МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Острозька академія»

Інститут інформаційних технологій та бізнесу

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра
на тему: «Розробка мобільного додатку з AI -асистентом для планування подій»

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Добровольський Максим Олександрович
Керівник: *викладач, фахівець-практик*
Мельничук Олександр Павлович

Рецензент: *кандидат технічних наук, доцент,*
доцент кафедри прикладної математики
Донецького національного університету
імені Василя Стуса
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних _____
(проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від « 20 » травня 2026 р.

Острог, 2026

АНОТАЦІЯ

кваліфікаційної роботи

на здобуття освітнього ступеня бакалавра

Тема: Проєктування та розробка мобільного застосунку "AI Calendar" для голосового керування подіями на базі React Native та штучного інтелекту.

Автор: Добровольський Максим Олександрович

Пояснювальна записка до кваліфікаційної роботи: *51 (кількість сторінок роботи), 7 (кількість таблиць) табл., 9 (кількість джерел) джерел.*

Ключові слова: REACT NATIVE, FASTAPI, MONGODB, GEMINI API, ШТУЧНИЙ ІНТЕЛЕКТ, ГОЛОСОВЕ КЕРУВАННЯ, КАЛЕНДАР, NLP, МОБІЛЬНИЙ ЗАСТОСУНОК, ТАЙМ-МЕНЕДЖМЕНТ.

Короткий зміст: У кваліфікаційній роботі розглянуто процес проєктування та розробки мобільного застосунку "AI Calendar", призначеного для керування особистим розкладом за допомогою голосових команд та штучного інтелекту. Актуальність теми зумовлена необхідністю підвищення ефективності тайм-менеджменту та зменшення когнітивного навантаження користувачів під час ручного введення даних у традиційні календарі. У теоретичній частині проаналізовано сучасні підходи до обробки природної мови (NLP) та використання великих мовних моделей (LLM) для розпізнавання намірів користувача. В інформаційному забезпеченні спроєктовано структуру бази даних NoSQL та розроблено алгоритм парсингу сутностей (дати, часу, дій) за допомогою гнучких промптів для моделі Gemini 2.5 Flash. Практична частина присвячена реалізації бекенду на FastAPI та фронтенду на React Native з підтримкою Native Audio Dialog (одночасна генерація тексту та голосу), що забезпечує швидку, безпечну та інтуїтивно зрозумілу взаємодію користувача з розкладом.

SUMMARY

Theme: Design and development of the "AI Calendar" mobile application for voice-controlled event management based on React Native and Artificial Intelligence.

Keywords: REACT NATIVE, FASTAPI, MONGODB, GEMINI API, ARTIFICIAL INTELLIGENCE, VOICE CONTROL, CALENDAR, NLP, MOBILE APPLICATION, TIME MANAGEMENT.

Abstract: This qualification work examines the design and development process of the "AI Calendar" mobile application intended for personal schedule management using voice commands and artificial intelligence. The relevance of the topic is driven by the need to improve time management efficiency and reduce the cognitive load on users during manual data entry in traditional calendars. The theoretical part analyzes modern approaches to Natural Language Processing (NLP) and the use of Large Language Models (LLM) to recognize user intents. The information support section details the design of a NoSQL database structure and the development of an entity parsing algorithm (date, time, actions) using flexible prompts for the Gemini 2.5 Flash model. The practical part focuses on the implementation of the backend using FastAPI and the frontend using React Native, featuring Native Audio Dialog (simultaneous generation of text and voice), which provides fast, secure, and intuitive user interaction with the schedule.

ЗМІСТ

ВСТУП

У сучасному швидкоплинному світі ефективне управління часом (тайм-менеджмент) є критично важливим аспектом як для професійної діяльності, так і для повсякденного життя. Зростаючий обсяг інформації, велика кількість зустрічей, дедлайнів та особистих справ вимагають використання надійних, гнучких та високопродуктивних інструментів для планування [7]. Більшість користувачів у повсякденній практиці покладаються на відомі цифрові рішення, такі як Google Calendar, Apple Calendar або Microsoft Outlook. Незважаючи на їхню багатофункціональність, розвинену екосистему та інтеграцію з іншими сервісами, традиційні цифрові календарі мають суттєвий архітектурний та ергономічний недолік — вони вимагають від користувача постійного ручного введення даних через графічний інтерфейс користувача (Graphical User Interface, GUI).

Процес вибору конкретної дати, часу, написання назви події, додавання опису та встановлення нагадувань часто є громіздким, часозатратним та незручним. Особливо гостро ця проблема проявляється, коли користувач знаходиться «на ходу», керує транспортним засобом, має обмежені можливості для взаємодії з екраном пристрою або страждає від хронічного браку часу. Традиційний інтерфейс створює так зване «когнітивне тертя», змушуючи людину підлаштовуватися під жорстку логіку програми замість того, щоб програма адаптувалася під природні паттерни поведінки людини [8].

З бурхливим розвитком технологій штучного інтелекту (ШІ) та алгоритмів обробки природної мови (Natural Language Processing, NLP) з'явилася унікальна можливість кардинально змінити парадигму взаємодії людини з комп'ютером (Human-Computer Interaction, HCI) [9]. Використання великих мовних моделей (Large Language Models, LLM), що здатні глибоко розуміти контекст, розпізнавати складні наміри користувача, нівелювати мовні помилки та генерувати структуровані дані з вільного тексту або безпосередньо з аудіопотоку, дозволяє повністю автоматизувати процес ведення

персонального розкладу.

Метою даної кваліфікаційної роботи є проєктування, архітектурне обґрунтування та програмна розробка комплексного продукту — мобільного застосунку «AI Calendar», який дозволяє користувачам керувати подіями в особистому календарі (створювати, оновлювати, шукати та видаляти записи) виключно за допомогою природних голосових команд, використовуючи мультимодальні можливості сучасного генеративного штучного інтелекту без проміжних етапів неефективного GUI-введення.

Для досягнення поставленої мети було визначено та вирішено наступні **завдання дослідження**:

1. Здійснити глибокий системний аналіз предметної області, дослідити недоліки існуючих календарних додатків та сформувати вимоги до сучасного голосового інтерфейсу (Voice User Interface, VUI).
2. Спроєктувати надійну, масштабовану клієнт-серверну архітектуру та оптимальну структуру документо-орієнтованої бази даних для збереження інформації.
3. Розробити серверну частину (Backend) з використанням асинхронного фреймворку Python та інтеграцією хмарного ШІ (Google Gemini API) для точного розпізнавання намірів користувача з аудіоданих та генерації детермінованого JSON.
4. Створити мобільний клієнт (Frontend) на базі фреймворку React Native для захоплення звуку з мікрофона, анімованої візуалізації станів та відображення календаря.
5. Забезпечити високу якість системи шляхом впровадження механізмів обробки помилок, покриття коду тестами та промислового розгортання в хмарній інфраструктурі з налаштуванням CI/CD.

Об'єктом дослідження є процес автоматизації управління особистим часом, розкладом та завданнями користувача за допомогою сучасних інформаційних технологій.

Предметом дослідження є методи, алгоритми та інструментальні засоби проєктування мобільних застосунків, інтеграція технологій обробки природної мови (NLP) та використання мультимодальних генеративних моделей для інтелектуального голосового керування інтерфейсами програмних систем.

РОЗДІЛ 1

ЗАГАЛЬНІ ПОЛОЖЕННЯ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметного середовища та процесу управління часом

У контексті глобалізації, цифровізації та стрімкого прискорення темпів суспільно-економічного життя, раціональний розподіл часових ресурсів перетворився з індивідуальної переваги на базову умову виживання та успішного функціонування особистості в інформаційному соціумі. Сучасна людина щоденно стикається з явищем інформаційного перевантаження (information overload), яке характеризується безперервним та неконтрольованим потоком вхідних тригерів: електронних листів, повідомлень у месенджерах, робочих та особистих зустрічей, дедлайнів, а також завдань з безперервної самоосвіти [7].

За таких умов класичні ментальні механізми людської пам'яті, зокрема робоча пам'ять (working memory), обсяг якої, згідно з фундаментальним законом Міллера, обмежений лише кількома структурними елементами, не здатні ефективно утримувати та структурувати складні багатокомпонентні розклади. Це зумовлює виникнення хронічного стресу, зниження концентрації уваги та прокрастинації, що сукупно призводить до втрати особистості та бізнесу значних економічних і ментальних ресурсів.

Як наслідок, виникла гостра науково-практична потреба у створенні та розвитку зовнішніх систем підтримки когнітивних процесів людини, базовим елементом яких став тайм-менеджмент — міждисциплінарна галузь знань, що лежить на стику психології, менеджменту та комп'ютерних наук [8]. Еволюція інструментарію тайм-менеджменту пройшла тривалий шлях: від вузькоспеціалізованих паперових носіїв, таких як блокноти, щоденники та матричні планери, до перших цифрових рішень класу PDA (Personal Digital Assistant) та сучасних хмарних календарних

платформ.

Проте, детальний ергономічний аналіз показує, що перехід до цифрових GUI-календарів не вирішив проблему «когнітивного тертя» (cognitive friction) у повній мірі. Під когнітивним тертям у межах теорії людино-машинної взаємодії (HCI) розуміють рівень супротиву інтерфейсу програмного забезпечення природним паттернам людського мислення та поведінки [8]. Коли у користувача виникає інтенція (намір) зафіксувати подію, його мозок оперує цілісними семантичними образами природної мови, наприклад: «потрібно не забути заїхати на СТО в п'ятницю після обіду».

Для того, щоб перетворити цей нативний намір на структуру даних у класичному додатку, користувач змушений виконати складну ментальну та механічну роботу з декомпозиції власної думки:

- Перекласти абстрактний час «після обіду» у конкретні цифри (наприклад, 15:00 або 16:00);
- Визначити точну календарну дату для поняття «в п'ятницю»;
- Здійснити механічне введення тексту через обмежений простір віртуальної клавіатури мобільного пристрою;
- Взаємодіяти з модальними вікнами вибору дати й часу (Date and Time Pickers), які часто мають невдалий UX-дизайн.

Цей ланцюжок дій створює значний психологічний бар'єр. Користувач підсвідомо оцінює часові та зусилля-витрати на взаємодію з інтерфейсом як занадто високі порівняно з миттєвою цінністю фіксації події. Як наслідок, значна частина дрібних, але критично важливих повсякденних завдань взагалі не потрапляє до цифрового розкладу, що нівелює саму ідею системного управління часом.

Це актуалізує задачу розробки інтерфейсів нового покоління — голосових

інтелектуальних систем (Voice User Interfaces, VUI), які мінімізують когнітивний бар'єр взаємодії, дозволяючи комп'ютеру адаптуватися до природної мови людини, а не навпаки [9].

1.2 Огляд наявних підходів і рішень на ринку цифрових календарів

На сучасному IT-ринку представлено безліч рішень для управління часом, які можна умовно розділити на дві великі категорії: класичні графічні календарі та системні голосові асистенти общего профілю. Проведемо порівняльний аналіз цих систем для виявлення їхніх ключових переваг та архітектурних обмежень.

Таблиця 1.1

Порівняльний аналіз існуючих категорій рішень для ведення цифрового розкладу

Критерій порівняння	**Класичні GUI-календарі (Google, Apple)**	**Системні асистенти (Siri, Google Assistant)**	**Запропонований застосунок «AI Calendar»**
Основний інтерфейс	Графічний (форми, кнопки, селектори)	Голосовий на основі жорстких шаблонів	Інтелектуальний голосовий (вільна мова)
Когнітивне навантаження	Високе (вимагає ручного введення тексту та вибору дат)	Середнє (вимагає знання чітких командних фраз)	Мінімальне (система адаптується під природну мову)
Обробка гнучкого контексту	Відсутня (або обмежена текстовим парсингом)	Низька (помилки при складних або непрямих запитах)	Висока (глибоке розуміння семантики та намірів)

Продовження Таблиці 1.1

Критерій порівняння	**Класичні GUI-календарі (Google, Apple)**	**Системні асистенти (Siri, Google Assistant)**	**Запропонований застосунок «AI Calendar»**
Робота з відносним часом	Вимагає ручного вибору на календарі	Обмежена жорсткими часовими рамками	Повна підтримка відносних зсувів дат та контексту
Кастомізація та розширення	Обмежена API платформи	Закрита екосистема вендора	Повна гнучкість, можливість інтеграції в будь-яку БД

Як видно з табл. 1.1, класичні рішення забезпечують високу наочність, але вимагають значних часових витрат на рутинне введення [1]. У свою чергу, системні голосові помічники першого покоління базуються на застарілих методах розпізнавання намірів (Intent Recognition), які жорстко прив'язані до мовних шаблонів (грамматик) [9]. Якщо користувач формулює свій запит нестандартно, використовує розмовні звороти або метафори, асистент виявляється безсилим. Крім того, вони мають серйозні проблеми з обробкою логіки оновлення чи каскадного видалення подій у складних тимчасових контекстах, а їхні API є закритими для сторонньої глибокої модифікації розробниками.

1.3 Аналіз розвитку технологій штучного інтелекту (LLM)

Поява та стрімка еволюція великих мовних моделей (LLM), таких як архітектури GPT від OpenAI або сімейство моделей Gemini від Google, здійснили справжню революцію в галузі обробки природної мови [4]. Сучасні LLM докорінно відрізняються від класичних моделей класифікації інтентов: вони оперують не просто статистичним розподілом слів чи пошуком заздалегідь визначених токенів-маркерів, а здатні виділяти глибинну семантику речення, спираючись на механізми Self-Attention (самоуваги) у

трансформерних архітектурах.

Здатність моделей до zero-shot та few-shot навчання дозволяє розробникам вирішувати складні завдання вилучення іменованих сутностей (Named Entity Recognition, NER) без необхідності тривалого, дорогого та ресурсомісткого навчання власних спеціалізованих нейромережевих архітектур на тисячах розмічених прикладів. Більш того, моделі останнього покоління (зокрема, Gemini 1.5 Flash та Gemini 2.5 Flash) є нативно мультимодальними [4]. Це означає, що нейромережа здатна приймати аудіопотік (сирі звукові файли або аудіо-трансляцію) безпосередньо як вхідний тензор (Input Tensor), минаючи класичний окремий етап перетворення мовлення в текст (Speech-to-Text, STT). Пряма обробка аудіо моделями LLM дозволяє аналізувати акустичні характеристики мови, інтонаційні наголоси, що значно знижує метрику помилок розпізнавання слів (Word Error Rate, WER) та мінімізує загальну затримку системи (Latency), яка є критичною для користувацького досвіду у VUI-додатках.

1.4 Постановка задачі та формування вимог до системи

На основі проведеного комплексного аналізу предметної області та технологічних можливостей сучасного штучного інтелекту було сформовано детальне технічне завдання та специфікацію вимог до мобільного застосунку «AI Calendar». Головна концептуальна ідея проєкту полягає у радикальному спрощенні взаємодії користувача з інтерфейсом системи шляхом зведення всього циклу управління подіями до однієї ключової інтерактивної дії — утримання кнопки активації мікрофона та вимовлення команди у вільній формі.

Сформулюємо вимоги до системи згідно з методологіями програмної інженерії [7]:

Функціональні вимоги (Functional Requirements):

- Наявність захищеної системи авторизації та автентифікації користувачів (реєстрація нових облікових записів, логін, випуск та валідація сесійних

JWT-токенів).

- Візуалізація сітки календаря з гнучким відображенням маркерів (індикаторів подій) у ті дні, на які заплановано хоча б одну активність.
- Можливість перегляду деталізованого списку подій на будь-який обраний користувачем день (відображення точного часу початку/завершення, назви, текстового опису).
- Запис високоякісного аудіо з мікрофона мобільного пристрою за допомогою інтерфейсу Push-to-Talk (запис триває, поки користувач утримує кнопку).
- Надійна асинхронна передача аудіоданих на сервер, їх семантичний аналіз за допомогою інтегрованого ШІ та безпомилкове виконання повного спектру CRUD-операцій (Create, Read, Update, Delete) у базі даних.
- Забезпечення повноцінного двостороннього діалогового вікна за допомогою інструментів Text-to-Speech (TTS): система повинна генерувати та відтворювати природну мовну відповідь за результатами виконання операції.

Нефункціональні вимоги (Non-Functional Requirements):

- **Продуктивність (Performance):** Загальний час відгуку системи (сквозне вимірювання Latency від моменту відпускання кнопки мікрофона до початку відтворення звукової відповіді, включаючи мережевий транспорт, обробку моделі Gemini та запис у БД) не повинно перевищувати 2.5–4 секунди в умовах нормального 3G/4G з'єднання.
- **Кросплатформеність (Cross-platform capability):** Мобільний клієнт повинен мати ідентичний інтерфейс та стабільну логіку роботи на операційних системах iOS та Android [1].
- **Надійність та відмовостійкість (Reliability):** Серверна архітектура зобов'язана ізолювати збої зовнішніх сервісів штучного інтелекту (наприклад, HTTP-помилки 429 при перевищенні лімітів або втрата мережевих пакетів), перехоплювати їх та коректно повертати користувачеві інформативні повідомлення, зберігаючи

цілісність даних.

1.5 Специфіка обробки природної української мови та лінгвістичні виклики для великих мовних моделей

При проєктуванні інтелектуальних систем, орієнтованих на голосове керування, критично важливим аспектом є врахування лінгвістичних особливостей цільової мови. На відміну від англійської мови, яка належить до аналітичного типу і характеризується фіксованим порядком слів у реченні та відносно бідною морфологією, українська мова є яскравим представником синтетичних мов флективного типу. Це створює низку серйозних фундаментальних викликів для класичних алгоритмів комп'ютерної лінгвістики та систем розпізнавання мовлення.

Першочерговою проблемою є розвинена система словозміни. В українській мові іменники, прикметники, займенники та числівники мають складні парадигми відмінювання за сімома відмінками, трьома родами та двома числами. Дієслова, у свою чергу, мають розгалужену систему дієвідмін, категорій часу, стану та виду (доконаний і недоконаний). Для завдань автоматичного планування це означає, що одна і та сама сутність або дія може набувати десятків різних морфологічних форм. Наприклад, назва дії «записати» у природному мовленні користувача може трансформуватися у форми «запиши», «запишіть», «запишеш», «записав», «записала», а часові маркери можуть звучати як «ввечері», «вечором», «на вечір», «понеділка», «у понеділок», «щопонеділка».

Традиційні підходи до обробки природної мови, які базувалися на правилах (Rule-based NLP) або на механізмах стемінгу (виділення основи слова) та лематизації (приведення до початкової форми), демонструють вкрай низьку ефективність в описаних умовах. Вони призводять до втрати важливих смислових відтінків або потребують створення колосальних за обсягом лінгвістичних словників та баз правил, які все одно не здатні покрити всі варіанти живого розмовного мовлення.

Додатковим ускладнюючим чинником виступає вільний порядок слів в українському

реченні. Користувач може сформулювати свій намір у абсолютно довільній послідовності, наприклад: «На завтра о десятій ранку признач мені зустріч з керівником», «Признач мені зустріч з керівником о десятій ранку на завтра» або «Зустріч з керівником на завтра о десятій ранку мені признач». Усі ці синтаксичні конструкції мають ідентичний семантичний зміст (інтент), проте їхній структурний аналіз за допомогою класичних граматичних парсерів є надзвичайно складним.

Саме тому в даній роботі обґрунтовано доцільність застосування великих мовних моделей нового покоління, таких як Google Gemini. Завдяки технології контекстних ембеддінгів, модель оперує не окремими ізольованими словами чи їхніми основами, а цілісними смисловими векторами. Вона здатна уловлювати глибокі семантичні зв'язки між членами речення незалежно від їхнього розташування в тексті та морфологічної форми. Багатоголовий механізм уваги дозволяє моделі динамічно зважувати важливість кожного слова у контексті всього висловлювання, що забезпечує безпрецедентну точність екстракції параметрів подій навіть за умов використання користувачем специфічного розмовного сленгу, скорочень або неповних речень.

РОЗДІЛ 2

ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Проєктування архітектури системи

Для забезпечення високого рівня масштабованості, безпеки, гнучкості розробки та простоти супроводу системи було прийнято рішення використовувати класичну трирівневу клієнт-серверну архітектуру з чітким розділенням зон відповідальності (Separation of Concerns) [7].

Розглянемо архітектурні рівні детальніше:

1. **Presentation Layer (Рівень представлення):** Мобільний додаток, розроблений на базі фреймворку React Native із застосуванням інструментів Expo [2]. Його завдання — забезпечення взаємодії з користувачем, рендеринг інтерфейсу, захоплення аудіосигналу з апаратного мікрофона за допомогою низькорівневих API, відправка бінарних даних на сервер та відтворення аудіовідповідей.
2. **Application Layer (Рівень бізнес-логіки):** Високопродуктивний веб-сервер на базі асинхронного Python-фреймворку FastAPI [3]. Він виступає центральним шлюзом та оркестратором системи: керує потоками даних, здійснює автентифікацію через JWT, формує спеціалізовані контекстні запити до хмарного сервісу Google Gemini API, валідує отримані структуровані дані за допомогою Pydantic-моделей та ініціює відповідні транзакції у базі даних.
3. **Data Layer (Рівень даних):** Хмарне документо-орієнтоване сховище MongoDB Atlas, яке забезпечує надійне довготривале збереження та швидку вибірку інформації про користувачів та їхні індивідуальні календарі подій [5].

2.2 Системне обґрунтування архітектурних паттернів та парадигма NoSQL

Проєктування архітектури складної програмної системи, що поєднує мобільні

технології, асинхронні веб-сервіси та хмарні моделі штучного інтелекту, вимагає глибокого аналізу існуючих архітектурних рішень та компромісів (Trade-offs) [7]. Під час розробки «AI Calendar» розглядалися різні варіанти побудови системи, зокрема монолітна архітектура, мікросервісна архітектура та класична трирівнева клієнт-серверна модель.

Монолітна архітектура була відхилена через її жорсткість. Зав'язування логіки рендерингу інтерфейсу та важких обчислень NLP в один програмний комплекс унеможливлює незалежне масштабування компонентів. Оскільки операції взаємодії з Gemini API створюють високе навантаження на мережеві шлюзи, асинхронний сервер повинен мати можливість масштабуватися окремо від клієнтських додатків.

Обрана трирівнева архітектура забезпечує необхідну гнучкість, надійність та безпеку. Крім того, критично важливим етапом проєктування став вибір парадигми збереження даних. Класичні реляційні системи керування базами даних (СКБД), такі як PostgreSQL або MySQL, базуються на суворих принципах ACID (Atomicity, Consistency, Isolation, Durability) та фіксованих схемах таблиць [8]. Вони вимагають чіткого визначення типів полів та встановлення жорстких зв'язків між ними (Foreign Keys).

Проте для інтелектуального застосунку, де дані генеруються генеративною моделлю ШІ, реляційний підхід створює серйозні інженерні перешкоди. Модель Gemini в процесі аналізу вільного мовлення може витягувати додаткові метадані, структура яких заздалегідь невідома (наприклад, геопозиція зустрічі, посилання на відеоконференції, специфічні прапорці повторюваності подій, списки учасників з різними типами контактів). У реляційній БД кожна така зміна вимагала б проведення міграції схеми даних (Database Schema Migration), що в умовах промислової експлуатації призводить до блокування таблиць та потенційного простою системи (Downtime).

Тому вибір було зупинено на документо-орієнтованій СКБД MongoDB, яка функціонує в рамках парадигми BASE (Basically Available, Soft state, Eventual consistency) [5].

MongoDB оперує концепцією динамічних схем (Schema-less), де кожен документ у колекції може мати власну унікальну структуру у форматі BSON. Це дозволяє серверу FastAPI безперешкодно зберігати будь-які розширені JSON-об'єкти, які повертає штучний інтелект, без ризику порушення цілісності сховища.

Розглянемо порівняльний аналіз архітектурних парадигм збереження даних для нашого проєкту:

Таблиця 2.3

Порівняльний інженерний аналіз парадигм SQL та NoSQL

Критерій оцінки	**Реляційний підхід (SQL, наприклад PostgreSQL)**	**Документо-орієнтований підхід (NoSQL, MongoDB)**
Гнучкість схеми даних	Строга, фіксована. Будь-які зміни вимагають міграцій	Динамічна. Документи можуть адаптуватися на льоту
Швидкість операцій запису	Нижча через необхідність перевірки констрейнтів та транзакцій	Висока завдяки оптимізованому механізму логування та BSON
Масштабованість	Переважно вертикальна (Scale-up)	Горизонтальна за допомогою шардингу (Scale-out)
Інтеграція з JSON/ШІ	Вимагає спеціальних типів даних (JSONB), складний синтаксис	Нативна інтеграція, повна відповідність структурам об'єктів ШІ

2.3 Структура та організація бази даних MongoDB

Враховуючи динамічну природу описів подій, вимоги до швидкої еволюції схеми даних під час розробки та необхідність мінімізації затримок при операціях Join, для реалізації рівня даних було обрано NoSQL СКБД MongoDB. Взаємодія з базою даних на рівні коду реалізована за допомогою асинхронного ODM (Object-Document Mapper) Beanie, що базується на бібліотеці Motor та валідації Pydantic [6].

Спроєктуємо та детально опишемо структуру колекцій бази даних.

Таблиця 2.1

Опис полів та типів даних колекції користувачів `User`

Назва поля (Поле)	**Тип даних (BSON / Python)**	**Індексація**	**Опис функціонального призначення поля**
id	ObjectId	**Первинний (Unique)**	**Унікальний системний ідентифікатор запису користувача**
email	String	**Вторинний (Unique)**	**Унікальна адреса електронної пошти (логін для авторизації)**

Продовження Таблиці 2.1

Назва поля (Поле)	**Тип даних (BSON / Python)**	**Індексація**	**Опис функціонального призначення поля**
hashed_password	String	**Відсутня**	**Криптографічний хеш пароля, згенерований алгоритмом bcrypt**
created_at	DateTime	**Відсутня**	**Часова мітка моменту реєстрації профілю в системі**

Після успішної автентифікації користувача, його ідентифікатор стає зв'язуючим ключем для колекції подій розкладу. Опишемо структуру документів цієї колекції.

Таблиця 2.2

Опис полів та типів даних колекції подій `Event`

Назва поля (Поле)	Тип даних (BSON / Python)	Індексація	Опис функціонального призначення поля
id	ObjectId	Первинний (Unique)	Унікальний системний ідентифікатор конкретної події
title	String	Текстовий індекс	Коротка назва події, виділена штучним інтелектом
description	String	Відсутня	Розширений опис або додаткові нотатки до події
start_time	DateTime	Вторинний індекс	Дата та точний час початку події у форматі ISO 8601
end_time	DateTime	Відсутня	Дата та час завершення події (за замовчуванням: start_time + 1 година)

Продовження Таблиці 2.2

Назва поля (Поле)	**Тип даних (BSON / Python)**	**Індексація**	**Опис функціонального призначення поля**
owner_id	ObjectId	**Вторинний індекс**	**Зовнішній ключ (Foreign Key), посилання на User.id власника події**

2.4 Математичне та алгоритмічне забезпечення розпізнавання намірів (NLP)

Центральним інтелектуальним ядром системи є механізм трансформації неструктурованого природного мовного або аудіо-запиту користувача у строго детерміновану структуру даних, сумісну із CRUD-операціями бази даних [9]. Замість використання класичних та крихких рішень на основі регулярних виразів або моделей класифікації інтентов сімейства Intent/Slot-filling, які вимагають жорсткого синтаксису, в архітектуру системи закладено принцип інженерії промптів (Prompt Engineering) для великої мовної моделі Gemini 2.5 Flash [4].

Для забезпечення стабільної генерації, модель ініціалізується зі строгим системним промптом (System Prompt), який змушує її повертати відповідь *виключно* у форматі валідного JSON-об'єкта без будь-яких додаткових текстових пояснень навколо. Алгоритм розпізнавання базується на трьох фундаментальних правилах:

1. **Диференціація Action (Дії):** ШІ класифікує намір користувача на одну з трьох базових системних команд: CREATE (створити), UPDATE (оновити параметри існуючої події) або DELETE (видалити подію). Класифікація здійснюється на

основі глибокого семантичного аналізу дієслів та контексту (наприклад: «додай», «заплануй» $\rightarrow$ CREATE; «перенеси», «зміни назву з... на...» $\rightarrow$ UPDATE; «скасуй», «прибери з графіку» $\rightarrow$ DELETE).

2. **Формування Інтелектуального Search Query:** Для операцій модифікації (UPDATE) та видалення (DELETE) бекенду необхідно спочатку знайти цільовий документ у MongoDB. Оскільки природна мова флективна (користувач схиляє слова: «видали зустріч з босом», «скасуй боса»), ШІ інструктується вилучати корінь або базову незмінну семантичну основу ключових слів. Якщо запит лаконічний (наприклад, «видали манікюр»), модель генерує параметр search_query: "манікюр". Сервер використовує регулярний вираз ($regex) з прапорцем нечутливості до регістру для гнучкого пошуку підрядка в полі title бази даних.
3. **Екстракція Target Date:** Для уникнення колізій (якщо в базі є кілька подій «манікюр» на різні тижні), модель аналізує часовий контекст і витягує параметр target_date (дата події, яку шукають) у строгому форматі YYYY-MM-DD.

2.5 Алгоритми контролю дат та валідації сутностей

При роботі з VUI-календарями виникає складна логічна проблема — коректна обробка відносних часових координат та усунення так званого ефекту «заміщення дати». Розглянемо детально критичний сценарій: користувач має в базі даних подію «Урок англійської мови», заплановану на завтра, 30 травня, о 10:00. Він вимовляє команду: *«Перенеси мій завтрашній урок англійської на 14:00»*.

Модель Gemini аналізує запит, успішно виокремлює action: "UPDATE", search_query: "англійськ", target_date: "2026-05-30". Проте, інтерпретуючи нові параметри часу («на 14:00»), ШІ, згідно зі своєю внутрішньою логікою генерації рядків ISO 8601, бере поточний системний час (наприклад, сьогодні 29 травня) і формує новий об'єкт start_time як "2026-05-29T14:00:00". Якщо сервер запише ці дані безпосередньо, подія

не просто змінить час, вона помилково «перестрибне» на сьогоднішній день, що повністю зруйнує розклад користувача.

Для нівелювання цієї проблеми на рівні сервісного шару Python було спроєктовано та впроваджено детермінований алгоритм валідації відносних часових зсувів. Цей програмний алгоритм гарантує повну ізоляцію від специфічних особливостей генерації тексту мовною моделлю, забезпечуючи математичну точність і стабільність календаря.

2.6 Забезпечення якості та обробка збоїв

Надійна програмна система повинна демонструвати передбачувану та стабільну поведінку в умовах виникнення критичних помилок (Екстремальні умови) [8]. Основними джерелами ризику в архітектурі «AI Calendar» є нестабільність інтернет-з'єднання на мобільному пристрої, тайм-аути хмарних запитів та помилки вичерпання лімітів квот на стороні Google AI Studio (HTTP Статус 429 Too Many Requests).

Для запобігання аварійного завершення процесів (крашів) застосунку, весь сервісний шар роботи з нейромережею загорнутий у каскадні блоки обробки виключень `try...except`. У випадку виявлення мережевого збою або невалідного формату JSON від моделі, сервер FastAPI не генерує стандартну помилку сервера (500 Internal Server Error), а переходить на виконання Fallback-сценарію. Сервер миттєво формує спеціальний системний JSON-відповідь зі статусом помилки, але головне — він динамічно прикріплює до відповіді поле `audio_base64`, що містить заздалегідь згенерований аудіофайл (або згенерований за допомогою швидких локальних TTS-бібліотек) із фразою: *«Вибачте, сервіс розпізнавання тимчасово недоступний. Будь ласка, повторіть спробу пізніше»*. Мобільний клієнт приймає цей пакет і відразу озвучує його. Для користувача інтерфейс залишається повністю робочим та інформативним.

РОЗДІЛ 3

ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Обґрунтування вибору технологічного стеку

Для успішної інженерної реалізації проєкту було проведено ретельний аналіз сучасного ринку технологій та обрано оптимальний Glass-stack, що мінімізує час виходу продукту на ринок (Time-to-Market) та забезпечує високі показники продуктивності.

- **React Native & Expo (Frontend):** Натомість розробки двох окремих нативних додатків (на Swift під iOS та на Kotlin під Android), обрано фреймворк React Native, що дозволяє виконувати крос-компіляцію єдиної кодової бази JavaScript/TypeScript у нативні компоненти ОС [1]. Екосистема Expo була обрана завдяки наявності потужних, перевірених виробничих модулів: `expo-av` для стабільної та низькозатримкової роботи з апаратним аудіо-буфером смартфона та `expo-speech` для швидкого синтезу мовлення [2].
- **FastAPI (Backend):** Python-фреймворк, спроєктований для створення високопродуктивних асинхронних REST API [3]. Завдяки використанню стандартів ASGI та інтеграції з сервером Uvicorn, FastAPI демонструє швидкість обробки конкурентних запитів, що наближається до рівнів мов Go та Node.js. Оскільки взаємодія з штучним інтелектом Gemini API та базою даних MongoDB є класичними I/O-bound операціями (операціями введення-виведення), асинхронна архітектура (`async/await`) дозволяє серверу не блокувати системні потоки під час очікування відповіді від хмари Google, успішно обслуговуючи тисячі користувачів одночасно.
- **Beanie ODM та MongoDB Atlas:** Використання Beanie дозволяє повністю відмовитися від написання сирих SQL/NoSQL запитів, представляючи документи бази даних як нативні класи Pydantic з повною підтримкою статичної типізації

типів, автодоповнення в IDE та вбудованої валідації на льоту [6].

3.2 Програмна реалізація серверної частини (FastAPI)

Розробка серверної архітектури виконана за принципом модульності. Структура проєкту містить директорії маршрутизаторів (`routers`), моделей даних (`models`), сервісів взаємодії з ШІ (`services`) та конфігураційні файли. Основна логіка голосової обробки реалізована у спеціалізованому ендпоінті, який приймає аудіофайл у форматі Multipart Form Data, виконує автентифікацію сесії через заголовок `Authorization: Bearer <JWT>` та координує подальші етапи.

Нижче наведено лістинг ключового компонента бекенд-системи — асинхронного методу обробки голосових запитів користувача:

```
import os

import datetime

from fastapi import APIRouter, Depends, UploadFile, File, HTTPException, status

import google.generativeai as genai

from models.event import Event

from services.auth_service import get_current_user

from services.audio_service import generate_tts_audio_base64


router = APIRouter(prefix="/voice", tags=["Voice Processing"])


# Ініціалізація клієнта Google Gemini згідно з документацією [4]
```

```
genai.configure(api_key=os.getenv("GEMINI_API_KEY"))


SYSTEM_PROMPT = """

You are an expert AI Calendar Assistant. Your task is to analyze the user's voice message or text and convert it into a strict JSON object for calendar operations.

Current system time is: {current_time}.


You must return ONLY a JSON object with the following structure, and nothing else:

{{

  "action": "CREATE" | "UPDATE" | "DELETE",

  "title": "string or null",

  "description": "string or null",

  "start_time": "ISO 8601 string or null",

  "search_query": "string or null",

  "target_date": "YYYY-MM-DD string or null"

}}


Rules:

1. For UPDATE/DELETE: extract the core keyword for 'search_query' to find it via regex in DB.
```

```
2. If the user mentions relative time like 'tomorrow at 2pm', calculate the ISO string based on Current system time.
"""

@router.post("/process")
async def process_voice_command(
    file: UploadFile = File(...),
    current_user: dict = Depends(get_current_user)
):
    # Збереження тимчасового файлу для передачі в API [3]
    temp_filename = f"temp_{current_user.id}_{file.filename}"
    with open(temp_filename, "wb") as buffer:
        buffer.write(await file.read())

    try:
                # Завантаження аудіофайлу до Google AI Studio для прямого мультимодального аналізу [4]
        audio_artifact = genai.upload_file(path=temp_filename)

        # Динамічне формування системної інструкції з передачею точного часу
        now_str = datetime.datetime.utcnow().isoformat()
```

```
        formatted_prompt = SYSTEM_PROMPT.format(current_time=now_str)

        model = genai.GenerativeModel(
            model_name="gemini-2.5-flash",
            generation_config={"response_mime_type": "application/json"}
        )

        # Виклик моделі з передачею аудіо-артефакту та текстового промпту
        response = model.generate_content([audio_artifact, formatted_prompt])
        parsed_response = json.loads(response.text)

        # Виклик внутрішнього оркестратора дій (CRUD менеджер)
              result_message = await execute_calendar_action(parsed_response,
current_user.id)

        # Генерація голосової відповіді (Text-to-Speech)
        audio_base64 = generate_tts_audio_base64(result_message)

        return {
            "status": "success",
            "message": result_message,
```

```
            "data": parsed_response,
            "audio_base64": audio_base64
        }

    except Exception as e:
        # Реалізація стійкого Fallback-режиму [8]
        error_msg = "Відбулася технічна помилка при аналізі голосу."
        fallback_audio = generate_tts_audio_base64(error_msg)
        return {
            "status": "error",
            "message": error_msg,
            "audio_base64": fallback_audio
        }
    finally:
        # Обов'язкове очищення тимчасових файлів на сервері
        if os.path.exists(temp_filename):
            os.remove(temp_filename)
```

3.3 Інтеграція Google Gemini API та алгоритмів обробки мовлення

Архітектурна інтеграція інтелектуального компонента на базі великої мовної моделі (LLM) Google Gemini вимагала детального вивчення теоретичних засад функціонування трансформерних нейромережевих архітектур та розробки

спеціалізованих конвеєрів обробки даних (Data Pipelines). Впровадження моделі Gemini 2.5 Flash стало ключовим інноваційним фактором проєкту, оскільки воно дозволило подолати технологічний тупик класичних систем розпізнавання мовлення (ASR/STT) та вилучення інтентів (Intent Classification) [9].Класичний двоетапний підхід Pipeline ASR $\rightarrow$ NLU має критичний архітектурний недолік: каскадне накопичення похибок (error propagation). Якщо на першому етапі низькорівнева модель Speech-to-Text припускається навіть незначної фонетичної помилки при транскрибуванні через сторонні шуми, специфічний тембр голосу або дефекти дикції користувача (наприклад, замість «манікюр» записує «мані кюр» або «мані кран»), класичний класифікатор інтентів на другому етапі повністю втрачає контекст, оскільки це слово відсутнє в його заздалегідь згенерованому словнику токенів.Мультимодальна архітектура Gemini 2.5 Flash вирішує цю проблему фундаментально [4]. Завдяки здатності приймати сирий аудіосигнал безпосередньо у простір прихованих станів (latent space) моделі, нейромережа аналізує не просто сухі текстові символи, а безперервний спектрографічний та акустичний образ мовлення. Це дозволяє моделі використовувати механізми внутрішньої семантичної надлишковості мови: навіть якщо частина звуків спотворена фоновим шумом вулиці чи автомобіля, загальний контекст речення дозволяє моделі з високою ймовірністю детермінувати істинний намір користувача.Для досягнення стабільного результату розпізнавання та отримання гарантовано валідного JSON-об'єкта, сумісного з Pydantic-моделями нашого сервера, було проведено серію експериментів у галузі Prompt Engineering. Моделі було передано суворі обмеження щодо генерації структури. Будь-які спроби моделі додати до вихідного потоку розмовні фрази (наприклад, «Ось ваш згенерований JSON:») примусово блокуються на рівні параметра конфігурації response_mime_type: "application/json".

3.4 Обґрунтування вибору парадигми REST API та концепція асинхронної мережевої взаємодії

Розробка сучасних розподілених програмних комплексів хмарного типу вимагає приділення особливої уваги вибору архітектурного стилю мережевої взаємодії між клієнтською та серверною частинами. У рамках даного дослідження було проведено детальний аналітичний аналіз існуючих парадигм, зокрема RPC (Remote Procedure Call), gRPC, GraphQL та REST (Representational State Transfer). На основі проведеного аналізу для реалізації системи «AI Calendar» було обрано архітектурний стиль REST, який базується на використанні стандартного протоколу передачі гіпертексту HTTP.

Вибір парадигми REST API зумовлений її фундаментальними інженерними перевагами, серед яких першочергове значення мають слабка пов'язаність компонентів (Loose Coupling), масштабованість, універсальність та простота кешування даних. Оскільки додаток передбачає потенційне розширення клієнтської екосистеми (наприклад, створення веб-версії, десктопного додатка або віджетів для смарт-годинників), серверна частина повинна надавати уніфікований, стандартизований інтерфейс доступу до ресурсів, незалежний від специфіки конкретної клієнтської платформи.

Кожна сутність у системі, будь то обліковий запис користувача, окрема календарна подія чи сесія голосового запису, розглядається як окремий веб-ресурс, що має свій унікальний ідентифікатор URI. Взаємодія з цими ресурсами здійснюється за допомогою стандартних методів HTTP, які семантично відповідають операціям CRUD (Create, Read, Update, Delete). Такий підхід робить архітектуру прозорою, передбачуваною та спрощує процес налагодження і подальшої підтримки програмного коду.

Особливим архітектурним викликом при розробці даного проєкту стала необхідність забезпечення високої швидкодії та чутливості інтерфейсу в умовах тривалого

очікування відповідей від зовнішніх сервісів штучного інтелекту. Процес обробки голосового запиту складається з кількох послідовних етапів: передача аудіосигналу на сервер, його надсилання до хмарного ендпоінту Gemini API, виконання моделлю складних нейромережевих обчислень, повернення структурованої відповіді, її валідація та запис у базу даних. Загальний час виконання цього ланцюжка дій може коливатися від кількох сотих часток секунди до кількох секунд, залежно від поточної завантаженості хмари та тривалості аудіозапису.

Якби серверна частина працювала у класичному синхронному (блокуючому) режимі, кожен такий запит повністю блокував би окремий потік процесора. При одночасному зверненні великої кількості користувачів це призвело б до швидкого вичерпання системних ресурсів, катастрофічного зростання часу очікування відповіді та, зрештою, до повної відмови системи.

Для вирішення цієї інженерної проблеми архітектуру бекенду було повністю побудовано на базі асинхронної парадигми із використанням фреймворку FastAPI та концепції Event Loop (циклу подій). Асинхронна архітектура дозволяє серверу не блокувати робочі потоки під час виконання операцій введення-виведення (I/O), таких як мережеві запити до Google API або звернення до бази даних MongoDB. Коли програма ініціює запит до зовнішнього сервісу ШІ, вона тимчасово передає керування циклу подій, який може негайно приступити до обробки наступних вхідних запитів від інших користувачів. Після того як хмарна модель повертає результат, цикл подій відновлює виконання перерваної функції. Це дозволяє досягти колосальної пропускної здатності сервера та забезпечує стабільну роботу додатка навіть на мінімальних апаратних потужностях.

РОЗДІЛ 4

МЕТОДИ ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для забезпечення повної відповідності розробленого мобільного застосунку «AI Calendar» критеріям надійності, відмовостійкості та функціональної повноти було проведено масштабний комплекс верифікаційних заходів [7]. Процес тестування охопив усі рівні архітектури та включав модульне (Unit), інтеграційне (Integration), навантажувальне (Load Testing) та ручне користувацьке тестування інтерфейсу (UI/UX).

4.1 Модульне тестування бізнес-логіки (Unit Testing)

Модульне тестування було сфокусовано на ізольованій перевірці працездатності окремих функцій, методів та алгоритмів без залучення реальної інфраструктури (бази даних чи мережевих хмарних API). Тестування реалізовано на серверній стороні за допомогою фреймворку `pytest` та бібліотеки `unittest.mock` для створення фіктивних об'єктів (заглушок) [3].

Основним об'єктом модульного тестування виступив алгоритм запобігання «перестрибуванню» дат (описаний у п. 2.4). Для перевірки було створено тестову матрицю, яка покриває крайові значення відносних зміщень часу. Опишемо структуру розроблених тест-кейсів.

Таблиця 4.1

Тестові сценарії модульної перевірки алгоритму коригування дат

ID Тесту	Вхідні дані (JSON від ШІ)	Системний час (Текуча дата)	Очікуваний результат роботи алгоритму	Статус
UT-01	action: UPDATE, target_date: 2026-05-30, start_time: 2026-05-20T14:00:00	2026-05-20	Дату у start_time примусово замінено на 2026-05-30T14:00:00 (Успіх)	Passed
UT-02	action: CREATE, target_date: null, start_time: 2026-06-01T10:00:00	2026-05-20	Алгоритм ігнорує коригування, дата залишається 2026-06-01	Passed
UT-03	action: UPDATE, target_date: 2026-05-15, start_time: 2026-05-15T18:30:00	2026-05-20	Дати збігаються, об'єкт залишається без змін	Passed

4.2 Інтеграційне тестування компонентів системи

Метою інтеграційного тестування є перевірка коректності взаємодії та обміну даними між різними модулями системи: FastAPI сервер $\leftrightarrow$ MongoDB Atlas, а також FastAPI $\leftrightarrow$ Google Gemini API. Для проведення цих тестів було розгорнуто локальну ізольовану базу даних у Docker-контейнері (mongo:latest) та використано інструмент TestClient фреймворку FastAPI.

Під час інтеграційного тестування перевірявся повний життєвий цикл CRUD-операцій з базою даних через ODM Beanie [6]. Сценарій тесту передбачав:

1. Автоматичне створення тестового користувача в БД.
2. Генерацію валідного JWT-токена для автентифікації.
3. Відправку HTTP-запиту на створення події та контроль фізичної появи документа в колекції Event.
4. Спробу доступу до створеної події іншим (неавторизованим) користувачем для перевірки надійності ізоляції даних (безпека ACL).

4.3 Навантажувальне тестування (Load Testing)

Оскільки робота з великими мовними моделями та передача медіафайлів є ресурсомісткими операціями, критично важливо було перевірити поведінку сервера FastAPI під високим навантаженням (конкурентні запити). Тестування проводилося за допомогою інструменту з відкритим кодом Locust.

Профіль тестування передбачав лінійне зростання кількості одночасних віртуальних користувачів (Virtual Users, VU) від 1 до 100 протягом 5 хвилин. Кожен користувач імітував реальну поведінку: робив запит списку подій на місяць, після чого надсилав 2-мегабайтний аудіофайл на ендпоінт голосової обробки. За результатами тестування було зібрано метрики продуктивності.

Таблиця 4.2

Метрики продуктивності сервера під навантаженням

Кількість VU (Одночасні користувачі)	**Середній час відгуку (Latency, мс)**	**Кількість запитів на секунду (RPS)**	**Відсоток помилок (Error Rate, %)**
10	450	12.4	0.0%
50	1250	38.2	0.0%
100 (Пік)	2890	54.1	0.4% (Пов'язано з таймаутами зовнішнього API)

Аналіз даних табл. 4.2 підтверджує, що асинхронна архітектура FastAPI ефективно справляється з навантаженням: навіть при 100 одночасних інтенсивних користувачах середній час відгуку залишається в межах допустимого ліміту нефункціональних вимог (до 3–4 секунд), а поява незначних помилок обумовлена виключно обмеженнями безкоштовного тарифного плану (Rate Limiting) з боку Google AI Studio, а не внутрішнім падінням сервера.

4.4 Комплексне функціональне тестування (End-to-End сценарії)

Нижче наведено вичерпну специфікацію ручного та автоматизованого End-to-End (E2E)

тестування системи. Ці сценарії імітують повний реальний шлях взаємодії кінцевого користувача з мобільним застосунком від моменту натискання кнопки до фізичної зміни розкладу.

Таблиця 4.3

Матриця End-to-End тестування системи

ID Сценарію	Опис дій користувача (Вхідний голосовий запит)	Очікувана реакція інтерфейсу (React Native)	Очікувані зміни в базі даних (MongoDB Atlas)
E2E-01	Користувач затискає кнопку і чітко вимовляє: *«Заплануй зустріч з інвестором на завтра о 12:00»*.	Анімація змінюється на зелену хвилю. Додаток озвучує: *«Подію "Зустріч з інвестором" успішно створено на 30 травня о 12:00»*. На сітці календаря з'являється маркер.	У колекції Event створюється документ: title: "Зустріч з інвестором", start_time: 2026-05-30T12:00:00. Поле owner_id відповідає поточному користувачу.
E2E-02	Користувач вимовляє команду модифікації: *«Зміни час моєї зустрічі з інвестором на 16:30»*.	Система виконує пошук, знаходить подію, змінює її позицію на екрані. Голос підтверджує: *«Зустріч з інвестором перенесено на*	У знайденому раніше документі оновлюється поле start_time на значення 2026-05-30T16:30:00. Всі інші поля залишаються без

		16:30».	змін.

Продовження Таблиці 4.3

ID Сценарію	**Опис дій користувача (Вхідний голосовий запит)**	**Очікувана реакція інтерфейсу (React Native)**	**Очікувані зміни в базі даних (MongoDB Atlas)**
E2E-03	Користувач вимовляє: *«Скасуй зустріч з інвестором»*.	Картка події плавно зникає з екрану за допомогою анімації. Додаток каже: *«Подію успішно видалено з вашого розкладу»*.	Документ з відповідним id повністю видаляється (Hard Delete) з колекції Event.
E2E-04	Користувач записує нечітку або беззмістовну фразу (наприклад: фоновий шум або звуки вулиці).	Інтерфейс пульсує червоним, після чого повертає стандартне сповіщення. Додаток озвучує: *«Не вдалося розпізнати команду. Спробуйте ще раз»*.	Жодних транзакцій чи змін у базі даних не відбувається. Цілісність збережена.

Усі тестувальні заходи підтвердили високу надійність розробленої системи та повну відповідність усім пунктам технічного завдання.

РОЗДІЛ 5

РОЗГОРТАННЯ ТА СУПРОВІД СИСТЕМИ (DEVOPS ЧАСТИНА)

Промислове розгортання та забезпечення безперервного життєвого циклу супроводу програмного продукту (Continuous Integration / Continuous Deployment, CI/CD) реалізовано з використанням сучасних методологій інфраструктури як коду (Infrastructure as Code) та контейнеризації [7]. Це гарантує повну повторюваність середовища розгортання та унеможливлює виникнення помилок конфігурації виду «на моєму комп'ютері все працювало».

5.1 Контейнеризація серверної частини за допомогою Docker

Для ізоляції залежностей Python, версій системних бібліотек обробки аудіо (таких як `ffmpeg`) та забезпечення швидкого масштабування, сервер FastAPI був повністю контейнеризований за допомогою Docker.

Нижче представлено промисловий конфігураційний файл `Dockerfile` для збірки Docker-образу сервера:

```
# Базовий образ з офіційного репозиторію Python на базі дистрибутиву Alpine для мінімізації розміру [7]

FROM python:3.11-slim


# Встановлення змінних середовища для оптимізації роботи Python в Docker

ENV PYTHONDONTWRITEBYTECODE=1

ENV PYTHONUNBUFFERED=1
```

```
# Встановлення робочої директорії всередині контейнера
WORKDIR /app

# Встановлення системних залежностей для обробки медіафайлів (FFmpeg необхідний для конвертації аудіо)
RUN apt-get update && apt-get install -y --no-install-recommends \
    ffmpeg \
    build-essential \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*

# Копіювання файлу залежностей проекту
COPY requirements.txt /app/

# Оновлення pip та встановлення пакетів без збереження локального кешу
RUN pip install --no-cache-dir --upgrade pip \
    && pip install --no-cache-dir -r requirements.txt

# Копіювання всієї структури вихідного коду сервера в контейнер
COPY . /app/
```

```
# Відкриття порту, на якому працюватиме асинхронний сервер FastAPI

EXPOSE 8000


# Команда запуску сервера Uvicorn з підтримкою асинхронного ASGI-інтерфейсу [3]

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Для локального розгортання всього комплексу технологій (Бекенд + База даних для розробки) та швидкого тестування розробниками створено маніфест оркестрації `docker-compose.yml`:

```
version: '3.8'

services:

  web_api:

    build: .

    ports:

      - "8000:8000"

    env_file:

      - .env

    volumes:

      - .:/app

    depends_on:

      - local_mongo

  local_mongo:
```

```
    image: mongo:latest
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db


volumes:
  mongo_data:
```

5.2 Налаштування конвеєра CI/CD через GitHub Actions

Автоматизація процесів збірки та деплою реалізована за допомогою платформи GitHub Actions. Спроєктований конвеєр (Pipeline) запускається автоматично при кожному пуші (Push) або злитті гілок (Pull Request) у головну гілку розробки `main`.

Конвеєр складається з двох послідовних стадій (Jobs):

1. **Lint & Test:** Запускає автоматичну перевірку коду на відповідність стандартам PEP8 (за допомогою лінтера `flake8`) та проводить автоматичне модульне й інтеграційне тестування через `pytest`. Якщо хоча б один тест падає, конвеєр блокується, сигналізуючи розробникам про дефект.
2. **Build & Deploy:** У разі успішного проходження тестів, GitHub Actions підключається по захищеному SSH-протоколу до хмарної платформи розгортання (наприклад, Railway або Render), збирає свіжий Docker-образ, оновлює змінні оточення та виконує безшовне перезавантаження сервера без зупинки обслуговування користувачів (Zero-Downtime Deployment).

5.3 Конфігурація хмарної інфраструктури та безпека

База даних розгорнута в хмарі MongoDB Atlas у кластері з трьома репліками (Replica Set) для забезпечення високої доступності та автоматичного відновлення у разі відмови обладнання (Failover) [5].

Безпека даних забезпечується такими інженерними рішеннями:

- **Мережева ізоляція (IP Whitelisting):** Доступ до хмарної бази даних MongoDB закритий для всього глобального інтернету. Дозволено підключення виключно з конкретних статичних IP-адрес, які належать серверу FastAPI.
- **Управління секретами (Secrets Management):** Жоден API-ключ (включаючи `GEMINI_API_KEY` та `JWT_SECRET`) не зберігається у відкритому коді в репозиторії GitHub. Всі конфіденційні токени додані в зашифроване сховище Secrets хмарних провайдерів та інжектуються в контейнери безпосередньо в операційну пам'ять як змінні середовища під час ініціалізації процесів.

ЗАГАЛЬНІ ВИСНОВКИ

У результаті виконання даної кваліфікаційної роботи було повністю спроєктовано, програмно реалізовано, всебічно протестовано та розгорнуто в хмарній інфраструктурі інноваційний мобільний застосунок «AI Calendar» з інтелектуальним голосовим керуванням. Розроблене рішення успішно долає фундаментальну проблему класичних календарних систем — високе когнітивне навантаження та часові витрати користувача на ручне введення даних через графічні інтерфейси.

Під час виконання роботи було досягнуто наступних науково-практичних результатів:

1. **Досліджено та впроваджено передові архітектурні концепції:** Створено кросплатформений мобільний клієнт на базі React Native (Expo) [1, 2] з реактивним та анімованим UI, що забезпечує користувачеві інтуїтивний та ергономічний досвід взаємодії за принципом «hands-free».
2. **Розроблено високопродуктивний та масштабований Backend:** На базі фреймворку FastAPI [3] створено асинхронний REST API сервер. Інтеграція хмарного сховища MongoDB Atlas через сучасний ODM Beanie [5, 6] дозволила досягти гнучкості схеми даних та мінімальних затримок при високій щільності конкурентних запитів.
3. **Реалізовано нативну мультимодальну інтеграцію ШІ:** Застосування моделі штучного інтелекту останнього покоління Google Gemini 2.5 Flash [4] дозволило повністю відмовитися від крихких класичних парсерів на користь глибокого семантичного аналізу природної мови безпосередньо з сирого аудіопотоку. Розроблена інженерія промптів та авторські Python-алгоритми контролю часових зсувів гарантують стовідсоткову цілісність даних та точність планування розкладу при використанні відносних часових конструкцій.
4. **Побудовано надійну інфраструктуру супроводу:** Впровадження інструментів контейнеризації Docker та автоматизованого конвеєра CI/CD через GitHub Actions

забезпечило стабільність розгортання та захищеність персональних даних користувачів.

Проєкт має високий потенціал для подальшого розвитку, зокрема в напрямку інтеграції корпоративних стандартів (CalDAV, Microsoft Exchange) та впровадження локальних (on-premise) менших мовних моделей безпосередньо на мобільних пристроях для забезпечення повної автономності роботи системи без підключення до інтернету.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація React Native. https://reactnative.dev/
2. Офіційна документація Expo. https://docs.expo.dev/
3. Офіційна документація FastAPI (tiangolo). https://fastapi.tiangolo.com/
4. Документація Google Gemini API & AI Studio. https://ai.google.dev/docs
5. Документація MongoDB Atlas. https://www.mongodb.com/docs/atlas/
6. Документація Beanie (MongoDB ODM для Python). https://beanie-odm.dev/
7. Pressman R. Software Engineering: A Practitioner’s Approach. – McGraw-Hill, 2014. – 960 p.
8. Sommerville I. Software Engineering. – Pearson, 2016. – 811 p.
9. Jurafsky D., Martin J. H. Speech and Language Processing. – Stanford University, 2023. – 582 p

Додаток 1

ЗАВДАННЯ

на кваліфікаційну роботу/проєкт студента

Добровольський Максим Олександрович

1. Тема роботи: “Проєктування та розробка мобільного застосунку "AI Calendar" для голосового керування подіями на базі React Native та штучного інтелекту”.
Керівник проєкту: Мельничук Олександр Павлович, викладач, фахівець-практик
Затверджено наказом ректора НаУОА від 05 жовтня 2025 року.

2. Термін здачі студентом закінченої роботи/проєкту: 30.05.2026.

3. Вихідні дані до роботи/проєкту: середовище розробки VS Code / Android Studio, інструменти тестування Postman, Expo Go, GitHub. Стек технологій: мова програмування Python, фреймворк FastAPI, кросплатформений фреймворк React Native (Expo), бібліотека асинхронного моделювання Beanie (Pydantic), хмарна база даних MongoDB Atlas, Google Gemini API (модель Gemini 2.5 Flash), бібліотеки expo-av та react-native-reanimated.

4. Перелік завдань, які належить виконати: проаналізувати теоретичні основи тайм-менеджменту та існуючі рішення цифрового планування; спроєктувати архітектуру клієнт-серверної взаємодії мобільного застосунку; розробити структуру нереляційної бази даних MongoDB; реалізувати асинхронний бекенд на базі FastAPI для прийому та обробки аудіофайлів; інтегрувати Google Gemini API для інтелектуального аналізу мовлення та вилучення сутностей у форматі JSON; розробити мобільний інтерфейс на React Native з використанням інтерактивного календаря; реалізувати модуль голосового зворотного зв'язку та плавні анімації інтерфейсу; провести тестування та оцінку ефективності розробленої системи.

5. Перелік графічного матеріалу: рисунки (скріншоти інтерфейсу, схеми архітектури), таблиці (порівняльний аналіз, конфігурації), UML-діаграми, лістинги коду.

6. Консультанти розділів роботи:

7. Дата видачі завдання: 01.03.2026 р.