МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»

(освітньо-кваліфікаційний рівень)

на тему «Платформа дистанційного навчання вищої математики для IT спеціальностей. Системний модуль»,

«Distance learning platform for higher mathematics for IT specialties. System module»

Виконав: студент денної форми навчання

спеціальності 123 – Комп'ютерна інженерія

(шифр і назва напряму підготовки, спеціальності)

Козлов Микита Сергійович

(прізвище, ім'я, по-батькові)

Керівник ст. викладач Трубіна Н.Ф. ________________

(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент канд. фіз.-матем. наук, доцент Петрушина Т.І.

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№ 11 від « 11 » 06 2021 р.

Завідувач кафедри

_____________ Є.В. Малахов

(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___

протокол № __ від «___» ____ 2021 р.

Оцінка _____________/_______/_____

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____________ Н.Ф. Казакова

(підпис) (прізвище, ініціали)

Одеса – 2021

АНОТАЦІЯ

Дипломна робота є складовою частиною проекту по розробці платформи для дистанційного навчання вищої математики.

Мета роботи – спроектувати і розробити архітектуру освітньої платформи. В якості архітектурного рішення використовується мікросервісна архітектура.

В рамках даної роботи, розроблені: сервіс управління контентом, який надає навчальну інформацію користувачеві; сервіс забезпечення безпеки, який реалізує подобу механізму OAuth2 для доступу до ресурсів платформи; з'єднувальний модуль, що забезпечує зв'язок між усіма сервісами, існуючими і потенційними до появи в майбутньому.

Для розробки використовувалися:

- фреймворк – Spring Boot;
- мова програмування – Java (v11);
- СУБД для зберігання призначених для користувача даних і даних контенту – PostgreSQL.

Результатом роботи є реалізовані мікросервіси і з'єднувальний модуль, створений і протестований механізм спілкування між розробленими сервісами, і сервісами інших учасників проекту.

ABSTRACT

The graduation work is an integral part of a project to develop a platform for distance learning in higher mathematics.

The purpose of the work is to design and develop the architecture of the educational platform. Microservice architecture is used as an architectural solution.

Within the framework of this work, the following have been developed: a content management service that provides educational information to the user; a security service that implements a semblance of the OAuth2 mechanism for accessing platform resources; a connection module that provides communication between all services, existing and possible to appear in the future.

Used for development:

- framework – Spring Boot;
- programming language – Java (v11);
- DBMS for storing user data and content data – PostgreSQL.

The result of the work is the implemented microservices and the connecting module, a mechanism for communicating with the created services and services of other project participants has been created and tested.

АННОТАЦИЯ

Дипломная работа является составной частью проекта по разработке платформы для дистанционного обучения высшей математике.

Цель работы – спроектировать и разработать архитектуру образовательной платформы. В качестве архитектурного решения используется микросервисная архитектура.

В рамках данной работы, разработаны: сервис управления контентом, который предоставляет учебную информацию пользователю; сервис обеспечения безопасности, реализующий подобие механизма OAuth2 для доступа к ресурсам платформы; соединительный модуль, обеспечивающий связь между всеми сервисами, существующими и возможными к появлению в будущем.

Для разработки использовались:

- фреймворк – Spring Boot;
- язык программирования – Java (v11);
- СУБД для хранения пользовательских данных и данных контента – PostgreSQL.

Результатом работы являются реализованные микросервисы и соединительный модуль, создан и протестирован механизм общения с созданными сервисами, и сервисами других участников проекта.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ 6

ВСТУП 7

1 ОГЛЯД ПРИНЦИПІВ ПРОЕКТУВАННЯ ТА АРХІТЕКТУРНИХ РІШЕНЬ У СФЕРІ НАВЧАННЯ 10

1.1 Використання методики SOLID в проектуванні 10

1.2 Архітектура освітньої платформи EDX 12

1.3 Архітектура освітньої платформи Stepik 15

1.4 Основні архітектурні рішення веб-додатків 16

1.4.1 Монолітна архітектура 16

1.4.2 Сервіс-орієнтована архітектура 19

1.4.3 Мікросервісна архітектура як похідна від SOA 20

1.4.4 Сполучний модуль 23

1.5 Вимоги до системи 26

2 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ОСНОВНИХ СЕРВІСІВ ПЛАТФОРМИ ДІСТАНЦІЙНОЇ ОСВІТИ 28

2.1 Обмін інформацією 28

2.2 Засоби реалізації 29

2.3 Архітектура сервісу 30

2.4 Реалізація базових сервісів 31

2.4.1 Сервіс забезпечення безпеки 31

2.4.2 Сполучний модуль. Обробка помилок. Фільтр запитів 35

2.5 Тестування. Аналіз результатів 40

ВИСНОВКИ 44

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 46

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

API – програмний інтерфейс програми

SOA – Сервіс-орієнтована архітектура

MSA – Мікросервісна архітектура

Gateway – сполучний модуль в мікросервісной архітектурі

AuthService – сервіс забезпечення безпеки

DTO – об'єкт передачі даних

ВСТУП

В усі часи освіта супроводжувалася однією глобальною проблемою – відсутністю універсального підходу до кожної людини, в якій би сфері діяльності вона не була. Безперечно, фундаментальні знання, експериментально доведені теорії – незмінні, проте викладати їх групам людей, існуючих в різних сферах діяльності, слід по-різному. Така потреба притаманна багатьом областям науки, проте саме в сфері математики це питання стоїть особливо гостро.

Людям, далеким від областей застосування знань з математики в чистому вигляді, складно використовувати їх на високому рівні. Проблема полягає в тому, що математика – особливо абстрактна наука, що оперує такими ж поняттями, і в якості пояснення своїх аспектів, використовує не менш абстрактні приклади. В якості такого прикладу розглянемо вивчення похідної функції.

Визначення похідної функції безпосередньо ґрунтується на понятті ліміту і, особливо, на нескінченно малих величинах. Таким чином, для повного засвоєння цієї теми і подальшого використання отриманих знань, учневі необхідно зрозуміти друге, не менш абстрактне поняття – ліміт функції. Такий підхід до розуміння не є безвихідним, тим не менш, він ймовірно займе більше часу, ніж підхід, який спочатку передбачає демонстрацію похідної на прикладах з реального життя.

Так, в будівництві часто виникає задача при монтажі промислових і сільськогосподарських будівель невеликої висоти. Для правильного вибору крана необхідно знати багато даних про споруджуваний об'єкт. Зокрема, габаритні дані об'єкта дозволять заздалегідь визначити потрібну довжину стріли крана. Використавши цей факт, отримаємо цілком конкретне завдання, для вирішення якої необхідне знання похідною, а саме – «Обчислити довжину стріли автомобільного крана, за допомогою якого можна поставити будівлю

висотою H, шириною – L з плоским дахом, враховуючи, що кран може рухатися навколо споруджуваного об'єкта ».

Сфера будівництва не єдина, де похідна використовується для вирішення фундаментальних проблем. Також, завдання, що вимагають знань з даної теми математики виникають при вирішенні транспортних задач, в процесі меліорації, в рішенні економічних задач і так далі. Таким чином, підхід до вивчення абстрактних математичних понять, що передбачає використання прикладів з реального життя в якості пояснення суті питання, є вкрай ефективним.

Не дивлячись на існування такого підходу, не менш важливим пунктом в подачі інформації є, власне, сам спосіб її подачі.

В наші дні величина популярності дистанційної форми навчання наближається до значень очної форми. Причиною тому є її наступні переваги:

1) можливість навчатися у зручний час, зрозуміло, з дотриманням терміну здачі всіх передбачуваних програмою завдань;
2) передбачає мобільність як для студента, так і для викладача;
3) дозволяє впроваджувати в процес навчання індивідуальний підхід.

Для ефективного і повного засвоєння навчального матеріалу студентом, в дистанційній формі навчання присутні такі види інформації:

– текст – як спосіб подачі теоретичного і практичного матеріалу;
– відеоролики, аналогічно тексту, за винятком того, що збуджують у студента інший вид сенсорної пам'яті – слухову;
– картинки, анімації – краще розкривають поняття, які вимагають у студента великих ресурсів уяви.

Місцем збереження інформації в раніше описаних формах є освітня платформа. Найчастіше, вона представляє з себе веб-додаток з режимом доступу через Браузер користувача. Також, існують електронні підручники, які припускають скачування на пристрій користувача та використання як з доступом в інтернет, так і без.

Ця робота є частиною проекту по розробці освітньої платформи, яка орієнтована на вивчення математики студентами и спеціалістами не математичних спеціальностей, перш за все, з IT-сфери. Дане рішення повинно мати наступні характеристики:

a) спрямованість на людей, далеких від сфери, яка передбачає застосування знань з математики в чистому вигляді;
b) підтримка різних видів надання інформації, а саме:
   1) текстового;
   2) відео;
   3) графічного.
c) можливість використання платформи конкретним користувачем будучи одночасно студентом і викладачем;
d) наявність практичних завдання у вигляді тестів для закріплення пройденого матеріалу;
e) аналіз процесу проходження курсів студентами для складання рекомендацій щодо вивчення пройденого матеріалу.

Метою даної роботи є розробка архітектури освітньої платформи, яка володіє усіма вищезазначеними характеристиками.

Для досягнення цієї мети необхідно:

1) проаналізувати архітектурні рішення з розробки веб-додатків у будь-яких сферах життя;
2) створити архітектуру освітньої платформи, враховуючи наявність окремих модулів;
3) обрати засоби реалізації;
4) реалізувати основні модулі системи и забезпечити зв’язок між модулями в ній.

1 ОГЛЯД ПРИНЦИПІВ ПРОЕКТУВАННЯ ТА АРХІТЕКТУРНИХ РІШЕНЬ У СФЕРІ НАВЧАННЯ

Перед проектуванням архітектури розроблюваної платформи, необхідно дослідити існуючі рішення, зрозуміти, чому ті чи інші використовуються: які переваги вони реалізують і які недоліки містять. Важливо розуміти, що описані нижче архітектурні рішення в сфері навчання можуть використовуватися і в інших областях діяльності людини. Отже, в аналізі архітектур доцільніше орієнтуватися на ті підходи, які зарекомендували себе як надійні, відмовостійкі і легкі в підтримці в будь-яких областях.

1.1 Використання методики SOLID в проектуванні

Оскільки SOLID є одною з найбільш популярних методик розробки програмного забезпечення, в області саме системного проектування буде доцільно розглянути її складові.

Принцип єдиної відповідальності (Single Responsibility Principle, SRP) полягає в тому, що, кожна ділянка коду може оновлюватися тільки для реалізації однієї задачі. Якщо ділянка коду реалізує кілька завдань і змінюється тільки для різного його використання в різних частинах програми, слід продублювати цю ділянку, по одному екземпляру для кожного реалізованого завдання. Тут доводиться відступати від загальноприйнятого принципу усунення дублювання.

Використання даного принципу прияє усуненню неявно можливих помилок, що виникають через наявність в коді наступних інваріантів:

- правильно написаний код часто використовується кілька разів;
- в кожному місці, де він використовується, очікується постійна поведінка, що приводить до одного і того ж результату;

– при його використанні в декількох місцях результат повинен задовольняти кожному конкретному місцю використання.

Тому всі місця використання будь-якої компоненти (ділянка коду, процедура, клас) повинні знаходитися в зоні єдиної відповідальності (Single Responsibility). [1]

Принцип відкритості/закритості (Open-Closed Principle, OCP) потребує оптимальне планування оновлення коду так, щоб в процесі вирішення нових завдань програмістом, шляхом додавання нового коду, не треба було оновлювати старий код. У загальному випадку, код повинен бути відкритий для доповнення і закритий для зміни. Мета даного принципу – мінімізація трудовитрат при додаванні нового коду і усунення неявно створюваних помилок, що виникають через наступні причини:

– для вирішення нового завдання розробник повинен додати нові компоненти або змінити поведінку старих;
– додавання нової тягне за собою перевірку роботи в місці використання;
– частіше в практиці розробки, витрати на додавання набагато менше витрат на зміну.

Останнє робить очевидним переваги використання принципу Open-Closed. При цьому існує маса прийомів для підтримки архітектури в спочатку запланованому стані, і в стані, коли реалізація нового завдання зводиться тільки до додавання нових компонент. Серед таких прийомів можна виділити:

– патерни проектування;
– динамічні бібліотеки посилань (dll);
– розвиток мов програмування, а саме підтримка раніше написаного коду. [2]

1.2 Архітектура освітньої платформи EDX

EDX – надійна платформа для освіти і навчання, що є глобальною некомерційною організацією. Її заснували в 2012 році два університети: Массачусетський технологічний інститут та Гарвардський університет. На поточний момент існує понад 150 шкіл, некомерційних організацій, міжнародних організацій і корпорацій, які співпрацюють з EDX, пропонуючи цій платформі розмістити у себе безліч курсів.

Курси EDX складаються з так званих послідовностей, тривалістю в один тиждень. Кожна послідовність містить у собі короткі відеоролики, з інтерактивними вправами, в яких учні відразу можуть практикувати раніше почуті концепції. Також присутня можливість організації дискусійних груп. Вони являють собою простір, що реалізує подобу онлайн-дискусійного форуму, де студенти можуть розміщувати і переглядати питання інших студентів, бачити питання і коментарі один до одного. [3]

Крім аспекту самої освіти, дана платформа використовується для досліджень в галузі дистанційного навчання. В якості основного джерела даних для аналізу розглядаються кліки користувача. Також в процесі аналізу процесу навчання в розрахунок беруться дані демографії конкретної особи.

EDX побудована на платформі Open EDX – програмному забезпеченні з відкритим кодом. В даному проекті є декілька основних компонентів, які обмінюються між собою даними за допомогою задокументованих API, а саме – LMS і Studio (див. рис. 1.1). [4]

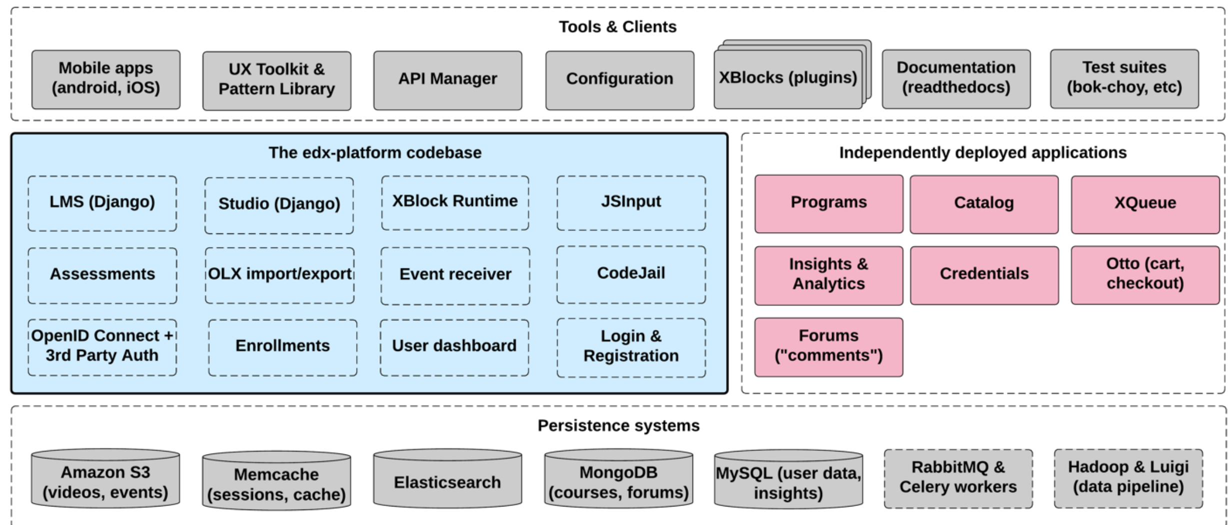


Рисунок 1.1 – Архітектура EDX

Вони служать ресурсами для управління навчанням і створення курсів відповідно. Ці служби підтримуються іншими, в свою чергу автономними веб-службами (далі IDA – independently deployed applications). Поступово, весь функціонал EDX планується розділити на нові IDA. Така стратегія зробить управління всією кодової базою платформи набагато легшим, що дозволить розробникам простіше підходити до проекту і доповнювати його.

Такий підхід відповідає практично на всі фундаментальні питання, що виникають при проектуванні архітектури веб-додатку:

1) Які границі (принцип єдиної відповідальності) кожної окремої служби продукту?
2) Як ізолювати сервіс (в деяких випадках - додаток) від інших?
3) Які основні бізнес-області і обмежені контексти? [5]

Відповіддю на усі вищевказані питаня сприяли роботи Еріка Еванса, автора таких книг як «Предметно-орієнтоване проектування. Структура складних програмних систем »,« Чистий архітектура. Мистецтво розробки програмного забезпечення »,« Високо-навантажені програми. Програмування,

масштабування, підтримка ». Найважливіші висновки з його робіт можна сформулювати так:

1) розподілити архітектуру необхідно за «посадами» або «обов'язками», а не за даними. Тобто, не сервіси на основі сутностей, а послуги на основі життєвого циклу;
2) оскільки концепції предметної області є локальними по відношенню до його обмеженому контексту, кожен контекст володіє семантикою свого власного іменування і управління своїми власними даними, оптимізованими для його власних служб. [6]

Архітектура EDX, яка базується на роботах Еріка Еванса і принципах проектування SOLID, можна зобразити у вигляді діаграми на рисунку 1.2.

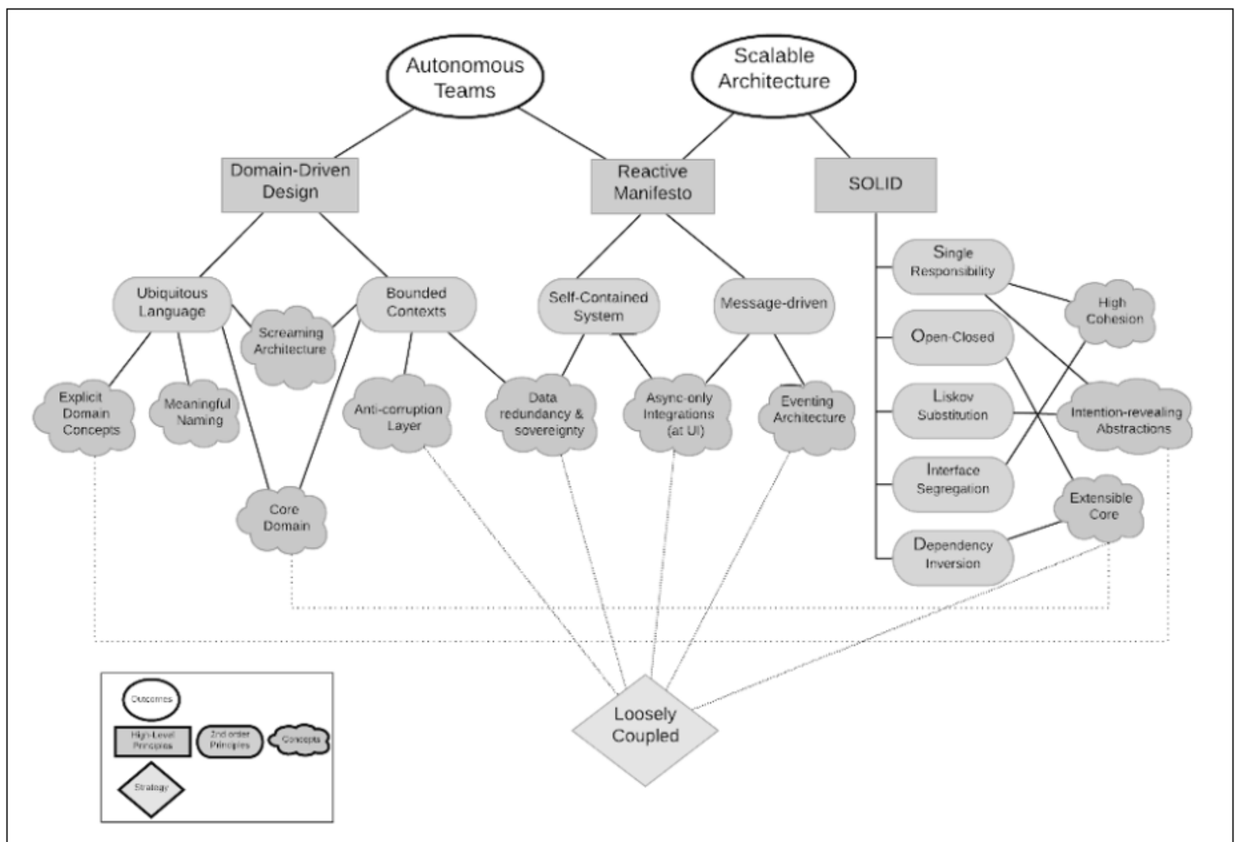


Рисунок 1.2 – Кінцевий варіант архітектури платформи EDX

1.3 Архітектура освітньої платформи Stepik

Stepik – це хмарна платформа, призначена для створення і поширення інтерактивного освітнього контенту, а також для надання різних типів автоматично оцінюваних знань зі зворотним зв'язком в режимі реального часу. Ці можливості передбачені для всіх користувачів платформи, які при створенні можуть використовувати текст, відео та різноманітні завдання з автоматичною перевіркою. В ході навчання для студентів передбачена можливість обговорення викладеного матеріалу і отримання зворотного зв'язку від викладача курсу.

У список організацій, що випустили курси на цій платформі, входять «Яндекс», «JetBrains», «Samsung», «Mail.ru». Найбільш важливою частиною платформи є система автоматизованої перевірки завдань – використовується в ряді курсів на платформі «Coursera», включаючи курси з біоінформатики від Каліфорнійського університету в Сан-Дієго і курс з аналізу даних від НДУ «Вища школа економіки».

Поступово під час підтримки та оновлення платформи були випущені програми під Android і IOS, чат-бот в Telegram та плагін для середовища розробки IntelliJ IDEA. Очевидно, таке широке поширення передбачає наявність декількох клієнтів і специфічних API для кожного. Даний факт, що говорить про наявність ряду різних API, структура проекту в системі контролю версій GitHub, не дивлячись на закритий для публічного доступу внутрішній устрій платформи, дозволяють зробити висновок, що використана архітектура є мікросервісною. [7]

1.4 Основні архітектурні рішення веб-додатків

В ході аналізу ринку онлайн освітніх платформ, було виділено два основних види веб-додатків: додаток-Моноліт і Мікросервісний додаток. Слід зазначити, що не існує однозначної відповіді на питання, яке рішення з двох доцільніше використовувати в розробці веб-додатку. Проте, під кожну конкретну задачу можна побудувати комбінацію вищезазначених архітектурних рішень. Кожне з них має свої переваги й недоліки, в яких слід було розібратися перед проектуванням архітектури для розроблюваної платформи.

1.4.1 Монолітна архітектура

Монолітний додаток являє собою додаток, що доставляється через єдине розгортання у вигляді, будь то WAR-додатоку або Node, з однією точкою входу. Як приклад такого додатка розглянемо інтернет-магазин. Він містить такі елементи, як: UI, бізнес-логіку і шар даних. Так, при побудові програми всі складові будуть управлятися з одного модуля (див. рис. 1.3).

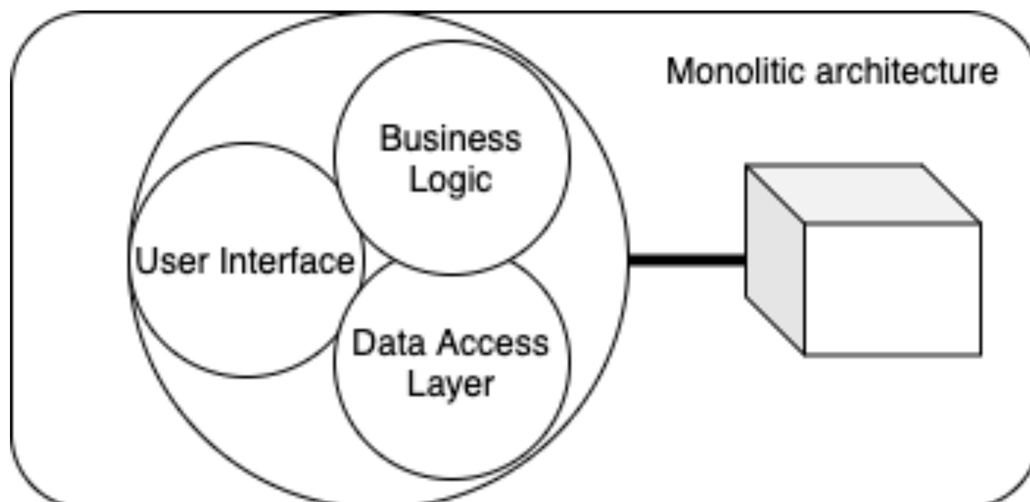
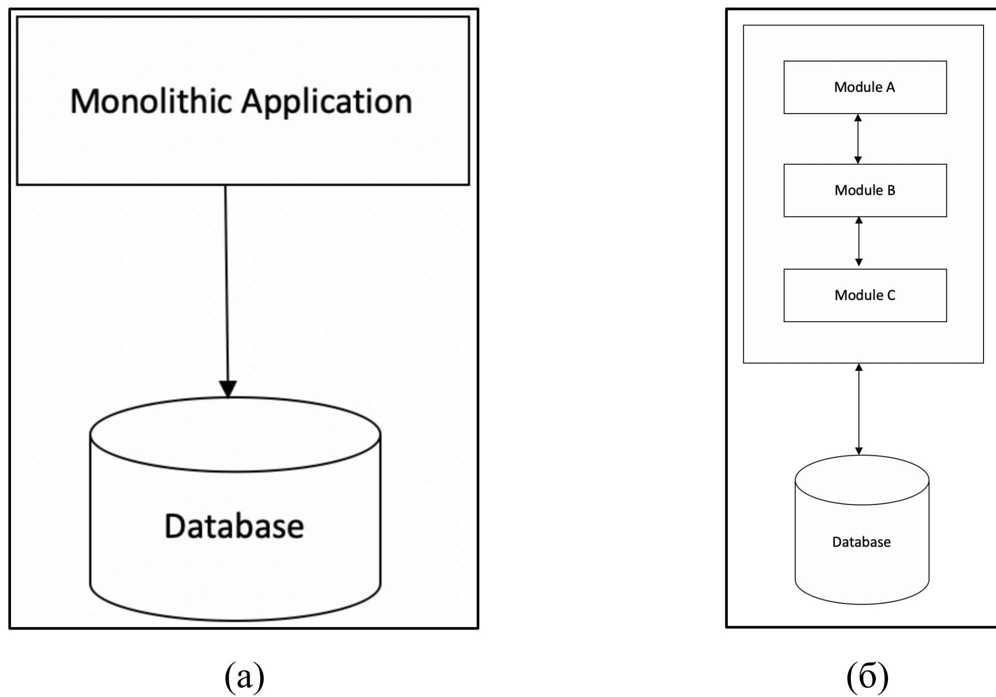


Рисунок 1.3 – Процес роботи Моноліт-додатку

Переважно існує два типи монолітної архітектури: з одним процесом і модульна. [8] Якщо весь код програми розгортається як єдиний процес, мова йде про архітектуру з одним процесом (див. рис. 1.4 а). У випадку ж, коли один процес додатку складається з декількох модулів, кожен з яких може працювати незалежно, передбачається модульна архітектура (див. рис. 1.4 б).



(а) (б)

Рисунок 1.4 – Типи монолітів

Слід зазначити, що в разі модульної монолітної архітектури, самі модулі мають інтерфейси для зв'язування між собою. База даних загальна, і всі модулі її використовуюсь для своїх операцій. Не дивлячись на, здавалося, єдине, що пов'язує модулі, роблячи їх залежними один від одного, важливо, щоб в кінцевому підсумку модулі були об'єднані в один файл для розгортання.

Проаналізувавши дане архітектурне рішення, можна зробити висновок, що наявність таких сильних зв'язків між модулями в разі модульного моноліту і повного зв'язку в разі моноліту з один процесом породжує безліч критичних недоліків.

Монолітна архітектура доцільна у використанні в невеликих групах розробників, з цієї причини вона ідеально підходить для стартапів. Проте, в процесі розробки продуктів, їх розширення, збільшення обсягу даних, структура таких проектів стає розмитою. Кодова база стає вкрай складною в розумінні, особливо для нових розробників. Також пошук небажаних залежностей стає складніше, а через збільшення обсягу коду перевантажуються інтегровані середовища розробки (Integrated development environment, далі – IDE). [9]

Наступні недоліки використання взаємопов'язані, тому можна об'єднати їх в певну групу:

– складність впровадження нових технологій. Якщо в додаток необхідно додати функціонал, який використовує нові версії мови, фреймворка, на якому розробляється додаток, розробники зіткнутися з деякими труднощами. Додавання нової технології означає переписування всієї програми, оскільки код не всіх його частин буде назад-сумісним;
– обмежена гнучкість. Кожне оновлення вимагає повторного розгортання. Таким чином, розробник, який робіть одну задачу, скажімо, всього лише по впровадженню функціоналу для роботи з новою базою даних, змушений чекати, поки весь проект не буде запущений, щоб перевірити внесені ним зміни. Таким чином, гнучкість розробки істотно знижується.

В кінцевому підсумку, не можна сказати, що монолітна архітектура застаріла, в деяких навіть великих проектах її використання доцільно. Наприклад, компанія Etsy як архітектуру для своїх продуктів використовує монолітну, не дивлячись на зростаючу популярність сервіс-орієнтованих рішень. [10]

1.4.2 Сервіс-орієнтована архітектура

Сервіс-орієнтована архітектура (далі – SOA) – стиль розробки архітектури, при якому передбачається створення модульного додатку, з дискретними і слабо зв'язаними програмними компонентами (агентами), які виконують однозначно закріплені за ними функції. Ці компоненти поділяються на дві категорії: постачальники і споживачі сервісів.

З точки зору побудови архітектури, SOA містить такі компоненти:

– клієнт – відповідає за ініціалізацію деякої події;
– сервіс – компонент програми з чітко визначеною функціональністю;
– репозиторій сервісу – забезпечує зв'язок сервісу з його сховищами інформації;
– сервісна шина (необов'язково) – забезпечу взаємодію між усіма компонентами. [11]

Загальну концепцію можна сформулювати наступним чином – проектування додатку передбачає наявність можливості повторного використання та оновлення існуючих модулів, а також легкої інтеграції та заміни нових. Ці можливості сприяють наявності наступних переваг даного архітектурного рішення:

1) повторне використання сервісів. Через природу самих сервісів, в SOA додатках їх можна використовувати повторно, без впливу на інші сервіси;
2) легкий супровід. Окремі служби легше налагоджувати і тестувати, ніж великі фрагменти коду або весь код повністю в разі монолітного додатку;
3) паралельна розробка. Дана властивість є одною з ключових переваг в SOA. Вона надає можливість паралельної розробки окремих модулів, що істотно підвищує швидкість самої розробки. [12]

1.4.3 Мікросервісна архітектура як похідна від SOA

Проаналізувавши архітектури існуючих освітніх платформ, факт популярності, надійності і доцільності SOA стає незаперечним. Розглянемо більш конкретний принцип побудови архітектури – принцип Мікросервісной архітектури.

Мікросервісна архітектура (далі – MSA) – принципова організація розподіленої системи на основі мікросервісів і їх взаємодії один з одним і з навколишнім середовищем по мережі. Один з варіантів такого рішення можна бачити на Рисунку 1.5.

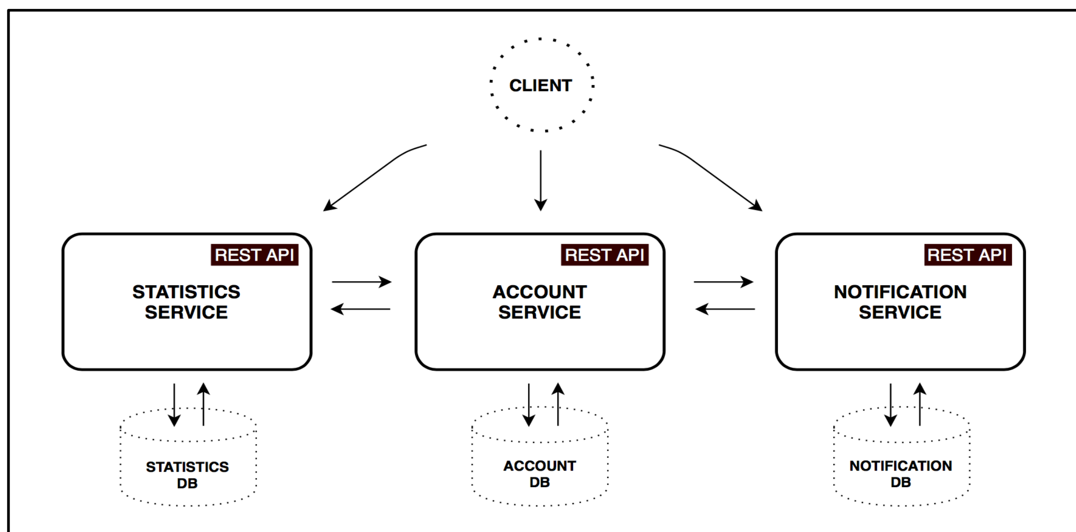


Рисунок 1.5 – Загальна схема MSA

MSA успадковує всі принципи проектування, характерні SOA, проте також і доповнює їх. [13] Саме визначення MSA неповно відповість на питання, чому варто використовувати саме її. Розуміння суті архітектури базується на визначенні і властивостях самого сервісу. Визначимо його основні характеристики:

1) невеликий;
2) незалежний;

3) будується навколо бізнес-логіки й використовує її обмежений контекст;
4) взаємодіє з іншими сервісами по мережі або за допомогою шини даних, або на основі патерну «Smart endpoints and dumb pipes».

Далі визначимо, що в собі містить кожна характеристика і якими засобами досягається її наявність.

Невеликий. Дана характеристика визначає «розмір» сервісу, а саме, що він повинен бути таким, щоб виконувалася одна з наступних умов:

1) один сервіс здатна розвивати одна команда розробників;
2) контекст роботи одного сервісу підлягає сприйняттю однією людиною.

Незалежний. MSA по своєму внутрішньому устрою є образом патернів «High Cohesion» і «Low Coupling». Їх комбінація забезпечує вкрай ефективну взаємодію з додатком, чиї принципи він використовує.

У випадку з «High Cohesion», береться до уваги скоріше факт наявності чималої кількості зв'язків між модулями програми, ніж серйозний недолік, який ускладнює розуміння логіки модулів, їх модифікацію і автономне тестування.

З «Low Coupling», для мікросервісной архітектури були запозичені ознаки правильного проектування системи зі слабо зв'язаними модулями. У підсумку, комбінуючи ці підходи, виходить система, яка відповідає загальним показниками гарної читаності.

Незважаючи на те, що будь-який масштабний продукт пишеться з розбивкою на компоненти, в разі моноліту, загальна кодова база сприяє порушенню принципу низькою пов'язаності. В цьому випадку виникає ймовірність спагетті-коду.

У той же час методологія розбиття на окремі мікросервіси вимагає дотримання жорсткого поділу як компонентів, так і їх бібліотек. Це необхідно робити для відповідності критеріям незалежності. У підсумку, кожен мікросервіс працює в своєму процесі і повинен явно позначити свій API.

Побудова навколо бізнес-логіки. Основний принцип такого побудови – сформувати зону відповідальності мікросервіса навколо бізнес-логіки таким чином, щоб вона була компактною, а її зв’язки з іншими сервісами були досить повно формалізовані. Чим краще виконуються ці умови, тим логічнішим стає створення нового мікросервіса.

Smart endpoints and dumb pipes. У первинному варіанті MSA передбачала відсутність сервісної шини (далі – ESB), однак ефективність її використання, що породжує ще менший розмір самих мікросервісів, змушує розробників її використовувати. Проте, застосовуючи підхід без використання ESB, організація зв'язку між сервісами переноситься на самі сервіси, іншими словами – складність інтеграції з центрального вузла переноситься на інтегровані компоненти за допомогою, так званих, «розумних кінцевих точок».

Такий пристрій сервісу наділяє MSA в цілому наступними перевагами:

1) покращує ізоляцію збою компонентів: так, функціонал решти додатку, яка в своїй роботі не зачіпає компонент, який вийшов з ладу, буде працездатною, і кінцевий користувач може навіть не помітити несправностей;
2) усуває «схильність» додатку до одного технологічного стеку: наприклад, може виникнути необхідність використання нового функціоналу, чиє впровадження, в порівнянні з монолітом, передбачає налаштування набагато меншої кількості залежностей, а в разі несправності – вкрай просто скасувати всі зміни;
3) створює сприятливе середовище для впровадження в процес розробки нових співробітників, яким в межах їх завдань, не знадобиться вивчати все додаток, а то пов'язані з їх завданнями сервіси. [14]

Не дивлячись на перераховані вище істотні переваги MSA, використання цього рішення накладає певні труднощі на процес розробки продукту:

1) розробка розподілених систем може бути важкою. Під цим мається на увазі, що всі компоненти тепер незалежні сервіси, тож слід акуратно обробляти запити, що проходять між модулями. Може виникнути сценарій, коли один модуль не відповідає, змушуючи писати додатковий код, щоб уникнути збою системи. Слід зазначити, що вирішення цієї проблеми шляхом написання додаткового коду неможливо буде досягти, якщо віддалені виклики чутливі до тимчасової затримки між моментом надсилання запиту і моментом отримання відповіді;
2) безліч баз даних і управління транзакціями може викликати масу складнощів;
3) тестування MSA ймовірно буде громіздким. Якщо для монолітного додатку нам потрібно підняти базу даних, запустити сервер і прогнати тестові сценарії, то в мікросервісах, кожен окремий сервіс і його бази даних (якщо такі є) повинні бути запущені часто в певному порядку перед тим, як почати тестування;

Не дивлячись на такий перелік серйозних недоліків, вони легко можуть бути розв'язні. Рішенням, в даному випадку, виступає повне розуміння бізнес-логіки додатку.

1.4.4 Сполучний модуль

Сполучний модуль або шлюз, за своїми характеристиками те ж саме, що і будь-який інший API-шлюз. Але при використанні його в мікросервісній архітектурі, він надає інтерфейс, який використовується для доступу до мікросервісів. Розміщення шлюзу в структурі MSA демонструється на рисунку 1.6.

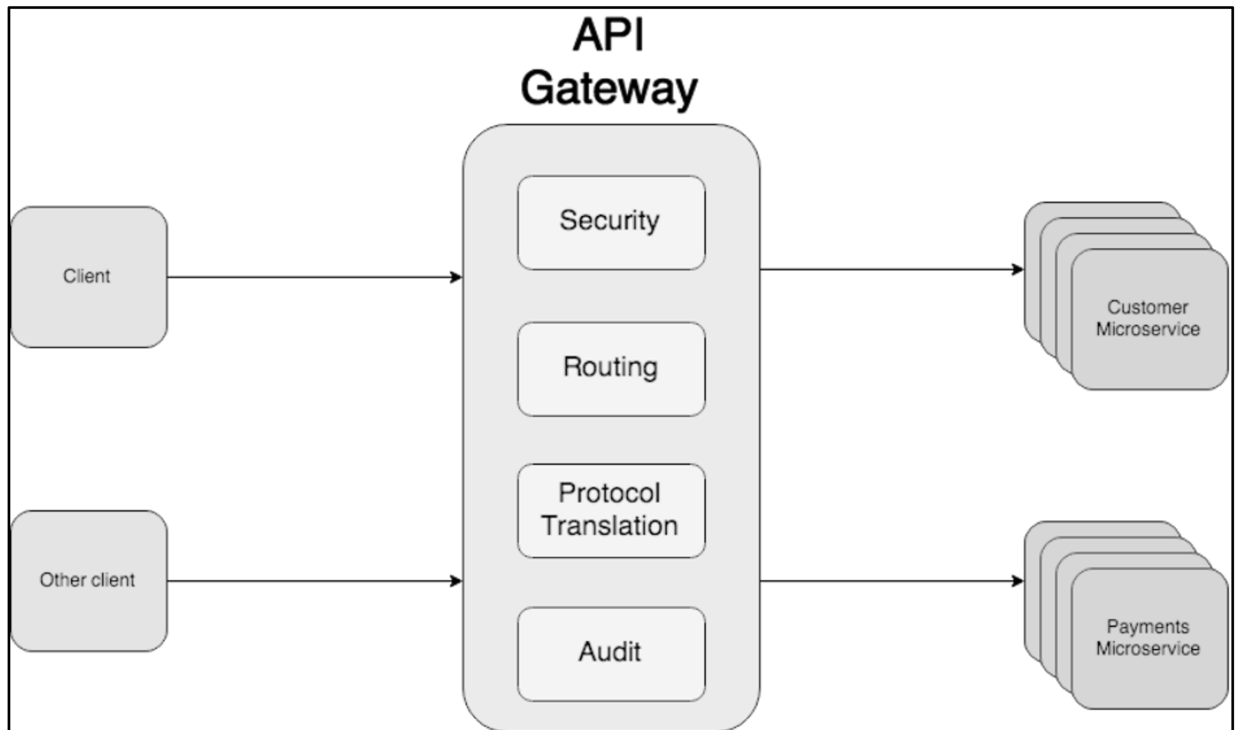


Рисунок 1.6 – Використання Gateway

Не дивлячись на визначення, Gateway робить більше, ніж просто створює єдиний інтерфейс для однієї програми. [15] Один шлюз мікросервісів може створювати кілька API – по одному для кожної платформи, яку необхідно підтримувати. Наприклад, може знадобитися підтримка додатків для смартфонів, браузерів і серверних додатків, для кожного з яких потрібен різний набір функцій мікросервісів, а також може знадобитися інший набір протоколів передачі інформації. Шлюз може створити власний API для кожного з цих клієнтів (див. рис. 1.7), щоб другий в свою чергу маніпулював тільки тими функціями, які йому потрібні.

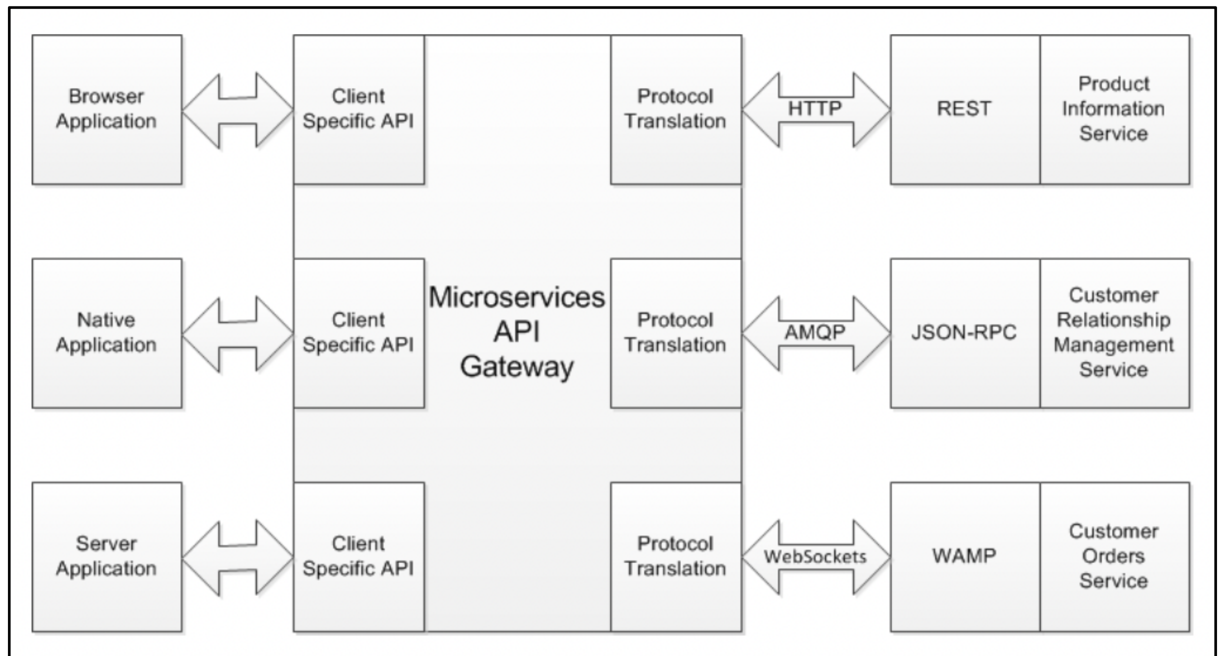


Рисунок 1.7 – Декілька API для кожного з клієнтів

Об'єднання мікросервісів за допомогою Gateway – це тільки частина організації MSA в додатку. Також, у кожного мікросервіса можуть бути особливі потреби. Найбільш часта – використання певного набору протоколів для доступу. Спроба отримати доступ до кожного мікросервісу, використовуючи його конкретне налаштування протоколу, в клієнтському додатку буде складно. Шлюз в свою чергу забезпечує трансляцію протоколів, так що кожна мікрослужба отримує запити з використанням необхідного протоколу, але при цьому клієнти використовують тільки на єдиний протокол, який вони найкраще підтримують.

На перший погляд здається, що використання шлюзу є ідеальним вирішенням проблеми роботи з мікросервісами. Проте, і у такого рішення є свої недоліки.

Найбільш очевидним з них є той факт, що в продукті з'явився ще один модуль, який вимагає налаштувань і підтримки. Сам процес налаштування займає куди менше часу, ніж написання коду для кожного мікросервісу, що забезпечує зв'язок між усіма модулями.

Інша проблема полягає в тому, що використання шлюзу мікросервісів створює додаткову точку відмови. Так, в разі відмови Gateway, зв'язок між усіма модулями неможливий і додаток стає непрацездатним.

У підсумку, використання шлюзу мікросервісів робить роботу більш ефективною. Однак для отримання такої переваги використовуваний сервер повинен підтримувати неблокуючий асинхронний ввід-вивід.

1.5 Вимоги до системи

Проаналізувавши існуючі архітектурні рішення, було прийнято використовувати MSA. Також слід передбачити такі функціональні групи, які будуть реалізовані у вигляді самостійних сервісів і сполучного модуля:

- Сервіс забезпечення безпеки. В його обов'язки входить маніпуляція з користувацькими даними, а саме:
  1) створення користувача;
  2) забезпечення механізму авторизації по принципу OAuth2;
  3) підтвердження будь-яких призначених для користувача дій на клієнті – перед тим, як дозволити/заборонити будь-яку дію користувача, логіка роботи програми передбачає звернення до даного сервісу.
- Аналітичний сервіс. Є рекомендаційною системою для отримання прогнозів про ймовірну оцінку для студента за весь курс на основі даних про вже пройдені їм тести. Також для демонстрації викладачеві загальної успішності за курсом, і в яких його областях у студентів залишилися невирішені питання.
- Сервіс по управлінню контентом. В його обов'язки входить управління таким базовими операціями як «Create, Read, Update, Delete» (далі CRUD), по відношенню до всіх об'єктів, що використовуються для надання навчальної інформації студенту. Також повинен містити в собі

«точку дотику» з Аналітичним сервісом для прямого (закритого) зв'язку між ними. Ця залежність, наявна по причині цілісності даних про тест, які зберігаються на поточному сервісі, використовується другим для підрахунку оцінки студента за курс.

– Сервіс тематичного моделювання та інтелектуальної обробки тексту. Визначає до яких тем віднести ту чи іншу текстову частину курсу. Алгоритм побудови на вході отримує колекцію документів, після чого на виході кожному ставиться у відповідність числовий вектор, кожна координата якого визначає ступінь приналежності конкретного тексту до тієї чи іншої теми.
– Сполучний модуль. Використовується для забезпечення зв'язку всіх сервісів з клієнтом за допомогою http-протоколу. Є транслятором відповіді сервісу, будь то відповідь про успішно виконаний запит або відповідь про помилку, на клієнтську дію.

В результаті проектування одержимо наступну архітектуру розроблюваної освітньої платформи (див. рис. 1.8):

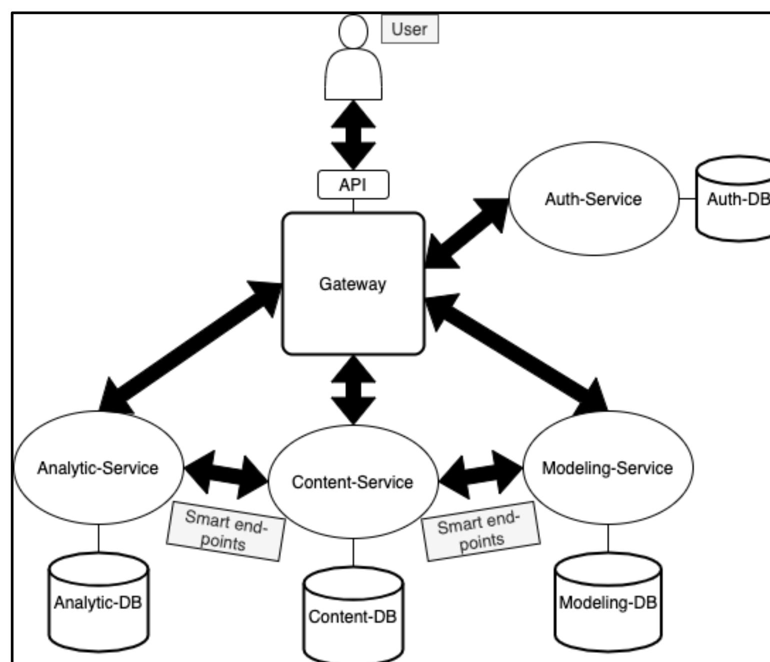


Рисунок 1.8 – Кінцевий варіант архітектури

2 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ОСНОВНИХ СЕРВІСІВ ПЛАТФОРМИ ДІСТАНЦІЙНОЇ ОСВІТИ

Для забезпечення безпечного доступу до даних освітньої платформи, зокрема: до курсів, уроків, аналітичних даних і так далі, перед початком розробки слід визначити:

1) Які користувачі існують в системі?
2) Які права на дані мають?

Дане рішення передбачає 3-х користувачів:

1) адміністратор. Має будь-який доступ на всі ресурси платформи;
2) користувач. У свою чергу ділиться на дві ролі користувачів: студент, викладач. Слід зазначити, що специфіка платформи передбачає однакові права доступу до ресурсів курсу на рівні таблиць, та різні на рівні окремих записів. Так, користувач-студент може створити курс і бути по відношенню до нього викладачем;
3) Гість. Не має доступу до матеріалів курсів.

2.1 Обмін інформацією

Процес обміну інформацією в загальному випадку, коли він ініціюється діями клієнта проходить за наступним сценарієм:

1) користувацька дія. Починає процес обміну інформацією між модулями – запит дійшов до сполучного модуля;
2) фільтрація запиту. Перед тим як запит потрапить на контролер, створений під конкретну дію користувача, запит потрапляє в фільтр;
3) виконання сполучним модулем запиту на конкретний сервіс. З контролера викликається метод обробки запиту, який в свою чергу, якщо фільтр «пропустив», шле запит на сервіс;

4) виконання бізнес-логіки сервісом. Запит від Gateway потрапляє на конкретний контролер сервісу, де викликається метод, що виконує певну бізнес-логіку;
5) відправлення відповіді. Якщо логіка успішно виконана – йде відправка відповіді клієнту по тому ж маршруту: на Gateway, після чого на клієнт.

2.2 Засоби реалізації

В якості основного інструменту реалізації було обрано фреймворк Spring Boot, а в якості мови програмування – відповідно Java (v11).

Даний вибір обумовлений безліччю переваг як перед базовим фреймворком для Spring Boot – Spring, так і перед іншими фреймворками і мовами програмування. Це рішення має вкрай широкий функціонал, особливо: легке управління залежностями, автоматична конфігурація, і вбудована контейнеризація сервлетів. [16]

Прискорення процесу управління залежностями досягається шляхом пакування необхідних сторонніх, для кожного типу додатку Spring, і подальшого їх надання розробнику у вигляді starter-пакетів, наприклад: spring-boot-starter-web, spring-boot-starter-data-jpa. Раніше згадані пакети використовуються відповідно для роботи програми з мережею і для взаємодії з шаром даних. Starter-пакети являють собою дескриптори залежностей, які можна включити в додаток. Такий підхід дозволяє досягти високого рівня універсальності для більшості рішень на Spring Boot.

Під автоматичною конфігурацією розуміється спроба фреймворка налаштувати Spring-додаток на основі всіх jar-залежностей. Так, в проекті використовувався пакет «Spring-boot-starter-web», що в свою чергу означає автоматичне конфігурування наступних бінів: DispatcherServlet, ResourceHandlers, MessageSource. Ще одним прикладом більш глибшої конфігурації є налагодження роботи з даними без самої бази даних. Якщо

логіка додатка передбачає такий сценарій, то Spring Boot автоматично налаштує базу в пам'яті, без будь-якої додаткової конфігурації з боку розробника (наприклад, H2). [17]

2.3 Архітектура сервісу

Для розробки веб-додатків на Spring Boot використовується безліч різних підходів до організації процесів усередині системи. Однак найбільш поширений і логічний наступний:

a) файли конфігурацій. Виконують налаштування проекту, підключення необхідних залежностей, які необхідні для побудови поведінки програми. Місцезнаходження - «/»;
b) доступ до даних. Місцезнаходження - «/dao» (Data Access Object). Містить файли, що описують взаємодію додатку з базою даних:
   1) опис сутностей;
   2) опис репозиторіїв;
   3) допоміжні дані, які стосуються сутностей.
c) робота з даними. Місцезнаходження - «/service». Містять файли, які реалізують бізнес-логіку додатку з використанням даних:
   1) наприклад, реалізація механізму авторизації, з використанням сутності користувача (його username і password);
   2) реалізація CRUD.
d) робота з мережею. Місцезнаходження - «/controller». Містить файли, які займаються обробкою вхідних запитів.

Перераховані файли і папки не єдині в проекті, проте саме ці демонструють внутрішній устрій та архітектуру більшості сервісів. [18]

2.4 Реалізація базових сервісів

Для реалізації механізму безпечного (авторизованого) управління контентом, потрібно спроектувати відповідно сервіс забезпечення безпеки та сервіс управління контентом. Також, найважливішим елементом є сполучний модуль, який реалізує «спілкування» за допомогою http-протоколу між сервісами.

2.4.1 Сервіс забезпечення безпеки

Специфіка використовуваного фреймворка передбачає можливість підключення всіх необхідних бібліотек вже на етапі створення нового проекту. Доступ до даного інструменту здійснюється за адресою «https://start.spring.io/». У процесі створення нової програми пропонується встановити основні метадані і налаштування (див. рис. 2.1).

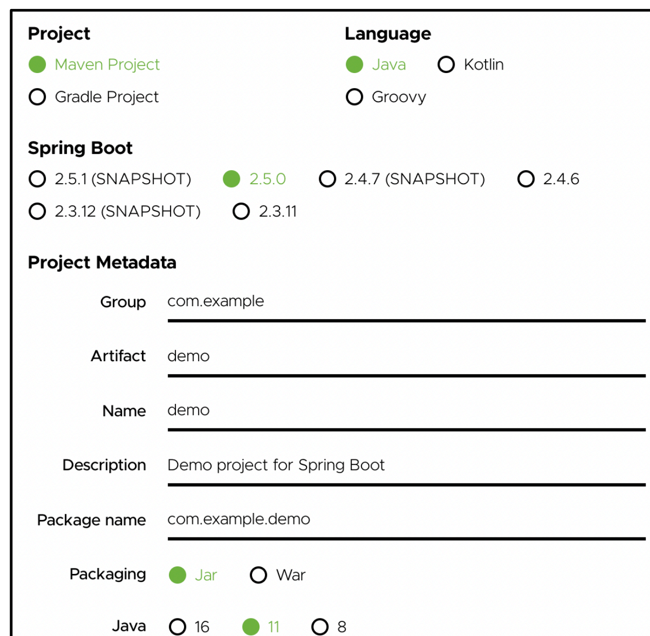


Рисунок 2.1 – Перший етап початкового налаштування

Далі йде етап підключення до проекту необхідних на поточний момент залежностей (див. Рис. 2.2). У процесі розробки, зрозуміло, передбачена можливість оновлення списку використовуваних залежностей.

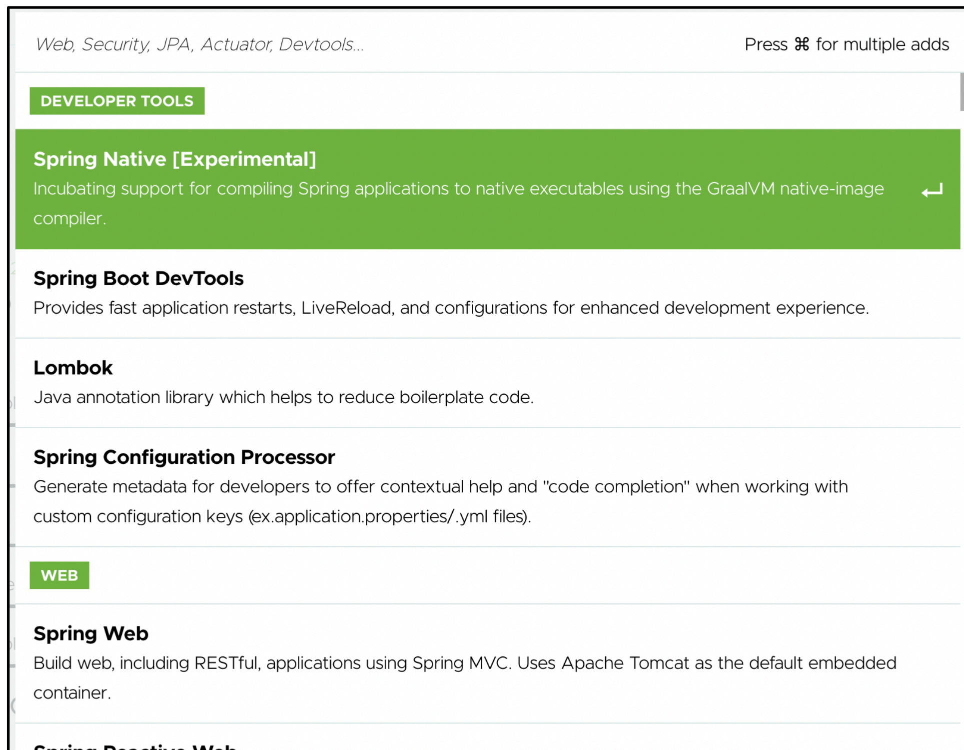


Рисунок 2.2 – Вікно вибору залежностей

Після налаштування всіх необхідних параметрів і підключення залежностей, генерується архів з уже повністю готовим для запуску проектом.

Як залежності для AuthService були використані наступні:

- spring-boot-starter-data-jpa. Використовується для роботи з базами даних;
- spring-boot-starter-web. Використовується для роботи з мережею;
- hibernate-validator. Необхідний для забезпечення можливості використовувати в проекті реалізовані валідатори у вигляді анотацій як над окремими полями, так і нам цілим класом;
- postgresql. Додає в проект можливість роботи з реляційною СУБД – PostgreSQL. Наприклад, дана залежність дає можливість транслювати Java-сутність (або клас) в таблицю в базі даних;

- lombok. Вкрай потужний і широко використовуваний інструмент розробки додатків на основі фреймворку Spring Boot. Надає перелік анотацій, чиїм функціоналом є досягнення більш короткої записи коду. Так, анотація `@Getter` даної бібліотеки, створює публічні методи для отримання всіх полів класу, без необхідності вручну реалізовувати кожен;
- spring-boot-starter-test. Надає інструменти автоматичного тестування додатку;
- spring-boot-starter-security. Дає розробнику можливість настроювання механізму забезпечення мережевої безпеки.

Для реалізації механізму авторизації OAuth2 необхідно було передбачити зберігання даних про сесії користувача. Основні дані, якими оперує даний механізм, є два токена: токен доступу (access-token) і токен оновлення (refresh-token). Так, для зберігання цих даних була створена відповідна таблиця (див. Лістинг 2.1).

```
create table auth(
    id serial primary key,
    access_token varchar(255) not null unique,
    refresh_token varchar(255) not null unique,
    ip_address varchar(255) not null,
    uuid varchar(36) not null,
    expires_in timestamp with time zone not null,
    created_at timestamp with time zone not null default
current_timestamp,
    updated_at timestamp with time zone not null default
current_timestamp);
```

Лістинг 2.1 – Скрипт створення таблиці «auth»

Механізм надання користувачеві токена доступу, іншими словами, login виглядає наступним чином:

1) пошук користувача по пошті, вказаній в полі «пошта» при авторизації;

2) перевірка введеного пароля на збіг;
3) створення запису про нову сесію користувача;
4) збереження даних про сесії в базі даних.

У разі, коли не виконується будь-яка із зазначених перевірок, з сервісу викидається помилка (див. Лістинг 2.2).

```
@Override
public AuthDto login(LoginRequestDto dto) {
    User user =
userRepository.findByEmail(dto.getEmail()).orElseThrow(Unauthori
zedException::new);
    verifyPassword(dto.getPassword(),
user.getUserCredentials().getEncodedPassword());

    Auth auth = Auth.builder()
            .accessToken(UUID.randomUUID().toString())
            .refreshToken(UUID.randomUUID().toString())
            .expiresIn(LocalDateTime.now().plusHours(3))
            .ipAddress(dto.getIpAddress())
            .user(user)
            .build();

    Auth saved = authRepository.save(auth);
    return mapper.buildAuthToAuthDto(saved);
}
```

Лістинг 2.2 – Процес «login»

Тут слід зазначити, що вибір `varchar` як тип для поля в таблиці, що містить дані користувача, обумовлений тим, що під час реєстрації необхідно заздалегідь знати ідентифікатор користувача, до моменту збереження в базу даних.

Для спілкування із зовнішнім середовищем використання самих сутностей є часто неприпустимим або не потрібним. Наприклад, клієнтові знати ідентифікатор будь-якого об'єкта зовсім не потрібно, а у випадку з сутністю користувача – зберігати на клієнті його пароль. Щоб уникнути таких ситуацій в якості даних для спілкування з зовнішнім середовищем використовуються DTO. За своєю суттю – це об'єкт виду «value-object», який зберігає дані. DTO ж поділяються на ті, що використовуються в запиті, і ті, що використовуються у відповіді.

2.4.2 Сполучний модуль. Обробка помилок. Фільтр запитів

Найважливішою відмінністю Gateway від звичайного сервісу є його основна логіка, а саме – відправка запитів на різні адреси мережі, точки дотику кожного сервісу. Як і сервіс, містить DTO і контролери для спілкування з зовнішнім середовищем, якою в разі Gateway, є клієнтська програма. Проте, на відміну від сервісу, не містить в собі файлів управління даними, що логічно, адже основним функціоналом цього модуля додатка є створення зв'язку між усіма.

Архітектуру Gateway можна описати таким чином:

1) файли конфігурації. Той же функціонал і місце розташування, що і в разі сервісу;
2) робота з мережею. Аналогічно;
3) створення зв'язку між сервісами. Функціонал, міститься в двох каталогах: «/client» і «/service». У першому випадку простежується якась схожість з репозиторіями, оскільки «клієнти» роблять запити (див. Лістинг 2.3) в інший шар додатку як одного цілого, отримують звідти якісь дані і викликаються в сервісах для реалізації бізнес-логіки.

```
@Override
public UserRegistrationDto register(UserRegistrationDto dto) {
    ResponseEntity<UserRegistrationDto> responseTemplate =
        restTemplate.postForEntity(REGISTRATION_URL, dto,
UserRegistrationDto.class);
    return responseTemplate.getBody();}
```

Лістинг 2.3 - Виконання запиту на AuthService для реєстрації

Самі методи, які виконують запити на конкретні частини програми, викликаються в сервісах, і крім цього, другі можуть реалізовувати додатковий функціонал, такий як: обробка помилок, перетворення даних і багато іншого. Однак в даному проекті, вони лише викликають методи виконання запитів з «/client».

В ході реалізації даного модулю, був використаний принцип «одному компоненту – одна логіка». Так, перераховані вище можливості сервісів Gateway реалізовані в різних компонентах. Розглянемо компонент обробки помилок.

Як приклад, візьмемо помилку, що виникає на AuthService через неправильний складений запит, наприклад, коли відбувається реєстрація користувача, який ввів вже існуючий email. Спершу на сервісі спрацьовує перевірка (див. Лістинг 2.4), яка в разі виконання описаного умови, викидає помилку.

```
private void validateExistenceByEmail(String email) {
    if (userRepository.findByEmail(email).isPresent()) {
        throw new AlreadyExistException(String.format("This
value \"%s\" is already in use.", email));
    }
}
```

Лістинг 2.4 - Метод, викликає до виконання логіки реєстрації

Далі згенерувану помилку слід обробити. Цьому служить спеціальний компонент – Exception handler, який є універсальним рішенням по обробці будь-яких помилок, що виникають в сервісі. Таку універсальність забезпечує Spring-анотація «@ControllerAdvice», яка дозволяє використовувати один раз створений обробник помилок для всіх контролерів, а не для єдиного, наприклад, в разі створення обробника для кожного контролера.

Для специфікації помилки, для якої необхідно викликати той чи інший обробник, використовується анотація «@ExceptionHandler()», в яку передається клас помилки, як показано в лістингу 2.5.

```
@ExceptionHandler(AlreadyExistException.class)
protected ResponseEntity<?> handleEntityAlreadyExist
(AlreadyExistException ex){
    ApiError apiError = ApiError.builder()
            .message(ex.getMessage())
            .timestamp(Timestamp.valueOf(LocalDateTime.now()))
            .code(400)
            .error("Bad Request")
            .build();
    return buildResponseEntity(apiError);}
```

Лістинг 2.5 – Оброблювач помилки «Вже існуючого користувача»

Принцип роботи таких обробників полягає в відловлюванні самої помилки, що виникає в сервісі, подальшій побудові DTO помилки з таким полями, як: повідомлення (задається в місці виникнення помилки), час виникнення, код помилки і її опис. Наступний етап – побудова суті відповіді за допомогою виклику методу buildResponseEntity. Результат, повернутий цим методом, замінить собою тіло відповіді на запит, який спровокував помилку. Таким чином, замість малоінформативної для клієнта інформації у вигляді величезного повідомлення про помилку, клієнт отримує осмислене, зрозуміле

йому повідомлення, яке в свою чергу в зручному вигляді доставляється користувачеві.

На даному етапі процес обробки помилок би закінчився, якщо б не використання мікросервісной архітектури, в якій така зручна форма запису дійде лише до Gateway. Далі необхідно таким же універсальним чином довести помилку до клієнта. Для вирішення цього завдання на шлюзі передбачений схожий обробник помилок – ServerExceptionAdviceHandler. Використовуючи ту ж інструкцію, що і обробник на стороні сервісу, він відловлює всі помилки в собі. Розглянемо на прикладі, обробник вищезгаданої помилки 400 (Bad request) виглядає так (див. Лістинг 2.6):

```
@ExceptionHandler(BadRequestException.class)
@ResponseBody
@ResponseStatus(HttpStatus.BAD_REQUEST)
protected ErrorDto handleBadRequest(
    BadRequestException ex, WebRequest request) {
    log.error(ex.getMessage());
    return gson.fromJson(ex.getMessage(), ErrorDto.class);}
```

Лістинг 2.6 – Обробка Bad request

Принцип роботи аналогічний:

1) відлов помилки;
2) побудова інформативної відповіді для клієнта. Для цього використовується бібліотека «com.google.gson.Gson»;
3) відбувається повернення сутності помилки на клієнт.

Реалізувавши такі обробники на кожному сервісі, і одного разу створивши обробник на Gateway, отримуємо повністю автоматичну систему по обробці будь-яких можливих помилок. Слід зазначити, якщо під якусь помилку не створений обробник, спрацює обробник за замовчуванням, якщо такий зрозуміло присутній в проекті. Принцип його роботи аналогічний, однак

спрацьовування відбувається охоплюючи ширший діапазон можливих помилок.

Коли додаток налаштований так, що при виникненні помилок, він не припиняє свою роботу, слід завершити механізм забезпечення безпеки в системі. Так, перед отриманням контролером всякого запиту, необхідно передбачити фільтрацію пакета по токену. Клас, який відповідає за предобробку запитів – GenericFilterBean, надає метод, перевизначення якого (див. Лістинг 2.7) дозволяє розробнику впливати на процес фільтрації запитів.

```
@Override
public void doFilter(
        ServletRequest request,
        ServletResponse response,
        FilterChain chain) throws IOException, ServletException{
    HttpServletRequest req = (HttpServletRequest) request;
    HttpServletResponse res = (HttpServletResponse) response;
    String requestURI = req.getRequestURI();
    if (requestURI.equals("/api/auth/register") ||
requestURI.equals("/api/auth/login")) {
        chain.doFilter(request, response);
    } else {
     Optional<String> tokenFromHeader = getTokenFromHeader(req);
        if (tokenFromHeader.isEmpty()) {
            res.sendError(401);
        } else if (!verifyToken(tokenFromHeader)) {
            res.sendError(401);
        } else {
            chain.doFilter(request, response);
        }
    }
}
```

Листинг 2.7 – Фільтрація запитів

Логіка роботи, наступна:

1) якщо URL запиту дорівнює тим значенням, коли не передбачається наявність токена доступу, наприклад, коли відбувається реєстрація користувача або його входить в систему, йде виклик базового методу, який «пропускає» запит, і той йде на відповідний контролер;
2) якщо ж ні, то виконується перевірка наявності токена доступу в заголовку запиту, по ключу - «Access-Token». У разі, коли його там не виявлено, виникає помилка, то і маршрут запиту на цьому завершується – користувач бачить помилку авторизації (401 – Unauthorized). У разі, коли не підтверджується валідність токена, повертається така ж помилка. Підтвердження відбувається шляхом звернення до AuthService, звідки приходить або відповідь у вигляді токена, або помилка, що сигналізує про те, що токен хибний.
3) якщо всі перевірки пройдені, також йде виклик базового методу, що пропускає запит до контролера. [19]

2.5 Тестування. Аналіз результатів

Як клієнт для тестування програми використовується Postman - зручний http-клієнт для тестування веб-сайтів. Як будь-який клієнт, здійснює зв'язок з бекенд за принципом спілкування:

– postman: «Дай мені таку інформацію через те що…»;
– backend: «Добре, запит правильний, отримуй інформацію».

Для перевірки зв'язку між Gateway і AuthService виконаємо процес реєстрації користувача:

1) Складемо запит на реєстрацію в Postman (см. рис. 2.3):

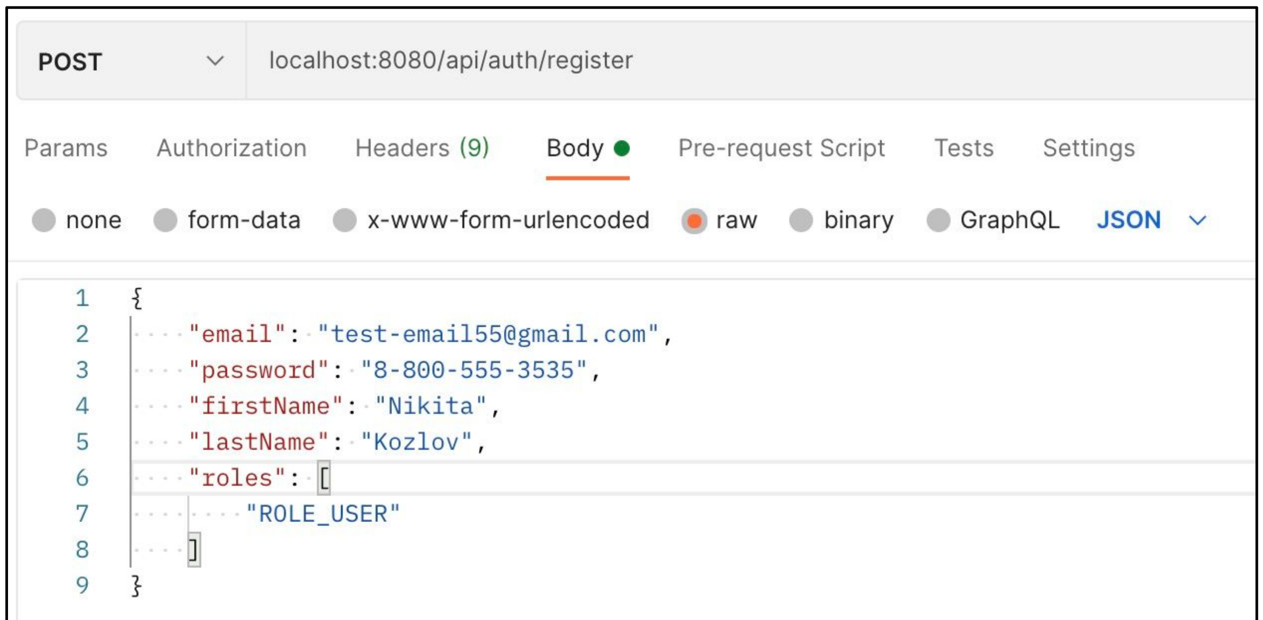


Рисунок 2.3 – Побудова запиту на Gateway

2) Перевіримо факт допуску запиту до його контролеру і факт отримання даних з запиту (см. рис. 2.4):

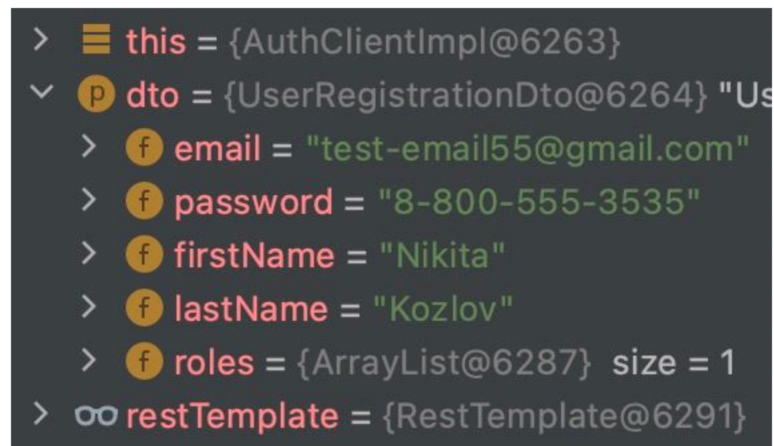


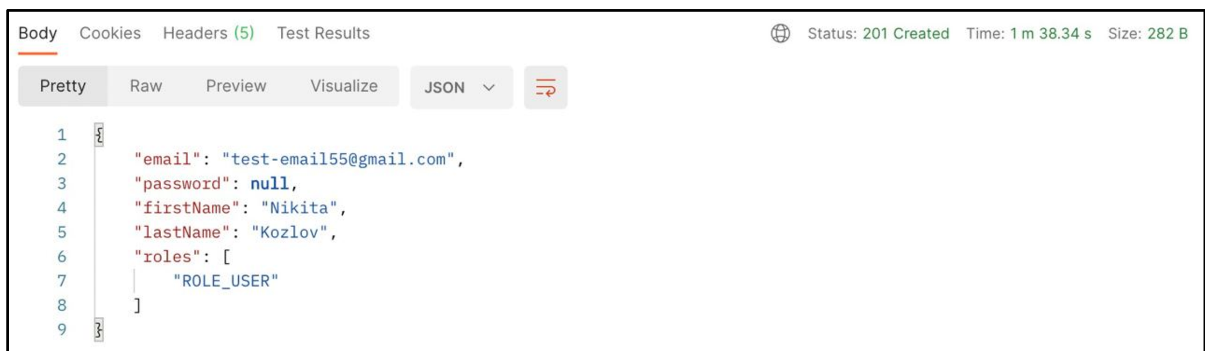
Рисунок 2.4 – Дані дійшли до обробника в «/client» на Gateway

3) Виконавши всі кроки обробки запиту, перевіримо дані, які прийшли на AuthService (см. рис. 2.5):

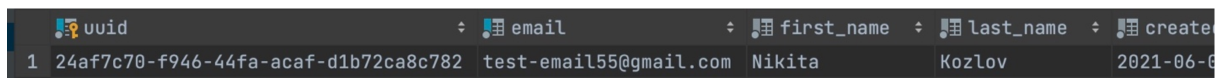
```
> this = {RegistrationServiceImpl@10619}
v dto = {UserRegistrationDto@10620} "Use
  > email = "test-email55@gmail.com"
  > password = "8-800-555-3535"
  > firstName = "Nikita"
  > lastName = "Kozlov"
  > roles = {ArrayList@12688}  size = 1
```

Рисунок 2.5 – Дані дійшли до сервісу в «/service» на AuthService

4) Припустимо всі кроки «дебагу» і бачимо відповідь про успішну реєстрацію в Postman і відповідний запис у базі даних (см. рис. 2.6 (а, б)):



(а)



	uuid	email	first_name	last_name	create
1	24af7c70-f946-44fa-acaf-d1b72ca8c782	test-email55@gmail.com	Nikita	Kozlov	2021-06-(

(б)

Рисунок 2.6 – Користувач збережений

Далі перевіримо механізм обробки помилок. Змоделюємо ситуацію, коли при реєстрації використовується вже існуючий email. Виконавши такий же запит, отримаємо повідомлення про помилку (см. рис. 2.7).

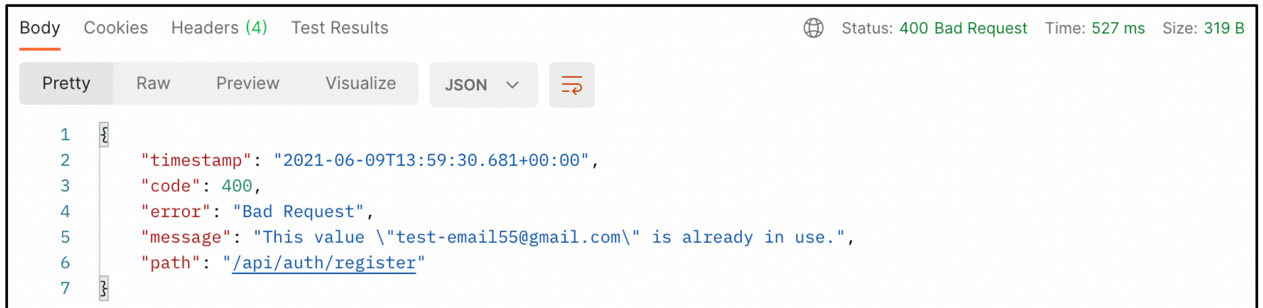


Рисунок 2.7 – Вивід помилки, що виникла на AuthService

Далі розглянемо зв'язок саме між сервісами, на прикладі спілкування між Сервісом аналітики (AnalyticService) і Сервісом з управління контентом (ContentService). Даний зв'язок необхіден першому для перевірки тесту, пройденого користувачем, оскільки дані про тест, а саме правильні відповіді на кожне питання зберігаються на другому. Виконаємо запит на отримання тесту за ідентифікатором «1» (см. рис. 2.8):

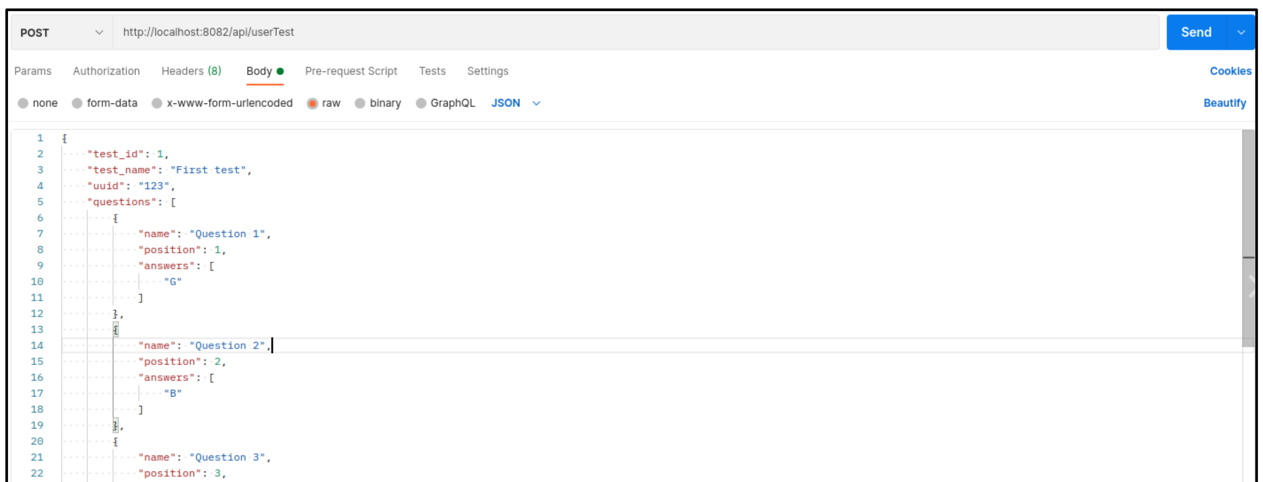


Рисунок 2.8 – Результат виконання запиту у вигляді даних про тест

ВИСНОВКИ

В ході виконання дипломної роботи розглянуті наступні підходи до викладання математики людям, чия сфера діяльності не передбачає використання знань даної науки в чистому вигляді:

– викладання математики в чистому вигляді;
– використання прикладів з багатьох сфер життя, для роз'яснення абстрактних математичних понять, таких як: похідна, межа функції і інші.

В результаті, вибір беззастережно упав на другий підхід, оскільки специфіка платформи саме в тому, що математичні курси орієнтовані на «НЕ математиків».

Проаналізовані існуючі архітектурні рішення як для освітніх платформ, так і для веб-додатків в цілому. Основними архітектурними рішеннями є:

1) додаток-моноліт;
2) сервіс-орієнтований додаток.

З причини безлічі плюсів сервіс-орієнтованого рішення, як архітектуру для системи обрано його, а саме – «під-архітектуру», Мікросервісну архітектуру. Переваги такого рішення очевидні:

– розробка розподілених систем нехай і є більш складною, в порівнянні з монолітом, проте процеси підтримки і масштабування таких систем набагато простіше;
– паралельна розробка. Надає можливість паралельної розробки окремих модулів, що істотно підвищує швидкість самої розробки.

Створені такі компоненти програми, де кожна компонента (мікросервіс) відповідає тільки за свою логіку і працює зі своїм локальним контекстом:

– сервіс забезпечення безпеки. Забезпечує авторизацію користувачеві за подобою механізму OAuth2;

- сервіс по управлінню контентом. В його обов'язки входить управління таким базовими операціями як «Create, Read, Update, Delete» по відношенню до всіх об'єктів, що використовуються для надання навчальної інформації студенту.
- сполучний модуль. Використовується для забезпечення зв'язку всіх сервісів з клієнтом за допомогою http-протоколу. Є транслятором відповіді сервісу, будь то відповідь про успішно виконане запиті або відповідь про помилку, на клієнтське дію.

Подальша розробки даної освітньої платформи може бути описана наступним чином:

- впровадити механізм забезпечення цілісності даних, наприклад, з використанням Kafka;
- зробити запити до Gateway асинхронними, в залежності від виконуваної логіки. Так, у разі, коли передбачаються тривалі обчислення перед відправкою відповіді, слід робити запит асинхронним або відкладеним;
- організувати більшу кількість зв'язків між сервісами без участі Gateway для виконання внутрішнього обміну даних.

Впровадження всіх вищеописаних пунктів збільшить швидкість роботи системи, зробить сервіси більш незалежними, дасть більше гарантій щодо цілісності даних при переході від одного сервісу до іншого, і в результаті, зробить розроблювану платформу більш конкурентно-спроможною на ринку.

Матеріали роботи доповідались на вісімнадцятій всеукраїнській конференції студентів і молодих науковців «Інформатика, інформаційні системи та технології» (Одеса, 23 квітня 2021 р.) та 77-ї звітній студентській науковій конференції Одеського національного університету імені І.І. Мечникова (26 – 27 квітня 2021 року).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Принципы SOLID. [Електронний ресурс] – Режим доступу: https://medium.com/webbdev/solid-4ffc018077da;
2. SOLID – принципы объектно-ориентированного программирования. [Електронний ресурс] – Режим доступу: https://web-creator.ru/articles/solid;
3. 2020 edX Impact Report. [Електронне видання] – Режим доступу: https://www.edx.org/assets/2020-impact-report-en.pdf;
4. Open Edx. [Електронний ресурс] – Режим доступа: https://open.edx.org/;
5. Architecture Principles. [Електронний ресурс] – Режим доступу: https://openedx.atlassian.net/wiki/spaces/AC/pages/1066500378/Architecture%2BPrinciples%2BWIP;
6. Open Edx – Документация. [Електронний ресурс] – Режим доступу: https://open-edx-proposals.readthedocs.io/en/latest/;
7. Stepik. [Електронний ресурс] – Режим доступу: https://welcome.stepik.org/ru;
8. Монолитная архитектура. [Електронний ресурс] – Режим доступу: https://ichi.pro/ru/monolitnaa-arhitektura-cto-zacem-i-kogda-167597237980675;
9. Монолитные приложения. [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/ru-ru/dotnet/architecture/containerized-lifecycle/design-develop-containerized-apps/monolithic-applications;
10. Почему монолитная архитектура – почти всегда плохо. [Електронний ресурс] – Режим доступу: https://m-i-kuznetsov.livejournal.com/187050.html;
11. SOA (Service Oriented Architecture). [Електронний ресурс] – Режим доступу: https://piter-soft.ru/knowledge/glossary/process/soa-system.html;

12. Что такое SOA. [Електронний ресурс] – Режим доступу: https://www.livebusiness.ru/tags/soa/;
13. Микросервисная архитектура и когда ее применять. [Електронний ресурс] – Режим доступу: https://proglib.io/p/microservices;
14. Микросервисная архитектура: характерные особенности. [Електронний ресурс] – Режим доступу: https://www.tadviser.ru/index.php;
15. Api Gateway. [Електронний ресурс] – Режим доступу: https://microservices.io/patterns/apigateway.html;
16. Spring Boot Reference Documentation. [Електронний ресурс] – Режим доступу: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#getting-started.system-requirements.servlet-containers;
17. Spring Framework Documentation. [Електронний ресурс] – Режим доступу: https://docs.spring.io/spring-framework/docs/current/reference/html/;
18. Spring Data JPA. [Електронний ресурс] – Режим доступу: https://spring.io/projects/spring-data-jpa;
19. Обзор протокола HTTP. [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/ru/docs/Web/HTTP/Overview.