МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Центр заочної, дистанційної та вечірньої форм навчання

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

22 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 – Комп'ютерних наук,

освітньо-професійної програми «Інформаційно-комунікаційні технології»

на тему: «Інформаційно-комунікаційна технологія підтримки навчального процесу в особливих умовах»

здобувачки групи ІК.мз-11с Шутилєвої Ольги Вікторівни

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Ольга ШУТИЛЄВА

(підпис)

Керівник,

доцент кафедри комп'ютерних наук,

к.ф.-м.н., доцент

Сергій ШАПОВАЛОВ _____

(підпис)

Суми – 2023

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр
зі спеціальності 122 – Комп'ютерних наук, освітньо-професійної програми
«Інформаційно-комунікаційні технології»
здобувачки групи ІК.мз-11с Шутилєвої Ольги Вікторівни

1. Тема роботи: «Інформаційно-комунікаційна технологія підтримки навчального процесу в особливих умовах»
затверджую наказом по СумДУ від «01» грудня 2023 р. № 1389-VI
2. Термін здачі здобувачем кваліфікаційної роботи до 22 грудня 2023 року
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми, постановка й формування завдань для роботи за обраною тематикою. 2) Огляд технологій, що використовуються для розробки інформаційно-комунікаційних технологій. 3) Розробка веб-застосунку з інтеграцією чат-боту. 4) Аналіз виконаної роботи.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» листопада 2023 р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми, постановка й формування завдань для роботи за обраною тематикою*		
2	*Огляд технологій, що використовуються для розробки інформаційно-комунікаційних технологій*		
3	*Практична реалізація*		
4	*Аналіз виконаної роботи*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 46 стор., 10 рис., 3 табл., 1 додаток, 21 джерел.

Обґрунтування актуальності теми роботи – актуальність обраної тематики для дослідження обумовлена необхідністю автоматизації та оптимізації процесів організації комунікації студентів та викладачів в особливих умовах навчального процесу.

Об’єкт дослідження – аналіз технічних рішень проєктування чат-ботів.

Мета роботи – провести літературний огляд існуючих платформ для чат-ботів, проаналізувати та обґрунтувати вибір технологій для практичної реалізації.

Методи дослідження – системний аналіз, компонентне проєктування, UML-проєктування.

Результати – на основі проведеного літературного огляду було реалізовано веб-застосунок, який підтримує розподілення на дві ролі та його інтеграція з Telegram чат-ботом.

ВЕБ-ЗАСТОСУНОК, ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНА ТЕХНОЛОГІЯ, МЕСЕНДЖЕР, ЧАТ-БОТ, REACT.

ЗМІСТ

ВСТУП

Актуальність. На сьогоднішній день існує безліч варіантів отримати необхідну інформацію абсолютно за будь-яким напрямком у залежності від інтересів, віку, сфери діяльності тощо. Для цього користувачі використовують усі можливі пристрої з виходом в Інтернет. Проте, коли розпочалась пандемія COVID-19, а потім повномасштабного вторгнення в Україну викладачі університетів зіштовхнулись з тією проблемою, що студенти не отримують вчасно необхідну інформацію, яка стосується навчального процесу. Проблеми у цьому випадку були пов’язані з тим, що зворотній зв’язок з викладачами, навіть у межах кафедри, відрізнявся: електронна пошта, месенджери, платформа mix.sumdu.edu.ua тощо.

Об’єкт дослідження: аналіз технічних рішень проєктування чат-ботів.

Предмет дослідження: інформаційно-комунікаційна технологія підтримки навчального процесу.

Гіпотеза. Несистематичний підхід до способів ведення комунікації між студентами та викладачами університету призвели до неефективного отримання студентами необхідної навчальної інформації, і впровадження інтегрованої та однорідної платформи має на меті покращити цю ситуацію та певною мірою підвищити якість навчання. Поширення та популярність чат-ботів можна пояснити тим, що вони мають здатність поєднувати функціонал кількох додатків у одному місці.

Новизна. Розробка веб-застосунку, який підтримує оптимізує процес навчання в складних умовах, дозволяє оптимізувати ряд процесів з вдало поєднаними методами і технологіями розробки.

Структура. Кваліфікаційна робота складається з вступу, аналітичного огляду, вибору методів рішення, програмної реалізацій, висновку, списку використаних джерел та додатку.

1. ІНФОРМАЦІНИЙ ОГЛЯД ПЛАТФОРМ ЧАТ-БОТІВ

1.1 Класифікація чат-ботів

Чат-бот – це програма, яка має здатність імітувати спілкування з користувачем на основі попередньо визначеного алгоритму та / або за допомогою штучного елементу, а також налаштована на автоматичне виконання необхідних дій. Вони використовуються в додатках, соціальних мережах, месенджерах або сайтах [1].

Месенджер – це програма або служба, яка призначена для обміну приватними повідомленнями у режимі реального часу між користувачами через мережу Інтернет. Його основними можливостями є відправка текстових повідомлень, медіаконтенту, а також відправка голосових повідомлень, здійснення дзвінків. Основною перевагою месенджерів є можливість одночасного підключення великої кількості користувачів та миттєвої комунікації між ними з будь-яких пристроїв та територій [2].

Згідно досліджень [3] станом на жовтень 2023 року медіа платформа Telegram входить до топ-10 найпопулярніших соціальних мереж світу і має 800 мільйонів активних користувачів. Стаття [4] свідчить про те що в Україні на кінець 2023 року Telegram продовжує залишатися основною соціальною мережею для спілкування та споживання новин, демонструючи значне зростання в обох сферах до 72%. Наразі месенджери щоденно використовуються для розв’язання різного роду завдань, які вже давно вийшли за рамки простого обміну повідомлень між користувачами: передавання показників лічильників, оплата за різноманітні послуги, клієнтська взаємодія з компаніями, пошук новин тощо [5, 6].

Розробка чат-ботів є одним з перспективних напрямків розвитку штучного інтелекту та інструментів автоматизації. Вони виконують широкий спектр завдань у сферах як персонального використання, так і в бізнесі.

Персональні або особисті чат-боти виступають у ролі цифрових помічників користувачів, допомагаючи організувати власний розпорядок дня, поліпшити рутинні завдання та підвищити продуктивність. Для цього чат-боти можуть надсилати сповіщення, керувати розкладом подій в календарі, виконувати підбір мультимедійного контенту на основі вподобань та багато іншого.

Корпоративні або ділові чат-боти призначені для підтримки, організації та оптимізації бізнес-процесів компаній, вони дозволяють автоматизувати рутинну роботу, підвищити лояльність клієнтів та конвертувати все це у дохід.

Сфера застосування чат-ботів швидко еволюціонує, тому потребує постійного аналізу та вдосконалення підходів. Важливо правильно класифікувати чат-боти за критеріями користувачів, інтерфейсів, функціональності, доступу та архітектури для обґрунтування технологічних рішень.

Існують різні способи взаємодії користувача з чат-ботом: за допомогою кнопок, текстових повідомлень та голосу. Кнопковий чат-бот має інтерфейс, що передбачає набір попередньо визначених опцій у вигляді кнопок, які користувач обирає для «розмови». Для взаємодії користувачу пропонується на вибір категорії, питання або пропозиції і в залежності від того що зацікавить користувача, він натискає на необхідну кнопку. Такий підхід до створення боту значною мірою спрощує його логіку, проте обмежує гнучкість діалогу рамками закладених сценаріїв.

Текстова взаємодія наближена до природної комунікації людини з людиною. Чат-бот підключений у цьому випадку до бази знань, проте можливості таких ботів залежать від алгоритмів розпізнавання ключових слів та обсягу знань, закладених розробником. Теоретично такий тип має більше функціональності ніж кнопковий, проте все залежить від розробника.

Управління голосом відноситься до типу розумних чат-ботів. Він доволі складний у проєктуванні, тому що вимагає перетворення голосових команд у текст, їх аналіз та генерацію відповіді. Цей підхід потребує потужностей

штучного інтелекту і машинного навчання. Комбінування підходів дозволяє розширити функціонал ботів [1, 7].

За формами доступу до чат-ботів існують три основні моделі:

– групове використання – чат-бот обслуговує одночасно декількох користувачів, що дозволяє масштабувати його можливості;

– індивідуальна підписка – персональний помічник з індивідуальними налаштуваннями та функціями для певного користувача;

– убудовані чат-боти – можуть бути викликані спеціальною командою в будь-якому діалозі для виконання конкретного запиту. Результат запиту відразу відображається в чаті [8].

За принципом функціонування чат-боти можна поділити на дві основні категорії: примітивні або ще їх називають шаблонні та здатні до навчання [1].

Примітивні працюють виключно у межах, попередньо закладеної розробником, жорсткої логіки; діалоги обмежуються наперед визначеними сценаріями (скриптами) у формі дерев рішень. Кількість відповідей для користувача для такого рішення є обмеженою. Такі боти не здатні виходити за межі своїх шаблонів і не можуть самостійно вдосконалюватись. Основна перевага – простота розробки та передбачуваність поведінки.

«Розумні» чат-боти працюють на базі методів штучного інтелекту – машинного навчання та обробки природної мови. Вони здатні аналізувати діалог з користувачем, виділяти за сенсом мовні конструкції та вдосконалювати свої комунікативні навички відповідаючи на запити. Такі боти значно складніші у розробці, проте їх можливості та гнучкість набагато вищі.

Для створення логіки роботи чат-ботів використовується безліч мов програмування, а найпоширенішими є Node.js, C++, Python та Java завдяки їх ефективності у розробці серверних додатків.

Розглянемо типову схему взаємодії чат-бота з платформою месенджера (рис.1.1). Спочатку користувач відправляє повідомлення через інтерфейс месенджера. Дане повідомлення надходить на сервер, де виконується

програмний код чат-бота. Він аналізує запит від користувача, формує відповідь та передає її назад у месенджер користувачеві.

Така архітектура дозволяє абстрагуватися від конкретної платформи месенджера, реалізовуючи основну логіку чат-бота мовою програмування на вибір розробника. Це спрощує подальше масштабування та інтеграцію бота з різними каналами комунікації.

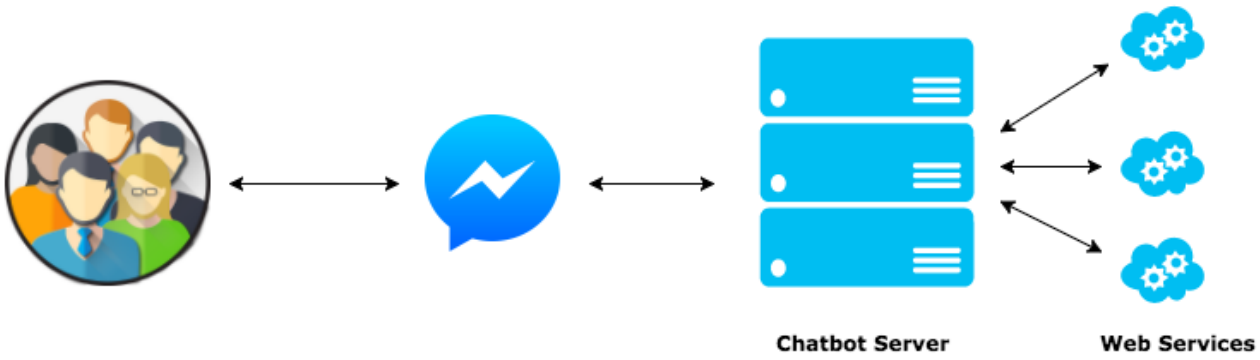


Рисунок 1.1 – Модель взаємодії чат-бота з месенджером [9]

Існують два основні механізми для організації взаємодії між платформою месенджера (клієнтська частина) та логіка безпосередньо самого чат-бота (серверна частина) – з використанням механізму вебхук та пулінг. На рисунку 1.2 представлено візуальне порівняння цих механізмів.

Вебхук передбачає, що клієнтська сторона самостійно ініціює надсилання HTTP POST запиту запити до серверу при настанні певних подій, найчастіше передаючи дані у форматі JSON. Такий підхід потребує налаштування автентифікації, наприклад, за допомогою білого листа IP-адрес, унікальних токенів, TLS шифрування чи SSL сертифікатів.

Пулінг полягає в тому, що сервер ініціює запити до клієнтської частини з певною періодичністю, очікуючи у відповідь дані про настання події. Автентифікація тут також необхідна, часто за допомогою унікальних токенів бота [10].

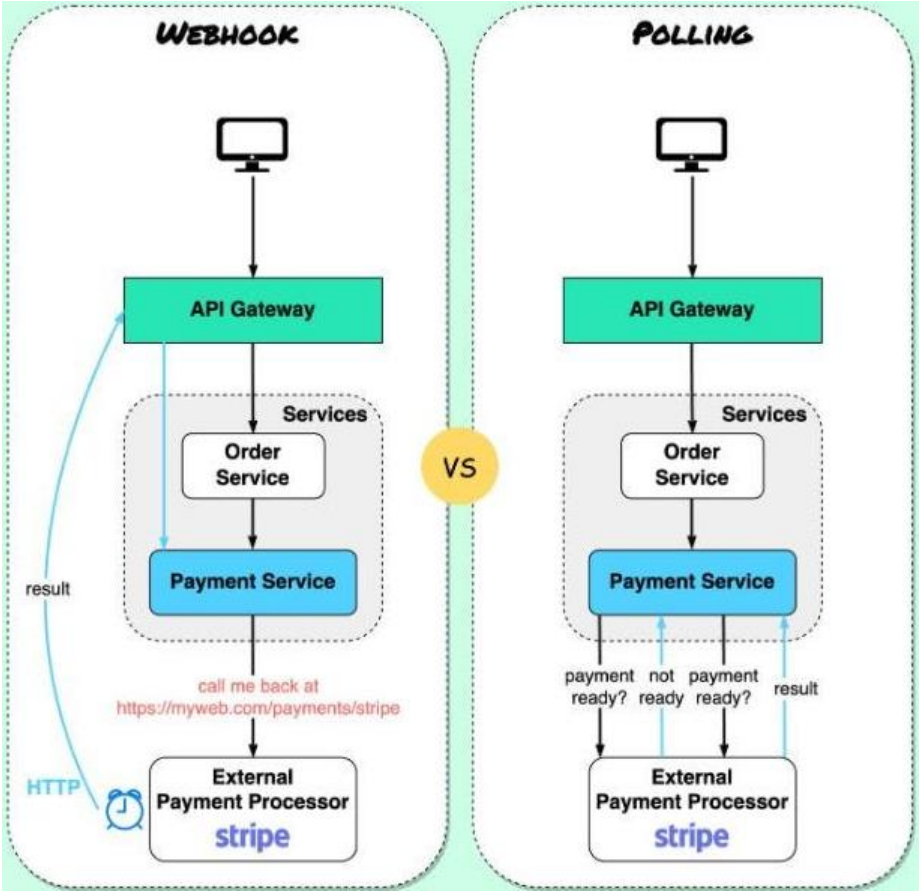


Рисунок 1.2. – Візуалізація механізмів вебхук та пулінг [10]

Для виконання кваліфікаційної роботи необхідно проаналізувати і обрати оптимальний месенджер, який підтримує платформу чат-бота. Для аналізу було обрано найпопулярніші: Telegram, Facebook та Viber. Основні критерії для аналізу наступні: оплата за користування, механізми зв'язку, протоколи передачі даних, формати передачі даних, основні функціональні можливості (листування, опитування, передача файлів, створення груп). Результати порівняння наведені у таблиці 1.1.

Таблиця 1.1 – Порівняння критеріїв основних месенджерів

Критерії	**Telegram**	**Facebook**	**Viber**
Оплата за користування	Умовно безоплатна		
Механізми зв'язку	вебхук, пулінг	вебхук	вебхук
Протоколи передачі даних	HTTPS		

Формати передачі даних	JSON, URL query, multipart/ form-data	JSON, multipart/ form-data	JSON, multipart/for m-data
Функціональні можливості	Листування, опитування, передача файлів, створення груп	Листування, опитування, передача файлів, створення груп	Листування, передача файлів, створення груп

Зважаючи на проведений аналізу месенджерів, найбільш оптимальним рішенням для реалізації мети проєкту є використання месенджера Telegram з наступних причин:

– він має повністю безкоштовну базову версію;

– підтримує механізм взаємодії пулінгу, що спрощує першочергову розробку та налагодження системи;

– функціональні можливості Telegram дозволять реалізувати всі необхідні вимоги до системи, а в подальшому – розширювати її за рахунок нових опцій та/або інтеграцій;

– останній але не менш важливий фактор – популярність серед студентів.

1.2 Функціонування чат-боту в Telegram

Розглянемо чат-бот у контексті месенджера Telegram. Фактично це є сторонній додаток, який працює на базі платформи Telegram. Користувачі мають змогу взаємодіяти з ботом шляхом надсилання повідомлень, команд або запитів. Для створення та управління чат-ботом використовується спеціальне API, яке використовує HTTPS протокол. Основні можливості, які дає цей API:

– відправка сповіщень, новин та іншого персоналізованого контенту користувачам;

– інтеграція з зовнішніми сервісами для розширення функціоналу – пошта, YouTube, Github тощо;

– приймання платежів від користувачів бота;

– створення корисних інструментів – перекладачів, нагадування, ботів для форматування текстів.

З точки зору месенджера Telegram, чат-бот – це спеціальний акаунт для взаємодії з яким:

– користувач може відкрити особисту розмову з ботом або додати його в груповий чат;

– через інлайн-команду ввести @ім'я_бота та отримати результат його роботи у будь-якому типі чату.

Є ряд особливостей на які також варто звернути увагу і врахувати:

– комунікацію завжди починає користувач, оскільки бот не може самостійно розпочати розмову. Це запобігає можливому спаму;

– існують обмеження Telegram на об'єм хмарного сховища повідомлень для ботів. Тому потрібно продумати і за необхідності реалізовувати власне локальне зберігання даних;

– імена облікових записів чат-ботів в завжди закінчуються на «bot» для їх ідентифікації та відмінності від звичайних акаунтів.

Розглянемо більш детально принцип роботи Telegram Bot API [11]. Під час розробки бота створюється унікальний токен авторизації (наприклад, «5df598f2cf7623ad:e19b6edab51abb0c3e270775d9415260»), який включається до URL для кожного запиту. Усі запити до Telegram Bot API мають використовувати HTTPS і здійснюватися на адресу «https://api.telegram.org/bot<token>/METHOD_NAME». Система підтримує як GET, так і POST HTTP запити. Можливі наступні способи передачі параметрів у запиті:

– через рядок запиту URL;

– використовуючи Application/x-www-form-urlencoded;

– застосувуючи Application/json;

– використовуючи Multipart/form-data.

Ці методи можуть застосовувати метод кодування символів, відомий як URL encoding або percent encoding. Цей процес використовується для вирішення проблем, пов'язаних із використанням символів, які не дозволені в URL (наприклад, пробіли). Суть цього механізму полягає у заміні 8-бітних символів, що не допускаються, на відповідні дозволені символи. Таким чином, це дозволяє безпечно передавати символи в URL, забезпечуючи їх сумісність та коректне відображення у різних системах і програмах. Наприклад, у випадку URL encoding символ пробілу в URL зазвичай замінюється на %20. Таким чином, якщо необхідно включити фразу «hello world» у URL, вона буде закодована як hello%20world.

Коли виконується запит до Telegram Bot API, у відповідь повертається об'єкт JSON, який завжди включає булеве поле «OK» та поле «description» з текстом. У випадку успіху запиту, «OK» буде «true»; у разі помилки – «false». Для помилок, у відповіді також з'являється поле «description» з деталями помилки та «error_code» з її кодом. Важливо відзначити, що всі методи Telegram Bot API не залежать від регістру літер, а передача даних відбувається у форматі UTF-8.

Розглядаючи функціональні можливості та обмеження Telegram-ботів, будемо вказувати короткий опис функції і те як саме вона працює з точки зору Telegram Bot API, включаючи використовувані методи API, та вказувати на обмеження, якщо такі є.

Важливо розуміти, що взаємодія з Telegram Bot API поділяється на дві основні категорії: отримання оновлень та виконання методів API. У першому випадку Telegram надсилає боту повідомлення про дії користувача, у другому – бот відправляє повідомлення користувачу. Отримання оновлень у Telegram Bot API здійснюється через об'єкт Update. Ці оновлення являють собою події, що виникають, коли користувачі взаємодіють із чат-ботом. Важливі поля об'єкту Update представлені у таблиці 1.2.

Основна функція месенджерів є можливість обміну повідомленнями між користувачами. Це листування реалізується за допомогою різних типів

повідомлень – текстових, графічних, відео, аудіо тощо. Для підтримки листування в API месенджерів використовується базовий об’єкт Message з полями для збереження різних типів контенту, метаданих та інформації про відправника/отримувача. Методи API дозволяють створювати нові повідомлення, редагувати та видаляти існуючі, а також отримувати історію листування з іншими користувачами. Основні поля об’єкту Message представлені у таблиці 1.3.

Таблиця 1.2 – Основні поля об’єкту Update [11]

Поле	Тип	Опис
update_id	Integer	Унікальний ідентифікатор оновлення. Якщо нових оновлень не було протягом щонайменше тижня, ідентифікатор наступного оновлення буде обрано випадково, а не послідовно.
message	Message	Опційно. Нове вхідне повідомлення будь-якого типу – текст, фото, стікер тощо.
edited_message	Message	Опційно. Оновлена версія повідомлення, яке відоме боту і було відредаговане.
callback_query	CallbackQuery	Опційно. Новий вхідний запит зворотного виклику (результат натискання на кнопку клавіатури).

Таблиця 1.3 – Основні поля об’єкту Message [11]

Поле	Тип	Опис
message_id	Integer	Унікальний ідентифікатор повідомлення в межах цього чату.
date	Integer	Дата надсилання повідомлення у форматі Unix часу.

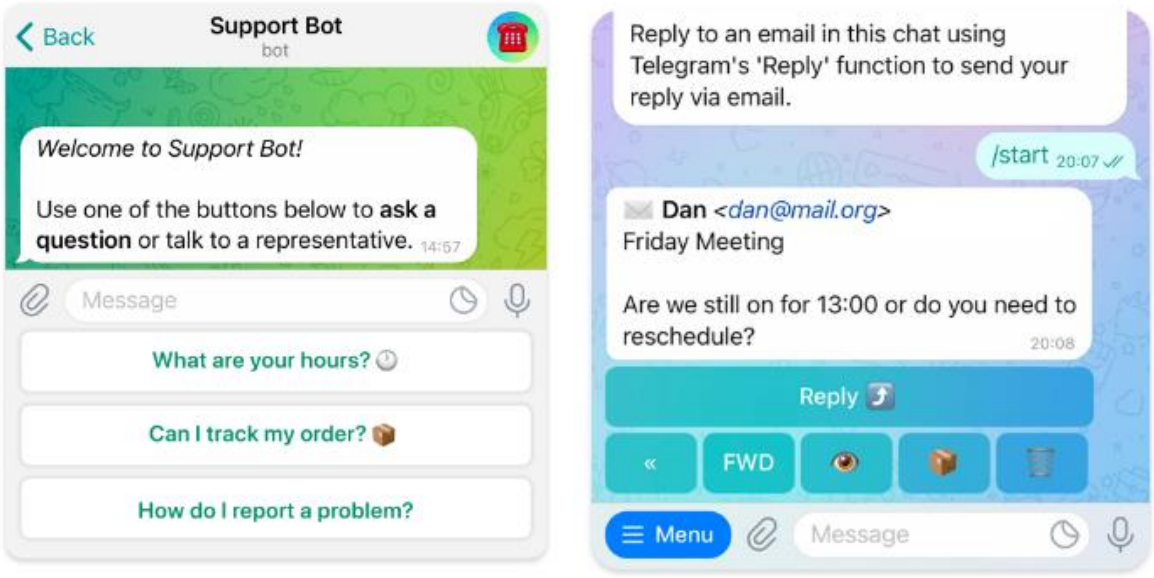
chat	Chat	Чат, до якого належить повідомлення.
edit_date	Integer	Опційно. Дата останнього редагування повідомлення у форматі Unix часу.
caption	String	Опційно. Підпис до анімації, аудіо, документу, фотографії, відео або голосового повідомлення.
text	String	Опційно. Для текстових повідомлень, фактичний текст повідомлення у форматі UTF-8.
entities	Array of MessageEntity	Опційно. Для текстових повідомлень, спеціальні елементи, такі як імена користувачів, URL-адреси, команди ботів тощо, які зустрічаються у тексті.

Telegram надає можливість додавати інтерактивні клавіатури з кнопками до повідомлень. Це розширює можливості взаємодії користувача і дозволяє створювати меню, посилання, опитування тощо в чат-ботах. Рекомендованою практикою є оновлення повідомлення з клавіатурою замість надсилання нового при кожній зміні. Це оптимізує інтерфейс, не засмічуючи чат численними повідомленнями. Telegram дозволяє ботам надсилати спеціальні клавіатури користувачам для полегшення взаємодії. Ці клавіатури можуть бути двох типів: звичайні (рис.1.3 а) та вбудовані (inline) (рис.1.3 б).

При створенні клавіатури, кожній кнопці задається унікальний ідентифікатор у параметрі callback_data. Коли користувач натискає певну кнопку, Telegram надсилає цей ідентифікатор на сервер у вигляді об’єкта CallbackQuery. Таким чином бот отримує інформацію, яку саме дію обрав користувач.

На основі callback_data розробник бота може з легкістю ідентифікувати запит від користувача та виконати відповідну команду зі свого API. Саме за

таким принципом формується система взаємодії між Telegram чат-ботом та користувачем за допомогою користувацьких (кастомних) клавіатур.



а б

Рисунок 1.3 – Типи клавіатури чат-бота: а – звичайні, б – вбудовані [11]

Тепер розглянемо таке поняття як команди – це гнучкий спосіб взаємодії користувача з Telegram ботом. Вони починаються зі слеша «/» та мають довжину до 32 символи. Інформація про команди міститься у полі MessageEntity об’єкта Message, який бот отримує від Telegram API. При введенні «/» користувачеві виводиться на екран список доступних команд цього бота. Команди виділяються посиланням і їх можна легко повторно виконати новим натисканням. Будь-який бот має підтримувати базові команди, наприклад, такі як «/start», «/help», «/settings». Команда «/start» обов’язкова для ініціалізації діалогу.

Розробник бота самостійно формує список команд та логіку їх обробки через API бота. Таким чином він визначає які дії буде виконувати чат-бот у відповідь на ту чи іншу команду від користувача. Приклад інтерфейсу зі списком команд показано на рисунку 1.4.

Telegram дозволяє форматувати текст повідомлень, які бот надсилає користувачам. Щоб задати формат тексту, потрібно передати параметр parse_mode при виклику методів надсилання повідомлень в Telegram Bot API. Підтримуються два формати:

– Markdown – дозволяє робити текст жирним, курсивом, додавати заголовки, посилання тощо за допомогою спеціального синтаксису.

– HTML – дає змогу форматувати текст з використанням HTML-тегів.

Приклад коду:

```
bot.send_message(
    chat_id=12345,
    text="<b>Bold text</b>",
    parse_mode='html'
)
```

Рисунок 1.4 – Приклад інтерфейсу чат-бота зі списком команд [11]

Зазначений функціонал буде використано у подальшій розробці.

1.3 Постановка задачі

Інформаційно-комунікаційна система – це комплекс програмних, технічних і організаційних засобів для збору, обробки, зберігання та поширення інформації.

Будь-яка інформаційна система вирішує такі основні задачі:

– передача даних між компонентами системи;

– аналіз та обробка даних для прийняття рішень;

– зберігання інформації.

Відповідно до цих задач, основу інформаційної системи складають:

– засоби передачі даних;

– засоби аналізу та обробки даних;

– системи зберігання даних.

При проєктуванні програмного забезпечення для такої системи потрібно:

– виконати літературний огляд;

– сформулювати вимоги;

– розробити специфікацію;

– здійснити програмну реалізацію;

– протестувати систему.

Інтерфейс інформаційно-комунікаційної системи буде розроблено у вигляді веб-додатку з React для спрощення підтримки на різних платформах.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Аналіз методів та вибір архітектури

При проєктуванні програмного забезпечення (ПЗ) ключовим етапом є визначення архітектури. Архітектура ПЗ полягає у структуруванні програмного забезпечення на незалежні та слабкозв’язані частини, визначенні зв’язків між ними та описі процесів передачі даних. Це робиться для забезпечення відповідності архітектури вимогам проєкту. Вибір архітектури базується на сформованих функціональних вимогах, і тому для розробки можуть бути розглянуті наступні архітектурні шаблони та стилі:

– клієнт-серверна архітектура;
– розділення на front-end та back-end;
– монолітна або мікросервісна архітектура;
– архітектурний стиль MVC.

У клієнт-серверній архітектурі [12] відбувається взаємодія між серверами, які обробляють запити, та клієнтами, які ці запити ініціюють, через мережу.

У контексті системи управління чат-ботом, взаємодія відбувається між трьома ключовими сторонами:

– Telegram Bot API;
– сервером з основною логікою роботи чат-бота;
– клієнтським інтерфейсом.

Тут сервери Telegram діють як серверна сторона щодо сервера логіки чат-бота, який в свою чергу служить сервером для клієнтського веб-інтерфейсу, а клієнтський веб-інтерфейс та сервери логіки чат-бота виступають як клієнти відносно один одного.

Клієнтський веб-інтерфейс керує представленням інформації (демонстрація даних та прийняття користувацьких команд), тоді як

мікросервіси відповідають за обробку та управління даними (запис і логічна обробка інформації).

У сфері розробки об’ємних або модульних систем вибір структури є вирішальним. Два домінуючих підходи до архітектури ПЗ є монолітний та мікросервісний [13-15]. Монолітна архітектура збирає всі компоненти програми в одному процесі операційної системи, які взаємодіють через внутрішні інтерфейси. Переваги монолітної архітектури включають:

– зручність розробки, оскільки багато інтегрованих середовищ розробки оптимізовані для створення монолітних додатків;

– простота запуску;

– прямолінійне масштабування шляхом розгортання кількох інстансів за допомогою балансувальника навантаження.

Однак, зі збільшенням складності та розмірів програмного забезпечення, монолітна архітектура може виявитися проблематичною:

– ускладнення підтримки та розуміння великої кодової бази, а також внесення змін;

– зниження продуктивності інтегрованих середовищ розробки та часу запуску програми;

– обмежені можливості для частих оновлень програми через взаємозалежність компонентів та необхідність перезапуску всієї системи при оновленні одного з них;

– помилка в одному компоненті може призвести до збою всієї програми;

– можливість масштабування тільки в одному напрямку;

– важкість зміни використовуваних технологій та їх версій.

Мікросервісна архітектура – це структура програмного забезпечення, де кожен компонент виконує свою функцію і є мінімально залежним від інших, працюючи в окремих процесах операційної системи [14]. Компоненти можуть бути розміщені на різних машинах, і взаємодія між ними відбувається через протоколи віддаленого виклику процедур (RPI) або через системи обміну повідомленнями.

Переваги мікросервісної архітектури включають:

– зрозумілість, простота розробки, тестування, підтримки, оновлення та розгортання завдяки невеликим і незалежним компонентам;

– збільшення швидкості розробки за рахунок вищої продуктивності інтегрованих середовищ розробки та швидкого завантаження окремих мікросервісів;

– підвищення надійності системи, оскільки проблеми в одному компоненті не впливають на інші частини системи;

– гнучкість у виборі та заміні технологій та фреймворків.

Однак, мікросервісна архітектура також має свої недоліки:

– необхідність розробки механізмів для взаємодії та обміну даними між мікросервісами;

– складність синхронізації даних між різними компонентами;

– ускладнення процесу розгортання.

Model-View-Controller (MVC) – це шаблон проєктування програмного забезпечення, що розділяє класичні функції відображення, збереження та обробки даних на три основні взаємозв’язані компоненти [16].

Модель відповідає за зберігання та управління даними, включно з їх станом та поведінкою, і змінюється залежно від взаємодії з користувачем через контролер. Вид є компонентом, що забезпечує представлення даних користувачеві, відображаючи поточний стан моделі. Вид оновлюється автоматично, коли змінюється стан моделі. Контролер обробляє вхідні дані від користувача, зазвичай через елементи інтерфейсу, такі як кнопки чи перемикачі, і передає команди моделі, якщо потрібно змінити стан чи відповідь виду.

У контексті розробки кваліфікаційної роботи та аналізу недоліків монолітної архітектури, було визначено, що оптимальним буде використання мікросервісної архітектури, яка пропонує такі переваги:

– можливість незалежної розробки кожного компонента системи;

– забезпечення більш простого та ефективного масштабування;

– гнучкість у застосуванні сучасних технологій та можливість їх легкої заміни при необхідності.

2.2 Опис API та бази даних мікросервісів

Визначення API служб є наступним кроком у розробці системи мікрослужб. Це включає в себе визначення операцій (команд) та подій для кожної служби. Службові команди виконують дві цілі: одні призначені для виконання операцій, які викликаються зовнішніми клієнтами або іншими службами, тоді як інші виключно підтримують взаємодію між службами. Події, як правило, використовуються для комунікації між службами для збереження синхронізації та консистентності даних (дані в БД повинні бути коректними, актуальними і сумісними в будь-який момент часу) або для сповіщення зовнішніх клієнтів (наприклад, події WebSocket). Початкова точка для визначення API служб полягає в ідентифікації операцій, які кожна служба повинна надавати. Події можуть бути введені за потреби для полегшення координації служб та зовнішніх сповіщень.

Розробка API сервісів полягає у визначенні операцій для кожного мікросервіса та взаємозв’язку між сервісами та операціями.

API мікросервісу взаємодії з Telegram Bot API має наступний функціонал:

– ініціалізація сервісу мікросервісу для взаємодії з Telegram Bot API здійснюється через виклик функцій connectToTelegramAPI(params), createBotGateServiceListener(params), createBotLogicServiceConnection(params) та createFileServiceConnection(params);

– обробка подій від Telegram Bot API використовує onTelegramAnyAction(action) для реагування на різні події, що надходять від Telegram Bot API;

– сервіс повідомлень включає функції onTelegramTextMessage(msg), onTelegramPhotoMessage(msg), onTelegramDocumentMessage(msg) для обробки текстових, фото, та документальних повідомлень;

– сервіс файлів використовує sendTelegramTextMessage(msg), sendTelegramPhotoMessage(msg), sendTelegramDocumentMessage(msg) для відправки повідомлень через Telegram Bot API.

Даний мікросервіс працюватиме по типу обгортки над методами обміну даними з Telegram Bot API. Сервіс зберігатиме всі необхідні для його роботи дані в оперативній пам’яті, без зайвої потреби у зверненні до БД. Оскільки сервіс відповідальний за створення з’єднання між системою та серверами Telegram: connectToTelegramAPI(params), то слід вказати, що саме на цьому етапі визначається механізм з’єднання: пулінг або вебхук. Також важливою частиною є обробка помилок, які можуть виникнути при запитах до Telegram Bot API. Можливі помилки при роботі:

– блокування користувачем чат-бота з різних причин;

– перевищення ліміту надсилання запитів;

– помилкові або невалідні параметри запиту;

– будь-які інші можливі помилки мережі.

Доступ для клієнтів до необхідних даних надається за допомогою протоколів HTTP та WebSockets.

Сервіс авторизації забезпечуватиме автентифікацію користувачів, керування їх сесіями та безпечне зберігання даних для доступу до інших сервісів веб-додатку (рис.2.1). Сервіс авторизації користувачів веб-додатку надаватиме наступний функціонал:

– ініціалізація сервісу за допомогою виклику функцій createAuthServiceListener() та createWebServerListener();

– керування сесіями користувачів: автентифікація за допомогою login(), перевірка сесії та токену через validateUserSession() та validateToken() відповідно;

– перевірка терміну дії сесій користувачів через сервіс service. Сесія діє протягом 1 години та автоматично поновлюється при кожному запиті користувача.

– зберігання паролів у БД у вигляді хешу MD5 з міркувань безпеки та захисту персональних даних (рис.2.2).

Сервіс впроваджує механізм створення сесій та токенів доступу для користувачів після їхньої успішної автентифікації за допомогою імені користувача та паролю. Для подальшої ідентифікації користувача буде використовуватися система HTTP-only cookies, як це показано на рисунку 2.1:

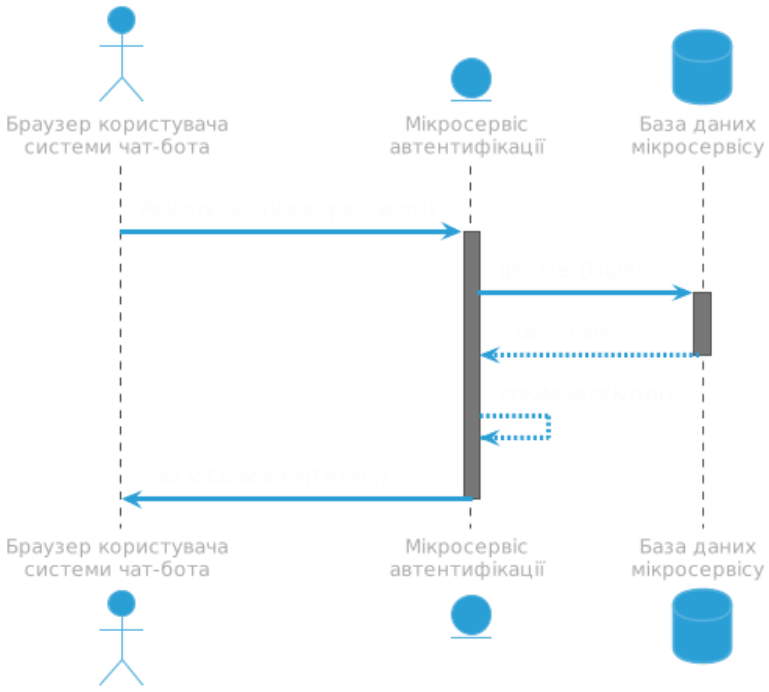


Рисунок 2.1 – Процес автентифікації користувача у системі

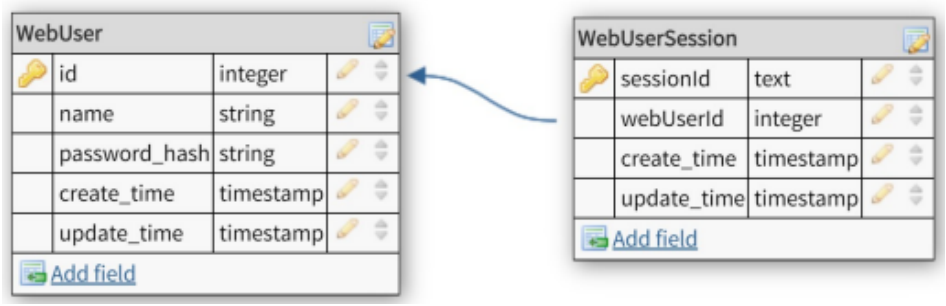


Рисунок 2.2 – Схема бази даних зберігання паролів

API мікросервісу управління файлами надає наступний функціонал:

– ініціалізація сервісу, що включає методи для створення слухачів на веб-сервері та веб-сокетах, а також ініціалізації зв’язків із сервісом авторизації;

– робота з фотографіями та документами – можливість створення, отримання, оновлення фотографій і документів за унікальним ідентифікатором;

– Взаємодія через HTTP та WebSockets – операції з фото та документами доступні через HTTP та WebSocket з’єднання, що дозволяє інтеграцію з різними клієнтськими додатками.

– завантаження файлів – функції для пошуку файлів за параметрами та завантаження файлів з мережі за URL.

Файлова обробка та конфігурація їх доступу мають свої нюанси через функціональність Telegram Bot API. Файли можна завантажувати в систему двома шляхами: через веб-додаток або через чат-бот. У системі для кожного файлу створюватиметься унікальна назва без вказівки розширення, що сприяє анонімності та підвищує безпеку. Усі дані про файли будуть збережені в базі даних, включаючи оригінальну назву, MIME-тип, розмір та параметри доступу. Також Telegram сервери надають унікальний ідентифікатор файлу (file_id), що дозволяє доступ до файлу та його відправку у повідомленнях без повторного завантаження, що економить трафік та час при розсиланнях.

API сервісу розсилок надає можливості для складного управління масовими повідомленнями, включаючи:

– планування повідомлень (таблиця PlannedMailing): організація та відправка текстових та медіа повідомлень для груп користувачів у заздалегідь визначений час;

– персоналізовані розсилки: автоматизоване створення та надсилання повідомлень новим користувачам чат-боту після певного інтервалу часу від моменту їх реєстрації;

– моніторинг розсилок: постійний відстеження статусу активних та минулих кампаній розсилки в реальному часі;

– управління розсилками: гнучке керування активними розсилками з можливістю виправлення помилок та автоматичного відновлення процесів після перерв у роботі сервісу.

Створення нового запису в таблиці PlannedMailing запускає процес планування нової розсилки за допомогою методу plannedMailingService(). Перед початком розсилки сервіс звертається до сервісу логіки чат-бота із запитом списку користувачів, вказуючи додаткові параметри (segment, users_filter_query, referral_id). Поле segment визначає сегмент користувачів для розсилки:

– усі користувачі чат-бота (з можливим фільтром по даті створення користувачів в полі users_filter_query);

– користувачі, конкретної групи (конкретний id у полі student_id).

Після отримання списку користувачів, в БД створюються нові записи в таблиці PlannedMailingJob – це гарантує відновлення розсилки при перезапуску сервісу. Далі нові записи беруться з таблиці і відправляються повідомлення відповідним користувачам через Telegram Bot API. При кожній успішній відправці, запис в PlannedMailing оновлюється, а записи в PlannedMailingJob видаляються. При оновленні таблиць генеруються внутрішні події, інформація про зміни відправляється клієнтам по WebSockets – так реалізується моніторинг розсилки у реальному часі.

Персональні розсилки працюють аналогічно. Нова розсилка створюється методом createPersonalMailing() при реєстрації користувача. Потім з таблиці PersonalMailing беруться всі записи, плануються завдання зі створенням записів в PersonalMailingJob. У вказаний час відправляється повідомлення користувачам через Telegram Bot API. При успішній відправці оновлюється PersonalMailing, видаляються записи з PersonalMailingJob. Аналогічним чином генеруються внутрішні події, а інформація про зміни надсилається по WebSockets.

2.3 Розгортання мікросервісів

Обрана мікросервісна архітектура дає можливість гнучкого розгортання окремих компонентів системи. Наприклад, файловий сервіс потребує багато постійної пам’яті та швидкого інтернет-з’єднання для зберігання і передачі великих файлів. А сервіс розсилок більше потребує швидкодіючий CPU та оперативну пам’ять.

Тому доцільно розгорнути файловий сервіс на хмарній інфраструктурі з відповідними необхідними характеристиками, а сервіс розсилок – на власному сервері з потужним процесором та ОЗУ. Розглянемо можливий варіант розгортання решти сервісів на одній фізичній машині.

Для балансування навантаження та маршрутизації запитів використовується NGINX як реверс проксі. Він також налаштовує HTTPS шифрування через обмін SSL сертифікатами.

Кожному сервісу призначено унікальні IP та порти – 127.0.0.1:[порт] для основного функціоналу і 127.0.0.1:[веб-порт] для веб-служби.

Клієнтські запити HTTP приймає NGINX. Для SPA застосунку використовується одна головна HTML сторінка. NGINX аналізує запит і перенаправляє його потрібному сервісу.

Сервіси ідентифікуються або за допомогою піддоменів (auth.domain.com), або шляхами URI (domain.com/auth). Піддомени вимагають додаткового налаштування CORS. Така гнучка архітектура дозволяє в подальшому масштабувати окремі сервіси відповідно до зростаючого навантаження та технічних вимог.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ

3.1 Програмна реалізація інтерфейсу

Інтерфейс користувача (UI) створюється для забезпечення ефективної взаємодії користувача з програмним продуктом, організовуючи його структуру та дизайн для досягнення користувацьких цілей. Основні напрямки дизайну UI охоплюють:

– UX дизайн, що зосереджений на створенні послідовного користувацького досвіду, що веде до бажаного результату;

– дизайн взаємодії, який аналізує поведінку користувачів;

– оцінка зручності використання продукту;

– інформаційна архітектура, що досліджує структурування інформації;

– візуальний дизайн, який включає колір, типографію та загальну естетику.

Процес проєктування є ітеративним та містить кілька важливих кроків:

– аналіз завдання, де визначаються основні завдання та їхні компоненти;

– створення моделі користувача, що базується на абстрактних об’єктах та їхніх діях;

– відображення інформації, що включає управління інформацією та отримання зворотного зв’язку;

– команди, що поділяються на глобальні та локальні рівні.

Під час розробки клієнтської частини системи було використано сучасні підходи, такі як:

– RWD (Responsive Web Design) – метод веб-дизайну, який адаптує інформацію для комфортного відображення на пристроях з різними розмірами екранів [17].

– MFD (Mobile-First Design) – підхід до верстки, який вважає пріоритетним вигляд на мобільних пристроях [18].

Підхід до проєктування було визначено як односторінковий додаток (SPA) – це тип веб-сайту, який здійснює оновлення вмісту на основі дій користувача, але не вимагає повторного завантаження нового HTML-документу від сервера. SPA завантажується один раз при першому зверненні до серверу і далі оновлює лише необхідні частини сторінки, зберігаючи взаємодію з сервером лише для отримання даних. Це забезпечує швидке взаємодія після ініціального завантаження сайту [19].

Переваги SPA включають:

– збільшення швидкості оновлення вмісту сторінок;

– поліпшення користувацького досвіду;

– розділення функціоналу бекенду і фронтенду, з усією логікою відображення на стороні SPA.

У контексті системи управління чат-ботом, де необхідно постійне оновлення даних, SPA є оптимальним рішенням. React.js часто вибирають для розробки SPA, завдяки його ефективності та гнучкості.

3.2 Опис компонентів проєкту

Основними компонентами системи є зовнішні залежності (модулі які завантажуються) та внутрішні компоненти (написані власноруч). Зовнішні залежності зазвичай вирішують низькорівневі завдання, а внутрішні – конкретні прикладні проблеми. Наприклад, у нашому випадку зовнішні залежності – це React.js, react-routerdom, socket-io-client та інші. Усі ці залежності та налаштування описуються у файлі package.json, який обробляється менеджером npm.

Розглянемо процес створення внутрішніх компонентів у середовищі React за вимогами до системи керування чат-ботом. Спочатку створюємо HTML документ з базовою розміткою та тегом з унікальним id – вхідною точкою для відображення компонентів React. Далі створюємо js файл, що

доєднує головний компонент до цієї вхідної точки, підключає основні стилі і систему маршрутизації.

Для маршрутизації використовуємо React Router – він створює навігацію у вигляді компонентів з присвоєними URL. Це реалізується через API HTML5 history. Щоб мати доступ до history з будь-якої точки додатку, експортуємо його з окремого модуля як Singleton.

Для стилів підключаємо Bootstrap, MDBootstrap та FontAwesome – для зовнішнього вигляду і адаптивної верстки. Маємо наступну структуру проєкту (рис.3.1), яка ще буде дещо змінена до кінця розробки.

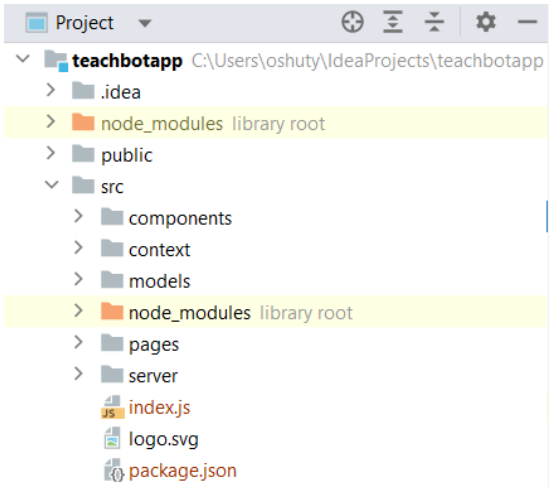


Рисунок 3.1 – Структура проєкту

У теці components, було створено компоненти, які виконують конкретне завдання і є незалежними одиницями. Створення дерева компонентів на одному рівні – це високорівневий погляд, також характерним є те, що кожний вузол проєкту ділиться на компоненти з меншими зонами відповідальності. Найнижчим рівнем компонентів можна вважати базові HTML-теги. Деякі низькорівневі компоненти доцільно використовувати в різних частинах системи, якщо вони реалізують спільний функціонал.

Прикладами таких загальних компонентів можуть бути SearchBar, FileInfoDialog тощо. Їх повторне використання дозволяє оптимізувати структуру і уникнути дублювання коду.

При переході на веб-застосунок виконується реєстрація або авторизація (рис.3.2) за допомогою виклику методу API мікросервісу авторизації validateUserSession() і у залежності від отриманої відповіді відбувається відображення основних компонентів системи або компонента Login, яка складається з двох полів: імені користувача і пароля.

У реалізації реєстрації нового користувача передбачено що, роль викладача можуть мати тільки користувачі в домені @cs.sumdu.edu.ua. На пошту реєстрації надходить лист, у якому є активне посилання на підтвердження аккаунту.

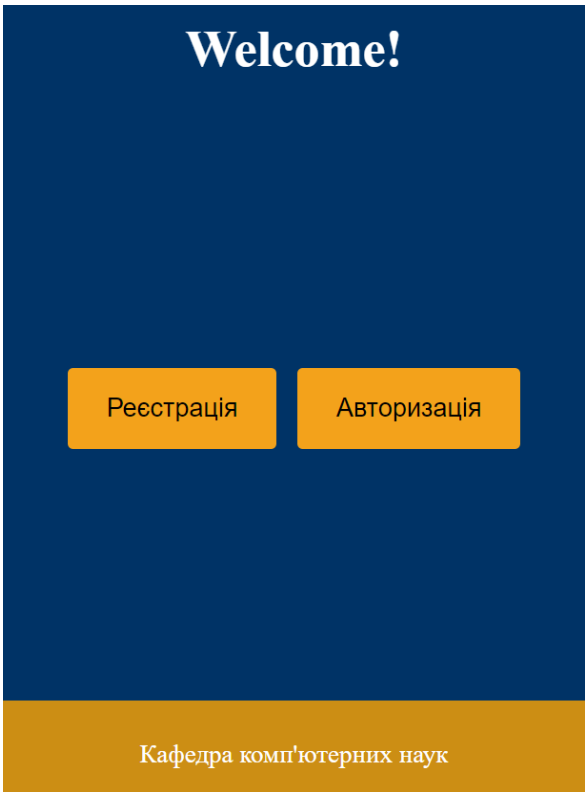
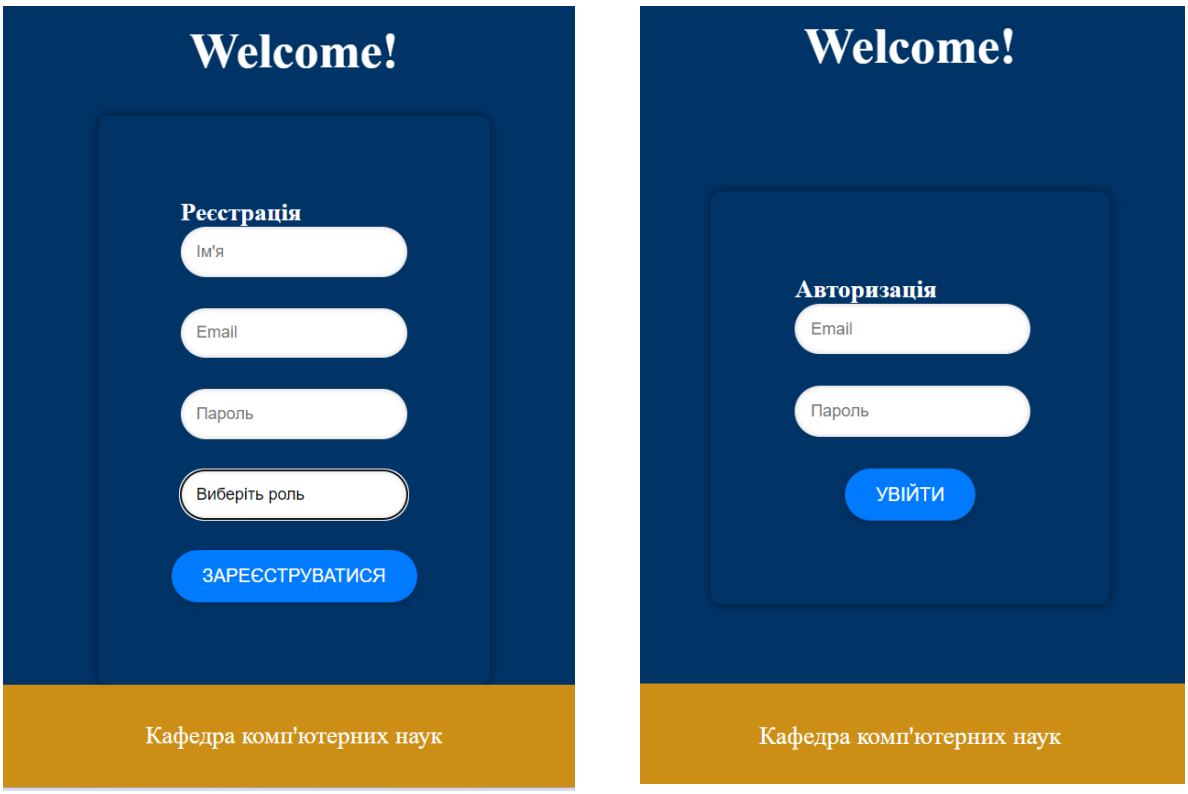


Рисунок 3.2 – Стартовий вхід на веб-застосунок



а б

Рисунок 3.3 – Форма реєстрації нового користувача (а) і авторизації (б)

Для того щоб передати користувачу попередження про будь-які помилки, використовується компонент ErrorMessages. Це невелике повідомлення з текстовим вмістом, яке з’являється у кутку екрану (рис. 3.4). Повідомлення автоматично зникають через кілька секунд і відображаються поверх інших вікон.

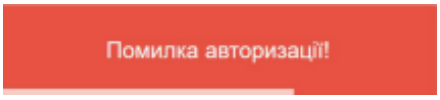


Рисунок 3.4 – Вікно помилки для користувача

Основні компоненти розробленої веб-системи:

– Header Component: цей компонент служить для навігації та доступу до загальних операцій у веб-додатку. Використання React.js дозволяє мати єдиний екземпляр цього компоненту, який автоматично оновлюється на всіх сторінках;

– Sidebar: включає посилання для навігації по веб-застосунку;

– Navbar: містить посилання до загальних операцій, які відрізняються набором для ролей;

– UsersManagement Component: відображає інформацію про користувачів у таблиці з можливістю налаштування кількості показаних записів, пагінації, пошуку та сортування. дані включають id, ім’я, статус, дати створення та оновлення. Доступний тільки для ролі «Викладач».

– FileManagement Component: дозволяє перегляд, створення, редагування, пошук за ключовими словами, фільтрацію за категоріями, а також зміну режиму публічності файлів.

– FileManagementCreateDialog: Використовується для створення файлів, дозволяє додавати файл, вказувати підпис і вибирати режим публічності. Максимальний розмір файлу обмежено 20 МБ.

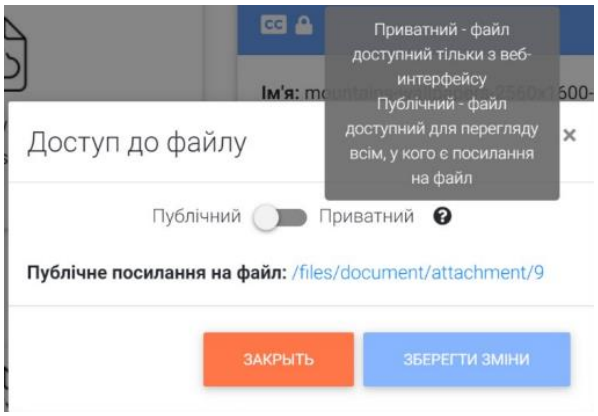


Рисунок 3.5 – Режим доступу до файлів

Лістинг коду основних компонент додано у Додатку А.

У веб-додатку імплементовано систему розсилок, котра поділяється на дві категорії: заплановані та персоналізовані розсилки. Розсилки в телеграм-бот мають на меті попередити про певні події, такі як дедлайни виконання, сповіщення про контрольні роботи, активні опитування, тощо. Для організації запланованих розсилок, користувач має змогу задати параметри такі як текст повідомлення чи файл, необов’язково додати клавіатуру, визначити цільову

аудиторію, фільтр за датою реєстрації користувачів та встановити конкретний час для відправлення.

Управління повідомленнями відбувається через компонент ChatsManagement, який забезпечує відображення списку користувачів, що вступили у діалог з чат-ботом, презентує історію переписки та надає можливість спілкування з певними користувачами. Це розширення стандартного функціоналу, оскільки Telegram за замовчуванням не передбачає вбудованих механізмів для такого типу взаємодії.

Компонент ChatsManagement складається з декількох підкомпонентів, включно з ChatRoomList для переліку кімнат чату, ChatRoom для представлення окремої кімнати, і ChatMessages для відображення повідомлень. Така структура дозволяє чітко відокремити різні функціональні частини системи управління повідомленнями.

Тестування функціональних можливостей застосунку показали задовільні результати взаємодії веб-застосунку та чат-боту. Подальша модифікація зможе розширити функції додатку.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досліджено технології чат-ботів та розроблено систему управління чат-ботом на React. Зокрема, проведено огляд сучасних платформ для створення чат-ботів, проаналізовано й обґрунтовано технології для розробки системи, створено серверну та клієнтську частини додатку.

До системи поставлені функціональні вимоги, які повністю реалізовані:

– перегляд користувачів чат-бота (загальна інформація, дата останнього візиту);

– функціонал чату для обміну повідомленнями між користувачами і ботом, збереження історії;

– файловий сервіс для зберігання файлів;

– можливість розсилок повідомлень окремим групам користувачів, збереження історії розсилок;

– облік та перегляд загальних статистичних даних.

Також реалізовано мережеву взаємодію мікросервісів, серверів Telegram, клієнтів. Увесь функціонал працює повноцінно. Можливе розгортання як на одній машині з однією ОС, так і на різних машинах та ОС. Надано рекомендації з розгортання та конфігураційні файли.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Словник термінів [Електронний ресурс] // tonyline.com.ua – Режим доступу до ресурсу: https://tonyline.com.ua/glossary/chatbot/.

2. Messenger App [Електронний ресурс] // https://www.helpshift.com/ – Режим доступу до ресурсу: https://www.helpshift.com/glossary/messenger-app/.

3. Global social media statistics [Електронний ресурс] // https://datareportal.com/ – Режим доступу до ресурсу: https://datareportal.com/social-media-users.

4. Українські медіа, ставлення та довіра у 2023 р [Електронний ресурс] // Опитування USAID-Internews щодо споживання медіа – Режим доступу до ресурсу: https://internews.in.ua/wp-content/uploads/2023/10/Ukrainski-media-stavlennia-ta-dovira-2023r.pdf.

5. Messaging apps are now bigger than social networks [Електронний ресурс] // Business Insider – Режим доступу до ресурсу: https://www.businessinsider.com/the-messaging-app-report-2015-11.

6. Важливі фактори взаємодії з користувачами чат-бота. [Електронний ресурс] // https://gerabot.com/ – Режим доступу до ресурсу: https://gerabot.com/article/detalno_pro_chatboti.

7. Все про чат-боти: типи і приклади, якому бізнесу підійде, список конструкторів для створення [Електронний ресурс] // https://web-promo.ua/. – 2020. – Режим доступу до ресурсу: https://web-promo.ua/ua/blog/vse-o-chat-botah-tipy-i-primery-kakomu-biznesu-podojdet-spisok-konstruktorov-dlya-sozdaniya/.

8. How to Make a Chatbot? [Електронний ресурс] // https://medium.com/. – 2022. – Режим доступу до ресурсу: https://medium.com/@appmasterio/how-to-make-a-chatbot-1f2765457855.

9. Create your first Facebook messenger bot using Flask & AWS Lambda [Електронний ресурс] – 2022. – Режим доступу до ресурсу: https://www.idiotinside.com/2017/12/22/create-fb-messenger-bot-aws-lambda/.

10. Webhook VS Short Pooling [Електронний ресурс] // https://medium.com/ – Режим доступу до ресурсу: https://medium.com/@saddy.devs/вебхук-vs-short-пулінг-bb84b5f1061c.

11. Telegram Bot API [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/api.

12. Client-Server Model [Електронний ресурс] // geeksforgeeks.org. – 2022. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/client-server-model/.

13. What Are Microservices? Definition, Examples, Architecture, and Best Practices for 2022 [Електронний ресурс] // www.spiceworks.com. – 2022. – Режим доступу до ресурсу: https://www.spiceworks.com/tech/devops/articles/what-are-microservices/.

14. What are Microservices? How Microservices architecture works [Електронний ресурс] // middleware.io. – 2023. – Режим доступу до ресурсу: https://middleware.io/blog/microservices-architecture/.

15. Microservices vs. monolithic architecture [Електронний ресурс] // www.atlassian.com – Режим доступу до ресурсу: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith.

16. MVC [Електронний ресурс] // developer.mozilla.org. – 2023. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/MVC.

17. Responsive web design basics [Електронний ресурс] // web.dev – Режим доступу до ресурсу: https://web.dev/articles/responsive-web-design-basics.

18. Mobile-First Design: The Best Practices [Електронний ресурс] // techmagic.co. – 2022. – Режим доступу до ресурсу: https://www.techmagic.co/blog/best-practices-for-mobile-first-design/.

19. Односторінковий додаток (SPA) vs багатосторінковий додаток (MPA): переваги та недоліки [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://merehead.com/ua/blog/spa-vs-mpa-application/.

20. React [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://react.dev/learn.

21. Сучасний підручник з JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://uk.javascript.info/.

ДОДАТОК А

Приклад конфігураційного файлу NGINX для тестування системи

```
http
{
    index index.html;
    sendfile on;
    include ./conf/mime.types;
    default_type application/octet-stream;
    server
    {
        listen 127.0.0.1:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        location /auth/
        {
            proxy_pass http://127.0.0.1:9081/;
        }
        location /bot/
        {
            proxy_pass http://127.0.0.1:9083/;
        }
        location /messages/
        {
            proxy_pass http://127.0.0.1:9084/;
        }
        location /files/
        {
            proxy_pass http://127.0.0.1:9085/;
        }
        location /mailing/
        {
            proxy_pass http://127.0.0.1:9086/;
        }
        root ../www/;
        location /
    }
}
}
```

Конфігураційний файл package.json клієнтської частини системи:

```
{
  "name": "teachbotapp",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-bootstrap": "^2.9.2",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "proxy": "http://127.0.0.1:8080",
  "scripts": {
    "start": "set HOST=127.0.0.1&&react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Лістинг коду основних компонентів:

```
import React, { useState } from 'react';
import Register from './Register';
import Login from './Login';
import './Welcome.css';

3 usages
CodiumAI: Test this function
function Welcome() {
    const [show : string , setShow] = useState( initialState: 'welcome');

    return (
        <div className="welcome-container">
            <div className="header">
                <h1>Welcome!</h1>
            </div>

            <div className="content">
                {show === 'welcome' && (
                    <div className="options-container">
                        <button onClick={() : void  => setShow( value: 'register')}>Реєстрація</button>
                        <button onClick={() : void  => setShow( value: 'login')}>Авторизація</button>
                    </div>
                )}
                {show === 'register' && <Register />}
                {show === 'login' && <Login />}
            </div>

            <div className="footer">
                <p>Кафедра комп'ютерних наук</p>
            </div>
        </div>
    );
}

2 usages
export default Welcome;
```

```
import { createContext, useState } from 'react';

2 usages
export const AuthContext : Context<unknown>  = createContext();

no usages
CodiumAI: Test this function
export default function AuthProvider({children}) {
    const [user, setUser] = useState( initialState: null);

    1 usage
    const login = (userData) : void  => {
        setUser(userData);
    }

    1 usage
    const logout = () : void  => {
        setUser( value: null);
    }

    return (
        <AuthContext.Provider value={{user, login, logout}}>
            {children}
        </AuthContext.Provider>
    )

}
```

```
import {Button, Form, FloatingLabel} from 'react-bootstrap';
import React from "react";

no usages
CodiumAI: Test this function
export default function Register() {
    return (
        <Form className="register-form">
            <FloatingLabel controlId="name" label="Ім'я" className="mb-3">
                <Form.Control type="text" placeholder="Ваше ім'я"/>
            </FloatingLabel>

            <FloatingLabel controlId="email" label="Email" className="mb-3">
                <Form.Control type="email" placeholder="name@example.com"/>
            </FloatingLabel>

            <FloatingLabel controlId="password" label="Пароль" className="mb-3">
                <Form.Control type="password" placeholder="Пароль"/>
            </FloatingLabel>

            <Form.Select className="mb-3">
                <option>Студент</option>
                <option>Викладач</option>
            </Form.Select>

            <Button variant="primary" type="submit">
                Зареєструватись
            </Button>
        </Form>
    );
}
```

```
import {useState, useContext} from 'react';
import {AuthContext} from '../contexts/AuthContext';

export default function Teacher() {
    const [disciplines, setDisciplines] = useState([]);
    const [groups, setGroups] = useState([]);
    const [notification, setNotification] = useState('');
    const [interactiveActivity, setInteractiveActivity] = useState('');
    const {user} = useContext(AuthContext);
    const addDiscipline = async () => {
        const disciplineName = prompt("Введіть назву дисципліни:");

        if (!disciplineName) {
            return;
        }

        try {

            const response = await fetch('/api/disciplines', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',

                },
                body: JSON.stringify({name: disciplineName})
            });

            if (response.ok) {

                setDisciplines(prevDisciplines => [...prevDisciplines,
disciplineName]);
            } else {

                console.error("Помилка при відправці даних");
            }
        } catch (error) {

            console.error("Помилка мережі", error);
        }
    };
```

```
const addGroup = async () => {
    const groupName = prompt("Введіть назву групи:");

    if (!groupName) {
        return;
    }

    try {

        const response = await fetch('/api/groups', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',

            },
            body: JSON.stringify({name: groupName})
        });

        if (response.ok) {

            setGroups(prevGroups => [...prevGroups, groupName]);
        } else {

            console.error("Помилка при відправці даних");
        }
    } catch (error) {

        console.error("Помилка мережі", error);
    }
};
const sendNotification = async () => {
    if (!notification.trim()) {
        alert("Будь ласка, введіть текст сповіщення.");
        return;
    }

    try {
        const response = await fetch('/api/notifications/send', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',

            },
            body: JSON.stringify({message: notification})
        });

        if (response.ok) {
            alert("Сповіщення успішно відправлено.");
            setNotification('');
        } else {

            alert("Помилка при відправці сповіщення.");
        }
    } catch (error) {

        alert("Помилка мережі при відправці сповіщення.");
    }
};

const createInteractiveActivity = () => {
    const activity = prompt("Опишіть вашу інтерактивну активність:");

    if (activity) {
```

```
            setInteractiveActivity(activity);
        }
    };

    return (
        <div>
            <h3>Особистий кабінет (викладач)</h3>
            <p>Викладач: {user.name}</p>

            <div>
                <h4>Мої дисципліни</h4>
                <button onClick={addDiscipline}>Додати дисципліну</button>
                {/* Відображення списку дисциплін */}
            </div>

            <div>
                <h4>Мої групи</h4>
                <button onClick={addGroup}>Додати групу</button>
                {/* Відображення списку груп */}
            </div>

            <div>
                <h4>Сповіщення для студентів</h4>
                <textarea
                    value={notification}
                    onChange={(e) => setNotification(e.target.value)}
                    placeholder="Напишіть сповіщення"
                />
                <button onClick={sendNotification}>Відправити
сповіщення</button>
            </div>

            <div>
                <h4>Інтерактив на занятті</h4>
                <input
                    type="text"
                    value={interactiveActivity}
                    onChange={(e) => setInteractiveActivity(e.target.value)}
                    placeholder="Введіть активність"
                />
                <button onClick={createInteractiveActivity}>Створити
активність</button>
            </div>

        </div>
    );
}
```