**Міністерство освіти і науки України**
**Національний технічний університет**
**«Дніпровська політехніка»**

Інститут електроенергетики
(інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

# ПОЯСНЮВАЛЬНА ЗАПИСКА

## кваліфікаційної роботи ступеня магістра

(бакалавра, спеціаліста, магістра)

**Студента** Пащенко Сергій Олександрович
(ПІБ)
**академічної групи** 126-22м
(шифр)
**спеціальності** 126 « Інформаційні системи та технології»
(код і назва спеціальності)
**за освітньо-професійною програмою**

«Інформаційні системи та технології»
(офіційна назва)
**на тему** Інформаційна технологія обробки природної мови на основі хмарних сервісів
(назва за наказом ректора)_

| **Керівники** | **Прізвище, ініціали** | **Оцінка за шкалою** | | **Підпис** |
|---|---|---|---|---|
| | | **рейтинговою** | **інституційною** | |
| кваліфікаційної роботи | доц. Соколова Н.О. | | | |
| розділів: | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

| | | | | |
|---|---|---|---|---|
| **Рецензент** | доц.Клименко А.В. | | | |

| | | | | |
|---|---|---|---|---|
| **Нормоконтролер** | проф. Коротенко Г.М. | | | |

**Дніпро**
**2023**

**ЗАТВЕРДЖЕНО:**
завідувач кафедри

інформаційних технологій
та комп'ютерної інженерії
(повна назва)

_____ Гнатушенко В.В.
(підпис) (прізвище, ініціали)

«_____»_____2023_року

# ЗАВДАННЯ

## на кваліфікаційну роботу

**ступеня** магістра

(бакалавра, магістра)

**студенту** Пащенко С. О. **академічної групи** 126-22м
(прізвище та ініціали) (шифр)

**спеціальності** 126 « Інформаційні системи та технології »

**за освітньою-професійною програмою**

«Інформаційні системи та технології»

**на тему** Інформаційна технологія обробки природної мови на основі хмарних сервісів

затверджену наказом ректора НТУ «Дніпровська політехніка» від 9.10.2023 р. № 1227-с

| Розділ | Зміст | Термін виконання |
|---|---|---|
| Розділ 1. | На основі матеріалів із зібраних джерел проаналізувати стан області рішення задачі | 9.10.2023 – 24.10.2023 |
| Розділ 2. | Визначити основні підходи до вибору інструментів для створення мобільного застосунку | 24.10.2023 – 11.11.2023 |
| Розділ 3. | Користуючись матеріалами науково-технічних джерел обрати необхідні методи та засоби й реалізувати проєктні рішення | 12.11.2023 – 12.12.2023 |

**Завдання видано** _____ Соколова Н. О.
(підпис керівника) (прізвище, ініціали)

**Дата видачі** 04.09.2023 р.

**Дата подання до екзаменаційної комісії** 21.12.2023 р.

**Прийнято до виконання** _____ Пащенко С.О.
(підпис студента) (прізвище, ініціали)

# РЕФЕРАТ

Кваліфікаційна робота: 74 с., рис. 29, джерел 16, додатків 3.

Метою даної роботи є створення хмарного сервісу з використанням моделей обробки природної мови, а також дослідження безкоштовних провайдерів, для подальшого використання послуг.

Об’єктом дослідження є хмарні сервіси з безкоштовним хостингом і інтелектуальні системи для розпізнавання голосу, а також бізнес-модель, при якій програмний продукт надається в форматі веб-сервісу на основі передплати.

Предмет дослідження є використання хмарних сервісів обробки природної мови в проєктах інтелектуальних систем.

Практичне значення отриманих результатів полягає в використанні та вдосконаленні створення хмарних сервісів з використанням обробки природної мови. Використання стандартного функціонала фреймворку Django. Детальна інструкція по розгортанню проєкта на хостингу, та налаштування SaaS для подальшого використання та вдосконалення.

Перший розділ дипломного роботи присвячено розбору SaaS-технології, пошуку безкоштовних хмарних сервісів для розгортання проєкту на сервері.

У другому розділі описано теоретичні питання роботи над хмарними сервісами, які допоміжні пакети та технології потрібні для зручного завантаження проєкту на сервер та подальша підтримка проєкта.

Третій розділ містить опис програмного забезпечення сервісу з використанням natural language processing, його тестування та аналіз отриманих результатів.

Ключові слова:, ХМАРНІ СЕРВІСИ, NATURAL LANGUAGE PROCESSING, РОЗПІЗНАВАННЯ ГОЛОСУ, SAAS-МОДЕЛЬ, DLANGO.

# ABSTRACT

Diploma project: 74 pages, fig. 29, sources 16, appendices 3.

The aim of this project is to create a cloud service using natural language processing models, as well as to research free providers for the further use of services.

The objects of research are cloud services with free hosting and intelligent voice recognition systems, as well as a business model in which the software product is provided in the format of a web service based on a subscription.

The subject of the research comprises cloud services for natural language processing in projects of intelligent systems.

The practical significance of the obtained results lies in the use and improvement of the creation of cloud services using natural language processing. This includes the use of the standard functionality of the Django framework. Detailed instructions for deploying the project on hosting and configuring SaaS for further use and improvement are provided.

The first section of the diploma project is devoted to the analysis of SaaS technology and the search for free cloud services for deploying the project on a server.

**ЗМІСТ**

# ВСТУП

У сучасному світі ми стаємо свідками безпрецедентного зростання обсягів інформації. Щодня людство генерує величезну кількість даних, що постійно збільшується з кожною секундою. Ці дані - від цифрових зображень і відео до відгуків споживачів, соціальних мереж, аналітичних досліджень, фінансових транзакцій - формують складний, але потенційно цінний світ інформації. Однак їх справжній потенціал можна розкрити лише тоді, коли вдається правильно їх обробити та проаналізувати.

У минулому, коли технологічні можливості були обмежені, обробка великих обсягів інформації вимагала надзвичайних зусиль і часу. Що могло зайняти дні або навіть місяці ручної праці, тепер стало недосяжним через неймовірний обсяг сучасних даних. Така ситуація спонукала до розвитку нових алгоритмів та інструментів, які дозволяють комп'ютерним системам ефективно обробляти, аналізувати та перетворювати ці масиви даних у цінну інформацію. Ці технології стали ключовими в розумінні та використанні великих даних, відкриваючи нові горизонти для інновацій та розвитку в найрізноманітніших галузях.

Важливим напрямом у цій сфері є розробка програмного забезпечення за моделлю SaaS (Software as a Service), яка дозволяє користувачам отримувати доступ до програм як до онлайн-сервісів.

Таким чином, користувач платить не за повне володіння програмою, а за її використання якийсь певний момент часу.

Мета даного дослідження є вивчення найновіших підходів обробки природної мови, що базуються на хмарних сервісах, а також виявлення можливостей їх ефективного використання у різноманітних областях, включаючи системи з елементами штучного інтелекту.

Для досягнення зазначеної мети було визначені ряд конкретних завдань:

- Вивчити основи технології обробки природної мови та її історичний розвиток.

- Проаналізувати сучасні хмарні платформи, які надають можливості для NLP.
- Розглянути практичні приклади застосування хмарних сервісів у проектах NLP.
- Визначити переваги та обмеження використання хмарних сервісів у сфері обробки природної мови.
- Розробка та демонстрація програмного додатка, який використовує хмарні сервіси для обробки мовних даних, щоб продемонструвати практичне застосування досліджених технологій.

Методи дослідження включають аналіз наукової літератури, вивчення сучасних хмарних платформ, порівняльний аналіз існуючих рішень, та практичну розробку власного додатка, що використовує хмарні сервіси.

Структура роботи включає аналіз теоретичних основ NLP та хмарних технологій, дослідження сучасних хмарних платформ для NLP, а також детальний опис та аналіз розробленого програмного додатка на фреймворку Django.

Як основний мовний засіб було використано високорівнева мова програмування python, а в якості середовища розробки SaaS-моделі було обрано фреймворк Django.

# 1 ОГЛЯД СТАНУ ПИТАННЯ

## 1.1 Сучасні хмарні технології SaaS, PaaS та IaaS

Хмарні обчислення відкривають нові перспективи в інформатиці, швидко змінюючи спосіб, яким ми взаємодіємо з технологіями. Основна ідея хмарних обчислень полягає у перенесенні організації обчислень і обробки даних із персональних комп'ютерів на віддалені сервери, доступні через Інтернет. Це дозволяє користувачам доступ до своїх даних без необхідності управління інфраструктурою чи програмним забезпеченням, з яким вони працюють.

З огляду на їхню ефективність і гнучкість, хмарні технології стрімко набирають популярності, особливо у бізнес-середовищі. Вони дозволяють компаніям масштабувати свої IT-функції і обчислювальну потужність, уникаючи витрат на придбання додаткового обладнання чи програмного забезпечення, а також найм додаткового персоналу. Технічне обслуговування інфраструктури, розміщеної у хмарі, входить до зони відповідальності провайдера.

Сучасні хмарні технології можна класифікувати за трьома основними напрямками:

Інфраструктура як послуга (IaaS): надає базову інфраструктуру, таку як сервери і мережеве обладнання.

Платформа як послуга (PaaS): забезпечує розробникам необхідні інструменти і середовище для створення та випуску програмного забезпечення.

Програмне забезпечення як послуга (SaaS): дозволяє користувачам отримувати доступ до програмного забезпечення через Інтернет без необхідності його встановлення.

Модель 'as a Service' свідчить про можливість використання цих ресурсів за принципом підписки, тобто користування за потребою. Сервісна модель архітектури хмарних обчислень, представлена на рисунку 1.1, показує, що на базі IaaS будується PaaS, а поверх PaaS розташовується SaaS. Така ієрархія забезпечує гнучкість і масштабованість хмарних рішень.

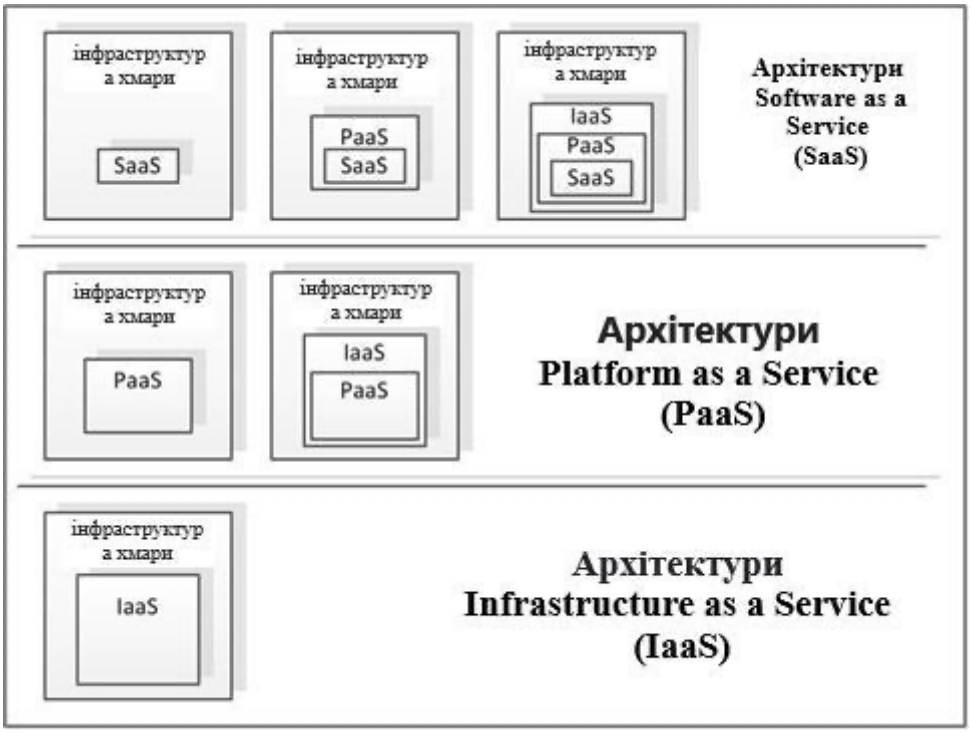



Рисунок 1.1 – Сервісна модель архітектури хмарних обчислень

Інфраструктура як послуга (IaaS) представляє собою модель надання обчислювальних ресурсів через хмару, яка перевтілює традиційний підхід до IT-інфраструктури. Ця модель пропонує оренду серверів та іншого обладнання, забезпечуючи високу гнучкість та масштабованість. Основна концепція IaaS полягає у переході від внутрішнього обслуговування інфраструктури до її використання від сторонніх провайдерів, що включає сервери, мережеве обладнання та інші необхідні ресурси.

Уявімо аналогію з побутовими послугами. У минулому люди самостійно забезпечували себе водою з колодязя та теплом, використовуючи дрова. З розвитком суспільства ці функції були передані спеціалізованим організаціям. Аналогічно в IT-галузі: раніше компанії самі відповідали за весь спектр IT-послуг, від обслуговування серверів до прокладки мереж. Нині, з IaaS, вони можуть перенести ці завдання на зовнішніх постачальників.

IaaS надає компаніям можливість швидко адаптуватися до змінних бізнес-вимог. Замість інвестицій у фізичне обладнання, компанії можуть використовувати хмарні ресурси, налаштовуючи їх під свої поточні потреби. Це включає в себе доступ до хмарних процесорів, пам'яті, дискового простору та мереж, що

забезпечує високу ступінь контролю над розгортанням і масштабуванням IT-інфраструктури.

Завдяки IaaS, компанії можуть також використовувати передові технології, такі як штучний інтелект, машинне навчання та великі дані, без необхідності інвестувати у власну інфраструктуру для цих цілей. Це дозволяє їм залишатися конкурентоспроможними та інноваційними в швидко змінному технологічному світі.

Платформа як послуга (PaaS) революціонізує підхід до розробки додатків, надаючи розробникам готове середовище через інтернет. Це середовище включає в себе необхідні інструменти та платформи, які спрощують процес створення як простих мобільних, так і складних корпоративних додатків. За допомогою послуг PaaS розробники можуть створювати все, від простих мобільних додатків до складного програмного забезпечення для бізнесу. PaaS забезпечує високу швидкість розробки, тестування та розгортання додатків, значно знижуючи час і витрати, оскільки всі витрати на інфраструктуру та платформу бере на себе провайдер.

PaaS відрізняється гнучкістю у керуванні ресурсами: розробники можуть масштабувати ресурси відповідно до потреб проєкту. Ця модель також сприяє співпраці, оскільки декілька користувачів можуть працювати над одним проектом через єдину платформу, що інтегрується з різними веб-службами та базами даних.

Платформа як послуга хороша тим, що відразу ж готова до роботи.

З іншого боку, програмне забезпечення як послуга (SaaS) надає клієнтам доступ до готових програм через інтернет. Це звільняє компанії від необхідності встановлювати, налаштовувати та обслуговувати програмне забезпечення на власних серверах. SaaS є ідеальним рішенням для компаній, які шукають економічно вигідні, швидкі та прості у використанні рішення для короткострокових проєктів чи повсякденних бізнес-завдань.

SaaS-програми, як правило, доступні через веб-браузер, що означає, що користувачам не потрібно встановлювати спеціалізоване програмне забезпечення. Це також робить SaaS ідеальним для мобільного доступу, забезпечуючи

користувачам можливість доступу до додатків з будь-якого місця та будь-якого пристрою.

Крім зазначених моделей, таких як Інфраструктура як послуга (IaaS), Платформа як послуга (PaaS) та Програмне забезпечення як послуга (SaaS), існують інші, менш відомі, але не менш важливі категорії хмарних послуг. Це включає Комунікації як послуга (CaaS) та Платформу даних як послуга (DaaS), кожна з яких задовольняє унікальні потреби користувачів.

Комунікації як послуга (CaaS) - це підхід, який забезпечує користувачам можливість використання різноманітних комунікаційних технологій, таких як голосові зв'язки, відеоконференції та чати, через хмарну інфраструктуру. Це дозволяє організаціям уникати необхідності вкладення у власні комунікаційні системи, замість цього вони можуть використовувати високоякісні та гнучкі комунікаційні послуги на основі передплати.

Платформа даних як послуга (DaaS) пропонує користувачам використання комплексних баз даних без необхідності їх розгортання чи обслуговування. Користувачі можуть ефективно управляти та аналізувати великі обсяги даних, використовуючи інструменти та можливості, що надаються хмарною платформою. Це забезпечує високу доступність, безпеку та масштабованість обробки даних, що є критично важливим для бізнесів, що залежать від обробки великих масивів інформації.

Таким чином, різноманітність хмарних послуг дозволяє задовольнити специфічні потреби різних сегментів ринку, від індивідуальних користувачів до великих корпорацій, забезпечуючи їм гнучкість, ефективність та масштабованість у їх цифрових потребах.

Цікаво, що концепція надання обчислювальних ресурсів як послуги не є новою. Споконвіку комп'ютерна індустрія використовувала модель оренди, оскільки перші комп'ютери були надзвичайно дорогими і доступними лише для великих організацій або урядових установ. Ці первісні комп'ютери надавали обчислювальну потужність клієнтам на основі орендних угод, але це значно відрізнялося від сучасних хмарних сервісів. Тоді замовники мали безпосередній

фізичний доступ до комп'ютерного обладнання, відрізняючись від сьогоднішнього використання глобальних мереж зв'язку.

В сучасній ері хмарних обчислень, доступ до суперкомп'ютерних центрів через Інтернет набуває нового значення. Ці послуги можна умовно порівняти з моделями Інфраструктури як послуга (IaaS) та Платформи як послуга (PaaS). Однак, навіть з урахуванням цих сучасних технологій, існують певні відмінності. У випадку IaaS та PaaS, клієнтам надається не тільки доступ до обчислювальної інфраструктури та платформ, але й до повного спектру підтримки та сервісів. Так, користувачі мають можливість використовувати розширені можливості хмарних центрів обробки даних, однак часто беруть на себе відповідальність за налаштування обчислювальних процесів та маршрутизацію даних.

Сучасні хмарні сервіси значно відрізняються від ранніх моделей оренди обладнання, пропонуючи більшу гнучкість, масштабованість та доступність. Вони надають користувачам можливість швидко та ефективно розгортати застосунки, зберігати великі обсяги даних та обробляти складні обчислення без необхідності вкладення в дороге фізичне обладнання. Таким чином, розвиток хмарних технологій є прикладом того, як інновації та технологічний прогрес відкривають нові можливості та моделі ведення бізнесу в інформаційну епоху.

### 1.2 Приклади комп'ютерних систем на основі хмарних технологій

У світі сучасних інформаційних технологій, хмарні сервіси відіграють ключову роль, пропонуючи широкий спектр рішень для бізнесу та особистого користування. Нижче наведено декілька прикладів комп'ютерних систем, які використовують хмарні технології:

Послуги електронної пошти, такі як Gmail, Outlook і Yahoo! Mail, є класичними прикладами хмарних сервісів. Користувачі входять у свої облікові записи через Інтернет, часто за допомогою веб-браузера. Саме програмне забезпечення, а також збережені електронні листи розміщуються на серверах провайдера, надаючи доступ з будь-якого місця, де є Інтернет.

Попередні приклади — це безкоштовні служби для особистого користування. Оплата використання таких додатків здійснюється за передплатою або відповідно до рівня використання.

Платформи для управління взаєминами з клієнтами (CRM), такі як Salesforce, Zoho CRM, і HubSpot. Вони надають компаніям інструменти для управління контактами, продажами, обслуговуванням клієнтів та маркетинговими кампаніями. Клієнти оплачують ці послуги, як правило, на основі щомісячної підписки.

Платформи як Google Drive, Dropbox, і Microsoft OneDrive дозволяють користувачам зберігати файли у хмарі та ділитися ними з іншими. Ці сервіси включають можливість створення документів, таблиць, презентацій, а також забезпечують інструменти для спільної роботи в реальному часі.

Платіжні сервіси, такі як PayPal, Приват24, Google Pay, Fondy, пропонують рішення для обробки онлайн-платежів за моделлю SaaS. Вони забезпечують безпечний прийом платежів для бізнесу, а також допомагають у зборі та аналізі фінансових даних.

Компанії, які пропонують хостинг веб-сайтів і баз даних, такі як Amazon Web Services, Bluehost, і GoDaddy, пропонують хмарне зберігання для веб-ресурсів і баз даних. Ці сервіси забезпечують високу доступність, швидкість і безпеку сайтів своїх клієнтів.

Кожен з цих прикладів демонструє, як хмарні технології можуть бути використані для забезпечення гнучких, ефективних і масштабованих рішень в різних галузях. Від індивідуального користувача до великих корпорацій, хмарні сервіси стають невід'ємною частиною цифрової екосистеми, впливаючи на способи ведення бізнесу та взаємодії з технологіями.

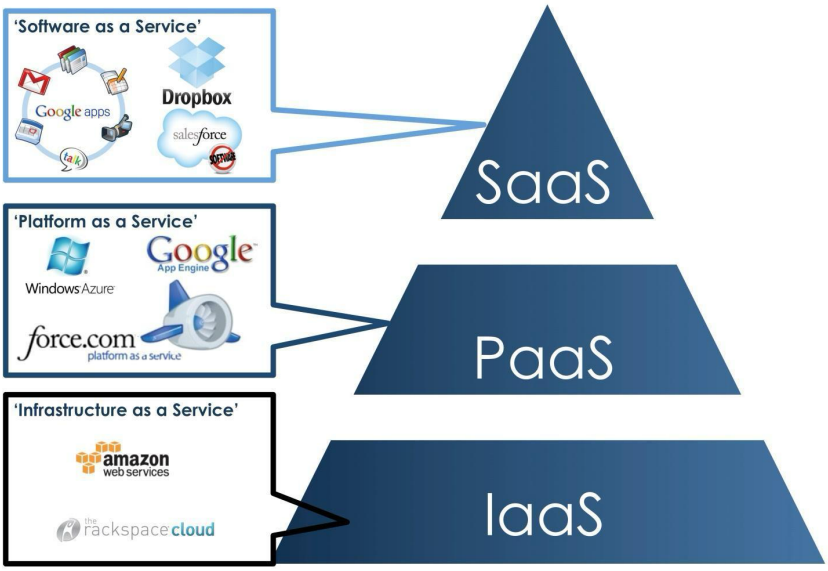



Рисунок 1.2 – Приклади комп'ютерних систем на основі хмарних технологій

Месенджери та служби електронної пошти, такі як Gmail, Slack, Microsoft Teams, і Telegram, є відмінними прикладами додатків, що базуються на хмарних сервісах. Ці платформи використовують хмарні технології для забезпечення надійного, доступного з будь-якого місця, та ефективного способу обміну повідомленнями та інформацією.

## 1.3 Принципи та підходи до розробки програмного забезпечення з використанням хмарних сервісів

Щоб хмарна служба відповідала цілям інтеграції і досягла цільових показників економічно життєздатним чином, хмарні сервіси повинні будуватися на основі, яка проявляє певні принципи на фундаментальному рівні. Ці принципи включають відкритість, доступність, економічну здійсненність, масштабованість і підтримку.

Відкритий сервіс — це той, який може використовуватися споживачами послуг з мінімальним втручанням з боку постачальника послуг протягом усього життєвого циклу отримання послуг.

Доступний сервіс — це той, який доступний у всьому світі і обслуговує клієнтів, що охоплюють великі підприємства і малий бізнес. Вибір постачальника хмарної платформи і архітектура сервісу будуть відігравати важливу роль в забезпеченні доступності служби SaaS.

Економічна здійсненність: хмарна служба повинна бути економічною для роботи в різних масштабах використання клієнтів. Плата за підписку на послугу повинна бути більше суми вартості використання хмари, як прямий, так і непрямої. Постачальники хмарної платформи повинні допомогти розробникам SaaS з архітектурою орієнтованої на вартість (CSA), яка включає економічне використання ресурсів в якості основного принципу.

Масштабованість: хмарний сервіс повинен підтримувати компоненти платформи для декількох користувачів, необхідні для забезпечення узгодженої продуктивності в різних умовах використання.

Підтримка: підтримка поступається в порівнянні з функціональністю в звичайних додатках, тому що існують спеціальні служби ISV-сервісів, групи підтримки клієнтів, а також команди підтримки клієнтів і ISV для підтримки і усунення проблем додатків. Вони часто мають повний доступ до всіх механічних елементів інфраструктури і станом додатки під час його роботи. Архітектура хмарних сервісів повинна здійснювати підтримку як один з основних принципів, які визначають розробку рішення та його реалізацію.

Створення SaaS-сервісів є багатоетапним процесом, який починається з ретельного аналізу ринку і потреб цільової аудиторії. Це дозволяє розробникам ідентифікувати унікальні потреби та вимоги, які повинен задовольняти майбутній продукт. Важливим етапом є розробка прототипу, який відтворює базову функціональність та логіку роботи сервісу, а також дає уявлення про майбутній інтерфейс. Ключові ознаки SaaS включають віддалений доступ через веб-інтерфейс, розгортання в єдиному дата-центрі, оплата за підписку.

Архітектура проєкту та його функціональна реалізація потребують комплексного підходу до розробки програмного забезпечення. В цьому процесі фахівці втілюють задумані можливості та забезпечують надійну і ефективну роботу програмного продукту. Паралельно з програмною частиною розробляється унікальний дизайн інтерфейсу, який повинен бути інтуїтивно зрозумілим та привабливим для користувача.

Тестування продукту є критично важливим для забезпечення його надійності, безпеки та відповідності встановленим стандартам. Воно включає перевірку різних інструментів, інтеграцію з іншими сервісами та перевірку якості захисту.

Після завершення розробки і тестування, наступним кроком є підтримка і навчання співробітників замовника, щоб вони могли ефективно користуватися новим сервісом.

Ключові характеристики SaaS-додатків включають віддалений доступ через веб-інтерфейс, централізоване розгортання в єдиному дата-центрі та модель оплати на основі підписки, що включає регулярні платежі. Оновлення та підтримка забезпечуються провайдером SaaS, звільняючи замовника від необхідності встановлення оновлень чи патчів.

Сучасні вимоги до SaaS-додатків включають декларативний формат опису для налаштування та встановлення, що мінімізує витрати часу та ресурсів. Додаток повинен мати мінімальну залежність від операційної системи, бути легко інтегрованим з хмарними платформами та підтримувати неперервне розгортання для гнучкості та ефективності

Додатково, важливо зазначити:

- Хмарні платформи підтримують автоматизовані процеси CI/CD, що дозволяє розробникам швидко інтегрувати та розгортати оновлення.
- Хмарні платформи підходять для розробки з використанням мікросервісів і контейнерів, які забезпечують високий рівень ізоляції, гнучкості та масштабованості.
- Використання хмарних сервісів забезпечує централізацію управління, що спрощує адміністрацію та розподіл ресурсів.

Для ефективної роботи хмарних додатків потрібно враховувати залежності та інструменти системи. Хмарні додатки повинні чітко декларувати всі свої залежності за допомогою маніфестів залежностей, що забезпечує ясність та спрощує процес розгортання. Інструменти, такі як Pip для Python або Gem Bundler для Ruby, використовуються для оголошення та ізоляції залежностей, що гарантує консистентність середовища від розробки до виробництва.

Розробка SaaS-додатків вимагає ефективного управління кодом. Застосування системи управління версіями, наприклад Git, відіграє ключову роль для колективної розробки та управління кодом. Єдина кодова база дозволяє кільком розробникам ефективно працювати над одним проектом, де кожен може розгортати свою копію коду локально. Це сприяє гнучкості та співпраці в команді.

Хмарні додатки часто розробляються з використанням методології неперервного розгортання, що вимагає автоматизації багатьох процесів. Автоматичне тестування, інтеграція змін коду, та розгортання оновлень забезпечують високу швидкість розробки та гарантують стабільність додатку.

Особлива увага приділяється безпеці в хмарних додатках. Розробники повинні враховувати аспекти шифрування даних, управління доступом, та регулярні оновлення безпеки. Безпека в хмарі – це не тільки захист даних, але й забезпечення надійності та доступності сервісу.

## 1.4 Бібліотеки функцій з реалізації інформаційної технології Natural Language Processing

Однією з найпотужніших інструментів у сфері обробки природної мови (Natural Language Processing, NLP) є Natural Language Toolkit (NLTK), бібліотека для мови програмування Python. NLTK вирізняється своєю універсальністю та зручністю використання, пропонуючи користувачам доступ до широкого спектру корпусів та лексичних ресурсів, таких як WordNet. Ця бібліотека об'єднує інструменти для різноманітних завдань обробки тексту: від класифікації та токенізації до стемінгу, тегування, синтаксичного аналізу, а також забезпечує інструментарій для семантичного аналізу.

NLTK становить особливу цінність для широкого кола спеціалістів - від лінгвістів та інженерів до студентів та науковців, оскільки вона сприяє вивченню основ програмування та комп'ютерної лінгвістики. Завдяки своїй вичерпній документації та активному форуму для обговорення, NLTK легко адаптується до різних дослідницьких та розвиткових потреб. Варто підкреслити, що NLTK є відкритим проектом, що забезпечує його безкоштовне використання та можливість співпраці у спільноті розробників.

NLTK ефективно використовується для створення широкого спектру NLP-систем, включаючи, але не обмежуючись, системами розпізнавання мови, автоматизованим узагальненням текстів, машинним перекладом, виявленням спаму, визначенням іменованих сутностей та системами автоматичного відповіді на запитання. NLTK допомагає у реалізації складних задач обробки мови, забезпечуючи потужний інструментарій для дослідників і розробників.

Застосування NLP вже є неодмінною частиною нашого повсякденного життя. Від смартфонів з функцією розпізнавання голосу до комп'ютерів із вбудованими мовними асистентами, NLP є фундаментом для створення інтерактивних та інтуїтивно зрозумілих технологій. Чатботи, наприклад, використовуються для оптимізації роботи колл-центрів і служб обслуговування клієнтів, допомагаючи в обробці великої кількості запитань і значно знижуючи навантаження на операторів. Випадок із впровадженням чатбота Зоряною компанією "Київстар" є яскравим

прикладом ефективності використання чатботів у великих компаніях, демонструючи їх здатність вирішувати до 70% стандартних запитань клієнтів.

Завдяки своїй універсальності, NLTK і загалом технології NLP відкривають нові можливості для розвитку різних сфер - від бізнесу та освіти до розважальних застосунків, забезпечуючи необхідні інструменти для аналізу та обробки природної мови.

Процес розуміння мови машинами за допомогою алгоритмів обробки природної мови можна описати наступними кроками:

- Взаємодія з Людиною: Спочатку людина взаємодіє з машиною, висловлюючи свої думки або команди.
- Запис Звуку: Машина фіксує ці звукові сигнали, перетворюючи голосові дані у цифровий формат.
- Перетворення в Текст: Наступний етап включає конвертацію аудіо в текст, переводячи голосові команди в письмову форму.
- Аналіз Тексту: Після цього NLP система аналізує текст, розбираючи його на складові, визначаючи контекст розмови та намір людини. Це включає в себе розпізнавання ключових слів, фраз, інтенцій та інших лінгвістичних особливостей.
- Визначення Дій для Виконання: На основі отриманих даних машина вирішує, які дії необхідно виконати, відповідаючи на запит або виконуючи певні команди.

Проте, людська мова надзвичайно складна і багатогранна. Вона переповнена багатозначними словами, омонімами, іноді містить приховані підтексти, які машина може інтерпретувати буквально. Природна мова також часто включає еліпсиси, коли певні слова опускаються для уникнення повторення, як у прикладі: «Мені подобається Siri, а йому Alexa». Неперервна еволюція мови та поява нових слів і виразів створюють додаткові виклики для NLP систем.

Основною задачею NLP є перетворення мовлення або тексту у формат, зрозумілий для машини. Це включає широкий спектр завдань, таких як машинний переклад, перевірка правопису, розпізнавання мови, голосове керування,

сумаризація текстів, аналіз настроїв, класифікація текстів, контекстуальна онлайн реклама та прогнозування.

NLP вже активно використовується в багатьох сферах, включаючи медицину, правоохоронні органи та бізнес для структурування та аналізу великих обсягів даних. В медицині, наприклад, NLP використовується для аналізу медичних записів, допомоги у постановці діагнозів і відстеження перебігу хвороб.

Таким чином, NLP відкриває шлях до нових горизонтів в області машинного розуміння, спрощуючи взаємодію людей і машин та розширюючи можливості використання машин для вирішення складних завдань.

Існують різноманітні бібліотеки для роботи з NLP, кожна з яких пропонує свої унікальні можливості та інструменти. Однією з таких бібліотек є Stanford CoreNLP, розроблена на базі Університету Стенфорда. Ця бібліотека є відкритим програмним продуктом і націлена на комплексну обробку неструктурованого тексту. Основні можливості Stanford CoreNLP включають аналіз частин мови, структури тексту, розпізнавання іменованих сутностей, аналіз тональності тексту та багато іншого[6].

Natural Language Toolkit (NLTK) - це ще одна відома бібліотека для мови програмування Python, яка спеціалізується на різноманітних аспектах NLP. NLTK надає багатий набір інструментів та ресурсів для обробки тексту, включаючи можливість інтеграції з іншими бібліотеками, такими як Stanford CoreNLP для аналізу тональності та розмітки пропозицій, а також Scikit learn для різних видів класифікацій[7].

Scikit learn, хоча і не специфічно орієнтована на NLP, пропонує широкий спектр інструментів для машинного навчання, включаючи різноманітні класифікатори та моделі нейронних мереж. Ця бібліотека може бути ефективно використана у комбінації з іншими інструментами для створення складних і високоякісних систем обробки природного мовлення[8].

Tensorflow, первісно розроблена для машинного навчання та штучних нейронних мереж, також знайшла своє застосування у сфері обробки природної мови. Бібліотека включає такі інструменти як Seq2seq та Word2Vec, які вже

успішно використовуються в наукових дослідженнях та розробці програмних продуктів для аналізу та генерації природного мовлення[9].

Таким чином, кожна з цих бібліотек пропонує свої унікальні можливості для розвитку сфери обробки природної мови, дозволяючи розробникам вибрати найбільш відповідний інструментарій для своїх конкретних завдань та проектів.

### 1.5 Програмні середовища для створення проєктів інтелектуальних систем

У сфері створення інтелектуальних систем, які охоплюють машинне навчання, штучний інтелект (AI) та обробку природної мови (NLP), застосовуються спеціалізовані програмні середовища. Ці середовища забезпечують розробників потужними інструментами та фреймворками, необхідними для розробки, тестування та імплементації інтелектуальних систем. Розглянемо більш детально деякі з найвідоміших програмних середовищ:

PyTorch: Розроблений Facebook, PyTorch знаменитий своєю гнучкістю та динамічним виконанням графів, що робить його особливо привабливим для досліджень у сфері AI. PyTorch сприяє швидкому прототипуванню та ітеративному розвитку, що є ключовим для інновацій у галузі глибокого навчання.

Scikit-Learn: Ця бібліотека Python є однією з найбільш широко використовуваних у галузі машинного навчання. Вона пропонує широкий спектр алгоритмів, включаючи класифікацію, регресію, кластеризацію та зниження розмірності. Її простота і зручність використання роблять її ідеальною для багатьох задач машинного навчання.

Keras: Як надбудова над TensorFlow, Keras пропонує ще більшу спрощеність для створення глибоких нейронних мереж. Його високорівневий API робить його доступним для початківців, одночасно зберігаючи достатню гнучкість для досвідчених розробників.

Jupyter Notebook: Це інтерактивне середовище програмування є незамінним інструментом для дата-сайентистів. Воно підтримує візуалізацію даних та

інтерактивне програмування, дозволяючи користувачам наочно відображати результати їхньої роботи.

Microsoft Azure Machine Learning Studio: Це інтегроване хмарне рішення для розробки, тестування та розгортання моделей машинного навчання. Його інтуїтивно зрозумілий графічний інтерфейс і можливість працювати без безпосереднього написання коду роблять його привабливим для широкого кола розробників.

R Studio: Це інтегроване середовище розробки для мови R, відомої своїм застосуванням в статистиці та аналізі даних. R Studio забезпечує потужні інструменти для аналізу даних, статистичного моделювання та візуалізації.

TensorFlow, розроблений Google, представляє собою один з провідних фреймворків у сфері машинного навчання та глибокого навчання. Цей фреймворк є вибором багатьох розробників завдяки його гнучкості, масштабованості та інтуїтивно зрозумілому інструментарію, що робить його придатним як для академічних досліджень, так і для комерційного використання. Особливістю TensorFlow є його здатність використовувати графи потоків даних, які полегшують моделювання складних архітектур нейронних мереж.

Хоча Google обмежує використання TensorFlow одночасним використанням на кількох графічних процесорах однієї машини, це не зупиняє застосування цього інструменту на рівні підприємств. Звичайно, існують обхідні шляхи для цього обмеження, хоча вони можуть вимагати додаткових зусиль, знань та ресурсів.

TensorFlow знайшов своє застосування в ряді великих компаній, включаючи Airbnb, Airbus, Dropbox, Snapchat і Uber, що свідчить про його універсальність і ефективність. Він використовується для різноманітних цілей, від простих до складних, демонструючи свою гнучкість і потужність.

Одним із прикладів інноваційного використання TensorFlow є британський електронний супермаркет Ocado, який застосовує цю бібліотеку для різних завдань - від оптимізації маршрутизації роботів на складах до поліпшення прогнозування попиту. Особливо вражаючим є використання TensorFlow для управління вхідною

електронною поштою, дозволяючи Ocado обробляти листи в залежності від їх пріоритетності та змісту.

Використання TensorFlow спрощує задачі обробки природної мови та інші завдання, які традиційно вважаються складними. Відкритий код і наявність інтерфейсів API для Python і C/C++ дозволяють легко інтегрувати TensorFlow у різні проекти.

TensorFlow, на додаток до своєї потужності в області глибокого навчання, пропонує можливості для інтеграції штучного інтелекту в застосунки, що включають розпізнавання мови, комп'ютерний зір та обробку природної мови. Це робить TensorFlow одним з найбільш впливових інструментів у сучасному світі машинного навчання і штучного інтелекту[9].

## 1.6 Висновки по першому розділу

У першому розділі було розглянуто різноманітні аспекти хмарних сервісів, включаючи їхню гнучкість, масштабованість та ефективність. Аналіз показав, що хмарні технології є не лише потужними інструментами сучасної IT-індустрії, але й надзвичайно актуальними в контексті розвитку програмного забезпечення та обробки даних. Це підтверджує актуальність обраної теми та важливість подальших досліджень і розвитку в цій області.

# 2 SAAS-ТЕХНОЛОГІЇ ТА РЕАЛІЗАЦІЯ РОЗПІЗНАВАННЯ МОВИ

## 2.1 Основи Natural Language Processing для перетворення голосу в текст

Natural Language Processing (NLP) є ключовим напрямком у галузі інформатики та штучного інтелекту (AI), присвяченим аналізу і обробці природної мови машинами. Основна мета NLP полягає в тому, щоб навчити комп'ютери ефективно інтерпретувати, розуміти та маніпулювати людською мовою, перетворюючи її на корисні формати для комп'ютерної обробки.

Одним із основних застосувань NLP є перетворення голосу в текст, що дозволяє комп'ютерам розпізнавати і транскрибувати мовлення людини. Цей процес включає в себе різні техніки та алгоритми, які аналізують голосові дані і переводять їх у письмовий текст.

Процес розпізнавання мови включає декілька етапів:

- Захоплення Звуку: Спочатку голосовий сигнал захоплюється за допомогою мікрофона.
- Попередня Обробка: Сигнал обробляється для усунення шумів та покращення якості звуку.
- Сегментація: Звуковий потік розділяється на окремі слова чи фрази.
- Розпізнавання Слів: Слова розпізнаються за допомогою алгоритмів машинного навчання, які порівнюють захоплені звуки з великою базою даних мовних шаблонів.
- Транскрипція та Аналіз: Після розпізнавання слова перетворюються у текст та аналізуються для виявлення граматичної структури, контексту та інших мовних характеристик.

Застосування NLP у сфері розпізнавання мови є широким і різноманітним.

1. Системи голосового управління: Які дозволяють користувачам взаємодіяти з пристроями та сервісами за допомогою голосових команд.
2. Голосові помічники: Як Siri, Google Assistant та Alexa, які розуміють голосові команди та виконують відповідні дії.

3. Текстові системи узагальнення: Які автоматично створюють короткі зведення з довгих текстів.
4. Системи машинного перекладу: Які перекладають голосові або текстові дані з однієї мови на іншу.
5. Системи розпізнавання іменованих сутностей: Які ідентифікують і класифікують ключові елементи в тексті (наприклад, імена, місця, організації).
6. Системи відповідей на питання: Які обробляють запити користувачів і надають релевантні відповіді.

Розпізнавання мови або Speech-to-Text (STT) – технологія перетворення мови в текст. Це багаторівневий процес аналізу акустичних сигналів, їх структурування в слова, фрази, пропозиції й перетворення в текстовий формат.

В наші часи в усьому світі ведуться роботи зі створення більш природних, ніж існуючі, для людини засобів спілкування з комп'ютером, серед яких присутній і мовне введення інформації. Багатьма розробниками відзначені успіхи в області розпізнавання мови,

Непоганих успіхів достигла корпорація Google Inc., яка пропонує мовне введення при здійсненні пошуку в мережі

Проте, проблема мовного введення інформації стає складнішою із-за деяких факторів: відмінністю структури мов, специфікою вимови, шумами та перешкодами, акцентами, наголосами й т. п. Існуючі на сьогоднішні системи розпізнавання мови ґрунтується на зборі всієї доступної й навіть надлишкової інформації, необхідної для розпізнавання лексичних елементів. Системи подібні розпізнаванню мови великих компаній, таких як Google Inc. Використовують широку базу мовних патернів сотень і навіть тисяч дикторів, що дозволяє їм домагатися впевненого розпізнавання багатьох слів невідомих дикторів.

Класифіковані образи можуть являти собою різні структурні елементи, відрізки мовних даних певної тривалості (фонема, склади, слова). Чим більше ми припускаємо апріорної інформації про вхідний сигнал, тим якісніше ми можемо його обробити та розпізнати.

Зазвичай для введення використовуються або окремі слова і словосполучення, або потрібно знайти слова маркери в злитої промови. Розпізнавання злитої промови набагато важче у зв'язку з тим, що межі окремих слів не чітко визначені і їх вимова сильно спотворено змазуванням і ковтанням деяких вимовних звуків.

При аналізі мови, як базової одиниці аналізу можуть бути обрані окремі слова і словосполучення, склади, а також такі елементи як фонеми, алофони, діфони і, рідше, трифони. Від типу лексичної структурної одиниці залежить, як складність системи в цілому, так і якість розпізнавання, і частинка словника.

Одна з найпомітніших відмінностей в розпізнаванні мови – це здатність розбиратися з акцентами і фоновим шумом. Пряма причина цього полягає в тому, що база для тренування моделі складаються з української мови з різними діалектами, англійської мови з американським акцентом, а також російські дані з високим показником відношення до шуму.

Однак збільшення кількості даних для навчання, ймовірно, не вирішить проблему просто так. Існує безліч мов з великою кількістю діалектів і акцентів. Неможливо зібрати достатньо даних для всіх випадків. Створення якісної системи розпізнавання мови тільки для української мови вимагає сім тисяч годин транскрибуватися аудіо.[11]

Основний метод, який використовується в більшості сучасних систем автоматичного розпізнавання мови – метод прихованих Марковських моделей. Сама імовірнісна модель цього виду була запропонована А. А. Маркова в 1913 р для аналізу письмових текстів  і прекрасно себе зарекомендувала в цій області . З 70-х років почалися роботи з адаптації цієї моделі до автоматичного розпізнавання мови .

Опис мовних сигналів практично в довільній послідовності використовуються всі ті ознаки, які випробували в мовних дослідженнях: рівні в спектральних смугах, ті чи інші формантного ознаки, коефіцієнти лінійного прогнозу. Жоден з можливих наборів не дає явної переваги, і результат практично залежить від акуратного набору статистики. З вибором сегментів, для яких

будується Марковська модель, ситуація складається аналогічним чином. У початкових варіантах моделі передбачалося, що відповідні сегменти можуть бути прив'язані до фонем, аллофон або будь-яким іншим фонетично виправданим елементам.

Таким чином, побудова систем автоматичного розпізнавання мови на основі прихованої Марківського моделі передбачає вірогідну організацію мовної поведінки людини. Але таке припущення не є очевидним.

Крім того, повинна забезпечуватися достатня точність передачі смислової інформації при різних варіантах порушень (не тільки патологічних, але частіше за все ситуаційних) процесів створення і сприйняття. На відміну від письмової мови у звичайному усному варіанті немає можливості повернутися до початку тексту і спробувати його заново осмислити.

## 2.2 Програмне забезпечення Docker, git, Django та пакет Speech Recognition з Recognize Google

Програмне забезпечення таке як Docker та git допомагають у розробці SaaS-сервісу, та грають важливу роль в подальшій підтримці проєкту. Як було написано вище до SaaS-додатків є наступні вимоги:

- Максимальна платформонезалежність: Додатки мають бути сумісними з різними операційними системами, що забезпечує гнучкість щодо середовищ виконання.
- Сумісність з хмарними платформами: Важливо, щоб додатки могли легко інтегруватися з сучасними хмарними сервісами, що сприяє їхньому швидкому розгортанню.
- Зменшення розбіжностей між розробкою та виробництвом: Це дозволяє використовувати неперервне розгортання (CI/CD) для забезпечення безперебійної інтеграції та доставки оновлень.

- Масштабованість без значних змін: Додатки повинні легко адаптуватися до змін у вимогах та навантаженні, не вимагаючи глобальних змін в інструментах, архітектурі чи процесі розробки.

Docker контейнери та git технології є ключовими елементами для досягнення цих цілей. Docker забезпечує необхідну ізоляцію та консистентність середовища, в той час як git пропонує ефективне управління версіями та співпрацю розробників.

Незалежно від вибраної мови програмування або напрямку розробки, код, який пише програміст, залишається звичайним текстом, записаним в безлічі файлів на диску. Ці файли регулярно додаються, видаляються і змінюються. Деякі з них можуть містити сотні рядків коду, а інші тисячі.

Допустимо що над проєктом працює понад два програміста або потрібно додати новий функціонал для подальшої роботи програми. Саме Git допомагає зробити це без проблем та зайвих зусиль. Виконується за допомогою спеціальних програм, які вміють відстежувати зміни коду. Ось деякі з можливостей даних систем:

- Повернення до будь-якої версії коду з минулого.
- Перегляд історії змін.
- Спільна робота без остраху втратити дані або затерти чужу роботу.

Git – це розподілена система керування версіями, створена Лінусом Торвальдсом, автором ядра Linux. Завдяки своїм особливостям і гнучкості, git є однією з найпопулярніших систем керування версіями серед розробників програмного забезпечення.

Git дозволяє розробникам відстежувати та керувати змінами у коді, забезпечуючи можливість відновлення попередніх версій та аналіз змін.

Завдяки можливості створення гілок, розробники можуть паралельно працювати над різними завданнями або функціями проекту, а потім зливати свої зміни без конфліктів.

Інтеграція з іншими інструментами: git інтегрується з багатьма іншими інструментами розробки, такими як системи неперервної інтеграції/доставки (CI/CD), трекери завдань та інші.

Git-репозиторій, завантажений на GitHub, доступний за допомогою інтерфейсу командного рядка Git і Git-команд. Також є й інші функції: документація, запити на прийняття змін (pull requests), історія коммітов.

Для швидкого та легкого розгортання проєкту на сервері будемо використовувати Docker.

Docker – це стандартизоване пакетне програмне забезпечення, для розробки і розгортання проєктів. Його принцип роботи найпростіше порівняти з транспортними контейнерами. [13]

Docker дозволяє стандартизувати і уніфікувати процеси розробки, тестування та впровадження, гарантуючи консистентність між різними середовищами.

Docker використовує принцип контейнеризації для ізоляції додатку та його залежностей в одному контейнері. Це забезпечує, що додаток буде працювати однаково в будь-якому середовищі, оскільки всі необхідні компоненти (бібліотеки, системні інструменти, код, тощо) упаковуються разом.

Дозволяє створювати стандартизовані блоки (контейнери), які можна легко переміщати та розгортати на різних системах. Це сприяє легкості впровадження та масштабування додатків.

Вирішення проблеми "працює у мене": Однією з основних переваг Docker є його здатність вирішувати проблему, коли додаток працює на машині розробника, але не працює в іншому середовищі. Контейнери Docker забезпечують однакове середовище виконання незалежно від локальних налаштувань.

Ізоляція та безпека: Контейнери Docker ізольовані один від одного та від хост-системи, що забезпечує вищий рівень безпеки. Зміни в одному контейнері не впливають на інші контейнери або на хост-систему.

Це допоміжне програмне забезпечення для побудови SaaS-сервісу, але головний фреймворк, за допомогою якої будуються запити до сервера та взаємодія з інтерфейсом користувача є Django.

По-перше, сервер повинен дізнатися про те, що ми чекаємо від нього веб-сторінку. Коли на сервер приходить запит, він переадресовується Django, який

намагається збагнути, що ж конкретно від нього просять. Для початку він бере адресу веб-сторінки і намагається зрозуміти – що ж потрібно зробити. Цю частину процесу в Django виконує urlresolver. Django звіряє шаблони зверху вниз і, якщо щось збігається, він перенаправляє запит відповідної функції (яка називається view).

Функціональність "view" у Django відіграє ключову роль, оскільки дозволяє запитувати дані з бази даних. Розробка Django була націлена на спрощення роботи з новинними порталами, що зумовило її архітектуру: фреймворк містить інструменти для швидкої побудови інформаційних сайтів. У Django існує вбудований інструментарій для управління контентом, що дозволяє уникнути ручної роботи зі створення контролерів та адміністративних сторінок, полегшуючи управління декількома сайтами одночасно. Django як потужний інструмент на Python надає розробникам широкі можливості для оптимізації розробки веб-додатків. [15]

Django, потужний веб-фреймворк на Python, пропонує широкий спектр ключових можливостей, які спрощують і оптимізують процес розробки веб-додатків.

Однією з таких можливостей є ORM (Object-Relational Mapping) та API для доступу до баз даних, де Django ORM дозволяє розробникам інтерактивно працювати з базами даних, використовуючи Python-класи замість прямих SQL-запитів, а підтримка транзакцій забезпечує надійність і цілісність даних. Додатково, вбудований інтерфейс адміністратора забезпечує просте управління даними і підтримує багатомовність, завдяки вже наявним перекладам на численні мови.

Диспетчер URL використовує регулярні вирази для гнучкого визначення маршрутів до обробників запитів.

Система шаблонів у Django є дуже гнучкою, пропонуючи розширювальні шаблони з можливістю використання тегів та спадкування, дозволяючи відокремлювати візуальне представлення від бізнес-логіки. Крім того, система

кешування значно покращує продуктивність, зменшуючи час завантаження сторінок та навантаження на сервер.

Архітектура Model-View-Template (MVT), що використовується у Django, представляє собою варіацію традиційної моделі Model-View-Controller (MVC) і є ключовою для розробки веб-додатків, оскільки вона забезпечує чітке розділення логіки, інтерфейсу та управління даними.

У MVT, модель представляє структуру даних та бізнес-логіку, взаємодіючи безпосередньо з базою даних і виконуючи такі операції, як створення, оновлення та видалення.

Моделі описуються у Django за допомогою класів Python, де вони визначають поля та поведінку даних.

Уявлення в Django відповідають за обробку HTTP-запитів, виклик відповідної бізнес-логіки, і, виходячи з цих даних, вибирають відповідний шаблон для відображення відповіді. Це може бути функція або клас, що виконує певну логіку та повертає HTTPResponse.

Шаблони в Django відповідають за відображення інтерфейсу користувача і містять статичний HTML-код, який може бути доповнений спеціальними тегами та змінними Django для динамічного відображення контенту. Важливо, що шаблони не містять бізнес-логіки, вони лише відображають дані, отримані з уявлень.

Особливості архітектури MVT в Django включають розділення відповідальності між компонентами, що спрощує розробку, тестування та обслуговування веб-додатків, а також гнучкість та розширюваність, що дозволяє легко модифікувати кожен аспект веб-додатку. Також вона підтримує перевикористання компонентів, особливо моделей та шаблонів, у різних частинах додатку або навіть у різних проектах. Завдяки архітектурі MVT, Django забезпечує чітку структуру та організацію коду, роблячи процес розробки інтуїтивно зрозумілим і ефективним, особливо для великих і складних веб-проектів.[15]

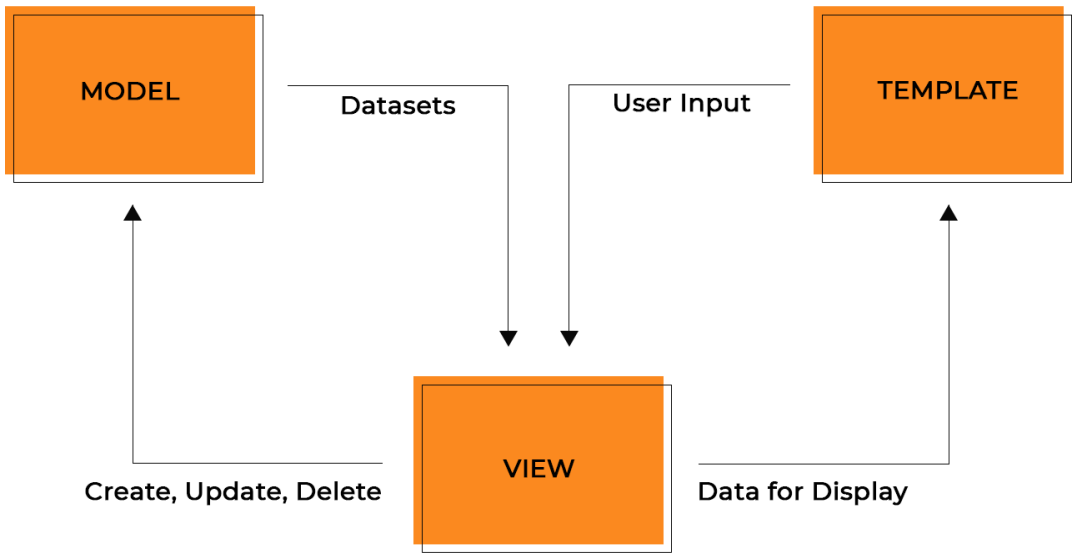



Рисунок 2.1 – Схема архітектури MVT в Django

Django підтримує різні бази даних, включаючи PostgreSQL, MySQL, SQLite та Oracle, забезпечуючи гнучкість у виборі засобів зберігання даних.[16]

Django є ідеальним вибором для розробки сучасних веб-додатків, завдяки своїм високо рівневим функціям та легкості використання. Він дозволяє розробникам швидко створювати надійні та ефективні веб-додатки, зосереджуючись на важливих аспектах проекту, а не на низькорівневих деталях.

### 2.3 Хостинг власної SaaS-системи

Як звертатися до сервера зрозуміло, тепер потрібно вибрати хостинг, в якому є всі можливості для нашої задачі.

Кожен хостинг по-своєму визначає межі допустимого в безкоштовному тарифі і всі вони пропонують платні додаткові опції для тих, кому потрібні більш просунуті інструменти. Моєю задачею було визначити, як розвернути свою SaaS-модель безкоштовно.

Перший сервіс який пропонує безкоштовний хостинг – GearHost.

Сервіс пропонує п'ять різних хмарних тарифів, самий базовий з яких не просто безкоштовний, але безкоштовний назавжди.

Пам'ятайте, що root-доступу тут немає, так що налаштування доведеться вносити єдино через інтерфейс системи. У такого підходу, зрозуміло, є обмеження: ми не зможемо використовувати відмінні від .NET або PHP середовища розробки. Втім, це лише спрощує процес запуску хмарного інстанси онлайн – консоллю користуватися не доведеться.

Всі пропозиції CloudSite дають вам докладну інформацію про роботу серверів – навантаженні на процесори, особливостям взаємодії відвідувачів з вашим проєктом, а також відомості про можливі проблеми.

Хмарний вебхостинг від Amazon (AWS). Безкоштовний тариф AWS має 12-місячний доступ і по 750 обчислювальних годин в місяць.

У безкоштовному тарифі також доступні мікроінстанси, які можна запустити на Linux і Windows - вони підходять для ситуацій, коли вам потрібна продуктивність від невеликої до середньої. 1 vCPU і 1 ГБ RAM, також зможемо залучили додаткові ЦП при необхідності.

Google Cloud – безкоштовні інстанси в США з великим дисковим сховищем. GCP - це хмарний сервіс від Google, де є безліч рішень. Як наслідок, заплутатися тут дуже просто. Кількість опцій настільки велике, що розробники початківці можуть всерйоз задуматися про те, чи не варто краще скористатися послугами звичайних провайдерів.

Інстанс має 1 vCPU, 0,6 ГБ RAM, 30 ГБ хмарного сховища, 5 ГБ сховища для снапшотов і 1 ГБ пропускної здатності.

Azure пропонує безкоштовний сервіс, який дуже схожий на AWS: щось безкоштовно протягом 1 року, щось безкоштовно назавжди, з чимось можна попрацювати в пробному режимі, використовуючи кредити. Хмарні ресурси, безкоштовно доступні протягом 12 місяців, містить віртуальні машини Linux, дисковий простір, бази даних і пропускну здатність.

Ще один хостинг, не зовсім безкоштовний, проте є 30-дневний пробний період. Звичайно, безкоштовні хмарні хостинги можуть здатися дуже привабливим варіантом, однак в першу чергу необхідно задуматися, чи готові ви до того рівня відповідальності та компетенцій, яка потрібна для роботи з ними.

Beget - це мій вибір, бо у цього сервісу відмінна техпідтримка, чудову якість роботи. Завдяки співпраці зі світовими лідерами високих технологій, такими як: Intel, Supermicro, Cisco, Juniper, WD, Seagate і ін. Сервіс пропонуємо розміщення сайтів тільки на найпотужнішому, надійному і швидкому обладнанні.

Надійність і безпека роботи серверів забезпечується:

- Багаторівневої перевіркою серверів;
- Регулярними профілактичними роботами;
- Системою резервного копіювання;
- Резервними каналами інтернету;
- надійним ЦОД.

### 2.4 Функції реалізації інтерфейсу NLP розпізнавання мови

Розпізнавання мови в SaaS-моделі було реалізовано за допомогою відкритої бібліотеки Speech recognition. Ця бібліотека представляє з себе обгортку над багатьма популярними сервісами / бібліотеками розпізнавання мови.

Для встановлення бібліотеки потрібно в командному рядку ввести наступну команду:

Pip install SpeechRecognition.

Для перетворення мови в текст нам потрібен клас Recognizer з модуля speech_recognition. Залежно від базового API, використовуваного для перетворення мови в текст, клас Recognizer має наступні методи:

1. recognize_bing (): Використовує Microsoft Bing Speech API;
2. recognize_google (): Використовує Google Speech API;
3. recognize_google_cloud (): Використовує Google Cloud Speech API;
4. recognize_houndify (): Використовує Houndify API від SoundHound;
5. recognize_ibm (): Використовує IBM Speech to Text API;
6. recognize_sphinx (): Використовує PocketSphinx API.

В даному проєкті використовується recognize_google (), бо в даному програмному інтерфейсі значно краще розпізнавання мови в зашумлених даних,

так що матеріал не потрібно попередньо очищати, обробляючи фільтрами або використовуючи дороге обладнання і мікрофони для шумозаглушення.

Розпізнавання мови працює для 80 мов. Можливо розпізнавання мови в прямому ефірі через мікрофон або аудіозаписів з файлів (ймовірно, до 2 хвилин).

Для деяких мов підтримується автоматична фільтрація небажаного контенту.

Існує кілька способів обробки аудіо, отриманого через мікрофон, і для цього були розроблені різні бібліотеки. Однією з таких бібліотек є PyAudio.

Встановити її можна за допомогою: pip install PyAudio.

Захоплення звуку з мікрофона. Для цього потрібно викликати метод listen () класу Recognizer (), який потім передаємо методу recognize_google ().

```
@login_required
def add(request):
    if request.method == 'POST':
        form = AddPostForm(request.POST)
        if request.POST.get("sub"):
            if form.is_valid():
                diary_entries(text=form.cleaned_data['text'], user_id=request.user.id).save()
                return redirect('home')
        elif request.POST.get("rec"):
            r = sr.Recognizer()
            with sr.Microphone(device_index=1) as source:
                audio = r.listen(source)
            if audio:
                try:
                    query = r.recognize_google(audio, language='uk-UA')# uk-UA
                except sr.UnknownValueError:
                    print("Google Speech Recognition could not understand audio")
                except sr.RequestError as e:
                    print("Could not request results from Google Speech Recognition service; {0}".format(e))
            if form.is_valid():
                text = form.cleaned_data['text'] + " "
            text = text + query
            data = {'text': text}
            form = AddPostForm(initial=data)
    else:
        data = {'text': ' '}
        form = AddPostForm(initial=data)
    return render(request, 'Nlp/add.html', {'form': form, 'menu': menu, 'title': 'Запис'})

```

Рисунок 2.2 – View add реалізація додавання запису в базу даних

@login_required відповідає за те, щоб незареєстрований користувач не зміг додати запис в базу даних, бо для додавання запису потрібно знати id користувача.

Функція render об'єднує заданий шаблон add.html з заданим контекстним словником і повертає HttpResponse об’єкт з цими даними.

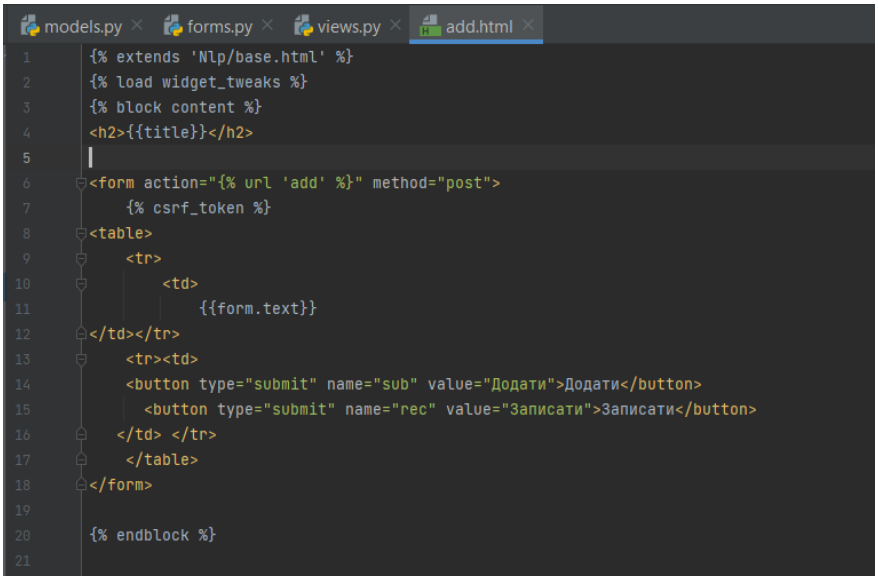



Рисунок 2.3 – Шаблон add.html

Завдяки базовому шаблону, який ми розширюємо шаблоном base.html, все що потрібно зробити, це показати форму, котра передавалася в шаблон, встановити токен для захисту даних користувача. Тег шаблону забезпечує простий захист від підробки міжсайтових запитів.

### 2.5 Налаштування робочих параметрів Django

В Django автоматично створюється багато допоміжних файлів, один із них це settings.

Файл налаштувань Django містить всю конфігурацію установки Django. У цьому документі пояснюється, як працюють налаштування і які налаштування доступні.

Основні зміні наступні:

DEBUG = true/false;

ALLOWED_HOSTS = [‘127.0.0.1’]

DATABASES = … до якої бази даних будемо підключатися та параметри підключення таки, як логін та пароль;

LOGIN_URL = ‘/login’ – в даному випадку, цей параметр відповідає за @login_required.

### 2.6 Висновки по другому розділу

Після ретельного аналізу наявних інструментів та технологій, було обрано специфічний набір ресурсів для виконання поставленої задачі. Використання Natural Language Processing (NLP) для перетворення голосу в текст, дозволяючи ефективно обробляти та інтерпретувати мовлення людини.

PyAudio - бібліотека, яка дозволяє захоплювати звук з мікрофона для подальшого аналізу.

Також важливим компонентом є recognize_google від бібліотеки Speech Recognition, який ми використовуємо для перетворення мови в текст.

Для розробки та розгортання SaaS-сервісу було обрано такі інструменти, як Docker, Git і Django, які забезпечують необхідну інфраструктуру, гнучкість та ефективність в розробці. Вибір цих технологій заснований на їх здатності до масштабування, підтримці мінімізації розбіжностей між розробкою та виробництвом, а також інтеграції з хмарними платформами. Таким чином, вибір і використання цих інструментів та технологій дозволили створити високоефективний SaaS-продукт, який відповідає сучасним вимогам ринку.

# 3 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ОБРОБКИ ПРИРОДНОЇ МОВИ НА ОСНОВІ ХМАРНИХ СЕРВІСІВ

## 3.1 Склад і структура програмного забезпечення системи

Загальна структура інформаційної системи подана на рис. 3.1

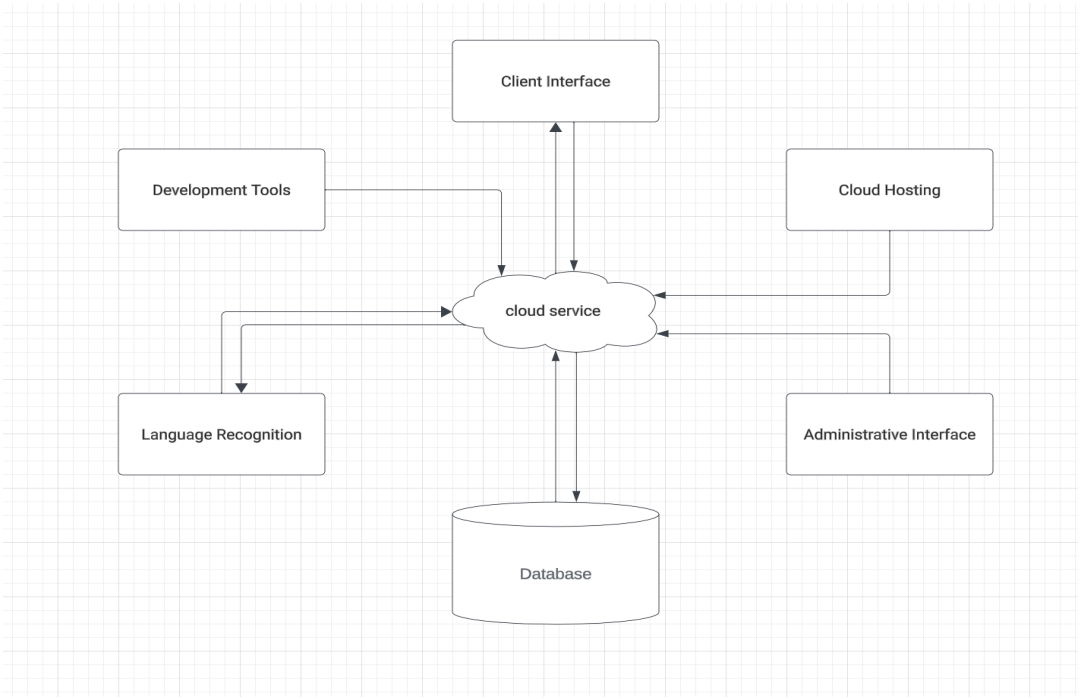



Рисунок 3.1 – схема інформаційної технології

У сучасному програмуванні ефективність розробки та підтримки великих і складних систем є критичною. Архітектурний шаблон Model-View-Controller (MVC) став золотим стандартом для створення розширюваних і легко підтримуваних веб-додатків. Цей паттерн впорядковує код та спрощує розподіл завдань у команді, роблячи процес розробки більш логічним і зрозумілим. Наступний опис детально розглядає кожен компонент архітектурного шаблону MVC, його роль та важливість у структурі сучасних програмних рішень.

Model - це ядро додатка, що управляє даними, логікою та правилами. Вона функціонує незалежно і не взаємодіє безпосередньо з користувацьким інтерфейсом. Модель може бути як шаром даних, так і менеджером бази даних або набором об'єктів, що забезпечують бізнес-логіку програми.

View відповідає за логіку відображення інформації користувачеві. Він отримує дані з моделі та представляє їх, маючи доступ лише на читання. В певних випадках, вид може містити бізнес-логіку, але основне його завдання - це відображення інформації.

Controller забезпечує зв'язок між моделлю та видом, керуючи бізнес-логікою та запитами користувачів. Він взаємодіє з моделлю для зміни даних та з видом для представлення результату.

Застосування MVC гарантує добре організований код, який відокремлює внутрішнє представлення інформації від способів її презентації та прийняття від користувача, забезпечуючи гнучкість і зручність в редагуванні та обслуговуванні. В нашій програмі в моделі знаходиться структура бази даних. Для зберігання даних багатьох користувачів в SaaS-моделі потрібно вирішити, як саме буде зберігатися інформація кожного користувача.

З одного боку по базі на клієнта – неприпустиме марнотратство.

Якщо ж база одна – незабаром якась таблиця, що швидко зростає стане «некерованою» і серйозно просадив продуктивність. Звичайно можна розносити такі таблиці, але також є більш цікаве рішення.

Можна тримати кілька БД і в кожній кілька тисяч користувачів. При цьому нові бази можна додавати в міру розвитку.

Спочатку взагалі буде достатньо однієї бази.

Одну з БД першу призначаємо головною або «системної» базою. В ній будуть зберігатися дані загальні для всієї системи. Наприклад новини системи, глобальні настройки і звичайно головне - список користувачів.

У кожного з твоїх підходів свої плюси і мінуси.

Якщо одна база плюси:

- Зручно оперувати однією базою
- Зручно накочувати скрипти поновлення на одну базу
- Зручно адмініструвати одну базу

Мінуси такого підходу база розростається сильно, якщо багато клієнтів

Таблиці розростаються, швидкодія знижується

Якщо багато баз даних то значно більше мінусів для даної програми, а саме:

- Не зручно оперувати декількома базами базою
- Не зручно накочувати скрипти поновлення на багато баз
- Не зручно адмініструвати багато баз, стають складнішими адміністративні інтерфейси.

Створення проєкту на Django починається з команди django-admin.py startproject NLP_SaaS-project.

В одному каталозі з файлом manage.py створюється новий додаток командою 'python manage.py startapp Nlp'.

Django дозволяє розробникам ефективно організовувати нові частини програми завдяки своїй структурі, що складається з пакетів Python. Це відволікає від потреби створення каталогів, дозволяючи зосередитися на написанні коду. Для налаштування баз даних на початковому етапі часто використовується SQLite3, яка зберігає всі дані в одному файлі на диску, що спрощує початкове налаштування бази даних.

```
class MyUser(AbstractUser):
    paid_subscription = models.BooleanField(default=False)


class diary_entries(models.Model):
    text = models.TextField(verbose_name='Diary entry')
    time_create = models.DateTimeField(auto_now_add=True, verbose_name='Time create')
    time_update = models.DateTimeField(auto_now=True, verbose_name='Time update')
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, verbose_name='User')

    def __str__(self):
        return self.text

    def get_absolute_url(self):
        return reverse('post', kwargs={'post_id': self.pk})

    class Meta:
        verbose_name = 'User record'
        ordering = ['-time_create']
```

Рисунок 3.2 – Модель бази даних

Структура бази даних визначається в файлі models.py (рис.3.2). Створимо таблицю записів користувачів та розширяємо вбудовану модель User.

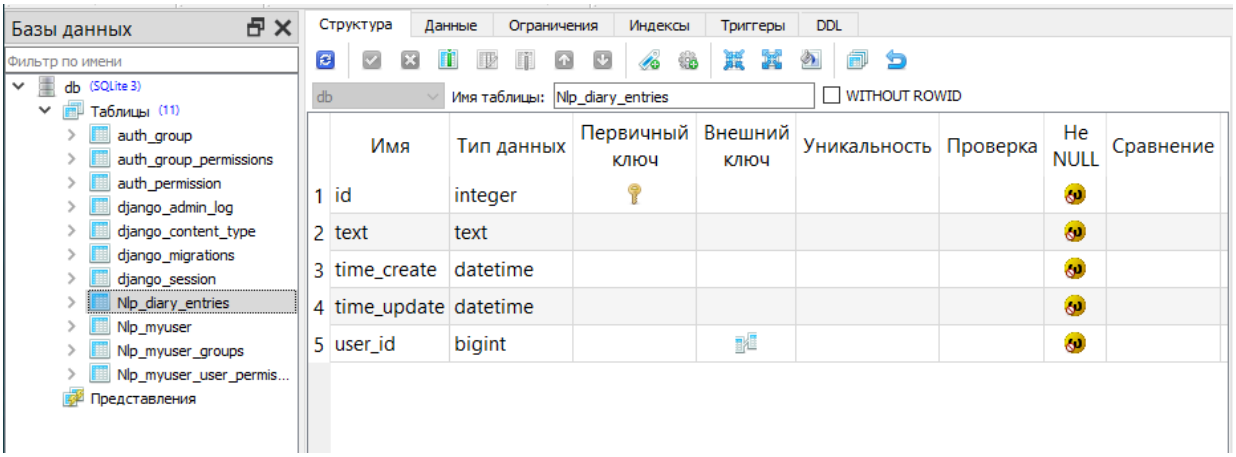



Рисунок 3.3 – База даних

Командою makemigrations створюємо таблиці:

python manage.py makemigrations.

Створює нові міграції на основі змін у моделях.

Синхронізує стан бази даних з поточним станом моделей і міграцій:

Python manage.py migrate.

В Django існують стандартні таблиці бази даних для проєкта.

- django.contrib.admin – Інтерфейс адміністратора. Ви будете використовувати його в частині 2 підручника;
- django.contrib.auth – Система аутентифікації;
- django.contrib.contenttypes – "content types" фреймворк;
- django.contrib.sessions – Фреймворк сесії;
- django.contrib.messages – Фреймворк повідомлень;
- django.contrib.staticfiles – Фреймворк для роботи зі статичними файлами.

Ці додатки включаються за замовчуванням, бо вони корисні для типових проєктів.

### 3.2 Інтерфейс користувача

Щоб зробити просту реєстрацію, потрібно використовувати базовий клас в Django «UserCreationForm», для створення форми й все, але для початку потрібно зробити шаблон, для початку базовий, а потім для реєстрації.

Базовий шаблон буде мати меню сторінки блок контенту, який будемо розширяти, а також footer. Меню та footer будуть статичні для кожної сторінки, саме тому їх потрібно помістити в базовий шаблон, щоб програмний код був максимально зручним для читання. (base.html Додаток А).

Далі створюємо форму реєстрації для того щоб потім передати її в шаблон (рис.3.4).

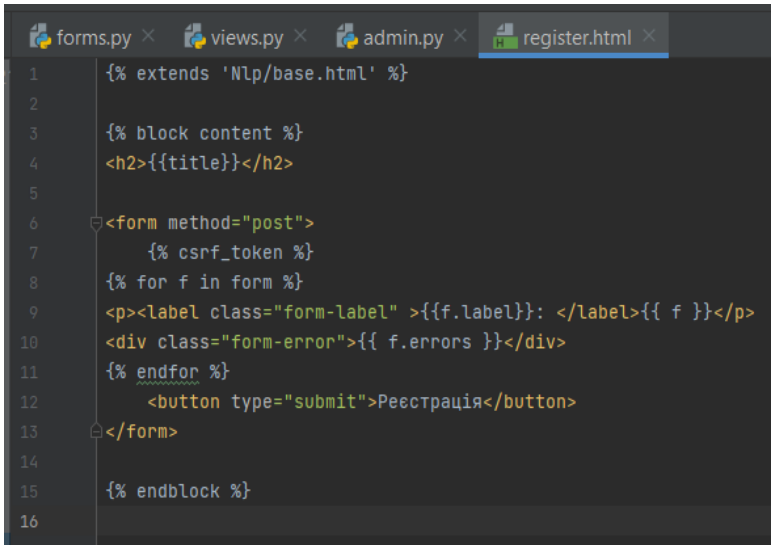
```
class RegisterUserForm(MyUserCreationForm):
    username = forms.CharField(label='Логін', widget=forms.TextInput(attrs={'class': 'form-input'}))
    first_name = forms.CharField(label="Ім'я",  widget=forms.TextInput(attrs={'class': 'form-input'}))
    last_name = forms.CharField(label='Прізвище',  widget=forms.TextInput(attrs={'class': 'form-input'}))
    email = forms.EmailField(label='Email', widget=forms.EmailInput(attrs={'class': 'form-input'}))
    password = forms.CharField(label='Пароль', widget=forms.PasswordInput(attrs={'class': 'form-input'}))

    confirm_password = forms.CharField(label='Пароль2', widget=forms.PasswordInput(attrs={'class': 'form-input'}, render_value=False))

    def clean_confirm_password(self):
        confirm_password = self.cleaned_data['confirm_password']
        original_password = self.cleaned_data['password']
        if original_password != confirm_password:
            raise forms.ValidationError("Password doesn't match")

        return confirm_password


    class Meta:
        model = MyUser
        fields = ('username', 'first_name', 'last_name', 'email', 'password')
```

Рисунок 3.4 – Форма реєстрації користувача

Функція clean_confirm_password потрібна обов’язково, якщо використовуємо стандартну розширену форму реєстрації. В базі даних пароль зашифровується, та при авторизації проходить зворотний шлях. В даному проєкті буде використовуватися стандартна форма авторизації тому потрібно пароль зберігати правильно.

Базовий шаблон розширяється за допомогою тега extends. Теги повинні поміщатися в {% extends 'Nlp/base.html' %}.

В шаблоні реєстрації потрібно тільки описати, що буде знаходитися в блоці контент, все інше буде взято з базового шаблона.




Рисунку 3.5 – вміст файлу register.html

CSRF, або Cross-Site Request Forgery (міжсайтовий підробка запиту) - це, можливо, одна з найбільш забуваються вразливостей. Розробники, як правило, знають про SQL ін'єкції і XSS атаки, але дуже часто забувають про CSRF-атаки.

CSRF-атака використовує браузер користувача – відкриті сесії й збережені куки, щоб надіслати запит зі шкідливими даними.

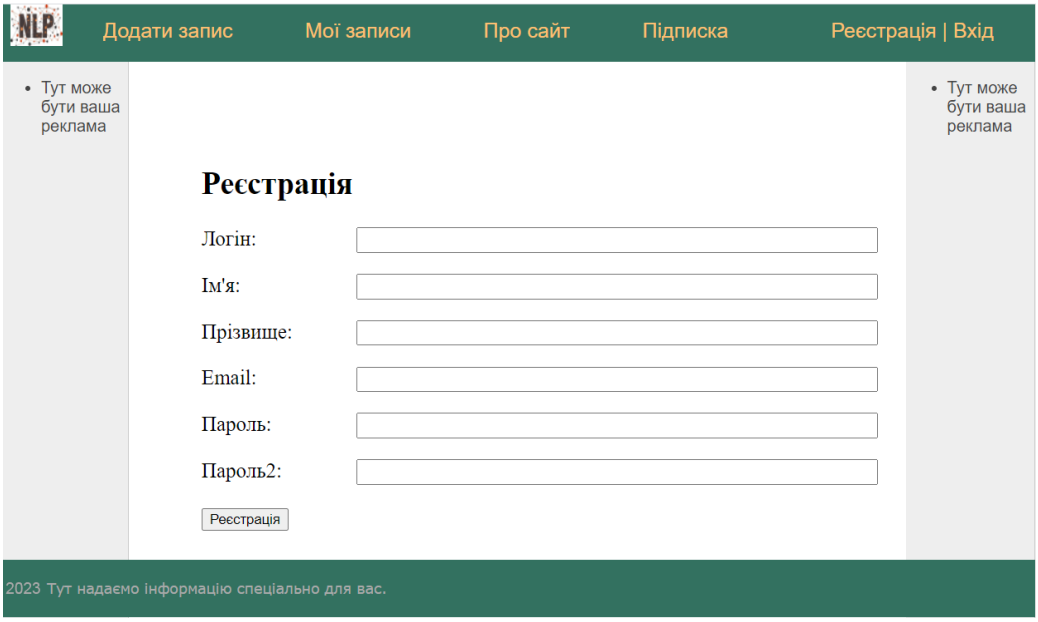



Рисунок 3.6 – Вкладка для реєстрації

Тепер створимо уявлення в Nlp/views.py.

```python
class LoginUser(DataMixin, LoginView):
    form_class = LoginUserForm
    template_name = 'Nlp/login.html'
    success_url = reverse_lazy('home')

    def get_context_data(self, *, object_list=None, **kwargs):
        context = super().get_context_data(**kwargs)
        c_def = self.get_user_context(title="Вхід")
        return dict(list(context.items()) + list(c_def.items()))

    def get_success_url(self):
        return reverse_lazy('home')
```

Рисунок 3.7 – Уявлення для авторизації користувача

Class LoginUser відповідає за авторизацію користувача. template_name спеціальна змінна яка приймає шаблон по якому будується html сторінка. success_url відповідає за перехід на сторінку після успішної авторизації.

Створюється форма для авторизації аналогічно попередній.

Головна сторінка представлена, як список всіх записів одного користувача. Поки користувач не авторизується доступ до головної сторінки заборонено та користувач перенаправляється на сторінку авторизації. За це відповідає команда @login_required. Куди саме перенаправляти користувача вказуємо в файлі settings.py вказавши зміну LOGIN_URL = '/login'.

```python
@login_required
def index(request):
    posts = diary_entries.objects.filter(user_id=request.user.id)
    context = {
        'posts': posts,
        'menu': menu,
        'title': 'Головна сторінка'
    }
    return render(request, 'Nlp/index.html', context=context)
```

Рисунок 3.8 – Уявлення головної сторінка

Django — це потужний фреймворк для веб-розробки, що містить вбудовану систему ORM для ефективної взаємодії з базами даних. ORM дозволяє

використовувати об'єктно-орієнтований підхід до баз даних, що спрощує розробку та підтримку проектів. Django також має вбудований шаблонний движок, який допомагає у створенні динамічних веб-сторінок, і набір інструментів для налагодження, які роблять розробку та відладку проектів більш зручною.[16]

diary_entries.objects.filter(user_id=request.user.id) – звертаємося до таблиці diary_entries та вибираємо всі записи певного користувача.

Саме ORM Django забезпечує незалежність програмного коду від конкретної СУБД і якщо в майбутньому знадобиться змінити тип бази даних, то зробити це буде дуже просто. Нема потреби переходити на рівень SQL запитів, бо ORM надає колосальну можливість взаємодіяти з базою даних.

ORM в Django добре оптимізує запити по швидкодії виконання і частоті звертання до таблиць, а також забезпечує захист від SQL ін'єкцій. Тому з легкістю можна створювати грамотний код по роботі з базою даних.

Уявлення потрібно зв’язати з URL адресою в файлі urls.py:

path('my/', index, name='my').

Також розширюємо базовий шаблон, та створюємо index.html (рисунок 3.9).

```
{% extends 'Nlp/base.html' %}

{% block content %}
<ul class="list-articles">
    {% for p in posts %}
    <li><div class="article-panel">
    <p class="last">Дата: {{p.time_update|date:"d-m-Y H:i:s"}}</p>
</div>
    {{p.text|linebreaks|truncatewords:50}}
            <div class="clear"></div>
        <p class="link-read-post"><a href="{{ p.get_absolute_url }}">Читати...</a>
            <a href="{{ p.get_delete }}">Видалити</a> </p>
            </li>
    {% endfor %}
</ul>
{% endblock %}
```

Рисунку 3.9 – вміст файлу index.html

В шаблоні задається фільтр, щоб показувати лише 50 символів із запису.

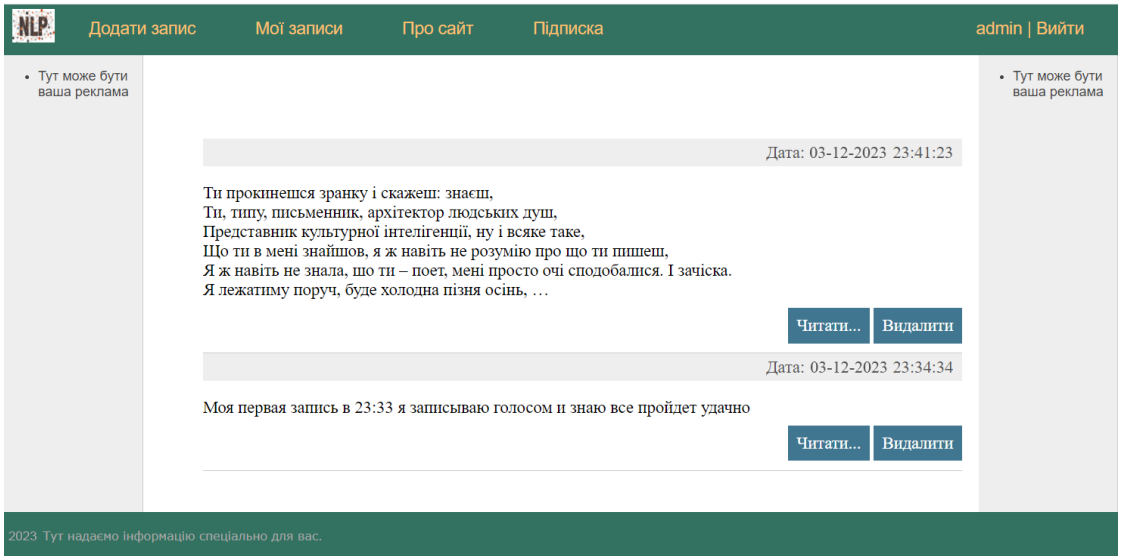



Рисунок 3.10 – Два записи користувача admin

Прочитати повністю запис можна натиснути «Читати…» та браузер перенаправить нас на get_absolute_url, який прописаний в models class diary_entries:

```
def get_absolute_url(self):
return reverse('post', kwargs={'post_id': self.pk})
```

View перегляд запису.

```
def show_post(request, post_id):
    post = get_object_or_404(diary_entries, pk=post_id)

    context = {
        'post': post,
        'menu': menu,
    }
    return render(request, 'Nlp/post.html', context=context)
```

На рисунку 3.11 – наведено вміст функції show_post

Програма використовує шаблон 'Nlp/post.html', щоб показати вміст на веб-сторінці.

Контекст— це набір даних, які визначають, що саме користувач побачить на сторінці. Django робить процес простим, автоматично обробляючи ці дані і відправляючи готовий веб-вміст користувачу. Усе, що потрібно розробнику — визначити дані для шаблону, а Django вже займається рештою.

Шаблон post.html розширяє базовий шаблон і щоб вивести запис достатньо передати в шаблон зміну, в якій знаходяться дані певного запису.

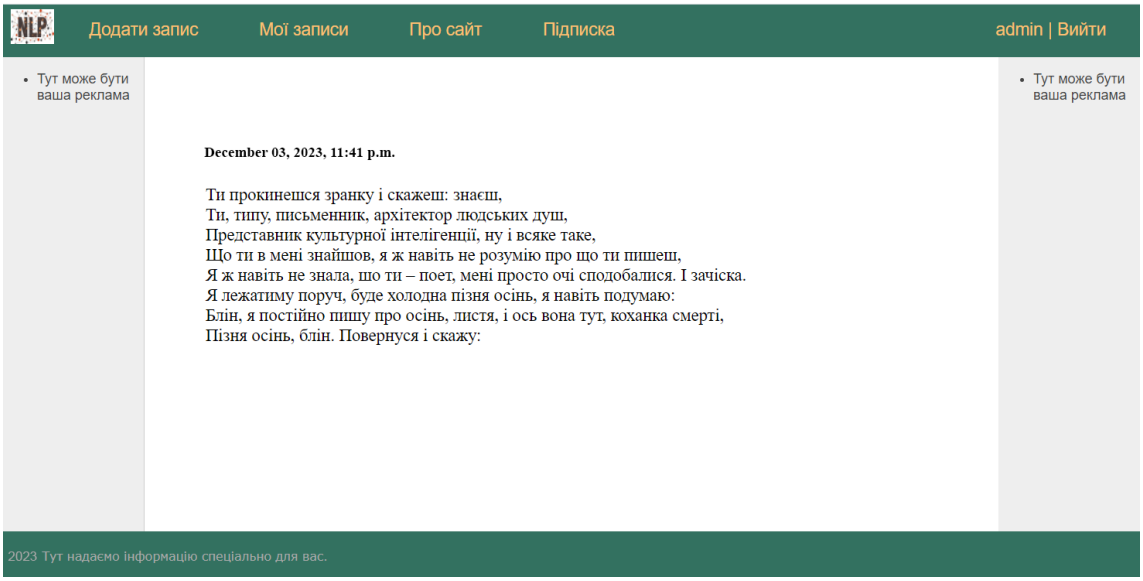



Рисунок 3.12 – Відображення повного тексту запису

Django дозволяє з’єднувати python код з шаблонами і це можна побачити в наступному view def add (Додаток А). За допомогою цієї можливості можна зв’язати з SaaS-моделлю будь яку модель штучного інтелекту, яка в даному випадку буде розпізнавати голос та переводити в текст за допомогою recognize_google.

Пакет PyAudio потрібен для захвату мікрофона та запису голосу для подальшої передачі на сервер google.

```
{% extends 'Nlp/base.html' %}
{% block content %}
<h2>{{title}}</h2>

<form action="{% url 'add' %}" method="post">
    {% csrf_token %}
<table><tr><td>
            {{form.text}}
</td></tr>
    <tr><td>
    <button type="submit" name="sub" value="Додати">Додати</button>
      <button type="submit" name="rec" value="Записати">Записати</button>
   </td> </tr></table>
</form>
{% endblock %}
```

Рисунок 3.13 – Шаблон html сторінки для додавання запису

В шаблон є дві кнопки типу submit але з різними назвами, в уявлені add зчитується яка кнопка була натиснута, та виконуються певні дії.

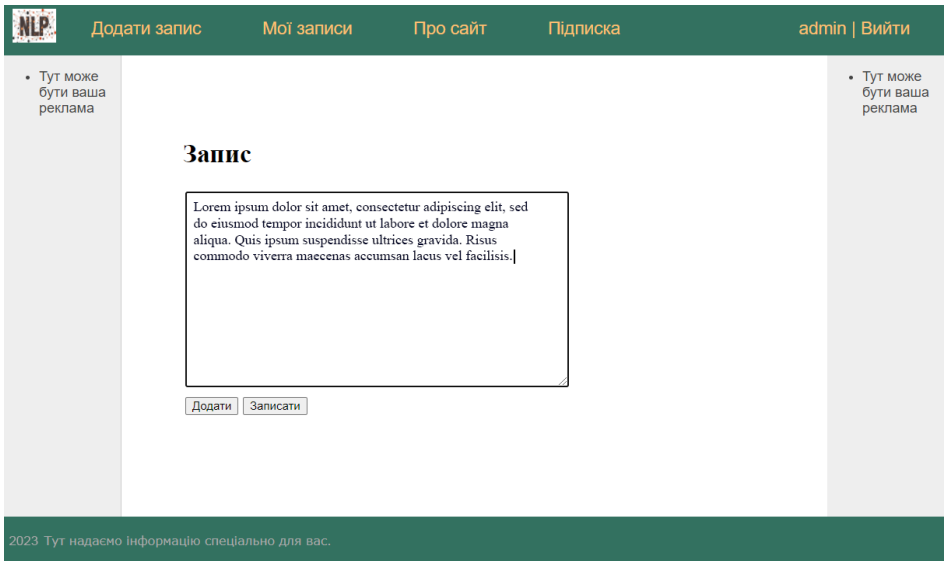



Рисунок 3.14 – Запис аудіо та перетворення голосу в текст

Натискаємо кнопку Записати і говоримо текст який хочемо записати, запис завершується коли користувач перестає говорити. Якщо модель не зможе розпізнати голос recognize_google викидає виключення UnknownValueError.

Django має вбудовану адмін панель і все, що потрібно зробити розробнику, налаштувати її під свої вимоги.

Створимо супер користувача командою python manage.py createsuperuser

Прописувати налаштування потрібно в файлі admin.py (Додаток А).

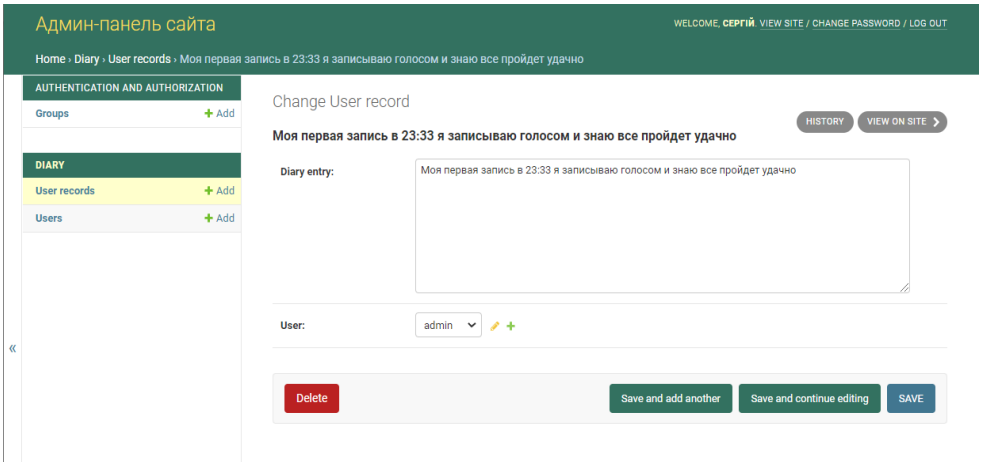



Рисунок 3.15 – Адмін панель SaaS-моделі

В адмін панель потрібно зареєструвати свою модель diary_entries, а також розширену версію користувачів, так як спочатку будо добавлено додаткове поле тариф в стандартну модель користувачів.

admin.site.register(diary_entries, diary_entriesAdmin).

Налаштовуються також поля які будуть показуватися в адмін панелі при перегляді користувачів.

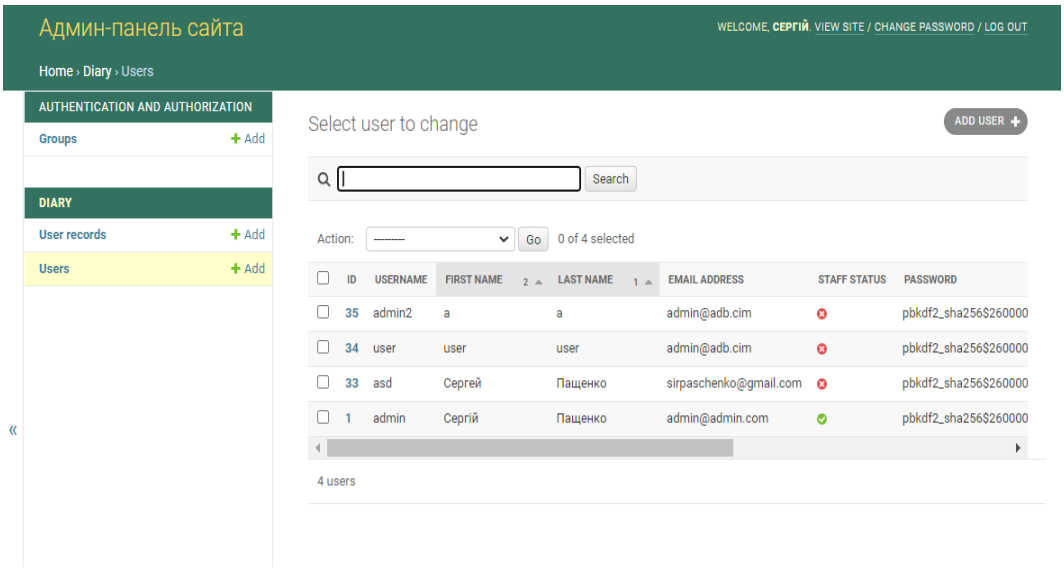



Рисунок 3.16 – Адмін панель користувачів

### 3.3 Завантаження проєкту на сервер та налаштування системи

Після завершення розробки проєкту на тестовому сервері, потрібно пред'явити його світу, тобто викласти на бойовому сервері. Для цього орендується хостинг або сервер де планується розташувати проєкт з файлами і базами даних. У кожного хостинг провайдера використовується своє конкретне обладнання та програмне забезпечення. Конкретні кроки по установці проєкту на різних хостингах буде відрізнятися, але загальний принцип схожий і далі буде показаний детально на хост сервері beget.

Спочатку нам потрібно вибрати тарифний план. Вибираємо найпростіший «блок» і у нього є тридцяти денний тестовий період. Після реєстрація нас перенаправляє на адмін панель і тут насамперед потрібно перейти на кладку FTP(рисунок 3.17) і активувати доступ ssh.

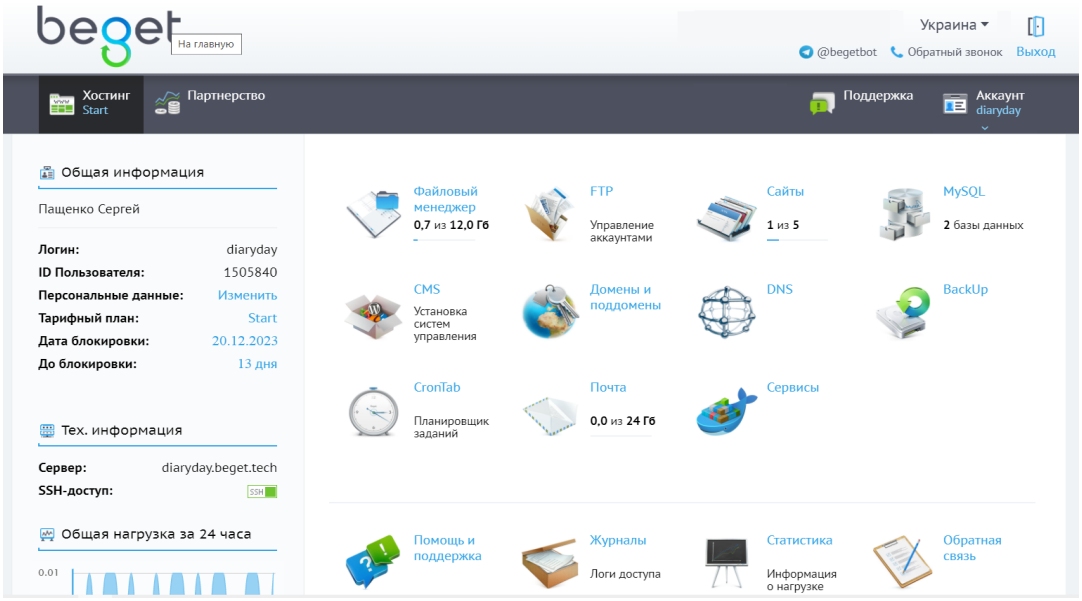



Рисунок 3.17 – Адмін панель beget

Для взаємодії з хостингом нам буде потрібно ssh клієнт. Потрібно завантажити PuTTy яка працює під операційною системою Windows. В результаті буде зазвичай виконуваний файл який після запуску дасть таке вікно (рис 3.18).

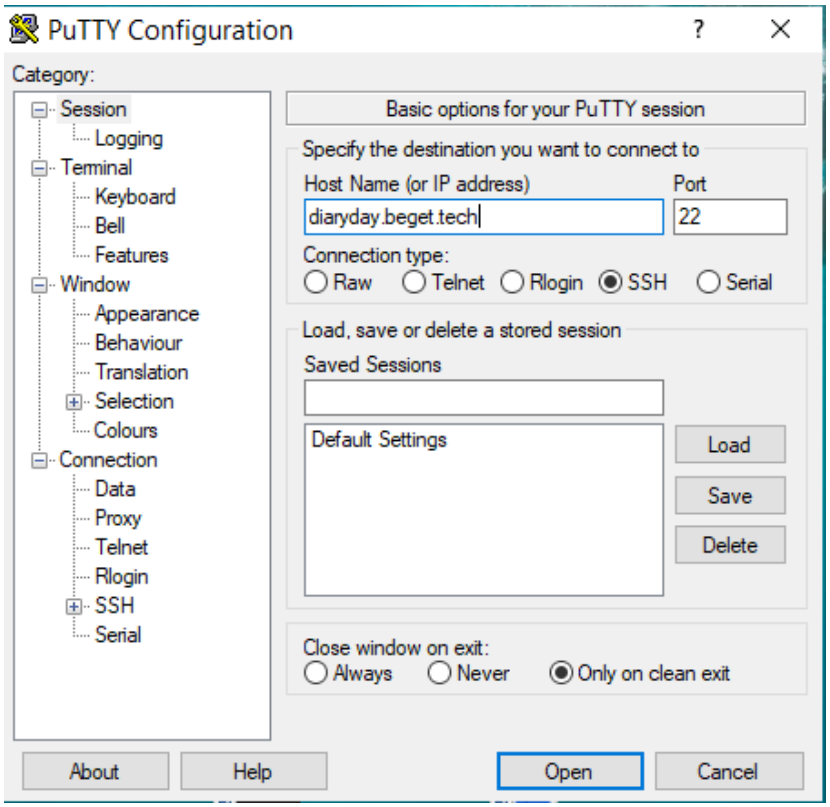



Рисунок 3.18 – З’єднання з сервером через програму PuTTy

Далі через додаток підключаємося до хостингу щоб виконувати команди на стороні сервера.

Далі потрібно ввести логін та пароль який прийде на електрону пошту.

Щоб встановити Python 3.8 на сервер потрібно виконати ряд наступних команд.

ssh localhost -p222 – ввійти в docker, щоб була можливість запускати інтерпретатор python і встановлювати необхідні пакети.

Для складання Python версії 3.7.0 і вище потрібно зібрати бібліотеку ffi. Скачати її можна наступними командами:

wget ftp://sourceware.org/pub/libffi/libffi-3.2.1.tar.gz;

tar -xf libffi-3.2.1.tar.gz – розпакуємо архів.

За допомогою configure налаштуємо все залежності, змінні, після чого буде згенерований Makefile.

Тепер запустимо процес компіляції і установки:

make -j33 && make install.

У директорії з вихідним кодом libffi-3.2.1 / x86_64-unknown-linux-gnu / include згенерує два файли: ffi.h і ffitarget.h. Їх потрібно скопіювати в ~ / .local / include командою cp.

І вже після цього можна встановити на сервер Python 3.8 в каталог ~/beget/tmp.

```
(docker) diaryday@billy:~/.beget/tmp [0] $ ls -la
итого 23528
drwx------+  3 diaryday newcustomers     4096 июн  5 22:38 .
drwx------+  4 diaryday newcustomers     4096 июн  7 15:01 ..
drwx------+ 18 diaryday newcustomers     4096 июн  5 22:39 Python-3.8.3
-rwx------+  1 diaryday newcustomers 24067487 мая 14  2020 Python-3.8.3.tgz
(docker) diaryday@billy:~/.beget/tmp [0] $ cd
(docker) diaryday@billy:~ [0] $ cd diaryday.beget.tech/
(docker) diaryday@billy:~/diaryday.beget.tech [0] $ ~/.local/bin/pip3.8 install virtualenv
```

Рисунок 3.19 – Python 3.8 встановлено

Проєкт потрібно розміщати в каталозі diaryday.beget.tech, тож всі подальші дії потрібно робити в ньому.

Якщо Python зібраний локально, для установки virtualenv вкажіть повний шлях до pip, як показано на рисунку 3.19.

Далі створюємо віртуальне оточення diary_django. Якщо створення пройшло без помилок, потрібно активувати оточення та встановити djando потрібної версії.

Щоб зв'язати проєкт Django з хостингом, потрібно створити спеціальний файл під назвою passenger_wsgi.py. В файлі потрібно прописати повний шлях до каталогу з проєктом та повний шлях до Django.

Потім потрібно створити файл .htaccess і вказати в ньому шлях до Python.

```
# -*- coding: utf-8 -*-
import os, sys
sys.path.insert(0, '/home/d/diaryday/diaryday.beget.tech/NLP_Saas_project')
sys.path.insert(1, '/home/d/diaryday/diaryday.beget.tech/diary_django/lib/python3.8/site-packages')
os.environ['DJANGO_SETTINGS_MODULE'] = 'NLP_Saas_project.settings'
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()

```

Рисунок 3.20 – Вміст файлу passenger_wsgi.py

Ще один крок, це встановити всі необхідні залежності в віртуальне оточення на хостингу. Додаткові модулі які встановлювалися для спільної роботи з проєктом. Тому щоб проєкт можна було перенести на хостинг повноцінно працездатним, всі необхідні пакети потрібно також установити. Як дізнатися, які саме пакети потрібно встановлювати і які версії брати для цього?

У терміналі на локальному комп'ютері виконав команду pip freeze > requiments.txt.

На локальному комп'ютері з'явиться файл requiments.txt з усіма залежностями, модулями які встановлені у віртуальному оточенні.

На сервері потрібно запустити команду pip install -r requirements.txt.

Додатково встановлюємо mysqlclient, щоб мати можливість працювати з базою даних MYSQL:

pip install django mysqlclient.

В каталог diaryday.beget.tech завантажуємо проєкт за допомогою git clone git@github.com:diaryDay/diaryDay.git.

В налаштуваннях проєкта встановлюємо ім’я хоста, а також налаштувати базу даних, так щоб проєкт працював з базою даних MySQL.

Які саме параметри потрібно прописувати для додатка, потрібно дивитися в документації хостингу, або запитати в техпідтримки.

```
DEBUG = True

ALLOWED_HOSTS = ['127.0.0.1']

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

```
DEBUG = False

ALLOWED_HOSTS = ['diaryday.beget.tech','www.diaryday.beget.tech']

DATABASES = {
    'default': {
#        'ENGINE': 'django.db.backends.sqlite3',
#        'NAME': BASE_DIR / 'db.sqlite3',
        'ENGINE': 'django.db.backends.mysql',
        'NAME':'diaryday_sql',
        'USER':'diaryday_sql',
        'PASSWORD': 'Admin1sql',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Рисунок 3.21 – Налаштування локального проєкта та працюючого на сервері

Залишилося декілька кроків і проєкт буде готовий до використання. Потрібно зібрати всі статичні файли, які присутні в проєкті. Це робиться стандартною командою Django:

Python manage.py collectstatic.

Щоб створити необхідні таблиці в базі даних перейдемо в командний рядок сервера і виконаємо: python manage.py migrate.

Останнє, що потрібно зробити  для коректної віддачі статичного контенту засобами Nginx, створити символьне посилання public, що вказує на public_html:

ln -s public_html public.

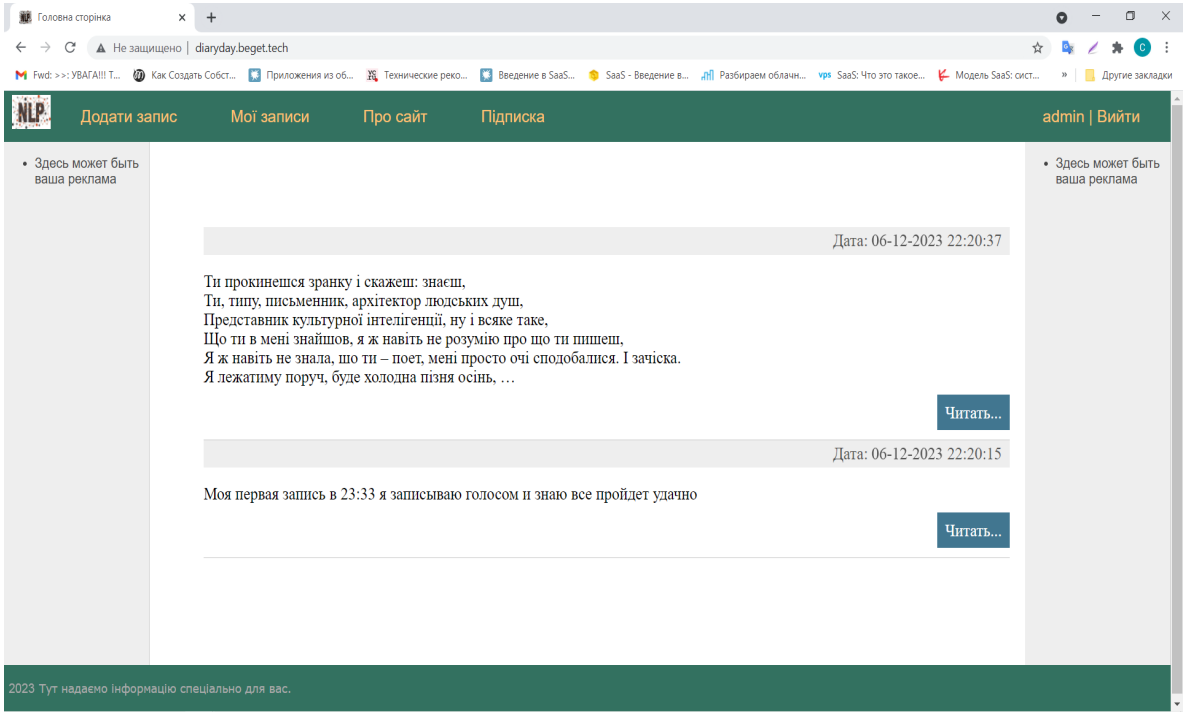



Рисунок 3.22 – Працюючий проєкт за посиланням http://diaryday.beget.tech/

### 3.4 Результати тестування програмного забезпечення

Природно виникає питання наскільки добре працює програма, тобто додаток який створювали. Тут слід відразу пояснити, що значить добре працює в даному випадку розглядаються наступні критерії:

- швидкість роботи самого додатка, як швидко віддаються сторінка;
- навантаження на СУБД тобто частота і складність запитів;
- Коректність показ даних;

Хоча останній пункт оцінюється візуально, який запит відправляємо, і що в результаті отримуємо.

Щоб відстежувати ці та деякі інші характеристики розробленої програми, в Django є досить зручний інструмент під назвою Django debug toolbar.

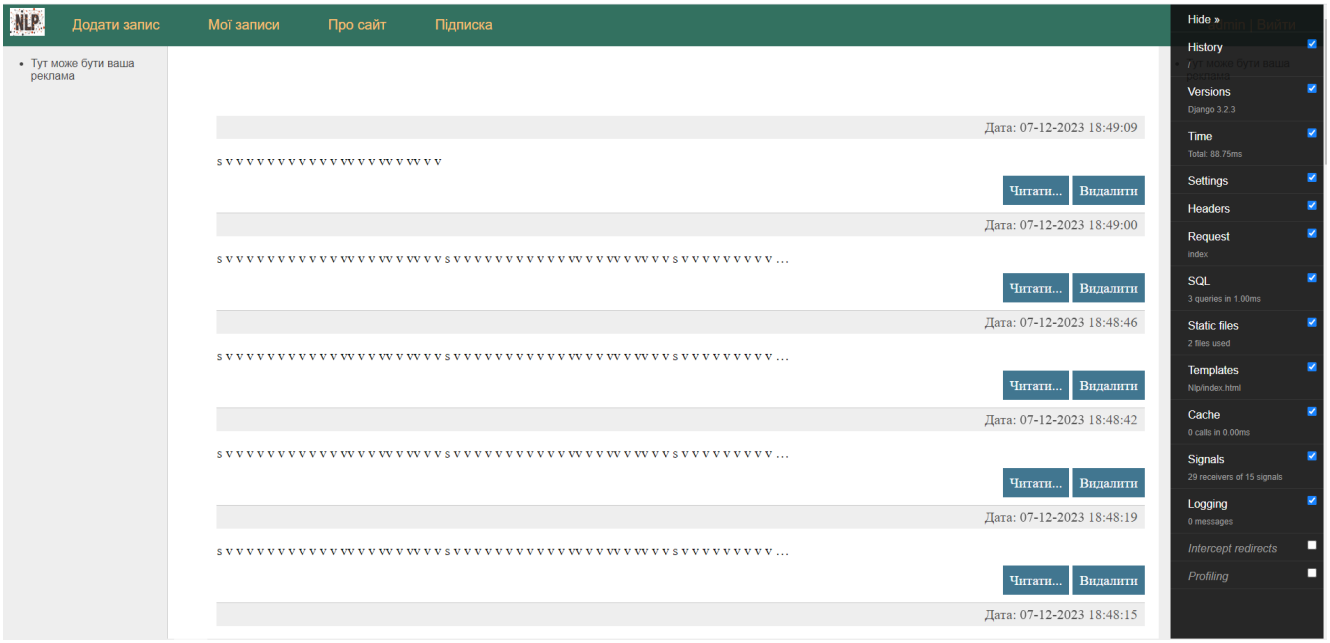
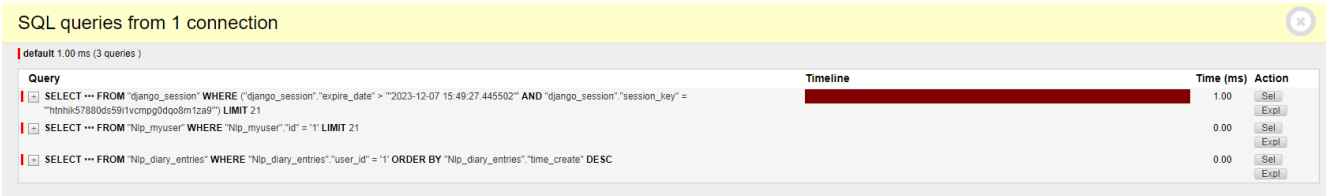



Рисунок 3.23 – Тестування головної сторінки

Праворуч з'явилася панель debug toolbar і тут бачимо версію Django 3.2.3, час формування сторінка 79.79 мілісекунд та кількість SQL запитів які виконувалися для формування цієї страніце3 sql запиту. Далі можна подивитися шаблони які використовуються для формування сторінок.

В Django debug toolbar можна побачити досить різну інформацію, про те як працює додаток.

Перше, що потрібно подивитися, це те щоб не було дублів sql запитів, і дійсно sql запити побудовані правильно, дублів немає.




На рисунок 3.24 – видно запити, що використовуються для створення сторінки

На сторінці додати запис працює все коректно, якщо користувач нічого не скаже, програма продовжує працювати. В тому випадку, коли користувач захоче записати голосом текст, а в полі вже присутні дані, текст буде додаватися в кінець уже записаного тексту.

В процесі тестування виникла проблема, якщо користувач записує голосом повідомлення, та одночасно починає редагувати зміст в полі TEXT, після обробки аудіо дані будуть записані з попереднім невідкоригованими даними.

### 3.5 Висновки по третьому розділу

У третьому розділі розроблено інформаційну технологію обробки природної мови на основі хмарних сервісів. Оптимальна структура програмного забезпечення реалізована за допомогою патерну MVC, що забезпечує чітке розділення бізнес-логіки, інтерфейсу та даних. Розроблені інструменти для роботи з базами даних і користувацьким інтерфейсом на основі Django, що забезпечують гнучкість і масштабованість проєкту. Також впроваджено функції безпеки, як CSRF, та виконано налаштування серверного середовища, що демонструє готовність системи до запуску і тестування. Випробування підтвердили працездатність системи, виявивши деякі обмеження, які були враховані для подальшого вдосконалення.

# ВИСНОВКИ

В ході виконання кваліфікаційної роботи були досліджені питання розробки SaaS-моделей з використання обробки природної мови. Проведено аналіз предметної області, актуальних технологій і програмних рішень.

Досліджено основні бізнес-моделі створення програмного продукту, а після аналізу і порівняння відповідних систем був обраний фреймворк Django для розробки програмне забезпечення.

Виконано аналіз основних методів обробки природної мови, знайдено сучасні алгоритми розпізнавання мови та проаналізовано вже існуючі моделі розпізнавання людської мови. Знайдено програмні рішення для роботи з аудіо даними. Визначено основні етапи для створення бізнес-моделі в проєктах інтелектуальних систем. Описано основні проблеми, що виникають в процесі завантаження проєкту на хостинг та використання обробки природної мови.

Для розробки програмного додатка була використана документація Django та хостингу beget для завантаження та налаштування проєкту на сервері компанії.

Розроблено програмне забезпечення на основі SaaS-моделі з використанням обробки природної мови. Функція запису тексту пов’язана з розпізнаванням голосу за допомогою моделі штучного інтелекту. Створено адмін панель для управління даними користувачів. Розпізнавання голосу реалізовано за допомогою бібліотеки Speech Recognition та використання пакету PyAudio для захоплення мікрофона. Взаємодія з сервером виконується за допомогою фреймворка Django. Візуалізація та тестування також виконувалося за допомогою Django. Створена база даних для локального проєкту використовувалася база даних sqlite3, але на хостингу використовується MySQL.

Тестування роботи програмного додатка показала його працездатність та можливість використання розпізнавання мови в SaaS-проєкті. Подана розробка функціонує без помилок і може бути використана на практиці.

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. DeepDive Tutorial. Extracting mentions of spouses from the news [Електронний ресурс] – Режим доступу: http://deepdive.stanford.edu/example-spouse

2. Аналіз тональності тексту [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Анализ_тональности_текста

3. Neural Machine Translation (seq2seq) Tutorial [Електронний ресурс] – Режим доступу: https://www.tensorflow.org/tutorials/seq2seq

4. Word2vec [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Word2vec

5. Cloudera Broadens its Collaboration with Thorn to Include Software and Services to Fight Child Sexual Exploitation [Електронний ресурс] – Режим доступу: https://www.cloudera.com/more/news-and-blogs/press-releases/2016-09-28-

6. Stanford CoreNLP [Електронний ресурс] – Режим доступу: https://stanfordnlp.github.io/CoreNLP/

7. Natural Language Toolkit [Електронний ресурс] – Режим доступу: http://www.nltk.org/

8. Creating a module for Sentiment Analysis with NLTK [Електронний ресурс] – Режим доступу: https://pythonprogramming.net/sentiment-analysis-module-nltk-tutorial/

9. TensorFlow [Електронний ресурс] – Режим доступу: https://www.tensorflow.org/

10. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin Attention Is All You Need – ARXIV, Електронна версія публікації arXiv:1706.03762, 06/2017 – PDF формат, версия 5 [Електронний ресурс] – Режим доступу: https://arxiv.org/pdf/1706.03762.pdf

11. Zhang, X. Character-level convolutional networks for text classification / Xiang Zhang, Junbo Zhao, Yann LeCun // In Advances in Neural Information Processing Systems. — 2015. — Feb. — 649 - 657 p.

12. Andrej Karpathy The Unreasonable Effectiveness of Recurrent Neural Networks – 04/2015 [Електронний ресурс] – Режим доступу: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

13. Що таке Docker [Електронний ресурс] – Режим доступу: https://docker-curriculum.com/

14. Документація docker [Електронний русурс] – Режим доступу: https://docs.docker.com/desktop/

15. Django: Build a Portfolio App [Електронний ресурс] – Режим доступу: https://realpython.com/get-started-with-django-1/

16. Поради по роботі з БД у Django [Електронний ресурс] – Режим доступу: https://devzone.org.ua/post/poradi-po-roboti-z-bd-u-django

## ДОДАТОК А.

Програмний код для сторінки реєстрації

В файлі View.py

```
menu = [{'title': "Додати запис", 'url_name': 'add'},
        {'title': "Мої записи", 'url_name': 'my'},
        {'title': "Про сайт", 'url_name': 'about'},
        {'title': "Підписка", 'url_name': 'pay'}
        ]


class RegisterUser(DataMixin, CreateView):
    form_class = RegisterUserForm
    template_name = 'Nlp/register.html'
    success_url = reverse_lazy('home')

    def get_context_data(self, *, object_list=None, **kwargs):
        context = super().get_context_data(**kwargs)
        c_def = self.get_user_context(title="Реєстрація")
        return dict(list(context.items()) + list(c_def.items()))

    def form_valid(self, form):
        user = form.save(commit=False)
        password = form.cleaned_data['password']
        #  Use set_password here
        user.set_password(password)
        user.save()
        login(self.request, user)
        return redirect('home')
```

В файлі forms.py

```
class RegisterUserForm(MyUserCreationForm):
    username = forms.CharField(label='Логін',
widget=forms.TextInput(attrs={'class': 'form-input'}))
    first_name = forms.CharField(label="Ім'я",
widget=forms.TextInput(attrs={'class': 'form-input'}))
    last_name = forms.CharField(label='Прізвище',
widget=forms.TextInput(attrs={'class': 'form-input'}))
    email = forms.EmailField(label='Email',
widget=forms.EmailInput(attrs={'class': 'form-input'}))
    password = forms.CharField(label='Пароль',
widget=forms.PasswordInput(attrs={'class': 'form-input'}))

    confirm_password = forms.CharField(label='Пароль2',
widget=forms.PasswordInput(attrs={'class': 'form-input'}, render_value=False))

    def clean_confirm_password(self):
        confirm_password = self.cleaned_data['confirm_password']
        original_password = self.cleaned_data['password']
        if original_password != confirm_password:
            raise forms.ValidationError("Password doesn't match")

        return confirm_password


    class Meta:
        model = MyUser
        fields = ('username', 'first_name', 'last_name', 'email', 'password')
```

В файлі urls.py

```
from django.urls import path

from .views import *

urlpatterns = [
    path('', index, name='home'),
    path('about/', about, name='about'),
    path('my/', index, name='my'),
    path('add/', add, name='add'),
    path('pay/', pay, name='pay'),
    path('login/', LoginUser.as_view(), name='login'),
    path('logout/', logout_user, name='logout'),
    path('register/', RegisterUser.as_view(), name='register'),
    path('post/<int:post_id>/', show_post, name='post')
]


handler404 = pageNotFound
```

Програмний код авторизації
В файлі  View.py

```
menu = [{'title': "Додати запис", 'url_name': 'add'},
        {'title': "Мої записи", 'url_name': 'my'},
        {'title': "Про сайт", 'url_name': 'about'},
        {'title': "Підписка", 'url_name': 'pay'}
        ]

class LoginUser(DataMixin, LoginView):
    form_class = LoginUserForm
    template_name = 'Nlp/login.html'
    success_url = reverse_lazy('home')

    def get_context_data(self, *, object_list=None, **kwargs):
        context = super().get_context_data(**kwargs)
        c_def = self.get_user_context(title="Вхід")
        return dict(list(context.items()) + list(c_def.items()))

    def get_success_url(self):
        return reverse_lazy('home')


def logout_user(request):
    logout(request)
    return redirect('login')
```

В файлі forms.py

```
class LoginUserForm(AuthenticationForm):
    username = forms.CharField(label='Логін',
widget=forms.TextInput(attrs={'class': 'form-input'}))
    password = forms.CharField(label='Пароль',
widget=forms.PasswordInput(attrs={'class': 'form-input'}))
```

Програмний код головної сторінки

В файлі View.py

```
menu = [{'title': "Додати запис", 'url_name': 'add'},
        {'title': "Мої записи", 'url_name': 'my'},
        {'title': "Про сайт", 'url_name': 'about'},
        {'title': "Підписка", 'url_name': 'pay'}
        ]
@login_required
def index(request):
    posts = diary_entries.objects.filter(user_id=request.user.id)
    context = {
        'posts': posts,
        'menu': menu,
        'title': 'Головна сторінка'
    }
    return render(request, 'Nlp/index.html', context=context)
```

В файлі forms.py

```
class AddPostForm(forms.ModelForm):
    class Meta:
        model = diary_entries
        fields = ['text']
        widgets = {
            'text': forms.Textarea(attrs={'cols': 60, 'rows': 10, 'class': 'form-input'}, )
        }
```

Програмний код додавання запису

В файлі View.py

```
@login_required
def add(request):
    if request.method == 'POST':

        form = AddPostForm(request.POST)
        query = " "
        if request.POST.get("sub"):
            if form.is_valid():
                text = form.cleaned_data['text']

                diary_entries(text=form.cleaned_data['text'],
user_id=request.user.id).save()

                return redirect('home')

        elif request.POST.get("rec"):
            r = sr.Recognizer()

            with sr.Microphone(device_index=1) as source:
```

```
            audio = r.listen(source)
        if audio:
            try:
                query = r.recognize_google(audio, language='ru-RU')# uk-UA
            except sr.UnknownValueError:
                print("Google Speech Recognition could not understand audio")
            except sr.RequestError as e:
                print("Could not request results from Google Speech Recognition
service; {0}".format(e))
        text = ''
        if form.is_valid():
            text = form.cleaned_data['text'] + " "
        text = text + query
        data = {'text': text}
        form = AddPostForm(initial=data)
    else:
        data = {'text': ' '}
        form = AddPostForm(initial=data)
    return render(request, 'Nlp/add.html', {'form': form, 'menu': menu, 'title':
'Запис'})
```

в файлі forms.py

```
class AddPostForm(forms.ModelForm):
    class Meta:
        model = diary_entries
        fields = ['text']
        widgets = {
            'text': forms.Textarea(attrs={'cols': 60, 'rows': 10, 'class': 'form-input'}, )
        }
```

Програмний код функції виходу користувача
В файлі View.py

```
def logout_user(request):
    logout(request)
    return redirect('login')
```

Програмний код функції показу повного тексту

В файлі View.py

```
def show_post(request, post_id):
    post = get_object_or_404(diary_entries, pk=post_id)

    context = {
        'post': post,
        'menu': menu,
    }
    return render(request, 'Nlp/post.html', context=context)
```

Програмний код сторінки про сайт

В файлі View.py

```
def about(request):
    context = {
        'menu': menu,
        'title': 'Про сайт'
    }
    return render(request, 'Nlp/about.html', context=context)
```

Програмний код сторінки тариф

В файлі View.py

```
def pay(request):
    context = {
        'menu': menu,
        'title': 'Тариф'
    }
    return render(request, 'Nlp/pay.html', context=context)
```

Програмний код адмін панелі
В файлі admin.py

```
class MyUserCreationForm(forms.ModelForm):
    """A form for creating new users.  Includes all the required fields, plus a repeated password."""

    class Meta:
        model = MyUser
        fields = ('username', 'first_name', 'last_name', 'email', 'password')


class MyUserChangeForm(forms.ModelForm):

    class Meta:
        model = MyUser
        fields = ['username',  'first_name', 'last_name', 'email', 'password']

    def clean_password(self):
        # Regardless of what the user provides, return the initial value.
        # This is done here, rather than on the field, because the field does not have access to the initial value
        return self.initial["password"]


class MyUserAdmin(admin.ModelAdmin):
    # The forms to add and change user instances
    form = MyUserChangeForm
```

```
    add_form = MyUserCreationForm

    # The fields to be used in displaying the CocoUser model.
    # These override the definitions on the base UserAdmin that reference specific
fields on auth.User.
    list_display = ('id', 'username', 'first_name', 'last_name', 'email', 'is_staff',
'password')

    fieldsets = (
        (None, {'fields': ('username', 'is_staff', 'first_name', 'last_name', 'email',
'password', 'is_superuser', 'paid_subscription', 'is_active')}),
    )

    search_fields = ('id', 'email', 'first_name', 'last_name',)
    ordering = ('last_name', 'first_name',)
    filter_horizontal = ()


class diary_entriesAdmin(admin.ModelAdmin):
    list_display = ('id', 'text', 'time_create', 'time_update', 'user')
    list_display_links = ('id', 'text', 'user')
    search_fields = ('text', 'user')


# Now register the new UserAdmin...
admin.site.register(MyUser, MyUserAdmin)

admin.site.register(diary_entries, diary_entriesAdmin)

admin.site.site_title = 'Админ-панель сайта'
admin.site.site_header = 'Админ-панель сайта'
```

Програмний код таблиць баз даних

В файлі models.py

```
        class MyUser(AbstractUser):
    paid_subscription = models.BooleanField(default=False)


class diary_entries(models.Model):
    text = models.TextField(verbose_name='Diary entry')
    time_create = models.DateTimeField(auto_now_add=True,
verbose_name='Time create')
    time_update = models.DateTimeField(auto_now=True, verbose_name='Time
```

```
update')
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE, verbose_name='User')

    def __str__(self):
        return self.text

    def get_absolute_url(self):
        return reverse('post', kwargs={'post_id': self.pk})

    class Meta:
        verbose_name = 'User record'
        ordering = ['-time_create']
```

# ДОДАТОК Б

Програмний код базового шаблону:

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
  <title>{{title}}</title>
  <link type="text/css" href="{% static 'Nlp/css/styles.css' %}" rel="stylesheet"
/>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <link rel="shortcut icon" href="{% static 'Nlp/images/main.ico' %}"
type="image/x-icon"/>
</head>
<body>
<table class="table-page" border=0 cellpadding="0" cellspacing="0">
<tr><td valign=top>
{% block mainmenu %}
        <div class="header">
                <ul id="mainmenu" class="mainmenu">
                <li class="logo"><a href=" {% url 'home' %}"><div
class="logo"></div></a></li>

{% for m in menu %}
                <li><a href="{% url m.url_name %}">{{m.title}}</a></li>

{% endfor %}
                        {% if request.user.is_authenticated %}
                        <li class="last"> {{user.username}} | <a href="{% url
'logout' %}">Вийти</a></li>
                        {% else %}
                        <li class="last"><a href="{% url 'register'
%}">Реєстрація </a>|<a href="{% url 'login' %}"> Вхід</a></li>
                        {% endif %}

                </ul>
                <div class="clear"></div>
        </div>
{% endblock mainmenu %}


<table class="table-content" border=0 cellpadding="0" cellspacing="0">
<tr>
```

```
<!-- Sidebar слева -->
  <td valign="top" class="left-chapters">
  <ul id="left-chapters">
            <li>Тут може бути ваша реклама</li>
  </ul>

<!-- Конец Sidebar'a -->
<td valign="top" class="content">
<!-- Блок контента -->
  <div class="content-text">
{% block content %}
{% endblock %}
  </div>
<!-- Конец блока контента -->
</td>
  <td valign="top" class="left-chapters">
  <ul >
            <li>Тут може бути ваша реклама</li>
  </ul>
</td></tr></table>
</td></tr>
<!-- Footer -->
<tr><td >
  <div id="footer">
            <p> 2023 Тут надаємо інформацію спеціально для вас.
  </div>
</td></tr></table><!-- Конец footer'a и страницы -->
</body>
</html>
```

Шаблон  додавання запису:

```
{% extends 'Nlp/base.html' %}
{% load widget_tweaks %}
{% block content %}
<h2>{{title}}</h2>

<form action="{% url 'add' %}" method="post">
   {% csrf_token %}
<table><tr><td>
          {{form.text}}
</td></tr>
   <tr><td>
   <button type="submit" name="sub" value="Додати">Додати</button>
     <button type="submit" name="rec" value="Записати">Записати</button>
```

```
  </td> </tr></table>
</form>
{% endblock %}
```

Шаблон головної сторінки:

```
{% extends 'Nlp/base.html' %}

{% block content %}
<ul class="list-articles">
  {% for p in posts %}
  <li><div class="article-panel">
  <p class="last">Дата: {{p.time_update|date:"d-m-Y H:i:s"}}</p>
</div>
  {{p.text|linebreaks|truncatewords:50}}
                        <div class="clear"></div>
            <p class="link-read-post"><a href="{{ p.get_absolute_url
}}">Читати...</a>
                        <a href="{{ p.get_delete }}">Видалити</a> </p>
                        </li>
  {% endfor %}
</ul>
{% endblock %}
```

Шаблон сторінки авторизації:

```
{% extends 'Nlp/base.html' %}

{% block content %}
<h2>{{title}}</h2>

<form method="post">
   {% csrf_token %}
  {% for f in form %}
<p><label class="form-label" for="{{ f.id_for_label }}">{{f.label}}: </label>{{
f }}</p>
<div class="form-error">{{ f.errors }}</div>
{% endfor %}


   <button type="submit">Вхід</button>

</form>

{% endblock %}
```

Шаблон сторінки реєстрації:

```
{% extends 'Nlp/base.html' %}

{% block content %}
<h2>{{title}}</h2>

<form method="post">
    {% csrf_token %}
{% for f in form %}
<p><label class="form-label" >{{f.label}}: </label>{{ f }}</p>
<div class="form-error">{{ f.errors }}</div>
{% endfor %}
   <button type="submit">Реєстрація</button>
</form>

{% endblock %}
```

# ДОДАТОК В

## Відомість матеріалів кваліфікаційної роботи

| № з/п | Позначення | Найменування | Кільк. аркушів | Примітки |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | Документація | | |
| 3 | | | | |
| 4 | **ІТКІ.КР.22.7.ДА.ПЗ** | Пояснювальна записка | 95 | Формат А4 |
| 5 | | | | |
| 6 | | Презентація | | |
| 7 | | | | |
| 8 | | Диск CD-R з презентацією | 1 | Диск CD-R |
| 9 | | | | |

| Зм. | Ар-куш | № докум. | Підпис | Дата | **ІТКІ.КР.22.7.ДА.ПЗ.** | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Розроб. | | | | | **Матеріали кваліфікаційної роботи** | Літ. | | Аркуш | Аркушів |
| Керівник | | Соколова Н.О. | | | | Н | | 1 | 1 |
| Рецензент | | Клименко А.В | | | | НТУ «ДП», 12; 126-22м-1 | | | |
| Н.контр. | | Коротенко Г.М. | | | | | | | |
| Зав. каф. | | Гнатушенко В.В. | | | | | | | |