

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

Сергій СТИРЕНКО

(підпис)

“__” _____ 2022 р.

**Дипломний проєкт
на здобуття ступеня бакалавра
за освітньо-професійною програмою “Інженерія програмного
забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”**

на тему: Мобільна система обміну освітнім контентом

Виконав : студент 4 курсу, групи ІІ-83

(шифр групи)

Чугай Ярослав Вікторович

(прізвище, ім’я, по батькові)

(підпис)

Керівник доцент, д.т.н., Сергієнко А. М.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Консультант _____

(назва розділу)

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Рецензент _____

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____

(підпис)

Київ – 2022 р.

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ

Завідувач кафедри

Сергій СТИРЕНКО

(підпис)

“ ____ ” _____ 2022 р.

ЗАВДАННЯ

на бакалаврський дипломний проект студента

Чугая Ярослава Вікторовича

1. Тема проекту Мобільна система обміну освітнім контентом
керівник проекту Сергієнко Анатолій Михайлович, доцент, д.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 11 травня 2022 року №1139-с
2. Термін здачі студентом закінченого проекту 11 червня 2022 р.
3. Вихідні дані до проекту технічна документація, теоретичні дані.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Розділ 1. Огляд існуючих рішень.
Розділ 2. Проектування системи.
Розділ 3. Реалізація системи.
Розділ 4. Огляд розробленої системи

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема, алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В. П.		

7. Дата видачі завдання «30» серпня 2021 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	<i>Затвердження теми проєкту</i>	<i>10.12.2021-15.12.2021</i>	
2.	<i>Вивчення та аналіз завдання</i>	<i>15.12.2021-15.04.2022</i>	
3.	<i>Розробка архітектури та загальної структури системи</i>	<i>15.04.2022-22.04.2022</i>	
4.	<i>Розробка структур окремих підсистем</i>	<i>22.04.2022-29.04.2022</i>	
5.	<i>Програмна реалізація системи</i>	<i>29.04.2022-05.05.2022</i>	
6.	<i>Оформлення пояснювальної записки</i>	<i>05.05.2022-20.05.2022</i>	
7.	<i>Захист програмного продукту</i>		
8.	<i>Передзахист</i>		
9.	<i>Захист</i>	<i>24.06.2022</i>	

Студент-дипломник _____ Ярослав ЧУГАЙ
(підпис)

Керівник проєкту _____ Анатолій СЕРГІЄНКО
(підпис)

АНОТАЦІЯ

У даному дипломному проєкті було розглянуто існуючі рішення мобільних систем обміну освітнього контенту, виявлено їхні переваги та недоліки з користувачької точки зору. На базі цього аналізу був розроблений функціонал додатку. Були розглянуті існуючі типи архітектур, виявлені їхні переваги та недоліки та згідно них обрана найбільш відповідна для розроблюваної системи. Також спроектована база даних. В результат роботи над проєктом була розроблена система, яка повністю виконує поставлені перед нею задачі.

ANNOTATION

In this diploma project, the existing solutions of mobile systems of educational content exchange were considered, and their advantages and disadvantages from the user point of view were revealed. Based on this analysis, the functionality of the application was developed. The existing types of architectures were considered, their advantages and disadvantages were identified, and according to them, the one most suitable for the developed system was selected. A database has also been designed. As a result of work on the project, a system, that fully fulfills the set tasks, was developed.

справки	Формат	Значення			Найменування	Кіл. листів	№ екземпля	Додаток
					Документація загальна			
					Знову розроблена			
	A4	ІАЛЦ.467200.002 ТЗ			Мобільна система обміну освітнім контентом	3		
					Технічне завдання			
	A4	ІАЛЦ.467200.003 ПЗ			Мобільна система обміну освітнім контентом	69		
					Пояснювальна записка			
	A4	ІАЛЦ.467200.004 Д1			Мобільна система обміну освітнім контентом	1		
					Структурна схема системи			
	A4	ІАЛЦ.4672008.005 Д2			Мобільна система обміну освітнім контентом	1		
					Функціональна схема			
	A4	ІАЛЦ.467200.006 Д3			Мобільна система обміну освітнім контентом	1		
					Алгоритм дій програмного забезпечення			
					ІАЛЦ.467200.001 ОА			
Зм.	Лист	№ докум.	Підп.	Дата				
Розроб.		Чугай Я.В.			Мобільна система обміну освітнім контентом Опис альбому	Літ.	Аркуш	Аркушів
Перев.		Сергієнко А.М.					1	1
						НТУУ “КПІ” ФІОТ ІП-83		

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Мобільна система обміну освітнім контентом»

ЗМІСТ

НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ	2
ПІДСТАВИ ДЛЯ РОЗРОБКИ	2
МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ.....	2
ТЕХНІЧНІ ВИМОГИ.....	2
ЕТАПИ РОЗРОБКИ	3

					<i>ІАЛЦ.467200.002 ТЗ</i>				
<i>Зм.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>					
<i>Розробив</i>		Чугай Я. В.			Мобільна система обміну освітнім контентом Технічне завдання				
<i>Перевірив</i>		Сергієнко А. М.							
<i>Н. Контр.</i>		Сімоненко В. П.							
<i>Затвердив</i>									
						<i>Літ.</i>	<i>Аркуш</i>	<i>Аркушів</i>	
							1	3	
					НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-83				

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку мобільної системи обміну освітнім контентом.

Областю застосування цієї системи є галузь онлайн навчання.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки системи - завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка мобільної системи обміну освітнім контентом із використанням відповідних технологій та архітектури, що дозволить використовувати систему згідно встановлених цілей.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є документації з певних ресурсів, які зберігають великі обсяги даних в мережі Інтернет , публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Надавати користувачам можливість переглядати навчальні матеріали
- Надавати можливість виконувати практичні завдання

					<i>ІАЛЦ.467200.002 ТЗ</i>	Аркуш
						2
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

- Надавати можливість відслідковувати прогрес навчання

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.

5.3. Вимоги до апаратної частини

- ЦП не менше ніж Intel® Core i5-2100T.
- ROM не менше ніж 64 ГБ.
- RAM не менше ніж 4 ГБ

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.01.2022-15.02.2022
Вивчення та аналіз завдання	15.02.2022-23.04.2022
Розробка архітектури та загальної структури системи	24.04.2022-07.05.2022
Програмна реалізація системи	08.05.2022-12.05.2022
Виправлення помилок	13.05.2022-16.05.2022
Оформлення пояснювальної записки	17.05.2022

					ІАЛЦ.467200.002 ТЗ	Аркуш
						3
Изм.	Аркуш.	№ докум.	Підпис	Дата		

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Мобільна система обміну освітнім контентом»

Київ – 2022

ЗМІСТ

ВСТУП.....	4
РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ	5
1.1 Класифікація мобільних систем обміну освітнім контентом	5
1.1.1 Цільові сайти навчання.....	5
1.1.2 Традиційна система управління навчанням	5
1.1.3 Система управління навчанням з відкритим кодом	6
1.1.4 Нетрадиційні системи навчання	6
1.2 Функціональність мобільних систем обміну освітнім контентом	6
1.2.1 Створення організації, курсу.....	6
1.2.2 Публікація лекцій	6
1.2.3 Реєстрація на курс	7
1.2.4 Пошук лекцій	7
1.2.5 Об'єднання лекцій в курс та спеціалізації	7
1.2.6 Тести та підсумкові роботи	7
1.2.7 Онлайн щоденник.....	8
1.2.8 Особистий кабінет.....	8
1.3 Аналіз представлених на ринку рішень	8
1.3.1 Coursera.....	8
1.3.2 Google classroom	13
ВИСНОВОК ДО РОЗДІЛУ 1	15
РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ.....	16
2.1 Архітектура проекту	16

					<i>ІАЛЦ.467200.002 ТЗ</i>						
<i>Зм.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>							
<i>Розробив</i>	Чугай Я. В.				Мобільна система обміну освітнім контентом Зміст	<i>Літ.</i>	<i>Аркуш</i>	<i>Аркушів</i>			
<i>Перевірів</i>	Сергієнко А. М.										
<i>Н. Контр.</i>	Сімоненко В. П.					НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІІІ-83					
<i>Затвердив</i>											
					<i>ІАЛЦ.467200.002 ТЗ</i>						
<i>Ізм.</i>	<i>Аркуш.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>							
					Аркуш 2						

2.2 Діаграма діяльності.....	17
2.3 Діаграма компонентів	21
2.4 Пакетна діаграма	22
2.5 Архітектурні підходи	23
2.5.1 Системи на основі мови програмування.....	23
2.5.2 Основна програма і підпрограми.....	24
2.5.3 Системи на основі репозиторіїв.....	26
2.5.4 Багатошарові системи	28
2.6 Види архітектур	29
2.6.1 Монолітна архітектура.....	29
2.6.2 Сервіс-орієнтована архітектура	31
2.6.3 Мікросервісна архітектура	32
2.7 Вибір оптимальної архітектури для мобільної системи обміну освітнім контентом	33
ВИСНОВОК ДО РОЗДІЛУ 2	35
РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ.....	36
3.1 Використаний технологічний стек	36
3.2 Проектування бази даних	37
3.2.1 Модель предметної області	37
3.2.2 Даталогічна модель	38
3.3 Реалізація серверної частини додатку.....	40
3.4 Реалізація кінцевих точок.....	46
3.4 Реалізація клієнтської частини додатку	50
ВИСНОВОК ДО РОЗДІЛУ 3	52
РОЗДІЛ 4. ОГЛЯД РОЗРОБЛЕНОЇ СИСТЕМИ	53
ВИСНОВОК ДО РОЗДІЛУ 4	64
ВИСНОВКИ.....	65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	66

					<i>ІАЛЦ.467200.002 ТЗ</i>	Аркуш
						3
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

ВСТУП

Кожен повинен розвиватися все життя, оскільки сучасний світ прогресує надзвичайно швидко. Однак, навіть при бажанні розібратись в тому чи іншому предметі буває складно по тій чи іншій причині. Це може бути як і недоступність знань в офлайн житті, так і відсутність людей, які могли б пояснити предмет.

І тут на допомогу приходять мобільні системи обміну освітнім контентом, де експерти в своїй галузі розміщують матеріали, в які вкладені їхній багаторічний досвід та знання. Будь-хто може переглянути ці матеріали і розібратися в темі.

Але, в існуючих таких системах є досить багато недоліків, які можуть завадити ефективному навчанню користувача. Тому, метою цього проекту є розробка системи, яка не має цих недоліків, але і зберігає всі переваги онлайн навчання.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						4
<i>Изм.</i>	<i>Аркуш.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Завдання цього розділу – класифікація, виявлення ключових функціональних особливостей мобільних систем обміну освітнім контентом, огляд існуючих рішень, виявлення їх переваг та недоліків.

В класичному розумінні мобільна система обміну освітнім контентом - це веб-простір або портал для поширення освітнього контенту та ресурсів, який пропонує користувачам необхідні функції для навчання: лекції, ресурси, можливості зустрічатися та спілкуватися з іншими студентами тощо.

1.1 Класифікація мобільних систем обміну освітнім контентом

Мобільні системи обміну освітнім контентом можуть бути класифіковані за функціоналом, який вони надають користувачам[1].

1.1.1 Цільові сайти навчання

Зазвичай це спільний веб-сайт, який пропонує курси від багатьох різних постачальників. Це виглядає як онлайн магазин з курсами.

Як правило, цільові сайти для навчання дозволяють вам створювати свої курси за допомогою інструмента розробки або системи керування навчанням, а потім завантажувати або пов'язувати курси з сайтом. Зазвичай сайти займаються реєстрацією учнів та електронною комерцією за вас.

1.1.2 Традиційна система управління навчанням

Традиційна комерційна система управління навчанням – це тип платформи, що забезпечує базову функціональність, необхідну для розробки та розміщення онлайн-курсу.

Це дає розробникам курсів засоби для створення курсів, їх зберігання, керування профілями навчання, звітування про оцінки та прогрес, а також надає інші прості функції, необхідні для онлайн-курсів.

На ринку існує понад 700 постачальників систем управління навчанням, і більшість орієнтованих на корпоративне навчання та розвиток — їхні продукти розроблені для внутрішнього навчання. Однак деякі з них мають «розширені

					ІАЛЦ.467200.003 ТЗ	Аркуш
						5
Изм.	Аркуш.	№ докум.	Підпис	Дата		

корпоративні» функції, які полегшують перепрофілювання вмісту для зовнішньої аудиторії.

1.1.3 Система управління навчанням з відкритим кодом

Система управління навчанням з відкритим кодом — це тип платформи, що забезпечує базову функціональність, необхідну для розробки та розміщення онлайн-курсу — так само, як комерційне рішення LMS.

Різниця в тому, що вони, як правило, безкоштовні і їх можна налаштувати.

Для кожної платформи зазвичай існують спільноти постачальників, які можуть налаштовувати, розміщувати та налаштовувати системи управління навчанням з відкритим кодом за певну плату. Поширеними навчальними платформами з відкритим кодом є Moodle, Sakai і Open edX.

1.1.4 Нетрадиційні системи навчання

Представляють з себе платформи, які мають нетрадиційний функціонал. Наприклад, Patreon, може виступати системою для навчання, якщо автор поширює навчальний матеріал.

1.2 Функціональність мобільних систем обміну освітнім контентом

Сучасні системи обміну освітнім контентом надають широкий спектр можливостей для користувачів[4]:

1.2.1 Створення організації, курсу.

Дозволяє користувачу створити власний курс. На рівні бази даних представляє з себе запис або в таблиці(SQL) або документі (NOSQL).

1.2.2 Публікація лекцій

Базова функція всіх систем онлайн навчання. Зазвичай містить в собі відео- або аудіо-запис з онлайн лекцією, текстовий варіант лекції, короткі нотатки до неї, додаткові матеріали тощо.

Для збереження файлів на платформі зазвичай використовуються хмарні сховища, такі як AWS S3 (Amazon Web Services Simple Storage Service), GCP Cloud Storage (Google Cloud Platform Cloud Storage) та їх аналоги з інших шмарних сервісів.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						6
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

Також деякі додатки використовують власні дата-центри або виділені сховища у дата центрах, такі як Hetzner.

Для інформаційного підходу: різні імплементації реляційних СУБД (Postgres, MsSql, Oracle), NoSQL СУБД (Firestore, MongoDB, Cassandra), а також багато інших способів.

1.2.3 Реєстрація на курс

Також одна з базових функцій будь-якої системи навчання. Користувач має змогу зареєструватись на курс та відслідковувати свій прогрес. Виставляти собі дедлайни, або використовувати ті, які встановлені викладачем.

1.2.4 Пошук лекцій

Зазвичай на платформах використовується пошук декількох типів: пошук за атрибутам, пошук по назві чи використовуючи систему рекомендацій.

Пошук за атрибутам - дає можливість відфільтрувати лекції по категоріям. Наприклад, по галузі чи по предмету.

Пошук за назвою - пошук по тексту в назві лекції чи в її матеріалах.

Система рекомендацій - сервіс відслідковує чим цікавиться користувач та рекомендує йому потенційно цікавий контент.

1.2.5 Об'єднання лекцій в курс та спеціалізації

Дозволяє об'єднати декілька лекцій в курс, вказати порядок лекцій при навчанні, додати опис курсу. Допомогає учню краще орієнтуватися в матеріалах та краще структурувати знання. Реалізується за допомогою баз даних. В SQL базах даних реалізується за допомогою зв'язків one-to-many або one-to-one, в залежності від можливих типів лекцій на платформі[3].

1.2.6 Тести та підсумкові роботи

Викладач повинен мати можливість розмістити тест або підсумкову роботу іншого формату, яку студент повинен виконати в певний час.

Можливі інтеграції з сторонніми сервісами, які зроблять процес виконання завдання більш зручним, наприклад, з сервісом Miro.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						7
Изм.	Аркуш.	№ докум.	Підпис	Дата		

1.2.7 Онлайн щоденник

Важливий компонент системи онлайн оцінювання, але не обов'язковий.

Потребує того, щоб викладач перевіряв роботи протягом курсу та виставляв оцінки в онлайн щоденник. Також можливий peer-to-peer варіант виставлення оцінок, коли робота перевіряється та оцінюється іншими студентами. Онлайн щоденник дозволяє студенту оцінити свої знання та відстежувати свій прогрес.

1.2.8 Особистий кабінет

Особистий кабінет дає можливість користувачу побачити курси на які він зареєстрований, прогрес в курсах, свої оцінки та іншу статистику.

1.3 Аналіз представлених на ринку рішень

Далі будуть розглянуті наявні на ринку рішення мобільних систем обміну освітнім контентом.

1.3.1 Coursera

Інтернет-ресурс для онлайн навчання. Дає можливість університетам або організаціям публікувати власні матеріали. Coursera пропонує своїм користувачам значну кількість безкоштовних онлайн-курсів з різних дисциплін. У разі успішного завершення курсу студент отримує сертифікат, який підтверджує проходження. Платформа співпрацює з десятками університетів по всьому світі. Наразі пропонуються курси в наступних галузях: інженерія, гуманітарні науки, медицина, біологія, суспільні науки, математика, бізнес, інформатика та інші[5].

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						8
Изм.	Аркуш.	№ докум.	Підпис	Дата		

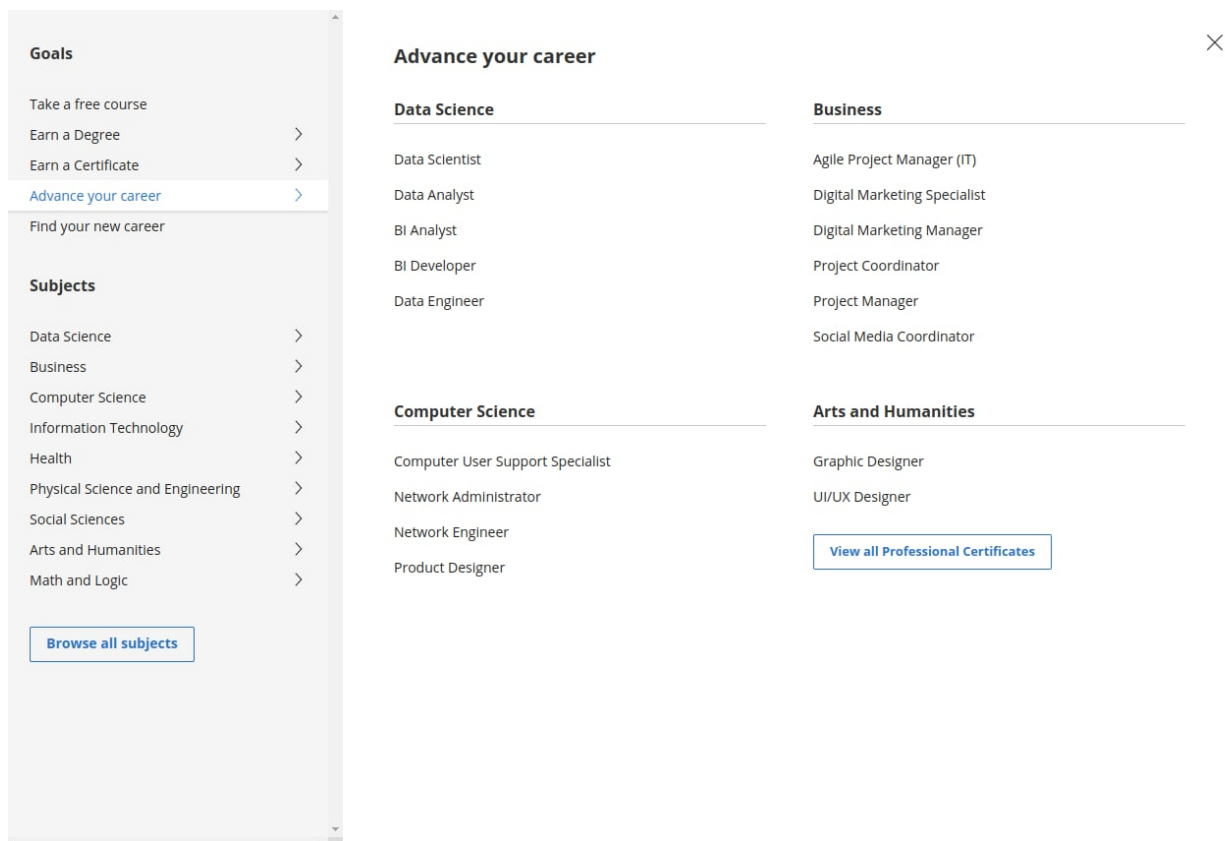


Рис 1.1 Пошук матеріалу на платформі coursera

Пошук на платформі згрупований по категоріям, в залежності від цілей, як зображено на рис. 1.1. У користувача є можливість відфільтрувати лише по безкоштовним курсам, по курсам з відповідним типом тощо. Також можна відфільтрувати по напрямленню, наприклад, комп'ютерні науки.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						9
Изм.	Аркуш.	№ докум.	Підпис	Дата		

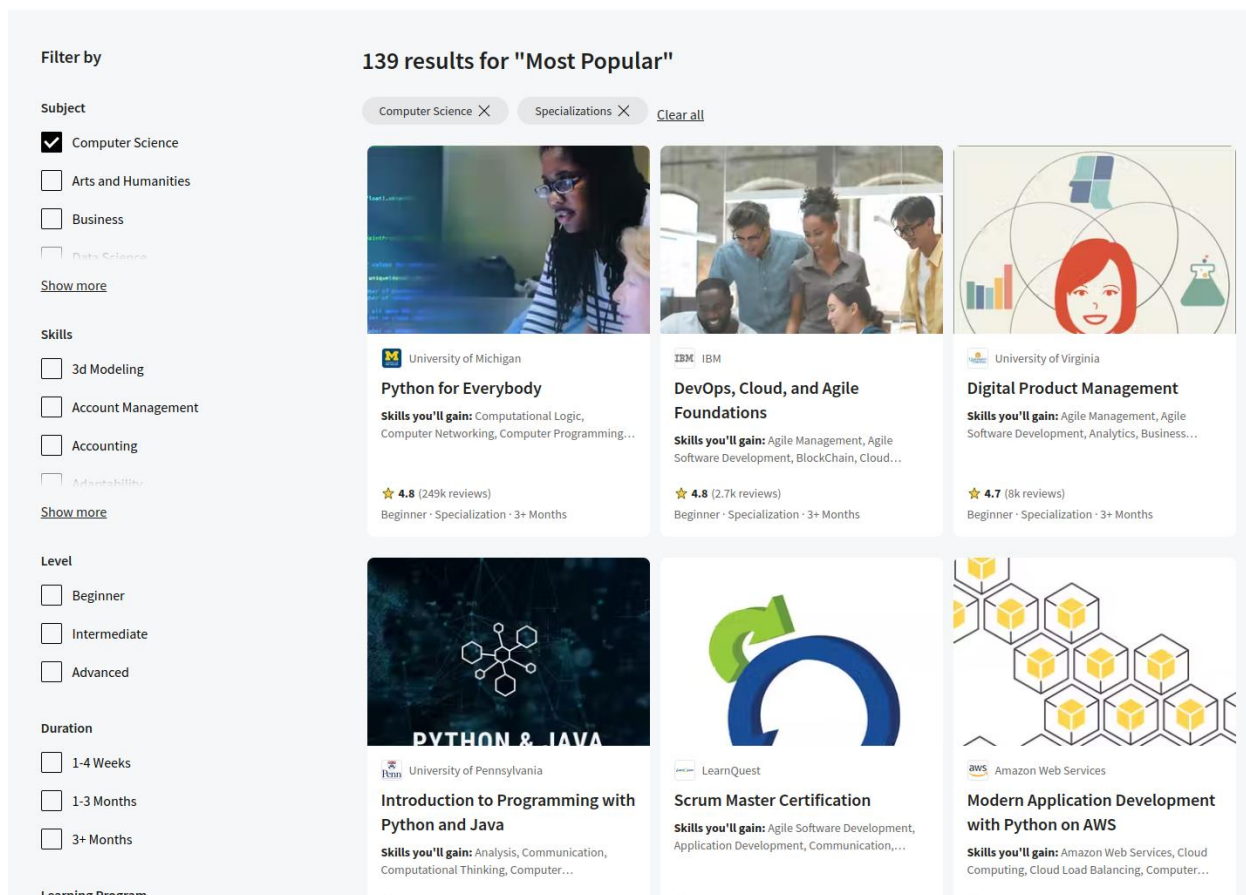


Рис 1.2 Приклад результату пошуку навчальної програми на платформі Coursera

Результат пошуку повертає список доступних спеціалізацій. В цьому випадку, відсортованих, по популярності.

При відкритті сторінки спеціалізації користувач має змогу побачити назву, викладачів, організацію, яка надає її та можливість зареєструватися в спеціалізацію.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						10
Изм.	Аркуш.	№ докум.	Підпис	Дата		

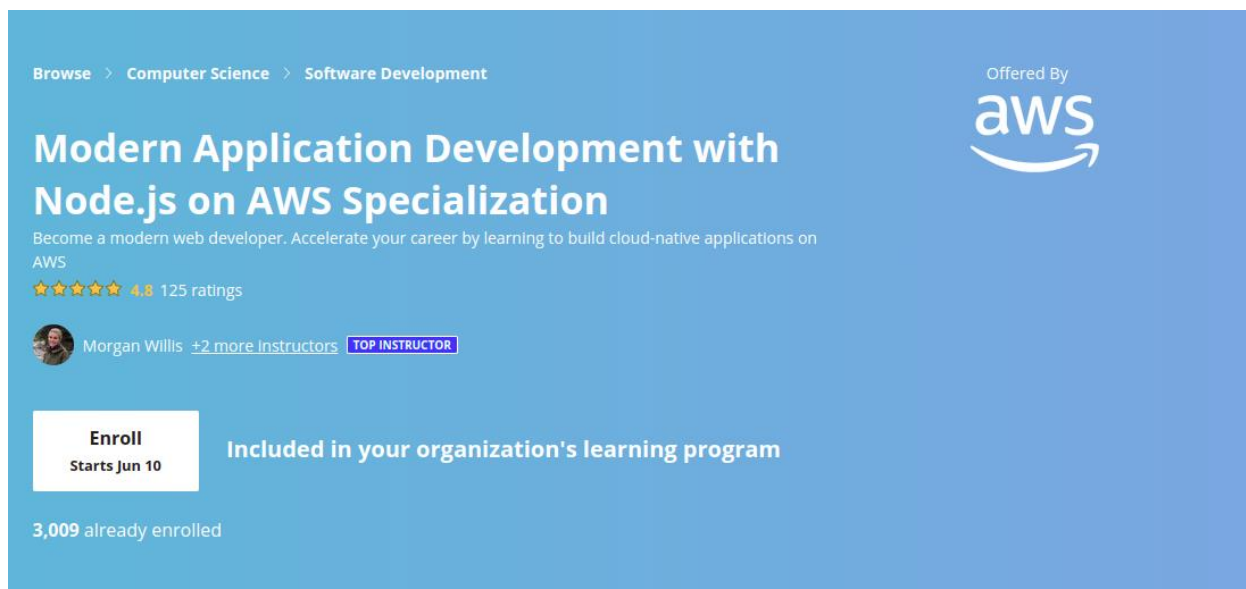


Рис 1.3 Заголовок сторінки спеціалізації на ресурсі Coursera

Нижче розташована інформація про переваги конкретної спеціалізації, його опис, що буде вивчено протягом спеціалізації, відповіді на часті запитання тощо.

[About](#) [How It Works](#) [Courses](#) [Instructors](#) [Enrollment Options](#) [FAQ](#)

WHAT YOU WILL LEARN

- ✓ How to build an API driven application using AWS Lambda for serverless compute
- ✓ You will be able to make an informed decision about when and how to apply key AWS services for compute, storage, and database to different use cases.
- ✓ How NoSQL databases differ from relational databases and the challenges they solve
- ✓ How NoSQL databases differ from relational databases
- ✓ How to provision, manage and interact with a DynamoDB table
- ✓ How to secure a DynamoDB database

SKILLS YOU WILL GAIN

- Cloud Computing Architecture
- Serverless Computing
- Application development
- Amazon Web Services (Amazon AWS)
- Amazon Dynamodb
- Compute
- Cloud Computing Security
- AWS cloud
- Networking
- Storage
- NoSQL Database
- Cryptography

About this Specialization

6,972 recent views

This specialization is designed to help you master the skills of designing and building cloud-native applications on AWS. We begin with the foundational technical and cloud knowledge that you need to have to build in the AWS Cloud. In the first course, you will be introduced to several AWS compute services, different storage and database offerings that AWS provides, AWS' networking capabilities, monitoring capabilities and the AWS IAM service.

The second course in the specialization explores how to build an API driven application using Amazon API Gateway for serverless API hosting, AWS Lambda for serverless computing, and Amazon Cognito for serverless authentication.

Modern applications require a modern database. Hence, the third course introduces you to NoSQL databases and the challenges they solve. We will dive deep into Amazon DynamoDB topics such as recovery, SDKs, partition keys, security and encryption, global tables, stateless applications, streams, and best practices. Amazon DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale.

Note: There are 4 versions of this specialization. We recommend you choose the Specialization based on your programming language or platform of choice: [Python](#), [Java](#), [.NET](#), & [Node.js](#).

- Shareable Certificate**
Earn a Certificate upon completion
- 100% online courses**
Start instantly and learn at your own schedule.
- Flexible Schedule**
Set and maintain flexible deadlines.
- Beginner Level**
No prior knowledge of AWS cloud is required. However, it will be helpful to have foundational to intermediate knowledge of Node.js.
- Approximately 4 months to complete**
Suggested pace of 3 hours/week
- English**
Subtitles: English, Arabic, French, Portuguese (European), Italian, Vietnamese, German, Russian, Spanish

Рис 1.4 Опис спеціалізації на ресурсі Coursera

					ІАЛЦ.467200.003 ТЗ	Аркуш
						11
Изм.	Аркуш.	№ докум.	Підпис	Дата		

Після реєстрації на спеціалізацію користувач автоматично реєструється і на перший курс цієї спеціалізації. На сторінці навчання курсу користувач має можливість перейти на сторінки відповідних тижнів навчання та на інші корисні сторінки, такі як оцінки, форум, матеріали та інші.

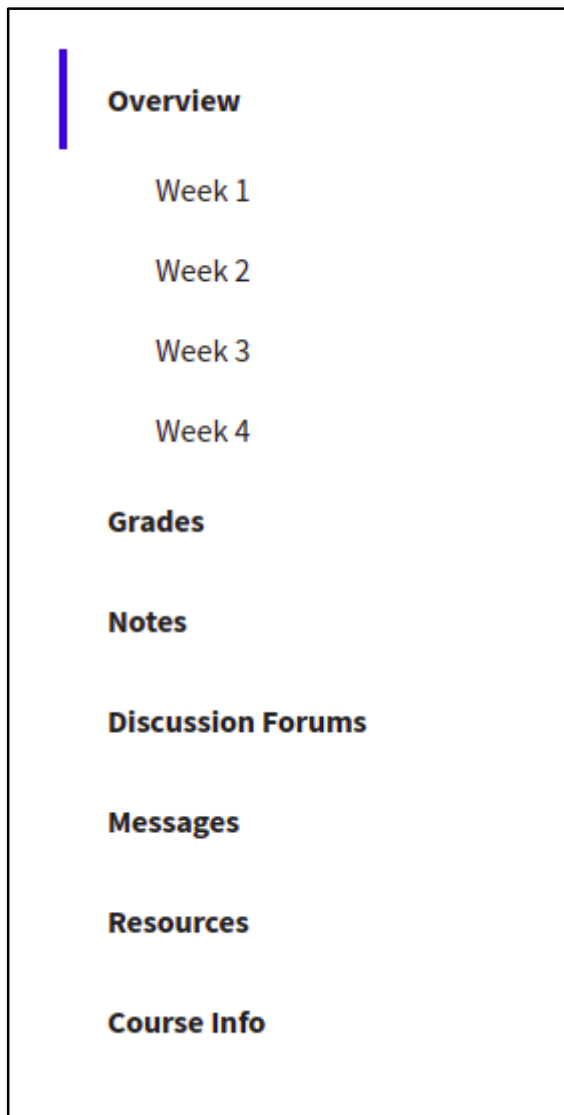


Рис 1.5 Меню навігації курсу на Coursera

Після початку навчання користувачу доступний перегляд свого прогрес, як це зображено на рис. 1.6.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						12
Изм.	Аркуш.	№ докум.	Підпис	Дата		

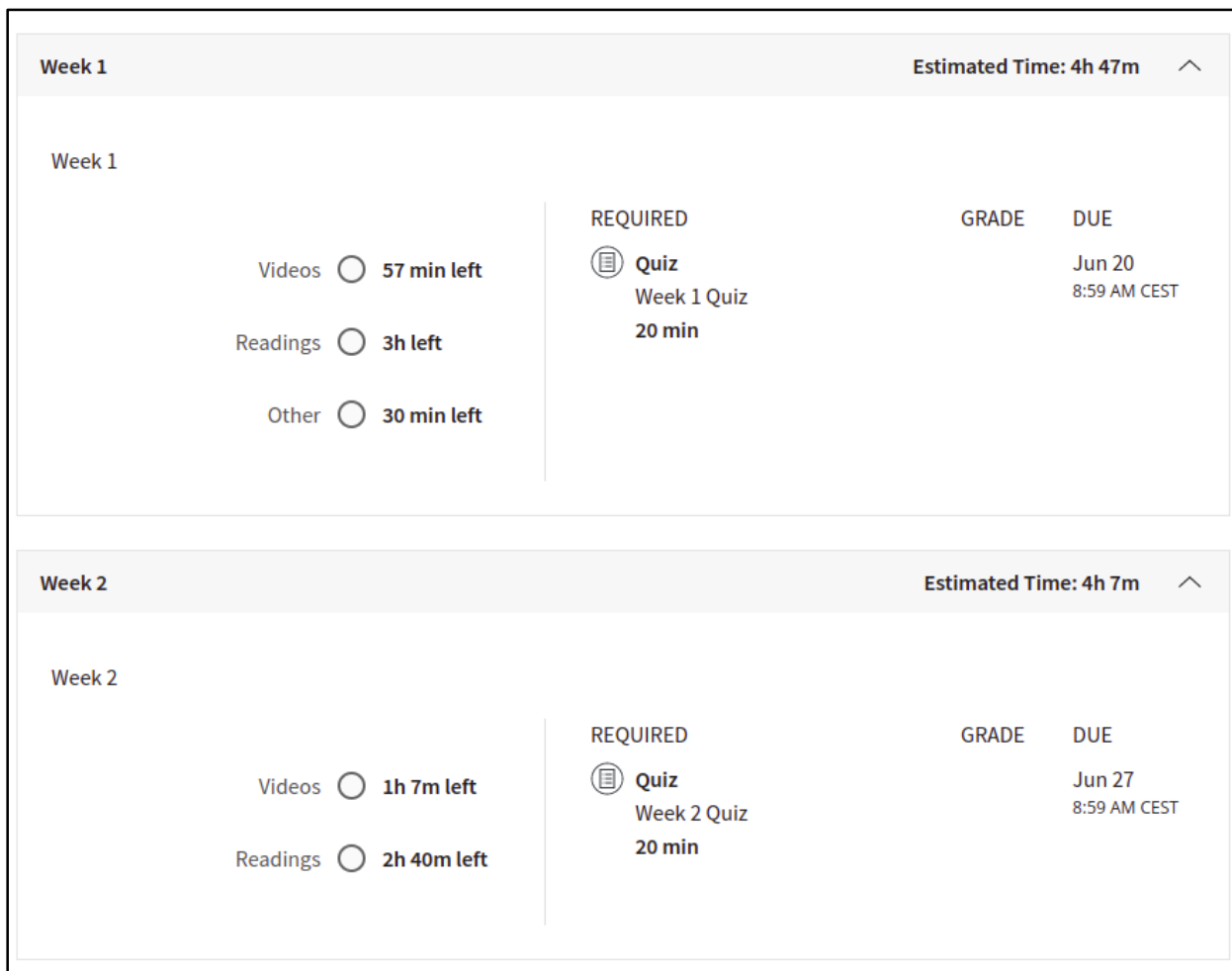


Рис 1.6 Сторінка прогресу навчання на Coursera

Переваги:

- навчальні матеріали об'єднані в курси, а курси в спеціалізації
- відстеження прогресу
- велика кількість матеріалів

Недоліки:

- розміщувати матеріали можуть лише організації
- платні матеріали

1.3.2 Google classroom

Google classroom – це безкоштовна платформа змішаного навчання, розроблена Google для навчальних закладів, яка спрямована на спрощення створення, розподілу та оцінювання завдань. Основна мета Google Classroom – оптимізувати процес обміну файлами між викладачами та студентами[6].

					ІАЛЦ.467200.003 ТЗ	Аркуш
						13
Изм.	Аркуш.	№ докум.	Підпис	Дата		

Станом на 2021 рік приблизно 150 мільйонів користувачів користуються Google Classroom.

Google Classroom інтегрує різноманітні інші програми Google для освіти, такі як Google Docs, Google Sheets, Google Slides, Gmail і Google Calendar, у єдину платформу для керування спілкуванням учнів і викладачів. Студентів можна запросити приєднатися до класу через приватний «код класу» або автоматично імпортувати з домену школи. Викладачі можуть створювати, розповсюджувати та позначати завдання в межах домену Google. Кожен клас створює окрему папку на Google Диску відповідного користувача, куди учень може подати роботу для оцінки вчителю. Завдання та терміни виконання додаються до Календаря Google, де кожне завдання може належати до категорії або теми. Вчителі можуть стежити за прогресом кожного учня, переглядаючи історію переглядів документа, а після оцінки вчителі можуть повертати роботу разом із коментарями та оцінками.

Переваги:

- безкоштовний
- можливість виставлення оцінок, додавання коментарів

Недоліки:

- відсутність відстеження прогресу
- немає можливості знайти курс та вступити до нього

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						14
Изм.	Аркуш.	№ докум.	Підпис	Дата		

ВИСНОВОК ДО РОЗДІЛУ 1

В цьому розділі була проведена класифікація типових систем онлайн навчання.

Були виявлені ключові функції, які присутні в кожній системі онлайн навчання.

Розглянуто більш детально деякі наявних систем на ринку, а саме:

- Coursera
- Google classroom

Виявлено їхні переваги та недоліки. Встановлено основні функціональні потреби для системи. Сформульовано вимоги для розроблюваної системи:

- безкоштовність
- відкритість
- гнучкість при навчанні та оцінюванні

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						15
Изм.	Аркуш.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ

Завдання цього розділу – опис архітектурних підходів, які використовуються при розробці сучасних систем. Аргументація вибору архітектури для мобільної системи обміну освітнім контентом.

2.1 Архітектура проекту

Архітектура програмного забезпечення є усвідомленою моделлю для завдання алгоритмів через програмування архітектури. Архітектура визначає, які елементи входять до системи, які функції виконує той чи інший елемент і як кожен елемент пов'язаний один з одним, відображає загальну структуру всієї системи[7].

З цього випливає, що при проектуванні архітектури потрібно враховувати багато факторів:

- призначення системи
- аудиторія або користувачі, на яку спрямована робота системи
- функції, які є найбільш важливими для користувачів

Архітектура програмного забезпечення важлива, особливо для великих систем. Якщо є чіткий проект загальної системи з самого початку розробки, то у розробників є основа, на яку вони можуть опиратися при розробці. Кожен розробник буде знати, що та як необхідно розробляти в рамках проекту. Це дозволяє уникнути конфліктів, дублювання та зайвої роботи.

Деякі переваги правильно обраної та спроектованої архітектури програмного забезпечення включають:

- вища продуктивність команди програмного забезпечення, оскільки чітко визначена структура допомагає координувати роботу, розробляти функціонал або знаходити потенційні проблеми
- підвищення якості програмного забезпечення шляхом ретельного врахування потреб і перспектив усіх зацікавлених сторін

					ІАЛЦ.467200.003 ТЗ	Аркуш
						16
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

Архітектура допомагає полегшити обслуговування, повторне використання та адаптацію системи. Щоб розробити архітектуру програмного забезпечення, потрібно враховувати зацікавлені сторони системи. Зацікавлені сторони – це люди, які цікавляться наявним програмним забезпеченням. Вони або використовують систему, або якось отримують від неї користь.

2.2 Діаграма діяльності

Діаграма діяльності UML дозволяє відобразити можливі варіанти діяльності при користуванні системою[8].

Діяльність - це дії, які виконуються в системі, після завершення викликають виконання іншої дії. Наприклад, дія може змінювати або створювати нові об'єкти[9]. Ці дії просувають виконання програми вперед. Щоб створити діаграму діяльності, необхідно визначити:

- базові функції
- види діяльності
- відповідні умови діяльності системи

У мобільній системі онлайн навчання повинні бути присутні декілька базових функцій, які можуть бути представлені в вигляді діаграм активності:

- вхід до системи та заповнення особистих даних користувачем
- реєстрація на курс
- перегляд лекцій
- виконання практичних завдань
- онлайн щоденник
- результати курсу

Після цього є можливим створення відповідних діаграм діяльності.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						17
Изм.	Аркуш.	№ докум.	Підпис	Дата		

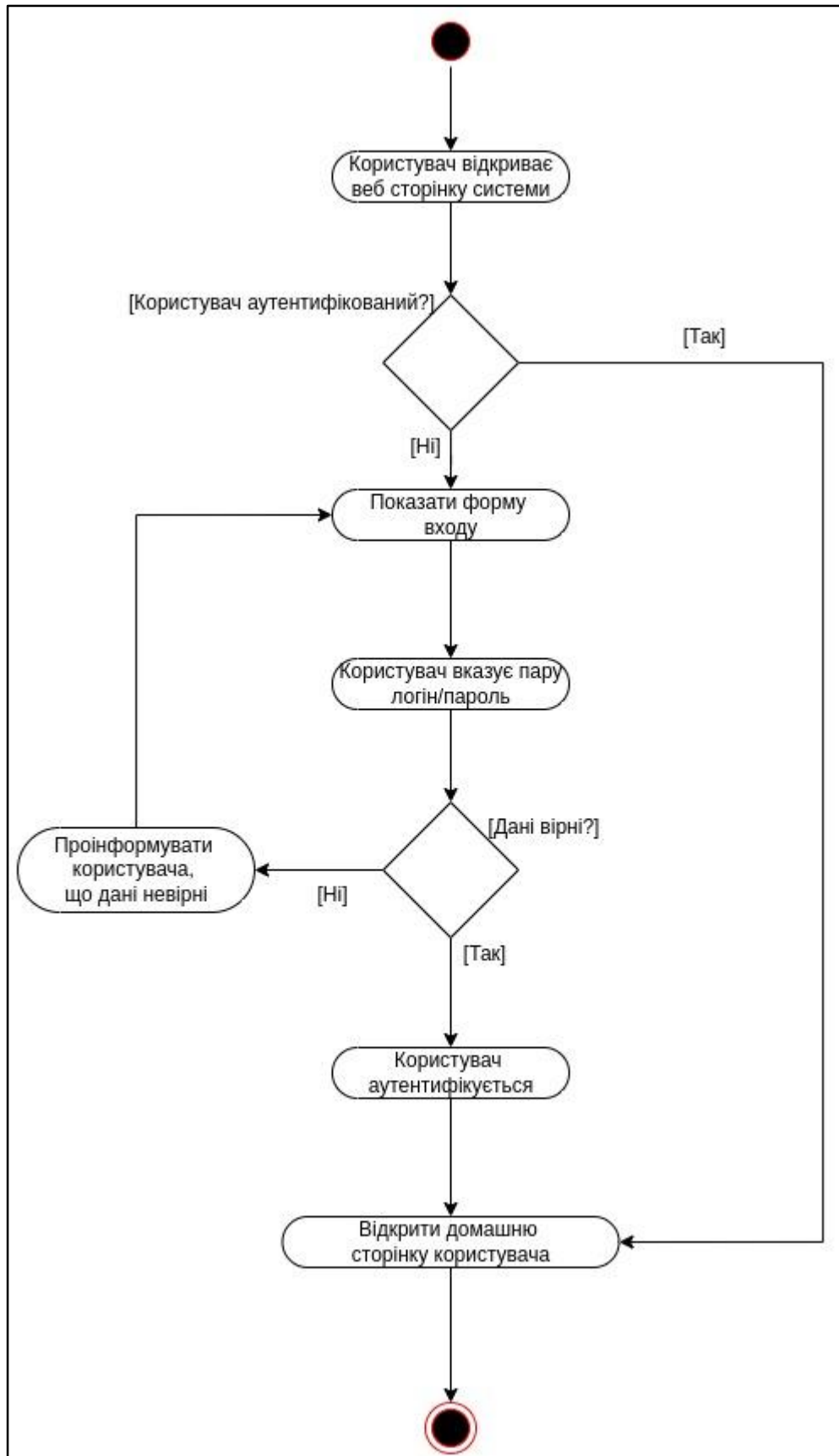


Рис 2.1 Діаграма активності входу в систему

Изм.	Аркуш.	№ докум.	Підпис	Дата

ІАЛЦ.467200.003 ТЗ

Аркуш

18

Після входу в систему користувач потрапляє на домашню сторінку платформи та має змогу знайти потрібний курс та почати навчання. Нижче зображена діаграма активності початку перегляду лекції.

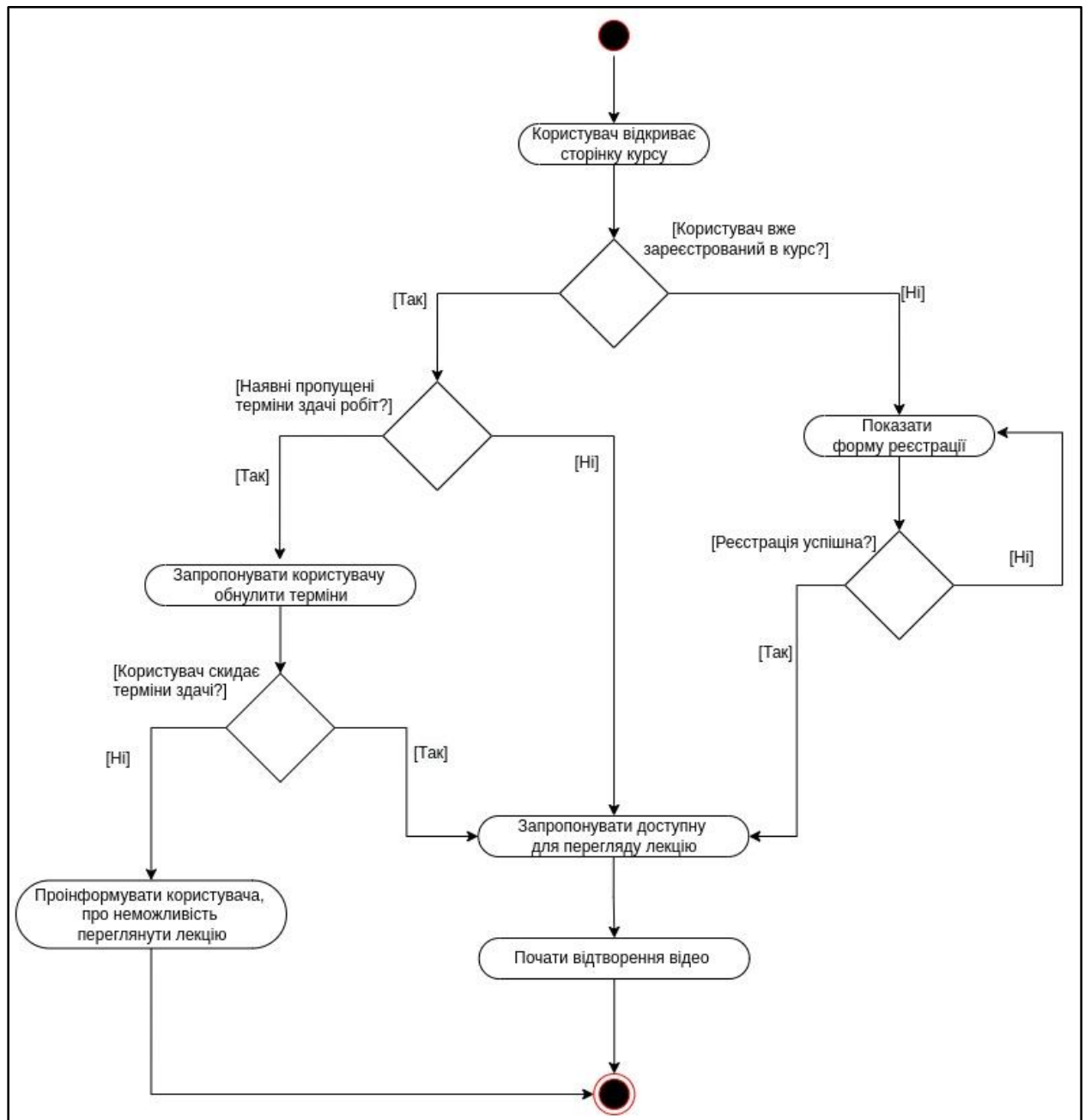


Рис 2.2 Діаграма активності входу в систему перегляду лекції

Курс складається з лекцій та практичних завдань. У випадку здачі практичного завдання - студент отримує оцінку автоматично або від викладача.

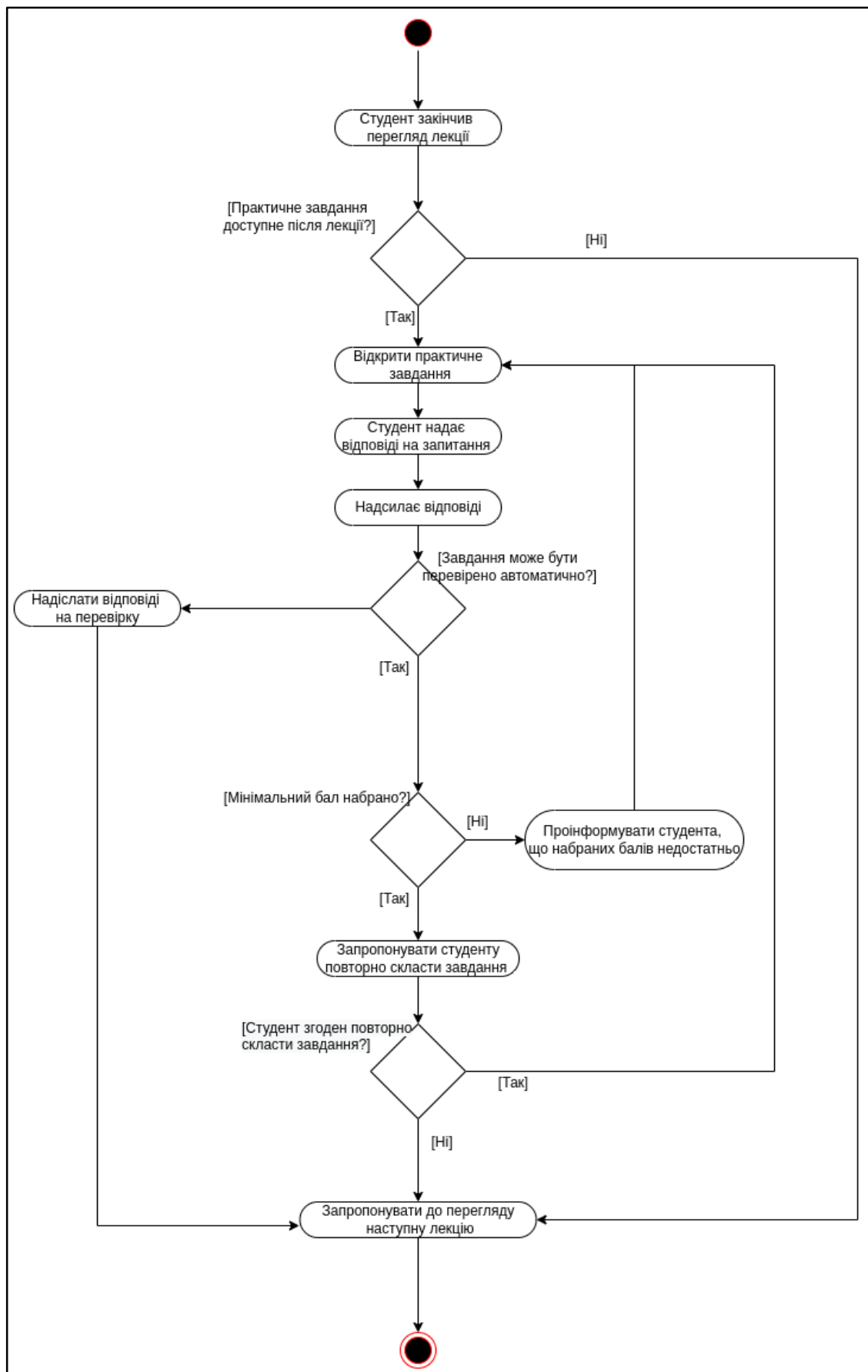


Рис 2.3 Діаграма активності перегляду лекції та складання практичного завдання.

Після перевірки практичного завдання студент отримує набрані бали та має можливість бачити їх в своєму щоденнику. Студент може завершити курс тільки, якщо переглянув всі лекції та склав практичні завдання.

2.3 Діаграма компонентів

Діаграми компонентів UML відображає компоненти системи[10]. Компоненти — це незалежні, інкапсульовані одиниці системи. Кожен компонент забезпечує інтерфейс для інших компонентів для взаємодії з ним. Діаграми компонентів використовуються, щоб представити взаємозв'язки між елементами системи.

Діаграми компонентів відрізняються від більшості інших діаграм, оскільки вони показують структуру високого рівня, а не такі деталі, як атрибути та методи. Вони зосереджені виключно на компонентах та їх взаємодії один з одним.

Діаграми компонентів є статичним уявленням про програмну систему та зображують проект системи на певному етапі її розвитку та еволюції.

Основа діаграм компонентів зосереджена на компонентах та їх взаємозв'язках. Кожен компонент на діаграмі має дуже специфічне відношення до інших компонентів через інтерфейс, який він надає.

Щоб побудувати діаграму компонентів, спочатку необхідно визначити основні об'єкти, які використовуються в системі. Далі необхідно визначити відповідні бібліотеки для системи. Нарешті, необхідно визначити взаємозв'язок між цими компонентами.

Коли відповідні бібліотеки визначені, це поширюється на залежності реалізації сторонніх розробників, які також слід інтегрувати в діаграму, якщо це необхідно. Нижче зображена діаграма компонентів мобільної системи обміну освітнім контентом.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						21
Изм.	Аркуш.	№ докум.	Підпис	Дата		

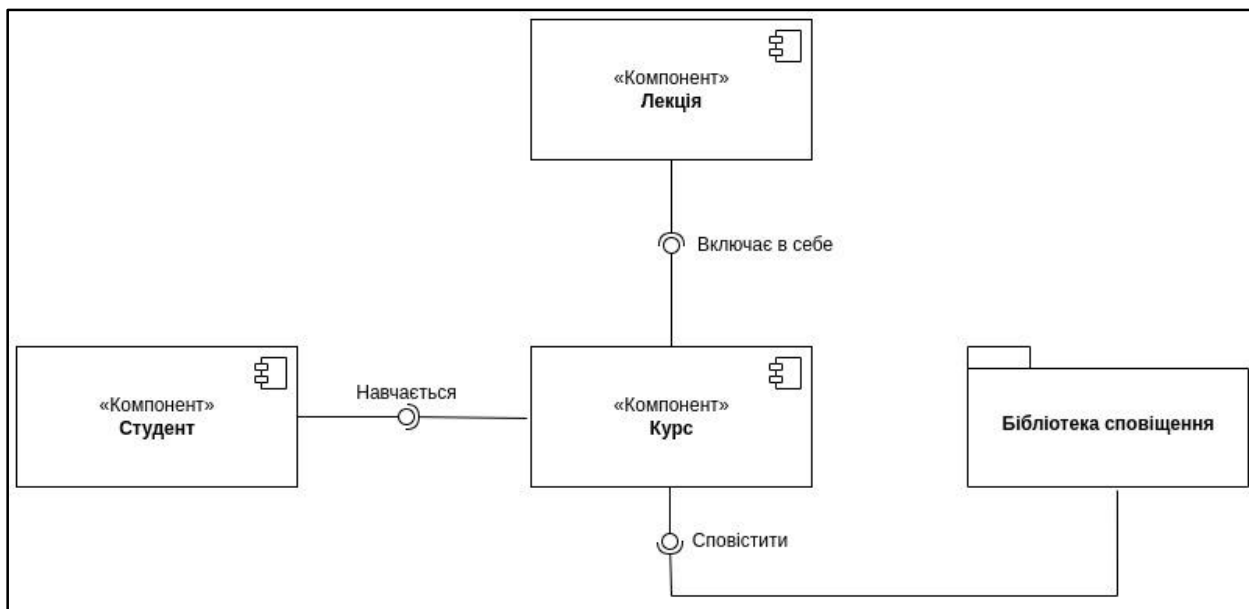


Рис 2.4 Діаграма компонентів мобільної системи обміну освітнім контентом

Діаграми компонентів особливо корисні на початку процесу проектування, оскільки вони мають високий рівень акценту. Вони можуть бути спроектовані на різних рівнях і дозволяють зосередитися не тільки на системах, а й на підсистемах.

2.4 Пакетна діаграма

Іноді класи об'єктно-орієнтованого програмного забезпечення певним чином пов'язані.

Пакет об'єднує елементи програмного забезпечення, які пов'язані між собою[11]. Елементи можуть бути пов'язані на основі даних, класів або завдань користувача. Пакет також може визначати «простір імен» для елементів, які він містить, тобто пакет має ім'я і може організувати названі елементи програмного забезпечення в окрему область. Елемент може бути однозначно ідентифікований в системі за повним «повним іменем», що базується на його власному імені та назві пакета, в якому знаходиться елемент.

Діаграми пакетів показують пакети та залежності між ними. Ці діаграми можуть організувати завершену систему в пакети пов'язаних пакуваних елементів, які можуть включати дані, класи або навіть інші пакети. Діаграми пакетів допомагають забезпечити групування системи високого рівня, щоб було легко побачити, як пакет

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						22
Изм.	Аркуш.	№ докум.	Підпис	Дата		

містить пов'язані елементи, а також як різні пакети залежать один від одного. Нижче наведено приклад діаграми пакетів для мобільної платформи для онлайн навчання:

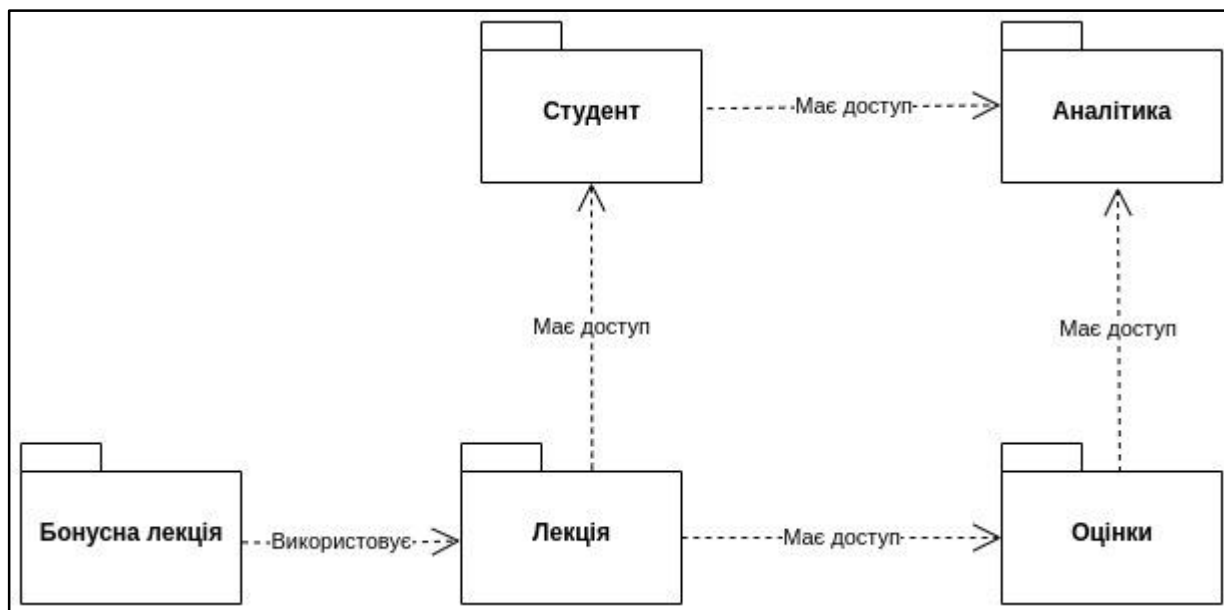


Рис 2.5 Діаграма пакетів мобільної системи обміну освітнім контентом

2.5 Архітектурні підходи

Правильно обраний архітектурний підхід допомагає більш ефективно розроблювати та використовувати систему, тому нижче будуть розглянуті деякі найбільш популярні підходи.

2.5.1 Системи на основі мови програмування

Парадигма програмування мови, обраної для реалізації системи, вплине на архітектурний стиль цієї системи[12]. Кожна парадигма програмування має власний набір конструкцій, принципів і шаблонів проектування, і їх використання формує створювану систему.

Архітектурні стилі об'єктно-орієнтованого проектування орієнтовані на дані.

При моделюванні системи в об'єктно-орієнтованому стилі проектування починається з огляду на типи даних, які обробляє система, щоб побачити, як систему можна розділити на абстрактні типи даних. Абстрактний тип даних може бути представлений у вигляді класу, тому логічно пов'язані атрибути групуються разом із

пов'язаними з ними методами. Інкапсуляція обмежує доступ до даних і визначає, що з ними можна робити.

Об'єктно-орієнтована – це система, що складається з об'єктів, де кожен об'єкт є екземпляром класу. Іншими словами, тип кожного об'єкта є його класом.

Об'єкти можуть взаємодіяти один з одним за допомогою своїх методів.

Об'єктно-орієнтована парадигма дозволяє успадковувати абстрактні типи даних. Це означає, що один абстрактний тип може бути оголошений розширенням іншого типу.

Самі ці класи формують архітектуру, засновану на мові, яка впливає з основних об'єктно-орієнтованих принципів. Класи всередині системи визначають загальну структуру системи. Іншими словами, загальний об'єктно-орієнтований архітектурний стиль системи безпосередньо впливає з того, що при розробці використовувався об'єктно-орієнтований підхід.

Деякі проблеми добре підходять для об'єктно-орієнтованого архітектурного стилю. Однак не у всіх ситуаціях можуть бути легко ідентифіковані класи. У деяких ситуаціях інший варіант архітектури може краще підходити для вирішення даної проблеми.

2.5.2 Основна програма і підпрограми

Архітектурний стиль основної програми та підпрограм зосереджені на функціях. Вони розвиваються з парадигми процедурного програмування[13]. Мова програмування C є прикладом мови, яка дотримується цієї парадигми.

У архітектурному стилі така система моделюється шляхом розбиття загальної функціональності системи на основну програму та підпрограми (рис 2.6).

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						24
Изм.	Аркуш.	№ докум.	Підпис	Дата		

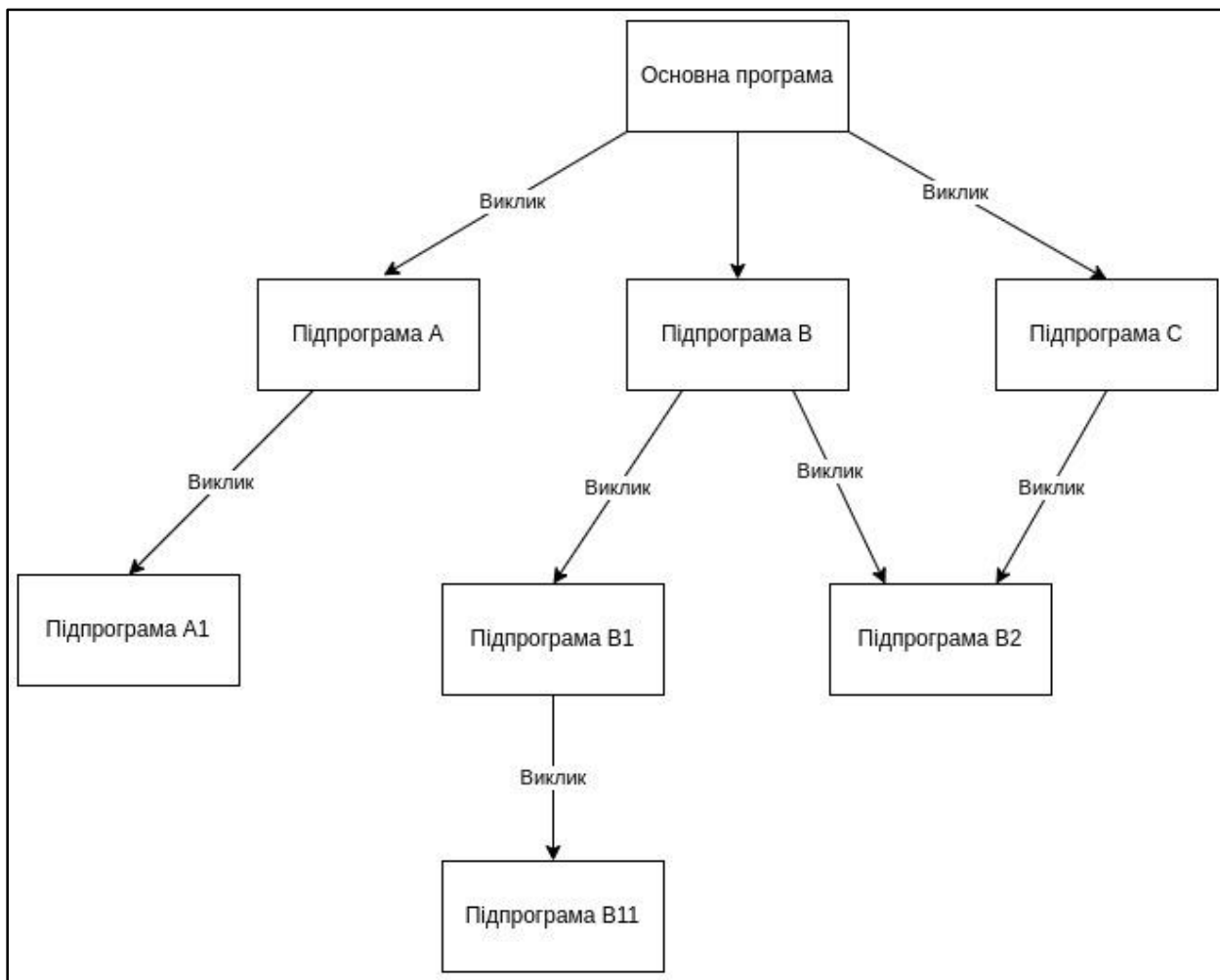


Рис 2.6. Схематичне зображення архітектурного стилю основної програми та підпрограми

На діаграмі вище зображено архітектурний стиль основної програми та підпрограм. На схемі підпрограми з'єднані викликами, і вони можуть мати вкладені виклики. У вкладених викликах підпрограми можуть викликати інші підпрограми, які можуть викликати ще більше підпрограм тощо. Це означає, що структура отриманого коду не є плоскою, а є ієрархічною, тому її можна моделювати як орієнтований граф. Структура підпрограм, які будують систему, впливає на структуру системи в цілому. Підпрограми, оголошені в коді, мають структуру як велике «дерево викликів».

У цій парадигмі дані зберігаються як змінні. У процедурному програмуванні підтримуються абстрактні типи даних, однак наслідування явно не підтримується. Відповідно до цієї парадигми нелегко зробити один абстрактний тип розширенням іншого. Таким чином, основна увага цієї парадигми приділяється поведінці функцій і

тому, як дані переміщуються через ці функції. Архітектурний стиль основної програми та підпрограм найкраще підходить для систем, орієнтованих на обчислення.

Кожна підпрограма може мати власні локальні змінні. Підпрограма має доступ до всіх даних у межах своєї області дії. Для доступу до даних, що виходять за межі її області, дані можуть передаватися в підпрограми як параметри, за значенням або за посиланням, а дані можуть передаватися з підпрограми як значення, що повертається.

Принцип цієї парадигми полягає в тому, що «один вхід, один вихід на підпрограму», що дозволяє легше виконувати потік керування підпрограмою. Цей архітектурний стиль сприяє модульності та повторному використанню функцій, що призводить до певних переваг. Якщо функції чітко визначені і не викликають побічних ефектів, вони вважаються «чорними ящиками» або «чистими» функціями. При однакових вхідних даних, вони завжди матимуть однаковий вихід.

Крім того, функції бібліотеки легко інтегруються в програми. У цього стилю є і певні недоліки. Підпрограми можуть змінювати дані несподіваним чином. На підпрограму можуть вплинути зміни даних, внесені іншою підпрограмою під час виконання.

Ця проблема особливо актуальна для глобальних даних, спільних між підпрограмами. Зміни в даних можуть бути непередбачуваними і призвести до того, що функція отримає вхідні дані, яких вона не очікувала, або навіть вхідні дані, які вона не в змозі обробити правильно, що може призвести до помилок під час виконання.

Парадигми процедурного програмування добре підходять для систем, зосереджених на обчисленні. Імплементация в об'єктно-орієнтованому стилі таких систем може бути складною і призвести до надто складних рішень. Однак для задач, де абстрактні типи даних з моделюванням полегшують вирішення, тоді краще підходять об'єктно-орієнтовані архітектурні стилі.

2.5.3 Системи на основі репозиторіїв

Сучасні програмні системи рідко працюють в ізольованих середовищах.

Будь-яка архітектура програмного забезпечення повинна мати можливість обмінюватися інформацією між окремими компонентами.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
Изм.	Аркуш.	№ докум.	Підпис	Дата		26

Для цього необхідно вирішити ряд задач:

- під час роботи компонента стан компонента є лише тимчасовим, це означає, що будь-які створені об'єкти та будь-який обсяг обробки даних будуть втрачені, коли компонент припинить роботу.
- компоненти можуть не мати всіх даних, які потрібні для його роботи., тому потрібен спосіб передати стан даних між кількома компонентами.

Щоб вирішити обидві проблеми, може бути використана архітектура програмного забезпечення, орієнтована на дані. Ця архітектура дозволяє зберігати дані та обмінюватися ними між кількома компонентами, таким чином підвищуючи придатність до обслуговування, можливість повторного використання та масштабованість системи[14]. Ця архітектура може бути досягнута шляхом інтеграції методу спільного зберігання даних, наприклад, бази даних.

В основі архітектури, орієнтованої на дані, є два типи компонентів:

- Центр даних – це компонент, який використовується для зберігання та обслуговування даних у всіх компонентах, які підключаються до них.
- Засоби доступу до даних – це компоненти, які підключаються до центрального компонента даних. Засоби доступу до даних здійснюють запити та транзакції щодо інформації, що зберігається в базі даних.

Нижче наведено схему архітектури програмного забезпечення, орієнтованої на дані(див. рис 2.7).

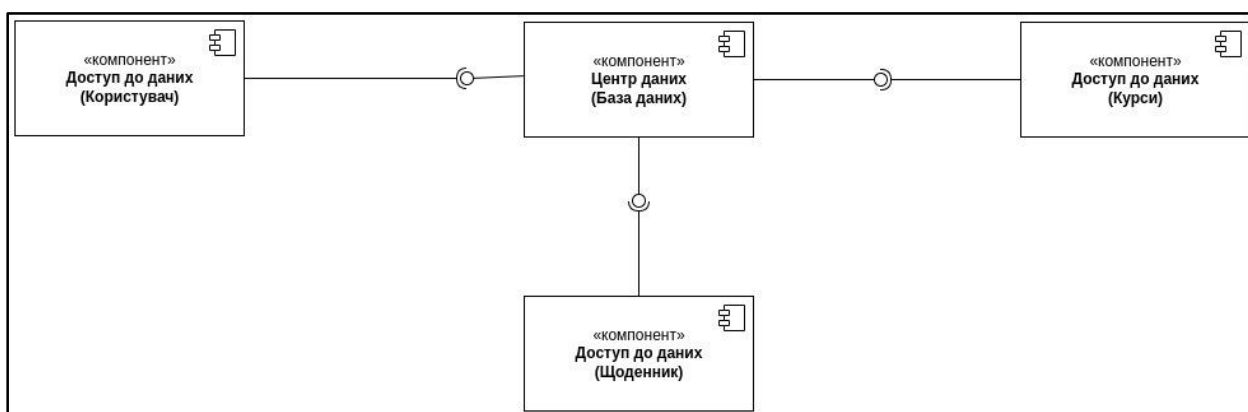


Рис 2.7. Приклад компонентної діаграми мобільної системи обміну освітнім контентом

Засоби доступу до даних відокремлені один від одного, і вони спілкуються лише з центром даних. Центр даних обмінюється даними, зберігає потрібну інформацію з поточного стану системи та обслуговує дані за запитом.

2.5.4 Багатошарові системи

У програмному забезпеченні шар — це сукупність компонентів, які працюють разом для виконання спільної роботи та досягнення певної мети[1].

Ключова характеристика багатошарової архітектури полягає в тому, що компоненти в шарі взаємодіють лише з компонентами власного шару або сусідніх шарів. Коли верхній шар взаємодіє з шаром безпосередньо під ним, він не може бачити жодного з більш глибоких шарів[15].

Зазвичай внутрішній або нижній шар надає послуги або абстракції для шару за межами або над ним. Інтерфейси, що надаються компонентами рівня, повинні бути чітко визначені та обумовлені потребами системи.

Шарування дозволяє застосувати «розділення проблем» до кожного з шарів. Багато багатошарових систем поділяються на рівні «презентації», «логіки» та «дані».

Операційна система для комп'ютера є поширеним прикладом багатошарової системи(див. рис 2.8).

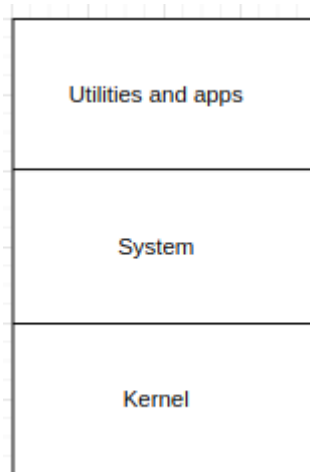


Рис. 2.8. Приклад багатошарової системи.

До переваг багатошарових систем можна віднести те, що:

- Шари можуть виконувати роботу, не беручи в увагу нижні шари

- Різні рівні можуть запускатися на різних рівнях авторизації або привілеїв. Як правило, верхній рівень вважається «простором користувача», який не має повноважень розподіляти системні ресурси або безпосередньо спілкуватися з обладнанням. Ця «пісочниця» забезпечує безпеку та надійність ядра.
- Проекти будуть більш слабо зав'язані один на одного, оскільки багат шарова архітектура дотримується принципу найменшого знання.

Нижчі рівні зазвичай надають інтерфейс для роботи вищому. Якщо шар замінено, наданий інтерфейс має відповідати попередній реалізації.

Багат шарову архітектуру не легко застосувати в кожному проекті. Не завжди може бути очевидним, як рішення можна розбити на шари і де буде поділ програмного забезпечення.

Існують певні компроміси, про які слід знати в багат шарових системах. Найбільш помітно, що примусові рівні мають компроміс з ефективністю. Якщо тільки суміжні шари можуть спілкуватися, то система, ймовірно, матиме певні взаємодії, які переходять від одного рівня до іншого, і іноді інформація повинна бути передана між шарами, які не є суміжними. Ця додаткова комунікація додає складності.

Якщо більшість комунікацій проходить безпосередньо через рівень, то багат шарова архітектура може бути послаблена, або ж інша архітектура може краще підійти для вирішення конкретної проблеми. Якщо проходить лише невеликий обсяг комунікації, то багат шарова система може бути хорошим вибором.

2.6 Види архітектур

Види архітектури – це сталі архітектури, які використовуються в тих чи інших випадках. Кожен з них мають переваги та недоліки. Далі будуть розглянуті найбільш популярні та ефективні види архітектур.

2.6.1 Монолітна архітектура

Монолітне програмне забезпечення є автономним, що означає компоненти програми тісно пов'язані.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						29
Изм.	Аркуш.	№ докум.	Підпис	Дата		

Монолітні програми є однорівневими, що означає, що кілька компонентів об'єднані в один великий додаток. Отже, вони, як правило, мають великі кодові бази, керувати якими з часом може бути важко[16].

Крім того, якщо один компонент програми необхідно оновити, інші елементи також можуть потребувати оновлення коду, і після цього всю програму потрібно буде перекомпілювати та протестувати. Цей процес може зайняти багато часу і може обмежити гнучкість і швидкість команд розробників програмного забезпечення[2]. Незважаючи на ці проблеми, цей підхід все ще використовується, оскільки має певні переваги. Крім того, багато ранніх програм були розроблені як монолітне програмне забезпечення, тому цей підхід не можна повністю нехтувати, коли цей вид архітектури все ще використовується.

Переваги:

- Просте розгортання. Використання одного файлу або каталогу, що виконується, спрощує розгортання.
- Розробка. Програму легше розробляти, коли вона створена з використанням однієї бази коду.
- Продуктивність. У централізованій базі коду та репозиторії один інтерфейс API часто може виконувати ту функцію, яку під час роботи з мікросервісами виконують численні API.
- Спрощене тестування. Монолітна програма є єдиним централізованим модулем, тому наскрізне тестування можна проводити швидше, ніж при використанні розподіленої програми.
- Зручне налагодження. Весь код знаходиться в одному місці, завдяки чому стає легше знаходити проблеми.

Недоліки:

- Зниження швидкості розробки. Великий монолітний додаток ускладнює та уповільнює розробку.
- Масштабованість. Неможливо масштабувати окремі компоненти.
- Надійність. Помилка в одному модулі може вплинути на доступність програми.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						30
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

- Перешкоди запровадження технологій. Будь-які зміни в інфраструктурі або мові розробки впливають на програму в цілому, що часто призводить до збільшення вартості та тимчасових витрат.
- Недостатня гнучкість. Можливості монолітних програм обмежені використовуваними технологіями.
- Розгортання. При внесенні невеликої зміни буде потрібно повторне розгортання всього монолітного додатка.

2.6.2 Сервіс-орієнтована архітектура

Сервісно-орієнтована архітектура — це вид архітектури, який базується на роботі з окремими сервісами. Сервіс — це чітко визначений і автономний функціонал програми. У цій архітектурі різні служби спілкуються один з одним для виконання певної діяльності[17]. Ці служби мають слабку зв'язність, тобто службі не обов'язково знати технічні деталі іншої служби, з якою вона взаємодіє(рис. 2.9). В основному, сервіс-орієнтована архітектура складається з сервісів, які надають будь-які послуги для інших сервісів(споживачів). Під час взаємодії і постачальник послуг, і споживач дотримуються попередньо визначеного протоколу.

Переваги:

- Масштабованість
- Надійність
- Незалежність сервісів
- Простіше обслуговування порівняно з монолітною архітектурою

Недоліки:

- Нижча пропусна спроможність
- Додаткове перевантаження
- Висока вартість

					ІАЛЦ.467200.003 ТЗ	Аркуш
						31
Изм.	Аркуш.	№ докум.	Підпис	Дата		

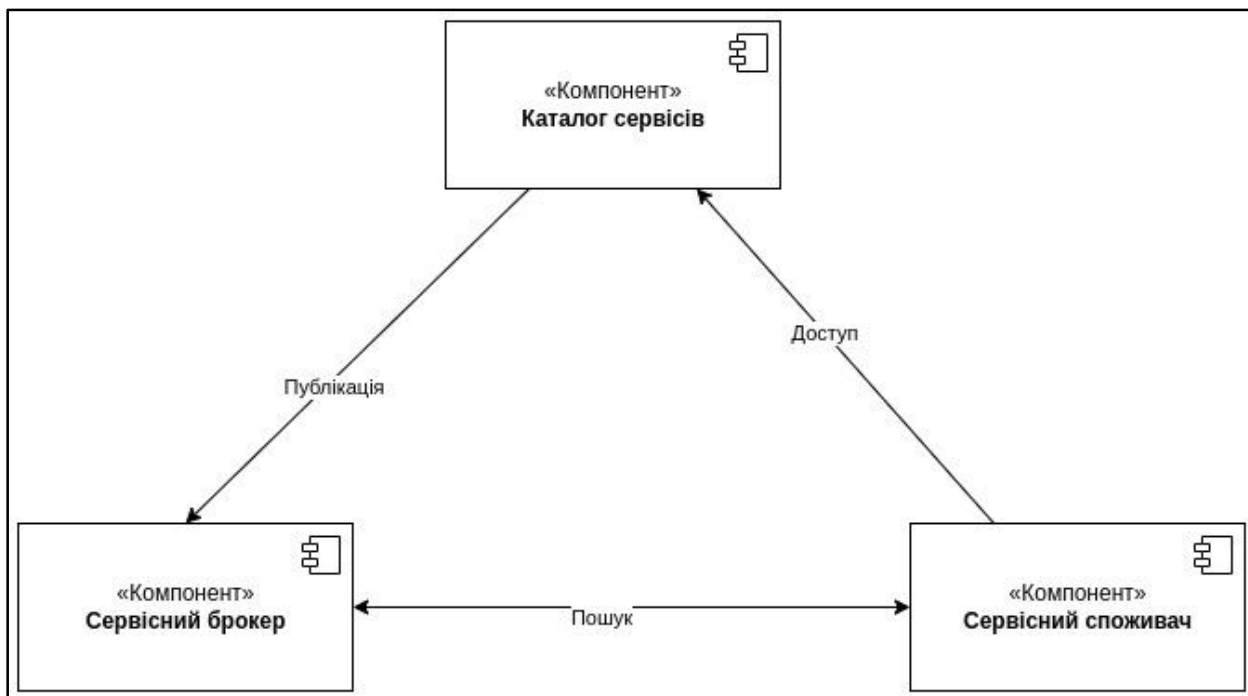


Рис 2.9 Схематичне зображення сервіс-орієнтованої архітектури

2.6.3 Мікросервісна архітектура

Мікросервіси — це спосіб розбиття великих програмних проектів на слабо пов’язані модулі, які взаємодіють один з одним за допомогою простих інтерфейсів програмного забезпечення[18]. Вони є прикладом модульного архітектурного стилю, заснованого на філософії розбиття великих програмних проектів на менші, незалежні та слабо пов’язані частини, який здобув популярність серед розробників завдяки своїм динамічним і гнучким якостям в управлінні API. Простіше кажучи, мікросервіси насправді є не що інше, як ще одне архітектурне рішення для проектування складних — переважно веб-додатків(рис. 2.10). Мікросервіси набули популярності як еволюція від сервісно-орієнтованої архітектури[19].

Переваги:

- Висока гнучкість системи
- Висока можливість до масштабування
- Ізольовані сервіси мають кращу відмовостійкість
- Масштабування процесу розробки

Недоліки:

- Складніший процес розробки на початку розробки

					ІАЛЦ.467200.003 ТЗ	Аркуш
						32
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

- Складність розгортання всієї системи
- Всі сервіси повинні весь час перебувати в консистентному стані один з одним

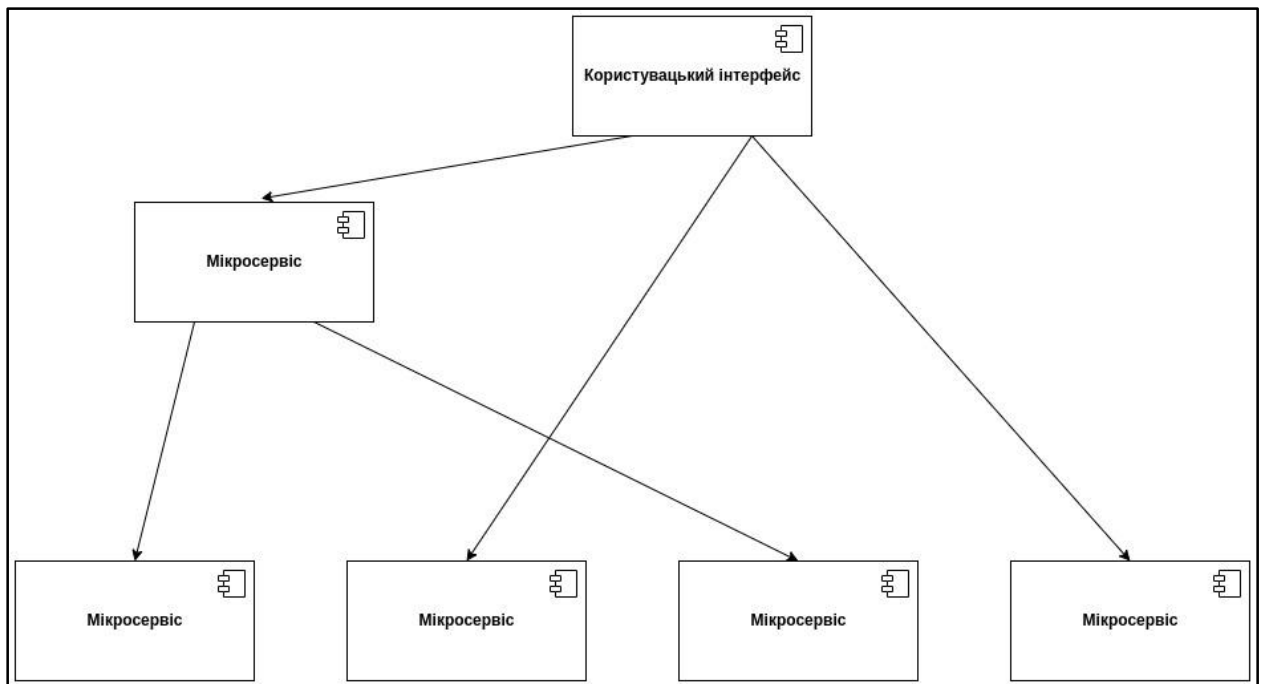


Рис 2.10 Схематичне зображення мікросервісної архітектури

2.7 Вибір оптимальної архітектури для мобільної системи обміну освітнім контентом

Мобільна система обміну освітнім контентом - це загальнодоступна платформа, якою може користуватися будь-який бажаючий. З цього випливають, деякі критерії вибору архітектури.

Система повинна:

- мати високий рівень відмовостійкості
- мати можливість для масштабування, оскільки система може отримати великий приток користувачів
- масштабування процесу розробки, оскільки команда розробки з часом може збільшуватися

Отже, для розроблюваної системи не підійде монолітна архітектура, оскільки вона не має високого рівня відмовостійкості, що є ключовим критерієм для вибору архітектури. Мікросервісна архітектура має більший рівень відмовостійкості та

					ІАЛЦ.467200.003 ТЗ	Аркуш
						33
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

можливість для масштабування, ніш сервіс-орієнтована архітектура. Тому, така архітектура найкраще підходить для веб-додатку.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						34
<i>Изм.</i>	<i>Аркуш.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		

ВИСНОВОК ДО РОЗДІЛУ 2

В цьому розділі було оглянуто основні сучасні архітектури програмного забезпечення, розглянуто їх особливості, переваги та недоліки. Проведено аналіз архітектурних потреб загальнодоступних веб-додатків та мобільних систем обміну освітнім контентом зокрема. В результаті цього аналізу було встановлено, що оптимальним вибором для розроблюваної системи є мікросервісна архітектура.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						35
<i>Изм.</i>	<i>Аркуш.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ

Завдання цього розділу – вибір конкретного технологічного стеку, детальний опис реалізації системи та обґрунтування тих чи інших рішень, зроблених в процесі розробки, а також основні сценарії використання системи.

3.1 Використаний технологічний стек

Ключовим фактором у виборі конкретних технологій у реалізації системи є інтерес автора. Проте всі технології, які вирішено використовувати в цьому проекті, є найкращими або рівноцінними варіантами серед конкурентів.

Серверна частина додатку реалізована за допомогою фреймворка NestJS та мови програмування TypeScript.

NestJS — це платформа для створення ефективних і масштабованих серверних додатків Node.js. Nest забезпечує рівень абстракції вище звичайних фреймворків Node.js, але і надає розробнику API Node.js. Це дає велику свободу використання сторонніх модулів. Причиною обрати NestJS є той факт, що він створює модульну архітектуру проекту, що дозволяє легше створити додаток з низькою зв'язністю. Це спрощує розділення програм на мікросервіси[20].

Для взаємодії між мікросервісами буде використовуватися технологія gRPC. Це сучасна високопродуктивна платформа віддаленого виклику процедур, яка може працювати в будь-якому середовищі. Він використовує транспортний протокол HTTP/2, Protobuf як мову опису інтерфейсу та надає наступні функції: аутентифікація, двонапрямлена потокова передача та керування потоком, блокувальні та неблокувальні зв'язування, а також дедлайни та скасовування[21]. У його складі наявні міжплатформені клієнтські та серверні бібліотеки для багатьох мов. Найбільш поширені сценарії використання включають комунікацію сервісів в мікросервісній архітектурі або підключення клієнтських мобільних додатків до серверних.

PostgreSQL є одним з декількох безкоштовних популярних варіантів СУБД, часто використовується для баз даних веб-додатків. Це була одна з перших розроблених

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						36
Изм.	Аркуш.	№ докум.	Підпис	Дата		

систем управління базами даних, тому в даний час вона добре розвинена, і дозволяє користувачам управляти як структурованими, так і неструктурованими даними[22]. Може бути використана на більшості основних платформ, включаючи Linux. Прекрасно справляється з завданнями імпорту інформації з інших типів баз даних за допомогою власного інструментарію. БД може бути розміщений в ряді середовищ, в тому числі віртуальних, фізичних і хмарних.

Swagger - це фреймворк для специфікації RESTful API. Ціль його використання полягає в тому, що він дає можливість не лише інтерактивно переглядати специфікацію, але й надсилати запити, за допомогою Swagger UI[23].

React - відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими стикаються в розробці односторінкових застосунків[24]. React дозволяє розробникам створювати великі вебзастосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками Як бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими бібліотеками, такими як Redux[25].

3.2 Проектування бази даних

Для проектування, візуалізації, конструювання та документування баз даних використовуються діаграми різних нотацій, в кожній з яких є свої переваги та цілі.

3.2.1 Модель предметної області

На основі сутностей і зв'язків можна побудувати ER-діаграму, що відображатиме інфологічну модель даної предметної області у нотації П. Чена.

Діаграма, яка зображена на рис. 3.1 відображає основні сутності, які будуть використовуватися в системі, їх зв'язки, атрибути та ключі[26].

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						37
Изм.	Аркуш.	№ докум.	Підпис	Дата		

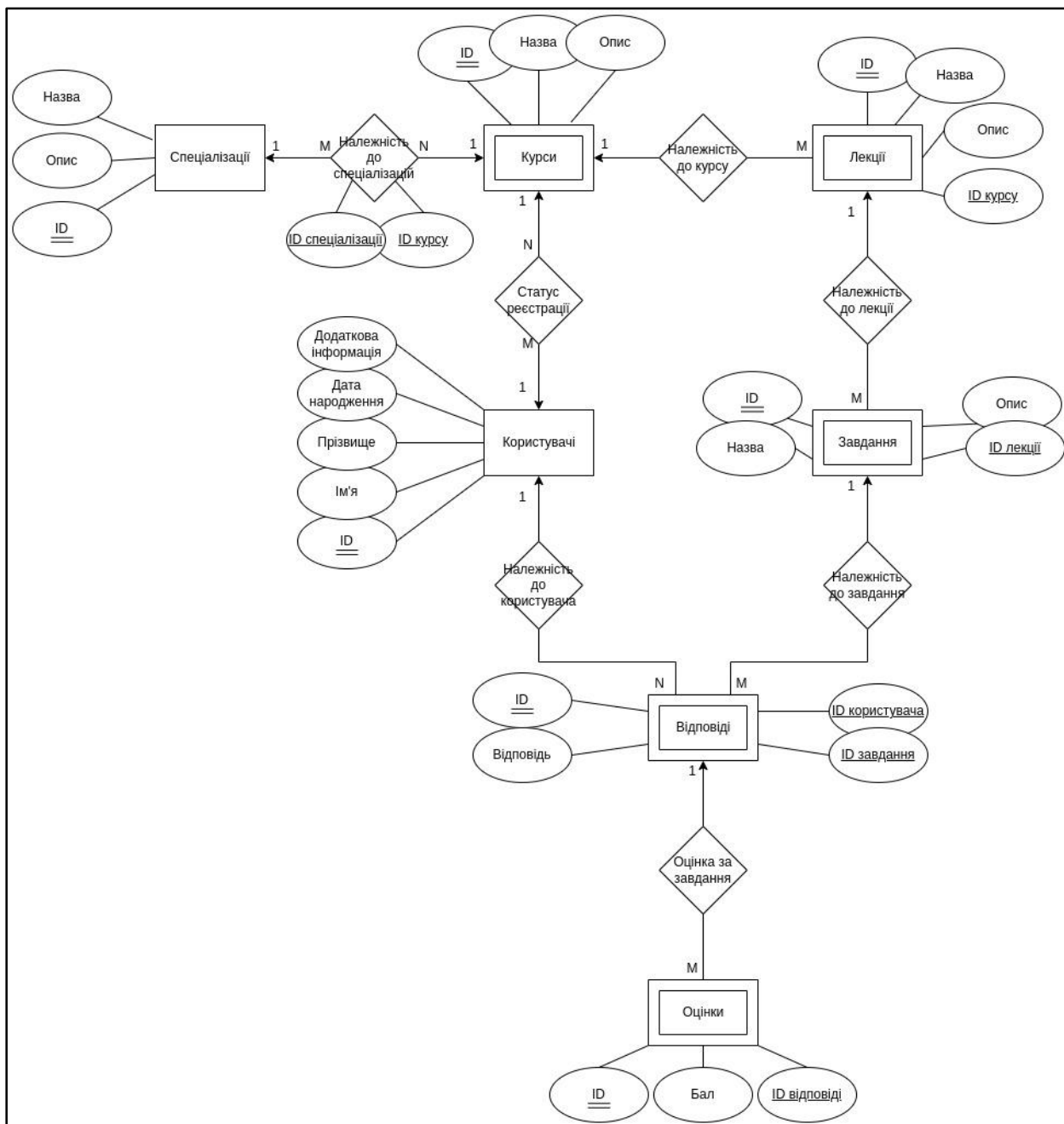


Рис 3.1 Інфологічна модель предметної області

3.2.2 Даталогічна модель

Після того, як визначені взаємозв'язки і залежності між різними сутностями можна впорядкувати дані в даталогічну структуру, яку потім можна відобразити в об'єкти зберігання, які підтримуються сконкретною СУБД. Взаємоз'язки зображаються як атрибути залучених сутностей.

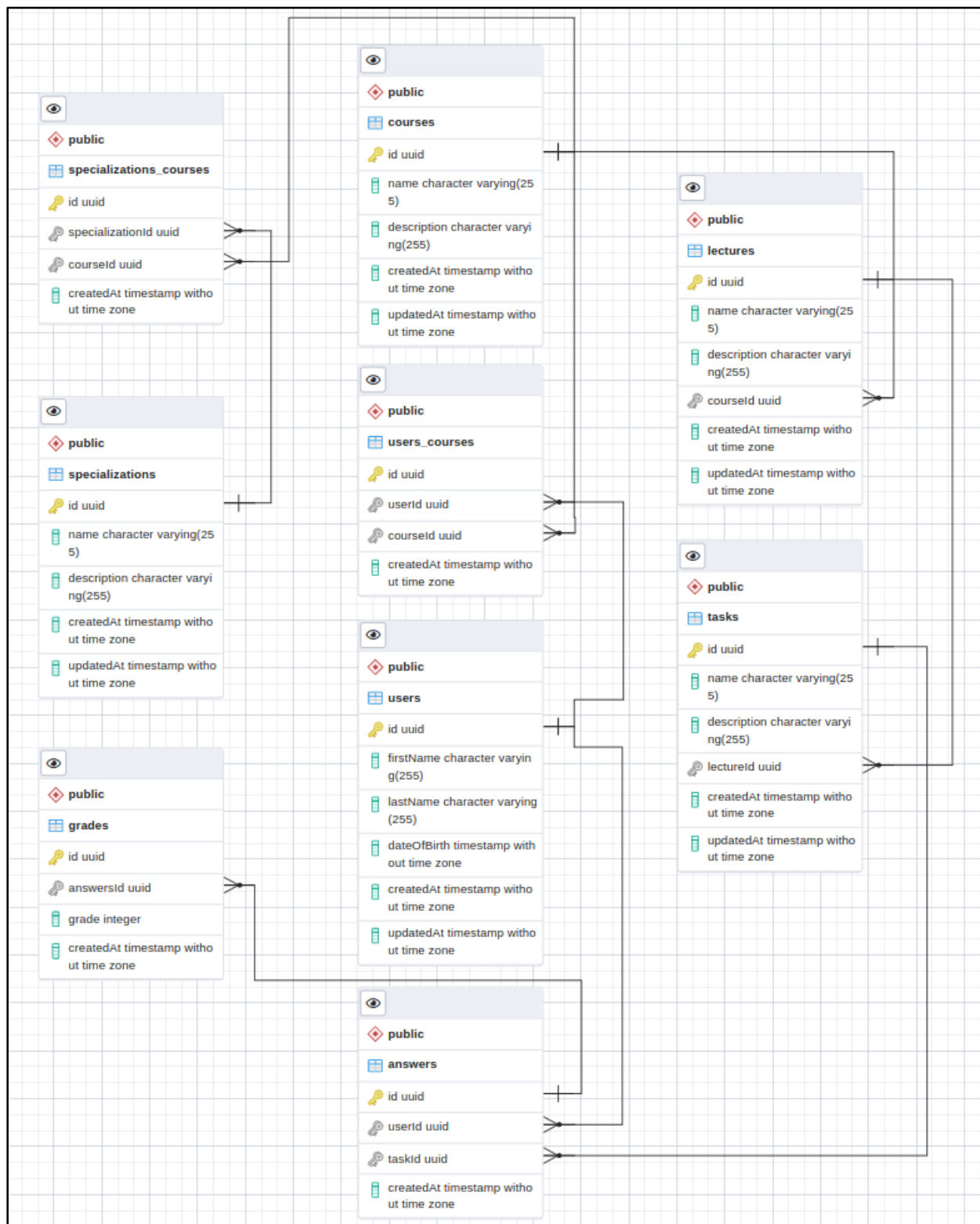


Рис 3.2 Даталогічна модель предметної області

Діаграма, яка зображена на рис. 3.2 відображає даталогічну модель розроблюваної системи.

3.3 Реалізація серверної частини додатку

Серверна частина представляє з себе сукупність декількох проектів, кожен з яких представляє відповідний мікросервіс.

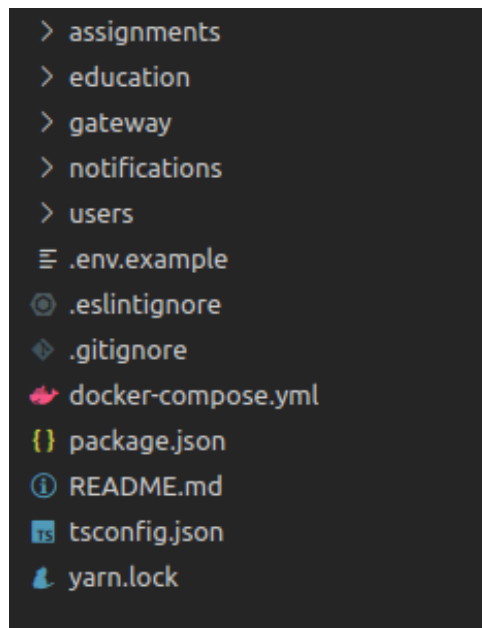


Рис 3.3 Файлова структура

Gateway представляє з себе “вхідну точку” в серверну частину, а також зв’язує між собою всі мікросервіси[27].

Мікросервіси:

- users - реалізує аутентифікацію та авторизацію користувачів, оброблює їхні персональні дані
- education - відповідає за реалізацію логіки пов’язаної з курсами та спеціалізаціями
- assignments - реалізує логіку проходження тестів та оприлюднення результатів
- notifications - надсилає користувачам повідомлення за запитом з інших мікросервісів

Кожен мікросервіс, в свою чергу, представляє з себе окремий проект з власною структурою.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						40
Изм.	Аркуш.	№ докум.	Підпис	Дата		

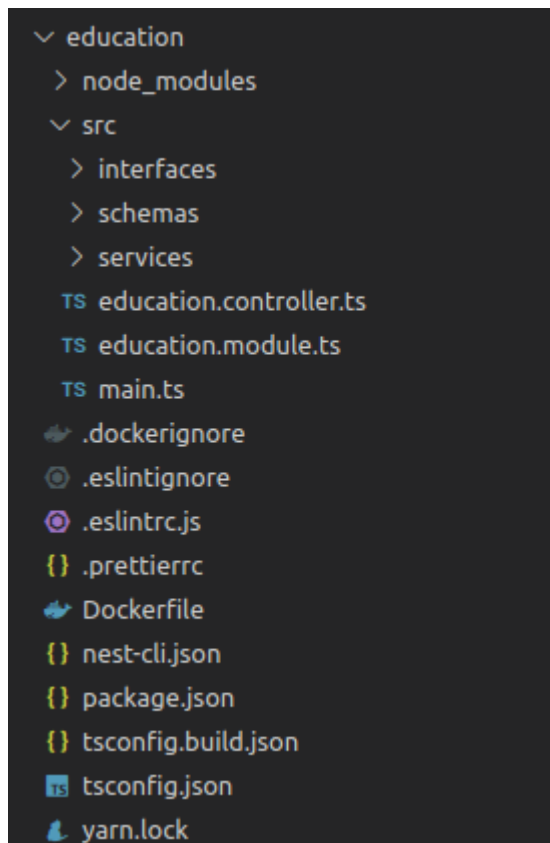


Рис 3.4 Файлова структура мікросервіса education

Кожен з мікросервісів має схожу структуру та архітектуру, тому для прикладу буде розглянутий мікросервіс, який відповідає за реалізацію логіки пов'язаної з курсами та спеціалізаціями.

На рис. 3.5 зображений файл з функцією початкового старту мікросервісу.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						41
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

```

1  import { NestFactory } from '@nestjs/core';
2  import { EducationModule } from '../education.module';
3  import { Transport } from '@nestjs/microservices';
4
5  import { ConfigService } from '../services/config/config.service';
6
7  async function bootstrap() {
8    const app = await NestFactory.createMicroservice(EducationModule, {
9      transport: Transport.GRPC,
10     options: {
11       package: 'education',
12       protoPath: new ConfigService().get('protoPath'),
13     },
14   });
15   await app.listen();
16 }
17 bootstrap();

```

Рис 3.5 Функція старту education мікросервісу

В параметри методу createMicroservice, окрім модуля, передається також і об'єкт з параметрами, де вказується тип транспорту, який буде використовуватися для взаємодії між мікросервісами, в розроблюваній системі це gRPC і назва пакету та шлях до proto файлів[28].

					ІАЛЦ.467200.003 ТЗ	Аркуш
						42
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

```

1  import { Module } from '@nestjs/common';
2  import { KnexModule } from 'nestjs-knex';
3  import { CoursesModule } from '../modules/courses/courses.module';
4  import { LecturesModule } from '../modules/lectures/lectures.module';
5
6  import { ConfigService } from '../services/config/config.service';
7
8  const config = new ConfigService();
9
10 @Module({
11   imports: [
12     KnexModule.forRootAsync({
13       useFactory: () => ({
14         config: {
15           client: 'pg',
16           connection: {
17             host: config.get('databaseHost'),
18             port: config.get('databasePort'),
19             user: config.get('databaseUser'),
20             password: config.get('databasePass'),
21             database: config.get('databaseName'),
22           },
23           pool: {
24             min: config.get('databasePoolMin'),
25             max: config.get('databasePoolMax'),
26           },
27         },
28       }),
29     ),
30     CoursesModule,
31     LecturesModule,
32   ],
33 })
34 export class EducationModule {}

```

Рис 3.6

Оголошення education модуля

Модулі використовуються Nest для організації структури програми в області. Модулі та їхні класи (контролери та провайдери) утворюють граф, який визначає, як Nest виконує ін'єкцію залежностей. На рис. 3.6 зображено оголошення education модуля[29].

Декоратор Module за допомогою ін'єкції залежностей позначає клас як модуль.

Поле imports представляє з себе перелік імпортованих модулів. Це можуть бути як сторонні модулі, встановлені за допомогою пакетного менеджера (npm, yarn тощо), так і локальні. За допомогою цього стає можливим ініціалізація підключення до бази

					ІАЛЦ.467200.003 ТЗ	Аркуш
						43
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

даних використовуючи пакет nestjs-knex, який має в собі інтеграцію конструктора запитів до бази даних[30]. Тому в ініціалізації модуля вказуються параметри підключення до бд, а також параметри пула підключень. Також імпортуються локальні модулі, які відповідають за реалізацію бізнес-логіки пов'язаної з курсами та лекціями (CoursesModule та LecturesModule).

```

1  import { Module } from '@nestjs/common';
2  import { CoursesController } from '../courses.controller';
3  import { CoursesService } from '../courses.service';
4  import { CoursesRepository } from '../courses.repository';
5
6  @Module({
7    controllers: [CoursesController],
8    providers: [CoursesService, CoursesRepository],
9  })
10 export class CoursesModule {}

```

Рис 3.7 Оголошення Courses модуля

Модуль courses відповідає за реалізацію бізнес логіки, пов'язаної з курсами. Цей модуль є дочірнім від модуля education. При оголошенні модуля за допомогою декоратора вказується перелік контролерів модуля. Контролери - це вхідні точки до модуля. А також вказуються провайдери, це класи, які містять в собі бізнес-логіку та роботу з сторонніми сервісами, такі як бази даних та інші.

Клас CoursesRepository відповідає за взаємодію з базою даних та реалізує патерн репозиторій.

Клас CoursesService реалізує основну бізнес логіку, пов'язану з роботою з курсами.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						44
Изм.	Аркуш.	№ докум.	Підпис	Дата		

```

2 import { Controller } from '@nestjs/common';
3 import { GrpcMethod } from '@nestjs/microservices';
4 import { CoursesService } from '../courses.service';
5 import { ICourse } from '../interfaces/course.interface';
6
7 @Controller()
8 export class CoursesController {
9   constructor(private readonly coursesService: CoursesService) {}
10
11   @GrpcMethod('CoursesController', 'FindOne')
12   async findOne(data: { id: string }): Promise {
13     const { id } = data;
14     const result = await this.coursesService.findOne(id);
15     return result;
16   }

```

Рис 3.7 Контролер модуля курсів

Контролер містить в собі набір методів, які є вхідними точками до модуля. В випадку використання gRPC, як основного протоколу передачі даних використовується декоратор GrpcMethod для оголошення методу.

На рис. 3.7 зображений один із методів, реалізованих в системі. Конкретно цей метод відповідає за пошук конкретного курсу в системі за допомогою унікального ідентифікатора. Метод контролера, в свою чергу викликає по ланцюжку викликів метод з сервісу та метод з репозиторію, щоб отримати дані з бази даних.

```

1 import { InjectKnex, Knex } from 'nestjs-knex';
2 import { ICourse } from '../interfaces/course.interface';
3
4 export class CoursesRepository {
5   constructor(@InjectKnex() private readonly knex: Knex) {}
6
7   findOne(id: string): Promise {
8     return this.knex.first().from('courses').where({ id });
9   }

```

Рис 3.8 Репозиторій модуля курсів

За допомогою ін'єкції залежностей репозиторій може отримати доступ до конструктора запитів в базу даних.

Модуль gateway - це сервіс, який є точкою входу в серверну частину для користувача. Він представляє з себе http сервер на який фронтенд частина може робити запити.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						45
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

```

1  import { NestFactory } from '@nestjs/core';
2  import { SwaggerModule, DocumentBuilder } from '@nestjs/swagger';
3  import { AppModule } from './app.module';
4  import { ConfigService } from './services/config/config.service';
5
6  async function bootstrap() {
7    const app = await NestFactory.create(AppModule);
8    const options = new DocumentBuilder()
9      .setTitle('API docs')
10     .addTag('education')
11     .addTag('assignments')
12     .addTag('users')
13     .setVersion('1.0')
14     .build();
15    const document = SwaggerModule.createDocument(app, options);
16    SwaggerModule.setup('api', app, document);
17    await app.listen(new ConfigService().get('port'));
18  }
19  bootstrap();

```

Рис 3.9 Функція старту gateway http серверу

З коду видно, що NestFactory ініціалізує інстанс app з модуля AppModule. Окрім того, створюється документ з документацією за допомогою Swagger[31].

Структура модуля та його складових (контролерів та сервісів) є схожими з модулями мікросервісів. Однак, точкою входу вже є не gRPC метод, а http метод.

```

@Get('/:id')
async getAssignment(
  @User() user: UserType,
  @Param() params: GetAssignmentDto,
): Promise<GetAssignmentResponseDto> {
  const data = await this.assignmentService.get(params.id, user.id);
  return { success: true, data };
}

```

Рис 3.10 Http метод для отримання завдання для користувача

3.4 Реалізація кінцевих точок

Таким чином, були реалізовані наступні кінцеві точки:

- *GET /api/specializations* - отримання списку спеціалізацій, доступна пагінація, сортування, фільтрація за допомогою параметрів page, perPage, sortBy, sordDirection, filter. Повертає статус-код 200 з списком спеціалізацій.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						46
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

- *GET /api/specializations/:id* - отримання конкретної спеціалізації. Повертає статус-код 200 якщо спеціалізація доступна для перегляду та наявна в системі. Якщо ні - повертає статус-код 404.
- *GET /api/specializations/:id/participants* - отримання списку учасників спеціалізації. Доступна пагінація, сортування, фільтрація за допомогою параметрів *page*, *perPage*, *sortBy*, *sortDirection*, *filter*. Повертає статус-код 200 з списком учасників. Якщо активних учасників немає в системі - повертає пустий масив. Якщо спеціалізації немає в системі з заданим унікальним ідентифікатором - то повертає статус-код 404.
- *GET /api/specializations/:id/participants/count* - отримання кількості учасників в спеціалізації. Повертає статус-код 200 з кількістю активних учасників. Якщо спеціалізації немає в системі з заданим унікальним ідентифікатором - то повертає статус-код 404.
- *POST /api/specializations* - створення нової спеціалізації. У випадку успішного створення - повертає статус-код 200. Якщо запит не виконався по провінці клієнтської частини системи, то повертається статус-код 400.
- *PUT /api/specializations/:id* - оновлення даних спеціалізації. Може бути виконаний лише користувачем, який створив спеціалізацію. У випадку успішного оновлення - повертає статус-код 200. Якщо запит не виконався по провінці клієнтської частини системи, то повертається статус-код 400. Якщо спеціалізації немає в системі з заданим унікальним ідентифікатором - то повертає статус-код 404.
- *PUT /api/specializations/:id/register* - реєстрація в спеціалізації. Може бути виконаний лише користувачем, який ще не зареєстрований. У випадку успішного оновлення - повертає статус-код 200. Якщо спеціалізації немає в системі з заданим унікальним ідентифікатором - повертає статус-код 404.
- *DELETE /api/specializations/:id* - видалення спеціалізації. Може бути виконаний лише користувачем, який створив спеціалізацію та якщо в спеціалізації немає активних студентів. У випадку успішного видалення - повертає статус-код 200.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						47
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

Якщо спеціалізації немає в системі з заданим унікальним ідентифікатором - то повертає статус-код 404.

- *GET /api/courses* - отримання списку курсів, доступна пагінація, сортування, фільтрація за допомогою параметрів *page*, *perPage*, *sortBy*, *sordDirection*, *filter*. Повертає статус-код 200 з списком спеціалізацій.
- *GET /api/courses/:id* - отримання конкретного курсу. Повертає статус-код 200 якщо курс доступний для перегляду та наявний в системі. Якщо ні - повертає статус-код 404.
- *GET /api/courses/:id/participants* - отримання списку учасників курсу. Доступна пагінація, сортування, фільтрація за допомогою параметрів *page*, *perPage*, *sortBy*, *sordDirection*, *filter*. Повертає статус-код 200 з списком учасників. Якщо активних учасників немає в системі - повертає пустий масив. Якщо курсу немає в системі з заданим унікальним ідентифікатором - то повертає статус-код 404.
- *GET /api/courses/:id/participants/count* - отримання кількості учасників курсу. Повертає статус-код 200 з кількістю активних учасників. Якщо курсу немає в системі з заданим унікальним ідентифікатором - то повертає статус-код 404.
- *GET /api/courses/:id/search/:text* - отримання списку курсів за допомогою пошуку по назві, доступна пагінація, сортування, фільтрація за допомогою параметрів *page*, *perPage*, *sortBy*, *sordDirection*, *filter*. Повертає статус-код 200 з списком курсів.
- *GET /api/courses/:courseId/lectures/:lectureId* - отримання посилання на матеріали лекції.
- *GET /api/courses/:courseId/lectures/assignments* - отримання завдання після лекції.
- *GET /api/courses/:courseId/lectures/results* - отримання результатів після виконання завдання.
- *POST /api/courses* - створення нового курсу. У випадку успішного створення - повертає статус-код 200. Якщо запит не виконався по провіні клієнтської частини системи, то повертається статус-код 400.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						48
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

- *POST /api/courses/:courseId/lectures/assignments* - надсилання відповідей до завдання.
- *PUT /api/courses/:id* - оновлення даних курсу. Може бути виконаний лише користувачем, який створив курс. У випадку успішного оновлення - повертає статус-код 200. Якщо запит не виконався по провині клієнтської частини системи, то повертається статус-код 400. Якщо курсу немає в системі з заданим унікальним ідентифікатором - то повертає статус-код 404.
- *PUT /api/courses/:id/register* - реєстрація в курс. Може бути виконаний лише користувачем, який ще не зареєстрований. У випадку успішного оновлення - повертає статус-код 200. Якщо курсу немає в системі з заданим унікальним ідентифікатором - повертає статус-код 404.
- *DELETE /api/courses/:id* - видалення курсу. Може бути виконаний лише користувачем, який створив курс та якщо в курсі немає активних студентів. У випадку успішного видалення - повертає статус-код 200. Якщо спеціалізації немає в системі з заданим унікальним ідентифікатором - то повертає статус-код 404.
- *GET /api/users/me* - повернення інформації про себе. Доступно лише авторизованим користувачам, в іншому випадку повертає 403 статус-код. При успішному запиті повертає 200 статус-код.
- *GET /api/users/:id* - повернення інформації про користувача. При успішному запиті повертає 200 статус-код, якщо користувач не знайдено в системі, то повертається 404 статус-код.
- *GET /api/users/:id/courses* - повернення списку активних курсів користувача. При успішному запиті повертає 200 статус-код, якщо користувач не знайдено в системі, то повертається 404 статус-код.
- *GET /api/users/:id/specializations* - повернення списку активних спеціалізацій користувача. При успішному запиті повертає 200 статус-код, якщо користувач не знайдено в системі, то повертається 404 статус-код.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						49
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

- *PUT /api/users/me* - оновлення особистих даних в системі. Доступно лише авторизованим користувачам, в іншому випадку повертає 403 статус-код. При успішному запиті повертає 200 статус-код.
- *DELETE /api/users/me* - видалення акаунту в системі. Доступно лише авторизованим користувачам, в іншому випадку повертає 403 статус-код. При успішному запиті повертає 200 статус-код.

3.4 Реалізація клієнтської частини додатку

Клієнтська частина реалізована за допомогою фронт-енд фреймворка React і представляє з себе SPA(Single Page Application)[32].

Клієнтський додаток поділений на 6 основних компонентів:

- Домашня сторінка
- Спеціалізація
- Курс
- Лекція
- Завдання
- Результати

Кожен компонент відповідає за відображення контенту, для написання бізнес логіки та збереження даних використовується бібліотека *redux*. Для імplementації навігації на клієнтській стороні використовується бібліотека *react-router-dom*. За її допомоги було створено такі шляхи (routes):

- *'/home'* - компонент: *HomeComponent*. Відкриває домашню сторінку системи.
- *'/specializations'* - компонент: *SpecializationsComponent*. Відкриває сторінку з доступними спеціалізаціями. Фронтенд отримує дані з бекенду *GET /api/specializations*. Користувач має можливість фільтрувати дані, робити пошук тощо.
- *'/specializations/:id'* - компонент: *SpecializationComponent*. Відкриває сторінку з спеціалізацією. Користувач має можливість бачити курси в спеціалізації, кількість учасників.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						50
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

- *'/courses'* - компонент: *CoursesComponent*. Відкриває сторінку з доступними курсами. Фронтенд отримує дані з бекенду *GET /api/courses*. Користувач має можливість фільтрувати дані, робити пошук тощо.
- *'/courses/:id'* - компонент: *CourseComponent*. Відкриває сторінку з курсом. Користувач має можливість бачити свій статус реєстрації в курсі, прогрес, кількість учасників тощо.
- *'/courses/:courseId/lectures/:lectureId'* - компонент: *LectureComponent*. Відкриває сторінку з лекцією. Є можливість переглянути лекцію.
- *'/courses/:courseId/lectures/:lectureId/assignments'* - компонент: *AssignmentComponent*. Відкриває сторінку з завданнями після лекції.
- *'/courses/:courseId/lectures/:lectureId/results'* - компонент: *ResultsComponent*. Відкриває сторінку з результатами завдання після лекції.
- *'/'* - перенаправлення на *'/home'*

					ІАЛЦ.467200.003 ТЗ	Аркуш
						51
Изм.	Аркуш.	№ докум.	Підпис	Дата		

ВИСНОВОК ДО РОЗДІЛУ 3

В даному розділі були розглянуті технології, що використовувались при створенні мобільної системи обміну освітнім контентом. Спроектowana схема бази даних, а саме, створена інфологічна та даталогічна моделі бази даних.

Були представлені конкретні технічні рішення, застосовані при створенні системи. Представлена файлова структура системи. Описаний процес всіх етапів розробки системи та представлені відповідні діаграми та рисунки.

Розроблена система повністю відповідає описаній в 2 розділі архітектурі.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						52
<i>Изм.</i>	<i>Аркуш.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		

РОЗДІЛ 4. ОГЛЯД РОЗРОБЛЕНОЇ СИСТЕМИ

В даному розділі представлений розгляд розробленої системи. Та зроблений загальний огляд основних функціональних особливостей.

Розроблена система являє собою веб-додаток, орієнтований на мобільні пристрої.

Вхідною точкою в додаток є вітальна сторінка, яка зображена на рис. 4.1.

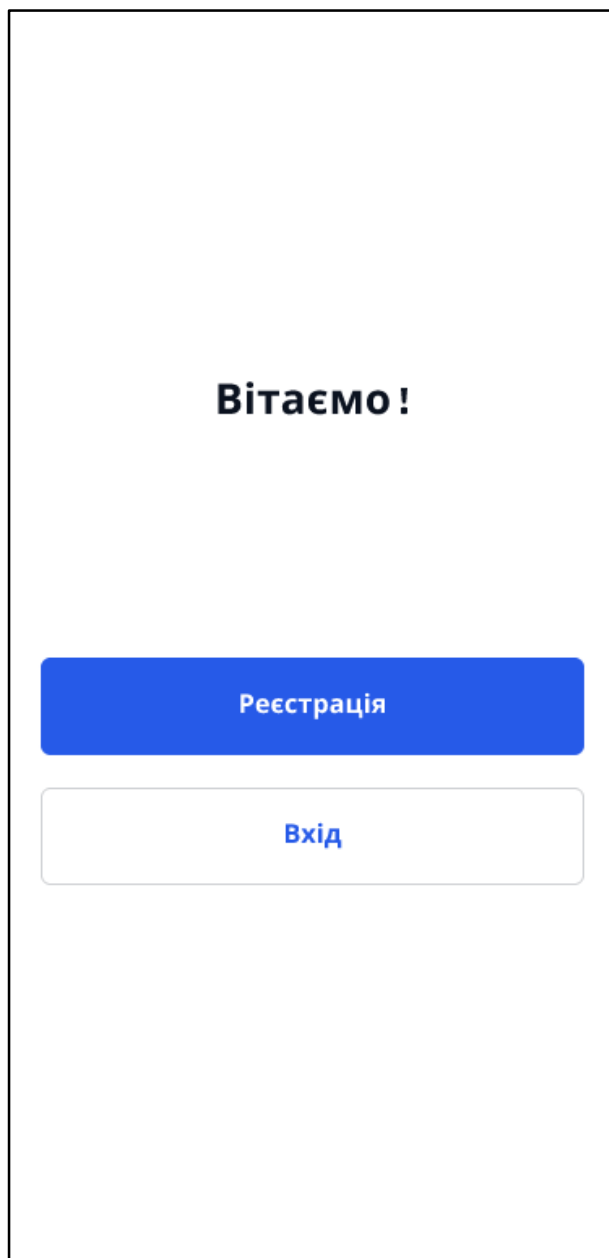


Рис 4.1 Вітальна сторінка

На ньому користувач має змогу перейти на сторінки реєстрації або входу. При натисканні на кнопку реєстрації користувач переходить на сторінку реєстрації, яка зображена на рис. 4.2.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						53
Изм.	Аркуш.	№ докум.	Підпис	Дата		

Вкажіть деталі

ім'я користувача

email адреса

Далі

Рис 4.2 Сторінка реєстрації

Користувач повинен вказати власні дані для реєстрації. Якщо ж користувач обрав опцію входу, то він направляється на сторінку входу, вона зображена на рис. 4.3.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						54
Изм.	Аркуш.	№ докум.	Підпис	Дата		

Вітаємо!

ім'я користувача

Вхід

Рис 4.3 Сторінка входу

Після успішного входу або реєстрації користувач направляється на головну сторінку.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						55
Изм.	Аркуш.	№ докум.	Підпис	Дата		

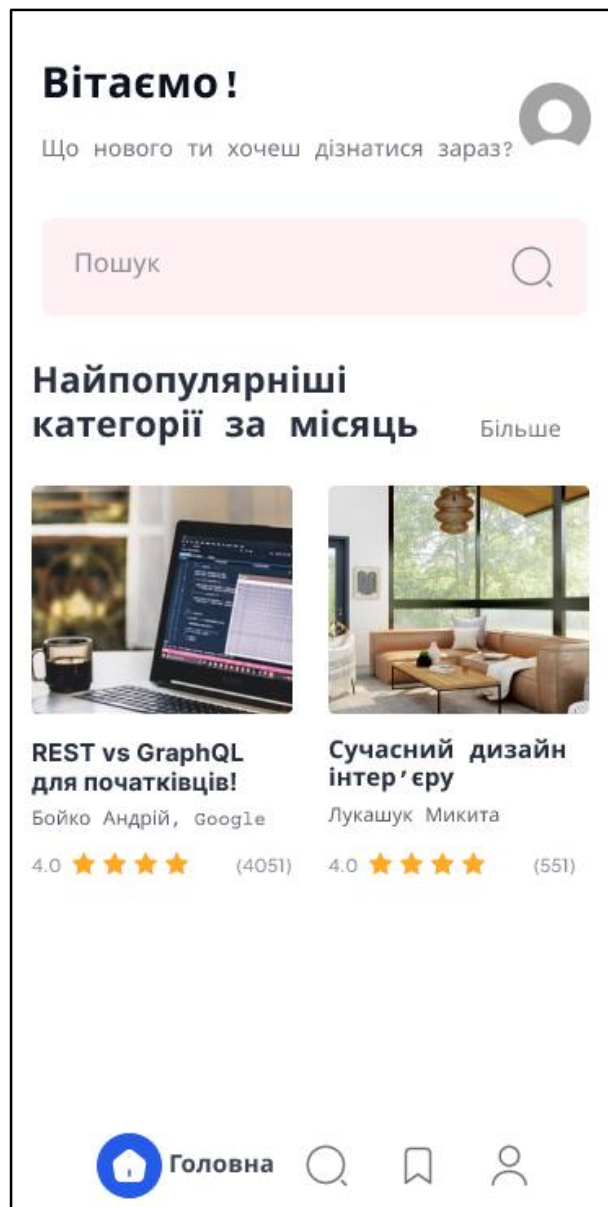


Рис 4.4 Головна сторінка

На головній сторінці користувач має можливість побачити найпопулярніші категорії за місяць, перейти до пошуку, або ж власного профіля.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						56
Изм.	Аркуш.	№ докум.	Підпис	Дата		

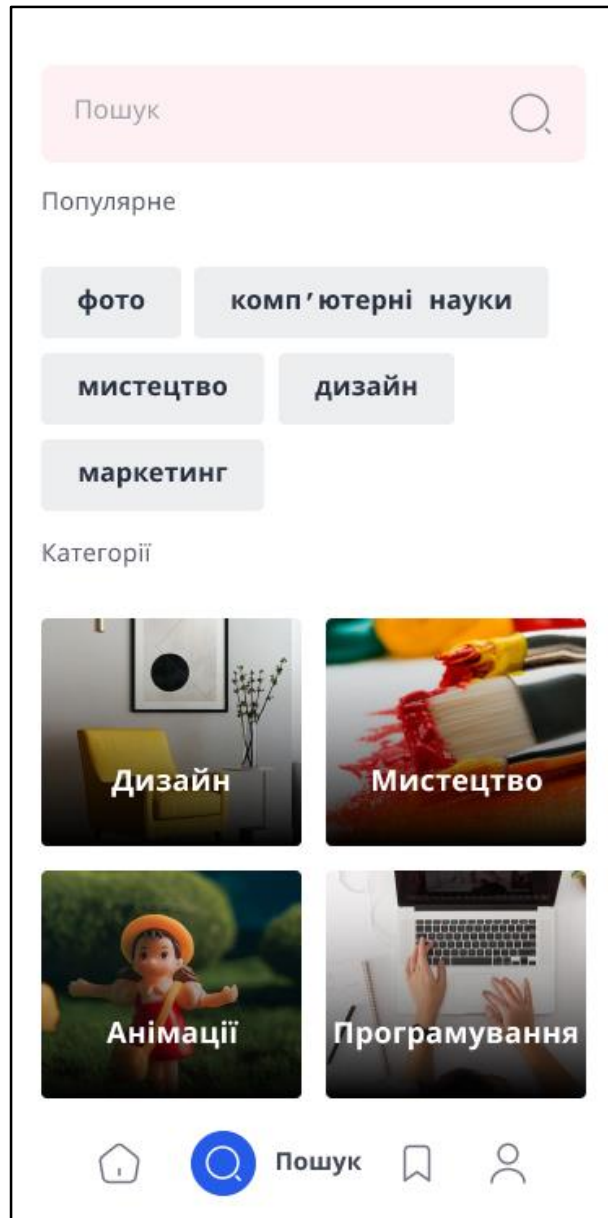


Рис 4.5 Сторінка пошуку

На сторінці пошуку користувачу пропонуються найбільш популярні варіанти пошуку, можливість зробити пошук за категоріями, або ж здійснити простий пошук. Приклад сторінки пошуку зображено на рис. 4.5.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						57
Изм.	Аркуш.	№ докум.	Підпис	Дата		

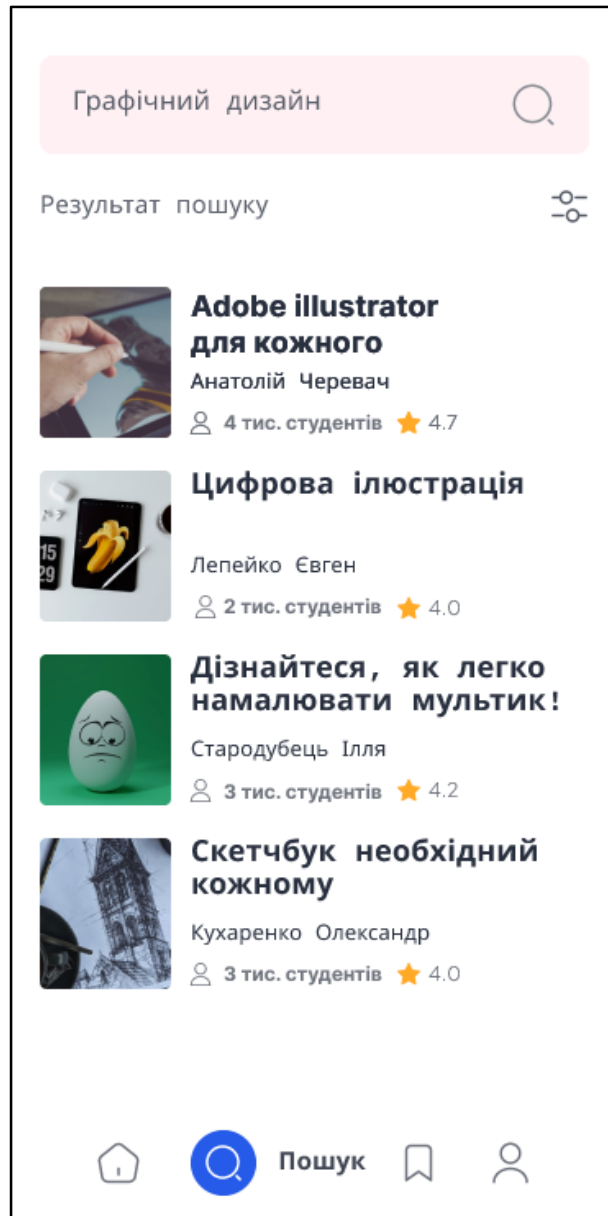


Рис 4.7 Сторінка результатів пошуку

Після здійснення пошуку на сторінці відображаються доступні варіанти курсів, які співпадають з пошуком, як зображено на рис. 4.7. Після цього, користувач має змогу перейти на будь який з них.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						59
Изм.	Аркуш.	№ докум.	Підпис	Дата		



Рис 4.8 Сторінка курсу

При переході на сторінку курсу з'являється коротка інформація про курс, а також два вікна:

- більш детальний опис
- перелік лекцій

					ІАЛЦ.467200.003 ТЗ	Аркуш
						60
Изм.	Аркуш.	№ докум.	Підпис	Дата		



Рис 4.9 Сторінка курсу з детальним описом

На рис. 4.9 зображена сторінка курсу з детальним описом, у користувача є можливість розгорнути текст.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						61
Изм.	Аркуш.	№ докум.	Підпис	Дата		

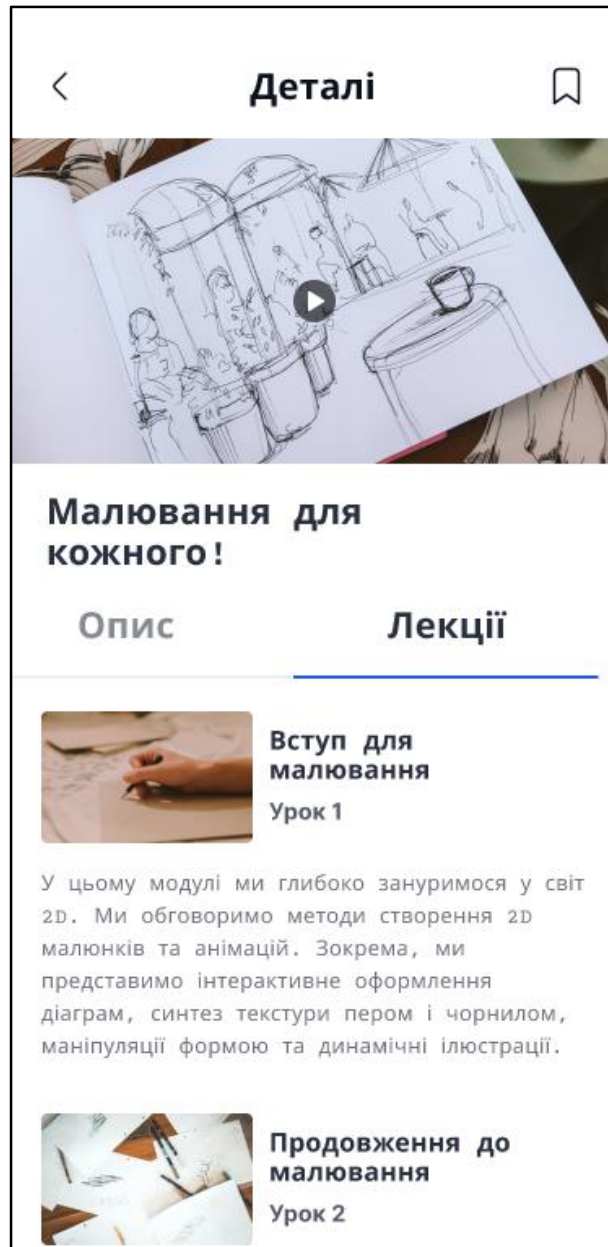


Рис 4.9 Сторінка курсу з переліком лекцій

Користувач може перейти на сторінку власного профіля та змінити особисті дані.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						62
Изм.	Аркуш.	№ докум.	Підпис	Дата		

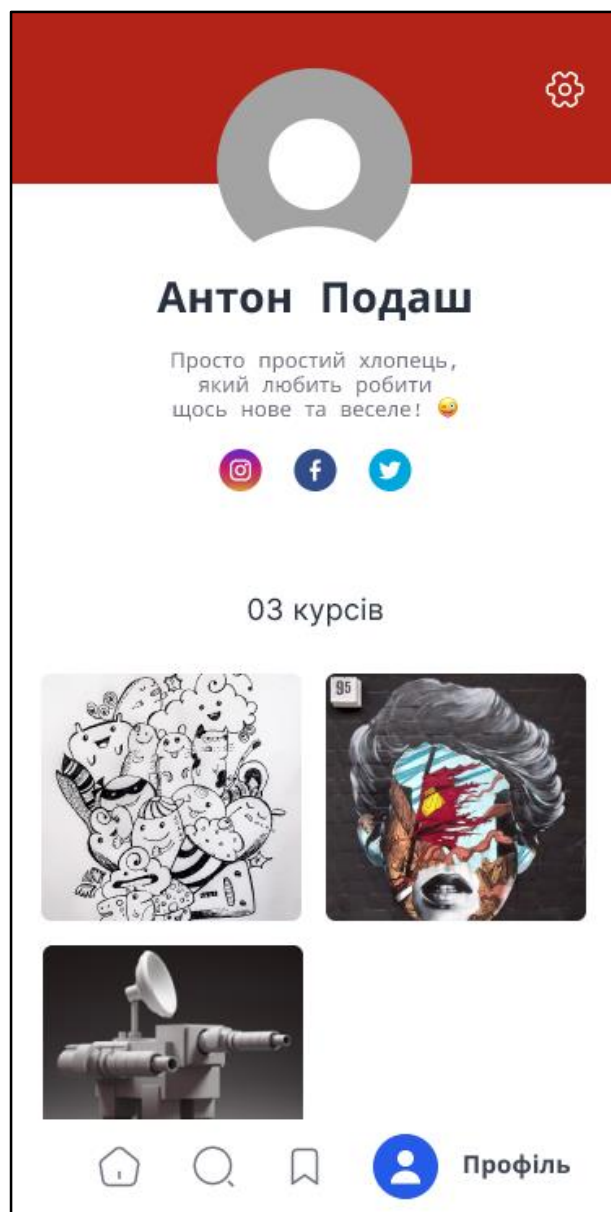


Рис 4.10 Сторінка особистого профіля

На рис 4.10 зображена сторінка особистого профіля користувача, де він може переглянути курси, в яких він навчається.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						63
Изм.	Аркуш.	№ докум.	Підпис	Дата		

ВИСНОВОК ДО РОЗДІЛУ 4

Оглянуто розроблену систему та встановлено що вона працює згідно з встановленими для неї вимогами.

Дана система надає такі функціональні можливості:

- пошук лекцій, курсів та спеціалізацій
- реєстрація в курс та спеціалізацію
- перегляд лекцій
- завантаження додаткових матеріалів
- відстеження прогресу навчання

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						64
Изм.	Аркуш.	№ докум.	Підпис	Дата		

ВИСНОВКИ

В рамках даної роботи проведено аналіз існуючих мобільних систем обміну освітнім контентом. Виявлені ключові функціональні особливості. Також результатом аналізу стало виявлення недоліків, що можуть обмежувати тим чи іншим чином використання цих систем.

Наступною метою роботи стала розробка альтернативної мобільної системи обміну освітнім контентом. Були розглянуті існуючі типи архітектур, виявлені їхні переваги та недоліки та згідно них обрана найбільш відповідна для розроблюваної системи. В результаті була обрана мікросервісна архітектура.

Наступним етапом стала власне розробка системи. Вона поділялась на два ключових етапи:

- розробка серверної частина
- розробка клієнтської частини

Систему було імплементовано за допомогою NestJS та React. Розроблена система повністю виконала поставлені перед нею задачі.

Протягом виконання завдання були використані та поглиблені навички і знання у сфері розробки програмного забезпечення.

					ІАЛЦ.467200.003 ТЗ	Аркуш
						65
Изм.	Аркуш.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Richards M. Fundamentals of software architecture: an engineering approach / Mark Richards, Neal Ford. – [Б. м.] : O'Reilly Media, 2020. – 432 с.
2. Kazman R. Designing software architectures: a practical approach / Rick Kazman, Humberto Cervantes. – [Б. м.] : Addison-Wesley Professional, 2016. – 320 с.
3. Different types of elearning platforms and methods - benefits & key players [Електронний ресурс] // FATbit Blog. – Режим доступу: <https://www.fatbit.com/fab/different-types-of-elearning-platforms-and-methods/>.
4. Top 10 elearning features that everyone should know about [Електронний ресурс] // eLearning Industry. – Режим доступу: <https://elearningindustry.com/elearning-features-top-everyone-know>.
5. Coursera`s mission, vision, and commitment to our community | coursera [Електронний ресурс]. – Режим доступу: <https://about.coursera.org/>.
6. Classroom | google for education [Електронний ресурс] // Google for Education. – Режим доступу: <https://edu.google.com/workspace-for-education/classroom/>.
7. Janssens T. Tom Janssens | The goal of software architecture [Електронний ресурс] / Tom Janssens. – Режим доступу: <https://tojans.me/posts/the-goal-of-software-architecture/>.
8. Learning UML 2.0: A pragmatic introduction to UML. – [Б. м.] : O'Reilly, 2006. – 228 с.
9. What is activity diagram? [Електронний ресурс] // Ideal Modeling & Diagramming Tool for Agile Team Collaboration. – Режим доступу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/>.
10. Component diagrams - see examples, learn what they are [Електронний ресурс] // SmartDraw - Create Flowcharts, Floor Plans, and Other Diagrams on Any Device. – Режим доступу: <https://www.smartdraw.com/component-diagram/>.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						66
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

11. What is package diagram? [Електронний ресурс] // Ideal Modeling & Diagramming Tool for Agile Team Collaboration. – Режим доступу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/>.
12. Language-based systems - claude pasquier [Електронний ресурс] // Claude Pasquier. – Режим доступу: <http://www.pasquier.net/research/language-based-systems>.
13. Main program and subroutine [Електронний ресурс] // Ivan's Blog. – Режим доступу: <https://ivanbrygar.com/2021/10/31/main-program-and-subroutine/>.
14. Henrich A. Document retrieval facilities for repository-based system development environments [Електронний ресурс] / Andreas Henrich // The 19th annual international ACM SIGIR conference, Zurich, Switzerland, 18–22 серп. 1996 р. – New York, New York, USA, 1996. – Режим доступу: <https://doi.org/10.1145/243199.243237>.
15. What is a multi layered software architecture? | Packt Hub [Електронний ресурс] // Packt Hub. – Режим доступу: <https://hub.packtpub.com/what-is-multi-layered-software-architecture/>.
16. Awati R. What is monolithic architecture in software? [Електронний ресурс] / Rahul Awati, Ivy Wigmore // WhatIs.com. – Режим доступу: <https://www.techtarget.com/whatis/definition/monolithic-architecture>.
17. SOA [Електронний ресурс] // IBM. – Режим доступу: <https://www.ibm.com/cloud/learn/soa>.
18. What are microservices? [Електронний ресурс] // microservices.io. – Режим доступу: <https://microservices.io/>.
19. What are microservices? | AWS [Електронний ресурс] // Amazon Web Services, Inc. – Режим доступу: <https://aws.amazon.com/ru/microservices/>.
20. NestJS - A progressive Node.js framework [Електронний ресурс]. – Режим доступу: <https://nestjs.com/>.
21. gRPC [Електронний ресурс]. – Режим доступу: <https://grpc.io/>.
22. PostgreSQL: a closer look at the object-relational database management system [Електронний ресурс] // IONOS Digitalguide. – Режим доступу: <https://www.ionos.com/digitalguide/server/know-how/postgresql/>.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						67
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

- 23.What is swagger [Електронний ресурс] // API Documentation & Design Tools for Teams | Swagger. – Режим доступу: <https://swagger.io/docs/specification/2-0/what-is-swagger/>.
- 24.React – JavaScript-бібліотека для створення користувацьких інтерфейсів [Електронний ресурс] // React – JavaScript-бібліотека для створення користувацьких інтерфейсів. – Режим доступу: <https://uk.reactjs.org/>.
- 25.Redux - A predictable state container for javascript apps. | redux [Електронний ресурс]. – Режим доступу: <https://redux.js.org/> (дата звернення: 12.06.2022).
- 26.ER-диаграммы в нотации Чена [Електронний ресурс] // Studme. – Режим доступу: https://studme.org/93800/informatika/er-diagramy_notatsii_chena.
- 27.Microservices Pattern: API gateway pattern [Електронний ресурс] // microservices.io. – Режим доступу: <https://microservices.io/patterns/apigateway.html>.
- 28.Documentation | NestJS - A progressive Node.js framework [Електронний ресурс] // Documentation | NestJS - A progressive Node.js framework. – Режим доступу: <https://docs.nestjs.com/fundamentals/lifecycle-events>.
- 29.Installation | knex.js [Електронний ресурс] // SQL Query Builder for Javascript | Knex.js. – Режим доступу: <https://knexjs.org/guide/#configuration-options>.
- 30.Documentation | NestJS - A progressive Node.js framework [Електронний ресурс] // Documentation | NestJS - A progressive Node.js framework. – Режим доступу: <https://docs.nestjs.com/modules#modules>.
- 31.Documentation | NestJS - A progressive Node.js framework [Електронний ресурс] // Documentation | NestJS - A progressive Node.js framework. – Режим доступу: <https://docs.nestjs.com/openapi/introduction>.
- 32.React | Введение и первое приложение [Електронний ресурс] // METANIT.COM - Сайт о программировании. – Режим доступу: <https://metanit.com/web/react/1.1.php>.

					<i>ІАЛЦ.467200.003 ТЗ</i>	Аркуш
						68
Ізм.	Аркуш.	№ докум.	Підпис	Дата		

ДОДАТОК 1

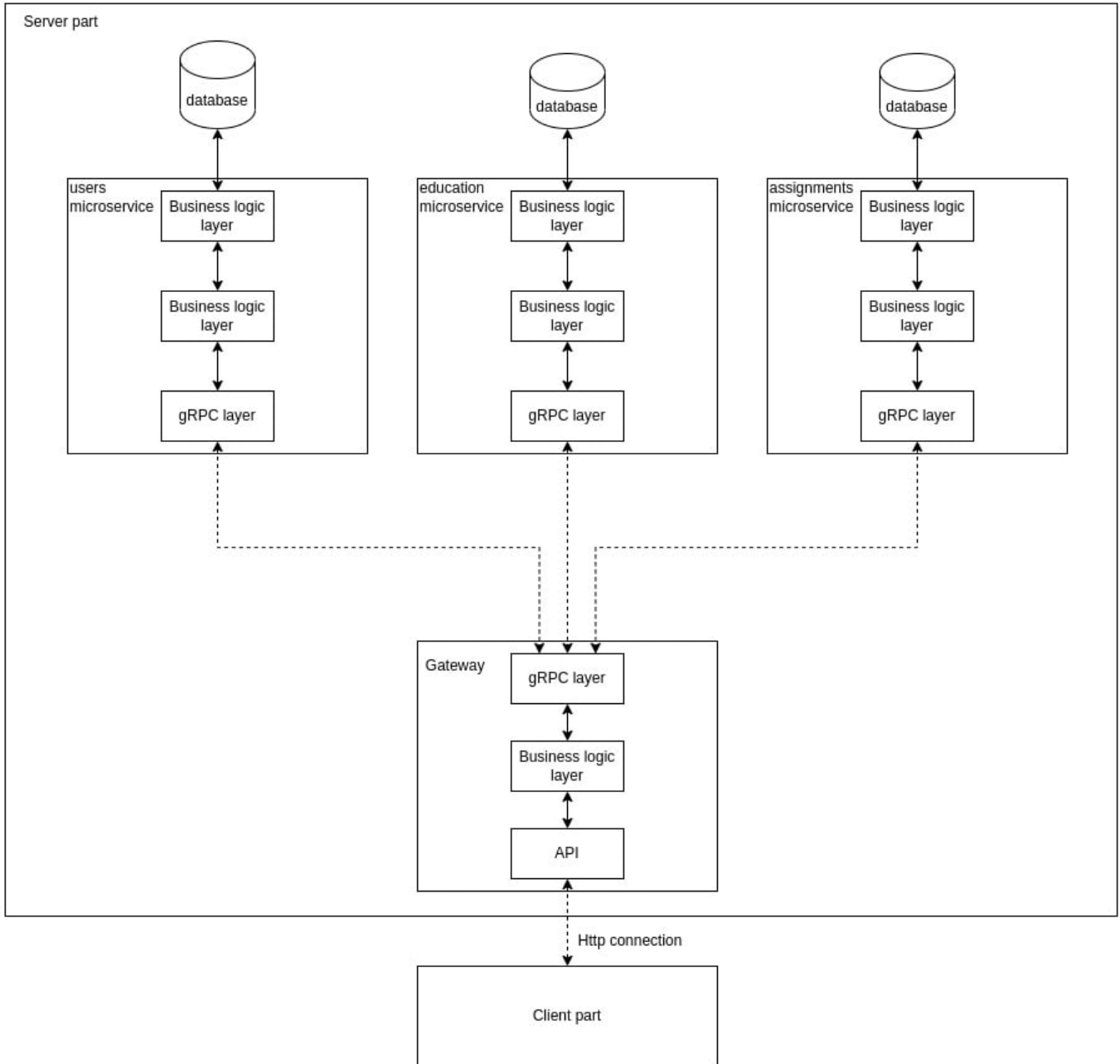
Мобільна система обміну освітнім контентом

Структурна схема системи

ІАЛЩ.467200.004 Д1

Аркушів 1

Київ 2022



					ІАЛЦ.467200.003 ТЗ			
					Мобільна система обміну освітнім контентом Структурна схема даних	Лім.	Маса	Маси.
Змн	Арк.	№ докум.	Підпис	Дата				
Розробив		Чугай Я. В.						
Перевір.		Сергієнко А. М.						
Т. контр.								
						Аркуш		Аркушів
					НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-83	1		1
Н. контр.		Симоненко В. П.						
Затверд.								

ДОДАТОК 2

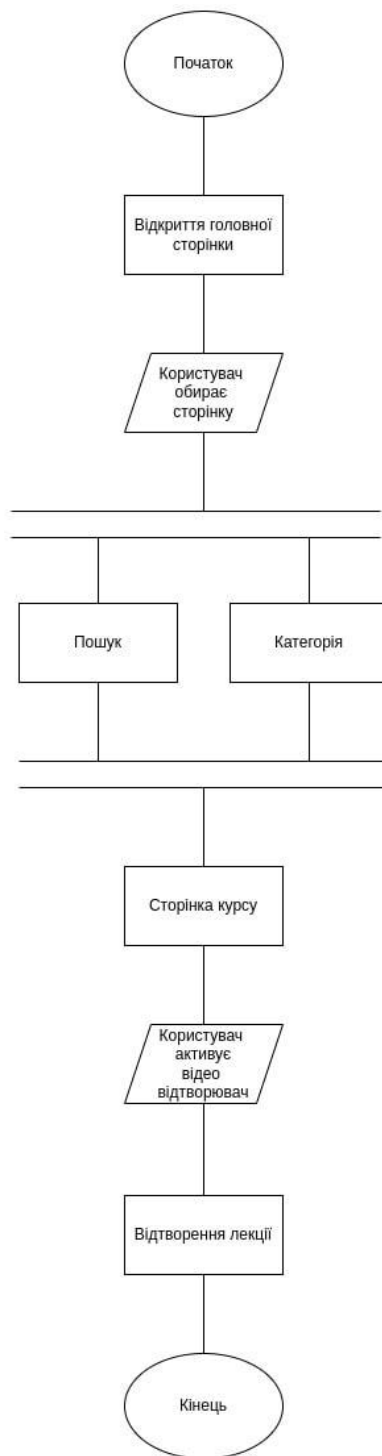
Мобільна система обміну освітнім контентом

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 ДЗ

Аркушів 1

Київ 2022



					ІАЛЦ.467200.003 ТЗ			
					Мобільна система обміну освітнім контентом Алгоритм дії ПЗ	Лім	Маса	Маси.
Змн	Арк.	№ докум.	Підпис	Дата				
Розробив		Чугай Я. В.						
Перевір.		Сергієнко А. М.						
Т. контр.						Аркуш	Аркушів	
					НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-83	1	1	
Н. контр.		Симоненко В. П.						
Затверд.								

ДОДАТОК 3

Мобільна система обміну освітнім контентом

Функціональна схема

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2022

