

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

ПОГОДЖЕНО

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Декан факультету

Завідувач кафедри комп'ютерних наук

_____ Ігор БОЛБОТ

_____ Белла ГОЛУБ

(підпис)

(підпис)

« ____ » _____ 2026 р.

« ____ » _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: “Android–додаток для планування та контролю виконаних завдань”

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо–професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми

К.Т.Н., доцент

Ганна ВАЙГАНГ

(підпис)

**Керівник бакалаврської
кваліфікаційної роботи,**

асистент

Віктор БОРОВИК

(підпис)

**Консультант бакалаврської
кваліфікаційної роботи**

К.Т.Н., доцент

Юлія БОЯРІНОВА

(підпис)

Виконав

Артем СТУПНИЦЬКИЙ

(підпис)

Київ–2026

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ

(підпис)

«__» _____ 2025р.

**ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ**

Ступніцькому Артему Олександровичу

(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

“Android-додаток для планування та контролю виконаних завдань”

затверджена наказом від «19» _____ грудня _____ 2025 р. № 3204 «С»

Термін подання завершеної роботи на кафедру «__» _____ 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи:

опис організації та змісту роботи Android-додатку для планування, створення та контролю виконання завдань із підтримкою ієрархічної структури та AI-консультацій.

Перелік питань, що підлягають дослідженню:

Android-додаток для планування та контролю завдань як об'єкт дослідження, проєктування програмного забезпечення, розробка Android-додатку.

Дата видачі завдання «__» _____ 2025 р.

**Керівник бакалаврської
кваліфікаційної роботи**

асистент

(підпис)

Віктор БОРОВИК

**Консультант бакалаврської
кваліфікаційної роботи**

к.т.н., доцент

(підпис)

Юлія БОЯРІНОВА

Завдання взяв до виконання

(підпис)

Артем СТУПНІЦЬКИЙ

ЗМІСТ

ВСТУП.....	4
Актуальність завдання, що вирішується	4
Мета розробки програмного додатку	5
Методи та технології	5
Апробація програмного додатку	6
Структура пояснювальної записки	8
1. Системний аналіз предметної області	9
1.1 Опис предметної області.....	9
1.2 Аналіз вимог до програмної системи	10
1.3 Моделювання предметної області.....	12
1.4 Огляд інформаційних джерел та існуючих рішень	18
1.5 Постановка завдання	19
2. Проектування інформаційного та програмного забезпечення.....	21
2.1 Логічна модель даних у вигляді ER–діаграми	22
2.2 Діаграма класів та кооперацій	24
2.3 Діаграма пакетів	28
2.4 Діаграма компонентів.....	29
3. Розробка інформаційного та програмного забезпечення	32
3.1 Система управління інформаційною базою	32
3.2 Розробка інформаційної бази	34
3.3 Вибір інструментарію для створення прикладного програмного забезпечення	41
3.4 Алгоритмізація та програмування програмних модулів.....	45
4. Рекомендації щодо впровадження та експлуатації системи	49
4.1 Тестування системи.....	49
4.2 Вимоги до апаратного та програмного забезпечення	55
4.3 Склад інсталяційного пакету	58
ВИСНОВКИ	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	63
ДОДАТОК А	64
ДОДАТОК Б	66

ВСТУП

Актуальність завдання, що вирішується

У нашому сучасному світі кожна людина щодня стикається з дуже великою кількістю інформації, задач та обов'язків. Робота, навчання, особисті справи, розвиток, відпочинок – усе це дуже сильно потребує чіткого планування та контролю. В умовах постійного інформаційного навантаження пам'ять вже не може виступати надійним інструментом для збереження важливих завдань і дедлайнів. Через це у користувачів виникає потреба у зручних, доступних і ефективних цифрових інструментах, які допомагають організувати діяльність людини, допомагають в роботі.

Ця проблема є дуже поширеною серед таких людей як: студенти, фахівці різних галузей, офісні працівники, підприємці а також люди, котрі потребують контролю над багатозадачністю.

Невиконані або забуті завдання можуть призводити до втрати часу, зниження ефективності роботи та навіть до фінансових втрат. Саме тому є дуже корисним використання спеціалізованих додатків для того, щоб планувати та контролювати виконання задач. І це не тільки про зручність, в першу чергу це про необхідність. Що є дуже важливим для користувачів, тому що якісне планування та контроль виконаних завдань є дуже корисним інструментом для організації процесу.

Розвиток мобільних технологій відкрив нові можливості для вирішення цієї проблеми. Смартфон сьогодні є невід'ємною частиною життя кожної людини, і саме він є основним інструментом для управління задачами. Мобільні додатки дозволяють швидко створювати, редагувати та відслідковувати завдання у будь-який момент часу, незалежно від місця перебування нашого користувача. Однак, незважаючи на велику кількість існуючих рішень, не всі вони відповідають потребам користувачів: одні є перевантаженими функціоналом,

інші – занадто простими, треті – незручними у використанні або не мають необхідної гнучкості. Тому дуже важливо було створити такий додаток, який буде функціональним і виконувати свою функцію, але дуже простим і зрозумілим для користувача.

Саме із-за цих причин розробка власного додатку на системі Android, для планування та контролю виконаних завдань, є дуже актуальною. Такий додаток може бути адаптований під конкретні потреби користувачів і забезпечувати ефективну організацію, планування та контроль їхньої діяльності.

Мета розробки програмного додатку

Метою даної роботи є розробка мобільного Android-додатку, який дозволяє користувачеві створювати список завдань та задач, встановлювати для них параметри (опис, терміни виконання, статус), а також здійснювати контроль їх виконання. І найбільша перевага цього додатку – це впроваджений штучний інтелект, завдяки якому людина може зручніше користуватись додатком контролювати та доповнювати завдання. Додаток має забезпечувати інтуїтивно зрозумілий інтерфейс для кожного користувача, швидкий доступ до інформації та можливість збереження даних для подальшого використання.

Для розробки цього додатку нам необхідно проаналізувати потреби користувачів, визначити основний функціонал додатку, спроектувати інтуїтивно зрозумілий та візуально привабливий інтерфейс, реалізувати логіку роботи із задачами, забезпечити збереження та синхронізацію даних, а також протестувати правильність роботи додатку.

Методи та технології

Методи та технології, що використовувалися у процесі розробки, включають сучасні підходи до створення мобільних додатків. Основним середовищем розробки було використано популярний фреймворк Android Studio – це офіційний інструмент для створення зручних та оптимізованих Android-

додатків, який надає широкий набір засобів та інструментів для програмування, тестування та налагодження.

Мова програмування Java була обрана як одна з найбільш стабільних та поширених мов для розробки Android–додатків. Вона забезпечує надійну роботу програми, підтримує об’єктно–орієнтований підхід та дозволяє ефективно організувати код. Завдяки цій мові програмування ми можемо забезпечити надійність та високу продуктивність нашого додатку, що є дуже важливим аспектом під час розробки.

Для створення інтерфейсу користувача використовувалася мова розмітки XML, яка дозволяє гнучко налаштовувати вигляд елементів, їх розташування та взаємодію. Завдяки цьому вдалося створити зручний та зрозумілий дизайн додатку, який не перевантажує користувача зайвою інформацією.

Також в проєкті ми використовуємо Firebase – це платформа яка забезпечує в нашому додатку зберігання даних у реальному часі, автентифікацію користувачів для того щоб прикріплювати інформацію за певним користувачем та інші дуже корисні функції. Завдяки Firebase додаток може зберігати інформацію про завдання, забезпечувати доступ до них з різних пристроїв та гарантувати безпеку даних, що є дуже важливим аспектом в розробці програмного забезпечення.

Таким чином, розробка Android–додатку для планування та контролю виконаних завдань є актуальною та практично важливою задачею, яка спрямована на підвищення ефективності організації діяльності користувачів.

Завдяки сучасним технологіям, цей додаток може використовуватись в різних сферах життя та під різні потреби користувачів.

Апробація програмного додатку

Апробація створеного програмного додатку проводилася не лише формально, а через його реальне використання в умовах, максимально

наближених до повсякденної роботи користувача. Основна ідея полягала в тому, щоб перевірити, як додаток поводить себе не “в теорії”, а на практиці – під час виконання звичних задач, з якими стикається людина щодня.

Під час апробації увага приділялася не тільки на технічну сторону, але й загальному досвіду взаємодії з додатком. Проводилась оцінка наскільки зручно працювати з інтерфейсом, також чи логічно побудована структура, чи легко створювати, редагувати та контролювати виконання завдань.

Окремо перевірялася стабільність роботи додатку, коректність реалізації основних функцій та відсутність критичних помилок під час використання. Таким чином, апробація дозволила оцінити якість розробленого рішення, визначити його сильні сторони та, за потреби, виявити місця, які можуть бути вдосконалені для досягнення кращого користувацького досвіду.

У процесі апробації додаток використовувався для створення, редагування та контролю виконання повсякденних завдань. Було перевірено основні функціональні можливості, зокрема:

- додавання нових задач із зазначенням термінів виконання;
- редагування та видалення завдань;
- зміна статусу виконання;
- збереження та завантаження даних;
- робота інтегрованого штучного інтелекту для допомоги користувачу у формуванні та оптимізації задач.

Тестування проводилося на різних мобільних пристроях із операційною системою Android, що дозволило перевірити стабільність роботи додатку на різних пристроях та версіях ОС. У ході перевірки було виявлено та усунуто незначні помилки інтерфейсу та логіки роботи програми.

Результати апробації показали, що розроблений додаток є зручним у використанні, має інтуїтивно зрозумілий інтерфейс та забезпечує ефективний

контроль виконання завдань. Додаток може бути використаний як у навчальній діяльності, так і в повсякденному житті користувачів.

Структура пояснювальної записки

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків та списку використаних джерел. У вступі обґрунтовано актуальність теми, сформульовано мету роботи, описано використані методи та технології, наведено результати апробації додатку. У першому розділі виконано системний аналіз предметної області, проаналізовано вимоги до системи, побудовано UML–діаграми та проведено огляд існуючих рішень. У другому розділі виконано проєктування інформаційного та програмного забезпечення: побудовано ER–діаграму, діаграми компонентів і пакетів, описано архітектуру системи та структуру бази даних. У третьому розділі описано програмну реалізацію основних модулів додатку, архітектуру застосунку та розробку інтерфейсу користувача. У четвертому розділі висвітлено тестування системи та визначено вимоги до апаратного і програмного забезпечення. У висновках підведено підсумки роботи та окреслено перспективи подальшого розвитку системи.

1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметну область цієї роботи – це організація особистих справ, планування часу і відстеження того, що вже зроблено, а що ще чекає своєї черги.

Проблема насправді банальна, але від цього не менш реальна. Кожен день людина має справу з купою різних обов'язків – і на роботі, і в навчанні, і в особистому житті.

Студенти тримають у голові дедлайни і розклад пар, підприємці – зустрічі і задачі команди, офісні працівники – доручення керівника і власні справи. І всі вони час від часу забувають щось важливе. Не тому що безвідповідальні, а просто тому що людська пам'ять на таке не розрахована.

Блокнот і стікери колись рятували, але зараз це вже поганий варіант контролю задач. По–перше, їх легко загубити. По–друге, вони не нагадують, не сортують і не показують що з чого пріоритетніше. Автоматизації – нуль.

Смартфон у цьому плані куди практичніший. Він завжди в кишені, завжди онлайн, і може нагадати про задачу рівно тоді коли треба. Тому мобільні додатки для планування зайняли таку помітну нішу – вони закривають реальну потребу, яка є у більшості людей.

У межах цієї роботи розглядається саме такий інструмент: додаток де можна створювати задачі, редагувати їх, виставляти пріоритети, відмічати що виконано і переглядати історію. Окремою перевагою є вбудований штучний інтелект – він допомагає користувачу доповнювати завдання та контролювати процеси, що робить роботу з додатком ще зручнішою. При цьому дуже важливо щоб інтерфейс не перевантажував – якщо додаток складніший за саму задачу яку він має вирішити, то сенс у ньому зникає.

Предметна область цієї роботи – це не лише код і архітектура, а й розуміння реальних потреб користувача при роботі з таким додатком.

1.2 Аналіз вимог до програмної системи

Перш ніж приступати до проектування чогось конкретного, треба було зрозуміти – а що взагалі має робити цей додаток? Які функції обов'язкові, а без яких цілком можна обійтись? Це і є аналіз вимог, і в даному випадку він виявився важливішим кроком ніж здавалось на початку.

Саме на цьому етапі ми формуємо розуміння того, що саме має бути в нашому додатку, які задачі він повинен вирішувати та яким очікуванням відповідати

Архітектура даного додатку побудована навколо трирівневої ієрархії: список завдань – завдання – підзадачі. Це не випадково – така структура відображає реальний спосіб мислення людини при плануванні. Спершу людина думає категоріями або проєктами, потім конкретними задачами, а потім вже кроками для їх виконання. Саме цю логіку і відтворює додаток.

У межах даної роботи вимоги до програмної системи можна умовно поділити на функціональні та нефункціональні.

Функціональні вимоги:

- Реєстрація та авторизація користувача через Firebase Authentication. Всі дані прив'язуються до акаунту – при повторному вході або переході на інший пристрій інформація зберігається.
- Створення списків завдань. Користувач може мати декілька окремих списків – наприклад, «Робота», «Навчання», «Особисте». Кожен список має назву і може бути відредагований або видалений.
- Додавання завдань до списку. Кожне завдання має назву, опис, дату виконання і пріоритет. При створенні завдання користувач може скористатись ШІ-підказкою – штучний інтелект генерує назву і опис на основі короткого введеного тексту або ключових слів.
- Підзадачі. Кожне завдання може мати список підзадач – дрібніших кроків для його виконання. Підзадачі можна додавати, редагувати, видаляти і відмічати як виконані незалежно одна від одної.

- Відмітка виконання. І завдання, і підзадачі можна позначати як виконані.
- Редагування і видалення. Будь-який елемент ієрархії – список, завдання або підзадача – можна відредагувати або видалити у будь-який момент.
- Вкладка з ШІ-асистентом. Окремий розділ додатку де користувач може вільно спілкуватись зі штучним інтелектом: ставити питання, просити поради щодо планування, уточнювати деталі задач. Це повноцінний чат-інтерфейс.
- Збереження і синхронізація даних через Firebase Realtime Database. Всі зміни зберігаються в хмарі в реальному часі.

Нефункціональні вимоги

- Продуктивність – перехід між екранами і завантаження списків мають відбуватись без затримок. Робота з Firebase виконується асинхронно щоб не блокувати інтерфейс.
- Зручність – ієрархічна структура список → завдання → підзадачі має бути зрозумілою без пояснень. Навігація між рівнями – інтуїтивна, з можливістю швидко повернутись назад.
- Надійність – дані не губляться при закритті додатку, відсутності інтернету або перезавантаженні пристрою. Firebase забезпечує локальний кеш для офлайн-режиму.
- Безпека – правила безпеки Firebase налаштовані так, що кожен користувач бачить тільки свої дані. Читання чужих списків і завдань неможливе.
- Сумісність – коректна робота на пристроях з Android 8.0 і вище

1.3 Моделювання предметної області

Моделювання дозволяє описати систему до початку її реалізації – побачити структуру даних, логіку взаємодії і можливі проблеми ще до того як написано перший рядок коду. В даній роботі використовуються кілька типів діаграм.

1.3.1 Use Case діаграма (діаграма прецедентів).

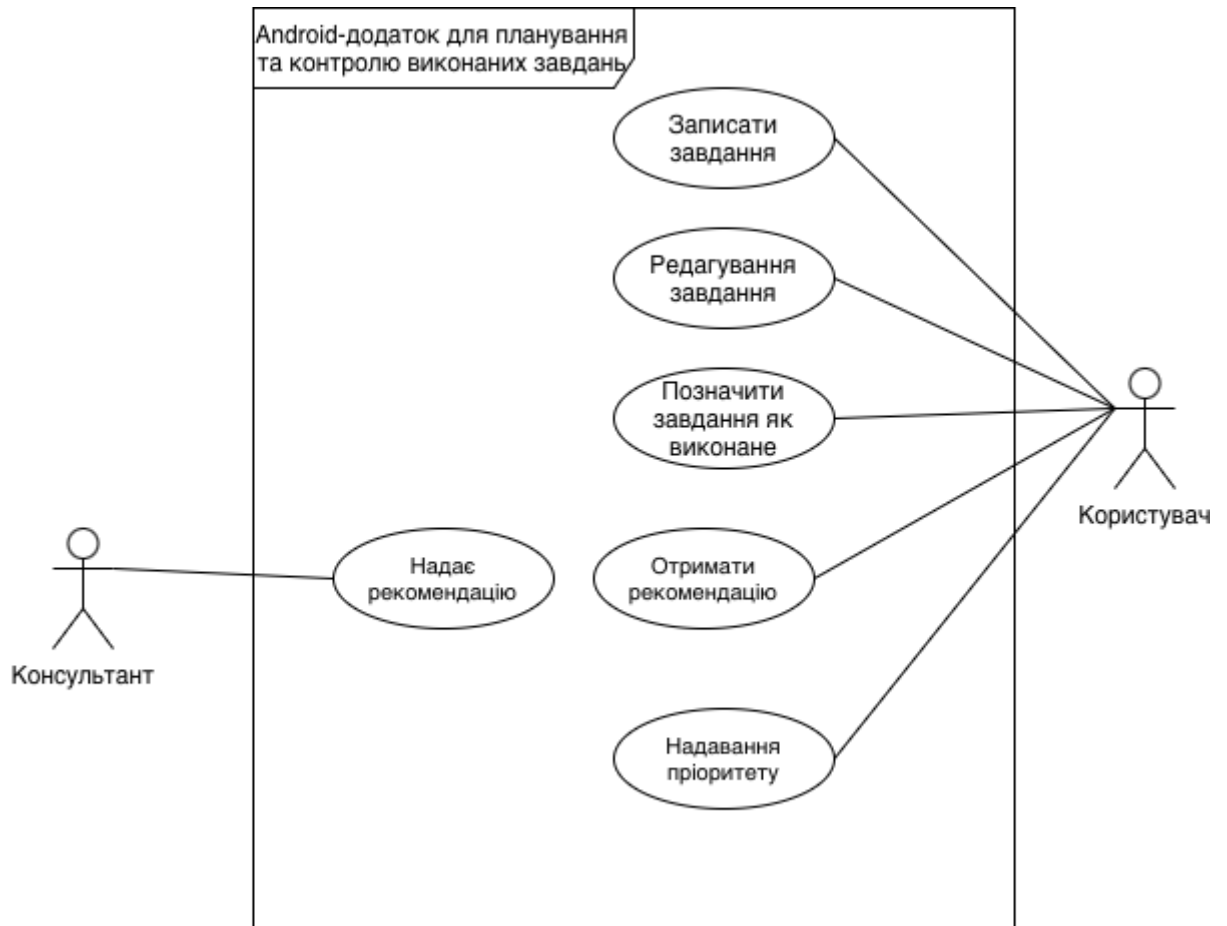


Рис. 1 Діаграма прецедентів.

Діаграма відображає взаємодію між двома акторами – Користувачем і Консультантом – та основними функціями системи.

Користувач має доступ до п'яти прецедентів.

Записати завдання – базова операція, з якої починається будь-яка робота в додатку.

Редагування завдання – зміна будь-яких параметрів вже існуючої задачі.
Позначити завдання як виконане – відмітка факту завершення роботи над задачею.

Отримати рекомендацію – звернення до ІІІ за порадою щодо формулювання задач.

Надавання пріоритету – встановлення важливості конкретного завдання відносно інших.

Консультант як зовнішній актор бере участь лише в одному сценарії – він надає рекомендацію, яку запитує користувач. Це відображає реальну роль ІІІ-модуля в системі.

1.3.2 Діаграма послідовності. Діаграма показує взаємодію між Користувачем, компонентом «Список завдань», компонентом «Завдання» і Консультантом. **Охоплює шість основних сценаріїв.**

Користувач створює список завдань – компонент «Список завдань» підтверджує створення і повертає порожній список. Після цього користувач вводить дані завдання – запит іде вже до компонента «Завдання», який зберігає нову задачу і повертає її у список.

Далі користувач позначає на якому етапі знаходиться завдання. Компонент «Завдання» оновлює статус і повертає підтвердження. Оновлений стан одразу відображається на екрані.

При редагуванні користувач вибирає завдання – відкривається форма редагування. Після змін дані передаються до «Завдання», яке їх зберігає і повертає оновлену задачу. Екран показує результат.

Видалення завдання – запит іде до «Завдання», після видалення компонент оновлює список і повертає актуальні дані.

Останній сценарій – запит консультації. Він іде напямку від Користувача до Консультанта, повністю минаючи компоненти «Список завдань» і «Завдання».

Консультант обробляє запит і повертає відповідь напряму користувачу. Це підкреслює що ІІІ-модуль є незалежним від основної логіки роботи з задачами.

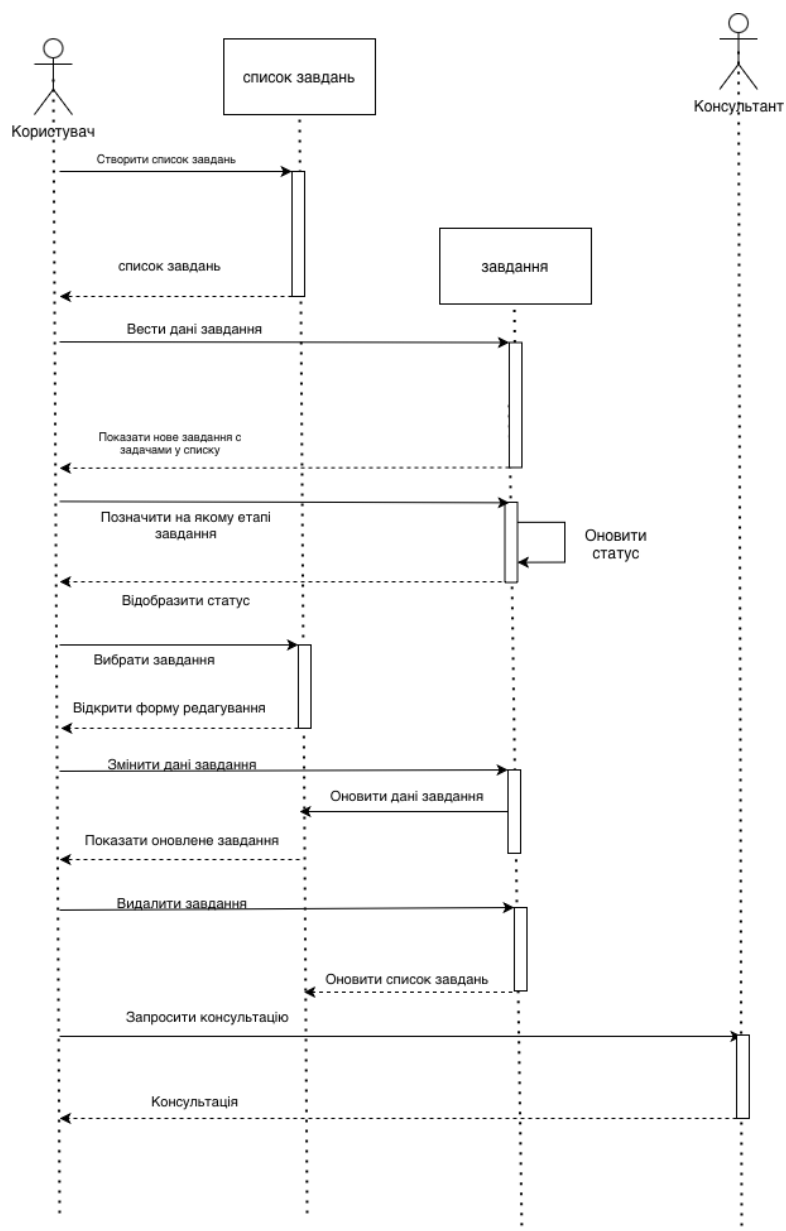


Рис. 2 Діаграма Послідовності

1.3.3 Діаграма активності.



Рис. 3 Діаграма Активності

Після старту додаток відкриває список завдань. **Якщо завдань немає** – користувач одразу переходить до створення нового. **Якщо є** – обирає існуюче. **Далі обидві гілки зходяться:** користувач вводить або змінює назву, дедлайн і статус.

Після цього система перевіряє чи потрібна консультація. Якщо ні – одразу зберігає. Якщо так – користувач отримує консультацію від ШІ, оновлює дані на її основі і тільки після цього зберігає. Завершується процес оновленням списку завдань.

1.3.4 Діаграма абстракції.

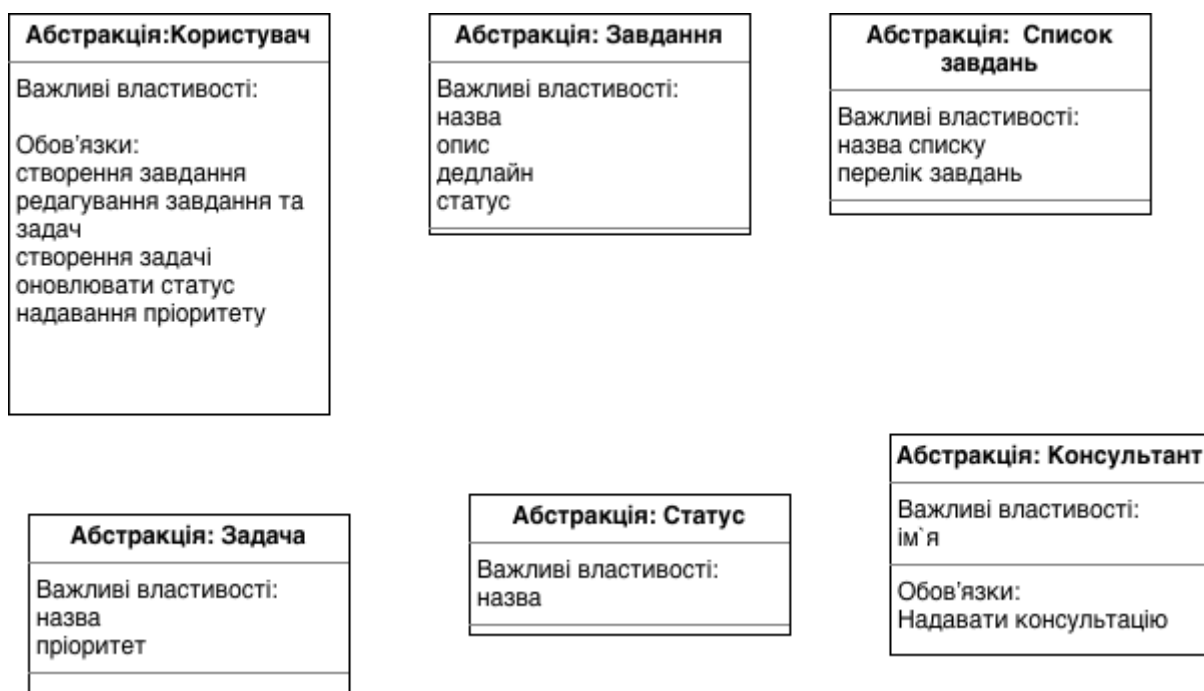


Рис. 4 Діаграма Абстракції

Модель включає шість сутностей системи:

Користувач – активний актор з найширшим переліком обов'язків: створення завдань, редагування завдань і задач, створення задачі, оновлення статусу і надавання пріоритету.

Список завдань – контейнер верхнього рівня з двома властивостями: назва списку і перелік завдань які до нього входять.

Завдання – основна робоча сутність. Має чотири властивості: назва, опис, дедлайн і статус.

Задача – підзадача всередині завдання. Має назву і пріоритет.

Статус – довідникова сутність з єдиною властивістю: назва. Відповідає за відображення поточного стану завдання.

Консультант – ШІ-модуль з властивістю ім'я і одним обов'язком: надавати консультацію. Проста і чітка сутність з мінімальним інтерфейсом взаємодії.

1.3.5 Діаграма класів.

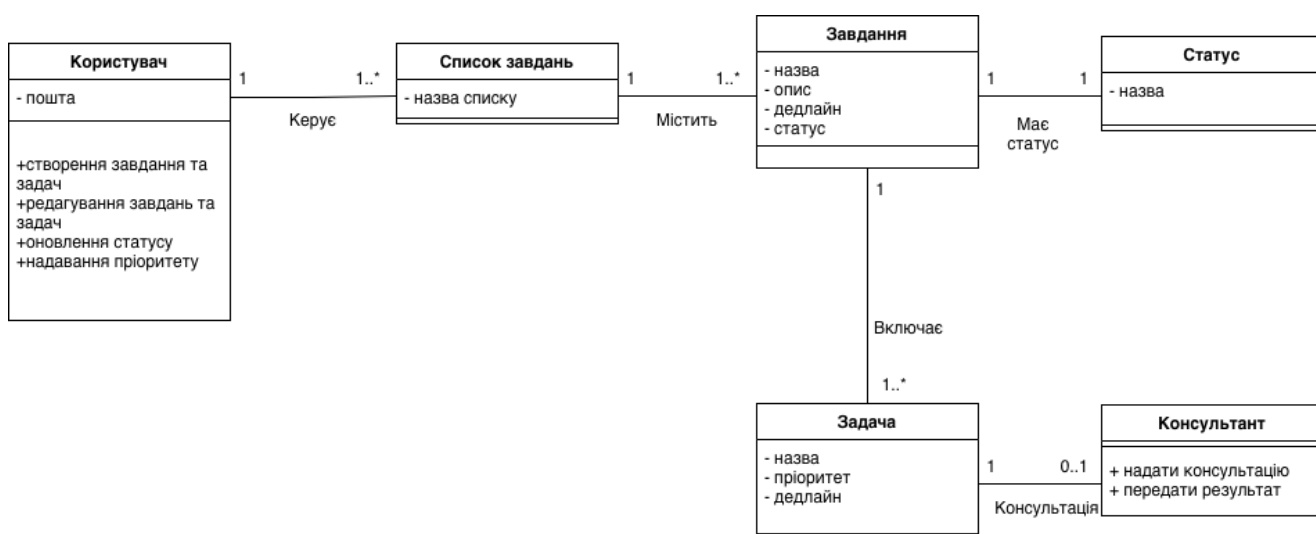


Рис. 5 Діаграма Класів

Діаграма відображає структуру системи через шість класів і зв'язки між ними. Користувач має одну властивість – пошта, і чотири методи: створення завдання та задач, редагування завдань та задач, оновлення статусу і надавання пріоритету. Це найактивніший клас у системі – майже всі операції ініціює саме він.

Зв'язок Користувач → Список завдань позначений як «Керує» з мультиплікатністю 1 до 1..* – один користувач може мати один або більше списків завдань.

Список завдань має єдину властивість – назва списку. Через зв'язок «Містить» з мультиплікатністю 1 до 1..* він пов'язаний з Завданнями – кожен список містить одне або більше завдань.

Завдання – центральний клас з чотирма властивостями: назва, опис, дедлайн і статус. Має два зв'язки. Перший – «Має статус» з класом Статус, зв'язок 1 до 1, тобто кожне завдання має рівно один статус. Другий – «Включає» з класом Задача, зв'язок 1 до 1..*, тобто кожне завдання включає одну або більше підзадач.

Статус – простий довідниковий клас з єдиною властивістю «назва». Він не виконує дій, лише зберігає поточний стан завдання.

Задача – підзадача всередині завдання. Має три властивості: назва, пріоритет і дедлайн. Через зв'язок «Консультація» з мультиплікатністю 1 до 0..1 пов'язана з Консультантом – тобто до задачі може бути прив'язана консультація, але це необов'язково.

Консультант – ШІ-модуль з двома методами: надати консультацію і передати результат. Власних властивостей не має – він суто функціональний, відповідає на запити і повертає результат.

1.4 Огляд інформаційних джерел та існуючих рішень

Перед початком розробки було проаналізовано кілька найбільш популярних додатків у категорії управління задачами. Мета – зрозуміти що вже є на ринку, де сильні сторони існуючих рішень і де вони не справляються.

ToDoList

Один з найпопулярніших task manager додатків. Підтримує проєкти, мітки, пріоритети, нагадування, інтеграції з іншими сервісами. Інтерфейс виглядає сучасно. Але переважна більшість корисних функцій – за платною підпискою. Безкоштовна версія суттєво обмежена по кількості проєктів і функціональності. Ієрархія типу «список → задача → підзадача» є, але в безкоштовному плані підзадачі обмежені. ШІ-інтеграції для генерації тексту завдань немає.

Google Tasks

Максимально простий і безкоштовний додаток від Google. Добре інтегрується з Gmail і Google Calendar. Але функціональність дуже базова: немає підзадач у повноцінному розумінні, немає пріоритетів, немає категорій і вкладок. Підходить якщо потрібно просто записати кілька справ. Для серйозного планування – замало.

Microsoft To Do

Непоганий безкоштовний варіант з підтримкою списків, нагадувань і базових пріоритетів. Є підзадачі у вигляді кроків. Але сильна прив'язка до екосистеми Microsoft – без акаунту Microsoft функціонал обмежений. ІІІ-асистента у вигляді окремої вкладки для діалогу немає. Синхронізація інколи поводить не стабільно.

TickTick

Найбільш функціональний з безкоштовних варіантів. Є пріоритети, теги, підзадачі, таймер Помодоро і статистика. Але інтерфейс перевантажений – новому користувачу треба час щоб розібратись. Частина функцій також за підпискою. ІІІ для генерації тексту задач і окремого чату немає.

Висновок з огляду

Проаналізувавши ці рішення, можна виділити спільні проблеми. По–перше, більшість корисних функцій платні. По–друге, жоден з розглянутих додатків не має повноцінної ІІІ-інтеграції одночасно у двох форматах – генерація тексту при створенні задач і окрема чат-вкладка для вільного спілкування. По–третє, трирівнева ієрархія список – задача – підзадача або відсутня або обмежена у безкоштовних версіях.

Саме ці спостереження і сформували основу для власного рішення: безкоштовний додаток з повноцінною ієрархією даних, двома форматами ІІІ-взаємодії і простим інтерфейсом без зайвого.

1.5 Постановка завдання

На основі всього вищесказаного – аналізу предметної області, вимог і огляду існуючих рішень – можна чітко сформулювати завдання даної роботи.

Потрібно розробити мобільний Android-додаток для планування та контролю виконання завдань з такими характеристиками.

Структура даних будується за ієрархією трьох рівнів. Перший рівень – списки завдань, які користувач створює під різні сфери своєї діяльності. Другий рівень – завдання всередині кожного списку з атрибутами назви, опису, дати і пріоритету. Третій рівень – підзадачі, які прив'язані до конкретного завдання і дозволяють розбити його на менші кроки.

Штучний інтелект інтегрується у двох місцях. При створенні завдання – користувач може натиснути кнопку і ШІ автоматично згенерує назву і опис на основі введених ключових слів. Це прискорює процес і допомагає краще формулювати задачі. Окрема вкладка з чат-інтерфейсом – де користувач може вільно спілкуватись з ШІ-асистентом, ставити питання щодо планування, просити поради або уточнювати деталі.

Авторизація і синхронізація даних реалізуються через Firebase. Всі списки, задачі і підзадачі зберігаються в хмарі і прив'язані до акаунту користувача. Реалізація виконується мовою Java в середовищі Android Studio. Готовий додаток має працювати на Android 8.0 і вище, пройти тестування основних сценаріїв і бути готовим до встановлення на реальний пристрій.

Кінцевий результат – не просто навчальний проєкт, а практично корисний інструмент який закриває реальну потребу і має чітку перевагу над існуючими безкоштовними рішеннями завдяки вбудованому ШІ і повноцінній трирівневій структурі задач.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Проектування інформаційного та програмного забезпечення мобільного додатку є одним з найважливіших аспектів розробки, оскільки на цьому етапі(рівні) ми можемо визначити структуру даних, логіку роботи нашого застосунку, а також принципи взаємодії користувача з системою.

Основною метою даного етапу є створення зрозумілої, логічної та масштабованої архітектури, яка забезпечує стабільну роботу додатку, простоту використання та можливість подальшого розвитку функціоналу.

Додаток який розроблював є системою для управління завданнями, яка дозволяє користувачу створювати списки завдань, завдання та додавати задачі, також відстежувати їх виконання, використовувати допоміжні інтелектуальні функції (ШІ) для максимального спрощення процесу створення контенту.

Інформаційне забезпечення програми базується на зберіганні даних користувача та забезпеченні синхронізації між пристроями. Як рішення, використовую хмарну базу даних Firebase, яка дозволяє зберігати дані у реальному часі, що дозволяє дуже зручно керувати даними, також забезпечує доступ до неї з будь-якого після авторизації користувача.

Уся інформація в системі під'єднується до конкретного облікового запису, що дозволяє зайти з будь-якого пристрою в свій обліковий запис і мати власний набір даних, який є ізольованим від інших користувачів системи.

Основними типами даних у системі є:

- Дані користувача
- Списки завдань

- Завдання
- Задачі
- Статус виконання

Структура даних побудована ієрархічно. На найвищому рівні знаходиться користувач. Далі йдуть списки завдань, які можуть містити багато завдань. Кожне завдання, у свою чергу, може містити набір підзадач. Така структура дозволяє гнучко організовувати інформацію та забезпечує логічний поділ даних.

Особливістю інформаційного забезпечення є використання синхронізації в реальному часі. Що дуже важливо в системі, адже це дозволяє керувати даними набагато зручніше. Це означає, що будь-які зміни, які користувач вносить у додатку (створення завдання, редагування, видалення або зміна статусу), одразу оновлюються у базі даних і стають доступними на інших пристроях.

2.1 Логічна модель даних у вигляді ER-діаграми

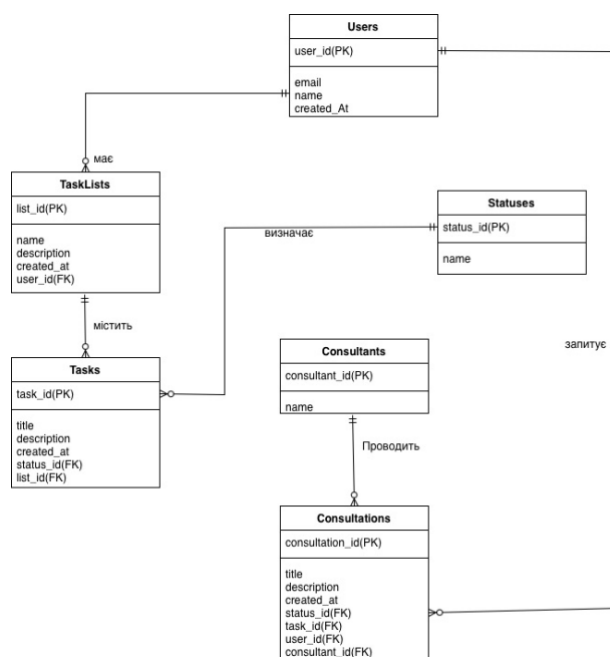


Рис. 6 ER-діаграма

Структура бази даних системи планування завдань

Розроблена база даних складається з шести взаємопов'язаних таблиць, які забезпечують повне функціонування системи управління задачами та консультаціями(ШП). Кожна таблиця виконує чітко визначену роль у архітектурі системи.

Таблиця "Користувачі" (Users) є центральною сутністю цієї системи. Вона містить основну інформацію про користувачів з полями: унікальний ідентифікатор користувача (user_id) як первинний ключ, електронну адресу для аутентифікації в нашій системі, ім'я користувача та дату створення облікового запису. Кожен користувач може мати одну або більше списків завдань, що забезпечує персоналізацію даних.

Таблиця "Списки завдань" (TaskLists) дозволяє користувачам організовувати свої задачі у логічні групи. Вона має власний первинний ключ (list_id), назву списку для швидкої ідентифікації, детальний опис мети чи тематики списку, дату створення та зовнішній ключ (user_id), що пов'язує список з конкретним користувачем. Відносина "має" визначає, що один користувач може створити багато списків.

Таблиця "Завдання" (Tasks) представляє окремі задачі, що належать до списків. Вона містить унікальний ідентифікатор завдання (task_id), заголовок задачі, його детальний опис, дату створення, зовнішній ключ на статус (status_id) та зовнішній ключ на список завдань (list_id). Відносина "містить" показує, що один список завдань містить багато окремих задач.

Таблиця "Статуси" (Statuses) служить довідником для можливих станів завдань та консультацій. Вона складається з унікального ідентифікатора статусу (status_id) та його текстового найменування. Ця таблиця дозволяє чітко визначити, у якому стані знаходиться та чи інша задача, забезпечуючи єдину точку управління всіма можливими статусами в системі.

Таблиця "Консультанти" (Consultants) містить інформацію про професіоналів, які можуть надавати консультації. Вона має унікальний ідентифікатор консультанта (consultant_id) та його ім'я. Цінність цієї таблиці полягає у можливості розширення системи функціоналом консультацій та експертизи.

Таблиця "Консультації" (Consultations) Вона містить унікальний ідентифікатор консультації (consultation_id), тему консультації, її детальний опис, дату проведення, статус виконання, посилання на пов'язане завдання (task_id), користувача (user_id) та консультанта (consultant_id). Це дозволяє відстежувати консультаційні процеси, пов'язані з конкретними завданнями, та забезпечує повну історію взаємодій.

Бази даних побудована таким чином, що запобігає дублюванню даних та забезпечує повну цілісність інформації. Такий дизайн дозволяє системі легко масштабуватися та додавати нові функції без значного переструктурування бази даних.

2.2 Діаграма класів та кооперацій

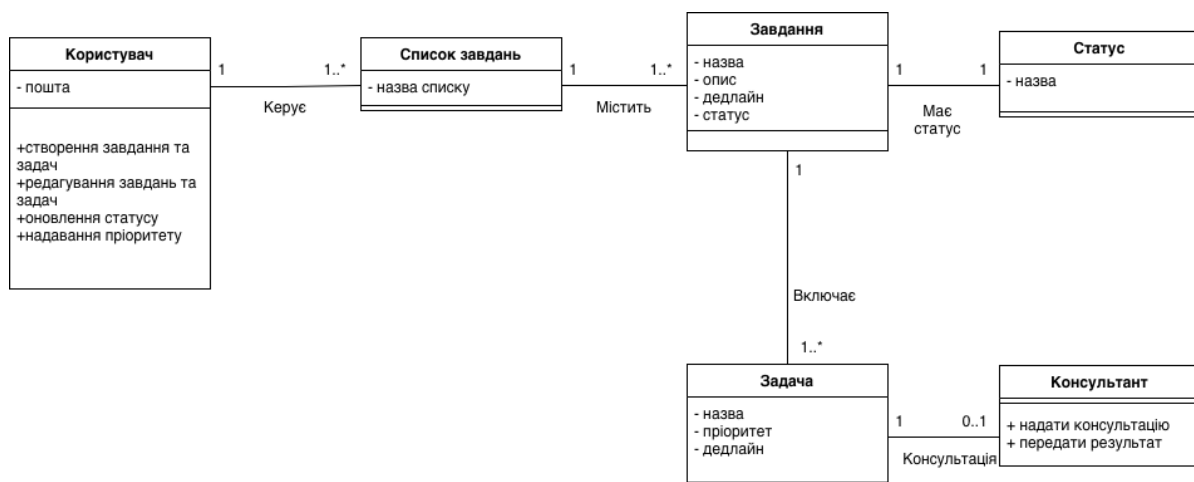


Рис. 7 Діаграма Класів

Діаграма кооперацій для створення завдань:

Діаграма кооперацій показує взаємодію трьох основних компонентів системи управління завданнями: Користувача, Списку завдань та Завдання.



Рис. 8 Діаграма Кооперацій для створення завдань

Користувач – це людина, яка працює з системою. Користувач має електронну адресу (email) та може виконувати такі операції:

- створювати нові завдання;
- редагувати існуючі завдання;
- оновлювати статус завдання;
- надавати пріоритет завданням.

Список завдань – це контейнер, який групує кілька пов'язаних завдань. Він має назву та містить список завдань всередині. Завдання – це окреме завдання, яке користувач має виконати. Кожне завдання містить такі характеристики:

- назва завдання (що потрібно зробити);
- опис (деталі про завдання);
- дедлайн (дата, до якої завдання повинне бути виконано);

- статус (поточний стан завдання: нове, в роботі, виконане).

Діаграма кооперацій для створення задач:

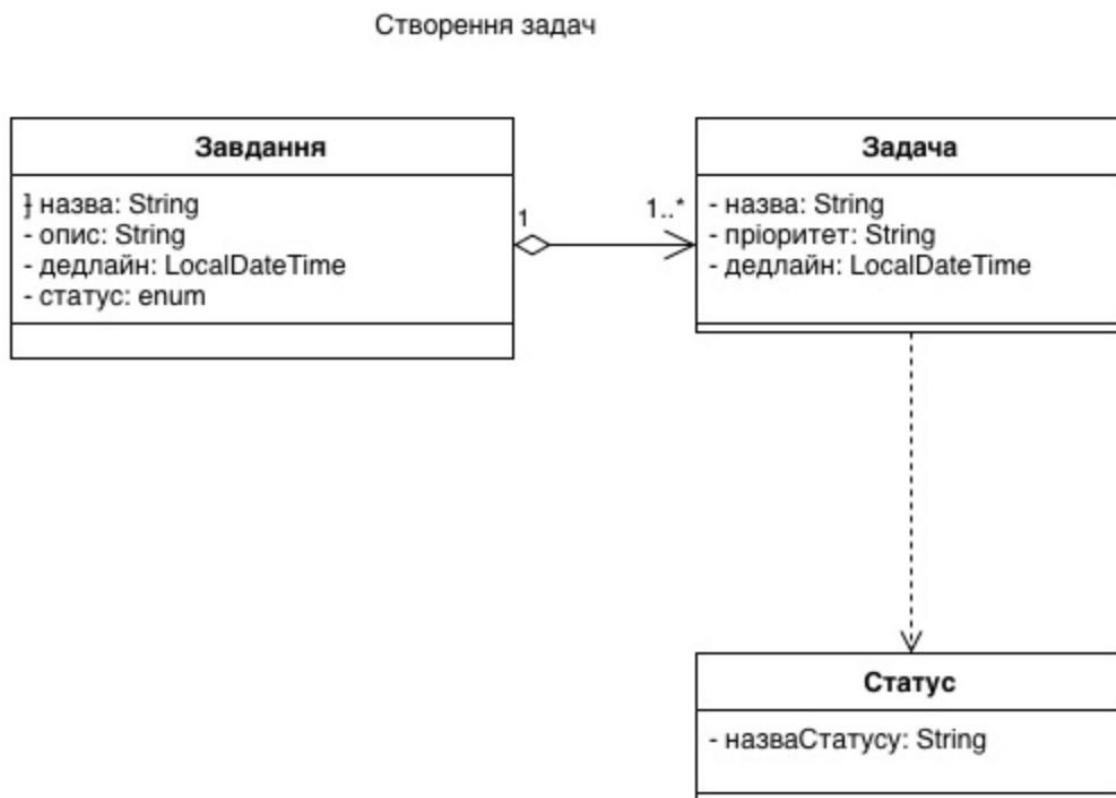


Рис. 9 Діаграма Кооперацій для створення задач

Завдання – це основна одиниця роботи в системі. Кожне завдання має такі властивості:

- назва – текстовий опис того, що потрібно зробити;
- опис – детальна інформація про завдання;
- дедлайн – дата та час, до якого завдання повинне бути виконано;
- статус – поточний стан завдання.

Задача – це конкретна робота, яка має бути виконана. Задача розширює поняття Завдання і містить додаткові параметри:

- назва – назва конкретної задачі;

- пріоритет – важливість задачі (низький, середній, високий);
- дедлайн – строк виконання задачі.

Статус – це опис поточного стану задачі. Він містить одну властивість:

- `назваСтатусу` – текстовий опис статусу.

Діаграма кооперацій для консультацій(ШІ):



Рис. 10 Діаграма Кооперацій для констультатції з ШІ

Ця діаграма показує, як Консультант взаємодіє із Завданнями, надаючи допомогу та передаючи результати.

Завдання – це завдання, яке користувач створив у системі. Воно має такі характеристики:

- `назва` – назва завдання;
- `опис` – детальний опис того, що потрібно зробити;
- `дедлайн` – строк виконання;
- `статус` – поточний стан завдання.

Консультант – це спеціаліст або система, яка надає допомогу при виконанні завдання. Консультант має два основних методи (функції):

1. **`надатиКонсультацію(Завдання завдання)`** – Консультант отримує Завдання та надає консультацію. Цей метод приймає на вхід об'єкт Завдання та не повертає значення.

2. **передатиРезультат(Завдання завдання)** – Консультант передає результати своєї консультації назад до Завдання. Цей метод також приймає об'єкт Завдання та не повертає значення.

2.3 Діаграма пакетів

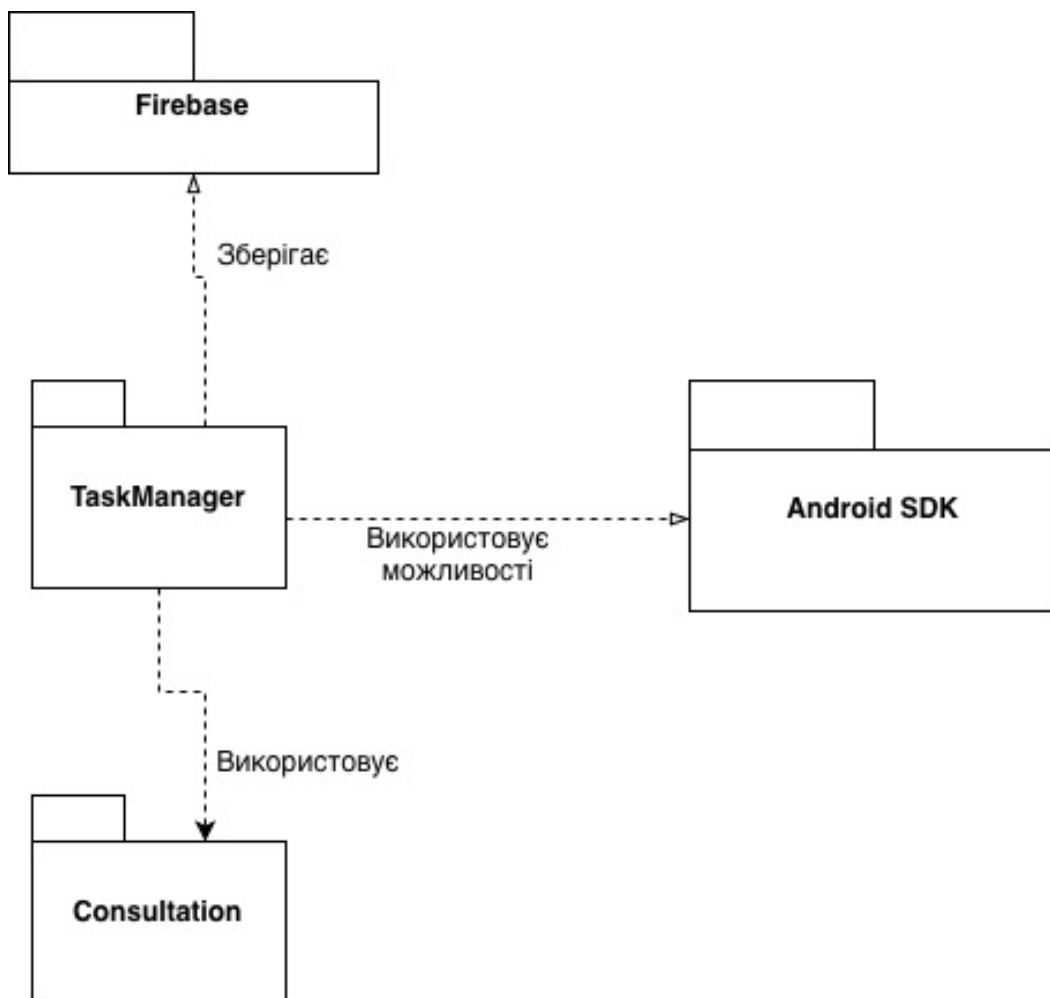


Рис. 11 Діаграма Пакетів

Firebase – це хмарна база даних, яка зберігає всю інформацію про завдання, користувачів та інші дані системи. Firebase розташована на вершині архітектури, оскільки вона є центральним сховищем даних.

TaskManager – це основний компонент системи, який управляє всіма завданнями. Він отримує дані з Firebase, обробляє запити користувача та передає інформацію до інших компонентів системи.

Android SDK – це набір інструментів для розробки Android-додатків. Він надає функціональність для створення користувацького інтерфейсу та взаємодії з користувачем на мобільному пристрої.

Consultation – це компонент, який надає консультації(III) та поради щодо виконання завдань. Він працює спільно з TaskManager для допомоги користувачам.

2.4 Діаграма компонентів

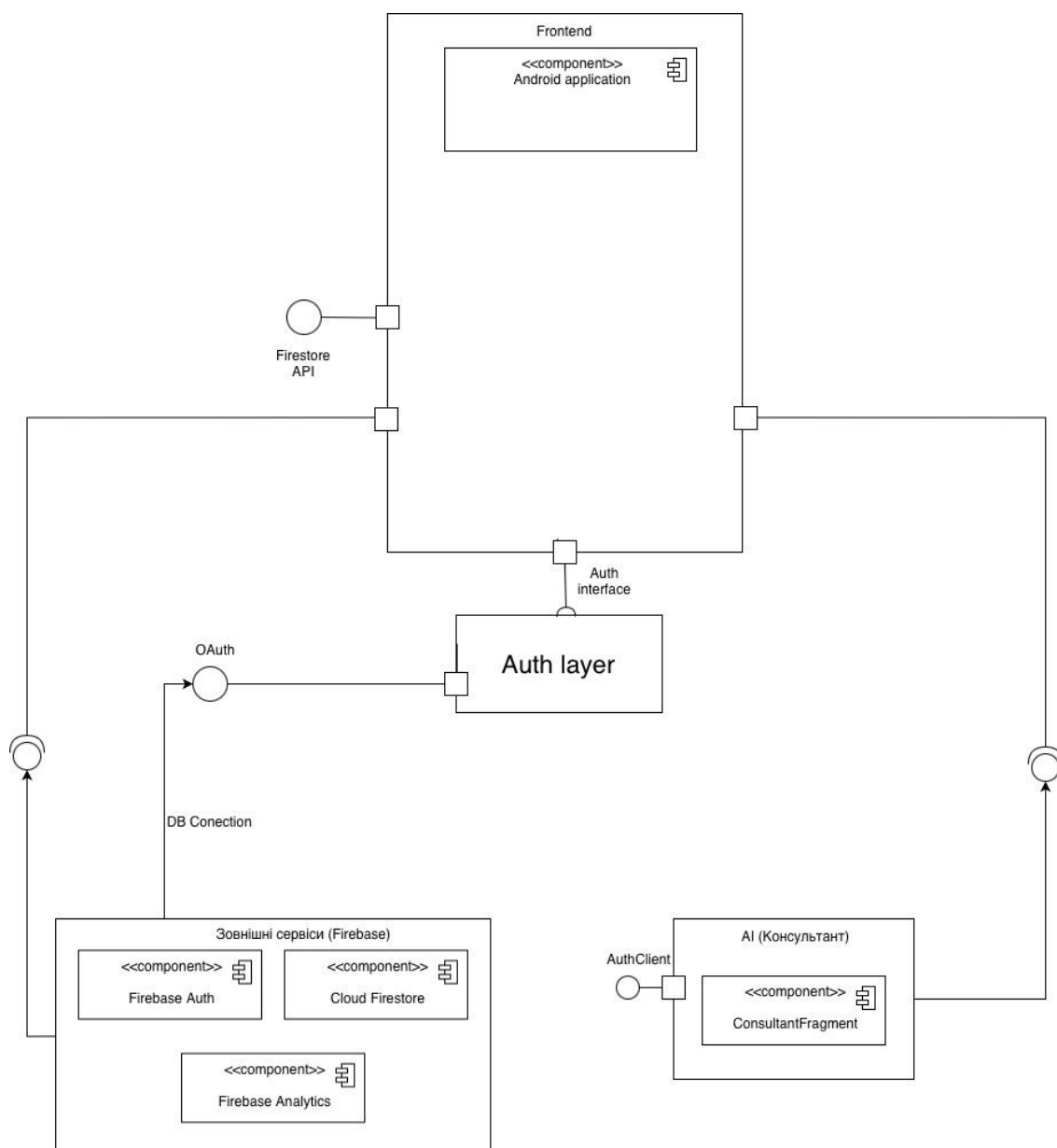


Рис. 12 Діаграма Пакетів

Ця діаграма показує архітектуру цілої системи та взаємодію всіх її компонентів. Система розділена на кілька рівнів та частин.

Основні шари (рівні) системи

1. Frontend (Фронтенд)

Android application – це мобільний додаток, який користувач встановлює на свій смартфон. Через цей додаток користувач взаємодіє з усією системою, створює завдання, переглядає їх та отримує консультації.

2. Auth Layer (Шар автентифікації)

Auth layer – це компонент, який відповідає за перевірку особистості користувача (вхід в систему). Він містить:

Auth interface – інтерфейс для входу користувача;

OAuth – протокол безпечної автентифікації, який дозволяє користувачеві входити через свій аккаунт;

AuthClient – клієнт, який взаємодіє з сервісами автентифікації.

3. Firestore API (API баз даних)

Firestore API – це проміжник між додатком та базою даних. Він отримує запити від Android application та передає їх до Firebase.

4. Зовнішні сервіси Firebase (Backend)

Це серверна частина системи, яка включає три компоненти:

Firebase Auth – сервіс автентифікації, який зберігає та перевіряє дані користувачів;

Cloud Firestore – хмарна база даних, яка зберігає всі завдання, списки та інші дані;

Firebase Analytics – інструмент для аналізу того, як користувачі користуються додатком.

5. AI (Консультант)

ConsultantFragment – це компонент штучного інтелекту, який надає консультації користувачам. Він отримує інформацію про завдання та надає поради щодо його виконання.

Як система працює

Користувач встановлює Android application на свій телефон. Перше, що він робить, – входить в систему через Auth layer, використовуючи OAuth. Після входу він має доступ до всіх функцій: може створювати завдання (дані зберігаються в Cloud Firestore), переглядати їх, редагувати та при необхідності запросити консультацію у компонента AI. Firebase Analytics стежить за тим, як користувач користується системою, щоб її можна було вдосконалити.

3. РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Цей розділ описує всі аспекти розробки системи управління завданнями та консультацій. Тут розглядаються архітектура інформаційної бази, структура даних, вибір технологій для розробки та реалізація програмних компонентів системи. Система розроблена як мобільний додаток для платформи Android, що забезпечує користувачам зручний спосіб керування своїми завданнями з будь-якого місця.

3.1 Система управління інформаційною базою

Система управління інформаційною базою (СУБД) – це комплекс програмних і апаратних засобів, які забезпечують зберігання, організацію, обробку та видачу інформації користувачам.

Для розроблюваної системи управління завданнями було обрано хмарну платформу Firebase, яка являє собою сучасне рішення для розробки мобільних та веб-додатків.

Firebase – це платформа від компанії Google, яка надає набір сервісів для розробки додатків без необхідності розробляти складну серверну частину. Вибір Firebase обумовлений кількома важливими факторами:

Хмарне розташування даних – сервери Firebase розташовані в різних точках світу, що забезпечує надійність та швидкість доступу до даних

Реальна синхронізація (Real-time Database) – будь-які зміни в базі даних миттєво передаються до всіх підключених користувачів. Це означає, що якщо один користувач змінить статус завдання, інші користувачі (якщо це спільні завдання) миттєво побачать оновлення

Автентифікація та безпека – Firebase Authentication забезпечує безпечне зберігання облікових даних користувачів, підтримує вхід через Google, Facebook

та інші популярні сервіси. Дані користувачів шифруються і захищені від несанкціонованого доступу

Масштабованість – система автоматично розширює свої ресурси при збільшенні кількості користувачів. Коли користувачів мало, система витрачає менше ресурсів, коли їх багато – система розширюється без впливу на швидкість роботи

Аналітика – **Firebase Analytics** дозволяє розробникам відстежувати, як користувачі користуються додатком, які функції популярні, де відбуваються помилки

Офлайн режим – якщо користувач втратить з'єднання з інтернетом, він все ще зможе працювати з додатком, а дані синхронізуються при повторному з'єднанні

Firebase складається з кількох компонентів, які використовуються в розроблюваній системі:

Cloud Firestore – це NoSQL база даних, яка зберігає завдання, списки завдань, інформацію про користувачів та інші дані системи. На відміну від звичайних баз даних (як SQL), Firestore зберігає дані у вигляді колекцій документів, що дозволяє більш гнучко працювати зі структурою даних

Firebase Authentication – сервіс, який відповідає за перевірку особистості користувача. Використовується для входу користувача в систему та забезпечення того, що кожний користувач може бачити тільки свої дані

Firebase Analytics – інструмент для аналізу того, як користувачі користуються додатком. Збирає інформацію про те, які кнопки клікають користувачі, скільки часу вони витрачають на роботу з завданнями, у яких місцях вони відмовляються від використання

Firestore Security Rules – це правила безпеки, які контролюють, хто може читати та писати дані в базі. Наприклад, користувач А не може переглядати завдання користувача В

Таким чином, Firebase забезпечує весь необхідний функціонал для управління даними системи, з гарантіями безпеки, надійності та швидкості доступу. Розробникам не потрібно створювати власну серверну частину – все уже готово на платформі Firebase.

3.2 Розробка інформаційної бази

На етапі проектування інформаційної бази було розглянуто кілька варіантів систем зберігання даних, включаючи традиційні реляційні бази даних (MySQL, PostgreSQL), локальні бази даних для мобільних платформ (SQLite), та сучасні хмарні рішення (Firebase, Amazon DynamoDB). Вибір оптимального варіанту здійснювався на основі аналізу вимог системи та порівняння можливостей кожного рішення.

Як основну систему управління базою даних було обрано Firebase Cloud Firestore від компанії Google. Це рішення являє собою документно-орієнтовану NoSQL базу даних, яка розташована в хмарній інфраструктурі Google. На відміну від традиційних реляційних баз даних, де дані організовані в таблиці з рядками та стовпцями, Firestore зберігає дані у вигляді документів (JSON-подібних структур), які об'єднуються в колекції. Такий підхід забезпечує більшу гнучкість при роботі зі складними та ієрархічними структурами даних, що є особливо корисним для мобільних додатків.

Вибір Firebase Firestore був обґрунтований наступними факторами. По-перше, Firestore забезпечує синхронізацію даних у реальному часі через механізм слухачів подій. Це означає, що при будь-якій зміні даних в базі всі клієнти, які слухають на ці дані, миттєво отримують оновлення без необхідності виконувати явні запити. Така функціональність особливо важлива для мобільних додатків, де

потрібно відображати актуальну інформацію без затримок і без виснаження батареї частими запитами до сервера.

По-друге, Firebase забезпечує безсерверну архітектуру, що означає, що розробник не повинен турбуватися про розгортання, налаштування та адміністрування власного сервера. Вся інфраструктура, включаючи масштабування, балансування навантаження, резервне копіювання та безпеку, управляється компанією Google. Це значно зменшує складність розробки та експлуатації системи, особливо для стартапів та невеликих команд розробників.

По-третє, гнучка документна модель Firestore природно відповідає структурі даних нашої системи. На відміну від реляційних баз даних, де довелося б створювати окремі таблиці для всіх підзадач, в Firestore можна безпосередньо вбудовувати підзадачі у вигляді масиву всередину документа задачі. Це означає, що при завантаженні однієї задачі отримуються всі її підзадачі за один запит, без необхідності виконувати додаткові запити до бази. Такий підхід суттєво покращує продуктивність системи, особливо на мобільних пристроях з обмеженою пропускнуою здатністю мережі.

Також Firebase інтегрує інтеграцію аутентифікації безпосередньо в систему управління базою даних. Firebase Authentication надає як вбудовані механізми входу (email/пароль), так і аутентифікацію через соціальні мережі та інші сервіси. Правила безпеки (Firestore Security Rules) дозволяють налаштувати контроль доступу до даних на рівні документів та полів. Це означає, що можна просто і надійно гарантувати, що кожен користувач має доступ лише до власних даних, без потреби в написанні складного коду на серверній частині.

По-п'ятому, Firestore забезпечує підтримку роботи в офлайн режимі. SDK для Android автоматично кешує дані локально на пристрої, дозволяючи користувачеві читати і навіть виконувати локальні операції без підключення до інтернету. При поновленні з'єднання всі локальні зміни автоматично синхронізуються з сервером. Це критично важливо для мобільних додатків,

оскільки користувачі часто опиняються в умовах нестійкого або відсутнього інтернету.

Крім того, Firebase надає безкоштовний рівень обслуговування (Free Tier) з досить великими лімітами на запити та зберігання, що дозволило розробити та тестувати систему без початкових фінансових витрат. При необхідності масштабування система легко переходить на платну версію зі сплатою за використані ресурси.

3.2.1 Архітектура та організація даних. Інформаційна база системи організована навколо трьох основних сутностей: користувача, списків задач та самих задач. Користувач є верхівкою ієрархії, кожен користувач має можливість створювати кілька списків задач, а кожен список містить кілька задач. Задачі, у свою чергу, можуть розбиватися на підзадачі для більш детального керування великими обсягами роботи.

Користувач ідентифікується унікальним ідентифікатором (uid), що генерується системою Firebase Authentication. Цей ідентифікатор використовується як ключ для фільтрації даних, гарантуючи, що кожен користувач бачить та може редагувати лише власні дані. Облікова інформація користувача (email, пароль) зберігається в Firebase Authentication, яке забезпечує цілком інший механізм захисту паролів з використанням хеширування та солей.

Списки задач мають можливість кольорового маркування для полегшення їх розпізнавання користувачем. Система підтримує набір з шести наперед визначених кольорів: фіолетовий, червоний, жовтий, зелений, синій та рожевий. При створенні нового списку система випадково обирає один з цих кольорів. Це упорядкує інтерфейс та робить його більш інтуїтивним для використання.

Задачі мають багато атрибутів, які дозволяють користувачеві повністю визначити характер роботи. Кожна задача має назву (обов'язкова) та опис (необов'язковий). Пріоритет задачі визначається однією з трьох категорій: важливо, середнє, звичайне. Це дозволяє користувачеві швидко виділити

найбільш критичні завдання. Дата виконання встановлюється у простому форматі (сьогодні, завтра), що спрощує вхідні дані та робить інтерфейс доступнішим. Статус виконання (done) зберігається як булеве значення, що дозволяє швидко фільтрувати виконані та невиконані задачі.

Підзадачі представляють собою внутрішню структуру задачі. Вони зберігаються не як окремі документи в базі, а як вбудований масив всередину документа основної задачі. Це рішення прийнято з кількох причин. По–перше, підзадачі нерозривно пов'язані з конкретною задачею і не мають сенсу поза її контекстом. По–друге, зберігання підзадач як вбудованого масиву забезпечує атомарність операцій при оновленні – весь набір підзадач завжди оновлюється разом з основною задачею, усуваючи можливість некоректного стану. По–третє, така організація мінімізує кількість запитів до бази даних, оскільки завантаження одної задачі автоматично забезпечує доступ до всіх її підзадач.

3.2.2 Структура колекцій в Firestore. База даних Firestore організована у вигляді двох основних колекцій: lists та tasks. Колекція lists містить документи, які представляють списки задач користувачів. Кожен такий документ містить ім'я списку, його колір, ідентифікатор власника та часову мітку створення. Колекція tasks містить документи, які представляють окремі задачі. На додаток до атрибутів самої задачі, документ також містить вбудований масив підзадач, кожна з яких має свою назву, статус виконання та часову мітку.

Документи в Firestore мають унікальні ідентифікатори, які генеруються автоматично системою або можуть встановлюватися експліцитно. У нашій системі використовуються автоматично згенеровані ідентифікатори, що гарантує унікальність без ризику колізій. Ці ідентифікатори потім використовуються для створення зв'язків між документами. Наприклад, задача містить поле listId, яке вказує на список, до якого вона належить, та поле uid, яке ідентифікує власника задачі.

Запити до Firestore виконуються через SDK для Android. Для отримання списків конкретного користувача система виконує запит з фільтром

`whereEqualTo("uid", currentUserId)`, що гарантує отримання тільки даних поточного користувача. Аналогічно, при завантаженні задач системи переважно використовує слухачів подій (`addSnapshotListener`), які забезпечують оновлення інтерфейсу при будь-якій зміні даних в базі. Для детального перегляду однієї задачі використовується одноразовий запит (`.get()`), що дозволяє економити ресурси при навігації.

3.2.3 Локальне сховище та управління сесією. На додаток до хмарної бази Firestore, система використовує локальне сховище на мобільному пристрої для зберігання інформації про поточну сесію користувача. Локальне сховище реалізовано за допомогою механізму Android `SharedPreferences`, який являє собою набір пар ключ–значення, що зберігаються в приватному файлі застосунку. `SharedPreferences` використовується для зберігання інформації про те, чи зареєстрований користувач, його ім'я для відображення в інтерфейсі, та налаштування щодо включення/вимикання push–сповіщень.

Використання `LocalStorage` мотивовано кількома факторами. По–перше, дані сесії потрібні часто, при кожному запуску застосунку, тому локальне сховище забезпечує миттєвий доступ без затримки на мережу. По–друге, дані сесії не критичні для роботи з завданнями, тому їх можна безпечно зберігати локально. По–третє, режим приватного доступу `SharedPreferences` гарантує, що дані недоступні іншим застосункам на пристрої, забезпечуючи необхідний рівень безпеки для інформації про користувача.

При запуску застосунку система виконує двосторонню перевірку. Спочатку вона перевіряє, чи активний токен `Firebase Authentication`, який вказує на те, що користувач зареєстрований на стороні Google. Потім вона перевіряє наявність прапора `isLoggedIn` у `LocalStorage`. Якщо обидві умови виконані, користувач автоматично переходить на головний екран додатка, мінаючи екран входу. Якщо ж одна з умов не виконана, користувач перенаправляється на екран входу, де він повинен ввести облікові дані. Це гарантує і найшвидший вхід для

постійних користувачів, і безпеку від несанкціонованого доступу у разі втрати пристрою.

3.2.4 Система автентифікації. Автентифікація користувачів в системі здійснюється через Firebase Authentication з використанням методу email та пароля. Однак, для спрощення користувацького досвіду, в інтерфейсі застосунку користувач не вводить повноцінну email-адресу, а лише логін (username). Логін потім перетворюється на email-формат за допомогою простої формули: `email = username.toLowerCase() + "@taskapp.local"`. Такий підхід дозволяє користувачеві не турбуватися про наявність реальної електронної пошти при реєстрації, робить процес реєстрації більш простим та інтуїтивним.

Firebase Authentication надає вбудовану функціональність для безпечного зберігання паролів. Паролі не зберігаються в базі даних у текстовому вигляді або в простому хешованому форматі. Замість цього Firebase використовує криптографічне хеширування для кожного пароля. Це означає, що навіть у разі витоку даних з серверів Google, зловмисник не зможе легко відновити оригінальні паролі. Крім того, Firebase автоматично виявляє та блокує підозрілі спроби входу, забезпечуючи захист від атак перебору.

При успішному входу Firebase видає JWT токен (JSON Web Token), який використовується для подальшої автентифікації всіх запитів до Firestore. Цей токен містить інформацію про користувача та підписаний Firebase, що гарантує його автентичність. Токен зберігається на клієнті (в оперативній пам'яті Android SDK) та автоматично додається до всіх запитів до Firestore. На серверній частині токен перевіряється перед кожним запитом, гарантуючи, що доступ надається тільки автентифікованим користувачам.

3.2.5 Механізм розмежування доступу. Розмежування доступу до даних реалізовано на двох рівнях: на рівні сервера (Firestore) та на рівні клієнта (Android додатка). На рівні сервера використовуються Firestore Security Rules, які визначають, хто може читати та писати конкретні документи в базі. Правила безпеки налаштовані так, щоб кожен користувач мав доступ лише до документів,

де поле `uid` збігається з його ідентифікатором. Це гарантує, що навіть якщо злоумисник якимось чином отримає доступ до бази, він не зможе отримати дані інших користувачів.

На рівні клієнта додаткова фільтрація даних забезпечується за допомогою явного фільтрування всіх запитів. Кожний запит до Firestore містить умову `whereEqualTo("uid", currentUserId)`, що гарантує отримання тільки даних поточного користувача. Хоча це може здатися надлишковим, така двошарова захист забезпечує оборону в глибину (*defense in depth*), що є однією з найважливіших практик в безпеці інформаційних систем.

3.2.7 Інтеграція з AI-сервісом. Для надання функціональності штучного інтелекту, система взаємодіє з зовнішнім API сервісу OpenAI, конкретно з моделлю GPT-4o-mini. Ця модель являє собою потужний мовний засіб, який може розуміти природну мову користувача та генерувати змістовні відповіді. Обмін даними з OpenAI API здійснюється за протоколом HTTPS у форматі JSON, що забезпечує безпеку передачі та стандартизацію.

Важливо відзначити, що дані AI-діалогів не зберігаються постійно в базі Firestore. Замість цього вся історія розмови зберігається виключно в оперативній пам'яті Android додатка протягом поточного сеансу. При закритті екрану чату вся історія розмови втрачається. Це рішення прийнято з міркувань конфіденційності користувачів та економії ресурсів бази даних. Користувачі часто можуть обговорювати в чаті з AI чутливу або приватну інформацію, яку вони не хотіли б зберігати на серверах. Крім того, це дозволяє системі залишатися легкою та швидкою, без необхідності зберігати великі обсяги діалогів.

AI-функціональність використовується в двох основних сценаріях. По-перше, при додаванні нової задачі користувач може запросити допомогу AI у вдосконаленні тексту. AI здатна розширити лаконічний опис користувача, запропонувати кращу структуру, виявити потенційні проблеми в задумі роботи.

По-друге, існує спеціальний екран AI-чату, де користувач може вести вільний діалог з AI щодо своїх завдань, отримувати рекомендації, обговорювати

стратегії керування часом та продуктивністю. У обох випадках AI контекстно обізнаний: їй надається список поточних завдань користувача, що дозволяє їй надавати більш релевантні та персоналізовані рекомендації.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Розробка мобільного додатка вимагає вибору оптимального набору інструментів, мов програмування та фреймворків. Вибір правильного інструментарію безпосередньо впливає на швидкість розробки, якість кінцевого продукту та простоту підтримки системи. Для розроблюваної системи управління завданнями було ретельно проаналізовано кілька варіантів та обрано набір інструментів

3.3.1 Вибір платформи розробки. Основною платформою для розробки обрано мобільну операційну систему Android. Вибір Android був обґрунтований кількома факторами. По-перше, Android є найпопулярнішою мобільною операційною системою у світі, з часткою ринку понад 70%. Це означає, що додаток буде доступний для більшості мобільних користувачів. По-друге, Android є відкритою системою з великою спільнотою розробників, що означає наявність безлічі бібліотек, інструментів та ресурсів для розробки. По-третє, Android забезпечує високий рівень контролю над функціональністю пристрою, дозволяючи розробнику використовувати більшість апаратних можливостей смартфона.

Розробка для Android здійснюється на мові Java або сучасніше на мові Kotlin. Java вже багато років є стандартом для розробки Android, забезпечуючи надійність та стабільність. Kotlin була розроблена компанією JetBrains спеціально як удосконалення Java для Android розробки.

3.3.2 Android Studio як середовище розробки. Для розробки андроїд-додатків використовується Android Studio, офіційне інтегроване середовище розробки (IDE). Android Studio будується на основі популярної платформи IntelliJ

IDEA від компанії JetBrains, що гарантує високу якість та надійність. Android Studio надає всі необхідні інструменти для розробки, тестування та публікування Android-додатків.

Основні можливості Android Studio включають: редактор коду з інтелектуальною підтримкою (автозавершення, підсвічування синтаксису, рефакторинг), редактор макетів (Layout Editor) з перетягуванням компонентів, емулятор Android для тестування на комп'ютері, налагоджувач (debugger) з точками зупину та відстежуванням значень змінних, профайлер для аналізу використання пам'яті, інструменти для управління залежностями через Gradle, вбудована підтримка контролю версій (Git), та інструменти для тестування включаючи Unit тести та UI тести.

Використання Android Studio значно прискорює розробку завдяки автоматизації багатьох повторюваних задач. Наприклад, при створенні нового проекту IDE автоматично генерує бойлер-плейт код та необхідні конфігураційні файли. При редагуванні макету інтерфейсу можна перетягувати компоненти та їх властивості автоматично генеруються в XML коді. IDE також виявляє потенційні помилки та пропонує їх виправлення на льоту.

3.3.3 Фреймворки та бібліотеки. На додаток до базової платформи Android, розробка включала використання кількох важливих фреймворків та бібліотек, які представлені у таблиці 3.1.

Основні бібліотеки та фреймворки

Бібліотека / Фреймворк	Версія	Призначення
Firebase SDK	11.0+	Аутентифікація, база даних Firestore, аналітика, push-сповіщення
OkHttp	4.11+	HTTP запити до зовнішніх API (OpenAI)
Retrofit	2.10+	REST API інтерфейси та серіалізація JSON
Material Design Components	1.10+	UI компоненти (кнопки, поля введення, карточки)
Lifecycle (LiveData, ViewModel)	2.7+	Управління станом та реактивне оновлення UI
Gradle	8.0+	Система управління проектом та залежностями

Firebase SDK для Android надає функціональність для взаємодії з Firebase сервісами, включаючи аутентифікацію, базу даних Firestore, аналітику та push-сповіщення. OkHttp є популярною бібліотекою для виконання HTTP запитів до

зовнішніх API, таких як OpenAI. Retrofit надає зручний спосіб визначення REST API інтерфейсів та автоматичної серіалізації JSON.

Material Design Components бібліотека надає набір готових компонентів інтерфейсу (кнопки, текстові поля, карточки, навігацію) які відповідають стандартам Material Design. Це забезпечує сумісність з іншими Android додатками та гарантує, що інтерфейс буде знайомим для користувачів. LiveData та ViewModel з архітектурного компонента Android забезпечують реактивний підхід до управління станом додатка та дозволяють легко підтримувати синхронізацію інтерфейсу з даними.

3.3.4 Дизайн інтерфейсу та Material Design. Дизайн користувацького інтерфейсу розробленого додатка базується на стандартах Material Design, системи дизайну розробленої Google. Material Design визначає принципи, які забезпечують створення красивих, функціональних та зрозумілих інтерфейсів. Головними принципами Material Design, які використовуються в нашому додатку, є матеріальність та простота.

Матеріальність означає, що компоненти інтерфейсу мають чіткі границі, тіні та глибину, що робить їх схожими на фізичні об'єкти з паперу. Це надає інтерфейсу інтуїтивність та зробиляє взаємодію з ним природною. Простота означає видалення всього непотрібного та фокусування на найважливішому контенті та функціональності.

Кольорова схема додатка обирається з урахуванням принципів контрасту та читабельності. Основний колір та додатковий колір повинні забезпечувати достатній контраст. Матеріали Design пропонує системи кольорів, які гарантують гарну читабельність та естетичність. Додаток використовує типографіку, яка легко читається, підзаголовками та звичайним текстом.

Навігація в додатку побудована за принципом послідовності та передбачуваності. Користувач завжди знає, як повернутися на попередній екран, та може легко орієнтуватися між різними екранами додатка. Переходи між

екранами анімовані, що надає додатку більш полірованого та професійного вигляду.

3.3.5 Архітектура застосунку. Архітектура розробленого додатка слідує модель MVVM (Model–View–ViewModel), яка є рекомендованою архітектурою для Android застосунків. MVVM розділяє додаток на три шари: Model (модель) представляє дані та бізнес–логіку, View (вид) представляє користувацький інтерфейс, а ViewModel служить посередником між ними.

Model шар включає класи, які представляють сутності (User, TaskList, Task, Subtask), а також сховища (Repository) які забезпечують доступ до даних. Repository абстрагує деталі того, звідки беруться дані (з Firebase, локального сховища чи кешу) та надає єдиний інтерфейс для доступу до них. View шар складається з Activity та Fragment компонентів Android, які визначають вигляд екранів та обробляють взаємодію користувача. ViewModel шар містить логіку представлення та управління станом, який підписується на дані з Repository.

3.4 Алгоритмізація та програмування програмних модулів

Розробка програмних модулів є критичною частиною реалізації системи. Кожен модуль відповідає за конкретну функціональність системи та повинен бути розроблений з урахуванням принципів чистого коду. Система управління завданнями складається з кількох ключових модулів, кожен з яких реалізує відповідні алгоритми та логіку.

Основні модулі системи представлені у таблиці 3.2.

Основні програмні модулі системи

Модуль	Основні функції	Залежності
TaskManager	Створення, редагування, видалення, отримання завдань	Repository, Firebase
TaskListManager	Управління списками завдань	Repository, Firebase
SubtaskManager	Управління підзадачами, стан виконання	TaskManager, Firebase
AuthenticationModule	Реєстрація, вхід, вихід користувачів	Firebase Auth, SharedPreferences
AIModule	Взаємодія з OpenAI API, генерація рекомендацій	OkHttp, Retrofit, OpenAI API
SyncModule	Синхронізація даних у реальному часі	Firebase, LiveData

3.4.1 Модуль управління задачами (TaskManager). TaskManager являє собою центральний модуль системи, який відповідає за всі операції з завданнями. Цей модуль реалізує створення нових завдань, редагування існуючих, видалення завдань, отримання списку завдань користувача, та оновлення статусу завдань.

Алгоритм створення нового завдання починається з валідації вхідних даних. Система перевіряє, що назва завдання не порожня, що дата у правильному форматі, та що пріоритет належить до допустимих значень. Якщо валідація не пройдена, користувачеві показується помилка. Якщо валідація пройдена успішно, система створює новий об'єкт Task з наданими даними та передає його до Repository для збереження в базі даних.

Алгоритм редагування завдання подібний до створення, але замість створення нового об'єкта система завантажує існуюче завдання з бази, оновлює його поля новими значеннями, та зберігає змінене завдання назад у базу. Алгоритм видалення завдання передає ідентифікатор завдання до Repository, який видаляє відповідний документ з Firestore. Система гарантує, що користувач може видалити лише власні завдання, перевіряючи ідентифікатор власника перед видаленням.

Алгоритм отримання списку завдань користувача сукупно дістає всі завдання з Repository та фільтрує їх за ідентифікатором користувача. Repository використовує слухачі Firestore для отримання оновлення у реальному часі. Коли дані поступають з бази, вони перетворюються в об'єкти Android класу Task та передаються до ViewModel, яка оновлює LiveData, що тригери перерисовування інтерфейсу.

3.4.2 Модуль управління списками завдань. Модуль управління списками завдань дозволяє користувачеві організовувати свої завдання в групи для кращої організації. Користувач може створювати нові списки, редагувати їх назву та колір, видаляти списки та переглядати завдання в межах конкретного списку. Алгоритм створення списку аналогічний алгоритму створення завдання: валідація даних, створення об'єкта TaskList та передача до Repository для збереження. При створенні списку система випадково обирає один з наперед визначених кольорів для обозначення списку.

3.4.3 Модуль автентифікації. Модуль автентифікації відповідає за реєстрацію нових користувачів та вхід існуючих користувачів у систему.

Алгоритм реєстрації приймає логін та пароль від користувача, перевіряє, що логін не порожній та має достатню довжину, генерує email-адресу у форматі `username@taskapp.local`, передає email та пароль до Firebase Authentication для створення облікового запису. Якщо Firebase успішно створив обліковий запис, система зберігає ім'я користувача у `SharedPreferences` та встановлює прапор `isLoggedIn` у `true`.

Алгоритм входу подібний: перевіряє дані, генерує email з логіна, передає до Firebase для автентифікації. Якщо автентифікація успішна, Firebase видає токен, який використовується для подальших запитів до Firestore. Алгоритм виходу з системи видаляє токен з пам'яті, видаляє дані з `SharedPreferences` та перенаправляє користувача на екран входу.

3.4.4 Модуль взаємодії з AI. Модуль взаємодії з AI забезпечує функціональність штучного інтелекту для допомоги користувачам. Алгоритм відправки повідомлення до AI починається з формування запиту у форматі JSON. Запит містить модель AI (`gpt-4o-mini`), систему повідомлень з системним контекстом (який включає інформацію про завдання користувача), та історія розмови.

OkHttp відправляє цей запит до сервера OpenAI через HTTPS. Сервер обробляє запит та повертає відповідь, яка парситься та передається користувачеві. Важливо відзначити, що історія розмови зберігається лише в оперативній пам'яті протягом поточного сеансу та видаляється при закритті екрану чату. Це рішення прийнято для забезпечення конфіденційності користувачів та економії місця в базі даних.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

Успішне впровадження та експлуатація мобільного додатка управління завданнями вимагає ретельного планування та дотримання низки рекомендацій. Цей розділ містить практичні вказівки щодо тестування системи, проведення пілотного впровадження, конфігурації для різних пристроїв та обслуговування системи на етапі експлуатації.

4.1 Тестування системи

Тестування є критичною фазою розробки, яка гарантує, що система працює як передбачено та відповідає вимогам користувачів. Комплексна стратегія тестування включає кілька рівнів тестування, кожен з яких фокусується на різних аспектах системи. Без ретельного тестування розроблений код може містити прихований дефекти, які проявляться у виробництві та негативно вплинуть на користувацький досвід. Тестування являє собою не лише пошук помилок, але й процес верифікації того, що система повною мірою реалізує встановлені вимоги та очікування користувачів.

Важливість тестування неможливо переоцінити. На ранніх етапах розробки, коли система ще становиться, тестування допомагає виявити дизайн-проблеми та архітектурні недоліки, коли їх ще дешево виправляти. На пізніших етапах тестування забезпечує впевненість, що система готова до видачі користувачам. Якість тестування безпосередньо впливає на якість кінцевого продукту та рівень задоволення користувачів.

4.1.1 Планування тестування. Перед початком тестування необхідно розробити план тестування, який визначає цілі, масштаб, розклад та ресурси для тестування. План тестування повинен включати тест-кейси (конкретні сценарії для тестування), очікувані результати та критерії прийнятності.

Основні тест-кейси для системи управління завданнями включають: створення, редагування та видалення завдань; управління списками завдань; роботу з підзадачами; тестування автентифікації та управління сесіями; синхронізацію даних між пристроями; офлайн функціональність; взаємодію з AI-асистентом; валідацію введених даних; тестування безпеки та захисту даних.

Рівні тестування та їх характеристики представлені у таблиці 4.1.

Таблиця 4.1

Рівні тестування системи

Рівень тестування	Що тестується	Інструменти
Unit тести	Окремі методи та функції	JUnit, Mockito
Integration тести	Взаємодія компонентів	Espresso, Firebase Emulator
UI/E2E тести	Користувацький інтерфейс	Espresso, UI Automator
Функціональні тести	Повний цикл операцій	Manual + Automated
Performance тести	Швидкість та ресурси	Android Profiler

4.1.2 Тестування на реальних пристроях. Хоча емулятор в Android Studio дозволяє виконувати значну частину функціонального тестування програмного забезпечення, повноцінна перевірка мобільного застосунку неможлива без тестування на реальних фізичних пристроях. Емулятор не може повністю відтворити поведінку справжнього апаратного забезпечення, продуктивність процесора, роботу акумулятора, сенсорів, камери, GPS,

стабільність мережі та інші фактори, які безпосередньо впливають на роботу системи.

Реальні мобільні пристрої відрізняються між собою технічними характеристиками: обсягом оперативної пам'яті, потужністю процесора, роздільною здатністю екрана, версіями операційної системи та особливостями оболонок виробників. Через це одна й та сама програма може працювати по-різному на різних моделях смартфонів або планшетів. Саме тому тестування необхідно проводити на декількох типах пристроїв – від бюджетних моделей із мінімальними характеристиками до сучасних флагманських пристроїв.

Особливу увагу потрібно приділяти сумісності застосунку з різними версіями операційної системи Android. Для забезпечення стабільної роботи система повинна підтримувати старі версії ОС, зокрема починаючи з Android 8.0, а також коректно функціонувати на нових версіях платформи. Під час перевірки необхідно тестувати запуск застосунку, роботу інтерфейсу, швидкодію, правильність відображення елементів та стабільність виконання основних функцій.

Також важливим аспектом є адаптивність інтерфейсу до різних розмірів екранів і орієнтацій пристрою. Тестування повинно охоплювати смартфони з малими та великими дисплеями, планшети, а також перевірку роботи застосунку в портретному та горизонтальному режимах. Це дозволяє переконатися, що всі елементи інтерфейсу правильно масштабуються, не виходять за межі екрана та залишаються зручними для користувача.

Крім цього, під час тестування потрібно оцінювати використання ресурсів пристрою. Надмірне споживання оперативної пам'яті, процесорного часу або заряду батареї може негативно впливати на роботу смартфона та викликати невдоволення користувачів. Тому система повинна бути оптимізована для енергозбереження та ефективного використання ресурсів навіть при тривалому використанні.

Окремо слід перевіряти поведінку застосунку в різних умовах мережевого з'єднання. Необхідно тестувати роботу при швидкому Wi-Fi з'єднанні, мобільному інтернеті, повільному інтернеті, нестабільному сигналі та повній відсутності мережі. У режимі офлайн застосунок повинен коректно обробляти помилки, повідомляти користувача про відсутність підключення та, за можливості, забезпечувати доступ до локально збережених даних. Це дозволяє гарантувати стабільність роботи системи незалежно від умов використання.

4.1.3 Тестування безпеки. Безпека даних користувачів є одним із найважливіших аспектів при розробці сучасних інформаційних систем та мобільних застосунків. Під час тестування програмного забезпечення необхідно приділяти особливу увагу захисту персональних даних, конфіденційної інформації та стабільності роботи системи в умовах потенційних кіберзагроз. Неналежний рівень безпеки може призвести до витоку даних, несанкціонованого доступу до акаунтів користувачів, фінансових втрат та втрати довіри до системи.

Одним із основних етапів тестування безпеки є перевірка механізмів аутентифікації та авторизації. Система повинна коректно перевіряти особу користувача під час входу та обмежувати доступ лише до дозволених ресурсів. Необхідно тестувати, чи не може користувач отримати доступ до даних інших користувачів шляхом зміни URL-адреси, параметрів запиту або маніпуляцій із токенами доступу. Також слід перевіряти стійкість системи до підбору паролів, багаторазових невдалих спроб входу та автоматизованих атак типу brute force.

Важливим компонентом безпеки є шифрування даних під час передачі мережею. Для цього система повинна використовувати захищений протокол HTTPS із сучасними алгоритмами шифрування. Тестування повинно перевіряти, чи всі мережеві запити передаються через захищене з'єднання та чи відсутня можливість перехоплення конфіденційної інформації. Особливу увагу необхідно приділити передачі логінів, паролів, токенів авторизації та персональних даних користувачів.

Також критично важливим є безпечне зберігання паролів та інших чутливих даних. Паролі не повинні зберігатися у відкритому вигляді ні на сервері, ні на мобільному пристрої. Для їхнього захисту необхідно використовувати сучасні алгоритми хешування та механізми шифрування. Під час тестування слід перевіряти, чи неможливо отримати доступ до збережених облікових даних через файлову систему, базу даних або резервні копії.

Тестування безпеки також повинно включати перевірку системи на стійкість до різних типів атак. Однією з найпоширеніших загроз є SQL-ін'єкції та інші види ін'єкційного введення, які можуть дозволити зловмиснику отримати доступ до бази даних або змінити її вміст. Для запобігання таким атакам необхідно використовувати параметризовані запити та проводити сувору перевірку введених даних. Крім того, слід тестувати систему на захист від атак (міжсайтового скриптингу), CSRF-атак, підробки запитів та інших поширених кіберзагроз.

Окрему увагу потрібно приділяти валідації введених користувачем даних. Усі поля введення повинні перевірятися як на стороні клієнта, так і на стороні сервера. Це дозволяє уникнути помилок обробки даних, переповнення буфера, введення шкідливого коду або некоректних символів. Тестування повинно включати введення некоректних, надто великих або спеціально модифікованих даних для перевірки стабільності системи.

Для проведення тестування безпеки рекомендується використовувати спеціалізовані інструменти статичного та динамічного аналізу коду. Статичний аналіз дозволяє виявити потенційні вразливості без запуску програми, а динамічний аналіз – перевірити поведінку системи під час роботи. Також ефективним методом є пенетраційне тестування, під час якого спеціалісти імітують дії потенційного зловмисника з метою пошуку слабких місць системи.

Додатково необхідно регулярно проводити аудит безпеки програмного забезпечення та перевіряти актуальність використовуваних бібліотек і фреймворків. Багато вразливостей виникає через використання застарілих

компонентів, для яких уже існують відомі методи атак. Саме тому важливо своєчасно оновлювати залежності, стежити за повідомленнями про нові вразливості та оперативно усувати знайдені проблеми.

4.1.4 Користувацьке тестування (UAT). На завершальному етапі тестування програмного забезпечення проводиться користувацьке тестування – User Acceptance Testing (UAT). Це один із найважливіших етапів перевірки системи, оскільки саме він дозволяє визначити, наскільки розроблений застосунок відповідає реальним потребам кінцевих користувачів та чи готовий продукт до повноцінного використання після релізу. На відміну від технічного тестування, яке виконується розробниками або тестувальниками, UAT проводять безпосередньо користувачі, які працюватимуть із системою у звичайних умовах.

Основною метою User Acceptance Testing є перевірка того, чи виконує система всі необхідні бізнес-вимоги, чи є вона зручною у використанні та чи забезпечує позитивний користувацький досвід. Під час UAT оцінюється не лише правильність роботи функцій, а й логічність інтерфейсу, зрозумілість навігації, швидкість виконання дій, стабільність роботи застосунку та комфорт користувача під час взаємодії із системою.

У процесі User Acceptance Testing тестувалися основні функціональні можливості системи. Зокрема перевірялася робота реєстрації та авторизації користувачів, коректність входу в систему, відновлення доступу до акаунта, стабільність роботи особистого кабінету користувача, правильність відображення інформації та взаємодія між екранами застосунку. Також тестувалася обробка помилок та повідомлень системи у випадках неправильного введення даних або відсутності мережевого підключення.

Окрему увагу було приділено тестуванню користувацького інтерфейсу та зручності використання застосунку. Користувачі оцінювали зрозумілість елементів меню, логічність розташування кнопок, швидкість навігації між сторінками, читабельність тексту та загальний дизайн системи. Це дозволило

виявити проблеми, які могли залишитися непоміченими під час автоматизованого або функціонального тестування.

Також у межах UAT проводилася перевірка продуктивності застосунку в реальних умовах використання. Було протестовано швидкість запуску системи, час завантаження сторінок, стабільність роботи при тривалому використанні та реакцію застосунку на одночасне виконання декількох дій користувачем. Крім цього, перевірялася робота системи при нестабільному інтернет-з'єднанні та в офлайн-режимі.

4.2 Вимоги до апаратного та програмного забезпечення

Для успішного впровадження та експлуатації системи управління завданнями необхідно забезпечити відповідне апаратне та програмне забезпечення як на стороні серверу, так і на стороні мобільного клієнта.

4.2.1 Вимоги до мобільного пристрою користувача. Система розроблена для операційної системи Android. Мінімальна версія Android, на якій може працювати додаток, встановлена на рівні Android 8.0 (API рівень 26). Це забезпечує сумісність з більшістю мобільних пристроїв, які знаходяться в обігу.

Мінімальні та рекомендовані вимоги до пристрою наведені у таблиці 4.2.

Вимоги до мобільного пристрою

Параметр	Мінімально	Рекомендовано
Версія Android	8.0 (API 26)	12.0+ (API 31+)
Оперативна пам'ять (RAM)	2 GB	4 GB або більше
Вільне місце на диску	100 MB	200 MB
Розмір екрану	4.5 дюймів	5.5 дюймів та більше
Роздільна здатність	720x1280 пікселів	1080x1920 пікселів або більше
Підключення	WiFi або мобільна мережа	Широкопasmугова з'єднання (WiFi або 4G+)
Батарея	2000 mAh	3000 mAh або більше

Додаток займає приблизно 80–100 MB дискового простору після установки. Для оптимальної роботи рекомендується мати не менше 200 MB вільного місця, щоб забезпечити кешування даних та зберігання тимчасових файлів.

4.2.2 Вимоги до серверної інфраструктури. Система використовує Firebase як серверну інфраструктуру. Firebase автоматично масштабується та управляється Google, тому розробник не повинен про це турбуватися. Однак, рекомендується дотримуватися найкращих практик при конфігурації Firebase.

Для Firebase Firestore рекомендується встановити відповідні правила безпеки, які забезпечать, що користувачі можуть отримати доступ лише до своїх даних. База даних повинна мати резервне копіювання (яке Firebase забезпечує

автоматично), та повинна бути налаштована збір аналітики для моніторингу використання.

Рекомендується встановити ліміти на використання Firebase Firestore для уникнення несподіваних витрат. Firebase надає варіанти тарифікації як 'pay-as-you-go', так і фіксованої вартості, залежно від очікуваного обсягу використання.

4.2.3 Вимоги до розробного середовища. Для розробки та підтримки системи необхідне розробне середовище з наступними компонентами:

Android Studio (версія 4.2 та вище) – інтегроване середовище розробки

Java Development Kit (JDK) версії 11 або вище

Android SDK з API рівнем 26 та вище

Gradle 7.0 або вище для управління залежностями

Git для контролю версій

Дополнительные інструменти для тестування (JUnit, Espresso, тощо)

Розробне середовище повинно мати наступні мінімальні вимоги: процесор Intel i5 або еквівалент, мінімум 8 GB оперативної пам'яті (рекомендовано 16 GB), мінімум 10 GB вільного місця на диску для Android SDK та додатків.

4.2.4 Мережеві вимоги. Система вимагає активного з'єднання з інтернетом для повної функціональності, хоча містить деякі можливості офлайн роботи. Мінімальна рекомендована швидкість інтернету становить 2 Mbps для завантаження та 1 Mbps для закачування. Для оптимальної роботи рекомендується з'єднання 10 Mbps та більше.

Система сумісна як з WiFi, так і з мобільними мережами (3G, 4G, 5G). Однак, рекомендується використовувати WiFi для синхронізації великих обсягів даних та для економії мобільного трафіку.

4.2.5 Рекомендації щодо безпеки. Для забезпечення безпеки системи та даних користувачів рекомендується дотримуватися наступних практик:

Користувачі повинні використовувати сильні паролі (мінімум 8 символів, включаючи літери, цифри та спеціальні символи)

Мобільні пристрої повинні мати встановлені останні оновлення безпеки операційної системи

Додаток повинен мати дозвіл на доступ до мінімально необхідних даних на пристрої

Регулярно проводити оновлення системи для отримання виправлень безпеки

4.2.6 Плани масштабування. При зростанні кількості користувачів система повинна мати можливість масштабування. Firebase автоматично масштабується, однак слід моніторити використання та витрати. При виході на більші обсяги користувачів можуть потребуватися наступні дії: оптимізація запитів до бази даних, встановлення індексів для прискорення пошуку, масштабування OpenAI API використання, розглядання географічного розподілу серверів для зменшення затримок.

Рекомендується проводити регулярне тестування масштабованості за допомогою навантажувального тестування, яке симулює роботу великої кількості користувачів одночасно.

4.3 Склад інсталяційного пакету

Інсталяційний пакет додатку розповсюджується у вигляді файлу з розширенням .apk (Android Package Kit) і містить усі необхідні компоненти для встановлення та роботи застосунку на мобільному пристрої під управлінням операційної системи Android версії 8.0 і вище.

До складу інсталяційного пакету входять такі компоненти:

Виконуваний код додатку – скомпільовані файли у форматі .dex (Dalvik Executable), що містять байт-код усіх класів Java, реалізованих у проєкті.

Ресурси інтерфейсу користувача – XML-файли розміток екранів, файли стилів та тем оформлення, векторні та растрові графічні елементи різних роздільних здатностей (drawable, mipmap).

Рядкові ресурси та локалізація – файли strings.xml із текстовими значеннями, що використовуються в інтерфейсі.

Конфігураційний файл AndroidManifest.xml – містить декларацію компонентів додатку (активностей, фрагментів, сервісів), необхідних дозволів та мінімальних вимог до версії Android.

ВИСНОВКИ

У процесі виконання даної бакалаврської кваліфікаційної роботи була успішно розроблена повнофункціональна мобільна система для планування та контролю виконання завдань на платформі Android. Розроблений додаток вирішує актуальну проблему, з якою стикаються мільйони людей щодня – необхідність організації та контролю великої кількості завдань у умовах постійного інформаційного навантаження.

Основною метою розробки було створення зручного, інтуїтивно зрозумілого та функціонального додатку, який не тільки дозволяє користувачу створювати та редагувати завдання, але й надає інтелектуальну підтримку через інтегрований штучний інтелект. Ця мета була повністю досягнута. Додаток забезпечує користувачеві повний спектр функціональних можливостей для управління своєю діяльністю.

Основні результати розробки включають:

Розроблена трирівнева ієрархічна архітектура системи (Список завдань – Завдання – Підзадачі), яка відображає природний спосіб мислення людини при плануванні. Ця структура дозволяє користувачу ефективно організовувати як великі проєкти, так і дрібні завдання.

Реалізована система аутентифікації на основі Firebase Authentication, що забезпечує безпечний доступ користувачів до своїх даних та синхронізацію інформації на декількох пристроях.

Впроваджена хмарна база даних Firebase Realtime Database, яка гарантує збереження даних у реальному часі, підтримку офлайн-режиму та надійність роботи системи на будь-яких пристроях.

Розроблена модульна архітектура на основі MVVM-паттерну, що забезпечує простоту тестування, масштабування та підтримки коду. Система

складається з окремих модулів: TaskManager, TaskListManager, AuthenticationModule, AIModule та SyncModule.

Інтегрований штучний інтелект на основі OpenAI API, який надає користувачеві інтелектуальну підтримку при формуванні завдань, генеруванні дескрипцій та отриманні рекомендацій щодо планування.

Розроблений інтуїтивний користувацький інтерфейс на основі Material Design, який забезпечує зручність використання без надлишкової складності. Дизайн було оптимізовано для швидкого доступу до основних функцій.

Проведена комплексна стратегія тестування на кількох рівнях: unit-тестування компонентів, інтеграційне тестування взаємодії модулів, UI/E2E тестування користувацьких сценаріїв, функціональне тестування основних можливостей, тестування продуктивності та безпеки.

Апробація розробленої системи показала, що додаток успішно вирішує поставлені завдання. Під час тестування на реальних пристроях з різними версіями Android було підтверджено стабільність роботи, коректність реалізації основних функцій та відсутність критичних помилок. Користувачі відзначили, що інтерфейс легко розуміється без додаткових пояснень, а штучний інтелект дійсно допомагає у плануванні.

З технічної точки зору розроблена система демонструє високий рівень програмної інженерії. Використання сучасних технологій та архітектурних паттернів забезпечило створення масштабованої, тестованої та легко підтримуваної системи. Модульна архітектура дозволяє легко додавати нові функції без переписування існуючого коду.

Розроблений додаток має широкий потенціал застосування. Він може використовуватися студентами для управління навчальним навантаженням, фахівцями для контролю проєктних задач, підприємцями для управління командними завданнями, та будь-якою людиною, яка потребує систематизації

своїх справ. Стабільна архітектура забезпечує можливість розширення функціональності у майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google. Android Developers Documentation. Official documentation for Android development. URL: <https://developer.android.com/docs> (дата звернення: 18.09.2025).
2. Google. Android Studio User Guide. Official guide for Android Studio IDE. URL: <https://developer.android.com/studio/intro> (дата звернення: 18.09.2025).
3. Google. Firebase Documentation. Complete Firebase documentation for backend services. URL: <https://firebase.google.com/docs> (дата звернення: 21.10.2025).
4. Google. Firebase Realtime Database. Official documentation for real-time database. URL: <https://firebase.google.com/docs/database> (дата звернення: 28.10.2025).
5. Google. Firebase Authentication. Security and authentication guide. URL: <https://firebase.google.com/docs/auth> (дата звернення: 08.12.2025).
6. OpenAI. OpenAI API Documentation. API documentation for AI integration. URL: <https://openai.com/docs/api> (дата звернення: 25.12.2025).
7. Google. Material Design Guidelines. Design system for Android applications. URL: <https://material.io/design> (дата звернення: 21.01.2026).

ДОДАТКИ

ДОДАТОК А

Таблиця функціональних тестових сценаріїв, які використовувалися для перевірки основних можливостей програмної системи

№	Назва сценарію	Передумови	Кроки виконання	Очікуваний результат	Фактичний результат
ТС –01	Реєстрація нового користувача	Додаток встановлено. Користувач не зареєстрований	1. Відкрити додаток. 2. Натиснути «Зареєструватися». 3. Ввести email та пароль. 4. Підтвердити реєстрацію	Обліковий запис створено. Користувач перенаправлений на головний екран	Обліковий запис успішно створено
ТС –02	Вхід у систему	Обліковий запис існує	1. Відкрити додаток. 2. Ввести email та пароль. 3. Натиснути «Увійти»	Користувач автентифікований, відображається головний екран	Авторизація успішна
ТС –03	Вихід із системи	Користувач авторизований	1. Відкрити меню. 2. Натиснути «Вийти». 3. Підтвердити дію	Сесія завершена, відображається екран входу	Вихід виконано успішно
ТС –04	Створення списку завдань	Користувач авторизований	1. Перейти до розділу списків. 2. Натиснути кнопку додавання. 3. Ввести назву списку. 4. Зберегти	Новий список відображається у переліку	Список створено та відображено
ТС –05	Редагування списку завдань	Існує принаймні один список	1. Відкрити список. 2. Натиснути «Редагувати». 3. Змінити назву. 4. Зберегти	Назва списку оновлена	Зміни збережені коректно
ТС –06	Видалення списку завдань	Існує список без вкладених завдань	1. Відкрити список. 2. Натиснути «Видалити». 3. Підтвердити	Список видалено, він відсутній у переліку	Список успішно видалено

№	Назва сценарію	Передумови	Кроки виконання	Очікуваний результат	Фактичний результат
ТС –07	Створення завдання	Існує список завдань	1. Відкрити список. 2. Натиснути «Додати завдання». 3. Ввести назву та опис. 4. Зберегти	Завдання з'являється у списку	Завдання створено та відображено
ТС –08	Встановлення дедлайну завдання	Існує завдання	1. Відкрити завдання. 2. Натиснути «Дедлайн». 3. Обрати дату і час. 4. Зберегти	Дата дедлайну збережена і відображається	Дедлайн встановлено коректно
ТС –09	Позначення завдання як виконаного	Існує невиконане завдання	1. Відкрити список. 2. Натиснути на чекбокс поряд із завданням	Завдання позначено виконаним, статус змінено	Статус оновлено успішно
ТС –10	Редагування завдання	Існує завдання	1. Відкрити завдання. 2. Натиснути «Редагувати». 3. Змінити поля. 4. Зберегти	Оновлені дані відображаються у завданні	Редагування виконано успішно
ТС –11	Видалення завдання	Існує завдання	1. Відкрити завдання. 2. Натиснути «Видалити». 3. Підтвердити	Завдання видалено зі списку	Завдання успішно видалено
ТС –12	Створення підзавдання	Існує завдання	1. Відкрити завдання. 2. Натиснути «Додати підзавдання». 3. Ввести назву. 4. Зберегти	Підзавдання відображається у межах батьківського завдання	Підзавдання створено та відображено
ТС –13	Позначення підзавдання як виконаного	Існує підзавдання	1. Відкрити завдання. 2. Натиснути чекбокс поряд із підзавданням	Підзавдання позначено виконаним	Статус підзавдання оновлено
ТС –14	AI-консультація	Користувач авторизований, доступ до Gemini API активний	1. Відкрити розділ консультацій. 2. Ввести запит. 3. Надіслати	Отримано відповідь від AI-асистента	Відповідь отримано коректно

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення веборієнтованої системи обліку та відтворення музичного контенту» є результатом моєї особистої праці.

2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.

3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- o ☒ інструменти генеративного ШІ не використовувалися.
- o ☒ інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL)
- . ChatGPT, Gemini були використані для:
- ☒ перекладу іншомовних джерел;
 - ☒ стилістичного редагування та перевірки граматики;
 - ☐ допомоги у написанні/відлагодженні програмного коду;
 - ☐ інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

« » 2026 p.

Ступніцький Артем

(підпис)