**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Сумський державний університет**
Факультет електроніки та інформаційних технологій
Кафедра комп’ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____________
(підпис)
«    » червня 2024 р.

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня бакалавр**

зі спеціальності 122 - Комп’ютерних наук, освітньо-професійної програми «Інформатика» на тему: «Інтелектуальна освітня система Maverkick»

Здобувача групи ІН - 01 Солов’я Євгена Юрійовича

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

Євген СОЛОВЕЙ
_____________
(підпис)

Керівник,
старший викладач, кандидат технічних наук          Артем КОРОБОВ          _____________
(підпис)

**Суми – 2024**

**Сумський державний університет**
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
____________ Ігор ШЕЛЕХОВ

# ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

**на здобуття освітнього ступеня бакалавра**

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-01 Солов'я Євгена Юрійовича

1. Тема роботи: «Інтелектуальна освітня система Maverkick»
затверджую наказом по СумДУ від *«22» квітня 2024р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 29 травня 2024 року*
3. Вхідні дані до кваліфікаційної роботи ______
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд програм та технологій, які використовуються в сучасних навчальних застосунках 3) Розробка інтелектуальної системи зі створення щоденних персоналізованих навчальних планів 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)______
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |

7. Дата видачі завдання «06» травня 2024 р.

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

# КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів кваліфікаційної роботи | Термін виконання | Примітка |
|---|---|---|---|
| *1* | *Аналіз проблем в галузі онлайн-освіти, постановка й формування завдань дослідження галузі* | 06.05.24-09.05.24 | |
| *2* | *Олгяд технологічних підходів для вирішенні освітніх проблем* | 10.05.24-13.05.24 | |
| *3* | *Розробка інтелектуальної системи зі створення щоденних персоналізованих навчальних планів* | 14.05.24-20.05.24 | |
| *4* | *Аналіз отриманих результатів* | 21.05.24-23.05.24 | |
| *5* | *Оформлення пояснювальної записки до кваліфікаційної роботи* | 24.05.24-31.05.24 | |

Здобувач вищої освіти ____________________ (підпис)

Керівник ____________________ (підпис)

# АНОТАЦІЯ

**Записка:** 100 стор., 16 рис., 13 табл., 2 додатки, 23 використаних джерела

**Обґрунтування актуальності теми роботи** — Тема роботи є актуальною, оскільки вона обумовлена розвитком онлайн-освіти і необхідністю знаходження кращих підходів для подачі навчальних матеріалів для втримання концентрації користвачів та забезпечення кращого засвоєння інформації.

**Об'єкт дослідження** — процес щоденного навчання з використанням коротких навчальних матеріалів, обмеження кількості інформації для створення кращого навчального середовища

**Предмет дослідження** — методи структуризації освітніх матеріалів для людей, які страждають від обмеженої концентрації уваги.

**Мета Роботи** — розробка мобільного додатку, який динамічно адаптуватиме навчальний контент відповідно до часу, який щоденно виділяє користувач, з метою підвищення ефективності самостійного вивчення матеріалу та використання коротких проміжків концентрації уваги для кращого засвоєння інформації.

**Методи дослідження** — структуризація освітніх матеріалів, створення щоденних навчальних планів, інтергграція GPT3.5 для взаємодії користувача з матеріалами

**Результати** — розроблено інтелектуальну систему, яку можна завантажити з Google Play на смартфон, в якій користувач вводить коротку інформацію про себе, обирає ті курси, які його цікавлять і зможе навчатися ту кількість хвилин в день, яку він захоче, використовуючи уроки, які обмежені в часі, що дозволить утримувати концентрацію користувача. Кожен урок обмежений в часі його проходженння (в діапазоні до 5 хвилин) для кращого засовоєння матеріалу та утримання концентрації уваги. Доступно два типи уроків: текстові та відео.

ІНТЕЛЕКТУАЛЬНА СИСТЕМА, ANDROID-ДОДАТОК, ЩОДЕННИЙ НАВЧАЛЬНИЙ ПЛАН, BITE-SIZED LEARNING, КОНЦЕНТРАЦІЯ УВАГИ, KOTLIN, GPT-3.5

# ЗМІСТ

Вступ ... 6
1. Аналітичний Огляд ... 8
1.1 Огляд сучасних технологій в освіті ... 8
1.2 Огляд існуючих сервісів освітніх послуг ... 10
1.3 Аналіз сучасних освітніх сервісів та ринку ... 12
1.4 Постановка задачі ... 15
2. Методи вирішення Проблеми ... 17
2.1 Інформаційна модель системи ... 17
2.2 Функціональна структура додатку ... 22
2.3 Вибір технологічних засобів для реалізації задачі ... 27
3. Дизайн та програмна Реалізація ... 33
3.1 Дизайн Інтерфейсу Додатку ... 33
3.2 Програмна реалізація ... 40
3.3 Тестування ... 77
4. Висновки ... 79
Список Використаних Джерел ... 81
Додаток А ... 83
А.1 Моделі даних ... 83
А.2 Денний навчальний план ... 93

# ВСТУП

Сьогодні ми стаємо свідками справжньої революції в доступності інформації. Дивовижні технологічні зрушення кардинально змінюють те, як ми отримуємо знання та навчаємося. Світ мчить вперед шаленими темпами, і освітня сфера не стоїть осторонь цих перетворень.

Демократизація освіти - одна з ключових тенденцій нашого часу. Ми, звичайні люди, отримали безпрецедентний доступ до величезних обсягів інформації - від класичної літератури до передових наукових напрацювань. Це відкриває небачені можливості для саморозвитку та вдосконалення. Проте, як ви певно здогадуєтесь, така доступність знань породжує і нові виклики.

Головний з них - це, власне, здатність людини ефективно засвоювати та утримувати увагу на великих масивах різноманітної інформації. Дослідження показують, що середній час концентрації уваги стрімко падає, ледь встигаючи за золотою рибкою [1]. Згідно зі статистикою середній час концентрації уваги в 2023 складає приблизно 8.25 секунд, що на 4.25 секунди менше ніж у 2000 році. І згідно певними прогнозами, даний показник може падати ще більше. Золота рибка має даний показник в 9 секунд. Це вимагає від нас пошуку нових підходів до подачі та структурування освітнього контенту.

Саме тому я вирішив присвятити свою бакалаврську роботу розробці мобільного додатку, який допоможе оптимізувати процес самонавчання з урахуванням особливостей сприйняття та уваги сучасних користувачів. Мій задум полягає у створенні адаптивної платформи, що зможе персоналізувати подачу матеріалів під кожного студента. Сподіваюсь, що це стане гідною відповіддю на виклики, з якими ми стикаємось у сфері освіти сьогодні.

Нижче опис головний моментів, які відкриють, про що насправді моя робота:

**Об'єкт дослідження**: Процес адаптивної організації та подання навчального контенту у вигляді коротких відео та текстових матеріалів у мобільних додатках для самостійного вивчення.

**Предмет дослідження:** Методи, технології та алгоритми персоналізації структури і темпу подачі освітніх матеріалів з урахуванням особливостей уваги та сприйняття інформації користувачами.

**Гіпотеза:** Розробка мобільного додатку, який динамічно адаптуватиме навчальний контент відповідно до індивідуальних характеристик уваги користувача, дозволить підвищити ефективність самостійного вивчення матеріалу та використовуватиме короткі проміжки концентрації уваги для кращого засвоєння матеріалу.

**Наукова новизна:** Полягає у розробці концептуальної моделі та прототипу мобільного застосунку, здатного персоналізувати процес самоосвіти шляхом оптимізації структури та темпу подачі навчальних матеріалів відповідно до особливостей сприйняття інформації сучасними користувачами.

**Структура:** Дана робота складається з детального аналітичного огляду інсуючих освітніх рішень, постановки задачі, вибору програмних засобів імплементації ідеї, проектування базових інтерефесів, програмної імплементації з використанням мови Kotlin та користувацького тестування розробленого Android-додатку, висновку, списку використаних в роботі джерел та додатку.

# 1. АНАЛІТИЧНИЙ ОГЛЯД

## 1.1 Огляд сучасних технологій в освіті

Світ технологій в галузі освіти рік за роком крокує до беззаперечної мрії людей всіх часів та народів – будь-яка інформація в будь-якому форматі.

Особливо це торкається теми некласичної освіти та використання принципу lifelong learning. Зараз ми мусимо навчатися протягом всього життя, і для цього не треба відвідувати університет знову і знову. Головне – отримати базові навички, а далі з використанням правильний засобів, додатків та книжок ми зможемо розширювати свої горизонти, вивчати як вузькоспеціалізовані, так і загальнолюдські дисципліни. Ці ідеї дали поштовх дали паралельному типу освіти – онлайн-освіта.

Справжнім витоком онлайн-освіти вважається 1960-і роки, коли Дональд Бітцер розробив першу систему електронного навчання - PLATO. Пізніше, у 80-90-і роки, інструменти та методи електронної освіти швидко розвивалися. З'явлення перших ПК у 1980-х роках дало змогу багатьом людям навчатися та здобувати нові навички.

У 90-і роки електронне навчання стало ще популярнішим, а курси стали домінуючим засобом забезпечення доступу до знань. За цей період зросла кількість шкіл і навчальних закладів, які повністю перейшли на дистанційне навчання, що сприяло зростанню доступності освіти для учнів, які раніше були обмежені географією або часом. У 2000-х роках компанії та організації також почали використовувати дистанційне навчання для підвищення кваліфікації своїх працівників.

Перед зародженням Інтернету віддалене навчання використовувало пошту для спілкування між учнями та вчителями, що дало поштовх до зручності та доступності освіти, але залишало значні затримки в обміні зворотнім зв'язком. Проте з розвитком комп'ютерів у 90-ті роки онлайн-освіта стала відомою такою, якою ми її уявляємо сьогодні. Широке використання комп'ютерів та перших стандартизованих програм для офісу в 90-ті роки сприяло розвитку онлайн-освітніх платформ та зміні освітнього контенту з фізичного на цифровий формат [2].

Уже на початку двохтисячних з’явилися потужні і відомі платформи як Edx, Coursera, Udemy.

Швидкість інтернету, можливість зберігання даних більшого розміру дозволила публікації лекцій відомих університетів як Гарвард та MIT. В цей же час максимально почав розвиватися YouTube, який зараз хоча і не є класичною освітньою платформою, та все ж зберігає найбільшу кількість саме освітніх відеоматеріалів.

Все це досягло свого піку в певний момент, адже інформація є цінної тільки до моменту, коли вона поширюється і стає доступною кожному. Особливо це показалося в розвиненому світі, де існує надзвичайна конкуренція в освітній галузі, щодня публікуються сотні навчальних програм та курсів. Але існує певне відчуття, що існуючі підходи, які було створено за останні два десятиліття дають тріщини, оскільки змінилися дві речі:

- Концентрація уваги, можливість фокусуватися на певній задачі чи процесі надзвичайно впала. Найбільший вплив тут здійснили соціальні мережі, де контент стає все коротшим і коротшим, людина звикає і вона вважає це за норму в інших галузях та активностях. Крім того, в освіті на це значно впливає наявність великого різноманіття інформації: різні мови, різні інституції, різні підходи до навчання. Інколи надзвичайно важко обрати, який саме курс ми хочемо пройти на Coursera, оскільки на кожну тему існує декілька варіантів. Але навіть коли ми обрали, то дуже важко мати достатньо уваги та мотивації для того, щоб дійти до кінця. Особисто я маю більше 10 різних курсів, які я закинув, і при цьому лише одиниці, де я дійшов до кінця.
- Персоналізація, можливість отримати інформацію у форматі, який приносить задоволення від процесу навчання. Це надзвичайно важливий крок, адже коли можна передати одну й ту саму думку, ідею, математичну формулу десятками різних способів, в залежності від того, хто є твоїм учнем, тоді це створює справжнє відчуття занурення, подібне до того, що ми відчуваємо в школі. Головним проривом в даній концепції стало застовування великих мовних моделей типу ChatGPT, Gemini, Claude.

Отже, вся структура освіти дещо змінюється і ці два пункти є ключовими в цій зміні, яка повинна принести нові підходи до уже гарно працюючої системи, додаючи більшу персоналізацію, адаптивність до проблем учнів та використання більш інноваційних підходів до створення правильних уроків, які струкрурно дозволять використовувати біологічні можливості мозку людини до концентрації і засвоєння отриманої інформації.

Майбутнє в цьому, в нашій можливості створювати продукти, які дозволяють прибрати цю лінію між класичною(академічною) та некласичною освітою, зробити послуги доступними більшості населення нашої планети.

Враховуючи всі ці тенденції, ми можемо виділити декілька ключових напрямків, які визначають розвиток сучасних освітніх технологій:

1) **Безперешкодний доступ та гнучкість:** Онлайн-освіта скасовує географічні та часові бар'єри, даючи змогу отримувати знання із будь-якої точки світу та в зручному для кожного ритмі. Навчання стає доступним для широких верств населення.
2) **Неосяжний вибір контенту:** Цифрові платформи пропонують величезний асортимент курсів на будь-який смак - від програмування до мистецтва, охоплюючи практично всі галузі знань. Кожен може підібрати програму, що ідеально відповідає його інтересам та кар'єрним прагненням.
3) **Персоналізація та адаптивність:** Застосування інтерактивних методик, адаптивних алгоритмів та штучного інтелекту дозволяє індивідуалізувати навчальний процес під унікальні потреби й здібності кожного учня, оптимізуючи засвоєння матеріалу.
4) **Застосування ігрових механік:** Гейміфікація, тобто включення ігрових елементів у освітній контент, стає потужним інструментом для підвищення залученості та мотивації студентів, перетворюючи навчання на захопливий досвід.

Над цим працюють багато різних компаній, стартапів та громадських організацій, адже зараз відкривається величезний доступ до технологій і ми лише повинні знайти найкращі можливості їх застосування.

Зараз же, ми оглянемо ключових гравців на ринку освітніх послуг, що визначає їхнє місце

## 1.2 Огляд існуючих сервісів освітніх послуг

На сучасному ринку освітніх послуг присутні різноманітні платформи та додатки, спрямовані на надання різноманітних освітніх можливостей. Деякі з них відповідають ключовим трендам у сфері освіти, таким як персоналізація, адаптивність, доступність та залучення користувачів. Нижче наведено огляд головних сегментів цього ринку:

### Онлайн-курси та MOOC

Провідними гравцями в цьому сегменті є Udemy, Udacity, Coursera та платформи масових відкритих онлайн-курсів (MOOC), такі як edX та FutureLearn. Вони пропонують широкий вибір курсів з різних галузей, від програмування та бізнесу до мистецтва та гуманітарних наук. Багато курсів надаються провідними університетами та експертами. Перевагами є доступність, можливість отримання сертифікатів та дипломів. Однак недоліками можуть бути обмежена персоналізація, висока вартість деяких програм та відсутність живого спілкування.

**Спеціалізовані платформи**

Цей сегмент охоплює платформи, зосереджені на певних тематиках, таких як вивчення мов (Duolingo, Babbel), особистісний розвиток (Imprint), STEM-дисципліни (Brilliant) або програмування (FreeCodeCamp, Codecademy). Їхніми сильними сторонами є застосування гейміфікації, інтерактивних методів, співпраця з експертами у своїх галузях та можливість сертифікації. Однак вони обмежені вузькою спеціалізацією.

**Корпоративне навчання**

Компанії, такі як Coursera for Business, Udemy for Business та Pluralsight, пропонують платформи для корпоративного навчання та професійного розвитку співробітників. Вони зосереджені на бізнес-навичках, лідерстві, технологіях та інтегруються з HR-системами компаній.

**Створення власних курсів**

Платформи Teachable, Thinkific та Podia дозволяють створювати, хостити та продавати власні онлайн-курси. Вони надають інструменти для створення контенту, маркетингу та аналітики.

**Навчальні відео та освітні ресурси**

YouTube, Khan Academy та Crash Course є прикладами ресурсів, що пропонують безкоштовні навчальні відео з різних тем, часто створені викладачами та ентузіастами.

Складемо просту табличку, де покажемо сильні і слабкі сторони деяких і цих сервісів. Всі ці оцінки є відносними і ми визнаємо, що наявність слабких сторін є позитивним показником, оскільки це створює можливості для подальшого розвитку та впровадження інновацій.

Таблиця 1.1 Огляд популярних освітніх сервісів із різних категорій

| **Категорія** | **Сервіс** | **Сильні сторони** | **Слабкі сторони** |
|---|---|---|---|
| Онлайн-курси | Coursera | Широкий вибір курсів, співпраця з провідними університетами, сертифікати | Деякі курси вимагають оплати, відсутність адаптивності |
| Вивчення мов | Duolingo | Безкоштовний, гейміфікація, зручний мобільний додаток | Обмежена глибина вивчення, відсутність живого спілкування |
| Особистісний розвиток | Imprint | Курси від відомих психологів, науковий підхід | Вузька спеціалізація, платний контент |
| STEM-дисципліни | Brilliant | Інтерактивні формати, ігри, головоломки | Обмежена тематика |
| Програмування | FreeCodeCamp | Безкоштовний, практичні проекти, спільнота | Обмежена кількість тем, відсутність структурованих курсів |
| Навчальні відео | Khan Academy | Безкоштовні відео високої якості, широкий вибір тем | Відсутність інтерактивності, труднощі з підтримкою уваги |

Кожен із цих сегментів має свої переваги та недоліки, задовольняючи різні потреби користувачів. Незважаючи на широкий вибір, існують можливості для вдосконалення та розробки нових рішень, які б поєднували переваги існуючих платформ та усували їхні недоліки, такі як відсутність персоналізації, обмежена інтерактивність та адаптивність.

Крім того, швидкий розвиток технологій, зокрема штучного інтелекту, віртуальної та доповненої реальності, відкриває нові горизонти для інновацій у галузі освітніх технологій, дозволяючи створювати більш захопливі та ефективні навчальні досвіди.

## 1.3 Аналіз сучасних освітніх сервісів та ринку

Огляд існуючих платформ та додатків освітніх послуг дозволяє виявити як їхні сильні сторони, так і певні недоліки та виклики, з якими стикається галузь.

**Сильні сторони:**

– Доступність знань незалежно від місцезнаходження та часу

- Безпрецедентний вибір курсів з різноманітних дисциплін
- Впровадження персоналізованих підходів та адаптивних алгоритмів
- Активне використання гейміфікації та інтерактивних елементів для залучення користувачів
- Розвиток спеціалізованих рішень для окремих тематик (вивчення мов, особистісний розвиток тощо)

**Недоліки та виклики:**

- Відсутність рішень пов'язаних із підтримкою концентрації та зацікавленості користувача у більшості із сервісів.
- Забезпечення високої якості та авторитетності контенту
- Впровадження штучного інтелекту як віртуального вчителя, який не замінює існуючі матеріали, а дозволяє задавати більше запитань і ширше розкривати теми.

**Перспективи та можливості:**

- Розробка комплексних рішень, що інтегрують переваги існуючих платформ (персоналізація, адаптивність, доступність, залученість)
- Створення мультимовного контенту для забезпечення максимальної доступності в різних регіонах світу
- Впровадження інноваційних підходів до структурування та подачі контенту, орієнтованих на оптимізацію концентрації уваги користувачів
- Застосування передових технологій штучного інтелекту для генерування персоналізованого контенту високої якості
- Розвиток гібридних рішень, що поєднують онлайн та офлайн формати навчання

Загалом, ринок освітніх послуг демонструє значний прогрес у впровадженні інноваційних технологій та методик, проте існують широкі можливості для створення більш комплексних та ефективних рішень, які відповідатимуть мінливим потребам користувачів у персоналізації, адаптивності, доступності та залученні. А враховуючи, що "Сектор освітніх технологій зростає в середньому на рівні 12.9% з 2023 по 2032 рік і досягне доходу у розмірі 429 млрд. доларів" [3], то це показує, що існує дуже багато місця на цьому ринку і потрібно звертати особливу увагу на потреби людей, проблеми, які в них виникають і будувати невеликі, але спеціалізовані рішення, оскільки існує велика кількість сегментів користувачів і в багатьох випадках вони є надзвичайно різними і тому рішення повинні бути різними, які задовольнятимуть потреби цих самих користувачів.

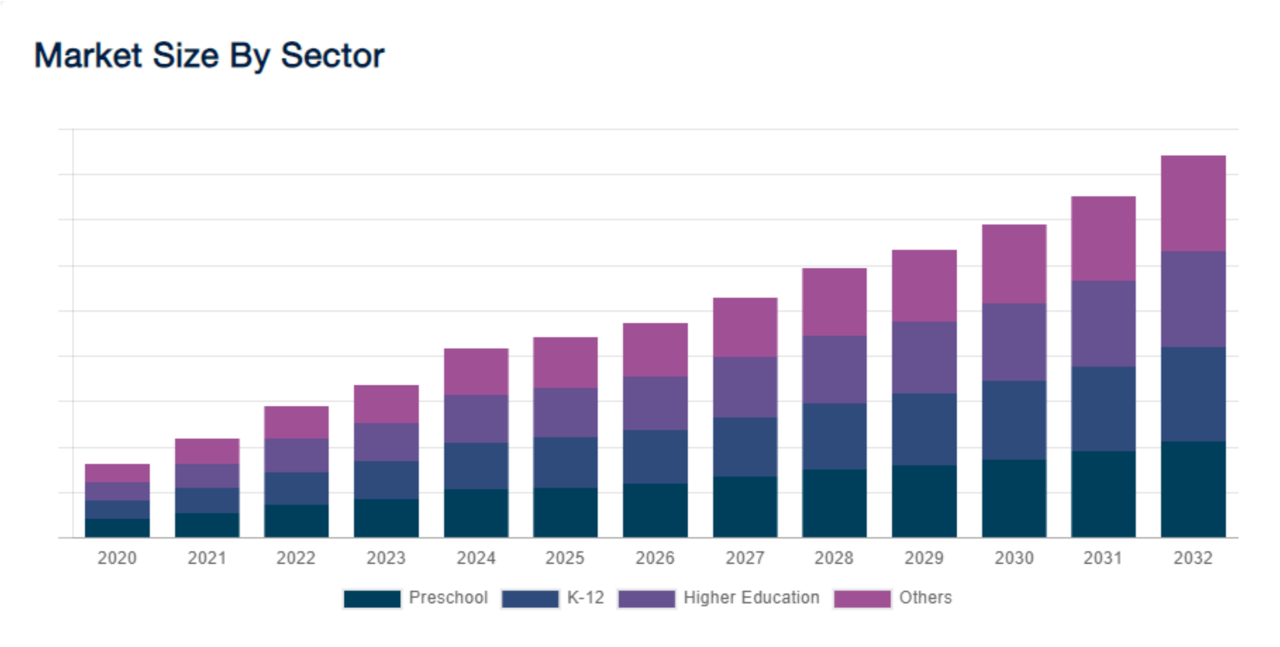


Рисунок 1.1 Сегментація ринку освіти за кінцевими користувачами

Наприклад, з часом помітне зростання частки ринку, яка не відповідає за дошкільну, шкільну та вищу освіту, а спрямована саме на користувачів, які просто навчаються від задоволення або ж намагаються посилити старі чи вивчити нові навички уже після закінчення академічної освіти. В цьому ми бачимо особливу перспективу.

## 1.4 Постановка задачі

Ретельний аналіз сучасного ринку освітніх сервісів та платформ виявив як їхні сильні сторони, такі як доступність знань, персоналізація, гейміфікація та залучення користувачів, так і певні недоліки та виклики, з якими стикається галузь. Незважаючи на широкий вибір курсів та тематик, більшість платформ не мають ефективних рішень для підтримки концентрації уваги користувачів протягом тривалих періодів часу. Ця проблема є критичною, оскільки здатність зберігати фокус є ключовим фактором для успішного засвоєння знань.

Беручи до уваги ці недоліки, а також власні спостереження щодо сучасних тенденцій споживання контенту, зокрема популярність коротких, захопливих форматів, таких як TikTok та додатки для огляду книжок, була сформульована ідея розробки Android додатку для навчання, який би враховував обмежену здатність людей зберігати концентрацію та пропонував відповідний формат подачі матеріалу.

Метою даної роботи є створення простого освітнього рішення, яке синтезує переваги існуючих платформ та усуває їхні недоліки, адаптуючись до сучасних тенденцій споживання контенту. Шляхом поєднання персоналізації, адаптивності та уваги до особливостей людської психології, ми маємо на меті забезпечити ефективний та захопливий процес отримання нових знань.

Ключовими функціями запланованого додатку є:

1) Система реєстрації та налаштувань вподобань користувача: Під час першого запуску додатку користувач матиме можливість вказати свої інтереси та тематики, які його цікавлять, а також визначити бажаний проміжок часу (від 5 до 30 хвилин), який він/вона готові присвятити навчанню щодня. Ці налаштування дозволять персоналізувати досвід навчання відповідно до індивідуальних потреб та уподобань користувача.
2) Можливість вибору курсів, перегляду уроків із яких складається даний курс і реєстрація на нього.
3) Обмеження тривалості навчальних матеріалів: З метою уникнення втрати концентрації та врахування обмеженої здатності людей зберігати увагу протягом тривалого часу, текстові матеріали та відео-уроки будуть обмежені приблизно 5 хвилинами. Це дозволить користувачам споживати контент у зручному для сприйняття форматі, у будь-якому місці, не відчуваючи перевантаження чи нудьги.

4) Можливість проходження кількох різних курсів: В межах визначеного користувачем часового проміжку, додаток запропонує кілька різних навчальних курсів, забезпечуючи різноманітність досвіду та уникаючи монотонності. Це дозволить користувачам вивчати декілька тем за один сеанс, підтримуючи інтерес та задоволення від процесу навчання.
5) Інтеграція великих мовних моделей в уроки, для можливості задати запитання, розкрити тему по-іншому або ж попросити навести приклади для розширення розуміння теми уроки і для того, щоб не просто отримати інформацію, а використати її в житті, а це найкраще робиться за допомогою питань. Для того, щоб симулювати це, ми створимо чат інтерфейс, який буде доступний з кожного урока.
6) Використання тестів або інших способів перевірки знань після кожного уроку, що відрізняється від класичних підходів, де тестування відбувається в кінці певного модуля або тижня. Але наш підхід із використанням невеликої кількості завдань (3-5), які можна виконати за хвилину-дві на мою думку підсилює це розуміння і засвоєння інформації.

Ключовим аспектом розробки є створення захопливого та приємного досвіду для користувача, здатного підтримувати їхню увагу та мотивацію до навчання. Для досягнення цієї мети планується застосувати низку методів, серед яких:

- Гейміфікація: Впровадження ігрових елементів, таких як підрахунок кількості завершених уроків у вигляді значків із цегли.
- Персоналізовані рекомендації: На основі наданої інформації під час реєстрації, система підбере курси, які найближче відповідають інформації.
- Зручний та інтуїтивно зрозумілий інтерфейс: Додаток матиме простий та лаконічний дизайн, зосереджений на зручності використання та забезпеченні безперешкодного доступу до навчальних матеріалів.

Реалізуючи цей проект, ми прагнемо створити сучасне та інноваційне рішення для навчання, яке відповідає потребам та очікуванням користувачів у епоху обмеженої уваги та переважання коротких форматів подачі контенту. Шляхом поєднання передових технологій, персоналізації, адаптивності та уваги до особливостей людської психології, ми маємо на меті забезпечити ефективний, захопливий та комфортний процес отримання нових знань для широкого кола користувачів.

# 2. МЕТОДИ ВИРІШЕННЯ ПРОБЛЕМИ

## 2.1 Інформаційна модель системи

Ми поставили задачу, розписали абстрактну репрезентацію того, що ми хочемо реалізувати під час роботи. Але ми не можемо приступити до дизайну інтерфейсу та програмування, тому що перед цим ми мусимо прописати інформаційну модель для нашої системи.

Це значно спростить процес подальшої розробки, оскільки ми зафіксуємо наше розуміння і створимо фізичну репрезентацію нашої ідеї і запропонованого рішення. Крім того, ми також зробимо візуалізацію, що дозволить візуально орієнтуватися під час проєктування та програмування в коді.

Інформаційна модель є ключовим елементом у процесі розробки будь-якої інформаційної системи. Вона являє собою концептуальне представлення системи, що описує її основні складові, атрибути та взаємозв'язки між ними. Інформаційна модель допомагає зрозуміти, які дані необхідно зберігати та опрацьовувати в системі, а також визначити способи взаємодії між цими даними [5].

Розробка інформаційної моделі є критично важливим етапом у проєктуванні інформаційної системи, оскільки вона закладає міцний фундамент для подальшого розвитку проєкту. Створення інформаційної моделі дозволяє чітко зафіксувати розуміння предметної області, ідеї та запропонованого рішення, перетворюючи їх на фізичну репрезентацію.

Крім того, візуалізація інформаційної моделі за допомогою діаграм або схем забезпечує наочне представлення системи, полегшуючи орієнтацію в процесі проєктування та програмування. Це сприяє кращому розумінню структури даних, взаємозв'язків між ними та загальної архітектури системи, що, в свою чергу, підвищує ефективність розробки та полегшує внесення змін у майбутньому.

Тепер перейдемо до безпосередньої моделі даних, на рис. 12 ми бачимо модель даних і відносини між різними моделями даних, яким чином вони взаємодіють між собою.

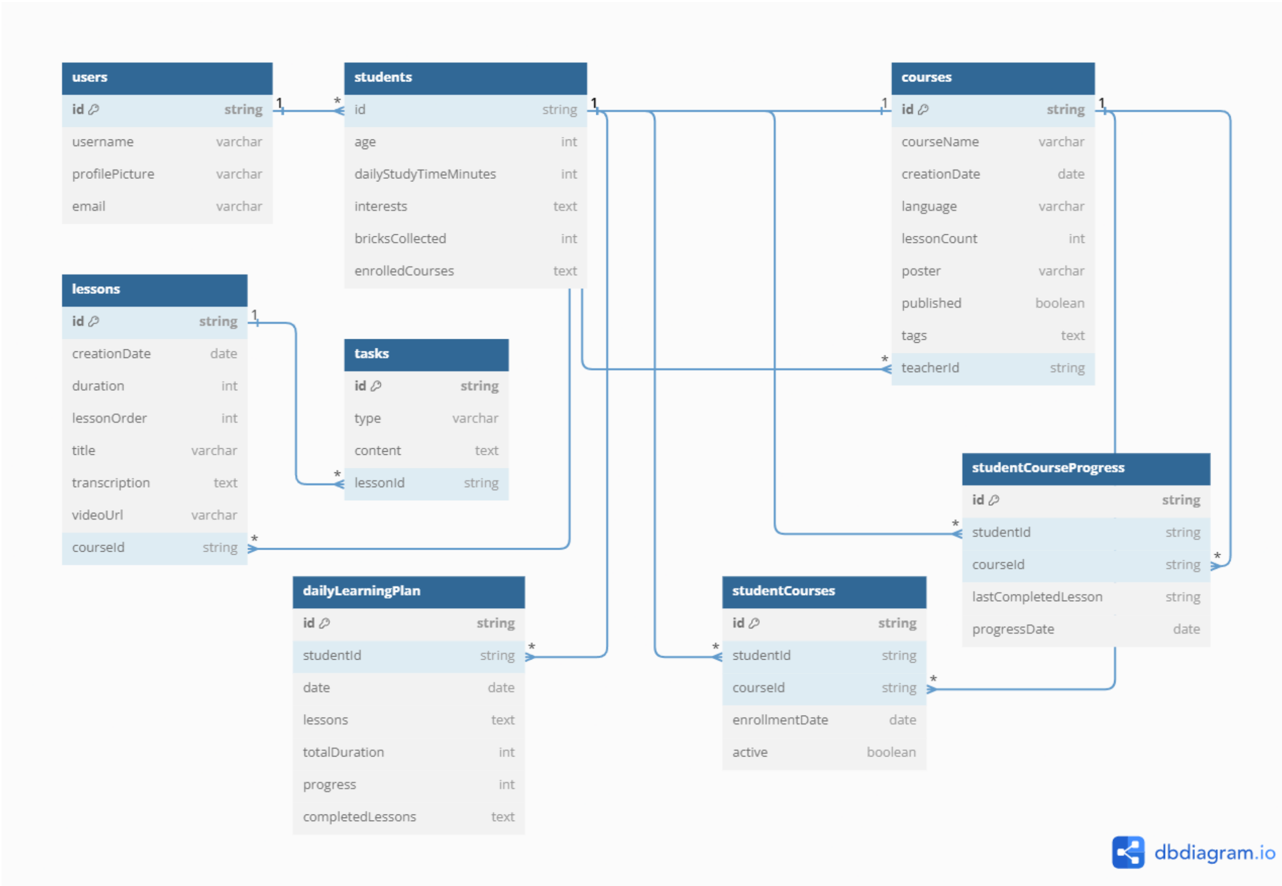


Рисунок 2.1 Модель відносин даних інтелектуальної освітньої системи

**Опис типів даних, необхідних для нашої системи:**

1) Почнемо із таблиці, яка буде презентувати базового користувача User, який зареєструвався в системі, але ще не є студентом, оскільки не надав необхідних даних (Таблиця 2.1 Опис типу даних User).

Таблиця 2.1 Опис типу даних User

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| userId | string | Ідентифікатор користувача |
| email | string | Email адреса користувача |
| profilePicture | string | URL профільного зображення |
| username | string | Ім'я користувача |

2) Коли користувач надає інформацію про кількість хвилин, які він хоче навчатися, наводить дисципліни, які його цікавлять, тоді створюється новий тип Student із тим самим studentId, що і userId для User (Таблиця 2.2 Опис типу даних Student).

Таблиця 2.2 Опис типу даних Student

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| studentId | string | Ідектифікатор студента, який відповідає ідентифікатору користувача |
| bricksCollected | number | Кількість зібраних "цеглин" (скільки уроків завершено) |
| dailyStudyTimeMinutes | number | Кількість хвилин, які людина хоче приділяти навчанню щодня |
| enrolledCourses | array of strings | Список зарахованих курсів |
| interests | array of strings | Інтереси студента |

3) Тип Даних Курс, який дозволить зберігати курси 2 типів: відео та текстові і описувати ключові характеристики, як тривалість чи кількість уроків (Таблиця 2.3 Опис типу даних Course)

Таблиця 2.3 Опис типу даних Course

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| courseId | string | ID курсу |
| authorId | string | ID автора курсу |
| courseName | string | Назва курсу |
| courseType | string | Тип курсу (“TEXT”, “VIDEO”) |
| creationDate | timestamp | Дата створення курсу |
| language | string | Мова курсу |

| | | |
|---|---|---|
| lessonCount | number | Кількість уроків у курсі |
| poster | string | URL постера курсу |
| published | boolean | Чи опублікований курс |
| totalDuration | number | Загальна тривалість курсу (у хвилинах) |

4) Коли студент реєструється в якомусь курсі, ми повинні відстежувати це, тому що це важливо для того, щоб відстежувати чи людина зареєстрована на курсі чи ні, і також змінювати статус, тобто якщо людина покине курс, то змінюється його активність, але вона зможе повернутися пізніше і ми збережемо її прогрес (Таблиця 2.4 Опис типу даних StudentCourse).

Таблиця 2.4 Опис типу даних StudentCourse

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| student_courseId | string | ID, яке поєднує ID студента та курсу |
| active | boolean | Чи зарахований студент на курс зараз зараз(true/false) |
| courseId | string | ID курсу |
| courseType | string | Тип курсу (“TEXT”, “VIDEO”) |
| enrollmentDate | timestamp | Дата зарахування на курс |
| studentId | string | ID студента |

5) Ми повинні якимось чином відстежувати прогрес кожного студента по курсу, враховувати уроки, які він закінчив, тому для цього ми створимо структуру StudentCourseProgress, адже це дозволить розділити безпосередньо відстеження курсів студента у StudentCourse та взаємодію та прогрес студента у курсі у StudentCourseProgress (Таблиця 2.5 Опис типу даних StudentCourseProgress).

Таблиця 2.5 Опис типу даних StudentCourseProgress

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| student_courseId | string | ID, яке поєднує ID студента та курсу |

| | | |
|---|---|---|
| completedLessons | array of strings | Список ID завершених уроків |
| courseId | string | ID курсу |
| courseType | string | Тип курсу (“TEXT”, “VIDEO”) |
| lastCompletedLesson | number | Порядковий номер останнього завершеного уроку |
| progressDate | timestamp | Дата відстеження прогресу |
| studentId | string | ID студента |

6) Далі ми мусимо мати структуру, яка буде зберігати уроки і при цьому мусить бути гнучкою, в тому плані, що дозволить зберігати текстові та відео уроки в одній структурі, і лише матиме певну різницю в полях. Тобто фактично ми маємо не дві окремі структури, які презентують відео та текстові уроки, а одну структуру, яка є гнучкою і дозволяє це зробити. Далі ми розберемося це коли використовуватимо певну базу даних (Таблиця 2.6 Опис типу даних Lesson).

Таблиця 2.6 Опис типу даних Lesson

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| lessonId | string | ID уроку |
| content \| videoUrl | string | Вміст уроку або посилання на відео, в залежності від типу уроку |
| creationDate | timestamp | Дата створення уроку |
| description | string | Опис уроку |
| duration | number | Тривалість уроку в секундах |
| lessonOrder | number | Порядковий номер уроку в курсі |
| title | string | Назва уроку |

7) Для того, щоб реалізувати ключову особливість нашого додатку – щоденний навчальний план – це коли студент бачить лише ті уроки, які він повинен пройти сьогодні, ми повинні створити структуру, яка дозволити зберігати створені навчальні плани (Таблиця 2.7 Опис типу даних DailyLearningPlan).

Таблиця 2.7 Опис типу даних DailyLearningPlan

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| dailyLearningPlanId | string | ID навчального плану |
| completedLessons | array of strings | Список ID завершених уроків |
| date | string | Дата навчального плану |
| lessons | array | Список уроків на день |
| progress | number | Кількість завершених уроків на день |
| studentId | string | ID студента |
| totalDuration | number | Загальна тривалість навчання на день (у секундах) |

8) Для того, щоб людина могла перевірити свої знання після кожного уроки ми повинні зробити структуру, в якій ми зможемо зберігати різні вправи, ця структура матиме назву Task і всередині неї зможуть зберігатися квізи, завдання на встановлення відповідностей, правда чи неправда і тд. Для початку ми можемо навести базову структура на приклад QUIZ завдання (Таблиця 2.8 Опис типу даних Task).

Таблиця 2.8 Опис типу даних Task

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| taskId | string | ID завдання |
| answer | string | Правильна відповідь на завдання |
| options | array | Варіанти відповідей |
| question | string | Поставлене питання |
| type | string | Тип завдання |

## 2.2 Функціональна структура додатку

Тепер, коли ми маємо розуміння, які типи даних ми матимемо і, яким чином вони будуть репрезентовані, ми можемо розписати базові user flow, тобто, як відбувається взаємодія користувача із додатком, які функції і як повинні бути доступні і яким чином користувач повинен взаємодіяти із програмою, як відбуватимуться зміни екранів і тд.

**Реєстрація та Онбординг**

Почнемо із головного – це реєстрація користувача (Рисунок 2.2 Процес реєстрації та онбордингу). Перш за все це має відбутися із використанням:

- Email
- Username
- Password

Якщо реєстрація відбулася успішно, то відбувається перенесення на наступний екран – екран онбордингу.

Онбординг (onboarding) - це процес ознайомлення нових користувачів із системою чи продуктом з метою полегшити їх інтеграцію та забезпечити ефективне використання. Це своєрідний вступний тур, який допомагає користувачам зрозуміти основні функції, навігацію та можливості продукту.

Якісний онбординг дозволяє скоротити криву навчання, підвищити задоволеність користувачів та зменшити відтік на ранніх етапах використання. Він передбачає чітку візуальну комунікацію, простий та інтуїтивний інтерфейс, а також поступове залучення користувачів до основних функцій продукту. [6]

Перша частина онбордингу – це екран, на якому користувач мусить використовуючи графічний інтерфейс обрати скільки хвилин на день він хоче приділяти навчанню – від 5 до 30 хвилин.

Ми обираємо такий час, враховуючи наш досвід використання освітніх сервісів, так і інших додатків із великою кількістю контенту, тому саме обмеження максимального часу до 30 хвилин дозволить збільшити кількість завершених навчальних днів, так і створити певну інтригу, коли користувач не зможе отримати більше і тому з більшою ймовірністю повернется завтра.

Далі після того, як було обрано час, ми переходимо до другої частини онбордингу – користувач мусить обрати декілька інтересів із списку заданих («історія», «економіка», «психологія», «програмування» і тд.), це використовується задля того, щоб боротися із проблемою «холодного старту» - коли користувач вперше користується якимось додатком, то ми не знаємо нічого про нього, а отже не можемо нічого порекомендувати, в нашому випадку, якщо кількість курсів значно збільшиться або ми змінимо дещо конфігурацію, нам дуже важливо знати дещо про цього користувача, для того, щоб рекомендувати те, що ми вважаємо йому буде цікаво побачити.

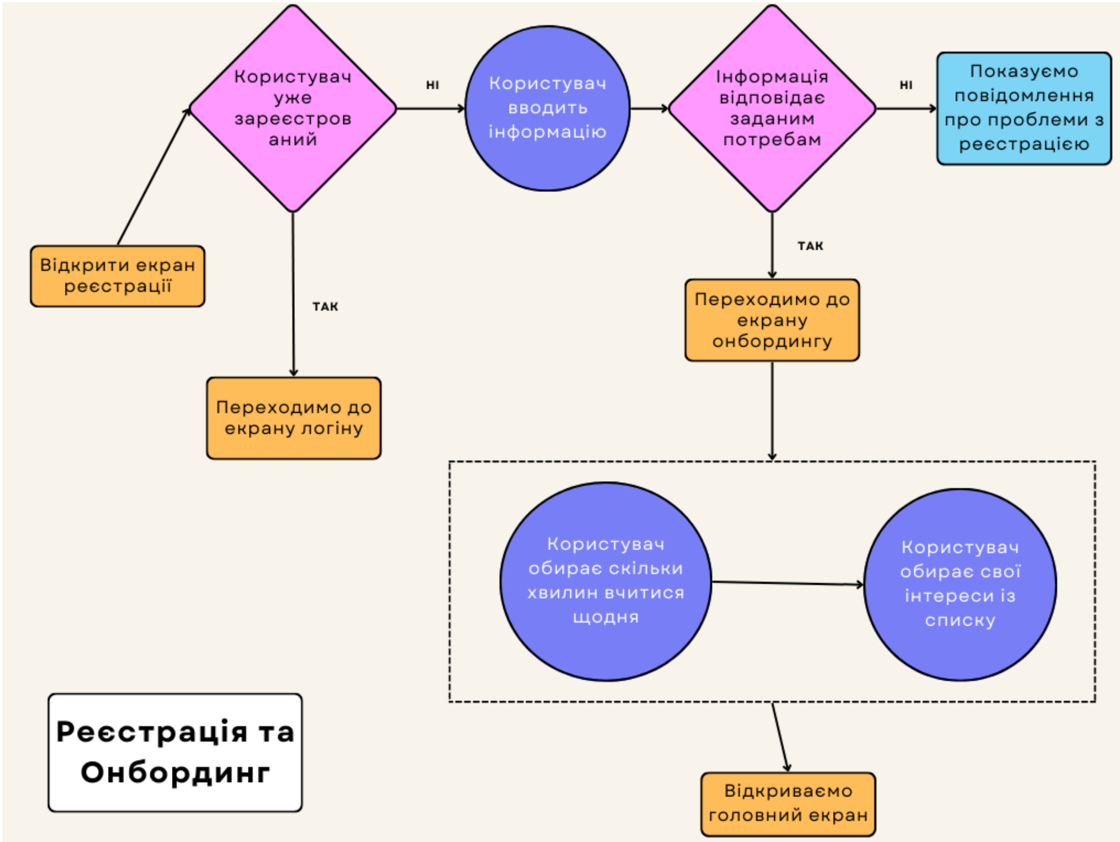


Рисунок 2.2 Процес реєстрації та онбордингу

**Меню та Навігація**

Коли, ми уже зареєстровані, ми повинні у додатку мати можливість переміщатися між декількома різними екранами, такими як «Галерея Курсів», «Домашній екран» чи «Екран профілю користувача», найкращим та найбільш визнаним способом зробити це є меню в нижній частині екрані з іконками різних екранів. [7]

Тому важливо описати як відбуватиметься ця навігація і які екрани повинні бути включені в меню.

Нижче ми опишемо екрани і, яка частина функціоналу повинна відбуватися на кожному з екранів (Рисунок 2.3 Процес навігації між різними екранами в меню):

- **Home** – основний екран, в якому демонструється навчальний план на сьогодні, кількість уроків, пройдених за весь час та запускається сам процес проходження різних уроків і відстеження прогресу.
- **Gallery** – екран, в якому ми демонструємо список курсів і при натисненні на певний курс відкривається картка курсу, в якій показано назву, кількість

уроків і надано перелік із уроків із назвою та приблизним часом необхідним для проходження.

- **Profile** – екран, на якому буде інформація, необхідна користувачу про курси, на які він зареєстрований і також демонстуватиме інтереси користувача та бажаний денний час, який хоче приділяти на навчання, і також зможе змінити цей час.

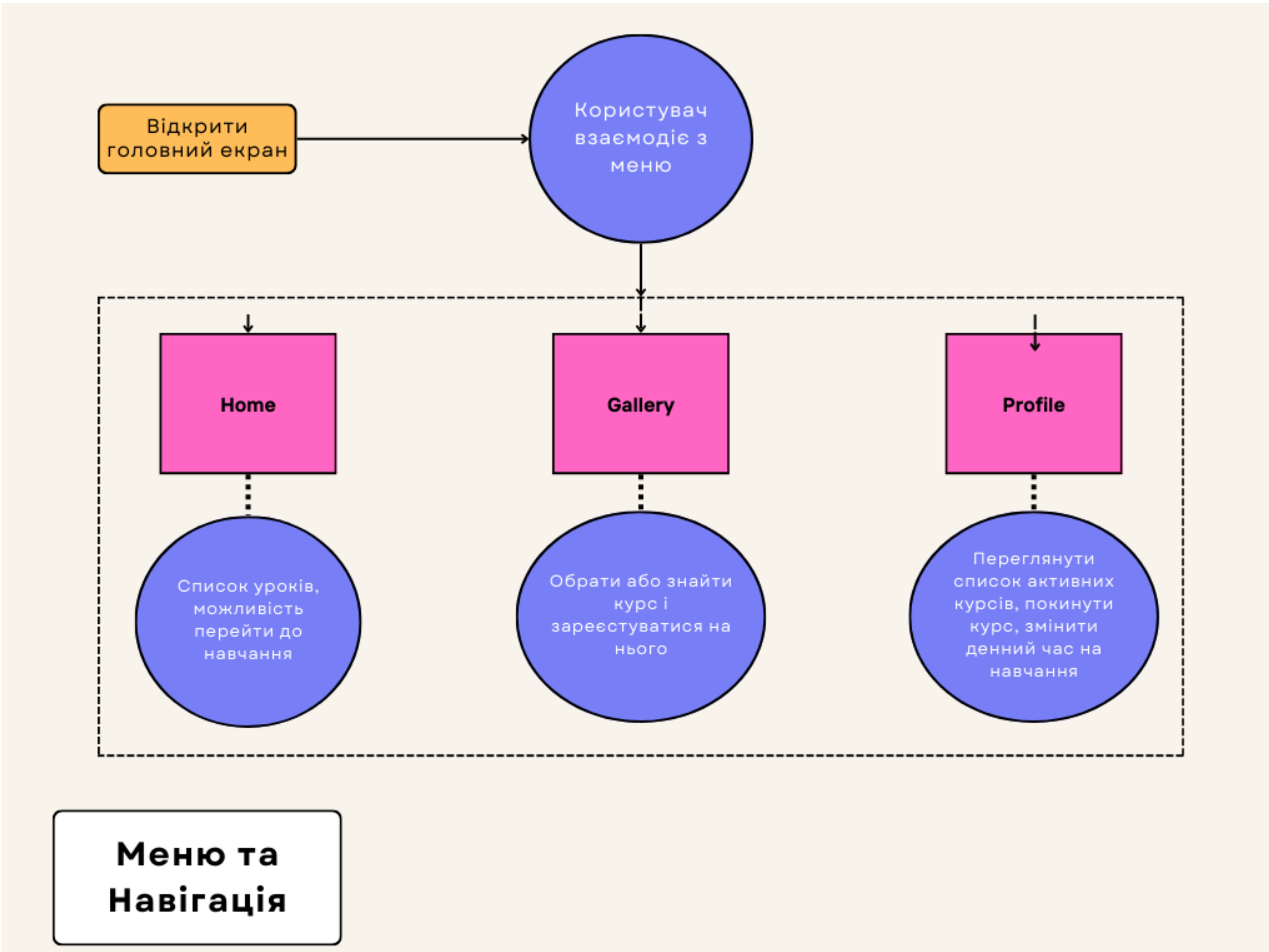


Рисунок 2.3 Процес навігації між різними екранами в меню

## Логін

Без процесу аутентифікації неможливо отримати доступ до будь-якої частини нашої системи. В нашому випадку це буде надзвичайно простий методі із використанням емейлу та паролю ().

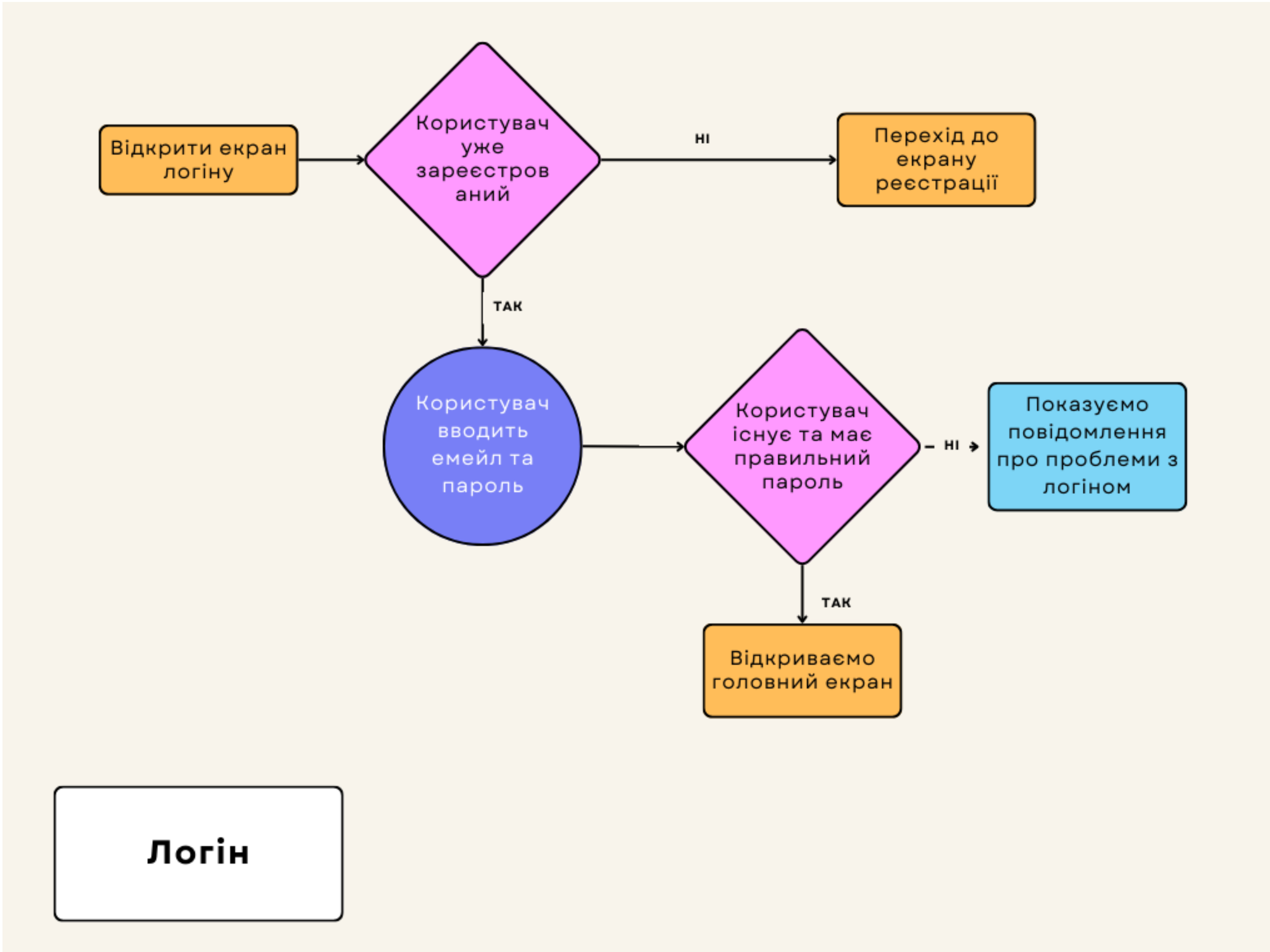


Рисунок 2.4 Процес логіну

**Процес виконання уроку**

Ключова частина нашого додатку на яку мусить припадати 99% часу, проведено в ньому. Ця частина складається із споживання контенту уроку, задавання запитань та отримання відповідей у форматі чат-боту, а також проходження тестів чи інших завданнь після кожного уроку для підтвердження якості засвоєної інформації (Рисунок 2.5 Процес виконання уроку).

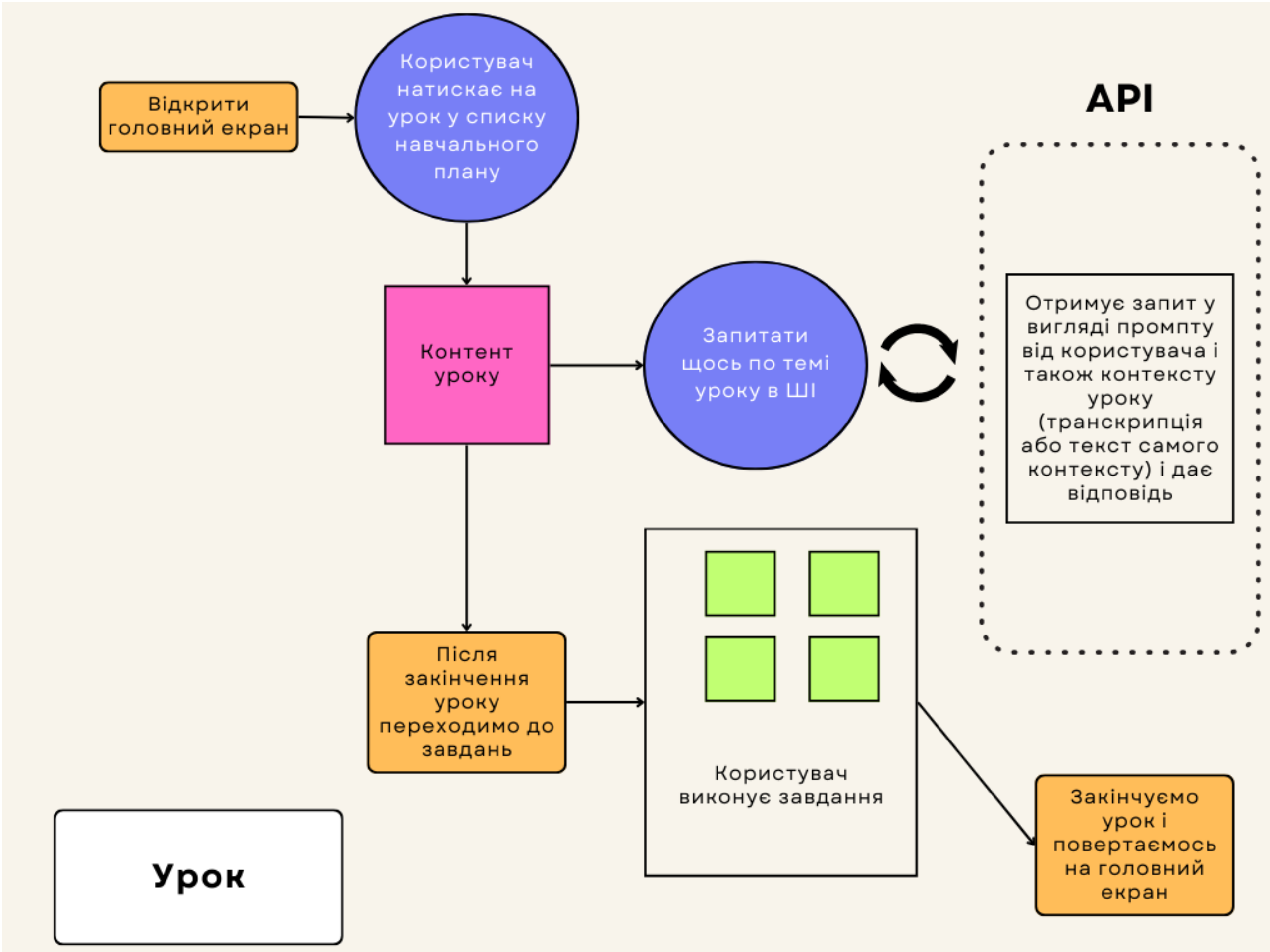


Рисунок 2.5 Процес виконання уроку

## 2.3 Вибір технологічних засобів для реалізації задачі

Маючи необхідні описи структури даних, а також докладний опис процесів всередині додатку, ми маємо конкретні задачі, які повинні бути імплементовані програмно, а для цього нам треба обрати правильний набір технологій, які будуть виконувати ті типи задач найкраще і при цьому будуть максимально зручними в використанні, спростять задачу розробки.

Отже, давайте зробимо огляд певних можливостей, які ми можемо використати і що саме ми використаємо.

### Платформа + Мова Програмування

При розробці Android-додатку найважливішим рішенням є вибір мови програмування. Основними варіантами для Android є:

**Java:** Як стандартна мова програмування для розробки Android-додатків протягом багатьох років, Java має широку підтримку та велику кількість ресурсів для навчання. Проте, в порівнянні з Kotlin, Java може мати більш високий рівень складності та меншу продуктивність. [8,10]

**Kotlin:** Kotlin став все більш популярним в останні роки як альтернатива Java для розробки Android-додатків. Він пропонує більш сучасний та компактний синтаксис, що дозволяє розробникам писати менше коду, при цьому забезпечуючи більшу безпеку та продуктивність. Kotlin також має низьку поріг входження для розробників, які вже знайомі з Java. [8,9,10]

**React Native:** Це кросплатформенний фреймворк, який дозволяє розробникам використовувати JavaScript для створення мобільних додатків для Android та iOS. React Native має велику спільноту розробників та підтримується компанією Facebook. Він дозволяє збільшити швидкість розробки та забезпечити одноразовий код для обох платформ. [11]

**Flutter:** Розроблений компанією Google, Flutter є іншим кросплатформним фреймворком, який використовується для створення мобільних додатків для Android та iOS. Він використовує мову програмування Dart та надає швидку розробку, високу продуктивність та можливості створення красивого та анімованого інтерфейсу. [9,11]

Таблиця 2.9 Порівняння Kotlin vs Java vs React Native vs Flutter

| Критерій | **Kotlin** | **Java** | **React Native** | **Flutter** |
|---|---|---|---|---|
| **Мова програмування** | Kotlin | Java | JavaScript | Dart |
| **Тип розробки** | Нативна (Android) | Нативна (Android) | Крос-платформна | Крос-платформна |
| **Швидкість розробки** | Швидка | Помірна | Швидка | Швидка |
| **Продуктивність** | Висока | Висока | Залежить від складності | Висока |
| **Спільнота та підтримка** | Широка | Дуже широка | Широка | Широка |
| **UI компоненти** | Багатий набір | Багатий набір | Нативні компоненти | Власні віджети |

| | | | | |
|---|---|---|---|---|
| **Сумісність з існуючим кодом** | Сумісний з Java | - | Сумісний з NPM пакетами | Сумісний з C/C++ бібліотеками |
| **Кросплатформенність** | Немає | Немає | Так | Так |
| **Тестування та відлагодження** | Інтегровані інструменти | Інтегровані інструменти | Через сторонні інструменти | Інтегровані інструменти |

Отже, всі вони виконують свою функцію, але оскільки я маю досвід роботи з Java і той факт, що компанія, яка випускає Android – Google нещодавно заявила, що вона розглядає мову Kotlin як основну при розробці і також враховуючи новини, що компанія Meta переписує більшість своїх сервісів із Java на Kotlin, незважаючи навіть на те, що вона є дещо повільнішою, але на її користь грають наступні чинники [12] :

– Більш лаконічний та виразний синтаксис, що сприяє підвищенню продуктивності розробки та читабельності коду.
– Акцент на безпеку та запобігання типовим помилкам, таким як NullPointerException.
– Покращена підтримка функціонального програмування та сучасних парадигм розробки.

А враховуючи, що потужності смартфонів досягли вражаючих результатів, то незначне падіння швидкодії не відчувається.

Отже, ми будемо використовувати Kotlin і набір функцій для неї при розробці нашого додатку.

**База Даних**

Як кажуть дані – це все, тому їх треба берегти та якісно зберігати. Саме тому без бази даних ніяк не обійтись, якщо ми хочемо побачити щось більше за “Hello world”.

Що ж ми порівняємо 4 найпоширеніші способи зберігати дані для мобільних додатків:

**PostgreSQL** - це потужна об'єктно-реляційна система управління базами даних з відкритим вихідним кодом. Вона забезпечує надійність, масштабованість та відповідність стандартам SQL. PostgreSQL часто використовується для великих веб-додатків та корпоративних систем. Однак налаштування та керування власним сервером PostgreSQL може бути складним для мобільних додатків. [13]

**Supabase** - це альтернативна реалізація відкритим кодом, що поєднує PostgreSQL як основну базу даних із хмарною платформою, аналогічною Firebase. Supabase пропонує зручний доступ до бази даних через веб-сокети, автентифікацію користувачів, хмарну функціональність та керування даними. Проте Supabase є відносно новим рішенням, і його екосистема може бути менш розвиненою, ніж у Firebase. [14,15]

**Firebase Firestore** - це хмарна NoSQL база даних, розроблена Google для різного виду додатків.Вона забезпечує синхронізацію даних через всі підключені клієнти, що робить її ідеальною для мобільних та веб-додатків. Firebase також пропонує безліч додаткових функцій, таких як автентифікація, хмарні функції, аналітика та хмарне зберігання файлів, що спрощує розробку. [14]

**MongoDB** - це популярна нереляційна база даних, що використовує документно-орієнтовану модель зберігання даних. Вона пропонує гнучку схему, горизонтальну масштабованість та високу продуктивність для операцій зчитування та запису. MongoDB часто використовується у веб-додатках та мобільних рішеннях, де потрібно працювати з неструктурованими або напівструктурованими даними. [15,16]

Таблиця 2.10 Порівняння ключових характеристик баз даних

| **Критерії** | **PostgreSQL** | **Supabase** | **Firebase Firestore** | **MongoDB** |
|---|---|---|---|---|
| Тип бази даних | Реляційна | Реляційна (PostgreSQL) | NoSQL | NoSQL (документо-орієнтована) |
| Доступ у реальному часі | Ні | Так | Так | Ні |
| Хмарне розміщення | Ні | Так | Так | Так |
| Автентифікація користувачів | Ні | Так | Так | Так |
| Додаткові функції | Ні | Частково | Так | Частково |
| Зрілість та підтримка | Висока | Низька | Висока | Висока |

У даному випадку ми знову ж таки будемо виходити із власного досвіду, який був набутий перед цією роботою. Оскільки бази даних навіть із усіма полегшеннями і мінімальною складністю входу, все ж таки лишаються досить специфічними структурами і вимагають час для освоєння.

Я працював із усіма базами даних наведеними в таблиці (Таблиця 2.10 Порівняння ключових характеристик баз даних), то з досвіду важливо враховувати 3 ключові речі:

- Чи є вона безкоштовною до певної міри зростання кількості записів в ній.
- Скільки часу займає підключити її та запустити, тобто підтвердити факт існування
- Чи є продукти, які йдуть в комплекті з базою даних, наприклад сховище для медіа-файлів чи система контролю аутентифікації

За цими метриками ніхто не може конкурувати із Firebase Firestore і тому наш вибір падає на неї. Крім того, сама структура, яка зберігає всю інформацію у вигляді документів, набагато краще відповідає нашим потребам, оскільки наші дані вони є однаковими, тобто ми маємо різні типи уроків (текстові та відео), хочемо мати різні типи вправ (quiz, true or false, put in order).

Тому, за даними оцінками саме Firebase Firestore є найкращим відображенням наших потреб:

- Вбудоване сховище дозволить зберігати певні матеріали, як зображення або відео-контент
- Система контролю за аутентифікацією дозволить керувати станом користувача у будь-який момент часу
- Наявність Realtime Database, вбудованої в Firebase дозволить використовувати її для ШІ-чату, щоб користувач міг задавити питання по уроку. А оскільки для того, щоб тримати контекст уроку і надати великій мовній моделі інформацію, яка вона зможе використовувати при генерації відповіді, то ми мусимо десь це зберігати і бд, яка працює в реальному часі дуже допоможе в даному випадку

**Велика мовна модель для чату**

Розглядаючи варіанти для інтеграції чат-функціональності на базі штучного інтелекту в наш додаток, ми звернули увагу на дві моделі від OpenAI - GPT-4 та GPT-3.5 turbo.

GPT-4 є найновішою і найпотужнішою великою мовною моделлю від OpenAI на даний момент. Вона демонструє видатні результати в різноманітних бенчмарках з обробки природної мови, логічного міркування та програмування. GPT-4 має близько 1,76 трильйонів параметрів та величезне вікно контексту 128 000 токенів. Однак така продуктивність має свою ціну - $10 за 1 мільйон вхідних токенів та $30 за 1 мільйон вихідних[17,18].

З іншого боку, GPT-3.5 turbo є оптимізованою версією попередньої моделі GPT-3.5 від OpenAI. Вона налічує 175 мільярдів параметрів і має вікно контексту до 4096 токенів. Незважаючи на менші розміри та потужність у порівнянні з GPT-4, GPT-3.5 turbo забезпечує прийнятну якість обробки природної мови для багатьох завдань за набагато нижчу ціну -  $0.5 за 1 мільйон вхідних токенів та $1.5 за 1 мільйон вихідних[17,18].

З огляду на доступний бюджет та функціональні вимоги нашого додатку, ми вирішили зупинитись на використанні GPT-3.5 turbo для забезпечення чат-функціоналу. Ця модель забезпечить прийнятний рівень якості за доступною ціною, що є достатнім для нашої цілі надати користувачам можливість ставити запитання та отримувати відповіді в межах уроків. У майбутньому, за наявності додаткових ресурсів, ми можемо розглянути перехід на новітню GPT-4 для підвищення якості.

# 3. ДИЗАЙН ТА ПРОГРАМНА РЕАЛІЗАЦІЯ

## 3.1 Дизайн Інтерфейсу Додатку

Ми маємо всі необхідні описи функціональних особливостей нашої інтелектуальної системи-додатку Maverkick, тому саме час підійти до візульної репрезентації наших ідей та бачень.

Ми використаємо сервіс Figma для того, щоб створити інтерфейси для прототипу нашого додатку.

Це безкоштовний і дуже легкий у використанні онлайн-сервіс, де можна створювати прототипи для різного типу додатків та програм, для різних геометричних розширень. Ми робимо для звичайного розширення смартфона.

За основу нашого дизайну ми візьмемо палетку, яку нам допоміг згенерувати сервіс Coolors – надзвичайно корисний для того, щоб знайти ті кольори, які віддзеркалюють ідею та дух [19].

Ось який вигляд мають 5 кольорів, які складатимуть основу нашого дизайну:

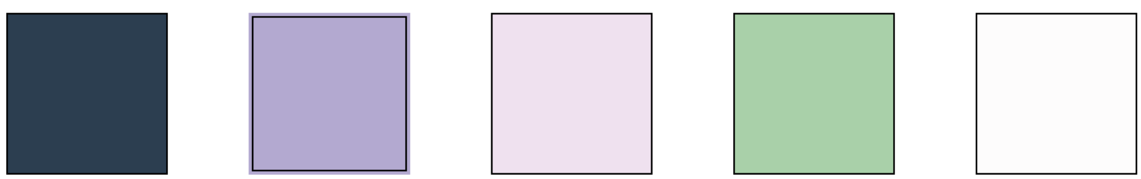

Рисунок 3.1 Кольорова палета додатку

Тепер час створити компоненти, які будуть використовувати ці кольори та функціонал, який ми описали в розділі 2.2.

**Реєстрація та Логін**

Розпочнемо дизайн додатку із створення двох екранів:

- SignUp, де користувач повинен ввести email, username та password і після цього буде зареєстрований. Також на цьому екрані має бути можливість перейти на Login екран у випадку, якщо користувач зареєстрований
- Login, де користувач повинен ввести email та password. Також на цьому екрані має бути можливість перейти на SingUp екран у випадку, якщо користувач ще не зареєстрований

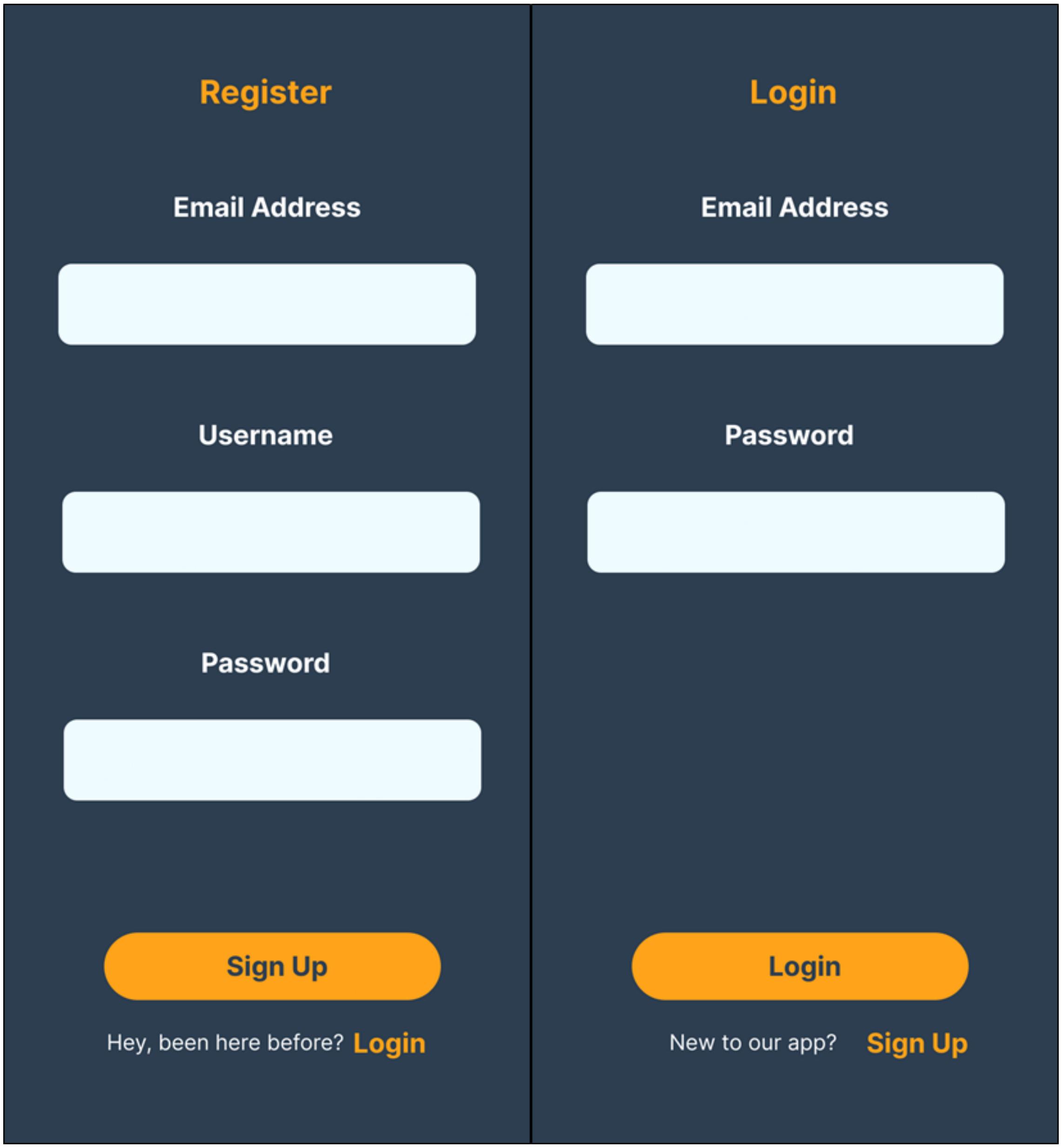

Рисунок 3.2 Дизайн Реєстрації та Логіну

**Home (Головний екран)**

Створимо дизайн для головного екрану, на якому ми будемо демонструвати уроки, які складено для користувача на сьогодні.

Ми маємо різні уроки у вигляді різних геометричних форм:

- Круг для відеоурока
- Заокруглений квадрат для текстового урока

Також ми демонструємо кількість уроків, які завершено у верхній частині екрану у вигляді «цеглинок», в нижній частині екрану ми маємо кнопку при натисканні на яку відкриється наступний урок на сьогодні.

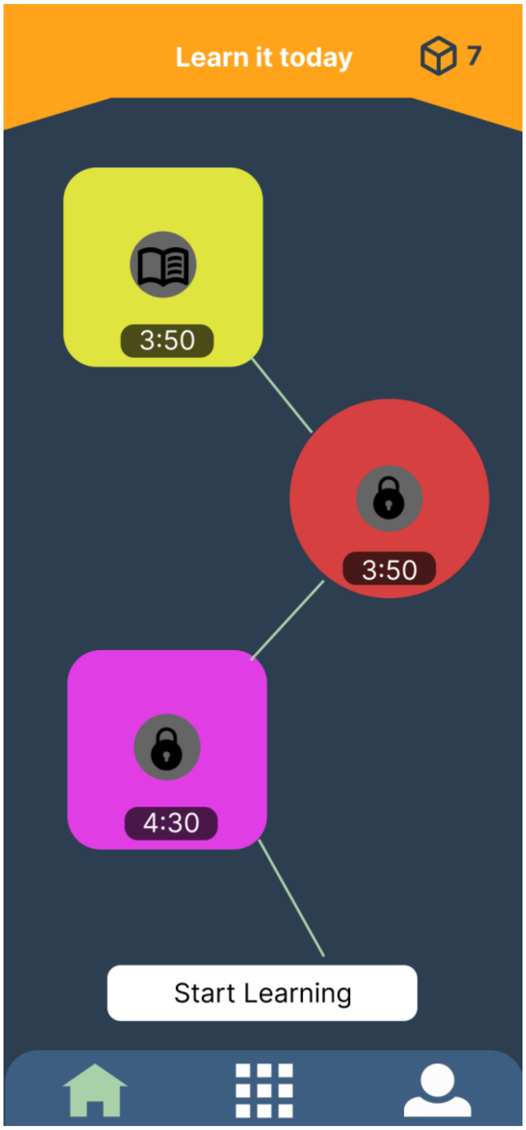


Рисунок 3.3 Дизайн Головного Екрану

**Lesson (Екран Уроку)**

Далі ми створюємо безпосередньо екрани для уроку: відео та текстового. Це повинно бути максимально просто і не відволікати користувача, оскільки ми хочемо сконцентрувати його увагу на процесі навчання. Ми створюємо два різних стилі для відео та текстових уроків:

- Для відео більш темний, чимось нагадує TikTok та YouTube
- Для тексту світліший, більш подібний до Medium та Google Books


Рисунок 3.4 Дизайн Екрану Уроків

### Чат-бот та Завдання

Під час проходження уроки ми можемо мати запитання і для цього потрібен чат-бот «El Bricko».

Також ми після кожного уроку демонструємо завдання, поки що лише тести, але можливе розширення для інтеграції і інших завдань.

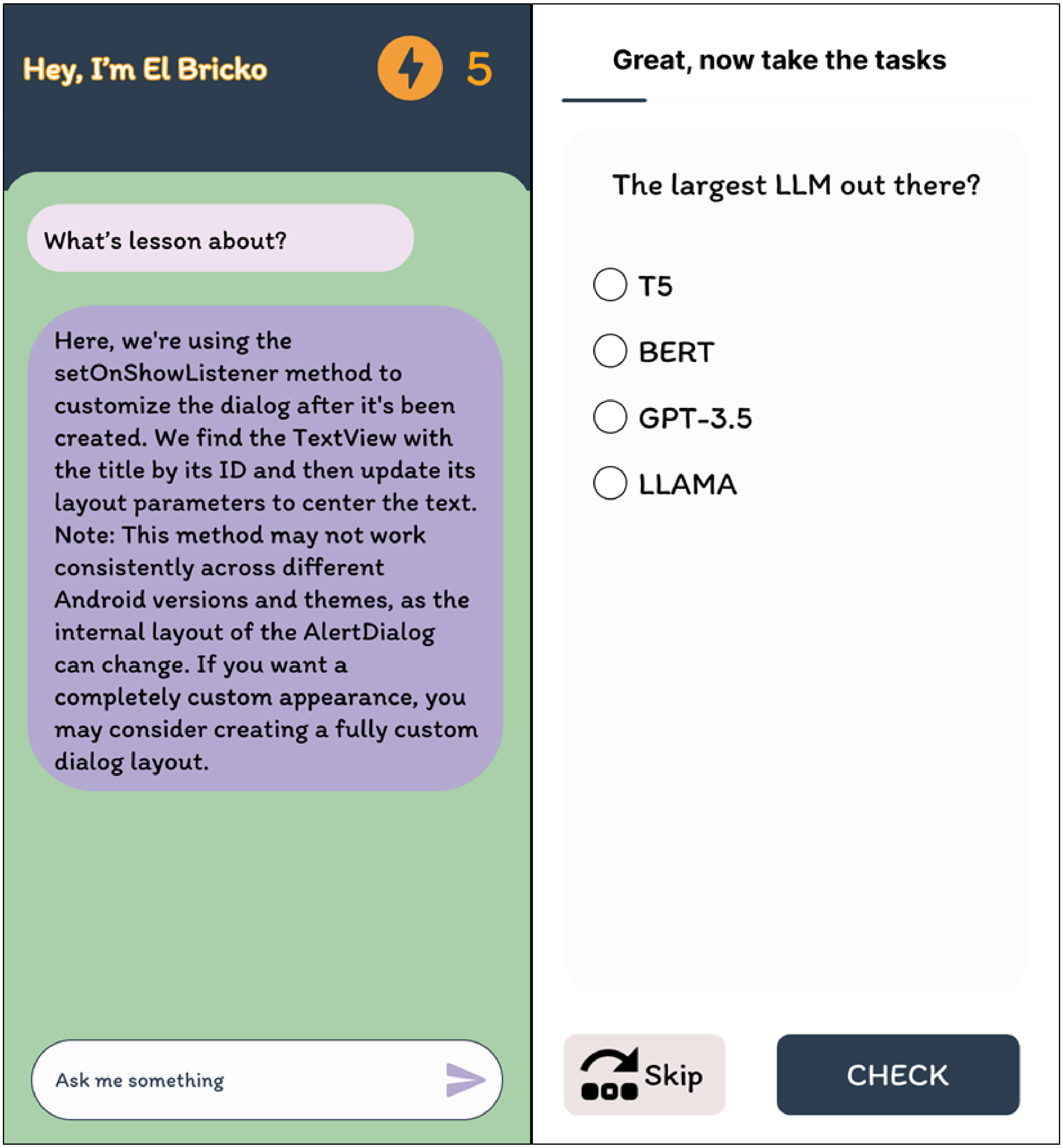


Рисунок 3.5 Дизайн Чат-боту та Екрану Завдань

### Gallery та Course (Екрани Галереї та Курсу)

В Галереї ми демонструємо список курсів, при натисненні на картку курсу відкривається екран курсу, в якому є певний опис про цей курс

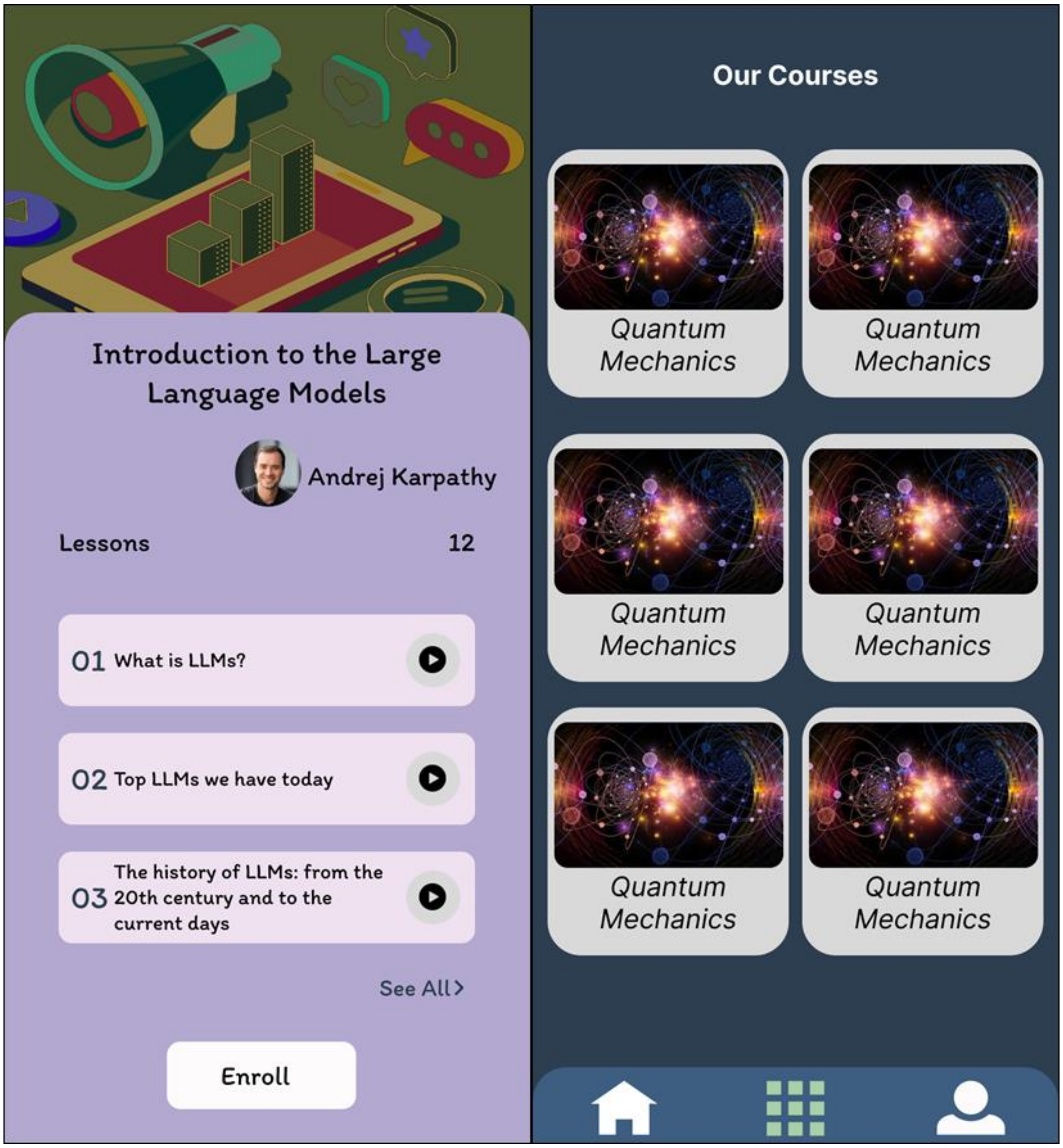


Рисунок 3.6 Дизайн Галереї та Курсу

**Profile (Профіль Користувача)**

На цьому екрані ми зробимо поділ на дві частини, які можна буде змінювати, використовуючи слайдер:

- Courses – де ми будемо показувати список курсів, які активні для користувача і де він зможе покинути курс
- Settings – де показано інформація про користувача, така як інтереси та час, який можна змінити при натисненні на значення

Крім того ми маємо спільний інтерефейс у вигляді username та зображення профілю користувача та іконки у правій верхній частині, які дозволить вийти з аккаунта.

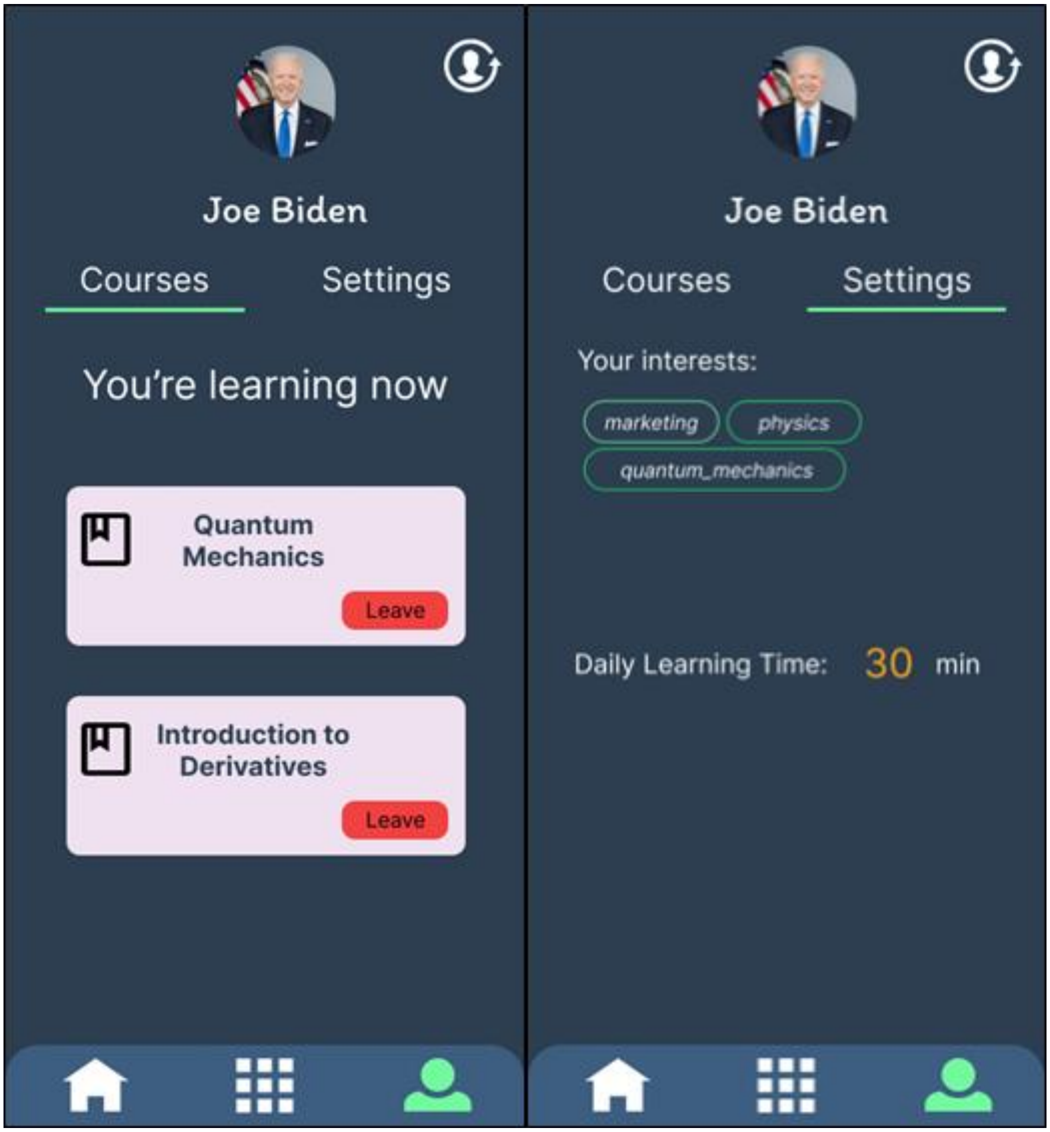


Рисунок 3.7 Дизайн Екрану Профілю

## 3.2 Програмна реалізація

### *3.2.1 Інструменти*

Перед тим як перейти до програмування і нарешті перетворити шаблони на реальний додаток, який можна буде запустить на власному смартфоні, ми спочатку визначимо набір програмних інструментів та девайсів, використовуючи, які ми й будемо програмувати наше рішення для інтелектуальної освітньої системи Maverkick (Рисунок 3.8 Інструменти для розробки додатку):

1) **Android Studio** - Інтегроване середовище розробки (IDE), яке ми використовуватимемо для створення додатків на Android. Це потужний інструмент, який забезпечує зручний інтерфейс для розробки, налагодження та тестування додатків.
2) **ChatGPT** - Потужна мовна модель з відкритим вихідним кодом, яку ми використовуватимемо для генерування тексту, допомоги у вирішенні проблем та відповідей на запитання. Ми зможемо використовувати ChatGPT для генерування коду, пошуку рішень проблем та прискорення процесу розробки.
3) **Samsung Galaxy M32** - Смартфон, на якому ми плануємо тестувати та запускати наш додаток під час розробки. Використання реального девайсу дозволить нам перевірити роботу додатку в реальних умовах та переконатися, що він працює коректно на відповідному обладнанні.
4) **Firebase Console** - Веб-консоль Google Firebase, яку ми використовуватимемо як набір інструментів для розробки, тестування та моніторингу нашого додатку. У Firebase Console ми зможемо відстежувати процес розробки, керувати базами даних, хмарними функціями та іншими ресурсами, необхідними для нашого додатку.


Рисунок 3.8 Інструменти для розробки додатку

Оскільки ми розробляємо хоча й базову MVP версію нашого продукту, та все ж за рахунок того, що це Android додаток, то кількість коду, яка буде написана є дуже великою і повний огляд всієї цієї бази поза межами цієї роботи, тому ми сконцентруємося на розборі найголовніших пунктів, в яких ми будемо наводити, яким чином це було запрограмовано і показуватимемо відповідні шматки коду.

### *3.2.2 Архітектура Додатку*

Але спочатку розпишемо архітектурну структуру нашого додатку:
Архітектурна структура нашого додатку базується на парадигмі MVVM (Model-View-ViewModel) [20]. Дозвольте розкрити основні складові цієї архітектури та їх взаємодію.

**Модель (Model):**

- Представляє дані та бізнес-логіку додатку.

- Може включати класи даних, бази даних, мережеві запити та інші компоненти, що відповідають за обробку та збереження даних.

**Вид (View):**

- Представляє графічні компоненти та розміщення UI елементів додатку.
- У нашому випадку, використовуються XML-файли для опису макетів екранів.

**ViewModel:**

- Діє як посередник між Моделлю та Видом.
- Відповідає за обробку даних, які отримує від Моделі, та підготовку їх для відображення відповідними UI елементами.
- Забезпечує можливість зв'язування даних (data binding) між Видом і Моделлю.

Структура нашого додатку включає декілька ключових елементів:

- **Activities та Fragments:** Вони відповідають за відображення різних екранів додатку. У них ініціалізується ViewModel та встановлюється зв'язок з відповідними XML-макетами.
- **RecyclerViews:** Використовуються для відображення списків даних. Вони взаємодіють з ViewModel для отримання необхідної інформації та оновлення UI.
- **Constraints та Guidelines:** Використовуються для позиціонування UI елементів на екрані. Це допомагає створювати адаптивний та зручний інтерфейс користувача.
- **Стилі та Теми**: Визначаються для забезпечення єдиної зовнішнього вигляду додатку.

Ця архітектура допомагає розділити логіку додатку на окремі компоненти, що робить код більш модульним, підтримуваним та легким для тестування.

### *3.2.3 Модулі Додатку*

Крім підходів до типу архітектури, також дуже важливо правильно структуризувати компоненти та складові у невеликі модулі. Цей підхід дозволяє розбити функціонал додатку на менші, самостійні компоненти, кожен з яких виконує певну частину функціоналу [21].

Основні переваги такого підходу включають:

- **Швидше тестування та виявлення помилок:** Модульний підхід дозволяє проводити тестування окремих модулів незалежно від інших частин програми. Це полегшує виявлення та виправлення помилок, оскільки вони зазвичай локалізуються в межах конкретного модулю.
- **Інкрементальний розвиток:** Розробка модуль за модулем дозволяє ефективно вдосконалювати додаток, додаючи нові функції або вдосконалюючи існуючі, не втручаючись у роботу інших компонентів.
- **Краща організація коду та ресурсів:** Розбиття додатку на модулі допомагає зберігати код організованим та легко зрозумілим. Кожен модуль має чітко визначену відповідальність та межі.
- **Спрощення командної роботи:** Кожен модуль може розроблятися та тестуватися окремо, що полегшує командну роботу і дозволяє розробникам працювати над різними частинами додатку паралельно.

Перелік модулів у нашій системі наведено в Таблиця 3.1 Модулі додатку

Таблиця 3.1 Модулі додатку

| **Назва модулю** | **Опис** |
|---|---|
| app | Відповідає за ініціалізацію та керування всіма іншими, а також за реалізацію логіки для відображення та управління користувацьким інтерфейсом. |
| video-lesson | Забезпечує функціонал для відображення відеоуроків та матеріалів у відеоформаті. |
| text-lesson | Відповідає за відображення текстових уроків та матеріалів. |
| auth | Забезпечує реєстрацію та логін |
| chat-helper | Надає функціональність для чату та спілкування користувачів. |
| data | Відповідає за взаємодію з сховищами даних та методи для отримання, збереження та оновлення даних. |
| common | Містить загальні ресурси, такі як стилі, кольори, рядки та інші, призначені для використання у всіх інших модулях. |
| profile | Забезпечує управління профілем користувача та його персоналізацію. |
| shared-ui | Містить загальні компоненти користувацького інтерфейсу, які можуть бути використані у різних частинах додатку. |
| student | Відповідає за функціональність, пов'язану зі студентами, курсами та учбовим процесом. |
| tasks | Надає функціонал для виконання завдань та тестів. |

### *3.2.4 Моделі Даних*

Фактично головним компонентом і фундацією будь-якого додатку є моделі – структури, які презентують різні типи даних, які зберігаються в базі даних і створюють певний рівень абстракції поверх цих даних для пришвидшення роботи і інкапсуляції даних всередині структури з можливістю ефективного використання в операціях.

Таблиця 3.2 Список моделей даних в додатку

| **Назва моделі** | **Опис** | **Параметри** |
|---|---|---|
| AiChatMessage | Представляє повідомлення в чаті з штучним інтелектом. | `text`: String, `isUser`: Boolean |
| User | Клас, що зберігає дані користувача. | **`userId`**: String, **`username`**: String, **`email`**: String, **`profilePicture`**: String? |
| CourseStatistics | Дані для зберігання статистики конкретного курсу. | `courseId`: String, `courseName`: String, `numberOfEnrollments`: Int, `numberOfCompletions`: Int, `numberOfDropouts`: Int, `totalNumberOfRatings`: Int, `sumOfRatings`: Int |
| DailyLearningPlan | Клас, що зберігає список уроків, які студент повинен вивчити протягом дня. | `studentId`: String, `date`: String?, `lessons`: List<Lesson>, `totalDuration`: Int, `progress`: Int, `status`: DailyLearningPlanStatus, `completedLessons`: List<String> |
| Student | Клас, що зберігає дані про студента. | `studentId`: String, `age`: Int, `dailyStudyTimeMinutes`: Int, `interests`: List<String>, `bricksCollected`: Int, `enrolledCourses`: List<String>, `enrolledGeneratedCourses`: List<String>, `generatedTextCourses`: List<String>, `courseGenerationTries`: Int, `finishedCourses`: List<String> |
| Course | Абстрактний клас, що представляє курс. | `courseId`: String, `courseName`: String, `language`: String, `numberLessons`: Int, `creationDate`: Date, `type`: CourseType, `authorId`: String |
| TextCourse | Клас, що представляє текстовий курс. | **`courseId`**: String, **`courseName`**: String, **`language`**: String, **`poster`**: String?, **`numberLessons`**: Int, **`tags`**: List<String>, **`creationDate`**: Date, **`authorId`**: String, **`published`**: Boolean |

| | | |
|---|---|---|
| FirebaseTextCourse | Клас, який використовується для взаємодії з Firebase з `courseId`. | `courseName`: String, `language`: String, `poster`: String?, `lessonCount`: Int, `tags`: List<String>, `creationDate`: Date, `authorId`: String, `published`: Boolean |
| VideoCourse | Клас, що зберігає інформацію про відеокурс. | `courseId`: String, `courseName`: String, `authorId`: String, `language`: String, `poster`: String?, `numberLessons`: Int, `tags`: List<String>, `creationDate`: Date, `published`: Boolean |
| FirebaseVideoCourse | Клас, який використовується для взаємодії з Firebase з `courseId`. | `courseName`: String, `authorId`: String, `language`: String, `poster`: String?, `lessonCount`: Int, `tags`: List<String>, `creationDate`: Date, `published`: Boolean |
| Lesson | Абстрактний клас для різних типів уроків у додатку. | `lessonId`: String, `courseId`: String, `title`: String, `duration`: Int, `lessonOrder`: Int |
| LessonFirebase | Базовий клас для взаємодії з Firebase без `lessonId` і `courseId`. | `title`: String, `duration`: Int, `lessonOrder`: Int |
| TextLesson | Клас, що зберігає текстовий урок. | `lessonId`: String, `courseId`: String, `title`: String, `duration`: Int, `lessonOrder`: Int, `content`: String |
| TextLessonFirebase | Клас, який використовується для взаємодії з Firebase без `lessonId` та `courseId`. | `title`: String, `content`: String, `duration`: Int, `lessonOrder`: Int |
| VideoLesson | Клас, що зберігає інформацію про відеоурок. | `lessonId`: String, `courseId`: String, `title`: String, `duration`: Int, `lessonOrder`: Int, `videoUrl`: String, `transcription`: String, `creationDate`: Date |
| VideoLessonFirebase | Клас, який використовується для взаємодії з Firebase без `lessonId` та `courseId`. | `title`: String, `duration`: Int, `videoUrl`: String?, `transcription`: String?, `lessonOrder`: Int, `creationDate`: Date |

Імплементація даних моделей у коді наведена в додатку до роботи (А.1 Моделі данихДодаток А ).

### *3.2.5 Огляд імплементації основних програмних компонентів*

Нарешті переходимо до безпосередніх алгоритмів виконання ключових функціональних компонентів нашого додатку.  Нижчи ми оглянемо і наведемо код лише для головних компонентів, хоча і важко визначити, що головне, а що ні для нас. Проте, через специфіку платформи під яку ми розробляємо – Android, та і взагалі мобільних додатків, то це займає набагато більший об’єм коду, тому представити все тут – це вище моїх можливостей і в принципі немає жодного сенсу. Для того, щоб подивитися весь код – можна перейти на GitHub:

https://github.com/Jack-0ut/maverkick-android

Що ж тепер перейдемо до огляду.

**Реєстрація**

Реєстрація в нашому додатку працює так:

1. Введення даних: Користувач вводить свою електронну адресу, ім'я користувача та пароль у відповідні поля.

   Для цього ми створимо activity_registration.xml, в якому додаємо форми для введення email, username та password:

```
<EditText
    android:id="@+id/email_input"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Електронна адреса"
    ... />

<EditText
    android:id="@+id/username_input"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Ім'я користувача"
    ... />

<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/password_layout"
    ... >

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/password_input"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:hint="Пароль"
        android:inputType="textPassword"
        ... />
</com.google.android.material.textfield.TextInputLayout>
```

2. Перевірка введених даних: При натисненні на кнопку система перевіряє, чи введені дані відповідають вимогам: електронна адреса повинна бути коректною, ім'я користувача повинно містити не менше п'яти символів, а пароль - не менше шести.

```
binding.registerButton.setOnClickListener {
    val email = binding.emailInput.text.toString()
    val username = binding.usernameInput.text.toString()
    val password = binding.passwordInput.text.toString()

    when {
        !isValidEmail(email) ->
showSnackbar(getString(R.string.invalid_email_message))
        !isValidUsername(username) ->
showSnackbar(getString(R.string.invalid_username_message))
        !isValidPassword(password) ->
showSnackbar(getString(R.string.invalid_password_message))
        else -> registerUser(email, username, password)
    }
}
```

3. Реєстрація: Якщо введені дані коректні, вони передаються для реєстрації через відповідний репозиторій (authRepository). Якщо реєстрація успішна, дані користувача зберігаються локально, і він переходить на наступний етап створення профілю.

```
authRepository.register(email, username, password,
    onSuccess = { user ->
        sharedPrefManager.saveUser(user)
        sharedPrefManager.setIsOnboarded(false)
        startActivity(Intent(this, StudentOnboarding::class.java))
    },
    onFailure = {
        showSnackbar(getString(R.string.registration_failed_message))
        sharedPrefManager.clearUser()
        sharedPrefManager.clearPreferences()
    })
```

AuthReposity

```
    fun register(email: String, username: String, password: String,
onSuccess: (User) -> Unit, onFailure: (Exception) -> Unit) {
        firebaseAuth.createUserWithEmailAndPassword(email,
password).addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val userId = firebaseAuth.currentUser?.uid!!
                val userMap = hashMapOf(
                    "email" to email,
                    "username" to username
                )

                databaseService.db.collection("users").document(userId)
.set(userMap)
                    .addOnSuccessListener {
                        Log.d("Firestore", "Document written
successfully")
                        val user = User(userId, username, email, null)
                        onSuccess(user)
                    }
                    .addOnFailureListener { e ->
                        Log.e("Firestore", "Error writing document", e)
                        onFailure(e)
                    }
            } else {
                Log.e("FirebaseAuth", "Registration failed",
task.exception)
                onFailure(task.exception!!)
            }
        }
    }
```

## Логін

1) Введення даних: Користувач вводить свою електронну адресу та пароль у відповідні поля.

```
<EditText
    android:id="@+id/email_input"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Email"
    ... />

<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/password_layout"
```

```
    ... >

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/password_input"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Password"
        android:inputType="textPassword"
        ... />
</com.google.android.material.textfield.TextInputLayout>
```

2) Перевірка введених даних: При натисканні на кнопку "Увійти" система перевіряє, чи обидва поля заповнені.

```
binding.loginButton.setOnClickListener {
    val email = binding.emailInput.text.toString()
    val password = binding.passwordInput.text.toString()

    if (email.isEmpty()) {
        showSnackBar("Please enter email")
        return@setOnClickListener
    }

    if (password.isEmpty()) {
        showSnackBar("Please enter password")
        return@setOnClickListener
    }

    authRepository.login(email, password,
        onSuccess = { userId ->
            userRepository.getCurrentUser(
                onSuccess = { user ->
                    sharedPrefManager.saveUser(user)
                    checkUserTypeAndNavigate(userId)
                },
                onFailure = { e ->
                    showSnackBar("Failed to fetch user: ${e.message}")
                }
            )
        },
        onFailure = { e ->
            showSnackBar("Login failed: ${e.message}")
        }
    )
}
```

3) Перевірка типу користувача та перехід: Після успішного входу система перевіряє, чи є користувач студентом, викладачем або обома. Якщо користувач є обома, він може обрати, яким обліковим записом увійти.

```
private fun checkUserTypeAndNavigate(userId: String) {
    var isStudent = false
    var isTeacher = false
```

```
    lifecycleScope.launch {

        val studentResult = async {
studentRepository.getStudentById(userId) }
        val teacherResult = async {
teacherRepository.getTeacherById(userId) }

        studentResult.await().fold(
            onSuccess = {
                isStudent = true
                sharedPrefManager.saveStudent(it)
            },
            onFailure = {
                showSnackBar(getString(R.string.error_fetching_data))
            }
        )

        teacherResult.await().fold(
            onSuccess = {
                isTeacher = true
                sharedPrefManager.saveTeacher(it)
            },
            onFailure = {
                showSnackBar(getString(R.string.error_fetching_data))
            }
        )

        // Check for the different cases when we have role == student |
teacher or both
        when {
            isStudent && isTeacher -> showRoleSelectionDialog()
            isStudent -> navigateToMainActivity("student")
            isTeacher -> navigateToMainActivity("teacher")
            else -> {
                showSnackBar(getString(R.string.no_role_found, userId))
            }
        }
    }
}
```

4) Вибір ролі користувача (якщо необхідно): Якщо користувач є і студентом, і викладачем, він може вибрати, яким обліковим записом увійти.

```
private fun showRoleSelectionDialog() {
    val builder = AlertDialog.Builder(this)
    builder.setTitle(getString(R.string.choose_account_type))
    builder.setItems(arrayOf(getString(R.string.student),
getString(R.string.teacher))) { _, which ->
        when(which) {
            0 -> navigateToMainActivity("student")
            1 -> navigateToMainActivity("teacher")
        }
    }
    builder.show()
}
```

5) Перехід на головний екран: Після успішного входу користувача програма переходить на головний екран, враховуючи його роль.

```
private fun navigateToMainActivity(role: String) {
    // Save user role to shared preferences
    sharedPrefManager.saveActiveRole(role)

    val intentUri = when (role) {
        "student" -> Uri.parse("maverkick://student/main")
        "teacher" -> Uri.parse("maverkick://teacher/main")
        else -> throw IllegalArgumentException("Unknown role: $role")
    }
    val intent = Intent(Intent.ACTION_VIEW, intentUri)
    startActivity(intent)
    finish()
}
```

**Створення навчального плану**

Напевно головна особливість та унікальність нашого додатку – це створення персоналізованих щоденних навчальних планів, які складаються з уроків із курсу(ів), на які ми підписані.

Це один із способів збільшити мотивацію користувача і бачити чіткі межі того, що людина повинна завершити сьогодні.

Починається алгоритм в StudentHomeFragment (головний фрагмент, який запускається після відкриття додатку), а точніше у HomeViewModel, яка працює на цей фрагмент і займається підготовкою даних. У loadDailyLearningPlan() ми передаємо id студента та кількість хвилин на день він хоче вчитися і dailyLearningPlanRepository.getDailyLearningPlanForStudent() повинна повернути готовий навчальний план:

```
init {
    if (student?.enrolledCourses?.isEmpty() == true) {
        _navigateToCourseEnrollment.postValue(true)
    } else {
        loadDailyLearningPlan()
    }
}

private fun loadDailyLearningPlan() {
    viewModelScope.launchSafe {
        val plan = dailyLearningPlanRepository.getDailyLearningPlanForStudent(
            student?.studentId ?: "",
            student?.dailyStudyTimeMinutes ?: 0
        )
        _dailyLearningPlan.value = plan
        _currentLessonIndex.value = plan.progress
    }
}
```

Тепер заглибимося всередину цієї функції.

**getDailyLearningPlanForStudent**: Цей метод перевіряє, чи існує вже план навчання для студента. Якщо так, він повертає існуючий план. В іншому випадку він намагається отримати сьогоднішні уроки для створення нового плану.

```
/** Check if learning plan exists **/
suspend fun getDailyLearningPlanForStudent(studentId: String,
dailyStudyTimeMinutes: Int): DailyLearningPlan {
    val existingPlan = getDailyLearningPlan(studentId)
    if (existingPlan != null && existingPlan.lessons.isNotEmpty()) {
        return existingPlan
    }

    // Try to fetch lessons and create a new plan
    val todayPairs = getTodayLessons(studentId, dailyStudyTimeMinutes * 60)
    if (todayPairs.isNotEmpty()) {
        val newPlan = createDailyLearningPlan(studentId, todayPairs)
        storeDailyLearningPlan(newPlan)
        return newPlan
    }

    throw Exception("Failed to generate a daily learning plan")
}
```

Безпосередньо створення відбувається у getTodayLessons(studentId, dailyStudyTimeMinutes * 60).

Ось короткий огляд того, що саме відбувається в даній функції:

1) Отримання активних курсів для студента: Функція fetchActiveCourses(studentId) викликається, щоб отримати список активних курсів для конкретного студента за допомогою його ідентифікатора. Ця функція повертає список документів з колекції курсів, які активовані для даного студента.
2) Отримання уроків для кожного активного курсу: Для кожного документа курсу у списку активних курсів ми отримуємо courseId та courseType. Потім викликається функція fetchLessonsThatFitsTime(courseId, studentId, courseType, dailyStudyTimeSeconds), яка повертає список уроків, які підходять за часом для конкретного курсу та студента. Ці уроки додаються до черги lessonQueues.

3) Підготовка до обробки уроків: Створюється порожній список todayLessons, який буде містити уроки для сьогодні. Змінна totalLessonTime ініціалізується як 0, індекс і встановлюється на 0.
4) Цикл розподілу уроків: Виконується цикл, поки загальний час уроків totalLessonTime менший за dailyStudyTimeSeconds та є щонайменше одна черга з уроками lessonQueues, яка не порожня. У цьому циклі ми пройдемо через усі черги уроків у lessonQueues та додамо уроки до todayLessons, які можна вмістити в сьогоднішній час навчання. Кожен раз, коли ми додаємо урок до todayLessons, ми також видаляємо його з черги. Це відбувається в тому випадку, якщо загальний час уроків разом з новим уроком перевищує dailyStudyTimeSeconds.
5) Повернення сьогоднішніх уроків: Після завершення циклу повертається список todayLessons, який містить усі уроки, які можна пройти за сьогоднішній час навчання.

```
/** Get the list of today's lessons for the student **/
private suspend fun getTodayLessons(studentId: String, dailyStudyTimeSeconds:
Int): List<Lesson> {
    // Fetch the active courses for the student
    val courseDocs = fetchActiveCourses(studentId)

    val lessonQueues = mutableListOf<Queue<Lesson>>()

    // Fetch the lessons for each active course
    for (doc in courseDocs.documents) {
        val courseId = doc.getString("courseId") ?: continue
        val courseType = CourseType.valueOf(doc.getString("courseType") ?:
continue)
        val lessons = fetchLessonsThatFitsTime(courseId, studentId, courseType,
dailyStudyTimeSeconds)
        lessonQueues.add(LinkedList(lessons))
    }

    // Round-robin through the lesson queues
    val todayLessons = mutableListOf<Lesson>()
    var totalLessonTime = 0
    var i = 0

    while (totalLessonTime < dailyStudyTimeSeconds && lessonQueues.any {
it.isNotEmpty() }) {
        val queue = lessonQueues[i % lessonQueues.size]
        if (queue.isNotEmpty()) {
            val lesson = queue.peek()
            val lessonTime = lesson!!.duration
            // Add the lesson to the list if there is enough remaining study
time
            if (totalLessonTime + lessonTime > dailyStudyTimeSeconds) {
                break // If the next lesson doesn't fit, stop adding lessons
            }
```

```
            todayLessons.add(lesson)
            totalLessonTime += lessonTime
            queue.poll() // now that we're sure we're using this lesson, remove
it from the queue
        }
        i++
    }
    return todayLessons
}
```

Повний код цього файлу можна побачити в додатку (А.2 Денний навчальний план).

**Відео-Урок**

Коли користувач запускає відеоурок, він отримує доступ до відеоматеріалу, що відтворюється через компонент ExoPlayer. Відеоплеєр налаштовується для відтворення відео, яке надходить через URL-адресу. Відображення відео та контроль за його відтворенням здійснюється в межах активності VideoLessonActivity.

Крім відтворення відео, користувач має можливість взаємодіяти з матеріалом. Натискання кнопки "Чат" призупиняє відтворення відео та відкриває діалогове вікно, де студент може задавати питання AI-помічнику. Це забезпечує можливість отримати пояснення або допомогу прямо під час перегляду відео.

Крім того, користувач має можливість виразити свою реакцію на відео, натиснувши кнопку "Реакція". Це відкриває діалогове вікно з емодзі, за допомогою яких можна висловити враження від матеріалу.

Після завершення відеоуроку користувач автоматично переходить до завдань, а також здійснюється збір даних про оцінку та реакцію користувача для подальшого аналізу.

Усі ці можливості відеоуроку реалізовані в одному класі - **VideoLessonActivity**:

```
package com.example.video_lesson.student

import android.app.Dialog
import android.content.res.Configuration
import android.graphics.Color
import android.graphics.drawable.ColorDrawable
import android.os.Bundle
import android.view.View
import android.view.WindowManager
import android.widget.GridView
import androidx.activity.viewModels
import androidx.core.content.ContextCompat
```

```kotlin
import androidx.core.os.bundleOf
import com.example.chat_helper.AskQuestionDialogFragment
import com.example.video_lesson.AbstractVideoActivity
import com.example.video_lesson.ExoPlayerWrapper
import com.example.video_lesson.R
import com.example.video_lesson.VideoPlayerInterface
import com.example.video_lesson.adapters.EmojiAdapter
import com.example.video_lesson.databinding.ActivityVideoLessonBinding
import com.maverkick.data.models.CourseType
import com.maverkick.tasks.ExerciseActivity
import dagger.hilt.android.AndroidEntryPoint

/**
 * This is the VideoActivity class. It is responsible for displaying the video
lessons to the students.
 * The activity allows students to interact with the video by pausing and
playing it.
 * They can also ask questions while the video is playing by clicking on the
'ASK' icon.
 */
@AndroidEntryPoint
class VideoLessonActivity : AbstractVideoActivity() {

    private lateinit var binding: ActivityVideoLessonBinding
    private val viewModel: VideoLessonViewModel by viewModels()

    private lateinit var lessonId: String
    private lateinit var videoUri: String
    private lateinit var transcription: String
    private lateinit var title: String
    private lateinit var courseId: String

    private val emojis = listOf("🤩", "😊", "😐", "😕", "😠")

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        window.statusBarColor = ContextCompat.getColor(this,
com.maverkick.common.R.color.main_tone_color)

        intent?.extras?.let {
            lessonId = it.getString("lessonId", "")
            videoUri = it.getString("videoUri", "")
            transcription = it.getString("transcription", "")
            title = it.getString("title", "")
            courseId = it.getString("courseId", "")
        }

        binding.apply {
            videoTitle.text = title
            videoView.keepScreenOn = true
            chatButton.setOnClickListener { onChatButtonClick() }
            reactionButton.setOnClickListener { onReactionButtonClick() }
        }
    }

    private fun onChatButtonClick() {
        player?.playWhenReady = false
```

```kotlin
        window.statusBarColor = ContextCompat.getColor(this,
com.maverkick.common.R.color.maverkick_main)

        val dialogFragment = AskQuestionDialogFragment().apply {
            arguments = bundleOf(
                "courseId" to courseId,
                "transcription" to transcription,
                "lessonId" to lessonId
            )
            dismissListener = { window.statusBarColor =
ContextCompat.getColor(this@VideoLessonActivity,
com.maverkick.common.R.color.main_tone_color) }
        }
        dialogFragment.show(supportFragmentManager, "AskQuestionDialogFragment")
    }

    private fun onReactionButtonClick() {
        val dialog = Dialog(this).apply {
            setContentView(R.layout.emoji_picker)
            window?.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))

            findViewById<GridView>(R.id.grid_view).apply {
                adapter = EmojiAdapter(this@VideoLessonActivity,
R.layout.item_emoji, emojis) { _, position ->
                    dismiss()
                    viewModel.setLessonRating(5 - position)
                }
            }
        }
        dialog.show()
    }

    override fun onConfigurationChanged(newConfig: Configuration) {
        super.onConfigurationChanged(newConfig)
        binding.videoTitle.visibility = if (newConfig.orientation ==
Configuration.ORIENTATION_LANDSCAPE) {
            window.addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN)
            View.GONE
        } else {
            window.clearFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN)
            View.VISIBLE
        }
    }

    override val videoUrl: String
        get() = videoUri

    override fun onPlayerEnd() {
        val intent = ExerciseActivity.newIntent(this, courseId, lessonId,
CourseType.VIDEO)
        startActivity(intent)
        finish()
        viewModel.updateRatings(courseId, {}, {})
    }

    override fun initializeBinding() {
        binding = ActivityVideoLessonBinding.inflate(layoutInflater)
        setContentView(binding.root)
```

```
    }

    override fun setPlayerView(player: VideoPlayerInterface) {
        val exoSimplePlayer = player as? ExoPlayerWrapper
        binding.videoView.player = exoSimplePlayer?.playerInstance
    }

    override fun createPlayerInstance(): VideoPlayerInterface {
        return ExoPlayerWrapper(this)
    }

    override fun onDestroy() {
        super.onDestroy()
        window.statusBarColor = ContextCompat.getColor(this,
com.maverkick.common.R.color.maverkick_main)
    }
}
```

І VideoLessonViewModel, в якій ми зберігаємо оцінку за урок:

```
@HiltViewModel
class VideoLessonViewModel @Inject constructor(
    private val courseStatisticsRepository: CourseStatisticsRepository
) : ViewModel() {

    private val _lessonRating = MutableLiveData<Int>()

    /** Store rating locally**/
    fun setLessonRating(rating: Int) {
        _lessonRating.value = rating
    }

    /** Update the rating for the course in the database **/
    fun updateRatings(courseId: String, onSuccess: () -> Unit, onFailure:
(Exception) -> Unit) {
        val rating = _lessonRating.value ?: return
        courseStatisticsRepository.updateRatings(courseId, rating, onSuccess,
onFailure)
    }
}
```

**Текстовий Урок**

Ця частина імлементована в одному невеликому класі TextLessonActivity. Коли користувач запускає урок, він отримує зміст уроку у форматі веб-сторінки. Це виконується за допомогою компонента WebView. Урок оформлений так, щоб було комфортно читати, з використанням стилів CSS та HTML. Ми зчитуємо інформацію у вигляді тексту в якому крім самого змісту є певні елементи, які показують який дизайн використати, наприклад різний рівень заголовків в залежності від кількості # перед текстом.

Крім того, користувач може почати чат з AI-помічником, натиснувши відповідну кнопку. Це дозволяє задавати питання та отримувати відповіді прямо в межах додатка. Після завершення уроку користувач може перейти до наступного кроку навчання, натиснувши кнопку "Закінчити". Весь текстовий контент уроку обробляється перед відображенням, щоб забезпечити зручність та читабельність.

Нижче можна побачити клас, який це імплементує:

```
@AndroidEntryPoint
class TextLessonActivity : AppCompatActivity() {

    private lateinit var binding: ActivityTextLessonBinding
    private lateinit var webView: WebView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityTextLessonBinding.inflate(layoutInflater)
        setContentView(binding.root)
        window.statusBarColor = ContextCompat.getColor(this,
com.maverkick.common.R.color.maverkick_main)

        webView = binding.lessonContent
        webView.settings.javaScriptEnabled = true

        val courseId = intent.getStringExtra("courseId") ?: ""
        val lessonId = intent.getStringExtra("lessonId") ?: ""
        val title = intent.getStringExtra("title") ?: ""
        val content = intent.getStringExtra("content") ?: ""

        binding.lessonTitle.text = title

        val htmlContent = convertToHtml(content)

        val finalContent = """
        <html><head>
        <style>
            body {
                font-family: 'Georgia', serif; /* Add your preferred font */
                font-size: 18px;
                line-height: 1.6;
                color: #333;
                background-color: #FDFCFC;
            }
            h1, h2, h3 {
                margin-top: 0px;
                margin-bottom: 0px;
            }
            h1 {
                font-size: 20px;
            }
            h2 {
                font-size: 18px;
            }
```

```
            h3 {
                font-size: 16px;
            }
            strong {
                font-weight: bold;
            }
            pre {
                white-space: pre-wrap;       /* Since CSS 2.1 */
                white-space: -moz-pre-wrap;  /* Mozilla, since 1999 */
                white-space: -pre-wrap;      /* Opera 4-6 */
                white-space: -o-pre-wrap;    /* Opera 7 */
                word-wrap: break-word;       /* Internet Explorer 5.5+ */
            }

            code {
                font-family: 'Courier New', Courier, monospace; /* or any other
monospaced font */
            }
        </style>
        <link
href='https://cdnjs.cloudflare.com/ajax/libs/prism/1.25.0/themes/prism.css'
rel='stylesheet' />
        <script
src='https://cdnjs.cloudflare.com/ajax/libs/prism/1.25.0/prism.js'></script>
        <script type="text/x-mathjax-config">
            MathJax.Hub.Config({
              tex2jax: {
                inlineMath: [['$','$'], ['\\(','\\)']],
                processEscapes: true
              }
            });
        </script>
        <script type="text/javascript" async

src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.7/MathJax.js?config=TeX-
MML-AM_CHTML">
        </script>
        </head><body>
        $htmlContent
        <script type='text/javascript'>
            MathJax.Hub.Queue(['Typeset',MathJax.Hub]);
        </script></body></html>
        """.trimIndent()

        webView.loadDataWithBaseURL(null, finalContent, "text/html", "UTF-8",
null)

        setupAskButton(courseId, lessonId, content)
        setupFinishButton(courseId, lessonId)
    }

    private fun setupAskButton(courseId: String, lessonId: String, content:
String) {
        binding.askButton.setOnClickListener {
            window.statusBarColor = ContextCompat.getColor(this,
com.maverkick.common.R.color.maverkick_main)

            val dialogFragment = AskQuestionDialogFragment().apply {
```

```
                arguments = Bundle().apply {
                    putString("courseId", courseId)
                    putString("transcription", content)
                    putString("lessonId", lessonId)
                }
            }
            dialogFragment.dismissListener = {
                window.statusBarColor = ContextCompat.getColor(this,
com.maverkick.common.R.color.maverkick_white)
            }
            dialogFragment.show(supportFragmentManager,
"AskQuestionDialogFragment")
        }
    }

    private fun setupFinishButton(courseId: String, lessonId: String) {
        binding.finishButton.setOnClickListener {
            val intent = ExerciseActivity.newIntent(this, courseId, lessonId,
CourseType.TEXT)
            startActivity(intent)
            finish()
        }
    }

    private fun convertToHtml(markdown: String): String {
        val lines = markdown.split("\n")
        val htmlLines = mutableListOf<String>()

        var inCodeBlock = false

        for (line in lines) {
            when {
                line.startsWith("```") -> {
                    inCodeBlock = !inCodeBlock
                    if (inCodeBlock) {
                        htmlLines.add("<pre><code>")
                    } else {
                        htmlLines.add("</code></pre>")
                    }
                }
                inCodeBlock -> {
                    htmlLines.add(line)
                }
                line.startsWith("#### ") ->
htmlLines.add("<h3>${line.replaceFirst("#### ", "")}</h3>")
                line.startsWith("### ") ->
htmlLines.add("<h3>${line.replaceFirst("### ", "")}</h3>")
                line.startsWith("## ") ->
htmlLines.add("<h3>${line.replaceFirst("## ", "")}</h3>")
                line.startsWith("# ") ->
htmlLines.add("<h2>${line.replaceFirst("# ", "")}</h2>")
                line.startsWith("[") && line.endsWith("]") -> {
                    val sectionContent = line.substring(1, line.length - 1)
                    htmlLines.add("<div
class='section'><em>$sectionContent</em></div>")
                }
                line.contains("**") -> {
                    var modifiedLine = line
```

```
                    val regex = Regex("\\*\\*(.+?)\\*\\*")
                    val matchResult = regex.findAll(line)
                    matchResult.forEach { match ->
                        modifiedLine = modifiedLine.replace(match.value,
"<strong>${match.groups[1]?.value}</strong>")
                    }
                    htmlLines.add(modifiedLine)
                }
                line.contains("(") && line.contains(")") -> {
                    // If the line contains LaTeX formula (between parentheses),
wrap it with appropriate MathJax delimiters
                    var modifiedLine = line
                    val regex = Regex("\\((.+?)\\)")
                    val matchResult = regex.findAll(line)
                    matchResult.forEach { match ->
                        modifiedLine = modifiedLine.replace(match.value,
"\\(${match.groups[1]?.value}\\)")
                    }
                    htmlLines.add(modifiedLine)
                }
                else -> htmlLines.add(line)
            }
        }
        return htmlLines.joinToString(separator = "<br/>")
    }

}
```

**Чат-бот для відповідей на питання**

1) Відкриття діалогового вікна чату: Користувач натискає на відповідну іконку або кнопку, щоб відкрити діалогове вікно чату.
2) Ініціалізація фрагменту чату: При натисканні на кнопку відкриття чату, відбувається ініціалізація фрагменту AskQuestionDialogFragment, який відповідає за відображення чату з AI-помічником.
3) Встановлення параметрів чату: Перед відображенням чату, необхідно встановити параметри, такі як courseId, lessonId та transcription, які використовуються для ініціалізації чату та отримання відповіді від AI-помічника.
4) Початок розмови: Після ініціалізації фрагменту та встановлення параметрів, починається розмова з AI-помічником. Фрагмент починає запитувати відповіді від AI-помічника на основі переданих параметрів.
5) Надсилання повідомлення користувачем: Користувач може вводити повідомлення в поле введення тексту та натискати на кнопку відправки, щоб надіслати повідомлення до AI-помічника.
6) Отримання та відображення відповіді в чаті: Після надсилання повідомлення AI-помічнику, фрагмент очікує на відповідь. Після отримання відповіді, вона відображається в чаті.

7) Оновлення UI та взаємодія з користувачем: Під час процесу розмови та отримання відповідей від AI-помічника, UI оновлюється для відображення нових повідомлень та інших елементів. Користувач може продовжувати спілкування та взаємодіяти з чатом до завершення.

Нижче наведено повну імплементацію ключових елементів коду фрагменту для діалогу, оскільки поділ на шматки не має сенсу:

```
package com.example.chat_helper
…
@AndroidEntryPoint
class AskQuestionDialogFragment : DialogFragment() {
    private val chatViewModel: ChatViewModel by activityViewModels()
    private lateinit var binding: FragmentAskQuestionDialogBinding

    private lateinit var courseId: String
    private lateinit var lessonId: String
    private lateinit var transcription: String
    private val maxRequests = 5
    private var progressBarGIFDialogBuilder: ProgressBarGIFDialog.Builder? =
null

    var dismissListener: (() -> Unit)? = null

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentAskQuestionDialogBinding.inflate(inflater, container,
false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        initVariablesFromArguments()
        initRecyclerView()
        observeViewModel()
        startConversation()
        handleSendMessage()
    }

    private fun initVariablesFromArguments() {
        arguments?.let {
            courseId = it.getString("courseId", "")
            lessonId = it.getString("lessonId", "")
            transcription = it.getString("transcription", "")
            if (courseId.isBlank() || lessonId.isBlank() ||
transcription.isBlank()) {
                showSnackbar("Missing argument data!")
                dismiss()
            }
        } ?: run {
            showSnackbar("No arguments found!")
```

```
            dismiss()
        }
    }

    private fun showSnackbar(message: String) {
        Snackbar.make(requireView(), message, Snackbar.LENGTH_SHORT).show()
    }

    private fun initRecyclerView() {
        val adapter =
com.example.chat_helper.adapters.ChatMessageAdapter(mutableListOf())
        binding.recyclerView.adapter = adapter
        binding.recyclerView.layoutManager = LinearLayoutManager(context)
    }

    private fun observeViewModel() {
        chatViewModel.messages.observe(viewLifecycleOwner) { messages ->
            (binding.recyclerView.adapter as
com.example.chat_helper.adapters.ChatMessageAdapter).updateMessages(messages)
            binding.recyclerView.scrollToPosition(messages.size - 1)
        }

        observeMessageGeneration()
        observeRequestCount()
    }

    private fun observeMessageGeneration() {
        chatViewModel.isMessageGenerationInProgress.observe(viewLifecycleOwner)
{ inProgress ->
            handleProgress(inProgress)
        }
    }

    private fun handleProgress(inProgress: Boolean) {
        if (inProgress) {
            showProgress()
        } else {
            hideProgress()
        }
    }

    private fun showProgress() {
        progressBarGIFDialogBuilder =
ProgressBarGIFDialog.Builder(requireActivity())
            .setCancelable(false)
            .setTitleColor(com.maverkick.common.R.color.black)
            .setLoadingGif(com.maverkick.common.R.drawable.progress)
            .setDoneTitle("Here it is!") // Set Done Title
            .setLoadingTitle("Wait, I'm working on answer...")

        progressBarGIFDialogBuilder?.build()
    }

    private fun hideProgress() {
        progressBarGIFDialogBuilder?.clear()
        progressBarGIFDialogBuilder = null
    }
```

```
    private fun startConversation() {
        chatViewModel.startConversation(courseId, lessonId, transcription)
    }

    private fun observeRequestCount() {
        chatViewModel.requestCount.observe(viewLifecycleOwner) { count ->
            handleRequestCount(count)
        }
    }

    private fun handleRequestCount(count: Int) {
        updateCounter(count)
        adjustIconAlpha(count)
        applyFadeInAnimation()
    }

    private fun updateCounter(count: Int) {
        binding.counter.text = "${maxRequests - count}"
    }


    private fun handleSendMessage() {
        binding.inputLayout.setEndIconOnClickListener {
            val messageText = binding.promptInput.text.toString()
            if (messageText.isNotBlank() && chatViewModel.requestCount.value!! <
maxRequests) {
                processUserMessage(messageText)
            } else if (chatViewModel.requestCount.value!! >= maxRequests) {
                Toast.makeText(context, "Maximum number of requests reached",
Toast.LENGTH_SHORT).show()
            }
        }
    }

    private fun processUserMessage(messageText: String) {
        val message = AiChatMessage(messageText, true)
        chatViewModel.addMessage(message)
        binding.promptInput.text?.clear()

        hideKeyboard()

        chatViewModel.sendMessage(courseId, lessonId, messageText)
        chatViewModel.incrementRequestCount()
    }
…
}
```

А також viewModel, яка безпосередньо звертається до сервера, щоб отрмати відповідь і додає відповідь до списку _messages.

Далі оскільки фрагмент бачить зміну в _messages через публічну список повідомлень messages, то нове повідомлення відображається в фрагменті.

```
package com.example.chat_helper
…
@HiltViewModel
class ChatViewModel @Inject constructor(
```

```
    private val sharedPrefManager: SharedPrefManager,
    private val chatApi: ChatApi,
) : ViewModel() {

    private val _messages =
MutableLiveData<MutableList<AiChatMessage>>(mutableListOf())
    val messages: LiveData<MutableList<AiChatMessage>> get() = _messages

    private val _isMessageGenerationInProgress = MutableLiveData(false)
    val isMessageGenerationInProgress: LiveData<Boolean> get() =
_isMessageGenerationInProgress

    private val _error = MutableLiveData<String>()
    val error: LiveData<String> get() = _error

    private val _requestCount = MutableLiveData(0)
    val requestCount: LiveData<Int> get() = _requestCount

    fun incrementRequestCount() {
        _requestCount.value = _requestCount.value?.plus(1)
    }

    fun addMessage(message: AiChatMessage) {
        val updatedMessages = _messages.value?.toMutableList() ?:
mutableListOf()
        updatedMessages.add(message)
        _messages.postValue(updatedMessages)
    }

    fun startConversation(courseId: String, lessonId: String, context: String) {
        sharedPrefManager.getStudent()?.studentId?.let {
            handleNetworkRequest(
                request = {
chatApi.startConversation(StartConversationRequest(it, courseId, lessonId,
context)) },
                onError = { _error.postValue("Sorry, can't start the
conversation") }
            )
        } ?: run { _error.postValue("User is not logged in.") }
    }

    fun sendMessage(courseId: String, lessonId: String, messageText: String) {
        val studentId = sharedPrefManager.getStudent()?.studentId
        _isMessageGenerationInProgress.postValue(true)
        studentId?.let {
            handleNetworkRequest(
                request = { chatApi.sendMessage(SendMessageRequest(it, courseId,
lessonId, messageText)) },
                onSuccess = { response ->
                    response.body()?.message?.let { responseMessage ->
                        if (responseMessage.isNotBlank()) {
                            val botMessage = AiChatMessage(responseMessage,
false)

                            addMessage(botMessage)
                        }
                    }
                },
                onError = { _error.postValue("Sorry, can't give you an answer")
```

```
},
                onComplete = { _isMessageGenerationInProgress.postValue(false) }
            )
        } ?: run { _error.postValue("User is not logged in.") }
    }

    private fun <T> handleNetworkRequest(
        request: suspend () -> Response<T>,
        onSuccess: suspend (Response<T>) -> Unit = {},
        onError: () -> Unit = {},
        onComplete: () -> Unit = {}
    ) {
        viewModelScope.launch(Dispatchers.IO) {
            try {
                val response = request()
                if (response.isSuccessful) {
                    onSuccess(response)
                } else {
                    onError()
                }
            } catch (e: Exception) {
                _error.postValue("Network Request Error: ${e.message}")
                onError()
            } finally {
                onComplete()
            }
        }
    }
}
```

На сервері, який є звичайною serverless інфраструктурою з використанням Google Cloud Platform інструменту Cloud Run.

Для імплементації серверу використали найпростіший Python фреймворк Flask [82]. Він дозволяє створювати прості API для будь-яких випадків і при цьому максимально легкий в налаштуванні та в деплойменті.

У нашому сервері, якщо користувач надіслав перше запитання, ми викликаємо /start endpoint, і в ній ми ініціалізуємо новий документ в базі даних реального часу Firebase Realtime Database.

Далі відбуваються запити до /chat. В тілі запити передаємо user_id, course_id, lesson_id та message, повертаємо також message, але вже згенеровану OpenAI GPT3.5-Turbo Api. Нижче наведено весь код нашого серверу:

```
from flask import Flask, request, jsonify
import openai
import os
import firebase_admin
from firebase_admin import credentials, db
import json
from dotenv import load_dotenv
```

```
load_dotenv()

app = Flask(__name__)

openai.api_key = os.getenv("OPENAI_API_KEY")

# Parse the JSON service account key from the environment variable
firebase_service_account_key =
json.loads(os.getenv("FIREBASE_SERVICE_ACCOUNT"))

# Use the parsed key to initialize the Firebase Admin SDK
cred = credentials.Certificate(firebase_service_account_key)
firebase_admin.initialize_app(cred, {
    'databaseURL': 'https://maverkick-app-default-rtdb.europe-
west1.firebasedatabase.app/'
})

@app.route('/start', methods=['POST'])
def start():
    user_id = request.json['user_id']
    course_id = request.json['course_id']
    lesson_id = request.json['lesson_id']
    context = request.json['context']

    # Adjust the Firebase reference to include the course_id
    ref = db.reference(f'conversations/{user_id}/{course_id}/{lesson_id}')
    ref.set([{"role": "system", "content": f"You are an education assistant.
You've been provided with a video transcription to help answer questions.
The transcription is: {context}. Please keep the conversation focused on
this topic."},
             {"role": "assistant", "content": f"Hi there, I'm El Bricko,
your friendly and funny assistant. "
             f"I'm here to make learning fun by providing explanations using
real-world examples and scenarios. "
             f"Let's not be too formal and keep things interesting!"}])

    return jsonify({"status": "success"})


@app.route('/chat', methods=['POST'])
def chat():
    user_id = request.json['user_id']
    course_id = request.json['course_id']
    lesson_id = request.json['lesson_id']
    user_message = request.json['message']
```

```python
    # Adjust the Firebase reference to include the course_id
    ref = db.reference(f'conversations/{user_id}/{course_id}/{lesson_id}')
    history = ref.get()
    if not history:
        return jsonify({"error": "No conversation found for this user,
course and lesson."}), 404

    history.append({"role": "user", "content": user_message})

    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=history
    )
    assistant_message = response['choices'][0]['message']['content']

    history.append({"role": "assistant", "content": assistant_message})
    ref.set(history)

    return jsonify({"message": assistant_message})


if __name__ == '__main__':
    # Use the 'PORT' environment variable
    port = int(os.environ.get('PORT', 8080))
    app.run(host='0.0.0.0', port=port, debug=False)
```

Для здійснення запитів на клієнтській стороні використовуємо retrofit2, яка дозволяє визначати api ендпоінти як функції і проводити серіалізацію отриманих даних у той формат даних, який ми викорситовуємо в Android [82]. Для того, щоб інкапсулювати логіку ми створили інтерфейс ChatApi, який і використовується в ChatViewModel для надсилання та отримання відповідей:

```kotlin
/**
 * ChatApi defines the REST API endpoints for interacting with 'El Bricko' chat.
 * This interface should be used with a Retrofit instance to initiate and manage
 * chat conversations.
 */
interface ChatApi {

    /**
     * Starts a conversation with 'El Bricko' and initializes it on the server.
     *
     * @param request The request object containing details needed to start the
conversation,
     *                such as user ID, course ID, lesson ID, and video lesson
transcription.
     * @return A Response object containing the conversation ID, needed for
further interaction.
```

```kotlin
     * @see StartConversationRequest for detailed information about the request
object.
     */
    @POST("/start")
    suspend fun startConversation(
        @Body request: StartConversationRequest
    ): Response<StartConversationResponse>

    /**
     * Sends a message to the server to get an answer from 'El Bricko'.
     *
     * @param request The request object containing details needed for the chat
interaction,
     *                such as user ID, course ID, lesson ID, and user's message
text.
     * @return A Response object containing the message from 'El Bricko'.
     * @see SendMessageRequest for detailed information about the request
object.
     */
    @POST("/chat")
    suspend fun sendMessage(
        @Body request: SendMessageRequest
    ): Response<SendMessageResponse>
}

/**
 * StartConversationRequest represents the JSON body for the `/start` endpoint.
 * It encapsulates the parameters needed to initiate a conversation with 'El
Bricko'.
 *
 * @property userID The unique identifier for the user.
 * @property courseID The unique identifier for the course.
 * @property lessonID The unique identifier for the lesson.
 * @property context The transcription of the video lesson.
 */
data class StartConversationRequest(
    @SerializedName("user_id") val userID: String,
    @SerializedName("course_id") val courseID: String,
    @SerializedName("lesson_id") val lessonID: String,
    @SerializedName("context") val context: String
)

/**
 * StartConversationResponse represents the server's response to starting a
conversation.
 *
 * @property conversationID The unique identifier for the initiated
conversation.
 */
data class StartConversationResponse(
    val conversationID: String
)

/**
 * SendMessageRequest represents the JSON body for the `/chat` endpoint.
 * It encapsulates the parameters needed to send a message to 'El Bricko'.
 *
 * @property userID The unique identifier for the user.
```

```
 * @property courseID The unique identifier for the course.
 * @property lessonID The unique identifier for the lesson.
 * @property message The text of the message sent by the user.
 */
data class SendMessageRequest(
    @SerializedName("user_id") val userID: String,
    @SerializedName("course_id") val courseID: String,
    @SerializedName("lesson_id") val lessonID: String,
    @SerializedName("message") val message: String
)

/**
 * SendMessageResponse represents the server's response to sending a message to
'El Bricko'.
 *
 * @property message The text of the message received from 'El Bricko'.
 */
data class SendMessageResponse(
    val message: String
)
```

### Завдання після уроку

Після завершення відеоуроку, користувач переходить до завдань, які служать для закріплення матеріалу. Це відбувається в активності ExerciseActivity. Користувачу пропонується серія завдань, пов'язаних з темою відеоуроку.

Після завершення кожного завдання користувач має можливість перейти до наступного завдання або до підсумкової сторінки. Кнопка "Пропустити" дозволяє перейти до наступного завдання без відповіді на поточне.

Коли всі завдання виконані, користувач переходить до підсумкової сторінки, де він може побачити кількість правильних відповідей та загальну кількість запитань. Після цього він повертається до домашньої сторінки студента.

Усі ці можливості реалізовані в класі **ExerciseActivity**, який запускається після завершення відеоуроку.

```
package com.maverkick.tasks

…

@AndroidEntryPoint
class ExerciseActivity : AppCompatActivity(), OptionSelectionListener {
    private lateinit var binding: ActivityExerciseBinding
    private val viewModel: ExerciseViewModel by viewModels()
    private lateinit var adapter: TaskPagerAdapter

    private var correctAnswersCount = 0
    private val finishedItems = mutableSetOf<Int>()

    override fun onCreate(savedInstanceState: Bundle?) {
```

```kotlin
        super.onCreate(savedInstanceState)
        initializeViews()
        observeViewModel()
        handleIntents()
    }

    private fun initializeViews() {
        binding = ActivityExerciseBinding.inflate(layoutInflater)
        setContentView(binding.root)
        window.statusBarColor = ContextCompat.getColor(this,
com.maverkick.common.R.color.maverkick_main)

        // disable check button, because initially it's not clicked on
        binding.checkButton.isEnabled = false

        adapter = TaskPagerAdapter(this)
        binding.viewPager.adapter = adapter
        binding.viewPager.registerOnPageChangeCallback(object :
ViewPager2.OnPageChangeCallback() {
            override fun onPageSelected(position: Int) {
                super.onPageSelected(position)
                updateProgress(position)

                if(finishedItems.contains(position)) {
                    binding.skipButton.visibility = View.GONE
                } else {
                    binding.skipButton.visibility = View.VISIBLE
                }
            }
        })

        binding.skipButton.setOnClickListener {
            val nextItem = binding.viewPager.currentItem + 1
            if (nextItem < adapter.itemCount) {
                binding.viewPager.currentItem = nextItem
                updateProgress(nextItem)
                setDefaultState()
            } else {
                navigateToQuizSummary()
            }
        }

        binding.checkButton.setOnClickListener {
            if (!binding.checkButton.isEnabled) return@setOnClickListener

            when (binding.checkButton.text) {
                getString(R.string.check_button_text) ->
handleCheckButtonState()
                getString(R.string.next_button_text) -> handleNextButtonState()
            }
        }
    }

    private fun navigateToQuizSummary() {
        val courseId = intent.getStringExtra("courseId") ?: return
        val lessonId = intent.getStringExtra("lessonId") ?: return

        val totalQuestions = adapter.itemCount
```

```kotlin
        viewModel.finishLesson(courseId, lessonId)

        val redirectIntent = Intent(this,
PostLessonSummaryActivity::class.java).apply {
            putExtra("CORRECT_ANSWERS", correctAnswersCount)
            putExtra("TOTAL_QUESTIONS", totalQuestions)
        }
        startActivity(redirectIntent)
        finish()
    }

    override fun onOptionSelected(isSelected: Boolean) {
        binding.checkButton.isEnabled = isSelected
    }

    /** Init the tasks when the activity is loaded first time **/
    private fun observeViewModel() {
        lifecycleScope.launch {
            lifecycle.repeatOnLifecycle(Lifecycle.State.STARTED) {
                viewModel.tasks.collect { tasks ->
                    binding.taskProgress.max = tasks.size
                    if (tasks.isEmpty() && !viewModel.isLoading.value) {
                        finishExercises()
                    } else {
                        adapter.setTasks(tasks)
                        binding.taskProgress.progress = 0
                    }
                }
            }
        }

        viewModel.checkAnswerEvent.observe(this) { event ->
            event.getContentIfNotHandled()?.let { result ->
                if (result.first) {
                    correctAnswersCount++
                }
            }
        }
    }

    private fun handleCheckButtonState() {
        checkAnswerForCurrentFragment()
        updateUIAfterCheckingAnswer()
    }

    private fun handleNextButtonState() {
        navigateToNextQuestionOrSummary()
    }

    private fun checkAnswerForCurrentFragment() {
        val currentFragment = adapter.getFragment(binding.viewPager.currentItem)
        currentFragment?.let { fragment ->
            if (fragment is TaskActionsListener) {
                fragment.checkAnswer { result ->
                    viewModel.onCheckClicked(result.first, result.second)
                }
            }
```

```
        }
    }

    private fun updateUIAfterCheckingAnswer() {
        binding.checkButton.text = getString(R.string.next_button_text)
        binding.skipButton.visibility = View.GONE
        finishedItems.add(binding.viewPager.currentItem)
    }

    private fun navigateToNextQuestionOrSummary() {
        val nextItem = binding.viewPager.currentItem + 1
        if (nextItem < adapter.itemCount) {
            binding.viewPager.currentItem = nextItem
            updateProgress(nextItem)
        } else {
            navigateToQuizSummary()
        }
    }

    private fun handleIntents() {
        val courseId = intent.getStringExtra("courseId") ?: return
        val lessonId = intent.getStringExtra("lessonId") ?: return
        val courseType =
CourseType.valueOf(intent.getStringExtra("courseType").toString())
        viewModel.loadTasks(courseId, lessonId, courseType)
    }

    private fun updateProgress(position: Int) {
        binding.taskProgress.progress = position
        setDefaultState()
    }

    private fun setDefaultState() {
        binding.checkButton.text = getString(R.string.check_button_text)
        binding.skipButton.visibility = View.VISIBLE
    }

    private fun finishExercises() {
        val courseId = intent.getStringExtra("courseId") ?: return
        val lessonId = intent.getStringExtra("lessonId") ?: return

        viewModel.finishLesson(courseId, lessonId)

        val redirectIntent = Intent(Intent.ACTION_VIEW,
Uri.parse("maverkick://student/studentHomeFragment"))
        startActivity(redirectIntent)
        finish()
    }

    companion object {
        fun newIntent(context: Context, courseId: String, lessonId: String,
courseType: CourseType): Intent {
            return Intent(context, ExerciseActivity::class.java).apply {
                putExtra("courseId", courseId)
                putExtra("lessonId", lessonId)
                putExtra("courseType", courseType.name)
            }
        }
```

```
    }
}
```

Також треба пояснити яким чином ми можемо інтегрувати декілька різних типів завдань в одному місці:

Адаптер `TaskPagerAdapter` відповідає за відображення різних видів завдань для користувача в одному макеті. Він використовується для ViewPager2, що дозволяє легко прокручувати завдання вліво та вправо.

```
package com.maverkick.tasks.task

import android.util.SparseArray
import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentActivity
import androidx.viewpager2.adapter.FragmentStateAdapter

/**
 * Adapter class which is responsible for displaying the list of different
 * tasks to the user, also choosing which tasks are gonna be displayed and
 * how many of them we should display
 **/
class TaskPagerAdapter(fragmentActivity: FragmentActivity) :
FragmentStateAdapter(fragmentActivity) {
    private var tasks: List<Task> = emptyList()

    // SparseArray to keep track of the fragments in the ViewPager
    private val fragmentMap = SparseArray<Fragment>()

    /** Method that sets the list of tasks for this Exercise **/
    fun setTasks(tasks: List<Task>) {
        this.tasks = tasks
        notifyDataSetChanged()
    }

    override fun getItemCount() = tasks.size

    /** This is used for creation of different tasks for the same lesson**/
    override fun createFragment(position: Int): Fragment {
        // The creation logic is now delegated to the task itself
        val fragment = tasks[position].createFragment()

        // Save the fragment instance to the map
        fragmentMap.put(position, fragment)

        return fragment
    }

    /** Method to get a reference to the fragment at the given position **/
    fun getFragment(position: Int): Fragment? {
        return fragmentMap.get(position)
    }
}
```

Цей адаптер приймає список завдань `Task`, які можуть бути різних типів, таких як текстовий тест, заповнення пропусків, зіставлення тощо.

```kotlin
package com.maverkick.tasks.task

/**
 * Base class for tens or even hundreds of possible tasks that could be.
 * This is used as a foundation and for the ExerciseDialogFragment
initialization
 **/
abstract class Task(
    open val type: TaskType
) : TaskFactory


/**
 * Class that store all of the possible Task Classes
 **/
enum class TaskType {
    TEXT_QUIZ,
    FILL_IN_GAPS,
    MATCHING,
    TRUE_OR_FALSE
}
```

При створенні фрагментів для відображення кожного завдання, використовується механізм фабрики `TaskFactory`. Цей механізм дозволяє кожному типу завдання мати власний метод створення фрагменту, що спрощує додавання нових типів завдань та підтримку різних видів фрагментів.

```kotlin
package com.maverkick.tasks.task

import androidx.fragment.app.Fragment
/**
 * The TaskFactory interface provides a mechanism for creating specific Task
Fragments.
 *
 * Classes that represent different types of tasks should implement this
interface to
 * provide their own logic for creating the corresponding fragment. This allows
each
 * task type to encapsulate the logic of creating its associated fragment,
promoting
 * better separation of concerns and making the code easier to maintain and
extend.
 *
 * This follows the Factory Method design pattern, where an interface is used to
define
 * a method for creating an object, but the actual implementation of this method
is
 * deferred to implementing classes.
 */
interface TaskFactory {
    /**
     * Creates and returns a new Fragment instance associated with the current
task type.
     *
     * The specific type of the Fragment (e.g., TextQuizFragment,
```

```
FillInBlanksFragment, etc.)
     * and the data it contains will depend on the specific implementation in
each task class.
     *
     * @return a new Fragment instance.
     */
    fun createFragment(): Fragment
}
```

Клас `TaskPagerAdapter` використовує `SparseArray` для зберігання посилань на фрагменти, що відображаються у ViewPager. Це дозволяє швидко отримувати доступ до фрагментів для роботи з ними, наприклад, перевірки відповідей користувача або зберігання їх стану при переході між сторінками.

Завдяки цій архітектурі можливо легко додавати нові типи завдань та розширювати функціональність додатка без необхідності змінювати велику кількість коду.

## 3.3 Тестування

Ми запустимо наш додаток на смартфоні Samsung для того, щоб перевірити як він працює.

Крім того, ми попросили декількох близьких людей протестувати і отримали наступні негативні відгуки, це майже дослівні цитати виразів людей:

1. «Додаток не локалізований, тому дуже важко знайти і зрозуміти, якщо людина не володіє українською мовою. Ти мусиш додати щось, що зробить його мультинаціональним»
2. «Не вистачає можливості аутентифікації через Google, Facebook і тд. Я б відчувавав полегшення, якщо б це було додано»
3. Не вистачає медіа, типу зображень для того, щоб зробити текстові уроки дуже цікавими, хоча мені й подобається як виглядає сам текст і оці історії»

Якщл ви зацікавитеся і самі захочете спробувати цей додаток, то маю для вас посилання на Google Play (Рисунок 3.9 Посилання на сторінку в Google Play).

Рисунок 3.9 Посилання на сторінку в Google Play

Ми створили сторінку для нього в Google Play (Рисунок 3.10 Додаток в Google Play) і уже навіть декілька людей здається завантажили його, тому нам дуже важливо якщо ви також спробуєте його і дасте най вашу оцінку.

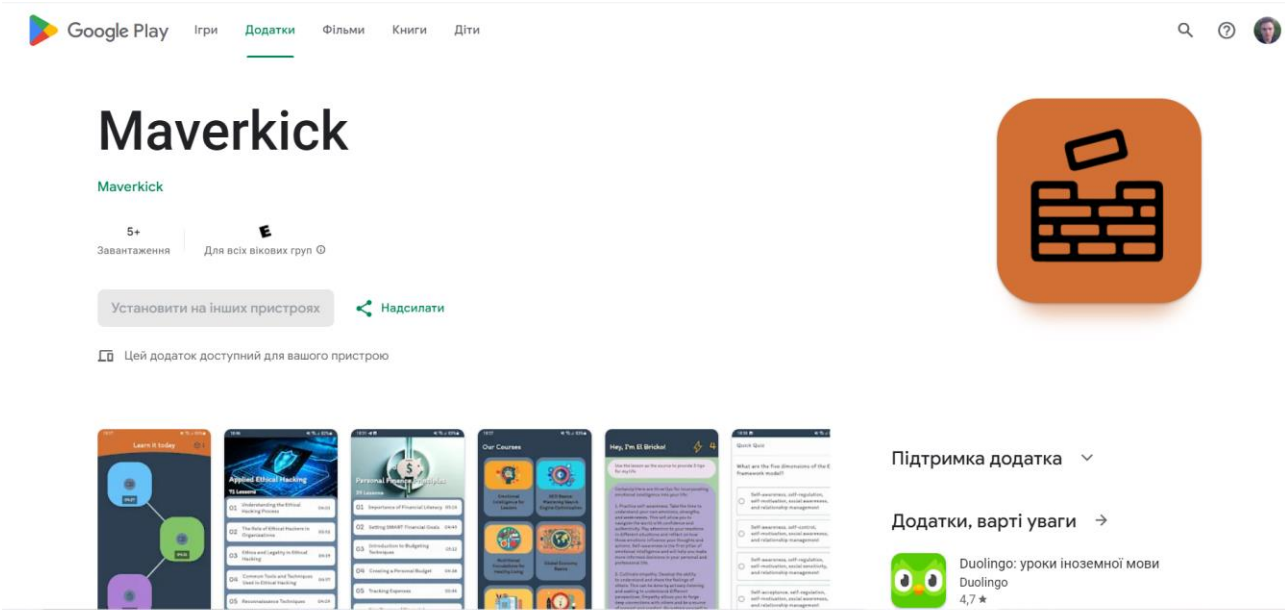


Рисунок 3.10 Додаток в Google Play

# 4. ВИСНОВКИ

Описавши всі частини нашого додатку для навчання, який використовує ваш вільний час для вивчення того, що вам подобається і імлементувавши його, ми можемо зробити один простий висновок – наскільки ми були праві в необхідності розвитку освіти і послуг в даному векторному напрямі покаже лише час. Наше рішення далеке від бездоганного, це лише MVP (мінальна версія продукту, яка забезпечує базовий функціонал).

Але вже наше MVP по своїй суті є тим, що дозволяє людині будь-якого віку, будь-якої професії, в будь-який момент дня приділити 5 хвилин для отримання інформації про тему, яка її цікавить і за певну кількість днів дізнатися щось нове, прояснити певні питання, які завжди хотіли, але не мали часу, або мали, але просто не могли визначити, де та яким чином знайти відповіді на ці питання.

У ході виконання роботи було виконано такі завдання:

1) Ми взяли ідею, яка зародилася в мене ще рік тому. Ідею, яка є максимально простою і полягає в кращому використанні концентрації людини в освітньому процесі, або як змусити людину захотіти вчитися.
2) Ми провели певний аналіз ринку освітніх послуг і сервісів суміжних до цього, які надають короткий контент, наприклад TikTok та YouTube. Ми отримали необхідну інформацію, для того, щоб використати її для формалізації нашого рішення.
3) Далі ми побудували інформаційну модель, описали необхідний фунціонал і базову структуру системи, яка буде інтегруватися в додаток на Android платформі.
4) Зробили короктий дезайн необхідних компонентів та екранів додатку в дизайнері Figma для того, щоб відштовхуватися від цього, при цьому ми робили певні зміни під час роботи, тому початковий дизайн не повністю збігатиметься із фінальною версією продукту на телефоні.
5) Було обрано інструменти реалізації, у вигляді мови програмування, фреймворку, бази даних, мови серверу.
6) Далі ми швидко написали необхідний код, у декілька ітерацій, постійно додаючи необхідні компоненти.
7) Провели функціональне тестування з використанням емулятора.
8) Залили додаток в Google Play і знову перевірили як це працює на справжньому телефоні.

Надалі планується виконати певну роботу, як над вдосконаленням існуючого коду, дизайну інтерфейсу, покращення роботи великих мовних моделей, додавання різного виду вправ, так і загальний підхід на комерціалізацію та монетизацію нашого рішення.

Нижче наведено пункти, над якими ми повинні попрацювати:

1) Покращення безпеки додатку і безпечної взаємодії сервера із ним.
2) Використання open-source великих мовних моделей по типу Llama 3 або Mistral, в якості заміни OpenAI для розмови з користувачем під час уроку та пояснення матеріалу чи відповідей на запитання.
3) Додавання можливості користувачам для генерації курсів за описом, наприклад "Я хочу, щоб ти пояснив мені основи фондового ринку", щоб користувач міг обирати налаштування моделі, в якому стилі повинен бути контент курсу (професійний, дружній, комедійний), яка кількість уроків (10,15,30) і тд.
4) Додавання функціоналу для викладачів та користувачів для створення власних відео-курсів. Це мусить бути простим, легким в використання та нагадувати підходи TikTok та YouTube, лише у форматі коротких відео-курсів на різні теми.
5) Комерціалізація проєкту за рахунок використання freemium бізнес-моделі із застосуванням безкоштовних та преміум аккаунтів із різним набором функціональних можливостей.
6) Підвищення рівня персоналізації та рекомендації контенту, використання алгоритмів, які дозволять використовувати шматки із різних курсів і поєднувати їх у великі навчальні програми, які можуть тривати місяці і охоплювати декілька різних курсів, для того, щоб поєднувати знання. Хоча дане завдання зараз на грані можливого через свою складність і обмеження кількості даних, необхідних для того, щоб досягти навіть базового рівня інтелекту для даних систем.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Ruben Kesherim. (2023, 24 лютого). *Average Human Attention Span (By Age, Gender & Race)*. Supportive Care ABA Therapy. https://www.supportivecareaba.com/statistics/average-attention-span
2. Anton Portianyi. (2023, 8 червня). *The Evolution of E-Learning Systems: A Brief History*. The Evolution of E-Learning Systems: A Brief History. https://academyocean.com/blog/post/history-of-e-learning-system
3. *Insights into the Latest EdTech Statistics: Shaping the Future of Education - Blue Tree Digital*. (б. д.-b). Blue Tree Digital. https://bluetree.digital/edtech-statistics/
4. *Education Technology (EdTech) Market Demand, Growth & Forecast by 2030*. (б. д.). Global Market Research and Industry Analysis. https://straitsresearch.com/report/education-technology-market
5. administrator. (2023, 8 червня). *The Evolution of E-Learning Systems: A Brief History*. The Evolution of E-Learning Systems: A Brief History. https://academyocean.com/blog/post/history-of-e-learning-system
6. UA, X. C. (2023, 20 липня). *Патерни онбордингу користувачів*. Medium. https://medium.com/@xd_community_ua/патерни-онбордингу-користувачів-666f72c0c42f
7. uxpeak. (2023, 11 грудня). *Top UI/UX Design Tips: How to Design a Great Bottom Mobile Navigation Bar, part 6*. Medium. https://medium.com/@uxpeak.com/top-ui-ux-design-tips-how-to-design-a-great-bottom-mobile-navigation-bar-part-6-97acd8b28453
8. *Kotlin vs. Java for Android development - LogRocket Blog*. (б. д.). LogRocket Blog. https://blog.logrocket.com/kotlin-vs-java-android-development/
9. *Kotlin vs. Flutter for Android development - LogRocket Blog*. (б. д.). LogRocket Blog. https://blog.logrocket.com/kotlin-vs-flutter-android-development/
10. *Kotlin vs. Java For Android Development Ultimate Comparison*. (б. д.). IT Outsourcing Services Company | Outsourcing Software and Web Development. https://lampa.dev/blog/kotlin-vs-java-for-android-development-ultimate-comparison
11. Technologies, E. (2023, 17 серпня). *Choosing the Right Tech for Android Development: Java, Kotlin, Flutter, and React Native*. Medium. https://etelligens.medium.com/choosing-the-right-tech-for-android-development-java-kotlin-flutter-and-react-native-d7cd51d3100f

12. *How Meta switched Android development to Kotlin*. (б. д.). Tech at Meta. https://tech.facebook.com/engineering/2022/10/kotlin-meta-migration-java/
13. Paul, S. (2019, 5 лютого). *Beginner's Guide to PostgreSQL*. Learn Data Science and AI Online | DataCamp. https://www.datacamp.com/tutorial/beginners-introduction-postgresql
14. *Firebase vs. Supabase: Choosing the right tool for your project - LogRocket Blog*. (б. д.-b). LogRocket Blog. https://blog.logrocket.com/firebase-vs-supabase-choosing-right-tool-project/
15. *Supabase vs MongoDB: A Feature-by-Feature Comparison*. (б. д.). Squash: On demand test environments for web apps and microservices. | On demand test environments for web apps and microservices. Save time and iterate faster with disposable virtual machines for each branch of code. https://www.squash.io/supabase-vs-mongodb-a-feature-by-feature-comparison/
16. Учасники проектів Вікімедіа. (2011, 13 вересня). *MongoDB — Вікіпедія*. Вікіпедія. https://uk.wikipedia.org/wiki/MongoDB
17. *Pricing*. (б. д.). OpenAI. https://openai.com/pricing
18. Kelly, W. (2024, 27 лютого). *GPT-3.5 vs. GPT-4: Biggest differences to consider | TechTarget*. Enterprise AI. https://www.techtarget.com/searchenterpriseai/tip/GPT-35-vs-GPT-4-Biggest-differences-to-consider
19. *Coolors - The super fast color palettes generator!* (б. д.). Coolors.co. https://coolors.co/
20. *MVVM (Model View ViewModel) Architecture Pattern in Android - GeeksforGeeks*. (б. д.). GeeksforGeeks. https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/
21. *Common modularization patterns  |  Android Developers*. (б. д.). Android Developers. https://developer.android.com/topic/modularization/patterns
22. *Quickstart — Flask Documentation (3.0.x)*. (б. д.). Welcome to Flask — Flask Documentation (3.0.x). https://flask.palletsprojects.com/en/3.0.x/quickstart/
23. *Retrofit*. (б. д.). Square Open Source. https://square.github.io/retrofit/

# ДОДАТОК А

Повний вихідний код доступний за посиланням:

[GitHub репозиторій з вихідним кодом додатку]

### А.1 Моделі даних

AiChatMessage.kt

```
/**
 * Represents a message in the AI chat.
 *
 * @property text The content of the chat message.
 * @property isUser Indicates whether the message was sent by the user or not.
If true, the message was sent by the user.
 *                 If false, the message was sent by the AI.
 */
data class AiChatMessage(
    val text: String,
    val isUser: Boolean
)
```

Course.kt

```
package com.maverkick.data.models

import java.util.Date

enum class CourseType {
    TEXT_PERSONALIZED, VIDEO, TEXT
}

/**
 * Abstract class representing a course.
 *
 * @property courseId       Unique identifier for the course.
 * @property courseName     Name of the course.
 * @property language       Language of the course content.
 * @property numberLessons  Total number of lessons in the course.
 * @property creationDate   Date of course creation.
 * @property type           Type of the course (text or video).
 * @property authorId       Unique identifier for the author of the course.
 */
abstract class Course(
    open val courseId: String,
    open val courseName: String,
    open val language: String,
    open val numberLessons: Int,
    open val creationDate: Date,
    open val type: CourseType,
    open val authorId: String
) {
    override fun equals(other: Any?): Boolean {
        if (this === other) return true
        if (other == null || this::class != other::class) return false
```

```kotlin
        other as Course

        if (courseId != other.courseId) return false
        if (courseName != other.courseName) return false
        if (language != other.language) return false
        if (numberLessons != other.numberLessons) return false
        if (creationDate != other.creationDate) return false
        if (type != other.type) return false
        if (authorId != other.authorId) return false

        return true
    }

    override fun hashCode(): Int {
        var result = courseId.hashCode()
        result = 31 * result + courseName.hashCode()
        result = 31 * result + language.hashCode()
        result = 31 * result + numberLessons
        result = 31 * result + creationDate.hashCode()
        result = 31 * result + type.hashCode()
        result = 31 * result + authorId.hashCode()
        return result
    }
}
```

## CourseStatistics.kt

```kotlin
package com.maverkick.data.models

/**
 * Data class to hold statistics for a specific course.
 *
 * @property courseId Unique identifier of the course.
 * @property numberOfEnrollments Total number of students that have enrolled in
the course.
 * @property numberOfCompletions Total number of students that have completed
the course.
 * @property numberOfDropouts Total number of students that have dropped out of
the course.
 * @property totalNumberOfRatings Total number of ratings given to the course.
 * @property sumOfRatings Sum of all ratings given to the course.
 */
data class CourseStatistics(
    val courseId: String,
    var courseName: String,
    var numberOfEnrollments: Int = 0,
    var numberOfCompletions: Int = 0,
    var numberOfDropouts: Int = 0,
    var totalNumberOfRatings: Int = 0,
    var sumOfRatings: Int = 0
) {
    // No-argument constructor required for Firestore
    constructor() : this("","", 0, 0, 0, 0, 0)

    /**
```

```
     * Calculates the completion rate of the course.
     * The completion rate is defined as the number of completions divided by
the number of enrollments.
     *
     * @return The completion rate, or 0 if there are no enrollments.
     */
    fun calculateCompletionRate(): Double = if (numberOfEnrollments > 0)
numberOfCompletions.toDouble() / numberOfEnrollments else 0.0

    /**
     * Calculates the dropout rate of the course.
     * The dropout rate is defined as the number of dropouts divided by the
number of enrollments.
     *
     * @return The dropout rate, or 0 if there are no enrollments.
     */
    fun calculateDropoutRate(): Double = if (numberOfEnrollments > 0)
numberOfDropouts.toDouble() / numberOfEnrollments else 0.0

    /**
     * Calculates the average rating of the course.
     * The average rating is defined as the sum of all ratings divided by the
total number of ratings.
     *
     * @return The average rating, or 0 if there are no ratings.
     */
    fun calculateAverageRating(): Double = if (totalNumberOfRatings > 0)
sumOfRatings.toDouble() / totalNumberOfRatings else 0.0

    fun toFirebaseCourseStatistics(): FirebaseCourseStatistics {
        return FirebaseCourseStatistics(
            courseName = this.courseName,
            numberOfEnrollments = this.numberOfEnrollments,
            numberOfCompletions = this.numberOfCompletions,
            numberOfDropouts = this.numberOfDropouts,
            totalNumberOfRatings = this.totalNumberOfRatings,
            sumOfRatings = this.sumOfRatings
        )
    }
}


data class FirebaseCourseStatistics @JvmOverloads constructor(
    var courseName: String = "",
    var numberOfEnrollments: Int = 0,
    var numberOfCompletions: Int = 0,
    var numberOfDropouts: Int = 0,
    var totalNumberOfRatings: Int = 0,
    var sumOfRatings: Int = 0
) {
    fun toCourseStatistics(courseId: String): CourseStatistics {
        return CourseStatistics(
            courseId,
            courseName,
            numberOfEnrollments,
            numberOfCompletions,
            numberOfDropouts,
            totalNumberOfRatings,
```

```
            sumOfRatings
        )
    }
}

enum class StatisticType {
    ENROLLMENTS,
    COMPLETION_RATE,
    DROPOUTS,
    AVERAGE_RATING
}
```

## Lesson.kt

```
package com.maverkick.data.models

/**
 * ILesson is a common abstract class for different types of lessons within the
application.
 * This abstract class helps unify the handling of various lesson types, such as
video and text lessons,
 * making it easier to create shared functionality around these types.
 *
 * @property lessonId - A unique identifier for the lesson.
 * @property courseId - The identifier of the course to which the lesson
belongs.
 * @property title - The title of the lesson.
 * @property duration - The duration of the lesson in seconds.
 *                    For video lessons, it represents the length of the video,
 *                    while for text lessons, it may represent an estimated
reading time.
 * @property lessonOrder - The order of the lesson within the course (e.g. 1, 2,
3, 4,...).
 */
abstract class Lesson(
    open val lessonId: String,
    open val courseId: String,
    open val title: String,
    open val duration: Int,
    open val lessonOrder: Int
) {
    fun isContentTheSame(other: Lesson): Boolean {
        return lessonId == other.lessonId &&
                courseId == other.courseId &&
                title == other.title &&
                duration == other.duration &&
                lessonOrder == other.lessonOrder
    }
}
```

## LessonFirebase.kt

```
package com.maverkick.data.models

open class LessonFirebase(
    val title: String = "",
```

```kotlin
    val duration: Int = 0,
    val lessonOrder: Int = 0
) {
    // Add a conversion function to be implemented in the child classes
    open fun toLesson(courseId: String, lessonId: String): Lesson {
        throw NotImplementedError("This method must be implemented in child
classes.")
    }
}
```

## Student.kt

```kotlin
package com.maverkick.data.models

/**
 * Class storage for the Student objects
 * @param studentId  the id of the student in the database
 * @param age age to make better recommendations and greater personalization
 * @param dailyStudyTimeMinutes  the number of minutes student would like study
daily
 * @param interests  the list of Tags(disciplines) in which student interested
 * @param bricksCollected shows the total number of lessons completed
 * @param enrolledCourses lists all the courses student enrolled in right now
 * @param enrolledGeneratedCourses lists all the text courses generated in which
it enrolled in
 * @param generatedTextCourses all the text courses generated by student
 * @param courseGenerationTries number of courses student could generate
 * @param finishedCourses lists all the courses, which have been finished
 **/
data class Student(
    val studentId: String,
    val age: Int,
    val dailyStudyTimeMinutes: Int,
    val interests: List<String>,
    var bricksCollected: Int = 0,
    var enrolledCourses: List<String> = emptyList(),
    var enrolledGeneratedCourses: List<String> = emptyList(),
    var generatedTextCourses: List<String> = emptyList(),
    var courseGenerationTries: Int = 0,
    var finishedCourses: List<String> = emptyList()
) {
    fun toFirebaseStudent(): FirebaseStudent {
        return FirebaseStudent(
            age,
            dailyStudyTimeMinutes,
            interests,
            bricksCollected,
            enrolledCourses,
            enrolledGeneratedCourses,
            generatedTextCourses,
            courseGenerationTries,
            finishedCourses
        )
    }
}
```

```kotlin
data class FirebaseStudent(
    val age: Int,
    val dailyStudyTimeMinutes: Int,
    val interests: List<String>,
    val bricksCollected: Int = 0,
    var enrolledCourses: List<String> = emptyList(),
    var enrolledGeneratedCourses: List<String> = emptyList(),
    var generatedTextCourses: List<String> = emptyList(),
    val courseGenerationTries: Int = 5,
    var finishedCourses: List<String> = emptyList()
) {
    fun toStudent(studentId: String): Student {
        return Student(
            studentId,
            age,
            dailyStudyTimeMinutes,
            interests,
            bricksCollected,
            enrolledCourses,
            enrolledGeneratedCourses,
            generatedTextCourses,
            courseGenerationTries,
            finishedCourses
        )
    }
}
```

## TextCourse.kt

```kotlin
package com.maverkick.data.models

import java.util.*

data class TextCourse(
    override val courseId: String,
    override val courseName: String,
    override val language: String,
    var poster: String?,
    override val numberLessons: Int,
    val tags: List<String>,
    override val creationDate: Date,
    override val authorId: String,
    val published: Boolean
) : Course(courseId, courseName, language, numberLessons, creationDate,
CourseType.TEXT, authorId)


data class FirebaseTextCourse @JvmOverloads constructor(
    val courseName: String = "",
    val language: String = "",
    val poster: String? = null,
    val lessonCount: Int = 0,
    val tags: List<String> = listOf(),
    val creationDate: Date = Date(),
    val authorId: String = "",
    val published: Boolean = false // Added published field with default value
```

```
as false
) {
    fun toCourse(courseId: String): TextCourse {
        return TextCourse(
            courseId,
            courseName,
            language,
            poster,
            lessonCount,
            tags,
            creationDate,
            authorId,
            published
        )
    }
}
```

## TextLesson.kt

```
package com.maverkick.data.models

/**
 * Class storage for the TextLesson objects.
 * It's a piece of content, which is read by user to learn.
 * @param lessonId - id of the lesson
 * @param courseId - id of the course
 * @param title - title of the text-lesson
 * @param duration - approximate time average user could read it in
 * @param content - the main text content of the lesson
 * @param lessonOrder - the order of the lesson in the course (1,2,3,4...)
 **/
data class TextLesson(
    override val lessonId: String = "",
    override val courseId: String = "",
    override val title: String = "",
    override val duration: Int = 0,
    override val lessonOrder: Int = 0,
    val content: String = ""
) : Lesson(lessonId, courseId, title, duration, lessonOrder)

/**
 * The same TextLesson class, but without lessonId and courseId, specifically
for interacting with Firebase
 **/
class TextLessonFirebase(
    title: String = "",
    val content: String = "",
    duration: Int = 0,
    lessonOrder: Int = 0
) : LessonFirebase(title, duration, lessonOrder) {
    override fun toLesson(courseId: String, lessonId: String): TextLesson {
        return TextLesson(
            lessonId,
            courseId,
            title,
            duration,
            lessonOrder,
```

```
            content
        )
    }
}
```

## User.kt

```kotlin
package com.maverkick.data.models

/**
 * Data Class for User to store it's locally in the database
 * @param userId - id of the user
 * @param username - username for the user, which identify it among the others
and displayed in the app
 * @param email - the way for user registration and login
 * @param profilePicture - the picture of the profile for the user, by default
we show some template picture
 **/

data class User(
    var userId: String,
    val username: String,
    val email: String,
    var profilePicture: String? = null
){
    // No-arg constructor for Firestore
    constructor() : this("", "", "", null)
}
```

## VideoCourse.kt

```kotlin
package com.maverkick.data.models

import kotlinx.serialization.SerialName
import kotlinx.serialization.Serializable
import java.util.*

/**
 * Class storage for the VideoCourse objects
 * @param courseId - unique id for the course
 * @param courseName- name that defines the course, shouldn't be unique
 * @param authorId - id of the creator of that course
 * @param language - the language of the course
 * @param poster - image/poster for the course to make it appealing and clear
 * @param numberLessons - number of lessons for the course
 * @param tags - tags describing 5 things on which course is concentrating
 * @param creationDate - date of the course creation
 * @param published - is this course available for the students
 **/
data class VideoCourse(
    override val courseId: String,
    override val courseName: String,
    override val authorId: String,
    override val language: String,
    var poster: String?,
    override val numberLessons: Int,
```

```
    val tags: List<String>,
    override val creationDate: Date,
    val published: Boolean
) : Course(courseId, courseName, language, numberLessons, creationDate,
CourseType.VIDEO,authorId){

    fun toFirebaseCourse(): FirebaseVideoCourse {
        return FirebaseVideoCourse(
            courseName = this.courseName,
            authorId = this.authorId,
            language = this.language,
            poster = this.poster ?: "",
            lessonCount = this.numberLessons,
            tags = this.tags,
            creationDate = this.creationDate,
            published = this.published
        )
    }
}

/**
 * The same VideoCourse class, but without courseId, specifically for adding to
Firebase
 */
data class FirebaseVideoCourse @JvmOverloads constructor(
    val courseName: String = "",
    val authorId: String = "",
    val language: String = "",
    val poster: String? = null,
    val lessonCount: Int = 0,
    val tags: List<String> = listOf(),
    val creationDate: Date = Date(),
    val published: Boolean = false
) {
    fun toCourse(courseId: String): VideoCourse {
        return VideoCourse(
            courseId,
            courseName,
            authorId,
            language,
            poster,
            lessonCount,
            tags,
            creationDate,
            published
        )
    }
}

/** Course class representation in the Algolia index, used for the courses
search*/
@Serializable
data class SearchCourseHit(
    @SerialName("objectID")
    val objectId: String, // this is the Algolia objectID, which is courseId
    val courseName: String,
    val language: String,
    val poster: String?,
```

```
    val tags: List<String>
)
```

## VideoLesson.kt

```
package com.maverkick.data.models

import kotlinx.serialization.SerialName
import kotlinx.serialization.Serializable
import java.util.*

/**
 * Class storage for the VideoCourse objects
 * @param courseId - unique id for the course
 * @param courseName- name that defines the course, shouldn't be unique
 * @param authorId - id of the creator of that course
 * @param language - the language of the course
 * @param poster - image/poster for the course to make it appealing and clear
 * @param numberLessons - number of lessons for the course
 * @param tags - tags describing 5 things on which course is concentrating
 * @param creationDate - date of the course creation
 * @param published - is this course available for the students
 **/
data class VideoCourse(
    override val courseId: String,
    override val courseName: String,
    override val authorId: String,
    override val language: String,
    var poster: String?,
    override val numberLessons: Int,
    val tags: List<String>,
    override val creationDate: Date,
    val published: Boolean
) : Course(courseId, courseName, language, numberLessons, creationDate,
CourseType.VIDEO,authorId){

    fun toFirebaseCourse(): FirebaseVideoCourse {
        return FirebaseVideoCourse(
            courseName = this.courseName,
            authorId = this.authorId,
            language = this.language,
            poster = this.poster ?: "",
            lessonCount = this.numberLessons,
            tags = this.tags,
            creationDate = this.creationDate,
            published = this.published
        )
    }
}

/**
 * The same VideoCourse class, but without courseId, specifically for adding to
Firebase
 */
data class FirebaseVideoCourse @JvmOverloads constructor(
    val courseName: String = "",
```

```
    val authorId: String = "",
    val language: String = "",
    val poster: String? = null,
    val lessonCount: Int = 0,
    val tags: List<String> = listOf(),
    val creationDate: Date = Date(),
    val published: Boolean = false
) {
    fun toCourse(courseId: String): VideoCourse {
        return VideoCourse(
            courseId,
            courseName,
            authorId,
            language,
            poster,
            lessonCount,
            tags,
            creationDate,
            published
        )
    }
}

/** Course class representation in the Algolia index, used for the courses
search*/
@Serializable
data class SearchCourseHit(
    @SerialName("objectID")
    val objectId: String, // this is the Algolia objectID, which is courseId
    val courseName: String,
    val language: String,
    val poster: String?,
    val tags: List<String>
)
```

## А.2 Денний навчальний план

```
package com.maverkick.data.repositories

import com.google.firebase.firestore.DocumentSnapshot
import com.google.firebase.firestore.FieldValue
import com.google.firebase.firestore.QuerySnapshot
import com.maverkick.data.IDatabaseService
import com.maverkick.data.models.*
import kotlinx.coroutines.tasks.await
import java.time.LocalDate
import java.util.*
import javax.inject.Inject
import kotlin.coroutines.resume
import kotlin.coroutines.resumeWithException
import kotlin.coroutines.suspendCoroutine

/**
 * Class responsible for creation and management of the Daily Learning Plan
 * for the particular student
 * @param databaseService - database to which we're connecting
 **/
```

```kotlin
class DailyLearningPlanRepository @Inject constructor(private val
databaseService: IDatabaseService){

    /** Check if learning plan exists **/
    suspend fun getDailyLearningPlanForStudent(studentId: String,
dailyStudyTimeMinutes: Int): DailyLearningPlan {
        val existingPlan = getDailyLearningPlan(studentId)
        if (existingPlan != null && existingPlan.lessons.isNotEmpty()) {
            return existingPlan
        }

        // Try to fetch lessons and create a new plan
        val todayPairs = getTodayLessons(studentId, dailyStudyTimeMinutes * 60)
        if (todayPairs.isNotEmpty()) {
            val newPlan = createDailyLearningPlan(studentId, todayPairs)
            storeDailyLearningPlan(newPlan)
            return newPlan
        }

        throw Exception("Failed to generate a daily learning plan")
    }

    /** Create the daily learning plan object **/
    private fun createDailyLearningPlan(studentId: String, lessons:
List<Lesson>): DailyLearningPlan {
        val totalDuration = lessons.sumOf{it.duration}
        return DailyLearningPlan(studentId, getCurrentDate(), lessons,
totalDuration)
    }

    /** Store the daily plan in the database **/
    private fun storeDailyLearningPlan(plan: DailyLearningPlan) {
        databaseService.db.collection("dailyLearningPlans")
            .document("${plan.studentId}_${plan.date}").set(plan)
            .addOnSuccessListener {}
            .addOnFailureListener {}
    }

    /** Get the daily plan from the database **/
    private suspend fun getDailyLearningPlan(studentId: String):
DailyLearningPlan? {
        val date = getCurrentDate()
        val docRef = databaseService.db.collection("dailyLearningPlans")
            .document("${studentId}_$date")
        val doc = suspendCoroutine { continuation ->
            docRef.get()
                .addOnSuccessListener { document ->
                    if (document != null && document.exists()) {
                        continuation.resumeWith(Result.success(document))
                    } else {
                        continuation.resumeWith(Result.success(null))
                    }
                }
                .addOnFailureListener { exception ->
                    continuation.resumeWith(Result.failure(exception))
                }
        }
        return convertToDailyLearningPlan(doc)
```

```
    }

    /*** A function that update daily learning plan on Course Enrollment **/
    suspend fun updateOnCourseEnrollment(studentId: String, courseId: String,
courseType: CourseType, dailyLearningTimeMinutes: Int) {
        val todayPlan = getDailyLearningPlan(studentId) ?: return

        // If today's learning plan is completed, exit early
        if (todayPlan.status == DailyLearningPlanStatus.COMPLETED) return

        // Calculate the remaining time in seconds
        val remainedTimeSeconds = dailyLearningTimeMinutes * 60 -
todayPlan.totalDuration

        // If there's no remaining time or only a minimal gap, exit early
        if (remainedTimeSeconds <= 120) return

        // Fetch lessons that fit within the remaining time for the specified
course
        val lessonsToAdd = fetchLessonsThatFitsTime(courseId, studentId,
courseType, remainedTimeSeconds)

        // If there are no lessons to add, exit early
        if (lessonsToAdd.isEmpty()) return

        // Add the lessons to the plan and update the duration
        val updatedPlan = todayPlan.copy(
            lessons = todayPlan.lessons + lessonsToAdd,
            totalDuration = todayPlan.totalDuration + lessonsToAdd.sumOf {
it.duration }
        )

        storeDailyLearningPlan(updatedPlan)
    }

    /** A function that update daily learning plan on Course drop **/
    suspend fun updateOnCourseDrop(studentId: String, courseId: String,
dailyLearningTimeSeconds: Int) {

        val todayPlan = getDailyLearningPlan(studentId) ?: return
        if (todayPlan.status == DailyLearningPlanStatus.COMPLETED ||
todayPlan.lessons.none { it.courseId == courseId }) { return }

        val lessonsToRemove = todayPlan.lessons.filter { it.courseId == courseId
}
        val removedDuration = lessonsToRemove.sumOf { it.duration }
        val remainingTimeSeconds = dailyLearningTimeSeconds -
(todayPlan.totalDuration - removedDuration)

        if (remainingTimeSeconds <= 120) { return }

        val additionalLessons = fillRemainingTimeWithLessons(studentId,
remainingTimeSeconds)

        val updatedLessons = todayPlan.lessons - lessonsToRemove +
additionalLessons

        val updatedDuration = todayPlan.totalDuration - removedDuration +
```

```
additionalLessons.sumOf { it.duration }
        val removedLessonIds = lessonsToRemove.map { it.lessonId }.toSet()
        val updatedCompletedLessons = todayPlan.completedLessons.filterNot { it
in removedLessonIds }

        val completedLessonIds = todayPlan.completedLessons
        val lessonsRemovedFromCompleted =
removedLessonIds.intersect(completedLessonIds).size
        val updatedProgress = todayPlan.progress - lessonsRemovedFromCompleted

        val updatedPlan = todayPlan.copy(
            lessons = updatedLessons,
            totalDuration = updatedDuration,
            progress = updatedProgress,
            completedLessons = updatedCompletedLessons
        )
        storeDailyLearningPlan(updatedPlan)
    }


    /** A function that checks if a lesson is completed **/
    suspend fun isLessonCompleted(dailyLearningPlanId: String, lessonId:
String): Boolean {
        val docRef =

databaseService.db.collection("dailyLearningPlans").document(dailyLearningPlanId
)

        val doc = docRef.get().await()

        val completedLessons = doc.get("completedLessons") as? List<*>
        return completedLessons?.contains(lessonId) ?: false
    }

    /** A function that adds a lesson to the list of completed lessons and
increment progress by 1**/
    suspend fun completeLessonAndUpdateProgress(dailyPlanId: String, lessonId:
String) {
        val docRef =
databaseService.db.collection("dailyLearningPlans").document(dailyPlanId)

        databaseService.db.runBatch { batch ->
            // Add the lesson to the completed lessons
            batch.update(docRef, "completedLessons",
FieldValue.arrayUnion(lessonId))
            // Increment the progress
            batch.update(docRef, "progress", FieldValue.increment(1))
        }.await()
    }

    /** Get the list of today's lessons for the student **/
    private suspend fun getTodayLessons(studentId: String,
dailyStudyTimeSeconds: Int): List<Lesson> {
        // Fetch the active courses for the student
        val courseDocs = fetchActiveCourses(studentId)

        val lessonQueues = mutableListOf<Queue<Lesson>>()
```

```
        // Fetch the lessons for each active course
        for (doc in courseDocs.documents) {
            val courseId = doc.getString("courseId") ?: continue
            val courseType = CourseType.valueOf(doc.getString("courseType") ?:
continue)
            val lessons = fetchLessonsThatFitsTime(courseId, studentId,
courseType, dailyStudyTimeSeconds)
            lessonQueues.add(LinkedList(lessons))
        }

        // Round-robin through the lesson queues
        val todayLessons = mutableListOf<Lesson>()
        var totalLessonTime = 0
        var i = 0

        while (totalLessonTime < dailyStudyTimeSeconds && lessonQueues.any {
it.isNotEmpty() }) {
            val queue = lessonQueues[i % lessonQueues.size]
            if (queue.isNotEmpty()) {
                val lesson = queue.peek()
                val lessonTime = lesson!!.duration
                // Add the lesson to the list if there is enough remaining study
time
                if (totalLessonTime + lessonTime > dailyStudyTimeSeconds) {
                    break // If the next lesson doesn't fit, stop adding lessons
                }
                todayLessons.add(lesson)
                totalLessonTime += lessonTime
                queue.poll() // now that we're sure we're using this lesson,
remove it from the queue
            }
            i++
        }
        return todayLessons
    }

    /** For a given course and student, find the number of the lesson student
finished in that course **/
    private suspend fun getLastWatchedLessonOrder(courseId: String, studentId:
String): Int {
        // Fetch the progress record for the student-course pair
        val progressDoc = suspendCoroutine<DocumentSnapshot> { continuation ->
            databaseService.db.collection("studentCourseProgress")
                .document("${studentId}_$courseId").get()
                .addOnSuccessListener { documentSnapshot ->
continuation.resume(documentSnapshot) }
                .addOnFailureListener { exception ->
continuation.resumeWithException(exception) }
        }
        return progressDoc.getLong("lastCompletedLesson")?.toInt() ?: 0
    }

    /** This function fetches lessons for each active course and fills the
remaining time with those lessons in a round-robin manner **/
    private suspend fun fillRemainingTimeWithLessons(studentId: String,
remainingTimeSeconds: Int): List<Lesson> {
        val lessonQueues = mutableListOf<Queue<Lesson>>()
```

```
        // Fetch the active courses for the student
        val courseDocs = fetchActiveCourses(studentId)

        // Fetch the lessons for each active course, considering the remaining
time
        for (doc in courseDocs.documents) {
            val courseId = doc.getString("courseId") ?: continue
            val courseType = CourseType.valueOf(doc.getString("courseType") ?:
continue)
            val lessons = fetchLessonsThatFitsTime(courseId, studentId,
courseType, remainingTimeSeconds)
            lessonQueues.add(LinkedList(lessons))
        }

        val filledLessons = mutableListOf<Lesson>()
        var totalLessonTime = 0
        var i = 0

        // Round-robin through the lesson queues
        while (totalLessonTime < remainingTimeSeconds && lessonQueues.any {
it.isNotEmpty() }) {
            val queue = lessonQueues[i % lessonQueues.size]
            if (queue.isNotEmpty()) {
                val lesson = queue.peek()
                val lessonTime = lesson!!.duration

                // If the next lesson doesn't fit, stop adding lessons
                if (totalLessonTime + lessonTime > remainingTimeSeconds) {
                    break
                }

                filledLessons.add(lesson)
                totalLessonTime += lessonTime
                queue.poll()
            }
            i++
        }

        return filledLessons
    }

    private suspend fun fetchLessonsThatFitsTime(courseId: String, studentId:
String, courseType: CourseType, remainingStudyTimeSeconds: Int): List<Lesson> {
        return when (courseType) {
            CourseType.TEXT_PERSONALIZED -> getNewTextCourseLessons(courseId,
studentId, remainingStudyTimeSeconds, "generatedCourses")
            CourseType.TEXT -> getNewTextCourseLessons(courseId, studentId,
remainingStudyTimeSeconds, "courses")
            CourseType.VIDEO -> getNewVideoCourseLessons(courseId, studentId,
remainingStudyTimeSeconds, "courses")
        }
    }

    private suspend fun getNewTextCourseLessons(courseId: String, studentId:
String, remainingStudyTimeSeconds: Int, collectionName: String):
List<TextLesson> {
        // Adjust your getNewCourseLessonsTemplate function to accept
collectionName as a parameter and use it in the query
```

```
        return getNewCourseLessonsTemplate(courseId, studentId,
remainingStudyTimeSeconds, collectionName, TextLessonFirebase::class.java)
    }

    private suspend fun getNewVideoCourseLessons(courseId: String, studentId:
String, remainingStudyTimeSeconds: Int, collectionName: String):
List<VideoLesson> {
        // Adjust your getNewCourseLessonsTemplate function to accept
collectionName as a parameter and use it in the query
        return getNewCourseLessonsTemplate(courseId, studentId,
remainingStudyTimeSeconds, collectionName, VideoLessonFirebase::class.java)
    }

    private suspend fun <T : Lesson, F : LessonFirebase>
getNewCourseLessonsTemplate(
        courseId: String, studentId: String, desiredTimeSeconds: Int,
collectionName: String, lessonFirebaseClass: Class<F>): List<T> {

        if (desiredTimeSeconds <= 120) {
            return emptyList()
        }

        val lastWatchedLessonOrder = getLastWatchedLessonOrder(courseId,
studentId)
        val newLessons = mutableListOf<T>()
        var totalLessonTime = 0
        var lessonOffset = lastWatchedLessonOrder + 1

        while (totalLessonTime < desiredTimeSeconds) {
            val lessonRef =
databaseService.db.collection(collectionName).document(courseId)
                .collection("lessons")
                .orderBy("lessonOrder")
                .startAt(lessonOffset)
                .limit(1)

            val lesson = lessonRef.get().await().documents.mapNotNull { doc ->
                doc.toObject(lessonFirebaseClass)?.toLesson(courseId, doc.id)
as? T
            }.firstOrNull()

            if (lesson != null && totalLessonTime + lesson.duration <=
desiredTimeSeconds) {
                newLessons.add(lesson)
                totalLessonTime += lesson.duration
                lessonOffset++
            } else {
                break
            }
        }
        return newLessons
    }

    /** Function that fetches active courses for a student from a database **/
    private suspend fun fetchActiveCourses(studentId: String): QuerySnapshot {
        return try {
            databaseService.db.collection("studentCourses")
                .whereEqualTo("studentId", studentId)
```

```
                .whereEqualTo("active", true)
                .get()
                .await()
        } catch (exception: Exception) {
            throw exception
        }
    }

    /** Convert document to the DailyLearningPlan Object **/
    private fun convertToDailyLearningPlan(doc: DocumentSnapshot?):
DailyLearningPlan? {
        val studentId = doc?.getString("studentId") ?: return null
        val date = doc.getString("date")
        val totalDuration = doc.getLong("totalDuration")?.toInt() ?: 0
        val progress = doc.getLong("progress")?.toInt() ?: 0
        val statusString = doc.getString("status") ?: "PLANNED"
        val status = DailyLearningPlanStatus.valueOf(statusString)

        // Get the completed lessons as a generic list and cast it if necessary
        val completedLessons = doc.get("completedLessons") as? List<String> ?:
emptyList()

        val lessonMaps = doc.get("lessons") as? List<Map<String, Any>> ?:
emptyList()
        val lessons = lessonMaps.mapNotNull { convertToLesson(it) }

        return DailyLearningPlan(studentId, date, lessons, totalDuration,
progress, status, completedLessons)
    }

    /** Convert lesson map to the particular Lesson object **/
    private fun convertToLesson(lessonMap: Map<String, Any>): Lesson? {
        val courseId = lessonMap["courseId"] as? String ?: ""
        val lessonId = lessonMap["lessonId"] as? String ?: ""
        val title = lessonMap["title"] as? String ?: ""
        val duration = (lessonMap["duration"] as? Long ?: 0L).toInt()
        val lessonOrder = (lessonMap["lessonOrder"] as? Long ?: 0L).toInt()

        return if ("content" in lessonMap) {
            val content = lessonMap["content"] as? String ?: ""
            TextLesson(lessonId, courseId, title, duration, lessonOrder,
content)
        } else if ("transcription" in lessonMap) {
            val videoUrl = lessonMap["videoUrl"] as? String ?: ""
            val transcription = lessonMap["transcription"] as? String ?: ""
            val creationDate = lessonMap["creationDate"] as? Date ?: Date()
            VideoLesson(lessonId, courseId, title, duration, lessonOrder,
videoUrl, transcription, creationDate)
        } else {
            null
        }
    }

    /** Get the current date in format YYYY-MM-DD as a string **/
    fun getCurrentDate(): String {
        return LocalDate.now().toString()
    }
}
```