;
3. Велика кількість фреймворків для машинного навчання, великих обчислень;
4. Автоматизація завдань та скриптів завдяки універсальності Python.

Ці можливості роблять Python ідеальним не лише для веб-розробки, а й для майбутніх розширень та інтеграцій веб-додатку обміну мовами.

2.4 Бібліотека django-allauth

У процесі створення веб-додатку для мовного обміну важливо забезпечити зручний та безпечний процес реєстрації та входу для користувачів. Саме тому в проєкті була інтегрована бібліотека django-allauth – набір іструментів для розширеної авторизації в Django.

Django-allauth - це пакет Python, який містить набір функцій аутентифікації та реєстрації, готових до інтеграції у будь-який Django проєкт. Ця бібліотека спрощує впровадження різних способів аутентифікації, включаючи соціальні мережі, електронну пошту тощо.

Ключові переваги використання django-allauth:

1. **Різноманітні способи аутентифікації:** django-allauth забезпечує можливість аутентифікації за допомогою електронної пошти, а також через популярні соціальні платформи, такі як Google, Facebook, Twitter(X) та інші;
2. **Просте налаштування та інтеграція:** Бібліотека легко інтегрується у будь-який Django проєкт та надає зрозумілу документацію для швидкого налаштування;

3. **Гнучкість та розширюваність:** django-allauth є високо налаштовуваним та розширюваним рішенням, що дозволяє додавати нові провайдери аутентифікації або модифікувати існуючі відповідно до потреб проєкту;
4. **Безпека:** Бібліотека слідує найкращим практикам безпеки та забезпечує захист від атак, наприклад CSRF.

Для розробки веб-додатку, django-allauth було інтегровано для забезпечення зручної реєстрації та входу користувачів через Google та електронну пошту. При авторизації через Google, бібліотека використовує лише необхідні публічні дані про користувача, зберігаючи таким чином приватність.

Django-allauth дає користувачам можливість швидко зареєструватися або увійти в систему за допомогою вже існуючого облікового запису Google, не потребуючи створення нового пароля. Альтернативно, вони можуть зареєструватися за електронною поштою та паролем.

2.5 PostgreSQL та бібліотека psycopg2-binary

PostgreSQL - це потужна об'єктно-реляційна система керування базами даних (СКБД) з відкритим вихідним кодом. Вона базується на клієнт-серверній архітектурі, де сервер PostgreSQL зберігає та обробляє дані, а клієнти (наприклад, веб-додатки або інструменти адміністрування) підключаються до нього та взаємодіють з базою даних.

Робота з даними в PostgreSQL відбувається за допомогою мови структурованих запитів (SQL). SQL - це стандартизована мова програмування, яка використовується для створення, маніпулювання та запитів до реляційних баз даних. PostgreSQL повністю підтримує стандарт SQL і додатково включає багато розширень.

Psycopg2-binary - це бібліотека Python, яка є адаптером для підключення та взаємодії з базами даних PostgreSQL. Вона забезпечує зручний інтерфейс для виконання SQL-запитів, отримання та обробки результатів з PostgreSQL з Python-коду.

Основні функції та можливості psycopg2-binary:

1. **Підключення до бази даних:** Бібліотека дозволяє встановлювати з'єднання з базою даних PostgreSQL, використовуючи параметри підключення, такі як ім'я хоста, порт, ім'я користувача та пароль;
2. **Виконання SQL-запитів:** psycopg2-binary надає методи для виконання різних видів SQL-запитів: SELECT (вибірка даних), INSERT (вставка даних), UPDATE (оновлення даних), DELETE (видалення даних) та інших;
3. **Обробка результатів:** Після виконання запиту psycopg2-binary повертає набір результатів, який можна обробляти та перетворювати на об'єкти Python для подальшої роботи;
4. **Підтримка транзакцій:** Бібліотека дозволяє працювати з транзакціями в базі даних, забезпечуючи атомарність, узгодженість, ізоляцію та стійкість операцій над даними;
5. **Безпека та обробка помилок:** psycopg2-binary забезпечує безпечну взаємодію з базою даних, автоматично виділяючи спеціальні символи та обробляючи виняткові ситуації.

У веб-додатку для обміну мовами між носіями, PostgreSQL було обрано як СКБД через її потужність, надійність, просту інтеграцію та можливість легкого розгортання. Використання psycopg2-binary як драйвера Python для PostgreSQL забезпечило надійну взаємодію між Django та базою даних, дозволяючи зберігати та обробляти дані користувачів, чатів, повідомлень та іншої важливої інформації додатку.

2.6 Бібліотеки channels, daphne та технологія WebSocket

WebSocket - це протокол зв'язку, який забезпечує двосторонній канал зв'язку поверх одного TCP-з'єднання. На відміну від традиційної моделі HTTP, де клієнт ініціює запит, а сервер дає на нього відповідь, WebSocket дозволяє серверу надсилати повідомлення клієнту в будь-який час, не очікуючи запиту. Це робить WebSocket хорошим рішенням для реалізації функціональності чату в режимі реального часу та інших сценаріїв, де потрібна двостороння комунікація.

Переваги WebSocket для веб-додатків, що потребують взаємодії в реальному часі:

1. **Двонаправлений зв'язок:** Сервер може надсилати повідомлення клієнту самостійно, без необхідності постійних запитів з боку клієнта;
2. **Ефективність:** WebSocket використовує лише одне TCP-з'єднання, що значно ефективніше, ніж встановлення нового з'єднання для кожного HTTP-запиту;
3. **Низька затримка:** Завдяки постійному зв'язку, повідомлення передаються з мінімальною затримкою;
4. **Масштабованість:** WebSocket дозволяє обробляти велику кількість одночасних з'єднань на сервері.

Channels - це бібліотека для Django, яка забезпечує підтримку асинхронних можливостей, таких як WebSocket, для створення веб-додатків в режимі реального часу. Channels надає інструменти для керування WebSocket-з'єднаннями, розподілу повідомлень між підключеними клієнтами та інтеграції з традиційною моделлю запитів/відповідей Django.

Daphne - це термінальний сервер, який входить до комплекту з Channels і є ASGI-сервером (Asynchronous Server Gateway Interface). Daphne відповідає за прослуховування вхідних з'єднань, маршрутизацію HTTP-запитів до Django та керування WebSocket-з'єднаннями через Channels.

Ключові переваги використання Channels та Daphne для роботи з WebSocket у Django:

1. **Інтеграція з Django:** Channels тісно інтегрується з Django, дозволяючи використовувати існуючі моделі, представлення, шаблони та інші компоненти Django для побудови веб-додатків в режимі реального часу;
2. **Обробка подій WebSocket:** Channels надає зручний спосіб визначення обробок подій WebSocket, таких як встановлення з'єднання, отримання повідомлень та закриття з'єднання.
3. **Групи та канали:** Channels дозволяє групувати підключених клієнтів та надсилати повідомлення на групи або окремі канали, що дозволяє реалізувати функціональності чатів та трансляцій.
4. **Масштабованість:** Channels та Daphne розроблені з урахуванням масштабованості, дозволяючи обробляти велику кількість одночасних з'єднань на одному сервері або розподіляти навантаження між кількома серверами.
5. **Моніторинг та налагодження:** Channels забезпечує інструменти для моніторингу та налагодження WebSocket-з'єднань та обробки повідомлень.

У даному проєкті бібліотеки Channels та Daphne були використані для реалізації функціональності 1-на-1 та групових чат-кімнат, а також для надсилання повідомлень у режимі реального часу за допомогою технології WebSocket. Це дозволило створити динамічний та інтерактивний інтерфейс чату, де користувачі можуть миттєво обмінюватися текстовими повідомленнями, не очікуючи оновлення сторінки.

2.7 Бібліотеки django-storages, boto3 та хмарне сховище AWS S3

Django-storages - це колекція інструментів зберігання даних для Django, що надає можливість роботи з різними типами сховищ, включаючи хмарні сервіси,

такі як Amazon S3. Boto3 є офіційною бібліотекою AWS SDK для мови програмування Python, яка забезпечує зручний інтерфейс для взаємодії з численними сервісами AWS, зокрема з S3.

У розробленому веб-додатку було використовувалось AWS S3 як сховище для файлів та зображень, завантажених користувачами. Це було обґрунтовано такими **ключовими перевагами хмарних сховищ даних AWS:**

1. **Безкоштовний пробний період:** AWS надає новим клієнтам безкоштовний пробний період, протягом якого можна безплатно користуватися деякими сервісами в межах певних обмежень;
2. **Масштабованість:** Хмарні сховища AWS є високо масштабованими, що забезпечує стабільну роботу додатку навіть при зростанні обсягів даних та кількості користувачів;
3. **Надійність:** AWS S3 гарантує високу доступність та надійність зберігання даних завдяки розподіленій архітектурі та резервному копіюванню.

Переваги використання django-storages та boto3 для інтеграції з AWS S3:

1. **Спрощена інтеграція:** Django-storages надає зручні абстракції для налаштування з'єднання з AWS S3, визначення правил доступу та конфігурації шляхів зберігання файлів;
2. **Безпечне завантаження та доступ до файлів:** Ці бібліотеки забезпечують безпечне завантаження, зберігання та доступ до файлів та зображень, що надсилалися користувачами додатку під час спілкування в чатах;
3. **Взаємодія з об'єктами AWS S3:** Boto3 дозволяє безпосередньо взаємодіяти з об'єктами Amazon S3 через Python, завантажувати, видаляти та керувати даними, збереженими в сховищі.

У даному проєкті бібліотеки django-storages та boto3 були використані для забезпечення функціональності надсилання файлів та зображень через Django форми в процесі спілкування в чатах. Завантажені користувачами файли

зберігалися на AWS S3, а доступ до них надавався з належним рівнем безпеки та конфіденційності. Це дозволило уникнути збереження файлів локально на сервері веб-додатку, що покращило масштабованість та надійність системи.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ

3.1 Структура проєкту

Проєкт "language_exchange_app" був розроблений з використанням фреймворку Django та дотримується рекомендацій офіційної документації Django та загальноприйнятих практик організації коду. Структура проєкту забезпечує модульність, масштабованість та простоту підтримки веб-додатку.

Головна директорія проєкту містить основні конфігураційні файли, які виконують ключові функції налаштування та керування Django-проєктом:

settings.py - Центральний файл налаштувань для всього Django-проєкту. Тут визначаються параметри підключення до бази даних, встановлені додатки, конфігурації безпеки, такі як секретні ключі, а також налаштування середовища, наприклад, список хостів, на яких додаток може бути розгорнутий.

urls.py - Визначає схему URL-адрес та маршрутизацію HTTP-запитів до відповідних представлень Django. Цей файл містить патерни URL-шляхів та зіставляє їх з функціями або класами-представленнями, які обробляють відповідні запити.

asgi.py - Налаштування Асинхронного Шлюзу Сервера-Інтернету (ASGI - Asynchronous Server Gateway Interface), які забезпечують підтримку асинхронних функцій, таких як WebSocket. ASGI дозволяє Django працювати з асинхронними протоколами, що є критично важливим для реалізації функціональності чату в режимі реального часу.

Функціональність веб-додатку для обміну мовами між носіями розподілена між трьома основними застосунками:

chat - Цей застосунок повністю відповідає за реалізацію функціоналу чату, включаючи створення кімнат для спілкування 1-на-1 та групових чатів, надсилання та отримання повідомлень, управління запрошеннями та налаштуваннями чатів. Він містить моделі даних для представлення кімнат, повідомлень та пов'язаних об'єктів, а також представлення та шаблони для обробки запитів та візуалізації інтерфейсу чату.

signup - Цей застосунок забезпечує можливість реєстрації та автентифікації користувачів за допомогою облікових записів Google та електронної пошти. Він використовує бібліотеку django-allauth для спрощення процесу автентифікації та інтеграції з провайдерами автентифікації.

main_app - Цей застосунок містить основні сторінки та функціональність веб-додатку, такі як:

1. Головна сторінка (name="index-page");
2. Сторінка профілю користувача (path="profile-page");
3. Створення особистого чату (path="create-presonal-chat");
4. Оновлення профілю (path="update_profile");
5. Оновлення зображення профілю (path="update_img");
6. Сторінка фільтру пошуку користувачів (path="filter-page");
7. Сторінка контактів (path="contact-page");
8. Інформація про користувача (path="userInfo");
9. Сторінка чорного списку користувачів (path="black-list-page");
10. Блокування користувача (path="block-user");
11. Розблокування користувача (path="unblock-user").

Крім основних застосунків, проєкт містить додаткові важливі директорії та файли:

1. **templates** - Загальна директорія для шаблонів, що містить базовий шаблон base.html, який успадковують інші шаблони застосунків. Це забезпечує уніфікований дизайн та структуру для всіх сторінок веб-додатку.
2. **.venv** - Віртуальне середовище для ізоляції залежностей проєкту. Використання віртуального середовища забезпечує незалежність між проєктами та спрощує управління встановленими пакетами та бібліотеками.
3. **.gitignore** - Файл, що визначає виключення для системи контролю версій Git. Це дозволяє уникнути додавання в репозиторій непотрібних файлів, таких як тимчасові файли, файли налаштувань середовища та інші артефакти розробки.
4. **.pre-commit-config.yaml** - Налаштування pre-commit хуків для автоматизованої перевірки коду перед відправленням у репозиторій. Це допомагає підтримувати високу якість коду та уникати потенційних помилок.
5. **poetry.lock та poetry.toml** - Файли залежностей та налаштувань для менеджера пакетів Poetry. Poetry використовується для управління залежностями проєкту та забезпечує відтворюваність встановлених бібліотек.
6. **pyproject.toml** - Файл конфігурації проєкту Python, що містить метадані проєкту та налаштування інструментів для роботи з кодом.
7. **ruff_cache** - Кеш для інструменту статичного аналізу коду Ruff. Ruff використовується для перевірки коду на відповідність кодовим стандартам та виявлення потенційних помилок або неефективного коду.

На рисунку 3.1 можна побачити загальну структуру проєкту, яка використовувалася для його реалізації.

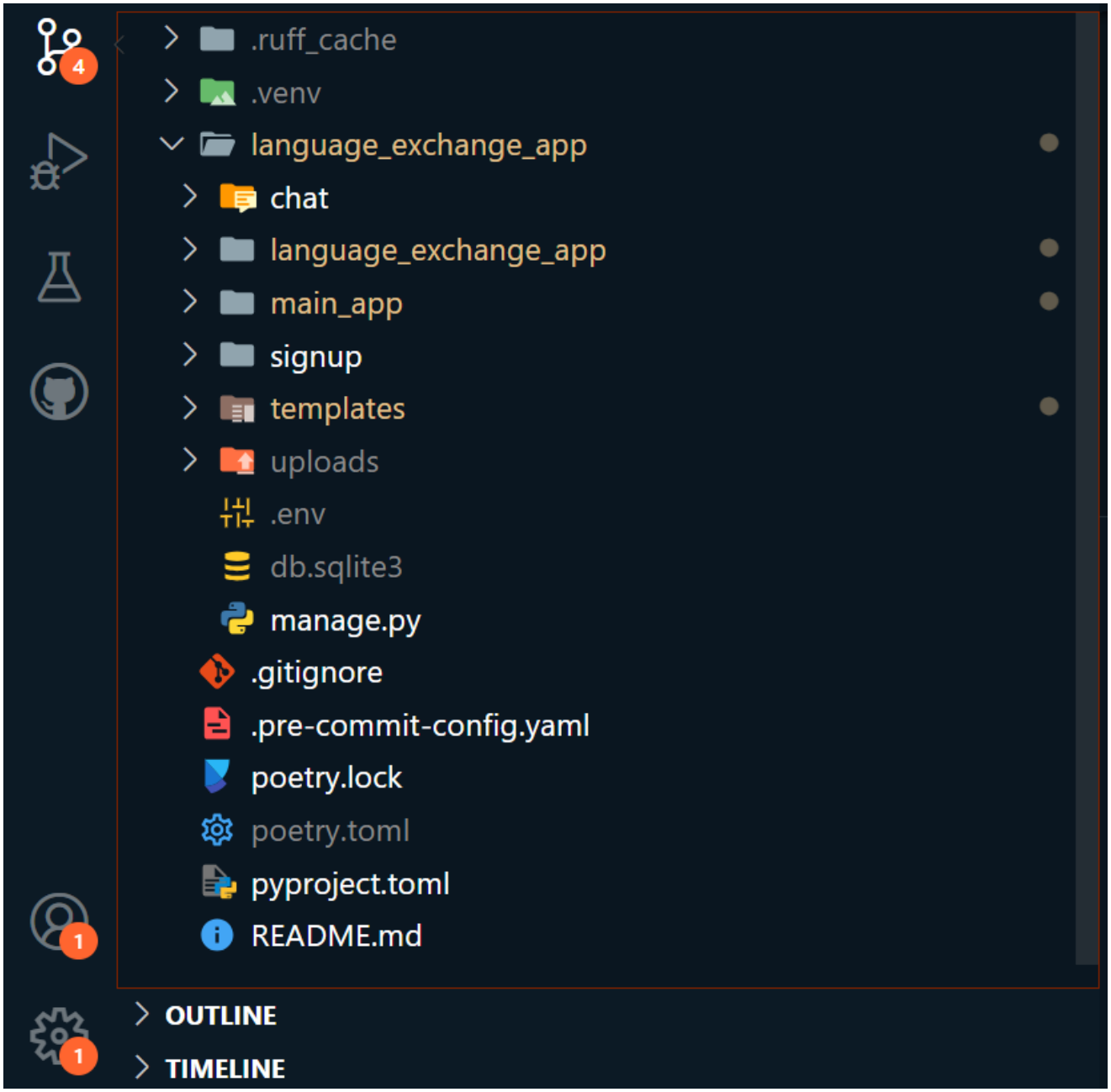

Рисунок 3.1 – Загальна структура веб-додатку

Дана структура відповідає рекомендованим практикам та загальноприйнятим шаблонам організації коду Django-проєктів, забезпечуючи модульність, масштабованість, простоту підтримки та розширюваність веб-додатку для обміну мовами між носіями.

3.2 Email реєстрація

Функціональність реєстрації користувачів через електронну пошту у веб-додатку для обміну мовами між носіями реалізована у класі Register, успадкованому від стандартного Django-класу View. Цей клас відповідає за обробку GET та POST запитів, пов'язаних із процесом реєстрації. Даний підхід має назву представлення на основі класів (Class based Views).

```
class Register(View):
    template_name = (
        "../templates/main_app/register.html"
    )
    def get(self, request):
        context = {"form": SignUpForm()}
        return render(request, self.template_name, context)

    def post(self, request):
        if request.method == "POST":
            form = SignUpForm(request.POST)
            if form.is_valid():
                form.save()
                username = form.cleaned_data.get("username")
                password = form.cleaned_data.get("password1")

                user = authenticate(username=username, password=password)
                login(request, user)
                UserProfile.objects.create(user=user)
                return HttpResponseRedirect(reverse("profile-page"))
        context = {"form": form}
        return render(request, self.template_name, context)
```

При отриманні GET запиту на сторінку реєстрації, метод get класу Register ініціалізує екземпляр форми реєстрації SignUpForm, який є спеціально створеним класом форми Django Forms. Форма SignUpForm передається в контекст render для візуалізації у шаблоні register.html.

Під час обробки POST запиту у методі post, спочатку відбувається перевірка, чи надійшов запит методом POST. Якщо так, то створюється екземпляр SignUpForm з даними, отриманими від клієнта у request.POST. Далі відбувається валідація даних форми SignUpForm за допомогою методу is_valid().

Валідація форми SignUpForm включає стандартну валідацію Django Forms, наприклад, перевірку обов'язкових полів, типів даних та форматів. Додатково форма SignUpForm забезпечує автозаповнення поля електронної пошти та перевірку унікальності імені користувача в базі даних.

Якщо дані форми валідні, відбувається збереження даних нового користувача у базі даних за допомогою методу form.save(). Після цього, з очищених даних форми (form.cleaned_data) отримуються ім'я користувача (username) та пароль (password1). Ці дані використовуються для автентифікації нового користувача за допомогою функції authenticate з django.contrib.auth. Далі функція login з того ж модуля виконує процес входу автентифікованого користувача у систему.

Після успішної реєстрації та входу користувача створюється екземпляр моделі UserProfile, пов'язаний з новим користувачем через зв'язок один-до-одного (one-to-one) з моделлю User. Модель UserProfile зберігає додаткові атрибути користувача, такі як вибрані мови, опис, стать, місце проживання, дата народження та фото профілю. Поля моделі UserProfile заповнюються користувачем після реєстрації.

Процес реєстрації завершується перенаправленням користувача на сторінку профілю за допомогою HttpResponseRedirect та reverse для маршрутизації URL-адрес Django.

У випадку невалідних даних форми, контекст з формою передається у render для повторного відображення сторінки реєстрації з відповідними повідомленнями про помилки.

3.3 Моделі та база даних

Веб-додаток для обміну мовами використовує об'єктно-реляційну модель даних Django для зберігання та управління ними. Ця модель тісно інтегрована із

системою керування базами даних (СКБД) PostgreSQL, яка забезпечує надійне та масштабоване середовище для зберігання даних.

Ключовими моделями у проєкті є UserProfile, Chat та Message (Додаток 2).

Модель UserProfile пов'язана з моделлю User з django.contrib.auth.models за допомогою відношення один-до-одного та є розширеною User моделлю. UserProfile містить додаткову інформацію про користувача, а саме поля:

1. description (CharField) - текстовий опис або біографія користувача;
2. gender (CharField) - стать користувача, з обмеженим вибором значень;
3. residence (TextField) - місце проживання користувача;
4. birth_date (DateField) - дата народження;
5. photo (ImageField) - завантажене фото профілю користувача.

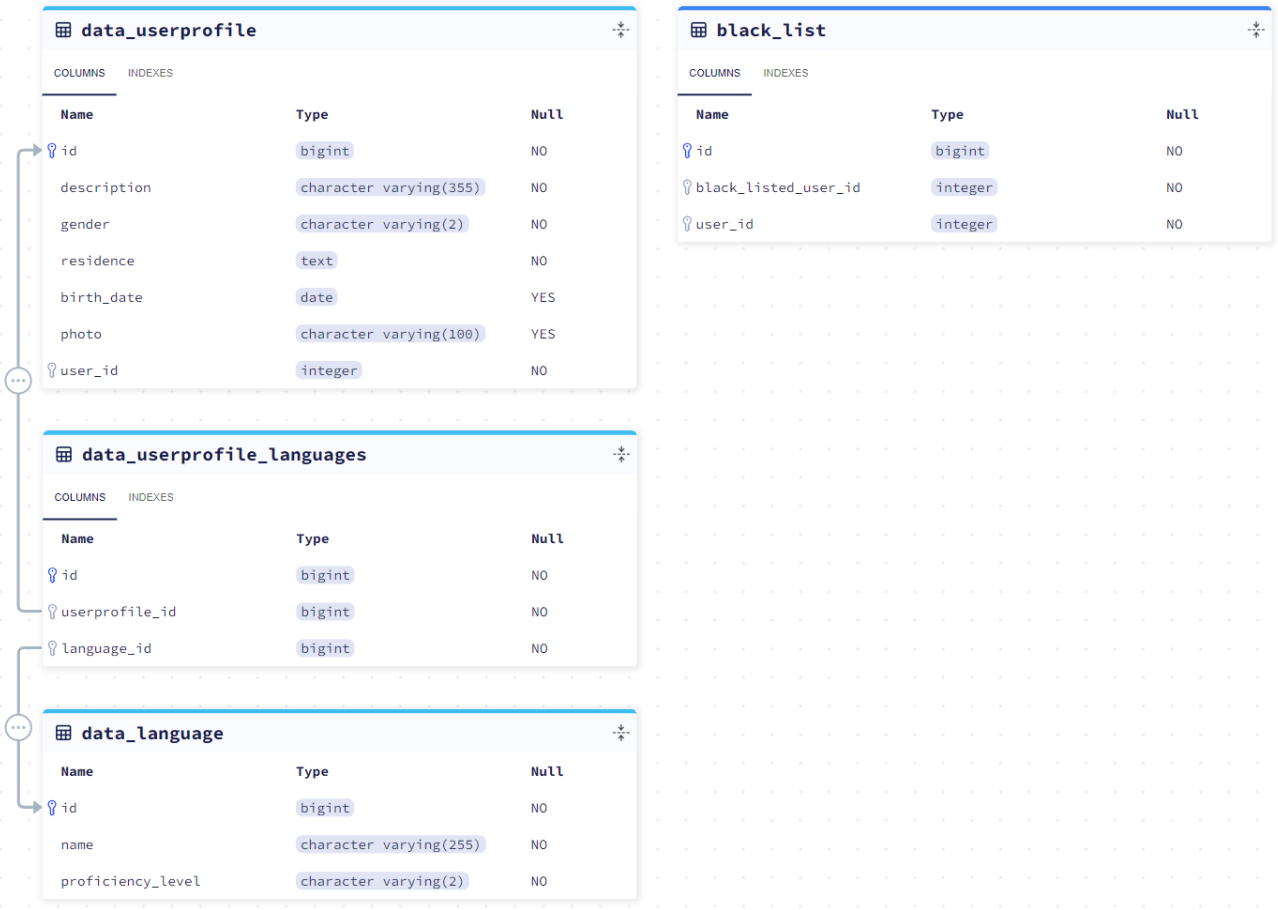

Рисунок 3.2 – Схема зв’язків бази даних між профілем користувача

Модель Chat представляє кімнати чату та пов'язана з моделлю User за допомогою відношення багато-до-багатьох. Вона містить такі поля:

1. name (TextField) - назва кімнати чату;
2. users (ManyToManyField до User) - відношення багато-до-багатьох з моделлю User для зберігання учасників кімнати;
3. slug (SlugField) - унікальний слаг для ідентифікації кімнати чату;
4. create_at (DateTimeField) - дата та час створення кімнати, генерується автоматично;
5. created_by (ForeignKey до User) - зовнішній ключ для користувача, який створив кімнату чату.

Ця модель є центральною у реалізації функціональності чатів, групуючи користувачів та їхні повідомлення.

Модель Message пов'язана з Chat та User за допомогою зовнішніх ключів (ForeignKey). Ця модель зберігає окремі повідомлення, надіслані у чатах, та містить такі поля:

1. chat (ForeignKey до Chat) - зовнішній ключ, що пов'язує повідомлення з певною кімнатою чату
2. user (ForeignKey до User) - зовнішній ключ для користувача, який надіслав це повідомлення
3. content (TextField) - текстовий вміст повідомлення
4. media_file (FileField) - необов'язковий медіа-файл, прикріплений до повідомлення
5. date_added (DateTimeField) - дата та час створення повідомлення, автоматично генерується
6. update_at (DateTimeField) - дата та час оновлення повідомлення, може бути null

На рисунку 3.3 показаний детальний вигляд зв’язків між цими моделями.

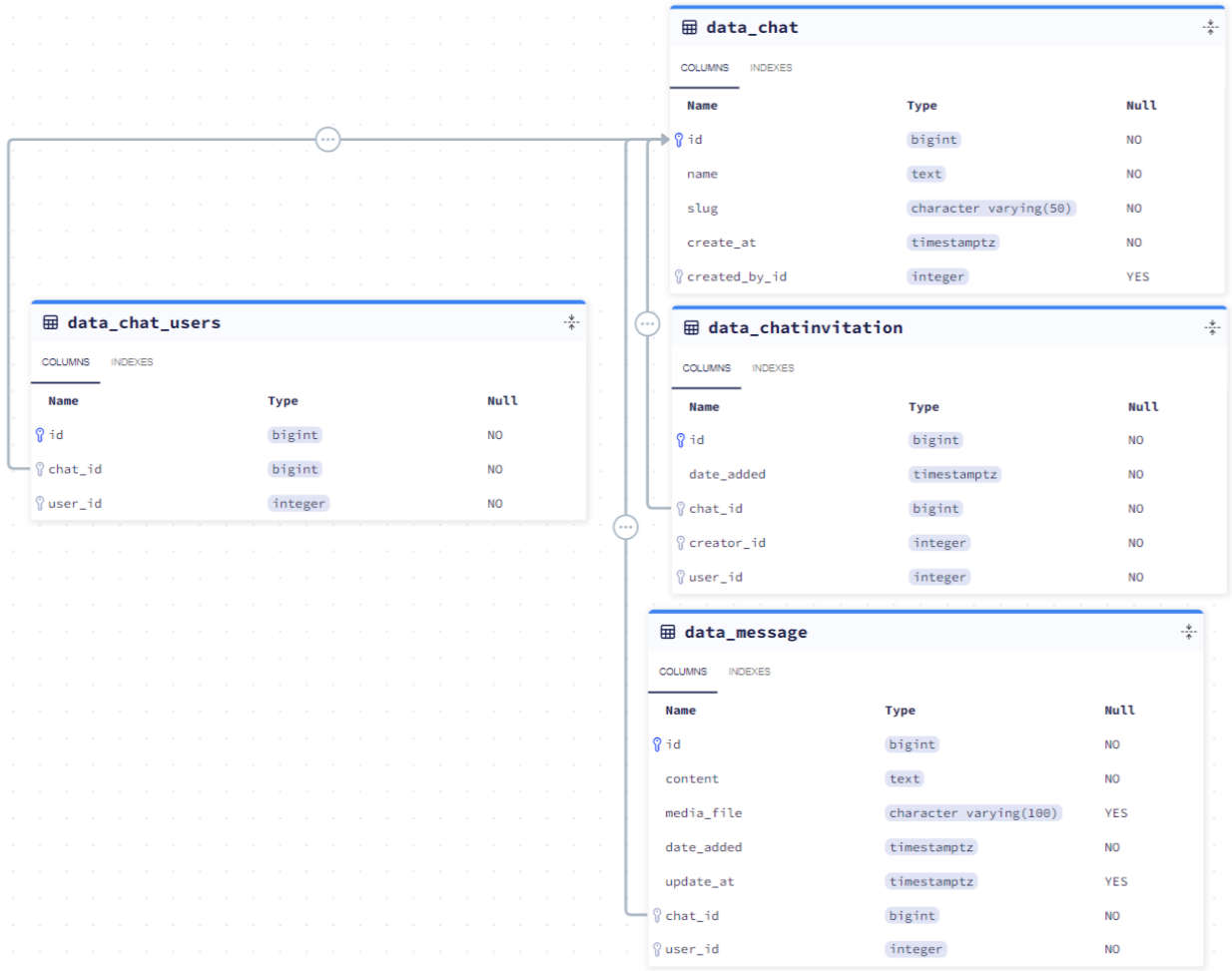

Рисунок 3.3 – Схема зв’язків бази даних між чатом та повідомленнями

Додаткові моделі BlackList, ChatInvitation та Language відіграють допоміжні ролі. BlackList використовується для зберігання інформації про заблокованих користувачів, ChatInvitation - для відстеження запрошень до чатів, а Language - для зберігання доступних мов та рівнів володіння ними.

Також було використано стандартні Django моделі, які відповідають за управління користувачами, надання дозволів та адміністрування. Детально побачити схему цих моделей можна на рисунку 3.4.

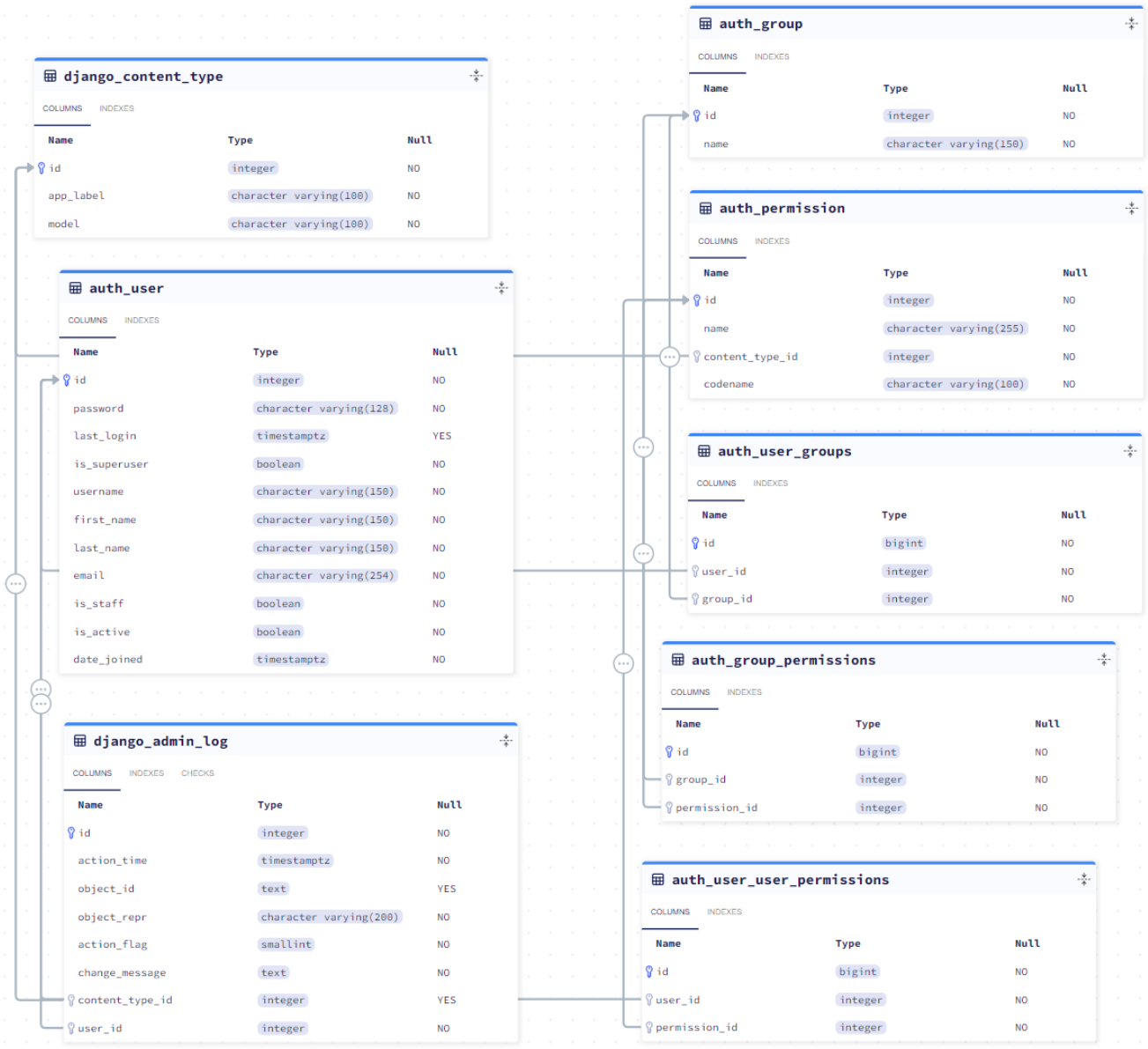

Рисунок 3.3 – Схема зв’язків бази даних між стандартними моделями Django

3.4 Реалізація функціоналу головної сторінки

Функціональність головної сторінки веб-додатку для обміну мовами між носіями реалізована у двох класах: функція index та клас SearchView, успадкований від стандартного Django-класу ListView. Ці представлення відповідають за обробку запитів, пов'язаних із відображенням та пошуком користувачів на головній сторінці.

Фукнція index:

```
@login_required
def index(request):
    current_user = UserProfile.objects.get(user=request.user)
    users = UserProfile.objects.order_by("?").exclude(id=current_user.id)[:10]
    context = {
        "users": users,
    }
    return render(request, "main_app/index.html", context)
```

При отриманні запиту на головну сторінку, представлення index спочатку отримує об'єкт UserProfile поточного авторизованого користувача з бази даних. Далі,за допомогою UserProfile.objects.order_by("?").exclude(id=current_user.id)[:10] формується список з 10 випадкових об'єктів UserProfile, які не співпадають з поточним користувачем. Сформований список передається у контекст render для відображення у шаблоні main_app/index.html.

Таким чином, представлення index забезпечує відображення списку випадкових користувачів на головній сторінці, коли не вказано, які саме мови він хоче вивчати.

Представлення SearchView реалізоване як клас, успадкований від ListView:

```
@method_decorator(login_required, name='dispatch')
class SearchView(ListView):
    model = UserProfile
    template_name = "main_app/index.html"

    def get_queryset(self):
        current_user = UserProfile.objects.get(user=self.request.user)
        username = self.request.GET.get("q")
        object_list = UserProfile.objects.all().exclude(id=current_user.id)
        if username and username != "" and username != current_user.user.username:
            object_list =
UserProfile.objects.filter(user__username__icontains=username).exclude(id=current_
user.id)
        return object_list

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['users'] = self.get_queryset()
        return context
```

Це представлення відповідає за пошук та фільтрацію користувачів за іменем та вивченими мовами на головній сторінці. Воно захищене декоратором @method_decorator(login_required, name='dispatch'), що дозволяє лише авторизованим користувачам отримувати доступ до нього.

У методі get_queryset спочатку отримується об'єкт UserProfile поточного користувача з бази даних. Далі витягується рядок пошуку з GET-параметрів запиту. Створюється базовий QuerySet object_list з усіх об'єктів UserProfile, крім поточного користувача. Якщо рядок пошуку наданий, не порожній та не співпадає з іменем поточного користувача, до object_list застосовується фільтр, який шукає користувачів, імена яких містять рядок пошуку (нечутливий до регістру). Отриманий object_list повертається як кінцевий результат.

У методі get_context_data отриманий object_list (список знайдених користувачів) додається до контексту під ключем 'users'.

Шаблон main_app/index.html отримує контекст з ключем 'users', який містить список користувачів, відфільтрованих за іменем або вивченими мовами (представлення SearchView), або випадковим чином згенерованих (представлення index). Цей список використовується для візуалізації користувачів на головній сторінці веб-додатку.

На головній сторінці присутні такі елементи інтерфейсу:

1. Бічна панель навігації, яка забезпечує доступ до різних розділів веб-додатку, таких як профіль користувача, список чатів, налаштування тощо.
2. Форма пошуку користувачів за іменем. Введений рядок пошуку обробляється представленням SearchView для фільтрації списку користувачів.
3. Форма фільтрації користувачів за вивченими мовами. Ця форма дозволяє користувачу вибрати одну або кілька мов, які він вивчає, і відфільтрувати список користувачів на головній сторінці за цими критеріями.

Якщо користувач не авторизований у системі, він буде перенаправлений на сторінку входу/реєстрації, оскільки представлення index та SearchView захищені декораторами, які вимагають автентифікації користувача.

Таким чином, функціональність головної сторінки веб-додатку для обміну мовами між носіями реалізована за допомогою двох представлень: index та SearchView. Перше представлення забезпечує відображення списку випадкових користувачів, а друге – пошук та фільтрацію користувачів за іменем та вивченими мовами. На головній сторінці також присутні елементи інтерфейсу для навігації по веб-додатку, пошуку користувачів та фільтрації користувачів за мовами. Доступ до головної сторінки можливий лише для авторизованих користувачів.

3.5 Програмна реалізація функціоналу чату

3.5.1 Створення чат-кімнат

Функціональність створення нових чат-кімнат між користувачами у веб-додатку, реалізована у функції create_chat. Вона використовує підхід представлення на основі функцій (Function-based Views) та зв'язана з навігацією у файлі urls.py шляхом застосування відповідного патерну URL-адреси.

Дана функція призначена для обробки HTTP POST запитів, що надходять від клієнтської частини веб-додатку. Функція create_chat очікує отримати у тілі POST запиту три обов'язкові параметри: name, users та created_by.

```
@csrf_exempt
def create_chat(request):
    name = request.POST.get("name", None)
    users = request.POST.getlist("users[]", None)
    created_by_id = request.POST.get("created_by", None)
    data = {"error": None}
    if name and users and created_by_id:
        try:
            created_by = User.objects.get(id=created_by_id)
```

```
            chat = Chat(name=name, slug=uuid.uuid4(), created_by=created_by)
            chat.save()
            for user_id in users:
                user = User.objects.get(id=user_id)
                invitation = ChatInvitation(
                    chat=chat, user=user, creator=created_by
                )
                invitation.save()
            chat.save()
        except User.DoesNotExist:
            data["error"] = "chat create error"
    else:
        data["error"] = "All field required"
    return JsonResponse({"redirect": "/chat/"})
```

Параметр name представляє собою рядок, що містить назву чат-кімнати, яку бажає створити користувач. Параметр users є масивом цілочисельних значень, кожне з яких представляє ідентифікатор користувача (ID користувача в таблиці User), якого потрібно запросити до новоствореної чат-кімнати. Параметр created_by також є цілочисельним значенням, що репрезентує ідентифікатор користувача, який ініціював процес створення нової чат-кімнати.

Після отримання цих параметрів функція create_chat виконує перевірку на їх наявність та коректність. Якщо усі необхідні параметри присутні, здійснюється спроба отримати об'єкт користувача (User) з бази даних за переданим ідентифікатором created_by. У випадку успіху створюється новий об'єкт Chat з наданою назвою (name) та унікальним ідентифікатором (slug), згенерованим за допомогою функції uuid.uuid4(). Новостворений об'єкт Chat пов'язується з користувачем, що його створив, через поле created_by.

Далі функція create_chat ітерується по масиву users та для кожного ID користувача отримує відповідний об'єкт User з бази даних. Для кожного такого користувача створюється новий об'єкт ChatInvitation, який пов'язує цього користувача з новоствореною чат-кімнатою та зберігає інформацію про те, що цей користувач був запрошений до чату користувачем, вказаним у полі creator.

Після успішного створення чат-кімнати, збереження її в базі даних та генерації усіх необхідних запрошень для користувачів, функція create_chat виконує перенаправлення на сторінку зі списком активних чатів користувача за допомогою повернення об'єкта JsonResponse з ключем "redirect" та значенням "/chat/".

3.5.2 Оновлення даних чат-кімнат

Можливість оновлення інформації про існуючі чат-кімнати у поєкті реалізована у функції update_chat. Вона також використовує підхід представлення на основі функцій, як і функція create_chat.

Важливою особливістю функції update_chat є те, що вона захищена декоратором @login_required, який гарантує доступ лише для автентифікованих користувачів системи. Функція також перевіряє, чи поточний користувач (request.user) є власником чат-кімнати, що оновлюється (room.created_by). Якщо ця умова не виконується, викликається виняток PermissionDenied, який забороняє несанкціонований доступ до функції та унеможливлює небажані дії з чат-кімнатою.

Функція update_chat призначена для обробки HTTP POST запитів, що надходять від клієнтської частини веб-додатку. Вона очікує отримати у тілі POST запиту декілька параметрів, необхідних для оновлення властивостей чат-кімнати. Серед них:

1. name - рядок, що містить нову назву чат-кімнати.
2. owner_id - ідентифікатор користувача, який має стати новим власником чат-кімнати.
3. user_ids[] - масив ідентифікаторів користувачів, які повинні бути включені до оновленого списку користувачів чат-кімнати.

Після отримання цих параметрів, функція update_chat виконує наступні дії:

1. Здійснюється пошук існуючої чат-кімнати в базі даних за переданим slug за допомогою методу get_object_or_404 з моделі Chat.
2. Назва чат-кімнати оновлюється згідно з отриманим параметром name.
3. Власник чат-кімнати оновлюється шляхом пошуку об'єкта User за переданим owner_id та присвоєння його до поля created_by моделі Chat.
4. Виконується обробка нового списку користувачів чат-кімнати:
    - Отримується старий список користувачів чат-кімнати за допомогою chat.users.values_list("id", flat=True).
    - Формується новий список користувачів на основі переданого масиву user_ids[].
    - Знаходяться користувачі, яких потрібно додати до чат-кімнати, шляхом віднімання старого списку від нового.
    - Для кожного нового користувача, що був доданий до списку, створюється об'єкт ChatInvitation, який зв'язує цього користувача з оновленою чат-кімнатою та зберігає інформацію про те, що цей користувач був запрошений до чату поточним користувачем (request.user).
5. Оновлений список користувачів зберігається у чат-кімнаті за допомогою методу chat.users.set(users), де users - це QuerySet об'єктів User, сформований на основі переданого масиву user_ids[].
6. Змінена чат-кімната зберігається у базі даних за допомогою методу chat.save().

Після успішного оновлення всіх властивостей чат-кімнати, функція update_chat повертає JSON-відповідь зі статусом "success". У випадку, якщо запит не є POST запитом, функція повертає JSON-відповідь з повідомленням про помилку "Invalid request".

РОЗДІЛ 4. ДЕМОНСТРАЦІЯ ПРОЄКТУ

4.1 Вигляд сторінок реєстрації та входу

Процес реєстрації нових користувачів та входу в систему є критично важливим аспектом для забезпечення ефективної взаємодії та використання веб-додатку для обміну мовами. З цією метою було розроблено інтуїтивно зрозумілий та лаконічний інтерфейс, який відповідає принципам зручності та простоти використання. Дизайн сторінки реєстрації продемонстровано на рис. 4.1.

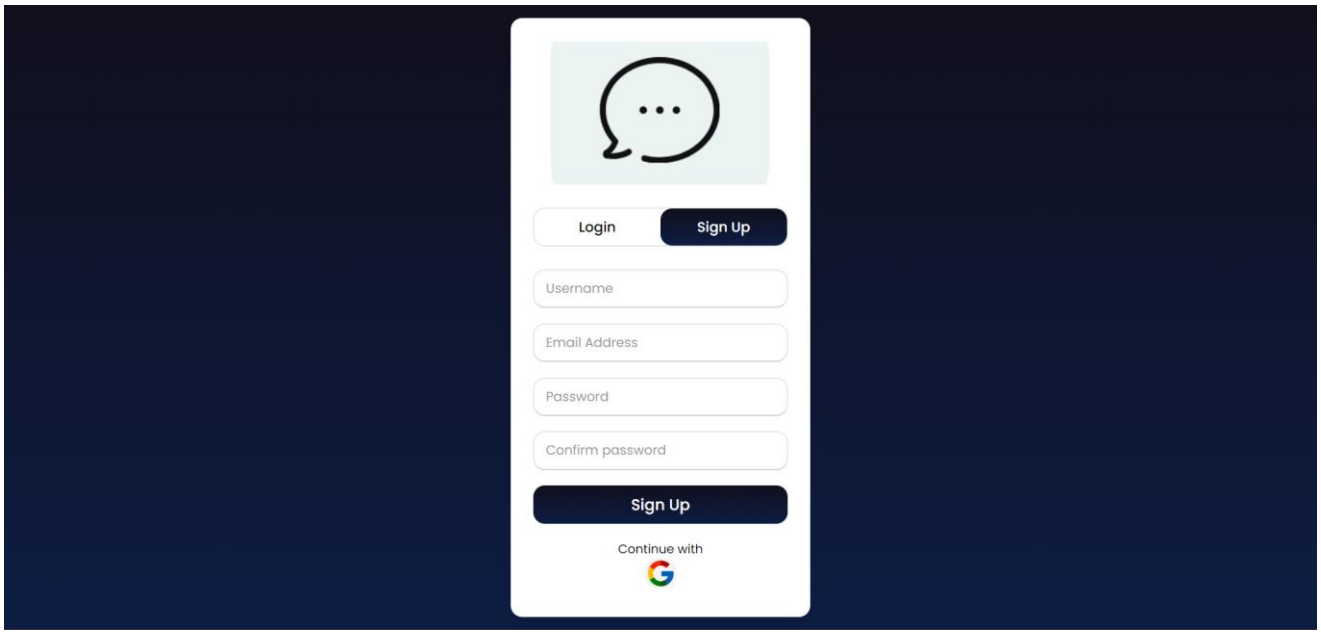

Рисунок 4.1 – Сторінка реєстрації

На сторінці реєстрації реалізовано поля для вводу наступних даних:

1. Ім'я користувача (**Username**);
2. Електронна адреса (**Email Address**);
3. Пароль (**Password**);
4. Підтвердження пароля (**Confirm Password**).

Ці поля чітко позначені та розташовані в логічному порядку, що забезпечує зручність введення необхідної інформації для створення нового облікового запису.

Після заповнення всіх полів користувачеві залишається лише натиснути кнопку "Зареєструватися" (**Sign Up**), розташовану нижче.

Для прискорення та спрощення процесу реєстрації було реалізовано можливість автентифікації через обліковий запис Google за допомогою окремої кнопки входу через Google у нижній частині сторінки реєстрації. При виборі цього способу реєстрації користувачеві не потрібно вводити пароль, електронну адресу та ім'я користувача, оскільки ці дані автоматично беруться з його облікового запису Google. У випадку, якщо згенероване ім'я користувача виявиться неунікальним, користувач має можливість легко змінити його одразу на сторінці свого профілю після успішної реєстрації.

Також було реалізовано зручну кнопку-слайдер у верхній частині сторінки реєстрації, яка забезпечує безперешкодний перехід між сторінками реєстрації та входу в обліковий запис, підвищуючи зручність навігації та користувацький досвід.

Сторінка входу на сайт також відрізняється простим та зручним інтерфейсом, що можна побачити на рис. 4.2.

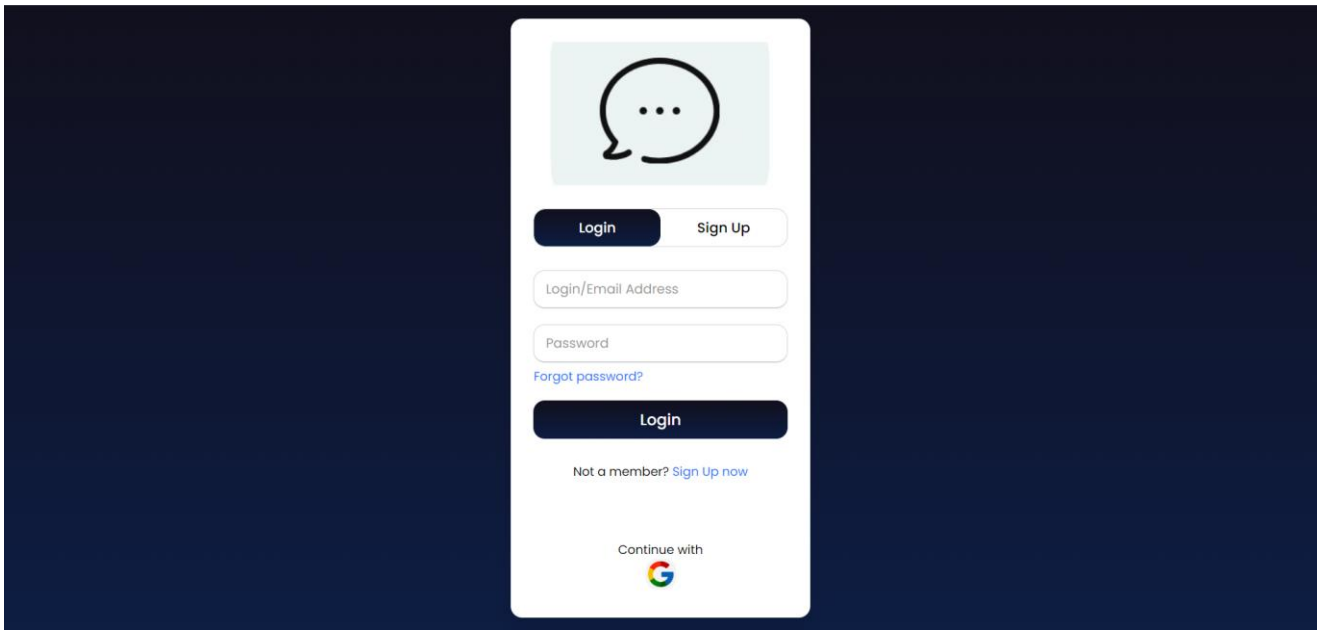

Рисунок 4.2 – Сторінка входу

На ній присутні два поля для вводу даних: "Логін/Електронна адреса" (**Login/Email Address**) та "Пароль" (**Password**). Користувач може ввести або свій логін, або електронну адресу, пов'язану з його обліковим записом.

Нижче цих полів розташована кнопка "Увійти" (**Login**), яка дозволяє здійснити вхід в систему після коректного введення даних. Кнопка "Увійти" є більш виділеною та розташована вище, ніж кнопка автентифікації через Google. Також присутня кнопка входу через обліковий запис Google, яка забезпечує швидку автентифікацію без необхідності вводити логін та пароль. При виборі цього способу входу дані беруться автоматично з облікового запису Google користувача, що є досить зручно зі сторони клієнта та надійно для перевірки даних для авторизації.

Однією із зручних особливостей сторінки входу є наявність валідації, що перевіряє коректність введених даних та інформує користувача про будь-які помилки або невідповідності, забезпечуючи зручність та зрозумілість процесу входу в систему.

Загалом, сторінка входу має продуманий та інтуїтивно зрозумілий дизайн, що вдало поєднується з сторінкою реєстрації та забезпечує безперешкодний доступ користувачів до функціоналу веб-додатку.

4.2 Відображення сторінки власного профілю та профілю іншого користувача

Сторінка власного профілю користувача має зручний та інтуїтивний інтерфейс для перегляду та редагування персональної інформації.

У центральній частині верхньої секції сторінки розташовано фото профілю користувача, яке можна легко змінити натиснувши на нього – ця дія викличе файловий провідник пристрою для вибору нового зображення. Зліва від фото профілю знаходиться кнопка для повернення на головну сторінку додатку, тоді як справа розміщена кнопка, що забезпечує перехід до списку активних чатів користувача. На рис. 4.3 можна побачити розташування всіх елементів.

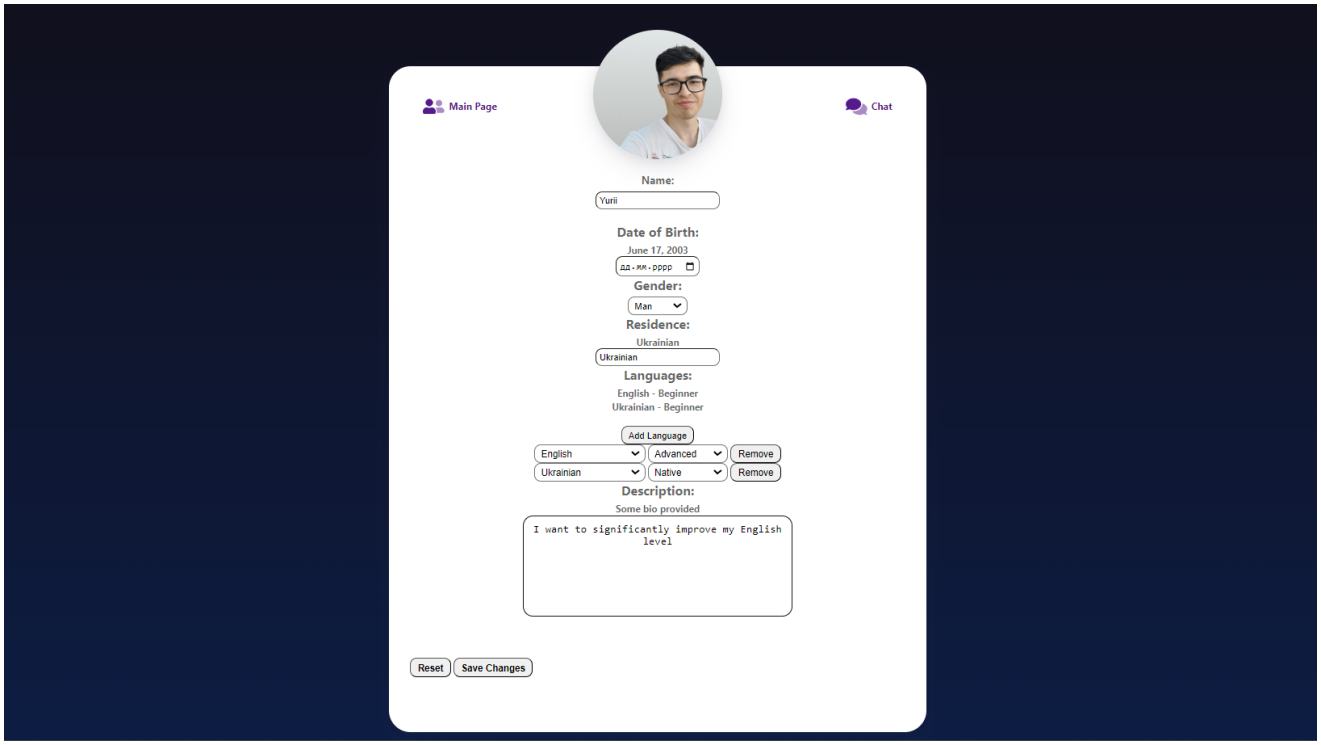

Рисунок 4.3 – Профіль користувача

Нижче, у основній частині сторінки, розташовані поля для редагування ключових даних профілю користувача:

1. Ім'я (**Name**) - поле для введення або зміни імені користувача, яке буде відображатися іншим учасникам додатку;
2. Дата народження (**Date of Birth**) - можливість вказати свою дату народження для додаткової персоналізації профілю;
3. Стать (**Gender**) - опція для визначення статі користувача;
4. Місце проживання (**Residence**) - поле для введення інформації про місце проживання або походження користувача;
5. Мови (**Languages**) - тут користувач може вказати мови, якими він володіє, для полегшення пошуку потенційних партнерів для обміну мовами;
6. Опис (**Description**) - розділ для додавання детального опису себе, своїх інтересів, мети використання додатку тощо.

Ці поля дозволяють користувачеві вносити або оновлювати свої персональні відомості, забезпечуючи актуальність та повноту інформації, що відображається на його сторінці. Після внесення бажаних змін, користувач може скористатися однією з двох кнопок, розташованих у нижній частині сторінки – "Скинути" (**Reset**) для відновлення початкових даних або "Зберегти зміни" (**Save Changes**) для підтвердження та збереження внесених правок.

Під час перегляду профілю іншого користувача сторінка набуває дещо іншого вигляду та функціоналу.

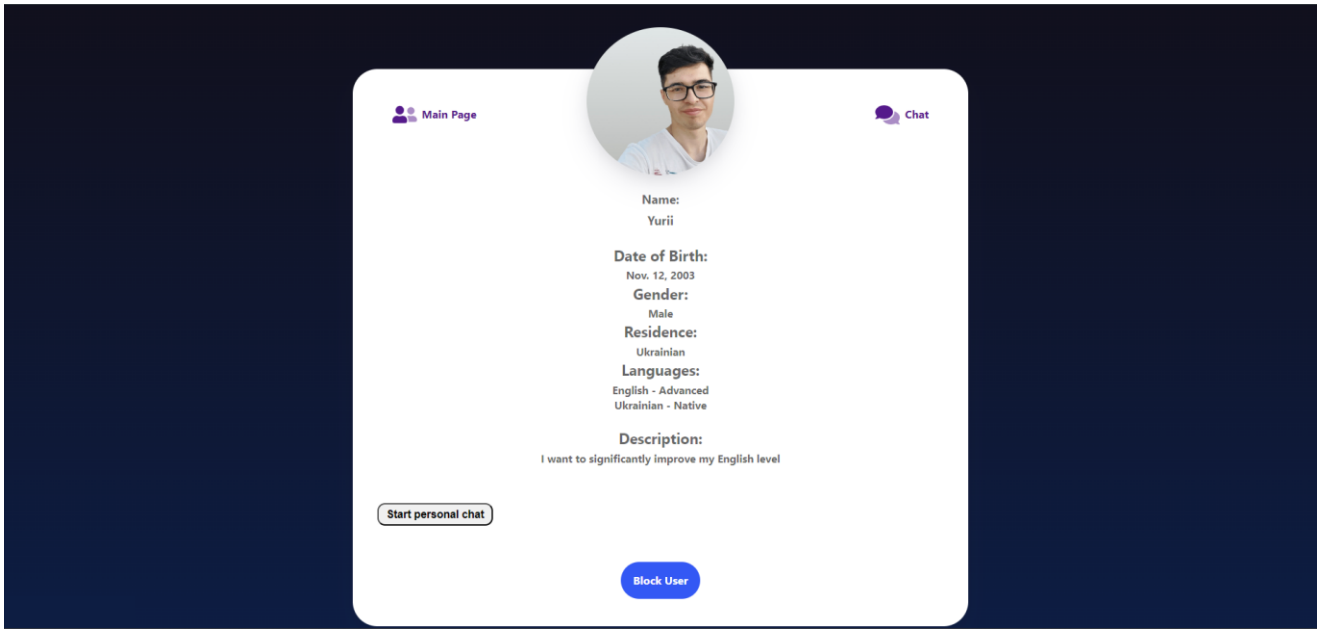

Рисунок 4.4 – Вигляд профілю іншого користувача

Як видно на рисунку 4.4, замість інтерактивних полів редагування, відомості про цього користувача відображаються у вигляді звичайного тексту для ознайомлення. Проте, керуючі елементи у верхній частині сторінки залишаються незмінними – зліва від фото профілю розташована кнопка повернення на головну сторінку додатку, а справа – кнопка переходу до списку чатів.

У нижній секції сторінки розміщені дві ключові кнопки для взаємодії з іншим користувачем. Перша – "Розпочати особистий чат" (**Start Personal Chat**), яка дозволяє відкрити приватний чат для безпосереднього спілкування та обміну мовами з цією особою. Друга – "Заблокувати користувача" (**Block User**), надає

можливість обмежити взаємодію та приховати профіль цього користувача у разі необхідності.

Обидві сторінки профілю, як власного, так і іншого користувача, мають привабливий та зручний дизайн з інтерактивним оформленням кнопок. Ці елементи керування реагують на наведення курсору миші, демонструючи ефекти наведення (hover) та активації (active), що забезпечує приємний та інтуїтивно зрозумілий користувацький досвід під час користування додатком для обміну мовами.

4.3 Вигляд головної сторінки

Головна сторінка веб-додатку для обміну мовами відрізняється сучасним та інтуїтивним дизайном, який забезпечує зручність навігації та взаємодії з ключовими функціями додатку.

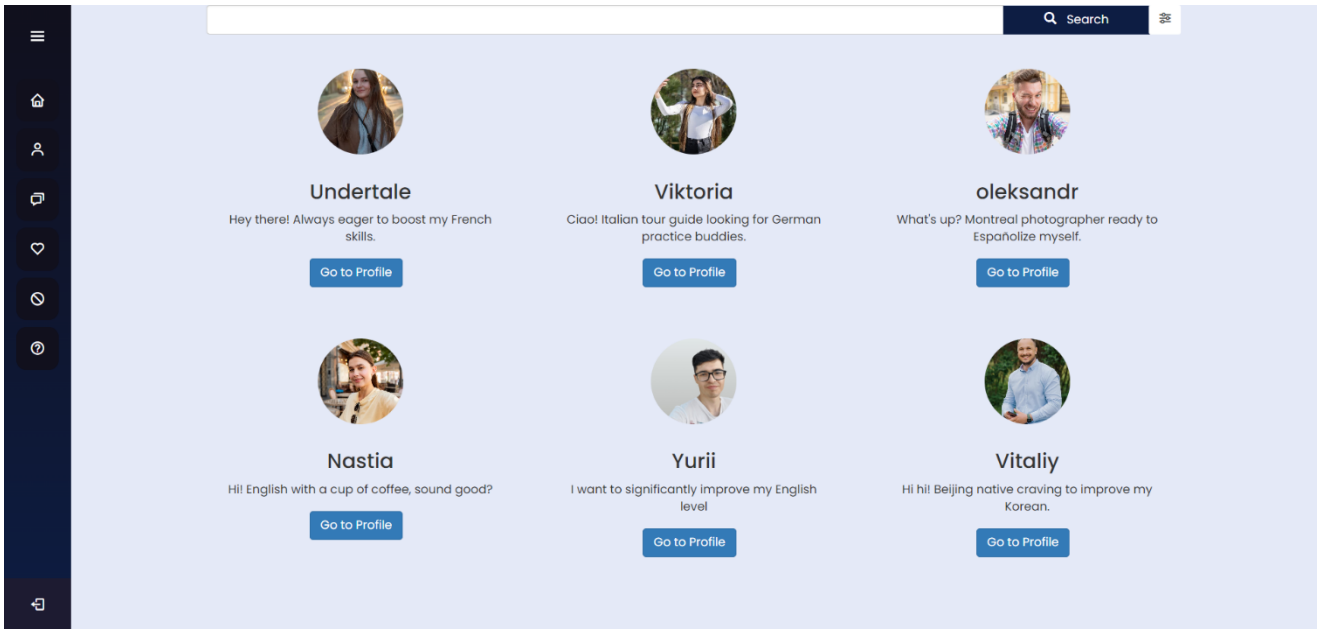

Рисунок 4.5 – Демонстрація головної сторінки з розширенням екрану 16:9

Ліва частина сторінки відведена під бічну панель навігації, яка може висуватися та приховуватися за допомогою спеціальної кнопки-тригера. Під час розгортання бічної панелі відбувається приємна анімація, що надає інтерфейсу динамічності та плавності. На цій панелі розміщені іконки основних розділів додатку, які супроводжуються підписами для зручності ідентифікації.

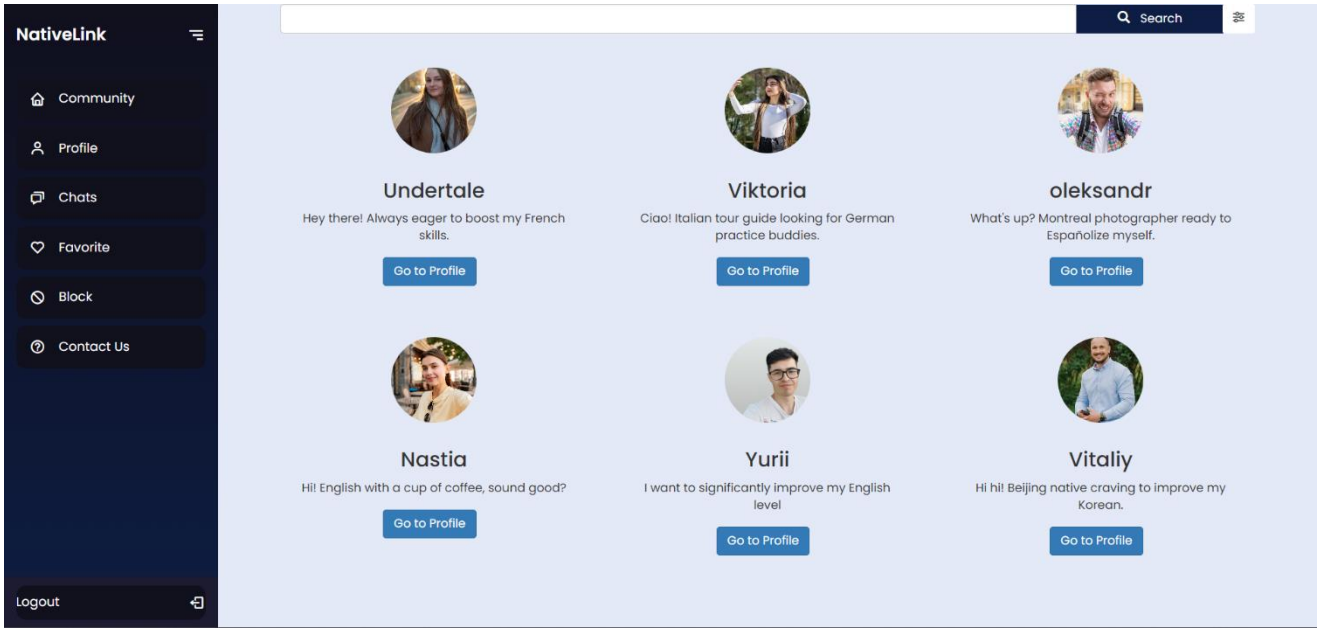

Рисунок 4.6 – Вигляд головної сторінки з відкритою панеллю навігації

У верхній частині головної області сторінки знаходиться поле пошуку, що дозволяє користувачам швидко знаходити потрібних співрозмовників за певними критеріями. Поряд з ним розташовані елементи керування фільтрами, які надають можливість відсортувати список користувачів за мовами, місцем проживання або іншими релевантними параметрами.

Основну частину головної сторінки займає сітка з профілями інших користувачів. Кожен профіль представлений у вигляді карточки, що містить фото профілю, коротку інформацію про користувача та кнопку "Go to Profile" для переходу на його сторінку. На великих екранах ці карточки розміщуються по три в ряду, забезпечуючи компактне та ефективне використання простору.

Для мобільних пристроїв реалізовано адаптивний дизайн, який автоматично підлаштовується під різні роздільні здатності екрану. На невеликих екранах профілі користувачів відображаються по одному в ряду, забезпечуючи зручність перегляду та взаємодії. При цьому бічна панель навігації зберігає свою функціональність та анімацію розгортання.

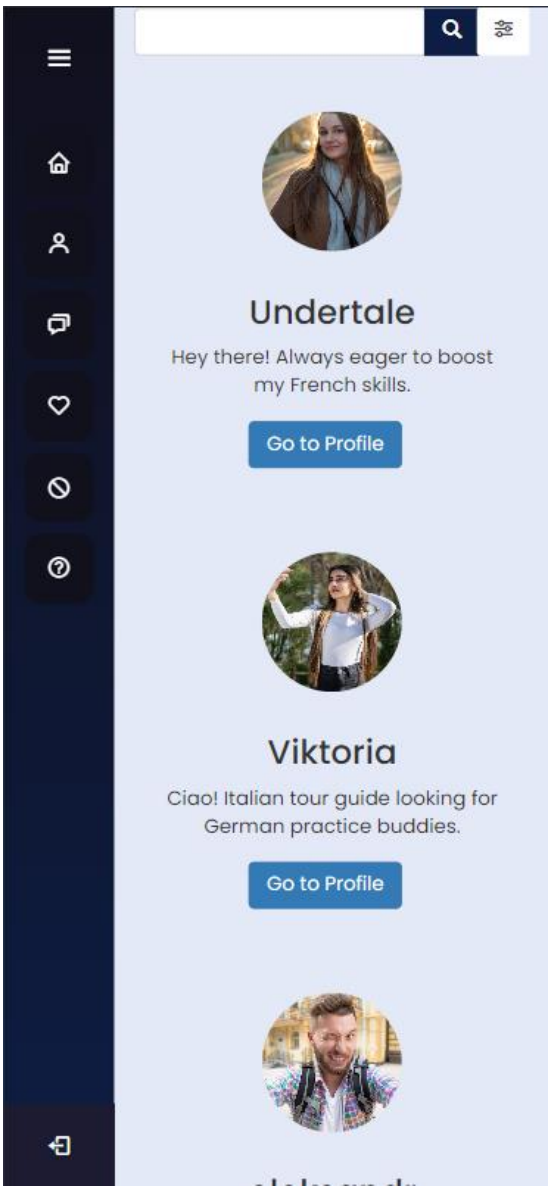

Рисунок 4.6 – Демонстрація головної сторінки на ерані телефону

Додатково, веб-додаток забезпечує можливість динамічного завантаження нових профілів користувачів під час прокручування головної сторінки вниз. Це дозволяє уникнути перевантаження сторінки великою кількістю контенту одразу та забезпечує плавний та безперервний досвід перегляду.

Таким чином, головна сторінка веб-додатку для обміну мовами поєднує в собі зручну навігацію, потужні можливості пошуку та фільтрації, компактне та інтуїтивне відображення профілів користувачів, адаптивний дизайн для різних пристроїв, а також динамічне завантаження контенту, забезпечуючи загальну привабливість та функціональність інтерфейсу.

4.4 Демонстрація чату, запрошень та функціоналу для управління ними

Додаток забезпечує зручний та функціональний інтерфейс для ведення чатів як в приватному, так і в груповому форматі. Сторінка чату розділена на дві основні секції: ліву панель, де відображаються активні чати та запрошення, і основну область для безпосереднього спілкування.

У лівій панелі користувач може бачити список своїх поточних чатів з іншими учасниками, а також запрошення до нових чатів, згруповані в окремий розділ "Запрошення". Запрошення генеруються автоматично під час створення групових чатів і відображаються тут для всіх бажаючих приєднатися. Над кожним запрошенням є дві кнопки: "Прийняти" та "Відхилити"; фото адміністратора та інформація про груповий чат, в який хочуть запросити.

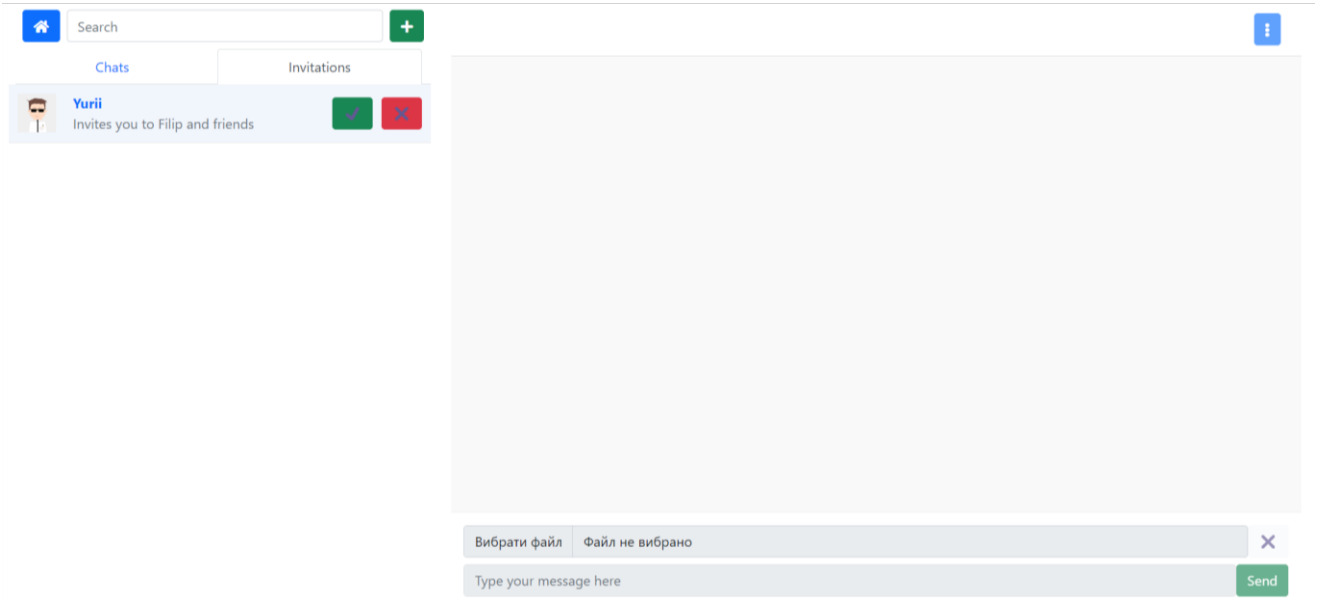

Рисунок 4.7 – Демонстрація окремих чатів та їх списків

Основна область чату складається з нових надісланих повідомлень. Проте можна подивитися також повідомлення, які були надіслані раніше прогортавши на початок розмови. Важливо зазначити, що історія повідомлень у чаті зберігається в базі даних веб-додатку, тому користувачі можуть переглядати всі повідомлення, які були надіслані з моменту створення даного чату, незалежно від того, коли вони приєдналися до нього.

Повідомлення різних користувачів чітко відрізняються за допомогою аватарок – власні повідомлення користувача відображаються зліва з його аватаркою справа, тоді як повідомлення від інших учасників розташовані справа з їхніми аватарками зліва. Це забезпечує зручність та легкість сприйняття під час читання чату.

Нижче знаходиться поле введення для написання нових повідомлень. Поряд з ним розміщена кнопка "Вибрати файл", яка дозволяє користувачеві відправляти файли або мультимедіа у чат. Після натискання на цю кнопку відкривається стандартний файловий провідник операційної системи, де можна вибрати потрібний файл для відправлення. Після вибору файлу його назва буде відображена поруч із кнопкою, і файл буде надісланий разом із повідомленням.

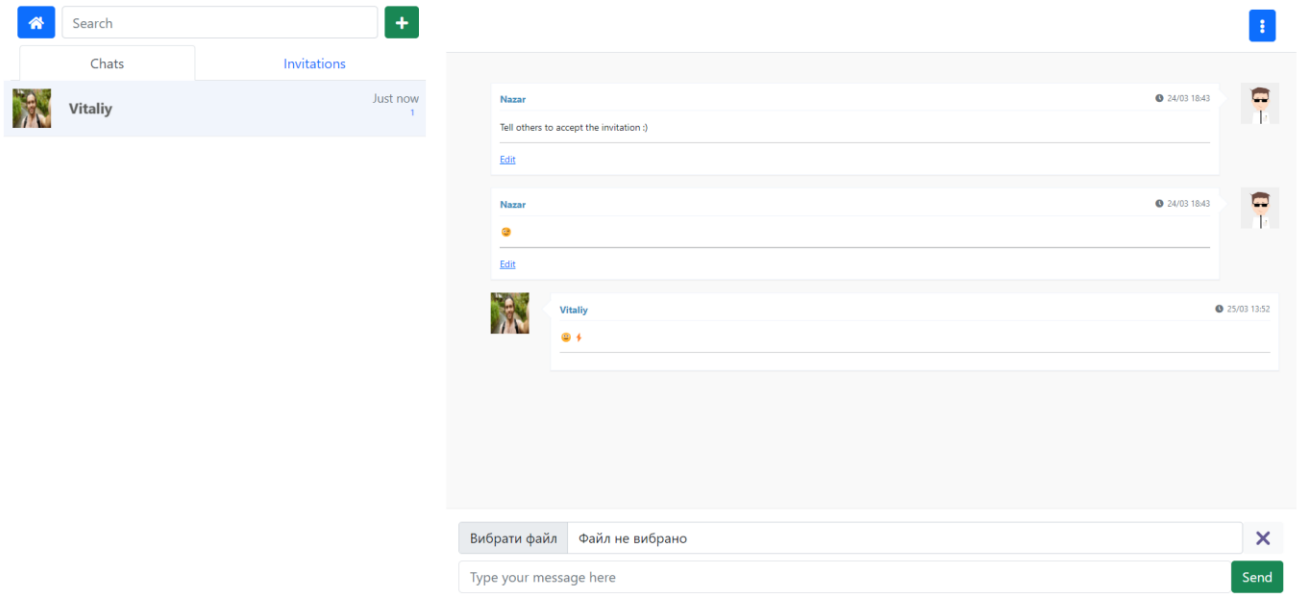

Рисунок 4.8 – Демонстрація окремих чатів та їх списків

Функціонал сайту також забезпечує створення нових групових чатів та надсилання відповідних запрошень іншим користувачам. Для цього передбачено окреме модальне вікно, яке відкривається поверх поточної сторінки.

У ньому присутнє поле для введення назви нового групового чату. Нижче розташоване поле для вводу імен користувачів, яких потрібно запросити. Після введення імені користувача необхідно натиснути кнопку "Add User", щоб додати його до списку запрошених. У правому нижньому куті вікна розміщені кнопки

"Cancel" для скасування дії та "Create New Chat" для підтвердження створення нового групового чату.

Після натискання кнопки "Create New Chat" відповідним користувачам буде надіслано запрошення приєднатися до новоствореного чату. Ці запрошення відображатимуться у спеціальному розділі "Запрошення", де вони зможуть їх переглянути, прийняти або відхилити.

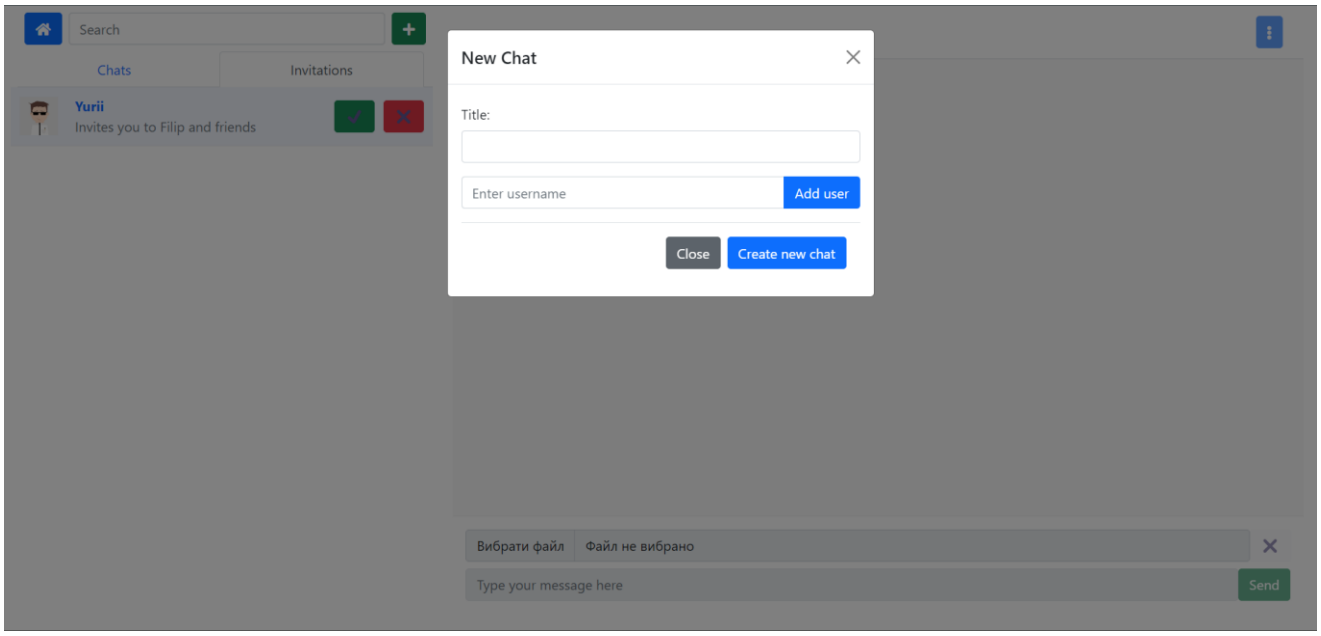

Рисунок 4.9 – Модальне вікно створення нового групового чату

Таким чином, функціонал створення групових чатів та надсилання запрошень забезпечує зручність організації спільного спілкування для обміну мовами між декількома учасниками веб-додатку.

4.5 Відображення сторінки заблокованого користувача

Веб-додаток передбачає механізм блокування користувачів для уникнення небажаних ситуацій та забезпечення позитивного досвіду спілкування. У разі блокування іншим користувачем, заблокована особа більше не має доступу до профілю та інформації того, хто її заблокував.

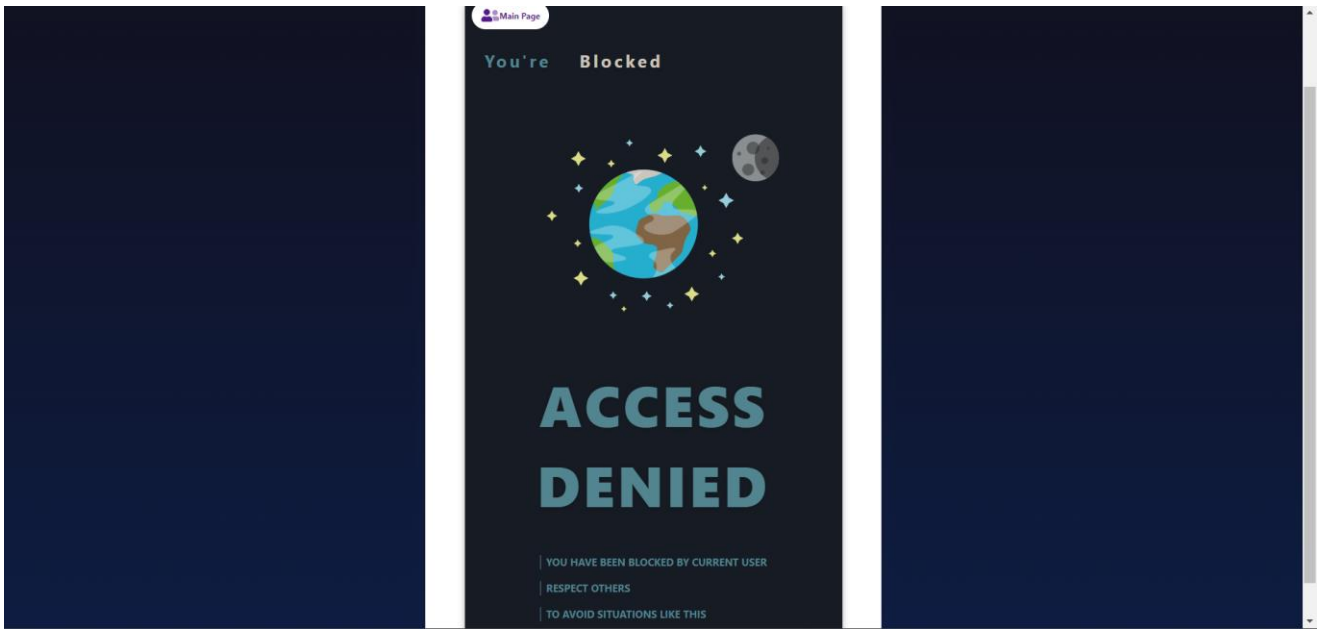

Рисунок 4.10 – Сторінка заблокованого користувача

Натомість користувачеві відображається спеціальна сторінка "Access Denied" з повідомленням про блокування. Верхня панель містить посилання для повернення на головну сторінку. Центральною ж частиною сторінки є яскравий напис "You're Blocked".

Нижче розміщено великими літерами текст "ACCESS DENIED", підкреслюючи заборону доступу. Відображається коротке пояснення причини блокування та попереджувальне повідомлення про необхідність поважати інших користувачів.

Мінімалістичний дизайн сторінки у темних тонах створює відчуття обмеженого простору, водночас зберігаючи зв'язок із загальною концепцією веб-додатку завдяки ілюстрації.

4.6 Вигляд сторінки зворотнього зв'язку

Для забезпечення зручної комунікації з користувачами та отримання відгуків, веб-додаток містить окрему сторінку контактної форми. Ця сторінка доступна через головне меню, що дозволяє легко знайти її в разі необхідності.

На сторінці міститься форма зворотного зв'язку під заголовком "Contact Us". Нижче розміщено три поля для заповнення.

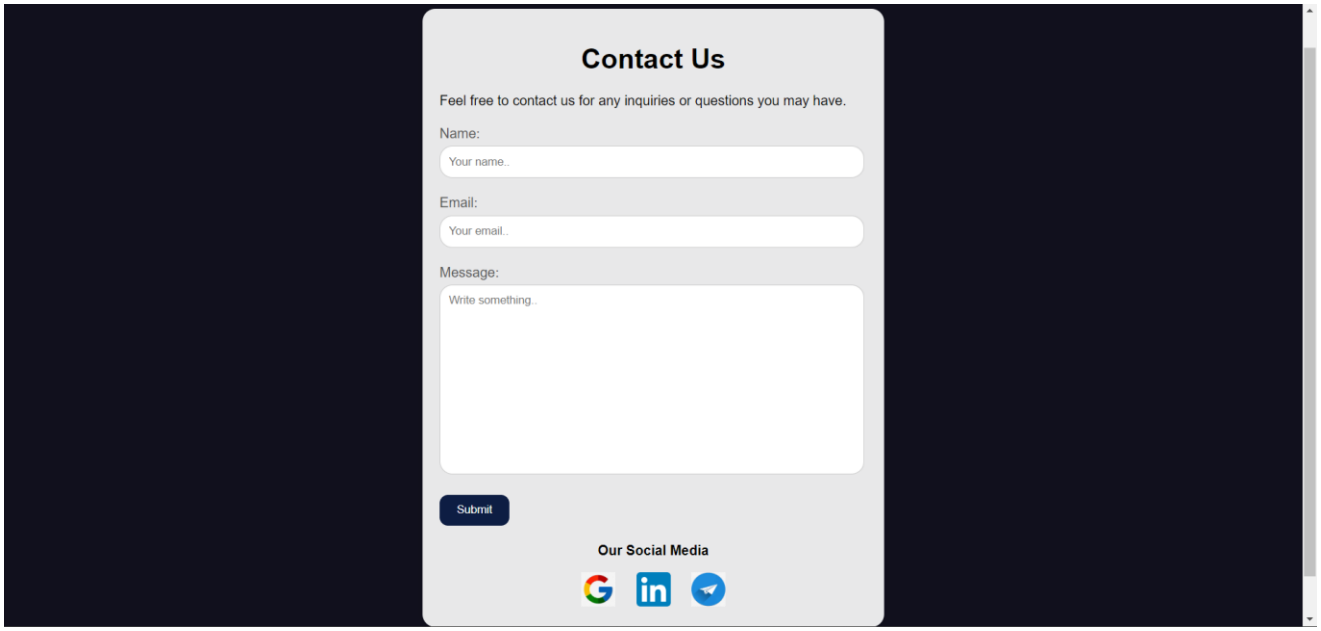

Рисунок 4.11 – Сторінка зворотнього зв’язку

Перше поле призначене для введення імені користувача. Система здійснює базову валідацію, перевіряючи, чи не залишилося воно порожнім. Наступне поле слугує для вказання електронної адреси, на яку можна надіслати відповідь. Також застосовується перевірка коректності введеного email.

Центральне і найбільше поле призначене для розміщення основного тексту повідомлення чи запитання. Тут користувачі можуть детально описати свої побажання та пропозиції щодо розширення функціоналу.

Після заповнення всіх необхідних полів користувач може натиснути кнопку "Submit" для відправлення форми. Повідомлення буде передане на спеціально створену електронну адресу, призначену для обробки запитів зворотного зв'язку.

Нижче розміщено іконки соціальних мереж - Google, LinkedIn та Telegram. Вони слугують посиланнями на альтернативні канали комунікації та інформування користувачів.

Загальний дизайн сторінки витриманий у мінімалістичному стилі, характерному для всього веб-додатку, із застосуванням світлих відтінків та акцентних кольорів для виділення важливих елементів.

ВИСНОВКИ

У теоретичній частині роботи було проведено ґрунтовний аналіз існуючих веб-додатків для обміну мовами між носіями. Він виявив певні недоліки та обмеження наявних рішень, такі як складність пошуку партнерів, проблеми з якістю звуку під час голосового спілкування, випадки недоброчесної поведінки користувачів. Крім того, був здійснений ретельний вибір технологій для реалізації веб-додатку, включаючи фреймворк Django, базу даних PostgreSQL, хмарне сховище AWS S3, Websocket, бібліотеки Channels, django-allauth та django-storages. Ці технології були обрані завдяки їх потужності, надійності, безпеці та відповідності вимогам проєкту.

У практичній частині було успішно розроблено веб-додаток для обміну мовами між носіями. Додаток забезпечує зручну платформу для пошуку партнерів, створення індивідуальних та групових чатів, спілкування в режимі реального часу, обміну файлами та мультимедіа, керування профілями, а також захист від небажаних користувачів за допомогою функціоналу блокування. Структура додатку була продумано спроєктована з використанням патерну Model-View-Template (MVT), що забезпечує модульність, масштабованість та простоту підтримки. Інтеграція з хмарним сховищем AWS S3 дозволила безпечно та ефективно зберігати завантажені користувачами файли. Загалом, розроблений веб-додаток відповідає поставленим вимогам та цілям проєкту.

Виконана робота має важливе значення для сприяння вивченню іноземних мов та міжкультурній комунікації в сучасному глобалізованому світі. Розроблений додаток надає зручну платформу для носіїв різних мов для практики спілкування та обміну мовними навичками. Його використання може сприяти налагодженню зв'язків між представниками різних національностей та культур і розширенню їх кругозору. Крім того, веб-додаток може бути застосованим для організації мовних клубів, пошуку мовних партнерів та інших сфер, пов'язаних з вивченням мов .

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Документація Django framework [Електронний ресурс] — 2024. Режим доступу: https://www.djangoproject.com.
2. Документація Django-rest-framework [Електронний ресурс] — 2024. Режим доступу: https://www.django-rest-framework.org/.
3. Документація postgresql. [Електронний ресурс] — 2024. Режим доступу: https://www.postgresql.org/docs/.
4. Вступ до мови JavaScript [Електронний ресурс] — 2024. — Режим доступу: https://javascript.info/intro.
5. Опис мови програмування Python [Електронний ресурс] — 2024. — Режим доступу: https://aws.amazon.com/what-is/python/.
6. Середовище розробки Visual Studio Code для фреймворку Django [Електронний ресурс] — 2024. — Режим доступу: https://code.visualstudio.com/docs/python/tutorial-django.
7. Документація системи управління залежностями Poetry [Електронний ресурс] — 2024. — Режим доступу: https://python-poetry.org/docs/.
8. Опис набору інструментів для форматування коду Ruff [Електронний ресурс] — 2024. — Режим доступу: https://docs.astral.sh/ruff/.
9. Робота з Github [Електронний ресурс] — 2024. — Режим доступу: https://docs.github.com/en.
10. Документація бібліотеки Django Channels [Електронний ресурс] — 2024. — Режим доступу: https://channels.readthedocs.io/en/latest/.
11. Документація бібліотеки Django Allauth [Електронний ресурс] — 2024. — Режим доступу: https://docs.allauth.org/en/latest/.
12. Посібник користування Amazon S3 [Електронний ресурс] — 2024. — Режим доступу: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html.
13. Архітектурний шаблон MVC Django [Електронний ресурс] — 2024. — Режим доступу: https://pbp-fasilkom-ui.github.io/ganjil-2023/en/assignments/tutorial/tutorial-1/.

14. Порівняльна характеристика сучасних веб фреймворків[Електронний ресурс] — 2024. — Режим доступу: https://www.educba.com/tutorials/.
15. Jon Duckett, HTML and CSS: Design and Build Websites. Режим доступу: – https://wtf.tw/ref/duckett.pdf.
16. О. Васильєв, Програмування мовою Python: Навчальна книга «Богдан», 2019. 206 с.

ДОДАТОК А

Код функціоналу аутентифікації

```
# singnup/views.py
from django.contrib.auth import authenticate, login, logout
from django.shortcuts import HttpResponseRedirect, render
from django.urls import reverse
from django.views import View
from main_app.models import UserProfile

from .forms import LoginForm, SignUpForm


class Register(View):
    template_name = (
        "../templates/main_app/register.html"
    )  #'signup/singup.html'

    def get(self, request):
        context = {"form": SignUpForm()}
        return render(request, self.template_name, context)

    def post(self, request):
        if request.method == "POST":
            form = SignUpForm(request.POST)
            if form.is_valid():
                form.save()
                username = form.cleaned_data.get("username")
                password = form.cleaned_data.get("password1")

                user = authenticate(username=username, password=password)
                login(request, user)
                UserProfile.objects.create(user=user)
                return HttpResponseRedirect(reverse("profile-page"))
        context = {"form": form}
        return render(request, self.template_name, context)


class Login(View):
    template_name = "../templates/main_app/register.html"  #'signup/login.html'

    def get(self, request):
        context = {"form": LoginForm()}
        return render(request, self.template_name, context)

    def post(self, request):
        if request.method == "POST":
            form = LoginForm(request.POST)
            if form.is_valid():
                username = form.cleaned_data.get("username")
                password = form.cleaned_data.get("password")
                user = authenticate(username=username, password=password)
                if user is not None:
```

```
                    login(request, user)
                    return HttpResponseRedirect(reverse("profile-page"))

        context = {"form": form}
        return render(request, self.template_name, context)


def logout_view(request):
    logout(request)
    return HttpResponseRedirect(reverse("register-page"))
```

ДОДАТОК Б

Код реалізації моделей

```
# user_profile.py
from django.contrib.auth.models import User
from django.db import models

from .user_language import Language


class UserProfile(models.Model):
    PROFICIENCY_LEVELS = [
        ("1", "None"),
        ("2", "Man"),
        ("3", "Woman"),
    ]

    user = models.OneToOneField(User, on_delete=models.CASCADE)
    languages = models.ManyToManyField(Language, blank=True)
    description = models.CharField(
        blank=True, default="No bio provided", max_length=355
    )
    gender = models.CharField(
        max_length=2,
        choices=PROFICIENCY_LEVELS,
        default="1",
    )
    residence = models.TextField(
        blank=True, default="No bio provided", max_length=355
    )
    birth_date = models.DateField(null=True)
    photo = models.ImageField(upload_to="chat/", null=True)

    def __str__(self) -> str:
        return self.user.username




# user_language.py
from django.db import models
```

```
class Language(models.Model):
    PROFICIENCY_LEVELS = [
        ("1", "Beginner"),
        ("2", "Intermediate"),
        ("3", "Advanced"),
        ("4", "Fluent"),
        ("5", "Native"),
    ]

    name = models.CharField(max_length=255)
    proficiency_level = models.CharField(
        max_length=2,
        choices=PROFICIENCY_LEVELS,
        default="1",
    )
```

 # chat.py

```
from django.contrib.auth.models import User
from django.db import models


class Chat(models.Model):
    name = models.TextField()
    users = models.ManyToManyField(User, related_name="user_rooms")
    slug = models.SlugField(unique=True)
    create_at = models.DateTimeField(auto_now_add=True)
    created_by = models.ForeignKey(User, on_delete=models.CASCADE, null=True)
```

 # message.py

```
from django.contrib.auth.models import User
from django.db import models

from main_app.models import Chat


class Message(models.Model):
    chat = models.ForeignKey(
        Chat, related_name="chat", on_delete=models.CASCADE
    )
    user = models.ForeignKey(
        User, related_name="messages", on_delete=models.CASCADE
    )
    content = models.TextField()
    media_file = models.FileField(upload_to="chat/", null=True, blank=True)
    date_added = models.DateTimeField(auto_now_add=True)
    update_at = models.DateTimeField(null=True)
```

```python
# chat_invitation.py
from django.contrib.auth.models import User
from django.db import models

from main_app.models import Chat


class ChatInvitation(models.Model):
    chat = models.ForeignKey(
        Chat, related_name="chat_invite", on_delete=models.CASCADE
    )
    user = models.ForeignKey(
        User, related_name="user_invite", on_delete=models.CASCADE
    )
    creator = models.ForeignKey(
        User, related_name="user_invite_creator", on_delete=models.CASCADE
    )
    date_added = models.DateTimeField(auto_now_add=True)
```

```python
# black_list.py
from django.contrib.auth.models import User
from django.db import models


class BlackList(models.Model):
    user = models.ForeignKey(
        User, on_delete=models.CASCADE, related_name="black_list_owner"
    )
    black_listed_user = models.ForeignKey(
        User, on_delete=models.CASCADE, related_name="black_listed_user"
    )

    class Meta:
        db_table = "black_list"
```