КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра
за спеціальністю 121 Інженерія програмного забезпечення
на тему:
РОЗРОБКА НАВЧАЛЬНОЇ ПЛАТФОРМИ ДЛЯ ТЕСТУВАННЯ ЗНАНЬ

Виконав студент 4-го курсу
Поліна НЕПОЧАТОВА

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Євгеній ІВАНОВ

(підпис)

Засвідчую, що в цій курсовій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студентка

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
« 29 » травня 2023 р.,
протокол № 11 Завідувач
кафедри Олександр
ПРОВОТАР

(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 56 (додатки – 2 сторінки) сторінки, 15 зображень, 7 таблиці, 29 джерел посилань, 2 додатки.

JAVA, POSTGRESQL, QUARKUS, БЕКЕНД, ВЕБ-ЗАСТОСУНОК, ВЕБ-СЕРВІС, НАВЧАЛЬНА ПЛАТФОРМА.

Об'єкт дослідження: дослідження розробки веб-сервісів. Об’єктом розробки є прототип навчальної платформи для тестування знань, використовуючи мову розробки Java та фреймворк Quarkus для серверної частини. Мета роботи: ознайомлення зі сферою розробки веб-сервісів для тестування знань, аналіз сучасних технологій веб-розробки, розробка прототипу веб-сервіса для тестування знань.

Методи та інструменти розробки: мова програмування Java, фреймворк Quarkus, об'єктно-реляційна база даних PostgreSQL, програмне забезпечення для контейнеризації Docker, середовище розробки Intellij Idea.

Результати роботи: проведено аналіз предметної області та існуючих альтернатив, спроектована архітектура проекту, розроблено вимоги до системи, реалізована серверна частина для актора Викладач та сервіс для роботи з Базою Даних.

Сфера застосування: веб-сервіс може використовуватись у повсякденному житті, а також для комерційного застосування у сферах, що потребують інструмент для проведення тестування знань.

Значимість роботи і пропозиції щодо розвитку: на даному етапі розробки програмне рішення реалізує базовий функціонал, який демонструє основний фундамент, на який можна буде добудувати інший функціонал. В майбутньому веб-сервіс можна покращити та розвинути ідею кваліфікаційної роботи до розмірів реального затребуваного стартапу.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API – Програмний інтерфейс застосунку;

IDE – Integrated Design Environment, інтегроване середовище розробки;

HTML – HyperText Markup Language;

HTTP – HyperText Transfer Protocol;

СУБД – Система Управління Базами Даних;

БД – База Даних;

SQL – Structured Query Language;

NoSQL – Not Only Structured Query Language;

ACID – набір вимог до транзакційної системи;

SSL – Secure Sockets Layer;

JSON – JavaScript Object Notation;

JVM – Java virtual machine;

ОС – Операційна Система;

PaaS – Platform as a Service;

DNS – Domain Name System.

ВСТУП

Актуальність роботи та підстави її виконання. Чим далі рухається технічний прогрес, тим поширенішою стає так звана дистанційна діяльність. В наші дні віддалено можна робити практично все: спілкуватися, працювати, робити покупки, отримувати освіту.

Стрімкий розвиток та зростаюча популярність глобальної мережі Інтернет дає можливість удосконалити систему освіти. На сьогоднішній день класичні методи навчання удосконалюються використанням сучасних здобутків науки, до яких входять Інтернет, засоби телекомунікації та інше. Пандемія показала, наскільки питання якості перевірки набутих знань та надання учбової інформації є важливими. За останні 10 років кількість учбової інформації неймовірно зросла.

Причиною цьому став шалений вплив Інтернету на наше життя та відносно низька вартість інформації[1]. Онлайн навчання має ряд переваг перед іншими способами. Люди отримують можливість зручніше керувати часом та навчатись в особистому темпі. Сучасний світ потребує нової системи неперервного навчання, яке дасть людині доступ до світових інформаційних ресурсів та баз даних, що дозволять покращувати її професійні навички та підвищать її професійну мобільність.

Мета роботи. Метою цієї роботи є розробка прототипу навчальної платформи для тестування знань. Для досягнення зазначеної вище мети необхідно виконати наступні завдання:

1. Розглянути існуючі на ринку аналоги.
2. Проаналізувати популярні технології, які використовуються для створення веб-сервісів та веб-застосунків.

3. Спроектувати веб-застосунок.
4. Розробити прототип веб-сервісу на основі розглянутої та проаналізованої інформації.

Об'єкт, методи й засоби розроблення. Об’єктом роботи є веб-система для навчання дистанційно. Для виконання роботи було використано мову програмування Java та середовище розробки Intellij Idea. Щоб керувати базами даних використано можливості об'єктно-реляційної системи PostgreSQL. Веб-сторінки відображаються за допомогою мови тегів HTML і спеціальної мови стилю сторінок CSS.

РОЗДІЛ 1 ДИСТАНЦІЙНА ОСВІТА

1.1 Онлайн навчання

Онлайн навчання – це окремий вид освіти, що реалізується через Інтернет. Від класичного дистанційного навчання воно відрізняється тим, що не потребує очних зустрічей викладача та учня для перевірки знань та практичного навчання. Передача матеріалу, його закріплення, контроль успішності та розвиток навичок відбуваються онлайн.

Такий вид навчання має переваги у порівнянні з традиційним чи класичним дистанційним [2]:

1. Комфорт для учня

   Для проходження онлайн-курсу потрібне лише підключення до Інтернету та електронний гаджет. Це дозволяє учням займатися із зручного для них місця та у зручний час, що робить навчання більш привабливим;
2. Висока швидкість передачі

   Незалежно від географічного положення учня та вчителя, інформація між ними передається практично миттєво;
3. Висока якість навчання

   Використання Інтернету, як способу реалізації дистанційного навчання, дозволяє зробити його доступним, наочним та інтерактивним
4. Зручність для автора курсу

   Навчання через Інтернет можна легко автоматизувати, унеможлививши рутинні процеси з роботи. Крім цього, при грамотному підході навіть цілою онлайн-школою може керувати лише одна людина.

Але не варто думати, що навчання через Інтернет – це набагато простіше, ніж традиційна форма. Онлайн навчання також передбачає продумування

занять, перевірку домашнього завдання, контроль успішності учнів. До цього додаються додаткові завдання, наприклад, відкривання доступ до уроків, підтримання зв'язку з учнями, мотивування їх займатися щодня та інші.
Рішенням цих завдань виступають платформи для набуття знань онлайн.

1.2 Види онлайн навчання

На даний момент можна виділити п’ять основних видів онлайн навчання.

1. Навчання, інструментом якого є електронна пошта. Учні отримують навчальні матеріали та завдання від викладача по електронній пошті та надсилають назад письмові звіти та результати своєї роботи;
2. Навчання із використанням освітніх телепрограм. Електронна пошта використовується в якості зворотного зв'язку, по якій учні отримують допомогу та завдання для самостійного опрацювання від вчителів і відправляють звіти;
3. Навчання за допомогою освітніх відеотрансляцій та відеоконференцій. Відеотрансляції використовуються на етапах викладацької діяльності для трансляції теоретичних матеріалів, аудіо та відеоконференції корисні для семінарської роботи або під час захисту практичних робіт студентами;
4. Навчання на базі комп'ютерних освітніх систем. З електронними навчальними виданнями, зазвичай включеними в навчально-методичний комплекс та складаються з підручника, навчальних планів, студент може працювати самостійно на своєму комп'ютері або безпосередньо в Інтернеті;
5. Онлайн навчання організоване в середовищі Інтернету з використанням веб-посібників, електронної пошти, чатів та відеоконференцій для зворотного зв'язку, комп'ютерних моделей та симуляцій для перевірки

знань та відпрацювання навичок, набутих під час опрацювання теоретичних матеріалів [5].

1.3 Недоліки онлайн навчання

Дистанційне навчання збільшує різноманітність методів завдяки використанню мультимедіа та різних добре сконструйованих додатків. Однак існують також недоліки та обмеження процесу дистанційного навчання [4]:

1. Невисока швидкість спілкування з іншими студентами та викладачем;
2. Розробка електронних навчальних матеріалів займає більше часу, ніж класичних друкованих;
3. Складність організації навчального процесу: робота в групі, користування послугами електронного деканату та цифрової бібліотеки;
4. Відокремлення від групи та вчителя зменшує соціальні зв’язки,мотивацію та наполегливість у навчанні;
5. Через відсутність контролю, учень часто може відволікатися на побутові проблеми та подразники;
6. Неможливість взяти участь у роботах, наприклад, лабораторних або експериментальних;
7. Відволікаючі можливості Інтернету для користувача;
8. Відсутність стимулюючої атмосфери для навчання, характерної для школи чи групи, відчуття ізоляції;
9. Необхідність мати схильність до самоосвіти та самоконтролю.

1.4 Платформа для навчання

Платформа для онлайн навчання – це сервіс, розміщений в Інтернеті, який є посередником між учнем і вчителем. Можна сказати, що це сполучна ланка, яка дозволяє максимально підвищити ефективність навчання через Інтернет.

Платформа є одночасно способом зв'язку між учнем та вчителем, сховищем уроків, координатором процесів навчання та набором інструментів для управління та підвищення якості навчання в онлайн-школі.

Усі навчальні матеріали, завантажені викладачем, зберігаються на сервері або у хмарі. Програмне забезпечення платформи дозволяє налаштовувати курс так, як автору буде зручно, а учням забезпечує доступ до уроків.

Використання навчальних платформ дозволяє автору автоматизувати монотонні процеси, такі як відкриття доступу до занять, перевірка домашніх завдань, аналіз статистики, зворотний зв'язок учням та інші. Крім того, створений на платформі курс може пройти необмежена кількість учнів.

РОЗДІЛ 2 ОГЛЯД СИСТЕМ ДИСТАНЦІЙНОГО НАВЧАННЯ

2.1 Вимоги до навчальних платформ

Навчальні платформи служать віртуальним навчальним середовищем для студентів і викладачів. Щоб переконатися, що вони забезпечують ефективне навчання, ось деякі ключові вимоги до освітніх платформ [6]:

- Зручний інтерфейс: інтерфейс має бути простим для навігації та розуміння, з чіткими інструкціями та інтуїтивно зрозумілим;
- Доступність: платформа має бути доступною для всіх студентів, незалежно від їхніх здібностей;
- Система керування вмістом: Платформа повинна мати надійну систему керування вмістом, що дозволяє вчителям завантажувати, упорядковувати та керувати матеріалами курсу;
- Інструменти для співпраці: платформа має надавати інструменти, які дозволяють студентам і викладачам ефективно співпрацювати, наприклад дискусійні форуми, чати та документи для спільної роботи;
- Інструменти оцінювання: Платформа повинна надавати низку інструментів оцінювання, таких як тести та завдання;
- Персоналізація: Платформа повинна забезпечувати персоналізований досвід навчання, який відповідає потребам окремих студентів;
- Безпека та конфіденційність: Платформа має забезпечувати функції безпеки для захисту даних та збереження конфіденційності студентів [9].

2.2 Аналіз існуючих аналогів

Для повного розуміння що таке дистанційне навчання розглянемо та порівняємо системи та сервіси, що наразі є найбільш широко вживаними.

2.2.1 Сервіс Google Classroom

Google Classroom — веб сервіс, створений Google для навчальних закладів з метою спрощення створення, поширення і класифікації завдань безпаперовим шляхом.

Google Classroom дозволяє вчителям створювати віртуальний клас, де вони можуть публікувати матеріали занять, завдання та спілкуватися зі студентами. Студенти можуть приєднуватись до потрібного курсу та переглядати та надсилати завдання, спілкуватися з викладачами та однокласниками, а також отримувати доступ до матеріалів курсу та ресурсів.

Google Classroom інтегрується з іншими службами Google, такими як Google Диск, Документи та Таблиці, що полегшує вчителям створення та обмін матеріалами для класу, а учням – співпрацювати над завданнями [7,8].

2.2.2 Сервіс Microsoft Teams

Microsoft Teams — це платформа для співпраці та спілкування, призначена для використання в різних середовищах, включаючи підприємства, навчальні заклади та некомерційні організації.

Microsoft Teams пропонує широкий спектр функцій і інструментів для підтримки командної співпраці, включаючи чат і обмін повідомленнями, відео та аудіо дзвінки, спільний доступ до файлів і зберігання, а також інструменти керування проектами.

Він інтегрується з іншими продуктами Microsoft, такими як Office 365, OneDrive та SharePoint, а також пропонує низку інструментів для співпраці та керування проектами, включаючи списки завдань, календарі та інтеграцію з популярними інструментами керування проектами, що полегшує користувачам доступ до файлів і спільний доступ до них [9–11].

2.2.3 Порівняння Google Classroom та Microsoft Teams

Google Classroom і Microsoft Teams — це потужні онлайн-платформи для співпраці та спілкування, розроблені для підтримки дистанційного навчання та командної роботи. Хоча вони мають певну схожість, можна виділити наступні відмінності:

- Цільова аудиторія. Google Classroom розроблено для навчальних закладів, Microsoft Teams більш універсальна платформа, використовується як у навчальних закладах, так і в бізнесі;
- Фокус платформи. Google Classroom зосереджено на управлінні онлайн-курсами. Microsoft Teams, розроблено для підтримки обміну файлами, відеоконференції та керування проектами;
- Функціонал. Microsoft Teams пропонує ширший спектр інструментів та інтеграцій, він має вбудовані засоби керування проектами, можливості відеоконференцій та інтеграцію з іншими продуктами Microsoft. Google Classroom зосереджений на освітніх завданнях, тому пропонує такі інструменти, як Google Forms і Google Sheets для тестів і завдань.

РОЗДІЛ 3 ТЕХНОЛОГІЇ РОЗРОБКИ

3.1 Мова розробки

Java — це багато платформна, об'єктно-орієнтована та мережево-орієнтована мова, яку можна використовувати як платформу саму по собі. Java була популярним вибором серед розробників протягом двох десятиліть, і сьогодні використовуються мільйони програм написані на Java. Це швидка, безпечна та надійна мова програмування для кодування будь-чого: від мобільних додатків до програм для великих даних і серверних технологій [12].

Оскільки Java є безкоштовною та універсальною мовою, вона створює локалізоване та розподілене програмне забезпечення. Деякі типові способи використання Java включають:розробка гри,хмарні обчислення, Big Data, штучний інтелект, Internet of Things та інші.

За індексом TIOBE [13] (індекс, який оцінює популярність мов програмування рахуючи кількість пошукових запитів) зараз Java займає третє місце після C та Python і обирався мовою року у 2005, 2015 роках. На рисунку 3.1 можна побачити прогрес мови за останні 20 років.

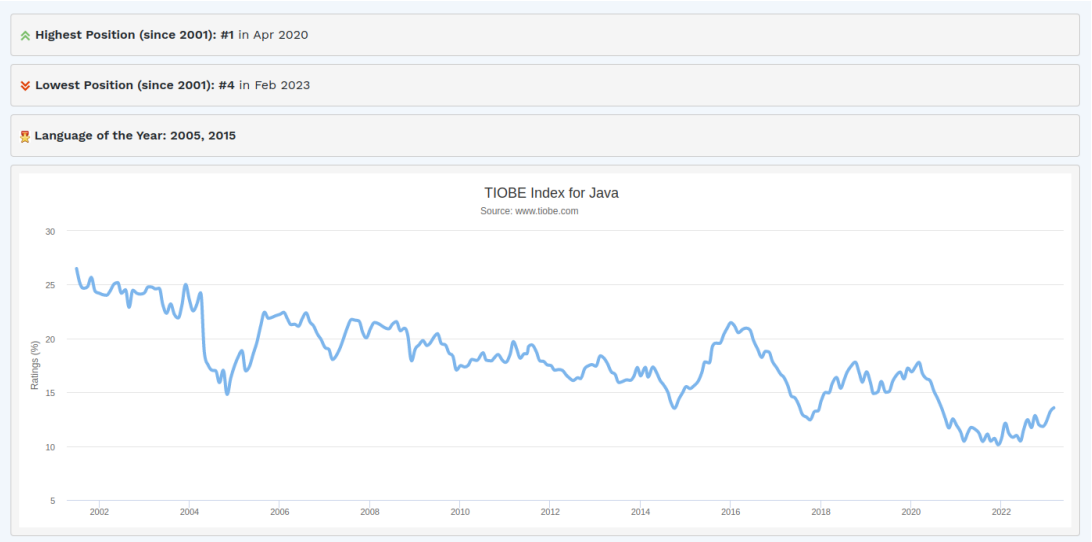

Рисунок 3.1 – Індекс мови Java за TIOBE

3.2 Основна технологія розробки

В якості бібліотеки розробки була обрана Quarkus утиліта, яка вважається однією з найсучасніших інструментів серед бібліотек Java призначених для розробки веб-додатків.

Quarkus дозволяє ефективно застосовувати Java на платформах Kubernetes та усуває розрив між традиційними Java-додатками та хмарно-орієнтованими середовищами. Quarkus як середовище виконання, дозволяє ефективно використовувати Java для вирішення актуальних завдань – при розробці хмарно-орієнтованих програм, а також реалізації нових моделей програмних систем, таких як мікросервіси, контейнери та serverless-обчислення.

3.2.1 Quarkus

Quarkus – це Java-фреймворк наступного покоління, орієнтований на Kubernetes з відкритим вихідним кодом. Він забезпечує дуже швидкий час завантаження програми та низьке споживання пам'яті. Це робить Quarkus ідеально придатним для робочих навантажень Java, що виконуються як мікросервіси в Kubernetes і OpenShift, з кращим використанням ресурсів та ефективністю.

Quarkus оптимізований для хмарних, безсерверних і контейнерних середовищ. Наразі, Quarkus цілком собі у трендах часу. Розробка backend коду стає все простішою, і цей фреймворк ще більше спрощує і прискорює розробку сервісів, додаючи “рідну” підтримку Docker та Kubernetes. Величезний плюс – вбудована підтримка GraalVM і генерації платформо-залежних образів, що дозволяє робити сервіси, що по-справжньому швидко стартують і займають мало місця в пам'яті [14].

3.3 Середовище розробки

Для розробки було обрано IntelliJ IDEA – багатомовне інтегроване середовище розробки з сімейства JetBrains, яке має необхідний набір засобів для ефективного програмування [15].

Розробка сучасних програм передбачає використання кількох мов, інструментів, фреймворків і технологій. IntelliJ IDEA розроблено як IDE для мов JVM, але численні плагіни можуть розширити його, щоб забезпечити багатомовний досвід.

Переваги використання IntelliJ IDEA :

- Сумісність з операційними системами Windows, Linux та Mac OS;
- Легка інтеграція з git, Mercurial і SVN;
- Розумний редактор коду: виділення ключових слів, класів і функцій, надання функції автозаповнення та додаткові інструкції.
- Навігація по коду: на додачу до легкого переходу по коду також є режим об’єктива. Він дозволяє розробнику ретельно перевіряти і налагоджувати весь свій вихідний код;
- Ефективний та швидкий рефакторинг, який дозволяє вдосконалювати внутрішню структуру, не змінюючи зовнішньої продуктивності коду;
- Підтримка популярних фреймворків Java: Spring, Quarkus, Micronaut, JavaFX та інші популярні фреймворки.

GitHub – це веб сервіс для хостингу та спільної розробки проектів, який базується на системі контролю версій під назвою Git. GitHub є безкоштовним для проектів з відкритим кодом та надає різноманітні платні пакети послуг для окремих індивідуальних проектів [16].

Особливості веб сервісу GitHub:

- вбудована система відстеження помилок;
- функція візуалізації гілок у проекті;
- створення необмеженої кількості приватних репозиторіїв;
- можливість спільної роботи розробників;
- можливість запропонувати власні рішення до коду інших розробників;
- легка публікація та завантаження коду.

3.4 База Даних

3.4.1 Порівняння SQL та NoSQL

SQL та NoSQL — це два різних моделі баз даних із різними підходами до зберігання та отримання даних.

Бази даних SQL засновані на структурованій моделі, яка визначає, як дані організовуються в таблиці з попередньо визначеними зв’язками між ними. Такі БД використовують мову SQL для обробки даних із суворими правилами щодо процедур збереження та оновлення даних. Бази даних SQL підходять для обробки структурованих даних і складних запитів, а також забезпечують послідовність і цілісність транзакцій.

Бази даних NoSQL, розроблені для обробки неструктурованих або напівструктурованих даних. Вони не мають попередньо визначених схем і не використовують SQL для запитів. Натомість вони пропонують гнучкі моделі даних, які здатні обробляти великі обсяги даних. Бази даних NoSQL жертвують узгодженістю транзакцій заради продуктивності та масштабованості [17].

Підсумовуючи, бази даних SQL підходять для структурованих даних зі складними зв’язками, тоді як бази даних NoSQL краще підходять для неструктурованих або напівструктурованих даних у масштабі.

3.4.2 Порівняння MongoDB та PostgreSQL

MongoDB і PostgreSQL — це дві популярні системи управління базами даних з різними підходами до зберігання та отримання даних. В таблиці 3.1 наведені деякі ключові відмінності між ними [18,19]:

	MongoDB	**PostgreSQL**
Модель даних	Зберігає дані в JSON подібних документах із гнучкими схемами	Зберігає дані в таблицях із суворими схемами та попередньо визначеними зв’язками між ними.
Масштабо-ваність	База даних може обробляти великі обсяги даних, розподіляючи їх між декількома серверами	Можна масштабувати горизонтально, але для цього потрібно більше зусиль і налаштувань
Продук-тивність	Висока швидкість обробки неструктурованих і напівструктурованих даних	Висока швидкість обробки складних транзакцій, але працює повільніше з неструктурованими даними

Мова запитів	Використовує мову запитів, подібну до JavaScript, створює потужні запити та агрегує дані	Використовує стандартизовану мову SQL для запитів до реляційних баз даних

Таблиця 3.1 – Порівняння MongoDB та PostgreSQL

Таким чином, MongoDB краще підходить для неструктурованих і напівструктурованих даних з високими вимогами до масштабованості та продуктивності, тоді як PostgreSQL краще підходить для структурованих даних зі складними запитами та високою узгодженістю транзакцій.

3.4.3 Обґрунтування вибору бази даних

Є багато причин, чому PostgreSQL є чудовим вибором для широкого кола проектів.

- Postgres є стабільною та надійнійною СУБД, успішно працює з великими та складними базами даних без проблем не пошкоджуючи дані;
- Система включає підтримку складних типів даних, повнотекстовий пошук, просторові дані та дані JSON;
- Postgres має високу масштабованість та пропонує різні методи масштабування;
- Postgres є безпечною БД та надає такі функції, як шифрування SSL, безпека на рівні рядків і механізми аутентифікації;
- Ще одна причина – потужна підтримка транзакцій ACID. Postgres гарантує послідовність та точність даних, навіть у разі системних збоїв чи інших проблем;

- Спільнота PostgreSQL забезпечує постійні оновлення та підтримку, тому можна покластися на базу даних для своїх довгострокових проектних потреб.

3.5 Вибір моделі розробки

Цикл розробки — це процес, який групи розробників програмного забезпечення використовують для проектування, створення, тестування та розгортання програмного забезпечення. Існують різні варіації циклу розробки, наприклад:

1. Agile Model – модель спринтів;
2. Waterfall Model — каскадна модель, или «водоспад»;
3. Code and fix — модель кодування и усунення помилок;
4. V-model — V-образна модель, розробка через тестування.

Для виконання роботи була обрана найпопулярніша модель розробки “Agile”. Розглянемо цю модель детальніше.

3.5.1 Agile Model

Agile – це ітеративний підхід до управління проектами та розробки програмного забезпечення, який допомагає командам швидше та з меншими проблемами постачати продукт клієнтам. Замість випуску всього продукту повністю, команда виконує роботу в рамках невеликих, але зручних інкрементів (Рисунок 3.4). Вимоги, плани та результати оцінюються безперервно, завдяки чому команди можуть швидко реагувати на зміни [20].

Ключові компоненти Agile управління проектами:

- Історії користувачів

Історія користувача — це високорівневе визначення робочого запиту, який містить інформацію, яка допомагає команді скласти оцінку зусиль, необхідних для виконання запиту;

- Спринт

  Спринт – це коротка ітерація, під час якої команди працюють над завданнями, визначеними на зустрічі з планування спринту;

- Зустрічі

  Щоденні Scrum зустрічі допомагають тримати команду в курсі справ;

- Гнучка дошка

  Гнучка дошка допомагає команді відстежувати прогрес проекту. Варіанти обирає команда, фізична дошка чи будь-яка віртуальна.

Фази Agile Model

В оригінальній моделі Ройса описані наступні фази [21]:

1. Системні та програмні вимоги

   На цьому етапі встановлюються вимоги, терміни, принципи та інші питання. Результатом є здебільшого документ із переліком вимог;

2. Дизайн

   Під час цього етапу вивчаються специфікації з першого етапу та готується проект системи. Проект системи допомагає у визначенні загальної архітектури системи. Результатом цієї фази є проектний документ високого рівня (HLD);

3. Кодування

   На третьому етапі код розробляється з використанням моделей продуктів, логіки та вимог попередніх етапів;

4. Тестування

   Система тестується великою кількістю користувачів, перш ніж система стане доступною для широкого загалу;

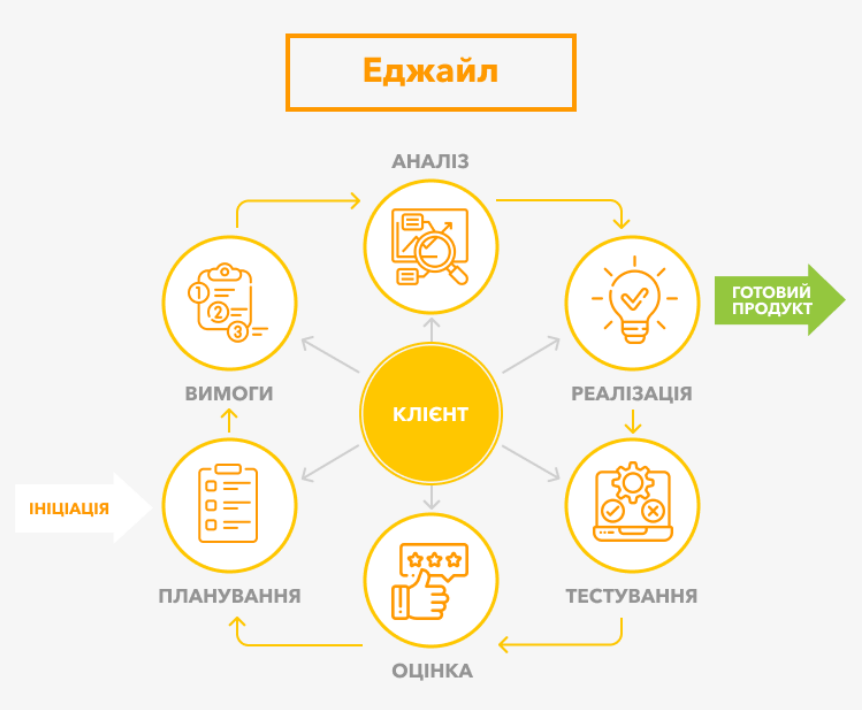

Рисунок 3.4 – Процес розробки Agile Model

5. Впровадження

   Передостанній етап складається з встановлення, міграції та обслуговування нового програмного забезпечення;

6. Підтримка та обслуговування

   На цьому етапі технічне обслуговування включає внесення невеликих частин системи для покращення продуктивності. Результатом цього етапу є посібник користувача.

3.5.2 Переваги та недоліки моделі

Управління проектами за гнучкою методологією — це спосіб керування проектом таким чином, щоб кожна частина могла мати однакові фази планування та виконання до досягнення кінцевого продукту.

В наступній таблиці наведені переваги та недоліки методу проектування [22].

Переваги	**Недоліки**
Своєчасна доставка Гнучка стратегія дозволяє відділу якомога швидше доставляти продукти клієнтам.	*Труднощі перенесення* Перехід до іншого стилю управління може потребувати час, щоб звикнути до стилю просування проектів у системі.
Адаптивність Через невеликий приріст удосконалення між датами постачання продукту, проекти можна легко змінювати.	*Змінні цілі* Змінні цілі всередині відділу можуть призвести до відсутності конкретних цілей для команди, що може призвести до невідомих кінцевих термінів.
Підвищення продуктивності Тестування продукт у міру його виробництва дає змогу співробітникам відділу розробки швидко реагувати на проблеми, які можуть виникнути.	*Відсутність документації* Реакційне планування та прогрес є пріоритетом, тому обов’язки ведення документації виконуються повільніше порівняно з іншими.
Прозорість Цикли в гнучкому підході явно демонструють потенційні проблеми продукту і процесів розробки.	*Зміщення фокусу мети* Методологія надає змогу змінювати фокус залежно від того, яка частина проекту потребує найбільшої уваги.

Менше підготовчих робіт Зосередження на продукті, а не на процесі розробки дозволяє створити продукт швидше.	*Менша передбачуваність* Команда іноді не може передбачити прибуток до початку виробництва, оскільки воно залежить від постійного вдосконалення та відгуків клієнтів.

Таблиця 3.2 – Переваги та недоліки Waterfall Method

3.5.3 Порівняння моделей Waterfall та Agile

Waterfall та Agile — це дві методології розробки програмного забезпечення, які суттєво відрізняються підходом до процесу розробки, однак між цими двома методологіями також є певна схожість [23].

- Цілеспрямованість: як Waterfall, так і Agile зосереджені на досягненні конкретної мети, будь то постачання готового продукту чи функцій;
- Командна співпраця: методології наголошують на співпраці між членами команди, хоча підхід до співпраці трішки різний;
- Гарантія якості: обидві методики включають тестування та гарантію якості, щоб переконатися, що кінцевий продукт відповідає необхідним вимогам і стандартам;
- Управління проектом: обидві методології вимагають певної форми управління проектом, щоб гарантувати, що проект залишається на правильному шляху;
- Документація: Методології певною мірою вимагають документації, щоб переконатися, що команда працює в одному напрямку щодо вимог, дизайну та реалізації.

Різницю між підходами можна побачити в наступних розділах:

- Підхід: Waterfall — це послідовний, лінійний підхід до розробки програмного забезпечення, тоді як Agile — це ітеративний і гнучкий підхід;
- Вимоги: Waterfall вимагає, щоб усі вимоги були визначені заздалегідь до початку розробки, тоді як Agile дозволяє вимогам розвиватися та змінюватися з часом;
- Планування: Waterfall вимагає попереднього детального планування та не дозволяє змінювати план після початку розробки, тоді як Agile дозволяє розвивати планування в міру зміни вимог;
- Документація: Waterfall робить великий акцент на документації, тоді як Agile робить більший акцент на робочому програмному забезпеченні;
- Тестування: Waterfall розміщує тестування в кінці процесу розробки, тоді як Agile включає тестування протягом усього циклу розробки;
- Структура команди: Waterfall часто покладається на сувору ієрархію та ролі в команді розробників, тоді як Agile наголошує на міжфункціональних командах і співпраці;
- Управління ризиками: Waterfall зосереджується на управлінні ризиками на ранніх стадіях процесу розробки, тоді як Agile керує ризиками протягом усього циклу розробки.

Загалом Agile зазвичай вважається більш гнучким і адаптованим до мінливих вимог, ніж Waterfall. Гнучкість забезпечує постійний зворотній зв’язок і коригування, що веде до більш орієнтованого на клієнта підходу до розробки.

Рисунок 3.5 – Приклад моделі водоспаду

Однак Waterfall (Рисунок 3.5) може бути більш передбачуваним і структурованим, що полегшує керування великими складними проектами з багатьма зацікавленими сторонами [30,31].

3.6 Вибір архітектури системи

Архітектура додатків — це структурний принцип, яким створено додаток. Кожному архітектурному вигляду властиві свої характеристики, властивості та відносини між компонентами. Компонент – дрібна або велика логічна та незалежна частина архітектурної системи програми. Наприклад: база даних, підсистема, бібліотека та інші [24].

Для даного проекту була обрана мікросервісна архітектура. Щоб аргументувати такий вибір були розібрані та порівняні популярні монолітна та мікросервісна архітектури.

3.6.1 Монолітна архітектура

Монолітна архітектура — це традиційна модель програмного забезпечення, яка побудована як уніфікована одиниця, самодостатня та незалежна від інших програм (Рисунок 3.6). Монолітна архітектура — це окрема велика мережа з єдиною кодовою базою, яка об’єднує всі бізнес-задачі[25].

Монолітна архітектура можуть бути зручною на ранніх стадіях розробки та використання проекту для полегшення керування кодом і розгортання. Однак такі проекти стають складнішими після великих змін та не зручними для підтримки після масштабування.

Рисунок 3.6 – Монолітна архітектура програмного виробу

Як і будь-яка інша технологія, монолітна архітектура має переваги та недоліки, продемонстровані в наступній таблиці:

Переваги	**Недоліки**
Легке розгортання	Повільність розробки
Зручність розробки	Погана масштабованість
Спрощене тестування	Ненадійність
Легке налагодження	Недостатня гнучкість
Безпечність транзакцій	Постійне повторне розгортання

Таблиця 3.3 – Переваги та недоліки монолітної архітектури

3.6.2 Мікросервісна архітектура

Мікросервісна архітектура передбачає принципово інший підхід до розробки, коли всі додаток зібрано з окремих незалежних модулів, складених з невеликих обсягів коду. У кожного модуля своя власна логіка та база даних, а їх взаємодія здійснюється через мережу за протоколонезалежною технологією (Рисунок 3.7).

Додаток на базі мікросервісної архітектури дає більше можливостей, проте влаштований він складніше [26].

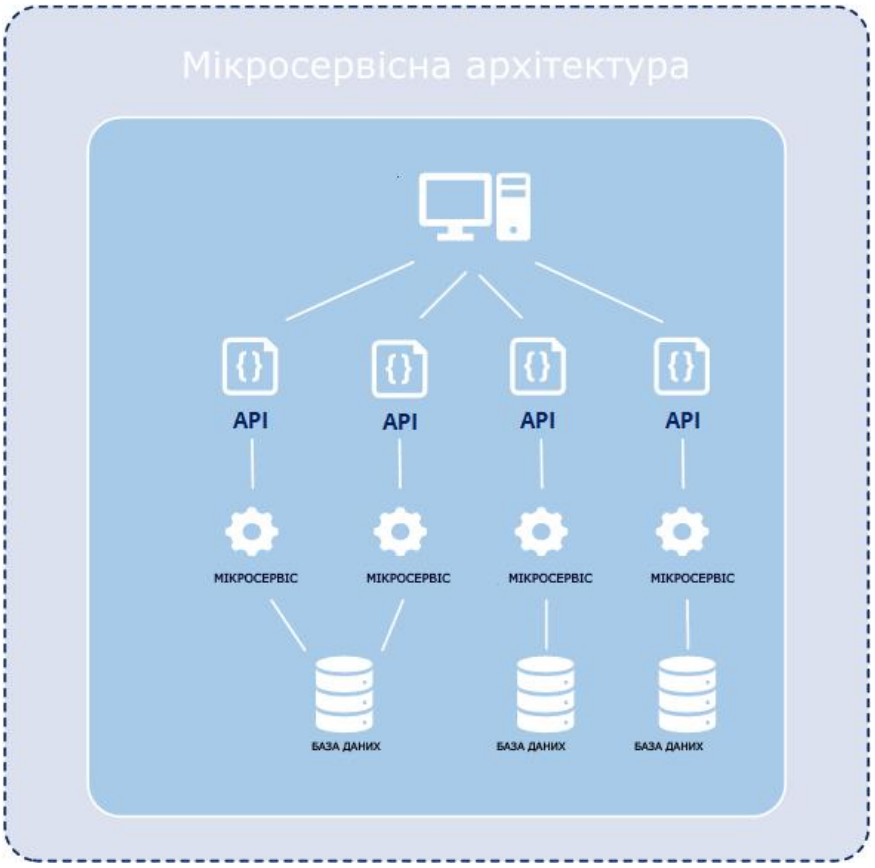

Рисунок 3.7 – Мікросервісна архітектура

Така архітектура часто зустрічається в розробці, вона має як низку незаперечних переваг так і список недоліків :

Переваги	**Недоліки**
Гнучкість	Розростання розробок
Масштабування	Витрати на інфраструктуру
Безперервне розгортання	Організаційні витрати
Зручність обслуговування	Налагодження
Незалежне розгортання	Приналежність

Таблиця 3.4 – Переваги та недоліки мікросервісної архітектури

3.6.3 Порівняння Монолітної та Мікросервісної архітектур

Монолітна архітектура та архітектура мікросервісів — це два різні підходи до розробки програмних систем, але вони мають певну схожість. Натсупна таблиця демонструє порівняння між монолітною архітектурою та архітектурою мікросервісів, які також можна побачити на рисунку 3.8:

	Монолітна архітектура	**Мікросервісна архітектура**
Компоненти	всі компоненти програми знаходяться в одному файлі.	програма розбивається на менші, більш керовані компоненти.
Розгортання	програма розгортається як єдине ціле, що може ускладнити масштабування окремих компонентів або розгортання оновлень програми без впливу на всю систему.	окремі компоненти можна розгортати й оновлювати незалежно. Можна розгортати як локально так і в хмарі.
Зв'язок між компонентами	взаємодіють один з одним безпосередньо в рамках одного процесу	взаємодіють один з одним через API
Маштабованість	складна для масштабування	є більш масштабованою через її розподілену природу

Розмір і складність	може ускладнити керування та масштабування в міру зростання програми.	більш керовані компоненти, які можна розробляти, тестувати та розгортати незалежно.
Технології	компоненти створюються з використанням одних технологій.	різні компоненти можна створювати з використанням різних технологій.
Команди розробників	передбачає організацію груп розробників навколо різних рівнів програми.	групи розробників організовуються навколо різних сервісів.

Таблиця 3.5 – Порівняння монолітної та мікросервісної архітектур

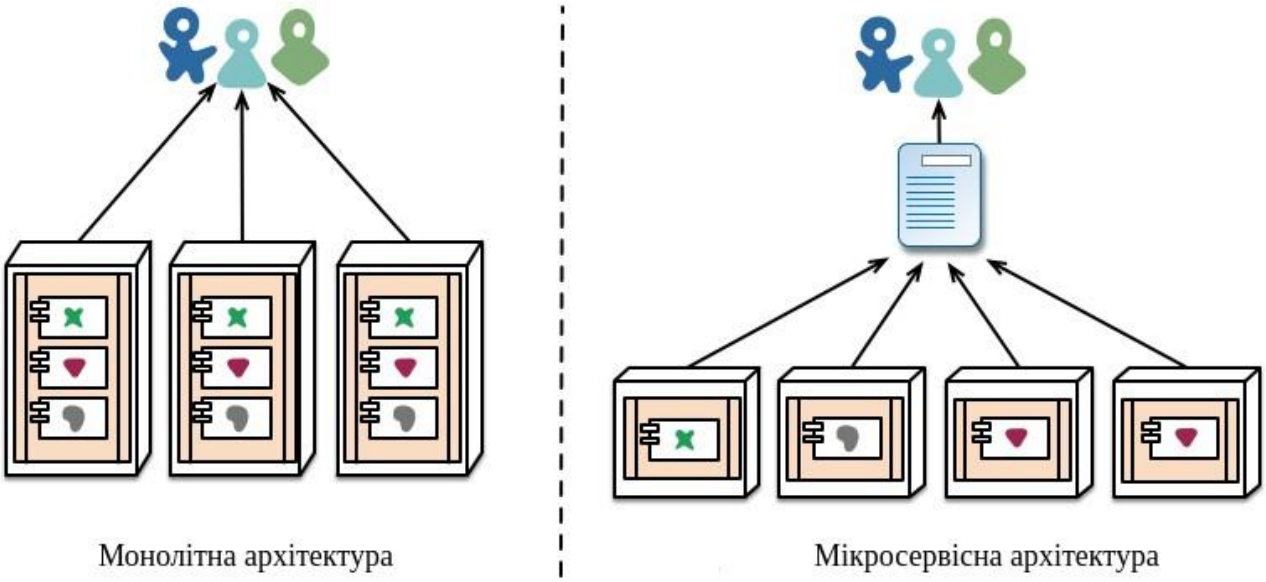

Рисунок 3.8 – Схематичне порівняння монолітної та мікросервісної архітектур

3.7 Система контейнерів

Контейнери — це легко виконувані компоненти програми, які поєднують результуючий код програми з всіма бібліотеками операційної системи і залежностями, необхідними для виконання коду в будь-якому середовищі.

Контейнери виводять розробку на вищий рівень — зокрема, крім спільного використання базового віртуалізованого апаратного забезпечення, вони також спільно використовують базове віртуалізоване ядро ОС. Контейнери пропонують таку саму ізоляцію, масштабованість і можливість використання, як і віртуальні машини, але оскільки вони не несуть корисного навантаження власного екземпляра ОС, вони займають менше місця,що є перевагою над віртуальними машинами.

Вони більш ресурсоефективні — дозволяють запускати більше програм на меншій кількості машин та з меншою кількістю екземплярів ОС. Контейнери легше переносити на робочий стіл, центр обробки даних і хмарне середовище [27].

3.7.1 Docker

Docker — це комерційна платформа контейнеризації та середовище виконання, яке допомагає розробникам створювати, розгортати та запускати контейнери. Він використовує архітектуру клієнт-сервер із простими командами та автоматизацією через єдиний API.

Docker надає набір інструментів, який зазвичай використовується для упаковки додатків у незмінні образи контейнерів шляхом написання Dockerfile і виконання відповідних команд для створення образу. Ці образи контейнерів можна розгортати та запускати на будь-якій платформі, яка підтримує контейнери, наприклад Kubernetes, Docker Swarm, Mesos або HashiCorp Nomad.

3.7.2 Kubernetes

Kubernetes — це платформа оркестровки контейнерів,також відома як «k8s» або «kube», для планування та автоматизації розгортання, керування та масштабування контейнерних програм.

Kubernetes був розроблений інженерами Google, але у 2014 році компанія надала вільний доступ до вихідного коду.

Сьогодні Kubernetes розвивається в обчислювальну платформу й екосистему загального призначення, яка конкурує з віртуальними машинами, пропонуючи як основні будівельні блоки сучасної хмарної інфраструктури та програм. Ця екосистема дозволяє організаціям запропонувати високопродуктивну платформу PaaS, яка вирішує численні завдання, пов’язані з інфраструктурою та операціями, а також проблеми, пов’язані з хмарною розробкою, щоб команди розробників могли зосередитися виключно на кодуванні та інноваціях.

Однією з переваг Kubernetes є те, що він значно спрощує створення та запуск складних програм. Kubernetes економить багато часу та зусиль розробників і дозволяє зосередитись на розробці функцій для своїх додатків, замість того, щоб з’ясовувати та впроваджувати способи забезпечення нормальної роботи своїх додатків у масштабі [28].

3.7.3 Kubernetes vs Docker Swarm

Docker Swarm або режим Docker swarm — це інструмент оркестровки контейнерів, такий як Kubernetes, тобто він дозволяє керувати декількома контейнерами, розгорнутими на кількох хостах, на яких запущено сервер Docker.

Kubernetes оркеструє кластери машин для спільної роботи та планує роботу контейнерів на цих машинах на основі їхніх доступних ресурсів. Через

декларативне визначення контейнери групуються в модулі, які є основною одиницею Kubernetes. Kubernetes автоматично керує такими речами, як виявлення служб, балансування навантаження, розподіл ресурсів, ізоляція та вертикальне або горизонтальне масштабування ваших модулів [29].

Розділ 4 СПЕЦИФІКАЦІЯ ВИМОГ

Даний розділ демонструє перший етап моделі Agile, а саме Системні та програмні вимоги розробки, та описує варіант програмного продукту.

Актор	**Опис**
Неавторизований користувач	Може відвідувати лише головну сторінку, сторінки реєстрації та входу. Може зареєструватись в системі в якості викладача або студента чи увійти в систему.
Студент	Редагує профіль,приєднується до курсу, переглядає оголошення, складає тести,надсилає повідомлення викладачу, отримує статистику своєї успішності, отримує сповіщення про нові завдання та матеріали, а також про оцінювання роботи .
Викладач	Редагує профіль,створює та переглядає створені їм курси, створює і редагує в них оголошення, додає матеріали, створює та переглядає тести, встановлює дедлайни на них. Викладач створює контрольні роботи та задає час їх публікації та час проведення з автоматичним закриттям наприкінці. Викладач отримує отримує статистику результатів за весь навчальний процес кожного студента та груп студентів тощо, у разі публічності курсу, Викладач отримує середній бал всіх учнів. Викладач курсу розсилає сповіщення про контрольну.

Таблиця 4.1 – Опис дійових осіб

4.1 Історії користувачів

У розробці програмного забезпечення та управлінні продуктами історія користувача — це неформальний опис природною мовою однієї або кількох функцій програмної системи. Історія користувача — це інструмент, який використовується в Agile-розробці програмного забезпечення для опису функції програмного забезпечення з точки зору кінцевого користувача. Історія користувача описує тип користувача, чого він хоче і чому. Історія користувача допомагає створити спрощений опис вимог.

Неавторизований користувач

Як <Неавторизований користувач>, я хочу <авторизуватися/зареєструватися>, щоб <мати можливість проходити курси>.

Як <Неавторизований користувач>, я хочу <переглянути список вільних курсів>, щоб <вибрати цікаві для мене курси>.

Як < Неавторизований користувач>, я хочу <переглядати інформацію про курс вільного доступу>, щоб <зрозуміти, чи я хочу його проходити>.

Студент

Як <Студент>, я хочу <проходити авторизацію>, щоб <отримати доступ до облікового запису та своїх курсів та реєструватися на нові курси>.

Як <Студент>, я хочу <переглядати список курсів, які я проходжу>, щоб <вибрати конкретний курс та працювати з ним>.

Як <Студент>, я хочу <переглядати курс, на якому я навчаюсь>, щоб < мати доступ до матеріалів>.

Як <Студент>, я хочу <мати доступ до свого облікового запису>, щоб <редагувати дані (ім'я, прізвище, дату народження, email, контакти)>.

Як <Студент>, я хочу <проходити тести>, щоб <отримувати бали>.

Як <Студент>, я хочу <отримувати сповіщення про нові матеріали>, щоб <вчасно їх переглядати>.

Як <Студент>, я хочу <отримувати сповіщення про оцінювання завдання>, щоб <розуміти свій рівень>.

Як <Студент>, я хочу <отримувати сповіщення про тести та контрольні роботи >, щоб <вчасно їх проходити>.

Як <Студент>, я хочу <отримувати сповіщення запрошення на курси>, щоб <вчасно їх проходити>.

Як <Студент>, я хочу <отримувати статистику балів>, щоб <відслідковувати успішність>.

Як <Студент>, я хочу <відзначати матеріал опрацьованим>, щоб <фіксувати прогрес>.

Як <Студент>, я хочу <відстежувати історію курсів>, щоб <мати доступ до пройдених матеріалів>.

Викладач

Як <Викладач>, я хочу <авторизуватися/зареєструватися>, щоб <отримати можливість створювати курси>.

Як <Викладач>, я хочу <створювати курси>, щоб <систематизувати навчання>.

Як <Викладач>, я хочу <додавати матеріали>, щоб <полегшити навчання>.

Як <Викладач>, я хочу <робити оголошення>, щоб <повідомляти про зміни в курсі>.

Як<Викладач>, я хочу <робити розсилку>, щоб <повідомляти всіх студентів одночасно>.

Як<Викладач>, я хочу <додавати студентів на курс>, щоб <контролювати кількість студентів>.

Як<Викладач>, я хочу <формувати із студентів групи на курсі>, щоб <контролювати кількість студентів в одній групі>.

Як<Викладач>, я хочу <призначати студентам тести>, щоб <перевіряти рівень знань>.

Як<Викладач>, я хочу <призначати студентам контрольні тести>, щоб <перевіряти знання за фіксований час із фіксацією процесу>.

Як <Викладач>, я хочу <відкривати курси>, щоб <відкривати доступ студентам до матеріалів та тестів та дозволити отримувати сповіщення про матеріали та тести в цьому курсі>.

Як<Викладач>, я хочу <закривати курси>, щоб <забороняти студентам проходити тести та відключити сповіщення у цьому курсі>.

Як<Викладач>, я хочу <встановлювати дедлайн>, щоб <структурувати навчання>.

Як<Викладач>, я хочу <бачити статистику проходження тестів>, щоб <регулювати подачу матеріалу>.

Як <Викладач>, я хочу <створювати тести>, щоб <перевіряти знання>.

Як<Викладач>, я хочу <редагувати непризначені тести>, щоб <покращувати, змінювати їх та виправляти помилки>.

Для легшого розуміння Історії користувачів було розроблено діаграму варіантів використання (додаток А, рисунок А.1).

Опис нефункціональних вимог знаходиться у таблиці 4.2, що подана нижче:

Назва вимоги	**Опис вимоги**
Інтуїтивний дизайн	Інтерфейс є простим і зрозумілим користувачу без спеціальної підготовки

Адаптованість дизайну	Продукт має бути доступним як із комп’ютера так і з мобільного пристрою
Конфіденційність особистої інформації	Надійне зберігання даних користувачів.
Надійність	Архітектура продукту та реалізація мають бути захищені та зрозумілі для спеціалістів, задля швидкого виявлення та усунення помилок.
Доступність	Програмний продукт має підтримуватись всіма популярними браузерами
Системні вимоги до сервера	Рекомендовані системні вимоги для запуску сервера соціальної мережі: Процесор: 4 ядерний Оперативна пам’ять: 8 Гб Вільне місце на жорсткому диску: 2 гб

Таблиця 4.2 – Нефункціональні вимоги

4.2 Проектування Бази Даних

Для проекту була обрана СУБД, що побудована на реляційній моделі, таким чином основними компонентами будуть таблиці, які забезпечують структуризацію та збереження інформації.

На основі функціональних вимог, які були описані в попередньому розділі, була спроектована та відображена (рисунок 4.2) база даних, що реалізує модель «Сутність – Зв'язок» та створена використовуючи описану вище СУБД

PostgreSQL. Звичайно, у процесі розробки БД може змінюватись, тому надані діаграми можуть не повністю відповідати кінцевому результату.

Для попереднього заповнення БД даними на мікросервісі БД працює фреймворк Flyway міграція. У разі потреби заміни СУБД, Flyway успішно створить та збере базу даних за вашими даними. Фреймворк зберігає дані SQL скриптами, а для слідкування за версіями створює в БД таблицю **flyway_schema_history** та кожного разу перевіряє правильність даних обчислюючи, зберігаючи та перевіряючи під час створення БД, контрольну суму. Файли з SQL мають зберігатись в директорії src/main/resources/db/migration, та називатись *V1.0.0__Quarkus.sql*, де V1.0.0 – версія БД, Quarkus – обов'язкова назва фреймворку.

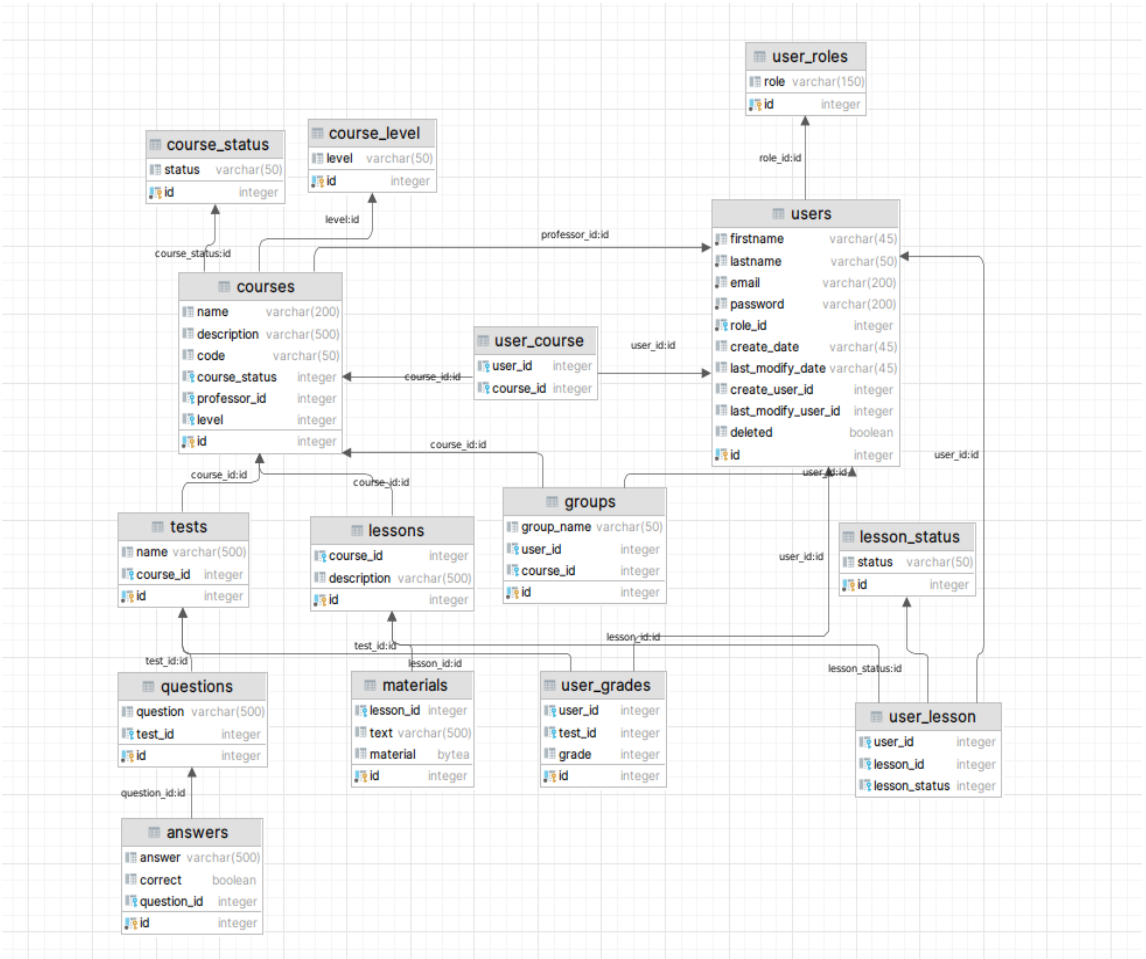

Рисунок 4.2 – Діаграма бази даних

На рисунку 4.2 наведена діаграма взаємозв’язків сутностей, що ілюструє відношення таблиць і взаємодію бази даних спроектованого сервісу.

В процесі проектування було розроблено 15 таблиць, 4 з яких є основними сутностями, 5 є допоміжними таблицями та 4 в якості реалізації зв'язку «багато-до-багатьох». **Спроектовані сутності:**

1. Користувач;
2. Курс;
3. Заняття;
4. Тест;
5. Питання;
6. Відповідь;
7. Матеріали.

Допоміжні таблиці

1. Статус курсу;
2. Статус заняття;
3. Роль користувача;
4. Рівень курсу.

Кожна сутність та елемент допоміжних таблиць мають унікальний первинний ключ.

Таблиці реалізації зв’язку

1. користувач – курс;
2. користувач – група;
3. користувач – заняття;
4. користувач – оцінка.

Такі таблиці не містять первинного ключа, проте посилаються на первинні ключі сутностей.

Кожен запис в усіх таблицях має housekeeping поля (поля, які зберігають інформацію про час створення запису, користувача,який його створив, час останньої зміни та користувача,який змінював запис)

Для виконання CRUD операцій із БД було вирішено використати *Реактивний SQL Клієнт*.

```
@Inject PgPool client;

public Course getCourseInfoByID(int id) {
  logger.info(STARTS);
  return client
          .preparedQuery(
                  sql: "select * from courses inner join course_level cl on cl.id = courses.course_level\n"
                          + "inner join course_status cs on cs.id = courses.course_status\n"
                          + "inner join lessons l on courses.id = l.course_id where courses.id = $1") PreparedQuer
          .execute(Tuple.of(id)) Uni<RowSet<...>>
          .onItem() UniOnItem<RowSet<...>>
          .transform(Course::fromSet) Uni<Course>
          .await() UniAwait<Course>
          .indefinitely();
}
```

Рисунок 4.3 – SQL запит з використанням Reactive SQL Client

За допомогою анотації **@Inject** (Рисунок 4.3) ми використовуємо готовий пул зв'язків. В самій функції **getCourseInfoByID()** записується SQL запит та, після виклику **.execute(Tuple.*of*(id)),** де функцією **Tuple.*of*(id)** динамічно надаємо значення **id**, построково формуємо з отриманих результатів екземпляр класу.

4.3 Модульна декомпозиція

Для чіткого розуміння архітектури додатка та суті пропонованого рішення було побудовано діаграму компонентів (Рисунок 4.4). Діаграма компонентів відображає основні програмні складові системи. Такими є користувацьки мікросервіси для *Студента* та *Викладача*, мікросервіс *Безпеки* та мікросервіс *Розсилок*.

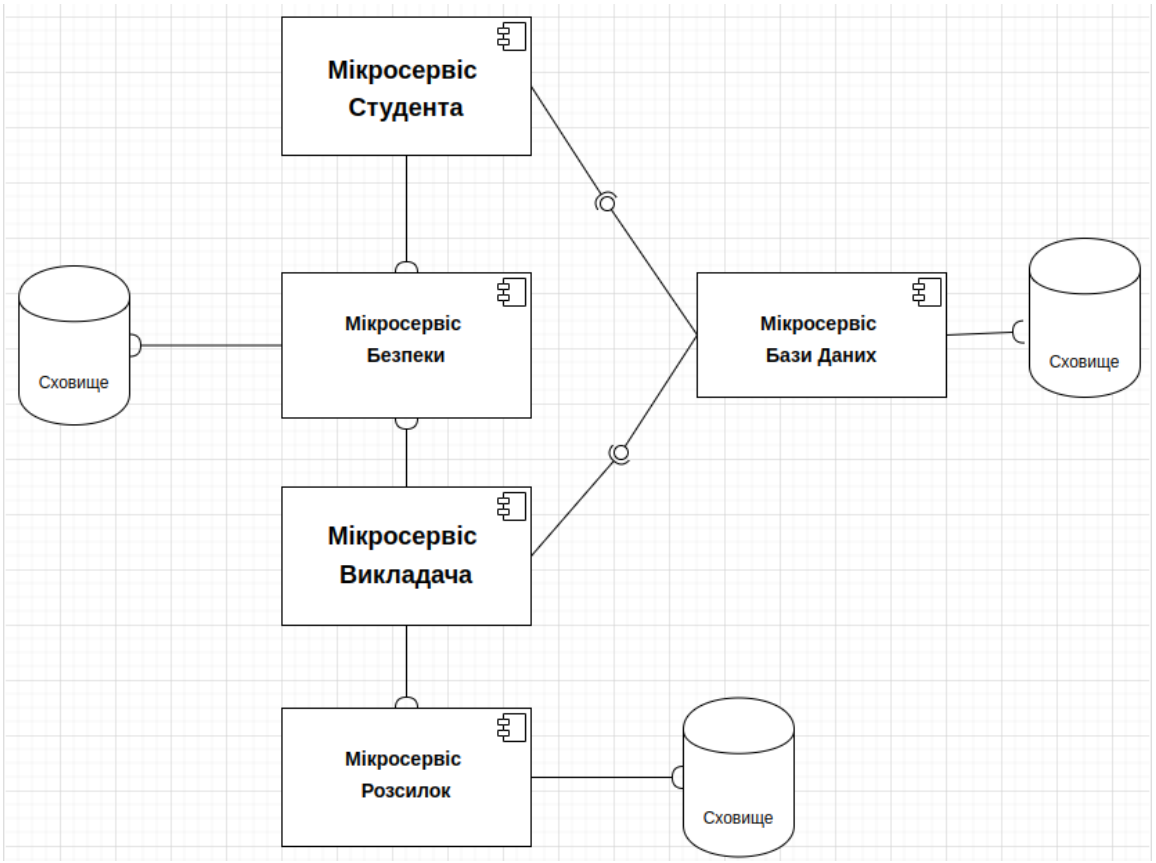

Рисунок 4.4 – Діаграма компонентів

Основним мікросервісом є сервіс БД, який единий має доступ до основної бази даних та реалізує її з'єднання з іншими сервісами системи. Мікросервіс *Безпеки* має власне сховище, яке зберігає інформацію про Користувача, яка використовується в процесі авторизації та автентифікації.

В свою чергу мікросервіс *Розсилок* також має власне сховище, в якому зберігає інформацію про способи зв'язку зі Користувачами.

Образи мікросервісів запускаються за допомогою технології Docker та docker-compose файлу, наведеного в додатку Б (Рисунок Б.1).

Спілкування між сервісами відбувається у закритій локальній мережі **study_network,** яка створюється одночасно із створенням контейнерів.

Зв’язок безпосередньо між мікросервісами відбувається за допомогою HTTP протоколу. Посилання, за якими відбувається спілкування, зберігаються в **application.properties** файлі.

Сама архітектура є вдалим рішенням для задачі такого типу, використовуючи мікросервіси ми отримуємо наступні переваги:

- *Динамічне регулювання навантаження*. У випадку занадто великої активності викладачів або студентів (одночасне складання тестів, написання контрольних робіт), система може розподілити навантаження методом створення додаткових контейнерів;
- *Плавний перехід між версіями*. При додаванні нового функціоналу система обережно та непомітно для Користувачів буде переводити їх на контейнери нового зразка, виключаючи старі;
- *Швидке реагування на помилки*. В ситуації відмови контейнера, система швидко реагує та замінює “впавший” контейнер на новий, завдаючи мінімум незручностей для Користувачів.

Як основний фреймворк був обран Quarkus, який має суттєві плюси для вирішення поставленої задачі. Quarkus — це реактивний фреймворк. З самого початку Reactive був основним принципом архітектури Quarkus. Він містить багато реактивних функцій і пропонує широку екосистему. В свою чергу Reactive — це набір принципів і вказівок для створення адаптивних розподілених систем і програм. Reactive має чотири характеристики:

Чуйність – вони повинні реагувати вчасно;

Еластичність – вони адаптуються до коливань навантаження;

Витривалість – вони витончено справляються з невдачами;

Асинхронна передача повідомлень – компоненти реактивної системи взаємодіють за допомогою повідомлень.

Так, асинхронна передача повідомлень найкраще підходить для мікросервісної архітектури, де всі компоненти спілкуються через системи повідомлень, еластичність вдало поєднується з можливістю архітектури збільшувати або зменшувати кількість контейнерів залежно від навантаження, а чуйність відображає швидку реакцію системи на невдачі в створенні, або помилки в контейнерах.

Для розробки програмного забезпечення системи була обрана методологія Agile. Методологія дозволяє отримувати та демонструвати робочий варіант системи на кожному циклі розробки. Наступні кроки були зроблені у ході виконання роботи:

1. *Розробка системних та програмних вимог*

   Під час цього кроку були написані Історії Користувачів та нефункціональні вимоги;

2. *Дизайн*

   На цьому кроці були обрані основні технології (Intellij Idea, Quarkus, PostgreSQL), архітектура додатку та методологія розробки;

3. *Кодування*

   Крок кодування має в основі Agile спринти і як результат ми отримали два репозиторії, демонструючі клієнтську частину для викладачів та серверну частину, яка спілкується з Базою Даних;

4. *Тестування*

   На кроці тестування були написані тести за допомогою функціоналу для тестування, який надає Quarkus фреймворк;

4.4 Інтерфейси системи

Для створення інтерфейсу було вирішено використовувати функціонал Quarkus Qute Template Engine. Qute — це система створення шаблонів, розроблена спеціально для задоволення потреб Quarkus. API поєднує як імперативний, так і неблокуючий реактивний стиль кодування. У режимі розробки всі файли, розташовані в src/main/resources/templates та ретельно під час створення перевіряються. Сторінки, які не потребують демонстрації інформації з БД зберігаються як звичайні HTML сторінки в src/main/resources.

Акаунт Користувача (рисунок 4.5) демонструє базову інформацію про Користувача та курси,які він створив, якщо Користувач є **Викладачем**, або курси, які він проходить, якщо Користувач є **Студентом**.

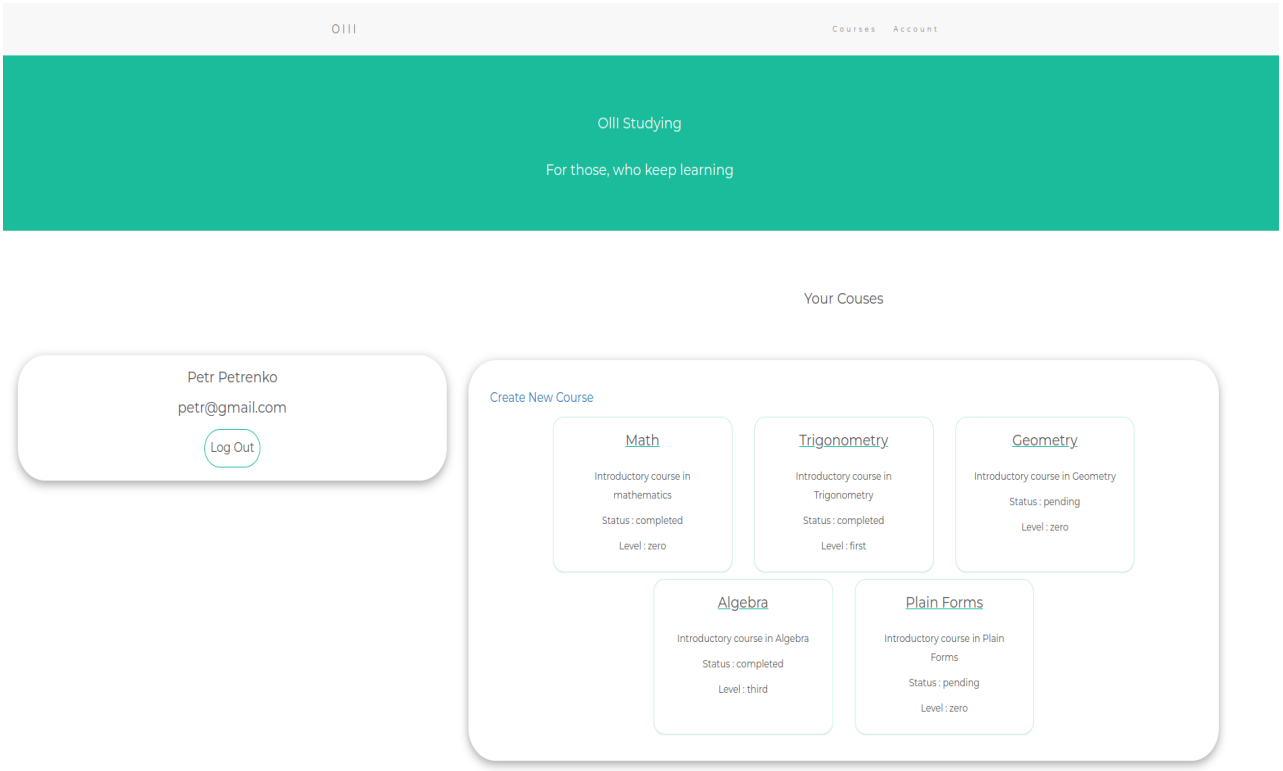

Рисунок 4.5 – Акаунт Користувача

Кнопка *створення курсу* доступна лише **Викладачу**, в акаунті **Студента** на цьому місті відображається кнопка приєднання до курсу, яка відкриває наступну форму.

В той час як Викладач, натискаючи на кнопку створення курсу, отримує форму:

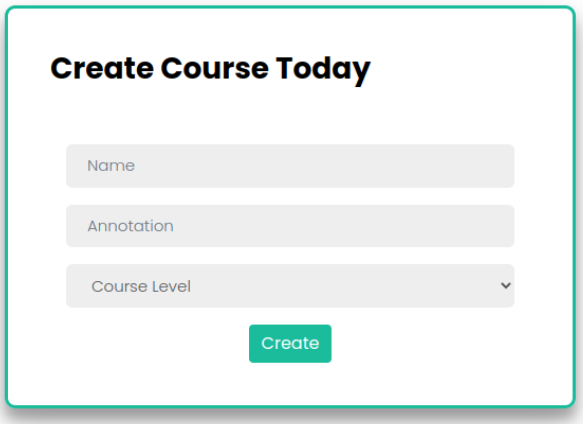

Рисунок 4.6 – Форма для створення нового курсу

Курс складається з *Занять* та *Тестів*. Заняття може зберігати як текстові пояснення, так і матеріали (зображення, файли). Переключаючи вкладки на сторінці, **Викладач** може побачити список занять, тестів, всіх студентів на курсі та статистику виконання завдань та складання тестів.

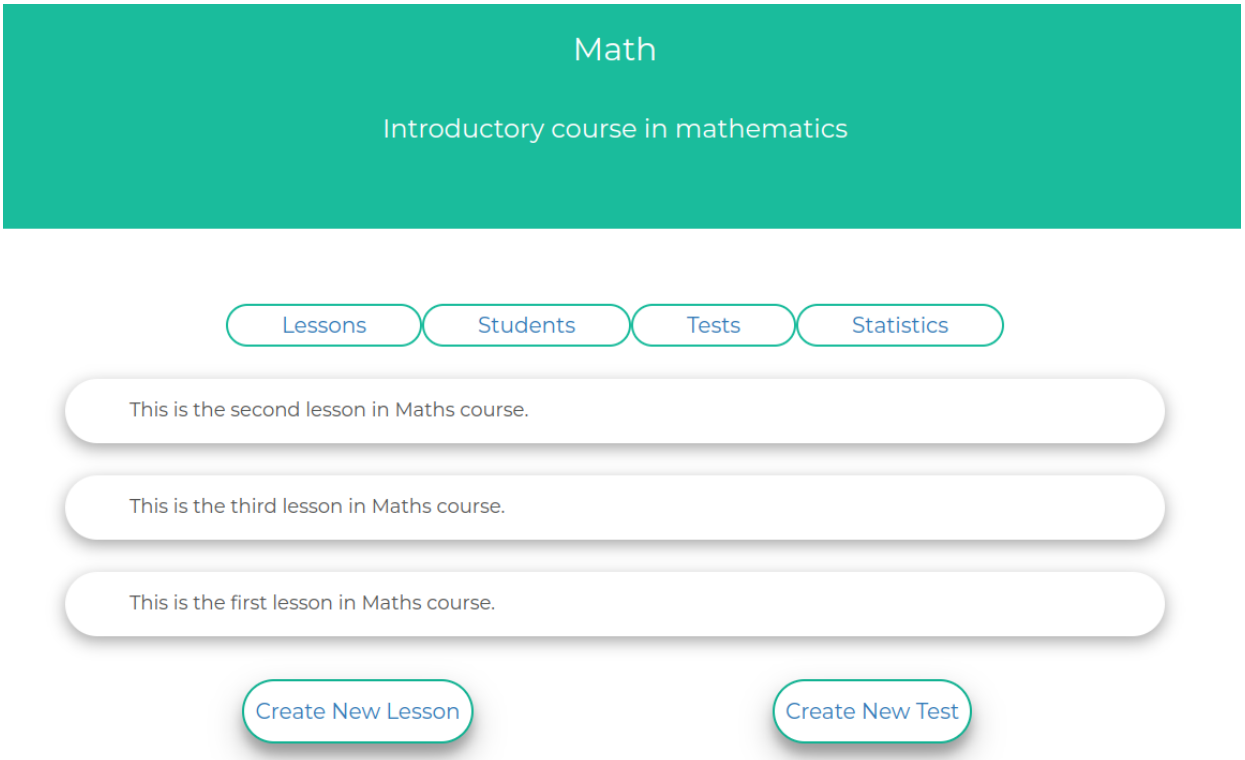

Рисунок 4.7 – Сторінка курсу із заняттями

За допомогою кнопок “Create New Lesson” та “Create New Test”, **Викладач** заповнює форми та створює нові заняття (Рисунок 4.9) та тести.

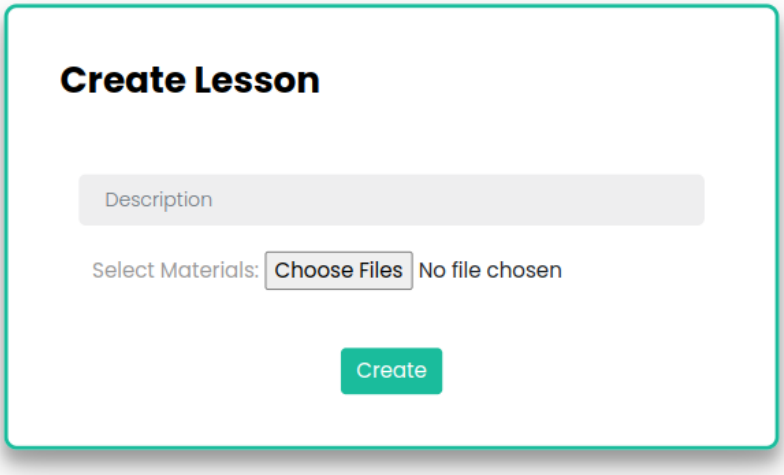

Рисунок 4.8 – Форма для створення заняття

Так, в формі для створення заняття потрібно ввести опис до нього, та додати матеріали, якщо вони передбачені. Цілком допустимо створити заняття лише з описом.

Всі дані, надіслані через форми пересилаються на сервер по HTTP, дані збираються та формується запит за допомогою наступного коду, написано із використанням JQuery (Рисунок 4.9).

```
$(document).ready(function () {
    $('#sign-in').click(function () {
        var email = $('#email').val();
        var password = $('#password').val();
        $.post({
            url: 'http://localhost:8082/api/v1/login',
            contentType: 'application/json',
            data: JSON.stringify({"email":email, "password":password})
        });
    });
});
```

Рисунок 4.9 – Приклад створення запиту

Через кількість пристроїв навколо людини зараз, дистанційне навчання не дає можливості реально оцінити знання студента, тому пропонується розділити тестування знань на два етапи: самостійні тести та контрольні роботи. Різниця заключається в методах здачі. Самостійна робота є звичайним тестом, в той час як контрольну роботу пропонується зробити тестуванням, які всі студенти курсу або групи проходять одночасно, з автоматичним відліком часу та доступом до модулів тесту, випадковим набором питань із переліку для кожного студента індивідуально тазахватом екрану, щоб уникнути одночасного пошуку в інтернеті.

4.5 Напрямки подальшого розвитку системи

Система є пластичною, тому поліпшувати базовий функціонал та розширювати систему новим можна нескінченно. Надалі наведено декілька можливих нових функцій:

- Розподілення курсів на приватні (зі зворотнім зв’язком від викладача) та публічні (самостійна обробка матеріалу);
- Створення спільної бази питань та відповідей, тестів та систем випадкового підбору питань в тести, розподілення варіантів та інше;
- Динамічне створення сегментів тесту, їх переміщення, часові обмеження на кожен блок питань;
- Створення тестових шаблонів, як приватних так і для всіх користувачів системи;
- Створення,збереження та відображення сертифікатів користувачів після пройдених курсів;
- Збільшення ролей та функціонал для них (асистент викладача, не може створювати курси самостійно, може брати участь у розробці курсу разом із викладачем);
- Надання викладачу можливості регулювати навантаження на студентів, їх кількість та кількість груп незалежно від груп в навчальному закладі;
- Створення учбових закладів та прикріплення до них студентів та викладачів;
- Викладання матеріалів в хмарах, стрімінг відео;
- Обмеження кількості курсів, які студент може проходити в певній навчальній галузі;
- Можливість редагувати вже опубліковані курси та тести та надавати оновлену версію новим учням, поступово замінюючи стару версію;

- Доступ до курсів за посиланням з лімітованою кількістю приєднань по ньому;

ВИСНОВКИ

У ході виконання роботи було проведено аналіз методів дистанційного навчання, виділено переваги і недоліки здобуття освіти дистанційно.

Проаналізовано технічні особливості доступних сучасних платформ розроблення веб-додатків та обрано технології для реалізації системи. Вироблено та аргументовано архітектурні та проектні рішення стосовно системи, призначеного до розроблення.

Перед виконанням роботи були поставлені та успішно виконані наступні задачі:

- Сформулювали вимоги до застосунку;
- Проаналізували та розробили архітектуру системи;
- Вибрали відповідні та сучасні технології для розробки застосунку;
- Проаналізували існуючі рішення в галузі онлайн навчання;
- Передбачили перспективи та напрямки розвитку системи.

Розроблена система підходить не тільки для учбових закладів. Гнучкість обраної архітектури робить можливим розширення функціоналу та подальше вдосконалення системи.

Результатом роботи є спроектована архітектура додатка розроблений функціонал Викладача та мікросервіс з’єднаний з базою даних. Програмне рішення розроблено, використовуючи сучасні технології з урахуванням принципів ООП, SOLID, KISS, DRY.

Наразі, веб-система демонструє програмний фундамент, здебільшого корисний для Викладача, на який можна легко надбудувати новий функціонал та розширити систему спираючись на бажання потенційного замовника.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дистанційна освіта. Дистанційне навчання. Дистанційна освіта – це можливість навчатися та отримувати необхідні знання віддалено від навчального закладу. [Електронний ресурс] – Режим доступу: http://www.myshared.ru/slide/1394637/
2. Дистанційна освіта в Україні: історія та сучасний стан [Електронний ресурс] – Режим доступу:http://irbis-nbuv.gov.ua/cgi-bin/irbis_nbuv/cgiirbis_64.exe?C21COM=2&I21DBN=UJRN&P21DBN=UJRN&IMAGE_FILE_DOWNLOAD=1&Image_file_name=PDF/Mir_2010_6_27.pdf
3. Tamm S. The History of E-Learning [Електронний ресурс] / Sander Tamm. – 2019 .– Режим доступу до ресурсу: https://estudent.org/history-of-e-learning/.
4. Переваги та недоліки дистанційного навчання [Електронний ресурс] – Режим доступу: https://kerivnyk.info/perevahy-ta-nedoliky-dystantsijnoho-navchannya
5. Засоби дистанційної освіти. Методи дистанційного навчання. Види дистанційного навчання [Електронний ресурс] – Режим доступу: https://santorpack.ru/uk/school-of-design/sredstva-distancionnogo obrazovaniya-metody-distancionnogo.html
6. Гультяєв А. К. Macromedia Authorware 6.0 Розробка мультимедійних учбових курсів [Електронний ресурс] / Гультяєв А. К. // СПб.: КОРОНА принт. – 2007. – Режим доступу до ресурсу: http://www.padaread.com/?book=207515&pg=400.
7. Google Classroom official website: [Електронний ресурс] – Режим доступу: https://edu.google.com/workspace-for-education/classroom/

8. Google Classroom Blog:[Електронний ресурс] – Режим доступу: https://blog.google/outreach-initiatives/education/
9. Microsoft Teams official website:[Електронний ресурс] – Режим доступу: https://www.microsoft.com/en-us/microsoft-teams/group-chat-software
10. Microsoft Teams documentation: [Електронний ресурс] – Режим доступу:https://docs.microsoft.com/en-us/microsoftteams/
11. Microsoft Teams blog: [Електронний ресурс] – Режим доступу:https://techcommunity.microsoft.com/t5/microsoft-teams-blog/bg-p/MicrosoftTeamsBlog
12. What is JAVA [Електронний ресурс] – Режим доступу: https://aws.amazon.com/what-is/java/#:~:text=Java%20is%20a%20multi%2Dplatform,applications%20and%20server%2Dside%20technologies
13. TIOBE Index for May 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/.
14. Quarkus Official Documentation [Електронний ресурс] – Режим доступу:https://quarkus.io/about/
15. Inteliij IDEA official documentation [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/help/idea/java-ee.html
16. Chacon S. ProGit [Електронний ресурс] / S. Chacon, B. Straub. – 2014. – Режим доступу до ресурсу: https://git-scm.com/book/ru/v2.
17. SQL vs NoSQL Databases [Електронний ресурс] – Режим доступу: https://www.ibm.com/cloud/blog/sql-vs-nosql
18. PostgreSQL vs MongoDB: Comparing Databases [Електронний ресурс] – Режим доступу:https://www.bmc.com/blogs/mongodb-vs-postgresql/
19. Comparing MongoDB vs PostgreSQL [Електронний ресурс] – Режим доступу:https://www.mongodb.com/compare/mongodb-postgresql

20. Beginner's Guide to Agile Project Management [Електронний ресурс] – Режим доступу: https://business.adobe.com/blog/basics/agile
21. What Is Agile SDLC: The Guide on Major Software Development Methodology in 2023 [Електронний ресурс] – Режим доступу: https://www.cleveroad.com/blog/agile-sdlc/
22. Pros & Cons of Agile Development Methods [Електронний ресурс] – Режим доступу: https://business.adobe.com/blog/basics/agile
23. Waterfall vs. Agile: Which is the Right Development Methodology for Your Project? [Електронний ресурс] – Режим доступу: https://www.seguetech.com/waterfall-vs-agile-methodology/
24. Application Architecture: Best Practices for Future-Proofing Your Apps Built with Low-Code [Електронний ресурс] – Режим доступу: https://www.outsystems.com/blog/posts/application-architecture/
25. What is a monolithic architecture? [Електронний ресурс] – Режим доступу: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith#:~:text=A%20monolithic%20architecture%20is%20a%20singular%2C%20large%20computing%20network%20with,of%20the%20service%2Dside%20interface.
26. Microservice Architecture[Електронний ресурс] – Режим доступу: https://microservices.io/
27. Containers in the enterprise Rapid enterprise adoption continues [Електронний ресурс] – Режим доступу: https://www.ibm.com/downloads/cas/VG8KRPRM
28. What is Kubernetes?[Електронний ресурс] – Режим доступу: https://www.mirantis.com/cloud-native-concepts/getting-started-with-kubernetes/what-is-kubernetes/
29. Kubernetes vs. Docker. The key differences between Kubernetes and Docker and how they fit into containerization [Електронний ресурс] – Режим

доступу:
https://www.atlassian.com/microservices/microservices-architecture/kubernetes-vs-docker

Додаток А

Діаграма варіантів використання

Рисунок А.1– Діаграма варіантів використання

Додаток Б

Docker

```yaml
services:
  server:
    container_name: server
    image: server
    ports:
      - 8084:8084
    depends_on:
      - db
    networks:
      - study-network
  bff:
    container_name: bff
    image: bff
    ports:
      - 8082:8082
    depends_on:
      - server
    networks:
      - study-network
  db:
    container_name: db
    image: postgres
    restart: always
    volumes:
      - db-data:/var/lib/postgresql/data
    ports:
      - "15436:5432"
    environment:
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=password
      - POSTGRES_DB=postgres
    networks:
      - study-network

networks:
  study-network:
```

Рисунок Б.1 – docker-compose файл для створення образів мікросервісів.

container_name – ім’я сервісу, за яким до нього будуть звертатися інші в мережі.

image – створений командою *docker build -f src/main/docker/Dockerfile.jvm -t <ім’я>* . образ, на якому базується контейнер.

ports – “порт, по якому до контейнера можна звернутись”:”порт, на якому знаходиться контейнер”.

depends_on – контейнер, в якому зазначена залежність, створюється після зазначеного контейнера.

networks – назва локальної мережі.

Контейнер *db* демонструє Базу Даних, тому має додаткові поля:

environment – задає дані для підключення до БД(ім’я користувача, пароль, та ім’я БД).

volumes – посилання на сховище даних.