НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

"На правах рукопису"
УДК____004.4____

«До захисту допущено»
В.о. зав.кафедри
_____ Олександр КОВАЛЬ
“___”_____202_ р.

Магістерська дисертація

За освітньою програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»
Спеціальності 121 Інженерія програмного забезпечення
на тему: _Моделі та методи розробки Back End для задач створення___
_інформаційного порталу кафедри_____

Виконав (-ла): студент (-ка) _2_ курсу, групи _ТВ-з11мп_
_____Мішин Антон Анатолійович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник
_____професор д.т.н., доц. Недашківський О. Л._____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент
_____професор д.т.н., доц. Шушура О. М._____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022

Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти другий, магістерський
За освітньою програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»
Спеціальності 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
в.о. зав. кафедри
_____ Олександр КОВАЛЬ
(підпис)

«____» _____ 202_р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ

_____Мішин Антон Анатолійович_____
(прізвище, ім'я, по батькові)

1. Тема дисертації: _Моделі та методи розробки Back End для задач створення___ інформаційного_порталу кафедри_____
Науковий керівник _професор Недашківський О. Л._____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від "25"листопада 2022 року №4316-с
2. Строк подання студентом дисертації 07 грудня 2022 року
3. Вихідні дані до роботи _мова Java, операційна система UNIX, _фреймворк_____ _Spring Boot, середовище розробки Intellij_____
4. Перелік питань, які потрібно розробити _ознайомитися з організаційною_____ _структурою закладів освіти, розглянувши їх функції, завдання та основні види__ _діяльності; виконати аналіз автоматизації бізнес-процесів і документообігу у___ _вищій школі; дослідити передовий досвід організації та узагальнення даних. Це_ _допоможе вам розробити моделі та методи ефективної організації зберігання___ _даних; використати результати для подальшого аналізу даних математичними__ _способами_____
5. Орієнтований перелік ілюстративного матеріалу _____
6. Орієнтований перелік публікацій _____
7. Дата видачі завдання «01» жовтня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів виконання магістерської дисертації	Строки виконання етапів магістерської дисертації	Примітка
	Отримання завдання	01.10.2021	виконано
	Дослідження предметної області	01.10.2021 - 01.11.2021	виконано
	Постановка вимог до проєктування системи	01.11.2021 - 15.11.2021	виконано
	Дослідження існуючих рішень	15.11.2021 - 01.12.2021	виконано
	Підготовка публікацій	01.12.2021 - 09.12.2021	виконано
	Розробка програмного продукту	05.03.2022 - 07.10.2022	виконано
	Тестування	07.10.2022 - 11.11.2022	виконано
	Захист програмного продукту	17.10-2022 – 21.10.2022	виконано
	Підготовка магістерської дисертації	22.10.2022 – 20.11.2022	виконано
	Передзахист	21.11.2022 - 25.11.2022	виконано
	Захист	19.12.2022 – 23.12.2022	виконано

Студент _____ Мішин А. А.
(підпис) (прізвище та ініціали)

Науковий керівник _____ Недашківський О. Л.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Актуальність теми: Майже будь яке значне за розміром підприємство потребує ряду бізнес процесів для повноцінного функціонування. Бізнес процеси мають ряд етапів із певною послідовністю дій, залучають ряд осіб та потребують залучення документообігу. Додатковою потребою є система інформування та сповіщення співробітників, клієнтів та інших осіб які можуть бути залучені до функціонування підприємства. Слід зазначити що кафедра або факультет університету також має вищезазначені вимоги.

Раніше, за відсутності розповсюдженого використання персональних комп'ютерів, усі процеси та документообіг потребували безпосередньо фізичної присутності осіб та фізичних підписів якщо мова йде про складову документообігу. На разі є масова тенденція до дігіталізації та автоматизації усіх процесів підприємства та використання цифрових підписів. Також дистанційний режим навчання став поширеним а іноді навіть необхідним, що створює нові виклики організації роботи навчальних закладів та їх складових.

Метою роботи є розробка backend системи яка реалізує функціонал для автоматизації роботи кафедри, матиме гнучку архітектуру для подальшого розширення або інтеграції зі сторонніми сервісами а також потенціал для масштабування ресурсів у залежності від навантаження.

Функціонал повинен надавати можливість ідентифікувати користувача у системі, його права та доступи. Необхідно розробити модель яка зможе пристосовуватися до нових вимог у бізнес процесах та документації без необхідності вносити зміни у код системи. Налаштування нових процесів повинно бути гнучким та не повинно потребувати від адміністратора системи спеціальних технічних навичок. Система повинна передбачати можливість інформування та сповіщення користувачів за її межами – одним із можливих рішень може бути автоматичне листування із використанням електронної пошти користувача.

Для досягнення мети необхідно вирішити наступні завдання:

1. Ознайомитися зі структурою кафедри і основними ролями які може займати особа у цьому контексті;
2. Дослідити процеси на кафедрі їх етапи та життєвий цикл;
3. Дослідити інформацію стосовно підходів до автоматизації бізнес процесів;
4. Провести аналіз та порівняти існуючих рішень та основних практик автоматизації процесів;
5. Розробити архітектуру системи та впровадити програмне рішення виходячи із результатів попередніх кроків.

Об’єктом дослідження є інформаційні системи для організації бізнес процесів та документообігу для освітніх закладів.

Предметом дослідження є моделі та підходи до організації бізнес процесів у домені освіти**.**

Методи дослідження. Метод об'єктно-орієнтованого аналізу для виділення сутностей предметної областей, метод аналізу структури та функціоналу існуючих аналогів інформаційних систем, метод проектування архітектури інформаційної системи.

Практичне значення одержаних результатів. Інформаційна система із гнучкою розподіленою архітектурою та можливістю пристосовуватися до нових вимог без потреби змінювати код.

Структура та обсяг роботи. Робота містить 95 сторінок, 27 рисунків, 22 таблиці та 18 посилань.

Ключові слова: бізнес процеси, low code/no code, backend, Java, Spring Boot, Microservices, Distributed Systems.

ABSTRACT

Relevance of the topic: Almost any large enterprise needs a number of business processes in order to function. Business processes have a number of stages each having certain sequence of actions, involve a number of people, also rely document management. An additional need is a system of informing and notifying employees, customers and other people who may be part of enterprise functioning. It should be noted that the cathedra or faculty of a university also possess above mentioned requirements.

Earlier, in the absence of widespread use of personal computers, all processes and document circulation required a direct physical presence of people and physical signatures if it is a document. Currently, there is a massive trend towards digitization and automation of all enterprise processes and the use of digital signatures. Also, the remote learning mode has become widespread and sometimes even a necessity, which creates new challenges for work organization of educational institutions and their components.

The purpose is the development of a backend system that implements functionality for automating the work of the cathedra, will have a flexible architecture for further expansion or integration with third-party services, as well as the potential for scaling resources depending on the load.

The functionality should provide an opportunity to identify the user in the system, his rights and accesses. It is necessary to develop a model that can adapt to new requirements in business processes and documentation without the need to make changes to the system code. Setting up new processes should be flexible and should not require special technical skills from the system administrator. The system should provide capabilities to informing and notifying users outside of it - one possible solution could be automatic correspondence using the user's e-mail.

To achieve the goal, it is necessary to solve the following tasks:

1. Familiarize with the structure of the department and the main roles that a person can play in its context;
2. Study the processes at the department, their stages and life cycle;

3. Research information on business processes automation approaches;
4. Analyze and compare existing solutions and basic practices of process automation;
5. Layout the system architecture and implement the software solution based on the results of the previous steps.

Object of research is information systems for organizing business processes and document management for educational institutions.

Subject of study are method of object-oriented analysis to extract entities of subject areas, the method of analyzing the structure and functionality of existing analogues of information systems, the method of designing the architecture of the information system.

Research methods object-oriented analysis to identify the essence of the subject area, methods of structural and functional analysis of similar existing software systems, forecasting methods for choosing development tools to create an information system that should be easily extended by developers with different skill levels.

The practical significance of the obtained results. An information system with a flexible distributed architecture and the ability to adapt to new requirements without the need to change the code.

Master's degree thesis. The work contains 95 pages, 27 pictures, 22 tables and 18 links.

Keywords: business processes, low code/no code, backend, Java, Spring Boot, Microservices, Distributed Systems.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JSON – JavaScript Object Notation

API – Application programming interface

IDE – Integrated Development Environment

SQL – Structured Query Language

REST – Representational state transfer

HTTP – Hypertext Transfer Protocol

HTML – Hypertext Markup Language

CRUD – Create, Read, Update, Delete

URL – Uniform Resource Locator

UI – User Interface

XML – Extensible Markup Language

JWT – JSON Web Token

CRM – Customer Relation Management

ERP – Enterprise Resource Planning

JVM – Java Virtual Machine

RPC – Remote Pocedure Call

ВСТУП

Освіта – дуже важливий фактор у розвитку суспільства, що визначає економічний розвиток та рівень життя у цілому. Із плином часу перед людством постають нові задачі та проблеми проте питання передачі знань зберігається не залежно від рівня розвитку та часового проміжку. Через свою значущість питання освіти врегульоване на державному рівні а також є ряд закладів, що надають освітні послуги на приватній основі.

Із плином часу організація освітнього процесу ускладнюється, через наступні фактори: людство накопичує нові знання та переглядає існуючі, доступність освіти зростає а разом із тим кількість студентів які її здобувають, нові виклики потребують нових форматів навчання – одним із прикладів є віддалена форма навчання. Глобалізації та технології для швидкого обміну інформацією позначаються зростання темпів оновлення існуючих знань.

Поточна організаційна структура освітніх закладів і їх внутрішні процеси добре пристосовані до вирішення поставлених задач, проте наявність нового інструментарію для часткової автоматизації надасть можливість швидше та більш гнучко пристосовуватися до змін, витрачаючи значно меншу кількість ресурсів.

Використання цифрових технологій для автоматизації процесів у освітніх закладах є доволі актуальною потребою та може зняти ряд питань що повстали із переходом на віддалену форму навчання а також покращити інформованість студентів заочної форми.

Процес документообігу може бути покращено – цифрові варіанти документів додають прозорості і не потребують фізичних підписів. Як результат час витрачений на організаційні моменти буде значно зменшено, усі учасники процесу будуть мати необхідну інформацію яку вони потребують для виконання своїх зобов’язань.

Ця робота має на меті дослідити підходи та процеси, що використовуються для організації роботи у домені освіти, порівняти існуючі рішення і системи та запропонувати модель яка усуне недоліки існуючих систем.

На основі попередніх досліджень слід створити інформаційну систему яка буде відповідати наступним вимогам, але не обмежуватися ними:

1. Збереження даних про учасників процесу таких як викладачі, студенти, адміністратори, тощо;
2. Збереження даних про організаційну структуру закладу та ролі користувачів у цій структурі;
3. Автоматизації взаємодії між учасниками процесів;
4. Використання електронної документації та електронних підписів;
5. Сповіщення користувачів, що знаходяться поза системою;
6. Налаштування нових процесів без необхідності робити зміни у коді;
7. Користувачі системи не повинні мати спеціальних технічних знань для взаємодії із системою.

Вибір архітектурного підходу та технічних засобів, для впровадження функціоналу системи, також є дуже важливо частиною роботи – правильно обраний інструмент та практика проектування вплинуть на розробку, інтеграцію зі сторонніми сервісами, підтримки системи і успіх продукту у довгостроковій перспективі.

1 ЗАДАЧА РОЗРОБКИ BACKEND СИСТЕМИ ДЛЯ ПОТРЕБ СТВОРЕННЯ ІНФОРМАЦІЙНОГО ПОРТАЛУ КАФЕДРИ

1.1 Постановка задачі розробки backend системи для потреб створення інформаційного порталу кафедри

Сьогодення відзначається швидкістю змін та їх обсягом, інновації відбуваються в усіх сферах життя та науки. Інформаційна база постійно оновлюється – додаються нові відкриття, знання що раніше були актуальними застарівають або починають вважатися хибними, народжуються нові дисципліни а деякі перестають існувати.

Кожен навчальний заклад має свою структуру та підходи до управління внутрішніми процесами, це визначається: напрямом у якому спеціалізується заклад, організаційною структурою та викладацьким складом, можливою кількістю студентів тощо. Ситуація також змінюється від факультету до факультету, залежно від предмету та викладача.

Освітня програма корегується на рівні міністерства освіти, тому кожен заклад який хоче отримати рівень державної акредитації повинен виконувати певні вимоги та відповідати нормам. Зміни щодо порядку акредитації та вимог також відбуваються регулярно.

Із переходом на віддалену форму навчання виникли нові виклики, також важливим є необхідність організовувати роботу студентів які навчаються на заочній формі навчання, що є здебільшого асинхронною.

Поєднання вищезазначених факторів може принести свою частку хаосу у адміністративні процеси а також у організацію навчального процесу безпосередньо. Особливо якщо у цих умовах потреба у змінах та адаптації виникає неочікувано.

Слід не забувати, що як і будь яке підприємство, керування навчальним закладом це оркестрація діяльності багатьох людей. Взаємодія між особами

відбувається у декілька етапів, кожен із яких може потребувати певних дій або документацію, і слід розуміти які етапи вже було виконано та чи проінформовані усі учасники процесу і хто є відповідальним за успішність завершення того чи іншого процесу чи етапу. Це все потребує значних зусиль зі сторони адміністрації у плані інформування викладачів і студентів, оформленню та зберіганню документації та надання відповідної звітності. У свою чергу викладачі та студенти, окрім свої прямих обов'язків також повинні витрачати ресурс, також повинні узгоджувати свою діяльність між собою та адміністрацією закладу.

Таким чином створення інформаційного порталу та часткова автоматизація процесів функціонування тієї чи іншої кафедри є доволі актуально потребою. Впровадження системи вирішить ряд проблем: покращить процес взаємодіє адміністрації, викладачів та студентів, заощадить велику кількість часу та зусиль, покращить інформованість учасників процесу а також спростить робо із документацією за рахунок її дігіталізації.

Для проектування та впровадження системи слід вирішити наступні часткові наукові завдання:

1. Проаналізувати існуючі інформаційних систем, які використовуються для організації процесів у навчальних закладах;
2. Проаналізувати методи та практики, що забезпечують функціонування кафедри та навчального закладу;
3. Розробити модель автоматизації та інформування для кафедри.

Для розробки системи слід обрати архітектурний стиль який буде робити систему гнучкою, здібною до масштабування, відмовостійкою, відкритою до інтеграції зі сторонніми сервісами – як у якості клієнта так і безпосередньо постачальника послуг. Проектування системи повинно передбачувати можливість використання різних технологій та мов програмування для розробки – таким чином можливо задіяти більш широкий спектр спеціалістів а також використати сильні сторони та можливості окремих мов програмування, бібліотек та технологій.

Розробка основної частини системи повинна виконуватися із використанням технологій та фреймворків, які можуть забезпечити стабільність системи, підтримку інтеграції із основними технологіями. Інструментарій повинен підтримуватися розробником і мати широку користувацьку базу, це гарантія правильного функціонування програмного продукту і доступність інформації для вирішення складнощів, які можуть виникнути під час розробки та підтримки продукту.

Як результат слід розробити систему у відповідності до вимог:

Функціональні вимоги до системи:

1. Збереження даних про учасників процесу таких як викладачі, студенти, адміністратори, тощо;
2. Збереження даних про організаційну структуру закладу та ролі користувачів у цій структурі;
3. Автоматизації взаємодії між учасниками процесів;
4. Використання електронної документації та електронних підписів;
5. Сповіщення користувачів, що знаходяться поза системою;
6. Налаштування нових процесів без необхідності робити зміни у коді;
7. Користувачі системи не повинні мати спеціальних технічних знань для взаємодії із системою.

Архітектурні вимоги до системи:

1. Гнучка архітектура спроможна до горизонтального масштабування;
2. API який надасть можливість для інтеграції зі сторонніми сервісами та користувацьким інтерфейсом;
3. Можливість доповнювати системи із використанням будь-яких технологій та мов програмування.

1.2 Аналіз існуючих рішень для автоматизації процесу функціонування кафедри

Складність керування підприємством, включаючи заклади освіти, була актуальною проблемою протягом довгого часу. Як відповідь було запроваджено ряд систем для автоматизації процесів та інформування учасників. Використання цих систем має позитивний вплив на роботу установ яка виражається також і у матеріально-економічному аспекті. Ці системи скорочують витрати часу на певні бізнес-процеси та оптимізують повсякденну роботу адміністрації, викладачів і студентів, тому без таких систем не може повноцінно функціонувати кожен навчальний заклад.

Перехід на цифрове курування підприємством дуже об’ємна задача, і в даний час для вирішення цієї проблеми використовуються ряд систем які можна умовно поділити на два підтипи: CRM та ERP.

CRM – це технологія, яка керує всіма відносинами та взаємодією компанії з клієнтами та потенційними клієнтами, частково інтегруючи у цей контекст керування внутрішніми процесами. Мета проста: покращити ділові відносини для розвитку бізнесу. Системи CRM допомагають компаніям підтримувати зв’язок із клієнтами, оптимізувати процеси та підвищити прибутковість.

ERP – програмне забезпечення, яке організації використовують для керування повсякденною бізнес-діяльністю, наприклад бухгалтерським обліком, закупівлями, управлінням проектами, управлінням ризиками та відповідністю, а також операціями ланцюга поставок. Повний пакет ERP також включає програмне забезпечення для управління продуктивністю підприємства, яке допомагає планувати, складати бюджет, прогнозувати та звітувати про фінансові результати організації.

Обидва моделі мають попит, проте не вирішують задач специфічних для домену освіти. CRM пріоритизує взаємодію компанія-клієнт. ERP системи у свою

чергу роблять значний акцент на фінансовій частині та документообігу, не враховуючи бізнес логіку процесу.

Оптимальною моделлю є рішення low-code/no-code – передбачає впровадження платформи для розробки програмного забезпечення, для використання розробниками і користувачам без спеціальних технічних навичок. Це нова модель впровадження систем, яка раніше не використовувалася при автоматизації роботи навчальних закладів і може надати можливість швидко пристосовуватися до нових вимог, створюючи або змінюючи автоматизовані процеси керування, без залучення технічних спеціалістів.

Однією системою, яка покликана автоматизувати роботу кафедри та надати функціонал для інформування є “Campus” – запроваджений у КПІ, домашню сторінку зображено на рисунку 1.1.

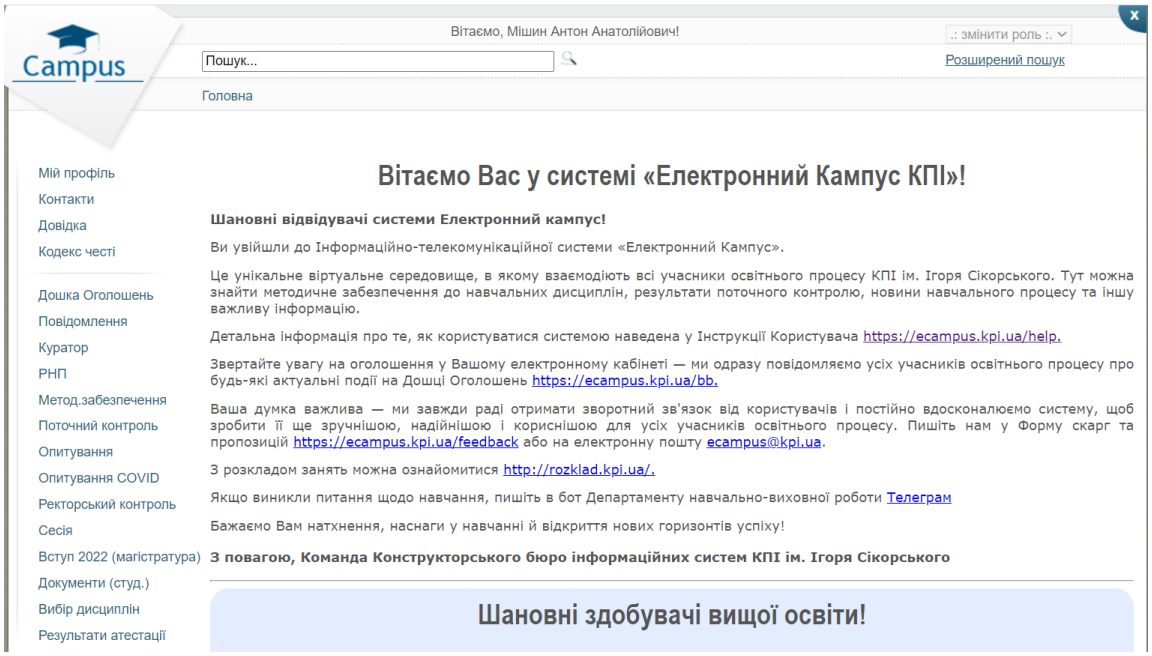

Рисунок 1.1 – Домашня сторінка системи “Campus”

Система надає необхідний базовий функціонал для адміністрації, викладачів та студентів. Має функціонал для авторизації користувача та зберігає його контактну інформацію. У системі існують окремі розділи які надають інформацію та послуги у відповідності – перегляд оголошень, визначеного куратора групи,

перегляд плану навчальної програми, результати сесії та ректорського контролю. Окремо слід відзначити можливість обміну файлами, що спрощує деякі процеси, наприклад подача документів на вступ до магістратури.

Проте основним фокусом системи є інформування студентів за допомогою відображення статичної інформації тож із недоліків функціоналу слід відзначити:

1. Відсутня автоматична комунікація поза межами системи;
2. Налаштування процесів здійснюється у ручному форматі та інколи потребує зміни у коді системи;
3. Відсутня можливість автоматизувати керування окремими предметами для конкретних груп чи студентів за потребами конкретного викладача.

Іншою системою із аналогічним функціоналом є веб-система НТУ «ХПІ». Частину інтерфейсу системи можна побачити на рисунку 1.2.

Рисунок 1.2 – Інформаційна система НТУ «ХПІ». Розділ “Навчальна частина”

Система більше зосереджена на адміністративній частині процесу і надає такий функціонал: введення та редагування особистих даних абітурієнтів, формування наказів на зарахування, введення та редагування особистих даних студента, робота із наказами щодо існуючих студентів, робота із навчальними

планами, формування розкладу, введення та редагування успішності студента, відображення різноманітної статистики.

Включно із недоліками, що було приведено під час огляду системи “Campus”, для НТУ «ХПІ» існують наступні:

1. Можливість комунікації відсутня навіть у рамках системи;
2. Відсутня можливість обміну документами;
3. Доволі складний функціонал, що дублюється між розділами.

Висновки до розділу 1

Аналіз існуючих методів показав, що існуючі методи – CRM та ERP, не можуть покрити потреби специфічні для домену освіти, хоч і мають потужний та устаткований функціонал.

Оптимальним рішенням задачі автоматизації є підхід low code/no code, який ставить на меті надання інструментів для автоматизації нових бізнес процесів та зміни у існуючи, без потреби змінювати код системи або залучати технічних спеціалістів. Такий підхід є особливо корисним для адаптації до умов, що динамічно змінюються.

Серед існуючих систем, що вирішують проблему автоматизації учбового закладі, є недоліки які можна подолати ввівши новий підхід при побудові інформаційної системи. Окрім існуючих плюсів оглянутих систем ключовими вдосконаленням при імплементації нової системи повинні бути:

1. Автоматизація комунікація поза межами системи;
2. Гнучке налаштування бізнес процесів;
3. Функціонал керування окремими предметами для конкретних груп чи студентів за потребами конкретного викладача.

2 АПАРАТИ ВИРІШЕННЯ ЗАДАЧІ ПОБУДОВИ BACK END ДЛЯ ЗАДАЧ СТВОРЕННЯ ІНФОРМАЦІЙНОГО ПОРТАЛУ КАФЕДРИ

2.1 Архітектурний підхід під час побудови системи

Архітектура програмного забезпечення — це метод проектування системи, що виконує певний набір завдань і задовольняє ряд характеристик. Архітектурні рішення вибираються на основі вимог до системи, відповідно до цілей, визначених на етапі проектування. Критеріїв, за якими оцінюється система:

– Гнучкість;

– Можливість масштабування;

– Відмовостійкість;

– Доступність;

– Узгодженість;

– Можливість перевикористання елементів;

– Швидкість відповіді.

На практиці реалізація програми, яка відповідає всім вищевказаним критеріям, є неможливою та надлишковою. Ось чому дуже важливо завжди мати фазу планування перед початком розробки, тому що в майбутньому внесення змін вимагатиме багато ресурсів, а іноді взагалі буде неможливим, що може призвести до необхідності повторного створення програми.

Архітектурне проектування — це процес опису системи на достатньо високому рівні абстракції для опису очікуваної поведінки системи та її компонентів. На цьому етапі вирішується, які архітектурні рішення будуть використані та які елементи системи будуть задіяні, визначаються можливі обмеження системи, і таким чином отримується набір компонентів і рішень, що потім перетворюється на основу для документації по якій буде відбуватися розробка. Це дуже важливий етап, який впливає на довгостроковий успіх проекту,

комфорт та досвід кінцевого користувача, ресурси для підтримки і розширення системи.

2.1.1 Монолітна архітектура

Монолітна архітектура – випадок коли додаток є цільною системою яка збирається із єдиної кодової базу та розгортається цілком, хоч і поділяється на декілька умовних компонентів як показано на рисунку 2.1.

Рисунок 2.1 – Модель монолітної архітектури

Даний підхід пришвидшує розробку на початковому етапі перш за все завдяки тому, що елементи системи інтегруються за допомогою фреймворку розробки, розгортаються як одне ціле та виконуються на єдиному фізичному сервері. Проте із плином часу швидкість розширення функціоналу значно знижується – для успішного впровадження змін розробник має добре розуміти логіку роботи системи та орієнтуватися у вихідному коді. Також важко залучати великі команди – через високу зв’язність компонентів зміни у коді приводять до порушень існуючого функціоналу та конфлікту версійності змін.

Переваги монолітного підходу:

– Часткове спрощення розробки - не потрібно планувати та розробляти механізми взаємодії між компонентами, можна зосередитися лише на функціоналі;

– Наявність інструментів та практик для вирішення типових задач, за рахунок довгої історії використання підходу;

– Простота розгортання – найчастіше не потребує використання великої кількості сторонніх сервісів та інтегрується лише з поширеними, використовуючи стандарти;

– Відповідь на запити отримується набагато швидше – зумовлено гомогенністю системи, яка не має компонентів розподілених у мережі, що потенційно можуть знаходитися на різних фізичних серверах.

Мінуси монолітної архітектури:

– Складність підтримки ізольованості модулів системи, із часом можливі компроміси які призведуть до зв’язування логіки із різних шарів чи модулів системи;

– Підтримка такого проекту також не є складною – оскільки може бути важко визначити місце знаходження необхідної логіки для виправлення помилок у великій кодовій базі;

Монолітна архітектура все рідше використовується у сьогоденні та вважається відносно застарілою. Деякі системи ще зберігають саме такий підхід через значну кількість ресурсів для внесення змін та можливі ризики.

Ще одним варіантом використання моноліту є перехідний період коли система лише формується і не має чітких компонентів які могли би стати самостійними, проте при першій нагоді систему намагаються мутувати.

Навіть системи із устаткованою монолітною архітектурою починають побудову інфраструктури довкола основного додатку, якщо це можливо під час розширення.

2.1.2 Serverless архітектура

Serverless архітектура – побудована на основі так званої хмарної концепції, не вимагає додаткових зусиль для налаштування середовища та інфраструктури.

Це спосіб автоматизувати налаштування та розгортання додатку без прямого втручання спеціаліста. Назва виникла не тому, що додаток виконувалося без використання сервера, а тому, що при розробці були опускаються всі нюанси, пов'язані з налаштуванням серверної частини. Усі клопоти щодо налаштування доступу до бази даних, керування ресурсами та масштабування делегуються сторонній службі, яка піклується про ці процеси, як показано на рисунку 2.2.

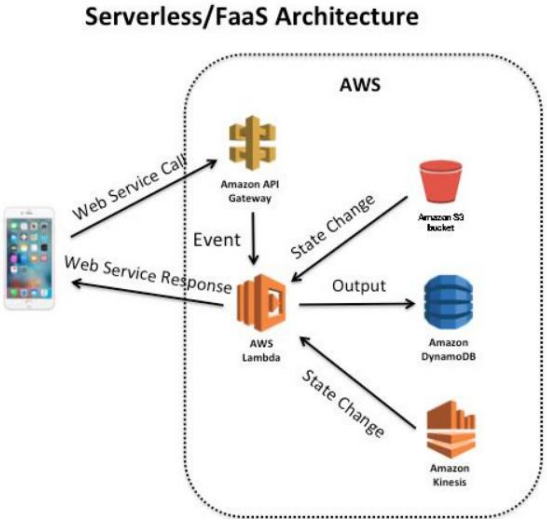

Рисунок 2.2 - Модель Serverless архітектури AWS

Такий підхід дозволяє приділяти усю увагу розробці програми та нюансам, які з нею пов'язані. Масштабування та ресурси є відповідальністю постачальника хмарних технологій, хоча в деяких випадках є можливість змінити налаштування самостійно у відповідності до конкретної потреби.

Переваги Serverless архітектури:

– Практично не потрібні зусилля розгортання програми, лише мінімальна конфігурація – нюанси вирішуються постачальником послуг. Також

найбільш популярні сервіси, такі як наприклад СУБД, надаються у сконфігурованому вигляді самим провайдером хмарних послуг;

– Економія людських ресурсів, оскільки для налаштування інфраструктури не потрібно залучати додаткових спеціалістів;

– Фінансова економія за використання хостингу, оскільки сума вираховується лише на основі фактичного використання ресурсів провайдеру. Іноді для економії можна зменшити кількість копій додатку або встановити ліміт на використання ресурсів;

– Можливість делегувати логіку масштабування на провайдера хмарних послуг.

Недоліки Serverless архітектури:

– Відсутність повного контролю над інфраструктурою, іноді ускладнює оновлення архітектури та перенесення програм у нові середовища;

– Постачальник послуг може накладати свої обмеження на використання ресурсів, час виконання запитів тощо;

– Для оптимізації ресурсів хмарні провайдери можуть зупиняти додатки, у разі відсутньої активності протягом деякого часу. Повторний запит до такого сервісу буде значно довшим через потребу запустити сервіс знову.

Найбільшим плюсом є винесення логіки щодо керування навантаженням та масштабування системи на сторону хмарного провайдера. Також усі фізичні сервери підтримуються та обслуговуються провайдером.

Serverless архітектура частіше використовується при хмарних обчисленнях або для сервісів-функцій, що надають прості послуги та не потребують постійної активності.

Можна використовувати підхід для сформованої системи із чітко виділеними компонентами, проте на початку розробки, коли система активно змінюється і мутує, може створювати більше проблем ніж вирішувати.

2.1.3 Мікросервісна архітектура

Доволі розповсюджений підхід, особливо для проектів, які змушені постійно адаптуватися до змін ринку та інтегруватися зі сторонніми сервісами.

Суть полягає в тому, щоб створити багато сервісів (які можуть бути і постачальником послуг і користувачем одночасно), кожен сервіс буде розгортатися окремо, відповідати за певні функції та взаємодіяти з іншими компонентами сервісу, вирішуючи певні завдання, як показано на рисунку 2.3.

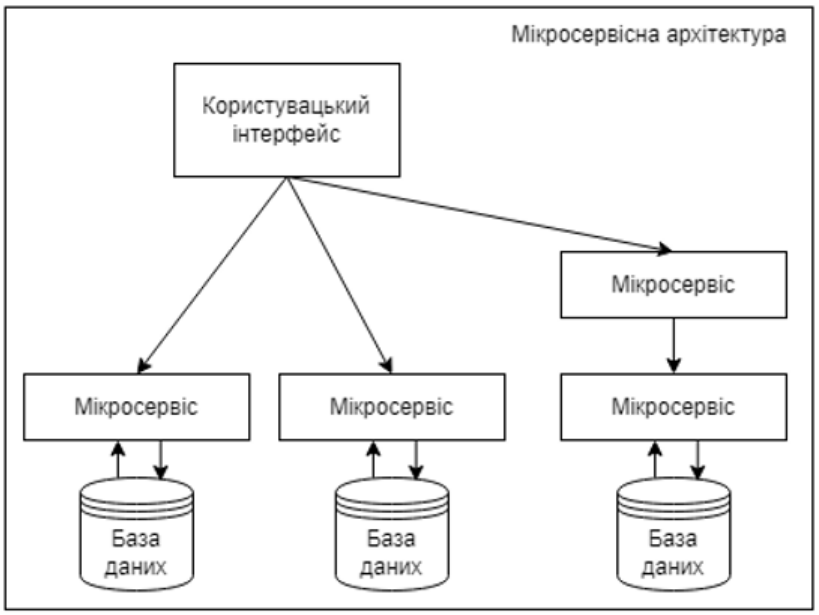

Рисунок 2.3 - Модель мікросервісної архітектури

Кожна служба розгортається у контейнері або віртуальній машині та обслуговується через API. У більшості випадків сервіси спілкуються на основі віддалених викликів або черг повідомлень. Дані передаються та приймаються в універсальному форматі, побудованому таким чином щоб дані можна було використовувати на рівні іншого додатку, не зважаючи на мову його імплементації. Такий підхід дає можливість створювати сервіси з використанням різних мов програмування або фреймворків, після чого взаємодія між ними буде здійснюватися тільки через уніфіковане API, незалежно від інструменту реалізації сервісу.

Переваги мікросервісів:

– Підвищена швидкість впровадження змін – існуючі компоненти чітко розподілені за функціоналом і кожен окремо взятий сервіс має відносно невелику кодову базу;

– Менше часу, на розгортання окремих компонентів – побудова займає менше часу як і розгортання окремого сервісу, при використанні певних технік можна оновлювати компоненти системи без зупинки системи;

– Менше часу, необхідного для перевірки змін – щоб перевірити новий функціонал, сервіс можна запустити локально у ізоляції та зімітувавши сервіси на які спирається нова логіка;

– Спрощує написання юніт тестів та інтеграційних тестів – кожен сервіс має власні специфічні цілі і відносно небагато логіки. Отже тести будуть набагато простіші та вимагатимуть покриття значно меншої кількості комбінацій.

Недоліки мікросервісів:

– Ускладнена структура проекту – планування архітектури мікросервісної системи доволі складний процес, який вимагає навичок і досвіду. Формати зв’язку повинні бути узгоджені, а версії API та контракти мають підтримуватися в будь-який час. Через таку специфіку кількість методів API є дуже високою, якщо порівнювати із монолітною архітектурою;

– Ускладнена безпека – більше API надає більше можливостей для атак, а впровадження заходів безпеки вимагає додаткових зусиль для кожного сервісу;

– Ускладнена E2E тестів – у той час як тестування компонентів системи у ізоляції спрощено, тестуванні системи як одного цілого набагато складніше. Необхідно запускати усі компоненти системи, налаштовувати кожен окремо і після робити перевірку усіх комбінацій. Також виконання таких тестів займає багато часу і не завжди можливо виконувати їх при кожній зміні коду.

Мікросервісна архітектура підходить як для великих проектів, так і для відносно невеликих. Розподілена природа додатків із мікросервісною архітектурою

значно полегшує масштабування елементів системи для компенсації навантаження. Архітектурний підхід дозволяє використовувати різноманітні технології та мови програмування під час розробки, при правильній реалізації значно спрощує інтеграцію зі сторонніми системами. Після побудови основної частини системи підтримка та розробка займають набагато менше часу та ресурсів. Гнучкий підхід при розгортанні систем допомагає досягти постійної доступності системи і впровадити стратегії для відновлення при відмові окремого компоненту.

2.1.4 Вибір архітектурного стилю

При розробці системи слід враховувати довгострокову перспективу та можливість інтеграції зі сторонніми сервісами, як приклад – сервіс цифрових підписів. Систему можуть використовувати багато користувачів одночасно, оскільки навчальний заклад може мати тисячі студентів а також декілька сотень співробітників адміністрації та викладачів, до того ж є періоди підвищеного навантаження: початок навчального року, початок атестації, сесія тощо.

Серед оглянутих архітектурних стилів найбільш вдалим вибором буде саме мікросервісна архітектура – що надасть можливість гнучко масштабувати систему, інтегруватися зі сторонніми сервісами та використовувати різноманітні технології та відповідно ширшого спектру розробників.

Із можливих недоліків слід відзначити підвищену складність розробки на початкових етапах, через налаштування інфраструктури та комунікації між сервісами. Проте у довгостроковій перспективі мікросервісний підхід виграє у плані легкості підтримки та нарощування функціоналу, порівняно із монолітною архітектурою, також можна частково або повністю делегувати налаштування інфраструктури та контроль ресурсів, за допомогою провайдера хмарних послуг, отримавши переваги і мікросервсної і Serverless архітектури.

2.2 Формат API

Оскільки систем матиме мікросервісну архітектуру то вибір формату API має важливе значення – взаємодія буде відбуватися не лише між системою та користувачами а також між компонентами додатку. Слід обрати формати, що буде агностичним до мови програмування, поширеним та простим у роботі – таким чином можна буде додавати нові технології та інтегруватися зі сторонніми системами.

2.2.1 Огляд протоколу SOAP

SOAP — це протокол, що використовує XML-повідомлення для автентифікації, авторизації та виконання віддаленого коду. Він намагається надати інтерфейс для віддаленого виконання методу.

Основна претензія щодо SOAP полягає в тому, що оскільки його комунікаційний рівень покладається на XML, він, як правило, дуже багатослівний. Навіть для виконання простого методу слід відправляти певну напів-порожню структуру і як результат додаткові дані, тому SOAP не найкращий спосіб комунікації у мережі коли мова йде про економію трафіку. Загалом це доволі застарілий підхід, що використовується здебільшого у легасі системах і як результат існує значно менше систем які би змогли легко інтегруватися при використанні такого підходу.

Однак SOAP має одну значну перевагу – забезпечення виявлення як частини протоколу. Для кожного сервісу, що надає API, потрібен файл WSDL з описом усіх методів інтерфейсу. Це також XML-файл, який використовують клієнтські програми, щоб зрозуміти, як виглядає API. Хоча у інших підходах також є свої методи виявлення API, вони не завжди є обов’язковими та можуть бути упущені

розробником. Також XML не залежить від технологій, і надає можливість використовувати різні технології разом.

2.2.2 Огляд GraphQL

У той час як інші підходи більше фокусуються на виокремленні ресурсу GraphQL фокусується на даних – це швидше мова запитів, ніж API, хоча надає усі необхідні інструменти для інтеграції із сервісом.

Його головна перевага полягає в тому, що він дозволяє вибирати та запитувати конкретні дані, які потрібні для виконання операції. Це може бути значним плюсом відносно інших підходів, де для агрегації результату іноді доведеться робити у декілька викликів API а також обробляти надлишкові дані.

Ще одна перевага GraphQL – можливість отримувати оновленні дані в режимі реального часу на основі подій із сервера, зазвичай це робиться через підключення WebSocket. Це допомагає заощадити багато ресурсів у порівнянні із механізмом опитування, підходом коли клієнт постійно надсилає запити до сервісу через деякий проміжок часу. У реактивному підході дані передаються лише при зміні на сервері і запит відправляється саме у той момент коли були зроблені зміни, а не через деякий проміжок часу – що може призвести до затримки у відображенні змін.

Однак, якщо API досить простий або не призначений для обслуговування випадків використання складних даних, використання GraphQL може бути надмірним і для деяких простих випадків читання даних все ж треба буде надсилати значну кількість інформації, під час запиту.

Слід також враховувати що підхід доволі новий і не усі технології маю надійні і перевірені бібліотеки для його імплементації на серверній частині, до того ж імплементація на стороні клієнта як правило буває доволі громіздка.

2.2.3 Огляд RPC

RPC або Remote Procedure Call, це методика взаємодії сервера та клієнта, яка дозволяє клієнту виконувати код, що розташований на сторонньому сервері, так наче код доступний локально. Іншими словами, клієнт абстрагується від того факту, що використовуваний код не є локальним та як передаються дані між клієнтом та сервером. Замість цього ви просто робиться виклик методу як із будь якою локальною залежністю.

Інколи API є дуже орієнтованим саме на певні дії – і намагатися побудувати абстракцію навколо ресурсу або через маніпуляцію даними стає надто складно і реалізація є дуже неочевидною. Натомість у таких випадках слід використати інші підходи такі як RPC – це надасть змогу абстрагуватися від деталей реалізації взаємодії клієнта та сервера, а розробники зможуть мати гарне уявлення про послуги що надає сервіс і зрозумілу абстракцію для їх використання.

Однак для систем які побудовані саме навколо доступу до певних ресурсів такий підхід не є оптимальним – слід впроваджувати механізми розпізнання контракту, така інформація потрібна на етапі компіляції коли використовується типізована мова для розробки. Оскільки інтеграція будується на діях то кількість методів значно зростає і при розширені слід впроваджувати зміни і на стороні клієнта і на стороні сервера. Така сильна зв’язність є значним недоліком і негативно впливає на можливість пере використовувати компоненти системи.

2.2.4 Огляд REST

REST — це абревіатура від Representational State Transfer. Підхід проектування API для розподілених систем. REST має 6 основних принципів, які мають бути виконані, щоб інтерфейс можна було назвати RESTful. На практиці не завжди виконуються усі вимоги, в такому випадку систему вважають RESTlike.

Принципи REST:

– Клієнт-сервер – завдяки відокремленню проблем інтерфейсу користувача від проблем обробки та зберігання даних можна підтримувати клієнтів із різними реалізаціями та легше розширювати систему;

– Відсутність стану сервера – слід уникати зберігання даних на сервері, для побудови відповіді, а саме сесій чи специфічних налаштувань користувача. Усі запити клієнти містити у собі достатньо інформації щоб надати правильну відповідь, у разі використання специфічних налаштувань для користувача клієнт повинен агрегувати ці дані один раз під час авторизації і передавати із кожним запитом. Такий підхід значно зменшує зв'язність сервера та клієнта;

– Кешування – при відповіді сервер вказує явно чи є дані кешованими. Якщо дані кешуються, клієнт залишає за собою право використовувати ці дані повторно без додаткового запиту до серверу, для покращення швидкодії;

– Уніфікований інтерфейс — спрощує загальну архітектуру системи та покращує потенціал до пере використання системи. Щоб отримати універсальний інтерфейс, необхідна велика кількість архітектурних обмежень для керування поведінкою компонентів;

– Багатошаровість системи – стиль у якому кожен окремий компонент системи розділяється на шари. Кожен шар має власні обов'язки та спектр задач, прикладом шарів системи можуть бути: шар обробки запитів, шар бізнес-логіки, шар доступу до даних. Рівні повинні бути відокремлені у реалізації та взаємодіяти лише через чітко описаний інтерфейс. Такий підхід підвищує гнучкість системи та спрощує її структуру;

– Постачальник коду – REST дозволяє розширити клієнтську сторону, завантажуючи скрипти або аплети з боку сервера. Це спрощує розробку клієнта та сприяє масштабованості та зворотній сумісності. Виконання цього обмеження є опційним.

REST виділяє основну абстракцію – ресурс, навколо якої будується логіка API. Ресурсом може бути будь-яка інформація: документи чи зображення, колекції

ресурсів тощо. Уніфікація надає можливості для розширення системи, повторного використання її компонентів і переходу на інші технології. Підхід REST не описує конкретну технологію, а лише набір практик, які можна реалізувати лише частково, щоб послугу можна було назвати подібною до REST, і лише після впровадження усіх RESTful.

2.2.5 Вибір формату API

Побудова такої системи навколо даних, як пропонує підхід GraphQL, є надлишковим і значно ускладнює реалізацію, на стороні сервері та клієнту. Цей підхід може призвести до зростання часу та кількості коду, при інтеграції зі сторонніми технологіями, оскільки підхід із побудовою API не є дуже поширеним та використовується для специфічних задач, частіше у сферах IoT та BigData.

Також система не скерована на велику кількість дій а скоріше на створення та редагування ресурсів для відображення поточного стану процесів у навчальному закладі – таким чином підхід RPC лише ускладнить імплементацію зробивши компоненти більш зв’язними. Також при використанні типізованих мов програмування необхідно буде впровадити інструментарій для розпізнання контракту API та генерації коду під час компіляції на основі цього контракту.

SOAP дуже близький до REST по своїй ідеології. Цей підхід теж є агностичним до мови програмування та може будувати API навколо ресурсів. Проте SOAP, як і RPC, змушує споживача впроваджувати логіку для розпізнання контракту і як результат механізм для генерації коду, при використані типізованих мов програмування. SOAP це застарілий підхід який майже не використовується у сучасних системах тому нові інтеграції потребуватимуть багато змін, з обох сторін комунікації. Також кількість даних, що передаються через мережу при використанні протоколу, значно більша, навіть для простих запитів – у

розподіленій системі, де велика кількість взаємодій відбувається саме через мережу, це значній удар по швидкодії.

Після аналіз підходів до побудови API – найбільш вдалим вибором є використання саме REST підходу. Система буде будуватися у домені освіти, ця галузь має чітко виділені сутності, наприклад: факультет, кафедра, викладач, студент, документ тощо. Оскільки сутності домену можна чітко виділити та структурувати а також при розширені системи можна пере використовувати вже існуючи визначення сутностей просто додавши новий тип, наприклад новий тип документу, то найближчим відображення домену слугує саме ресурс.

2.3 Мова програмування

Вибір мови є важливим етапом планування оскільки визначає парадигму у якій буде вестися розробка, відкриває можливості та накладає ряд обмежень. У випадку автоматизації роботи підприємства інструмент повинен відповідати наступним вимогам:

– Об’єктно орієнтована парадигма – робота із сутностями реального світу найкраще вписується у парадигмі ООП, що дозволяє створювати структуровані програми зі значним потенціалом до пере використання коду;

– Статична типізація – через природу застосунку і необхідність робити виклики до бази даних, зі своїми власними форматами даних, – треба мати чітко визначені типи із перевіркою на етапі компіляції;

– Можливість написання безпечного коду – при написанні масштабних систем дуже важливою частиною є організація ресурсів;

– Мова повинна бути популярно і мати версії із довгостроковою підтримкою, це гарантія уникнення небажаних проблем зі стабільністю а також популярні мови має значний ком’юніті і як результат велику кількість інформації та можливість отримати відповідь на нестандартну проблему із якою стикнувся розробник.

Мовою яка відповідає усім наведеним вище вимогам а також підтримує ряд інфраструктурних фреймворків є Java – об’єктно орієнтована мова програмування. Мова Java пропонує певні ключові характеристики, які роблять її ідеальною для розробки серверних програм, а саме:

– Простота. Java простіше, ніж більшість інших мов для створення серверних додатків через послідовну реалізацію об’єктної моделі. Велика стандартна бібліотека класів надає потужні інструменти для розробників;

– Автоматичне керування ресурсами. JVM автоматично виділення та звільнення пам’яті під час роботи та при завершені програми. Розробник не може явно виділити пам’ять для нових об’єктів або звільнити пам’ять яку використовують існуючі об’єкти. Натомість JVM відповідає за виконання цих операцій;

– Статична типізація. Типізація в Java дає змогу забезпечити безпечне рішення для міжмовних викликів, як наприклад виклик до СКБД;

– Java визначає класи та розміщує їх в ієрархії, яка відображає простір імен домену Інтернету. Можна поширювати свої програми Java і уникати конфліктів імен;

– Підключення до бази даних Java (JDBC) і SQLJ дозволяють коду Java отримувати доступ і маніпулювати даними в реляційних базах даних. Драйвери, надані Oracle, дозволяють портативному, незалежному від постачальника коду Java отримувати доступ до реляційних баз даних.

– Безпечність. Конструкція байт-коду Java та специфікація JVM дозволяють використовувати вбудовані механізми для перевірки безпеки двійкових файлів Java. Oracle Database інсталюється разом із екземпляром Security Manager, який у поєднанні з Oracle Database Security визначає, хто може викликати будь-який метод Java.

– Портативність. Java переноситься на різні платформи без потреби вносити зміни у код. За потреби можна також писати програми залежні від платформи.

2.4 Фреймворк

Основним фреймворком обрано Spring Boot. Вибір пов’язано із рядом переваг, але основними причинами є можливість швидкої розробки мікросервісних додатків а також велика популярність фреймворку та мови Java, на якій він написаний. Як результат наявність бібліотек для вирішення типових задач із добре написаним та відтестованим кодом а також постійна розробка та вдосконалення фреймворку і наявність інформації для вивчення фреймворку чи вирішення проблем.

Spring Boot є надбудовою над іншим фреймворком Java – Spring. Spring Boot надає усі можливості Spring, але покращує деякі нюанси роботи. Нижче наведено опис фреймворку Spring.

Spring часто називають фреймворком із фреймворків, оскільки він може інтегруватися майже з будь-яким модулем, написаним на Java. Його було створено, щоб спростити доволі складні технології Java EE. На відміну від Java EE, Spring не вимагає написання купи повторюваного, майже порожнього коду для реалізації дуже простих завдань, таких як доступ до бази даних або створення відповідей клієнтам. Автори роблять усе можливе, щоб розробнику можна було зосередитися на написанні бізнес-логіки програми під час створення коду. Таким чином програми на базі Spring легше підтримувати та розробляти завдяки меншій кількості рядків коду і як результат покращеної читабельності.

Основні функції, що надаються фреймворком Spring:

– Spring має можливість підключити додаткові залежності, зважаючи на потреби додатку;

– Spring є дуже легким фреймворком і без підключення додаткових залежностей займає 5 мегабайтів;

– Spring надає можливість описувати, налаштовувати та завантажувати в контекст програми так звані bean-класи – основні структурні елементи програми, які в основному є звичайними класами Java;

– Завдяки концепції bean-класу, Spring полегшує написання слабо зв’язаного коду, де функціональність залежить від абстракцій – інтерфейсів, які описують деякі функції, але не мають реалізації. Реалізація визначається на етапі запуску додатку, виходячи із конфігурації системи та коду розробника;

– Має модуль для інтеграції із базою даних, надає набір анотацій для опису сутностей ORM та інтерфейсів доступу до даних, що, у свою чергу, допомагає створити гнучкий і зручний рівень доступу до даних. Ці підходи використовуються для роботи із реляційними та NoSQL базами даних;

– Вбудована структура MVC з основними класами для обробки запитів і створення необхідного API, доступного у мережі. Модуль MVC може бути розширеним для впровадження специфічної обробки та авторизації запитів;

– Пропонує можливість аспектно-орієнтованої розробки, яка є стилем програмування, у якому частини функціоналу виокремлюється за відповідною потребою для використання у всіх шарах програми. Такі функції, або так звані аспекти, зазвичай виконують однаково важливі завдання в будь-якій частині програми, як наприклад логування, збір метрик, тощо;

– Інтеграція з великою кількістю проміжного ПЗ. Через популярність фреймворку багато розробників проміжного програмного забезпечення забезпечують взаємодію зі своїм продуктами;

– Надає бібліотеки тестування як розширення до найпопулярніших фреймворків тестування, із додатковим функціоналом для тестування саме додатків написаних на Spring.

Отже Spring Boot поєднує у собі усі переваги, що надає Spring, а також додає додатковий шар для значного пришвидшення конфігурації та запуску програм. Деякі із його переваг описано нижче:

– Скорочує час розробки та підвищує продуктивність команди розробників;

– Автоматичне налаштування компонентів програми, так званих bean-класів;

– Зменшує кількість коду, що пише розробник;

– Запускає та налаштовує HTTP серверер додатків.

Таким чином Spring Boot є оптимльним вибором при розробці системи із огляду на гнучкість та швидкість створення додатків із мікросервісною архітектурою, а також добре відтестовані та зручні у використанні бібліотеки.

Висновки до розділу 2

У розділі було проведено порівняння архітектурних підходів, механізмів створення API а також огляд технологій для побудови проекту таких як мова програмування та фреймворк. У результаті було запропоновано впровадження системи із мікросервісною архітектурою з використанням REST API. Основні засоби розробки мова Java та фреймворк Spring Boot.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Огляд архітектури додатку

Перед початком розробки було розроблено проект архітектури для побудови систем. Високорівнева діаграма системи наведена на рисунку 3.1.

Рисунок 3.1 – Діаграма високорівневої архітектури розробленої системи

Як можна побачити із діаграми система складається із окремих сервісів, кожен із яких має специфічні задачі, та умовно розділена на основний функціонал та сервіс процесів, що взаємодіють за допомогою брокеру повідомлень. Взаємодія між деякими сервісами відбувається за допомогою HTTP викликів.

Система використовує сховище об’єктів, для зберігання файлів, а також SMTP сервер для відправки email-повідомлень.

Основні елементи системи не відкривають своє API для глобальної мережі, та доступні лише через так званий шлюз, який доступний із зовні. Такий підхід допомагає відділити службові API, які створені для використання лише у системі, від публічних – які будуть використані клієнтами для інтеграції.

3.1.1 Шлюз API

Шлюз API забезпечує централізовану точку входу для зовнішніх споживачів, незалежно від кількості чи складу мікросервісів. Таким чином клієнти не можуть отримати прямий доступ до сервісів всередині системи. Шлюзи API часто можуть включати додаткові рівні обмеження кількості запитів за проміжок часу, швидкості та безпеки.

Цей підхід є реалізацією таких поширених шаблонів як: фасад і адаптер. Шлюз є фасадом що відкриває певні можливості для зовнішнього користувача, інкапсулюючи логіку що надає послуги. Також у ключі адаптеру може надавати клієнту послуги навіть якщо інтерфейс сервісу системи не сумісний із очікуваннями користувача.

Головні переваги використання шлюзу API:

1. Керування навантаження на систему. Шлюз може визначати максимальну кількість запитів від одного клієнту за визначений проміжок часу таким чином не навантажуючи систему до відмови. Не пропускає запити від анонімних користувачів до систему, що робить систему більш стійкою до DdoS атак. Можливість легко масштабувати шлюз, за рахунок створення копій, це зумовлено відсутністю стану шлюзу;

2. Підвищена безпека. Шлюз робить аутентифікацію запитів користувача і відкриває лише частину доступного API із системи, завдяки чому значно зменшує кількість можливих варіантів атак від зловмисників;

3. Інкапсулює службові функції системи. Шлюз може відповідати за додаткові функції системи, такі як збір метрик, логування, перевірка стану системи тощо. Завдяки чому можна отримати більш чіткі дані про роботу системи у цілому та виключити необхідність дублювати логіку пов’язану зі службовими функціями у кожному окремому сервісі;

4. Спрощення інтеграції. Надає можливість для агрегування даних із кількох сервісів а також модифікації чи зміни формату. За рахунок цього не виникає

потреби робити зміну у вже існуючому функціоналі сервісів і як результат уникання значного ускладнення системи. Також існує можливість переключення версійності API чи навіть постачальника послуг без потреби змін на стороні клієнта.

3.1.2 Брокер повідомлень

Мікросервіси відокремлені один від одного та існують автономно, але можуть спілкуватися один з одним. Перехресна залежність є типовою особливістю архітектури мікросервісів, що означає, що жодна служба не може працювати без допомоги інших служб. Частково ця потреба покривається наявним REST API системи але такий підхід є оптимальним не завжди.

Наприклад для роботи із бізнес процесом, що стартує у заплановану дату треба реалізовувати відкладену обробку його кроку. Збереження такої події системи у окремому брокері повідомлень, зазначивши дату для обробки, є одним із підходів для планування обробки подій, які заплановані на певний час.

Також можливість асинхронної обробки повідомлень робить систему більш відмовостійкою та надійною. Сервіси можуть обробляти події за можливості, обмеживши кількість паралельної оброки, за рахунок чого система не буде перевантажена, а події які не обробляються чекатимуть у черзі повідомлень. А у разі недоступності певного сервісу повідомлення не буде втрачено, як у випадку із REST API, і може бути оброблено пізніше.

Брокер повідомлень діє як медіатор для мікросервісів, зберігаючи запити від однієї програми-постачальника і передаючи програмам-споживачам. У нашому випадку використання простих черг не є оптимальним – іноді виникає потреба у обробці однієї і тієї ж події декількома сервісами. Натомість слід використати так звані топіки повідомлень – де повідомлення направляється у так званий топік сервісом-постачальником, а на стороні сервіса-споживача створюється підписка, на

основі якої сервіс користувач може отримувати повідомлення. За рахунок такого підходу можна розподілити одну подію для обробки кожним споживачем окремо, не дублюючи логіку і не створюючи ідентичні черги із дублікатами повідомлень. Схему обробки подій, із використанням, топіків зображено на рисунку 3.2.

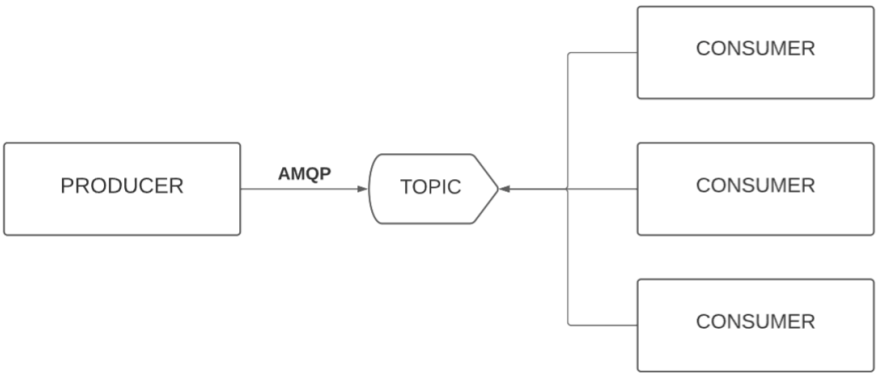

Рисунок 3.2 – Схема обробки на основі топіків

Для реалізації системи використано брокер повідомлень ActiveMQ. ActiveMQ дозволяє асинхронну обробку, відкладаючи обробку поміщених подій. Таким чином, ActiveMQ ідеально підходить для тривалих завдань чи при ліміті паралельної обробки подій сервісом, дозволяючи сервісам відповідати на запити за можливості, а не виконувати завдання з інтенсивними обчисленнями одразу.

3.2 Account сервіс

Одним із сервісів які є частиною системи є Account сервіс, відповідальністю цього сервісу є створення та збереження облікових записів користувачів систем, зберігання контактної інформації користувача а також його належність до певної кафедри чи факультету.

Цей сервіс поєднує усю інформацію про користувача, що може знадобитися для обробки запитів у системі, таким чином ці інформацію можна агрегувати та закодувати у JWT токен під час первинної аутентифікації користувача.

3.2.1 Механізм авторизації запитів

Механізмом для авторизації запитів виступає JWT токен. Це доволі гнучкий підхід який дозволяє визначити користувача, його ролі та доступи у системі а також може передавати додаткові дані. Такий підхід є дуже зручним при впроваджені систем які не містять стану, як наприклад локальне сховище сесій, і потребують додаткових даних від клієнта для авторизації запиту. Також дуже вдало використовується у розподілених системах, де лише один сервіс агрегує інформацію про користувача, для авторизації запитів та формування відповіді без додаткових запитів між сервісами.

JWT токен генерується на стороні серверу та складається із трьох частин:

– Header – як правило складається із двох полів, вказує тип токену та алгоритм для підпису;

– Payload – основна частина токену, що містить головну інформацію токену. Може містити будь яку інформацію залежно від імплементації, доволі часто використовується для визначення строку життя токену, вказує сервіс який згенерував токен, для якого користувача створено токен, та хто може бути потенційною аудиторією для обробки токену;

– Signature – підпис, що створено на основі алгоритму, вказаного у Header. Алгоритм створення підписує токен використовуючи секрет чи приватний ключ.

При обробці запиту клієнт перевіряє вірність підпису та цілісність даних токену а також використовує додаткову інформації із частини Payload, основними службовими полями є:

– sub – ідентифікує сутність чи користувача, що робить запит;

– scp – ідентифікує права та ролі;

– iat – ідентифікує точку у часі, в уніфікованому форматі, коли токен перестане було створено;

– exp – ідентифікує точку у часі, в уніфікованому форматі, коли токен перестане бути актуальним.

Після генерації та підпису токен зашифровується у 64-розрядну систему кодування та має формат EncodedHeader.EncodedPayload.Singniture. Після оброки на стороні клієнта можна розкодувати токен та отримати необхідну інформацію для обробки запиту.

Приклад інформації, у декодованих частинах JWT - Header та Payload, для токену, що використовує система наведено на рисунку 3.3.

Рисунок 3.3 – Приклад інформації закодованої у Header та Payload частинах JWT токену, що використовує система

Для підвищення безпеки тривалість життя токену робиться обмеженою – таким чином якщо зловмисники змогли перехопити токен вони би не отримували постійний доступ до системи.

З іншої сторони якщо користувач проводить значну кількість часу у системі то кожен раз коли життя токену вичерпується буде виникати потреба повторної авторизації у системі – це призведе до погіршення досвіду користувача, під час використання системи, спричинить ряд помилок у системі і за деяких обставин навіть може призвести до пошкодження цілісності даних.

Гарним рішенням для балансування між досвідом користувача та безпекою системи є впровадження додаткового токену, під назвою refresh_token, такий токен генерується під час первинної авторизації користувача. Refresh токен не містить додаткової інформації окрім ідентифікатора користувача якому його було видано та час вичерпання життя токену. Refresh токен залишається активним доволі довго,

як правило добами – таким чином можна бути впевненим що сесія користувача не буде призупинена неочікувано.

У момент коли система не може авторизувати запит від клієнта, за допомогою основного access_token та надсилає статус код 401, клієнт може скористатися refresh_token для отримання оновленого access_token. Схему алгоритму авторизації та оновлення токену зображено на рисунку 3.4.

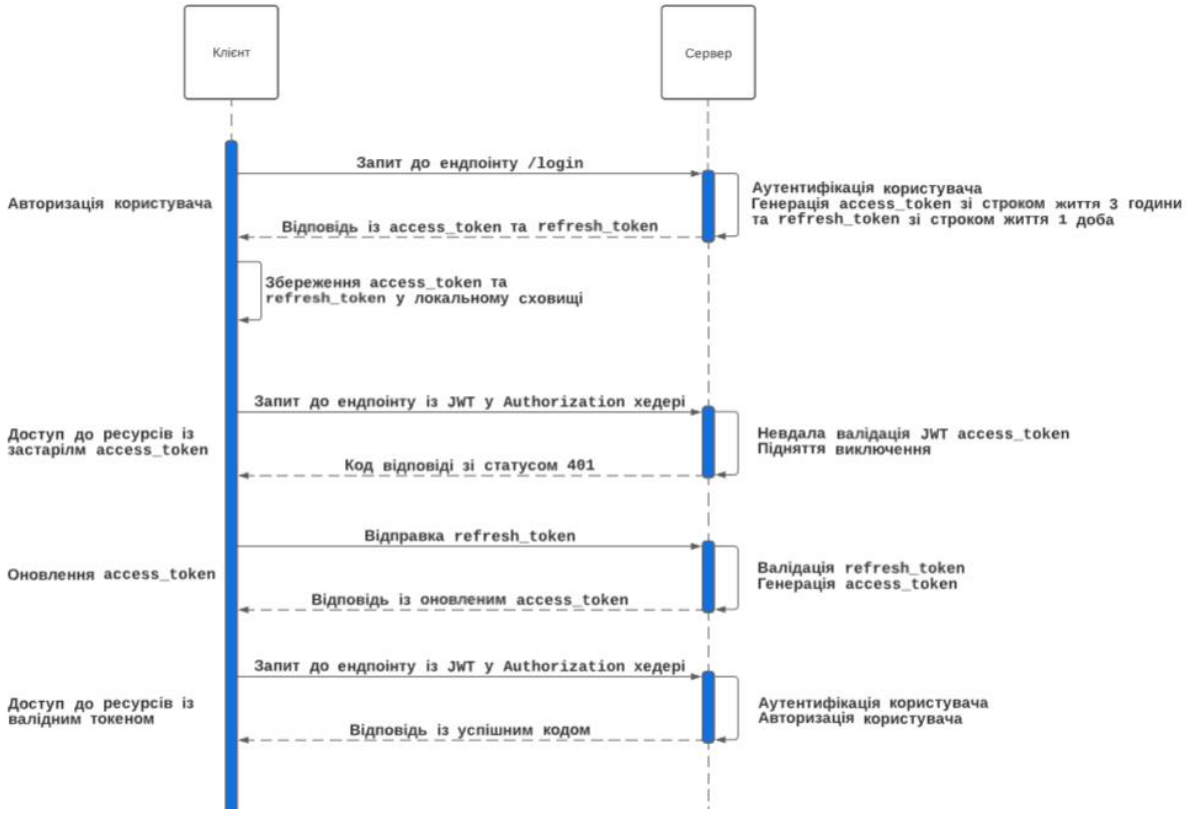

Рисунок 3.4 – Схема алгоритму авторизації запитів та оновлення access_token із використанням refresh_token

За рахунок того, що Refresh токен відправляється через мережу не часто – тільки у разі вичерпання основного токену, перехопити його стає набагато складніше.

Із використанням підходу можна регулювати довжину сесії користувача без потреби додатково авторизовуватися у систему та із мінімальним ризиком перехоплення токенів.

3.2.2 Огляд схеми бази даних Account сервісу

Інформація у базі даних зосереджена навколо таких сутностей як обліковий запис користувача, ролі, контактна інформація, кафедри та факультети.

Схему бази даних Account сервісу наведено на рисунку 3.5.

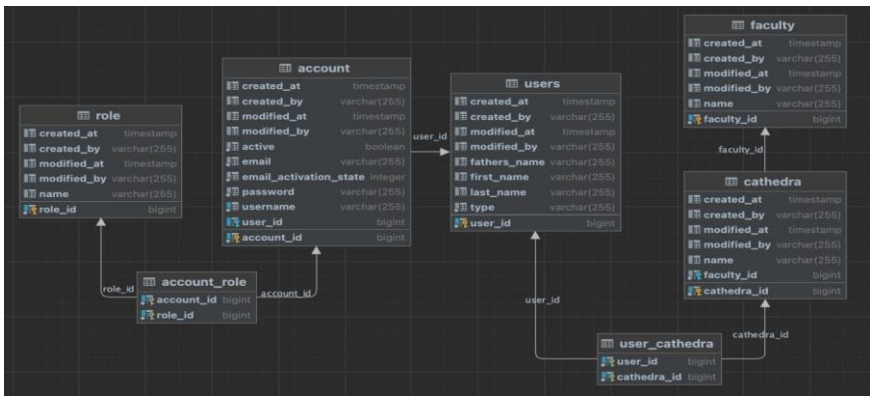

Рисунок 3.5 – Схема бази даних Account сервісу

Усі дані зберігаються у реляційній базі даних, за рахунок структурованості та чітко визначених зв’язків.

3.2.3 Огляд API Account сервісу

Account сервіс реалізує REST API, із основних задач сервісу є надання можливості створення облікового запису та авторизації користувача. Автозгенеровану документацію Swagger для створення облікового запису наведено на рисунку 3.6.

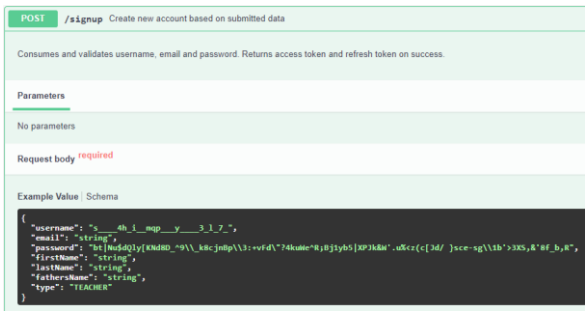

Рисунок 3.6 – Swagger документація API для створення облікового запису

Swagger документація для API авторизації зображено на рисунку 3.7.

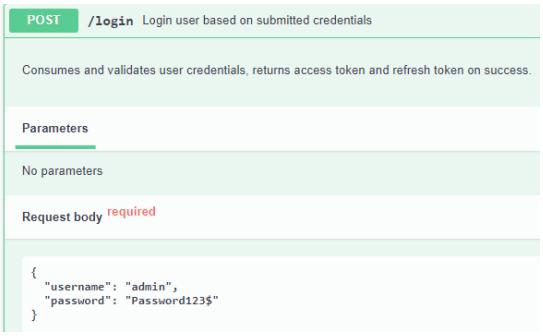

Рисунок 3.7 – Swagger документація API для авторизації користувача

Окрім цього створено CRUD API для маніпуляції головними ресурсами сервісу, такими як account, app-user, cathedra, faculty, role.

Приклад документації Swagger для API контролю ресурсу app-user наведено на рисунку 3.8.

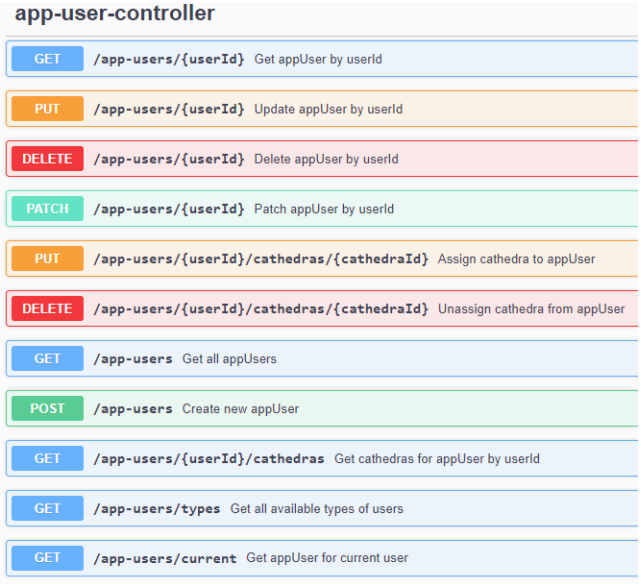

Рисунок 3.8 – Swagger документація із описом API для маніпуляції ресурсом app-user

API надає можливості для створення ресурсів, створення зав’язків, редагування ресурсів, видалення ресурсів та відображення ресурсів із сортуванням, фільтрацією та пагінацією.

Усі запити використовують JSON формат для обміну даними та HTTP хедер Authorization для передачі JWT токену у цілях аутентифікації та авторизації.

3.3 Storage сервіс

Storage сервіс відповідає за зберігання фалів, завантаження файлів зі сховища, адміністрацію доступу до файлу по окремим операціям: читання, видалення, запис. Таким чином користувач може зберегти файл у системі та за потреби делегувати частину доступів іншому користувачу.

У своїй суті Storage сервіс є фасадом над стороннім файловим сховищем для надання додаткового функціоналу: контрою доступу для окремих користувачів та перегляду мета даних без необхідності завантажувати файлу. Завдяки цьому можна отримати гнучке рішення у плані контролю доступу до даних, при цьому використавши відтестовані та ефективні механізми для безпосередньої маніпуляції файлами.

3.3.1 Azure Blob Storage

Для зберігання файлів використано Azur Blob Storage, сервіс для зберігання об’єктів. Сховище BLOB-об’єктів створено, щоб забезпечити потреби розробників додатків у масштабованості, безпеки та доступності, при роботі із великими файлами. Сховище підтримує найпопулярніші фреймворки розробки, включаючи Spring Boot і є єдиною службою хмарного зберігання, яка пропонує вдосконалений рівень зберігання об’єктів на основі SSD.

Файлове сховище зберігає та організовує дані в папки, подібно до файлів, які зберігаються на файловому носієві. Ця сама ієрархічна структура зберігання використана для зберігання файлів відносно окремого користувача системи, де відповідно до кожного користувацького логін створюється папка у яку зберігаються відповідні записи.

Також значно покращено швидкість завантаження, відносно локального сховища. Це зумовлено механізмом зберігання файлів на фізичних носіях а також можливістю робити репліки даних для доступу у різних частинах світу.

Такий функціонал може надати хмарний провайдер Azure, надійний постачальник послуг із гнучким тарифним планом, що базується безпосередньо на кількості збереженої інформації та читання і запису даних.

Дані на сервері набагато краще захищені за рахунок серйозного підходу у плануванні інфраструктури безпеки, а також безпосереднього захисту фізичних серверів – ресурсами постачальника хмарних послуг.

3.3.2 Огляд схеми бази даних Storage сервісу

Схему бази даних побудовано навколо таких сутностей як файли та доступи. Схему бази даних наведено на рисунку 3.9.

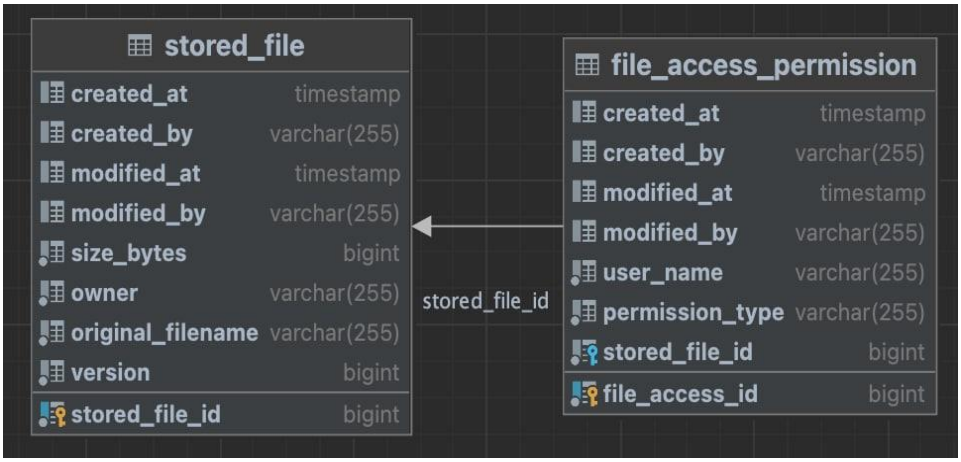

Рисунок 3.9 – Схема бази даних Storage сервісу

Модель бази даних передбачає збереження посилання на файл у Azure Blob Storage, об’єм пам’яті який займає файл, версію файлу та ім’я користувача який створив файл.

До кожного файлу можна задати рівні доступу для окремо взятого користувача.

3.3.3 Огляд API Storage сервісу

Storage сервіс реалізує REST API і однією із основних задач сервісу є надання можливості збереження файлів, редагування файлів, попередній перегляд файлів у браузері, завантаження файлів. Також доступний функціонал для адміністрації доступів на читання, оновлення та видалення файлу. Користувач, який зберіг файл може делегувати доступи іншим користувачам.

Приклад запиту до API для збереження файлу та відповідь серверу наведено на рисунку 3.10.

Рисунок 3.10 – Приклад виклику API для збереження файлу

На рисунку 3.11 зображено приклад виклику API для перегляду доступів до файлу.

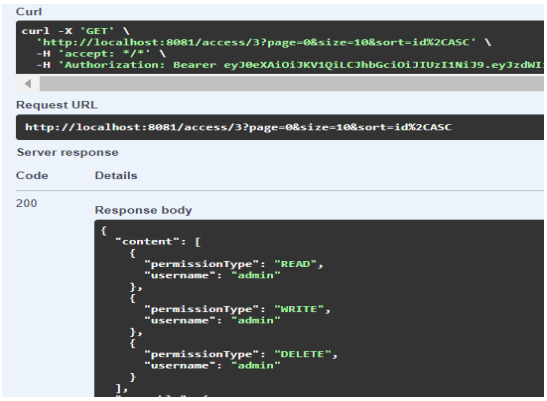

Рисунок 3.11 – Відображення доступів до файлу користувачем

Також існує API для основних маніпуляцій із такими ресурсами як файл та рівень доступу до файлу. На рисунку 3.12 зображено Swagger документацію для операцій доступних для ресурсу file.

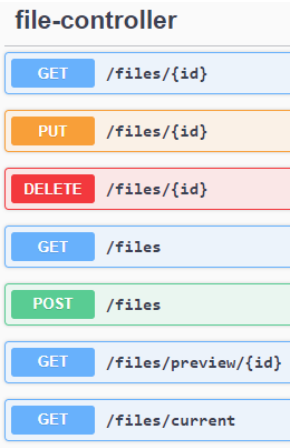

Рисунок 3.12 – Зображення Swagger документації для ресурсу file

Таким чином існуюче API надає усі необхідні функції для маніпуляції файлами, доступами до файлів, а також зберігає метаінформацію про файли.

3.4 Mail сервіс

Сервіс відповідає за створення та відправку імейлів на основі шаблонів, має функціонал для створення, оновлення та видалення шаблонів, зберігає історію відправлених повідомлень. Сервіс використовує Thymleaf у якості шаблонізатору, для генерації тексту повідомлень на основі шаблонів. Також надає можливість додавати файли, збережені у Storage сервісі, як вкладення до повідомлень.

Сервіс може відправляти повідомлення за готовими шаблонами у результаті певних подій в системі, або бути використаним для полегшення відправки повідомлення користувачем у ручному режимі.

3.4.1 Бібліотека шаблонізації Thymeleaf

Thymeleaf це система шаблонів Java для форматів XML/XHTML/HTML5, яка може працювати як у веб-середовищі, на основі сервлетів, так і в інших

середовищах Java. Добре підходить для обслуговування XHTML/HTML5 під час генерації сторінок на стороні серверу. Має повну інтеграцію із Spring Boot.

Метою Thymeleaf є надання стильного та добре сформованого способу створення шаблонів. Він заснований на тегах і атрибутах XML. Ці XML-теги визначають виконання попередньо визначеної логіки в DOM. Thymeleaf також дозволяє визначити наш власний режим, вказавши способи аналізу шаблонів у цьому режимі.

Функціонал, що надає бібліотека, можна просто пристосувати до потреб генерації повідомлень за шаблонами із форматів HTML та простого тексту. У випадку коли користувачу необхідно робити попередній перегляд повідомлення.

3.4.2 Огляд схеми бази даних Mail сервісу

Схему бази даних має сутності email та template, як показано на рисунку 3.13.

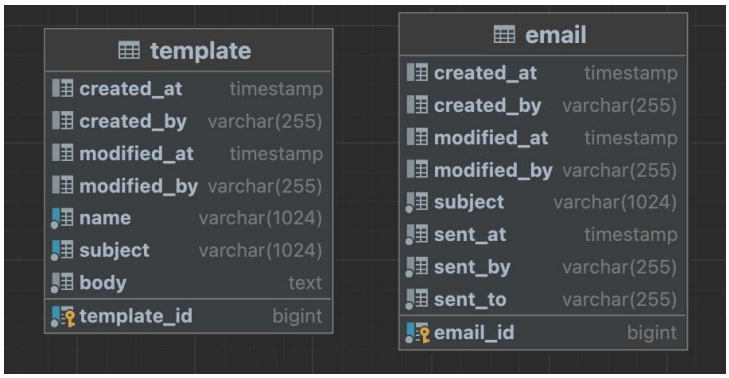

Рисунок 3.13 – Модель бази даних mail сервісу

Схема бази передбачає зберігання шаблонів повідомлень із основним змістом у форматі BLOB, темою повідомлення для відправки та ім’ям шаблону. Саме ця сутність використовується Thymleaf при побудові повідомлень.

Сутність Email зберігає основні дані про відправлені повідомлення – час відправки, кому відправлено повідомлення, ким відправлено повідомлення та тема повідомлення.

3.4.3 Огляд API Mail сервісу

API сервісу надає основний функціонал для створення шаблонів та відправки повідомлень. Приклад запиту збереження шаблону наведено на рисунку 3.14.

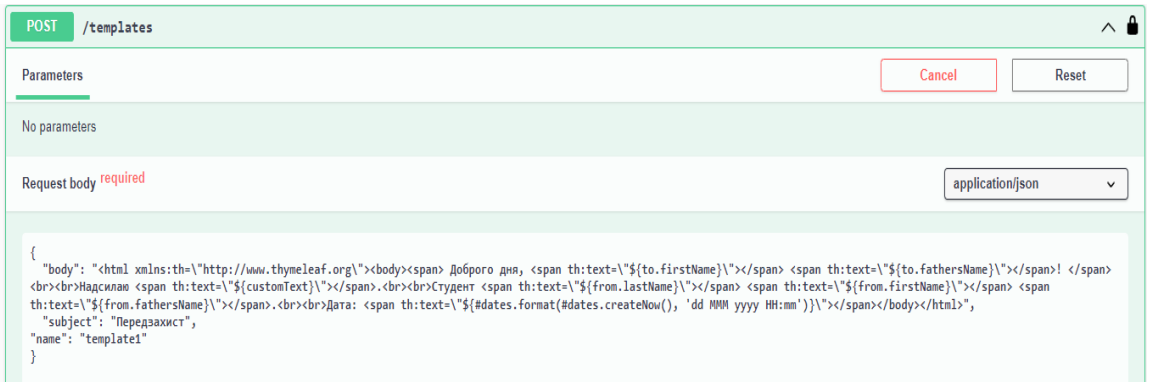

Рисунок 3.14 – Приклад запиту для збереження шаблону повідомлення

На рисунку 3.15 зображено запит для відправки повідомлення за шаблоном.

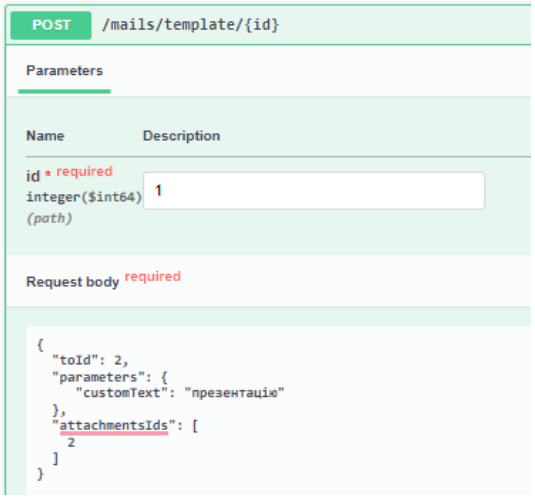

Рисунок 3.15 – Приклад запиту відправки повідомлення за шаблоном

На рисунку 3.15 показано приклад відправки повідомлення за шаблоном, із вказаним отримувачем, додатковими параметрами, що будуть використані Thymleaf для заповнення шаблону та ідентифікаторами прикріплених файлі, що зберігаються у Storage сервісі. Приклад отриманого повідомлення зображено на рисунку 3.16.

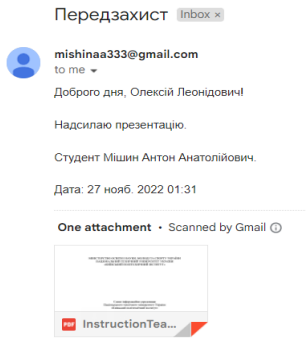

Рисунок 3.16 – Приклад повідомлення згенерованого на основі шаблону повідомлень

Завдяки такому API Email сервіс надає можливості для швидкої генерації тіла повідомлення для ручної відправки із додатковим редагуванням або автоматичної відправки під час виконання бізнес-процесу.

3.5 Process сервіс

Process сервіс відповідає за збереження схем процесів, запуск процесів та контроль їх виконання на усіх етапах, відповідними учасниками. Схеми процесів зберігаються у NoSQL базі даних, така вимога пов’язана із тим що процеси не можуть мати чіткої структури та найкраще представляються у якості JSON об’єкту.

Сервіс оркеструє усе що відбувається у системі за рахунок відправки подій, які потім будуть оброблятися основною частиною системи.

3.5.1 NoSQL база даних MongoDB

MongoDB — це документо-орієнтована база, яка утворена за стандартом NoSQL. Цей стиль популярний, тому що може зручно зберігати інформацію, яка не може бути збережена у реляційних базах даних, через неструктурованість даних.

База даних була обрана тому, що бізнес процеси передбачують гнучке налаштування та не можуть мати визначеної структури заздалегідь. Бібліотеки мають коди реалізації програмного забезпечення з відкритим доступом, які не вимагають опису таблиць бази даних. СКБД написана мовою програмування C++, тому швидкість виконання запитів висока, за рахунок оптимізації. Має високу гнучкість через відсутність потреби описувати схему даних та типізувати їх. Документи в MongoDB відображаються у форматах JSON чи BSON. Таким чином, використання такої моделі краще кодується та керується, а внутрішня згуртованість пов’язаних даних забезпечує швидке виконання запитів.

3.5.2 Огляд схеми бази даних Process сервісу

Частина бази даних знаходиться у MonogoDB та зберігає лише одну колекцію зі схемами бізнес процесів. Приклад документу зі схемою бізнес процесу зображено на рисунку 3.17.

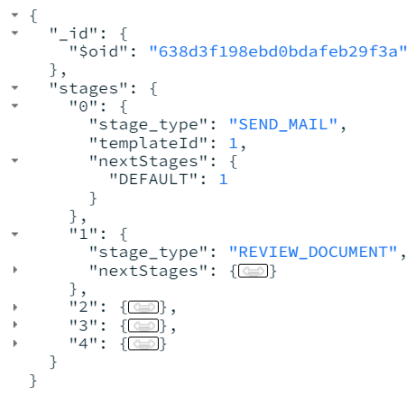

Рисунок 3.17 – Приклад схеми бізнес процесу збереженого у MongoDB

Інша частина даних зберігається безпосередньо у реляційній базі даних через наявність структури та відношень.

Сутності які зберігаються у реляційній базі це записи запущених бізнес процесів та коментарі до кроків бізнес процесу. Схему бази даних наведено на рисунку 3.18.

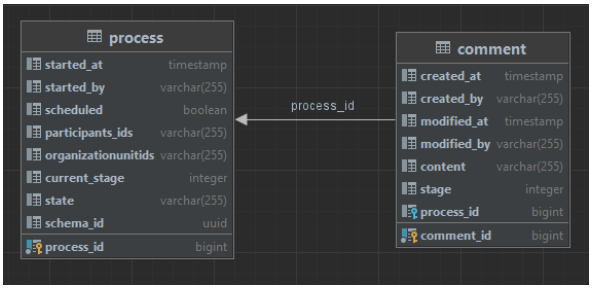

Рисунок 3.18 – Схема бази даних Process сервісу

Таким чином Process сервіс має можливість зберігати як структуровані так і не структуровані дані.

3.5.3 Огляд API Process сервісу

API Process сервісу передбачає створення схем бізнес процесів та запуск процесів.

Основний фокус на можливості створити модель бізнес процесу із використанням графу у JSON форматі, де кожен окремий вузол графу буде відповідати етапу процесу, а також наступний крок, залежно від виконання поточного.

Графічне відображення схеми процесу можна побачити на рисунку 3.20.

Рисунок 3.20 – Схема бізнес процесу складання предмету

Виклик API Process сервісу, для збереження схеми бізнес-процесу, що було зображено рисунку 3.20, можна побачити на рисунку 3.21.

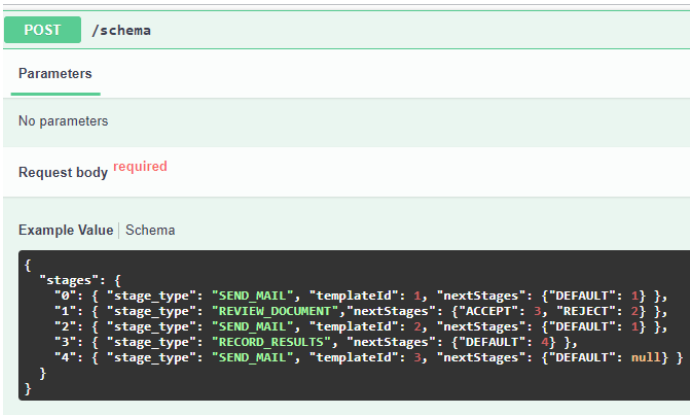

Рисунок 3.20 – Приклад запиту для створення схеми процесу

Як можна побачити бізнес процес представлений JSON об’єктом із форматом ключ-значення. Кожне значення має колекцію індексів наступних кроків, залежно від результату виконання поточного етапу.

Висновки до розділу 3

У розділі розглянуто програмну реалізацію інформаційної системи для автоматизації роботи кафедри. Зроблено огляд архітектури додатку на основі мікросервісного підходу, спосіб доступу до системи із зовнішньої мережі, методику взаємодії компонентів системи.

Було зроблено основного функціоналу, що надає кожен окремо взятий сервіс. Для кожного структурного елементу було розглянуто основні цілі, схему базу даних та API.

4 ТЕСТУВАННЯ

4.1 Модульне тестування

Модульне тестування — це метод спосіб тестування коду програми, при якому тестується окремий компонент коду у ізоляції від усієї системи. Розробники пишуть юніт тести для покриття створеного коду, щоб переконатися, що код працює правильно. Це спрямовано на виявлення та запобігання можливих несправностей програми. Модульні тести зазвичай пишуться у форматі сценарію, перевіряючи поведінку компоненту при певних умовах.

Переваги модульного тестування:

1. Швидке виявлення помилок. Код із тестовим покриттям надійніший, ніж код без нього. Якщо зміни до коду призведуть до помилок у функціоналі, ще на етапі розробки, за умови правильного написання тестів причину можна буде визначити одразу.

2. Заощадження ресурсів. При написанні модульних тестів на етапі побудови програмного забезпечення виявляється багато помилок. У результаті мануальне тестування спрощується та проблеми не потрапляють до кінцевого користувача, що призводить до менших витрат часу та фінансів.

3. Документація. При умові правильно побудованої структури системи та її компонентів, юніт тести можуть відображати функціонал системи покриваючи основні сценарії використання її компонентів. Таким чином ознайомлення із тестами надає можливість швидко розібратися із кодовою базою проекту.

4. Зменшення складності проекту. При написанні юніт-тестів розробник має змогу проаналізувати складність продукту та задачі, що вже вирішені кодовою базою. У результаті створення юніт тестів може слугувати підказкою для можливості рефакторингу коду або просто бути індикатором надлишкової складності. Загалом якість і складність юніт-тестів відображає напряму якість та складність самого коду системи.

Результат виконання модельного тесту, у середовищі розробки Intellij, можна побачити на рисунку 4.1.

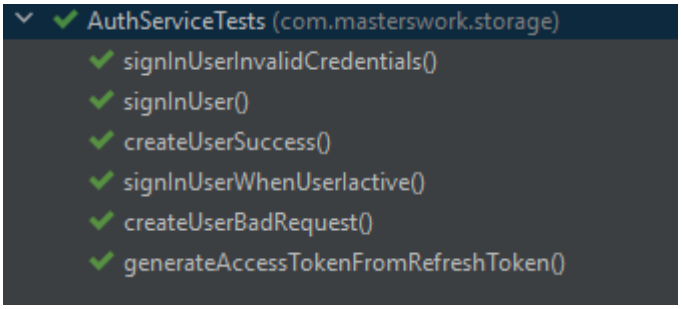

Рисунок 4.1 – Результат виконання тестів для компоненту AuthService

4.2 Інтеграційне тестування

Інтеграційне тестування визначається як тип тестування, у якому програмні модулі логічно інтегровані та тестуються як група. Типовий програмний проект складається з кількох програмних модулів, закодованих різними програмістами. Метою цього рівня тестування є виявлення недоліків у взаємодії цих програмних модулів під час їх інтеграції.

У випадку розробки мікросервісного додатку інтеграційне тестування буде зосереджено саме на окремому мікросервісі, намагаючись запустити сервіс із попередньою конфігурацією та станом для тестування певного наскрізного сценарію. Таким чином можна впроваджувати тестування сервісу у умова наближених до реальних із використанням бази даних та роботою усієї бізнес логіки. Найчастіше сценарії таких тестів побудовані навколо викликів тих чи інакших API, а успішність тестів визначається кодом відповіді та даними, які повертає API у відповідь. І у результаті отримується протестований сервіс, що виконує певний контракт, також такі тести, на відміну від юніт-тестів, можуть виявляти помилки системи пов’язані із інфраструктурою додатку.

Приклад виконання інтеграційних тестів для Account сервісу зображено на рисунку 4.2.

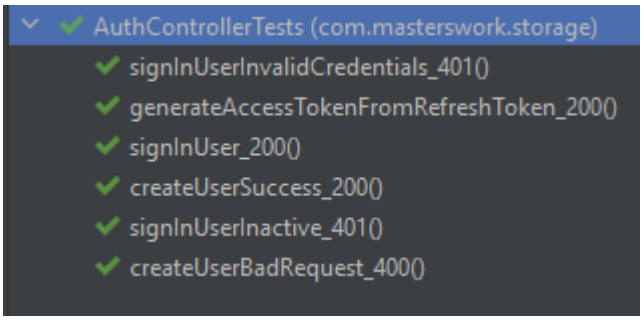

Рисунок 4.2 – Приклад виконання інтеграційних тестів

Висновки до розділу 4

У розділі було зроблено огляд основних підходів до тестування коду і системи у цілому. Було описано мету та преваги таких підходів як юніт-тестування та інтеграційне тестування, а також наведено приклад виконання модульних та інтеграційних тестів.

5 РОЗРОБКА СТАРТАП-ПРОЕКТУ

Метою цього розділу є маркетинговий аналіз стартап проекту для визначення основних можливостей його ринкової реалізації та можливого напряму розвитку реалізації.

5.1 Опис ідеї проекту

Першим пунктом слід проаналізувати зміст ідеї стартап проекту, напрямки застосування і вигоди для користувач. Інформація наведена нижче (табл. 5.1).

Таблиця 5.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розробити та впровадити інформаційну систему, для часткової автоматизації роботи кафедри та інформування, надати інструменти для використання електронного документообігу. Система повинна використовувати модель no code/low code для налаштування нових процесів без зміни коду.	1. Автоматизація процесів пов’язаних із керуванням кафедрою, автоматизації навчального процесу та взаємодії адміністрації за студентами та викладачами	1. Економія часу на виконання повсякденних задач, за рахунок впровадження гнучких бізнес-процесів і системи сповіщень. Інструмент для формалізації та аудиту взаємодій у рамках процесів на кафедрі.
	2. Система зберігання та адміністрування доступу до цифрових документів	2. Підвищення ефективності роботи працівників, завдяки використанню електронного документообігу. Можливість використовувати цифрові підписи

Отже, згідно з таблицею 5.1 – основною ідеєю інформаційної системи є впровадження функціоналу, на основі моделі no code/low code, що дозволить автоматизувати процеси на кафедрі, значно скоротить використання фізичних документів, що є очевидною економією коштів, а також підвищить продуктивності праці та умов праці тому що вчителі витрачають значно менше часу на створення, заповнення, обробку та облік документів. Допоможе викладачам налаштовувати процеси для викладання предметів та здачі завдань, що спростить організаційні моменти пов’язані із викладанням предмету, зробить вимоги більш прозорими і як результат підвищить інформованість та успішність студентів.

Наступний етап це аналіз потенційних техніко-економічних переваг ідеї порівняно із пропозиціями конкурентів. Проводиться порівняльний аналіз показників запропонованої системи та систем конкурентів – сильних сторін (S), слабких (W) та нейтральних (N). Результат порівняння приведено нижче (табл. 5.2).

Таблиця 5.2 – Визначення ключових характеристик ідеї проекту

Техніко-економічні характери-тики ідеї	Продукція конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
	Мій проект	Campus	НТУ “ХПІ”			
Можливість адміністрації індивідуальних користувачів та груп користувачів	+	-	-			Адміністрація рівнів доступу окремих користувачі та груп є значною перевагою, особливо для закладу із різних департаментів

Таблиця 5.2 (продовження)

Налаштування бізнес процесів без необхідності вносити зміни в код проекту	+	-	-	Високий рівень свободи користувача може призвести до непередбачуваних наслідків у роботі системи		Один із ключових переваг яку не має жоден із конкурентів на ринку, спрощує автоматизацію процесів без залучення технічних спеціалістів, заощаджує значну кількість ресурсів та часу
Електронний документообіг	+	+/-	+			Суттєво полегшує роботу із документами та убезпечує їх від втрати
Механізм сповіщення поза межами системи	+	-	-			Надає можливість користувачам отримувати сповіщення на пошту, не використовуючи систему безпосередньо

Аналіз ключових характеристик системи показав, що на ринку немає значних конкурентоспроможних продуктів. Основною перевагою є можливість впровадження нових бізнес процесів у короткі строки та без залучення розробників із технічними знаннями, така перевага значно зменшує витрати на розширення та підтримку системи. Також гнучка мікросервісна архітектура та REST API створюють умови для розширення системи та інтеграції зі сторонніми сервісами.

5.2 Технологічний аудит ідеї проекту

У рамках маркетингового аналізу було проведено аудит технологій, аудит враховує такі характеристики технології як наявність та доступність, результати аудиту наведено нижче (табл. 5.3).

Таблиця 5.3 – Технологічна здійсненність ідеї проекту

Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
Механізм налаштування та запуску бізнес-процесів.	Середовище розробки: IntellijIdea Бази даних: реляційна PostgreSQL, документо-орієнтована MongoDB. Серверна частина: Spring Boot, APIM Gateway, Hibernate, MapStruct, ActiveMQ	+	Технології доступні та мають як платні версії для комерційних проектів так і безкоштовні для етапу розробки. Для платних існує ряд альтернатив із різними ціновими рівнями.

Таблиця 5.3 (продовження)

Збереження та адміністрація доступу до документів	Об’єктне сховище Azure Blob Storage	+	Технології доступні та мають безкоштовну версію для розробки та гнучкі тарифи для комерційної версії
Система інформування користувачів	Шаблонізатор Thymleaf, Spring Boot Starter Mail	+	Технології доступні безкоштовно
Інтеграція зі сторонніми сервісами	JWT java auth, FeignClient, Spring Security Starter	+	Технології доступні безкоштовно

За результатами аналізу можна чітко стверджувати, що всі необхідні технології наявні, не потребують розробки чи модифікації та доступні. Тому проект є технічно здійсненним.

5.3 Аналіз ринкових можливостей запуску стартап-проекту

Для успішного розвитку проекту слід обрати зробити огляд ринкових можливостей та ринкових загроз. Оцінку проведено у нижче (табл. 5.4).

Таблиця 5.4 – Попередня характеристика потенційного ринку стартап-проекту

Показники стану ринку	Характеристика
Кількість головних гравців, од	2

Таблиця 5.4 (продовження)

Загальний обсяг продаж, грн/ум.од	350000
Динаміка ринку (якісна оцінка)	Зростає
Наявність обмежень для входу	Немає
Специфічні вимоги до стандартизації та сертифікації	Немає
Середня норма рентабельності в галузі (або по ринку), %	39%

Середня норма рентабельності у галузі порівнюється із відсотком на вкладення. Інвестиції в проект мають сенс якщо відсоток рентабельності вищий.

Порівняння груп потенційних клієнтів та характеристик клієнтів із відповідними до кожної групи вимогами до товару наведено нижче (табл 5.5).

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
Автоматизація роботи процесів кафедри	Заклади освіти	Мета використання програмного продукту: Автоматизація роботи кафедри, автоматизація навчального процесу	Зручний інтерфейс, швидкість, надійність у використанні, точність роботи

Після визначення груп потенційних споживачів аналізується ринкове середовище: показники (табл. 5.6, 5.7) містять перелік факторів, що сприяють і перешкоджають реалізації проекту на ринку.

Таблиця 5.6 – Фактори загроз

Фактор	Зміст загрози	Можлива реакція компанії
Збільшення кількості конкурентів	Поява нових конкурентів	Покращення рекламної компанії та гнучкі тарифи на продукт призведуть до більшої інформованості потенційних клієнтів та більш широкого спектру виходячи із гнучкої цінової політики
Зміни тенденцій ринку	Поява продуктів із новим функціоналом	Адаптація нового та покращення функціоналу, який запровадив конкурент, за умови успішності продукту конкуренту.
Економічний спад	Відсутність попиту на програмний продукт через економічну складову.	Дослідження ринку та виділення нової цільової аудиторії, можливо впровадження функціоналу для нової галузі.
Зниження репутації компанії	Можлива ситуація, коли продажі конкурентів та популярність на ринку почнуть превалювати.	Промо компанії із безкоштовними демо версіями продукту допоможуть залучити нових клієнтів.

Таблиця 5.7 – Фактори можливостей

Фактор	Зміст загрози	Можлива реакція компанії
Невелика конкуренція на ринку	На разі кількість продуктів що надають подібний функціонал не є великою. Існуючі аналоги не мають можливості адаптації до нових процесів	Використати стратегії для залучення клієнтів які ще не використовують жодних продуктів та клієнтів які користуються послугами конкурентів для встановлення монополії на ринку
Відповідні тенденції ринку	Кількість вищих навчальних закладів зростає, особливо приватних	Досліджувати ринок навчальних послуг та впроваджувати таргетовані рекламні кампанії

Наступний крок це аналіз конкуренції на ринку, тобто типу та інтенсивності можливої майбутньої конкуренції. Аналіз проводиться за конкурентним середовищем і протиставляє середовища та влив на дії компанії, який може мати таке середовище. За допомогою такого аналізу можна визначити основні підходи та середовище для розвитку продукту. Аналіз наведено нижче (табл. 5.8).

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	У чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії)
Тип конкуренції	Чиста	Покращення власного продукту

Таблиця 5.8 (продовження)

За рівнем конкурентної боротьби	Національна	Розповсюдження продукту на іноземному ринку за допомогою відповідних агентств
Конкуренція за видами товарів	Товарно-видова	Підкреслювати переваги та унікальність продукту
За характером конкурентних переваг	Нецінова	Надання функцій, які не надають конкуренти
За інтенсивністю	Марочна	Надання функцій, які не надають конкуренти

Далі відбувається більш детальний аналіз конкуренції у галузі за моделлю п’яти сил М. Портера, який має п’ять основних факторів, аналіз включає такі категорії як прямі конкуренти в галузі, потенційні конкуренти, постачальники, клієнти, товарозамінники. Аналіз робиться по вищезазначеним критеріям та надає можливість зробити висновки і таким чином оцінити конкурентоспроможність продукту у галузі. Результат аналізу наведено нижче (табл. 5.9).

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером

Складові галузі	Прямі конкуренти в галузі	Потенційні конкуренти	Клієнти	Товари-замінники
	НТУ "ХПІ"	Campus	Державні та приватні навчальні заклади	Google classroom, Google docs

Таблиця 5.9 (продовження)

Висновки	Інтенсивність конкуренції з боку прямих конкурентів є незначною, оскільки вони зосереджені на власних навчальних закладах.	Потенційні конкуренти не можуть запропонувати настільки універсальний продукт	Впровадження у НТУУ “КПІ” на безкоштовній основі, вихід на ринок після отриманого досвіду експлуатації	Лише часткове покриття без повноцінного досвіду керування кафедрою

Наступний крок – визначення та обґрунтування факторів конкурентоспроможності, що проводиться на основі аналізу конкуренції за Портером, характеристик ідеї проекту, характеристик споживачів, факторів загроз та можливостей. Результат аналізу зображено нижче (табл. 5.10).

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

Фактор конкурентоспроможності	Обґрунтування
Унікальний функціонал	Існуючі конкуренти не мають можливості налаштовувати керування бізнес процесами
Легкість і простота використання	Простий інтерфейс та відсутня потреба у спеціальних технічних знаннях
Підключення до мережі Інтернет	Потребує підключення до мережі інтернет

За визначеними факторами у таблиці 5.10 проведено аналіз сильних та слабких сторін стартап-проекту та наведено нижче (табл. 5.11).

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін проекту

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з розробленим проектом						
			-3	-2	-1	0	1	2	3
1	Ціна	13			+				
2	Орієнтованість на кінцевого користувача	15				+			
3	Обслуговування системи в перший рік після інтеграції в навчальному закладі/кафедрі	10			+				
4	Простота використання	12			+				
5	Масштабованість та гнучкість системи	16		+					

Наступний етап аналізу можливостей впровадження проекту є SWOT-аналіз. Виконується за допомогою побудови SWOT-матриці із перелічення сильних (S – strong) та слабких (W – weak) сторін проекту, можливостей (O – opportunities) і загроз (T – threats). Такий аналіз надає огляд продукту під різними кутами та відкриває можливість до планування.

Перелік ринкових загроз та ринкових можливостей складається на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Матрицю наведено у нижче (табл. 5.12).

Таблиця 5.12 – SWOT-аналіз проекту

Сильні сторони (S):	Слабкі сторони (W):
Налаштування бізнес процесів; Гнучка адміністрація доступів користувачів; Система документообігу; Механізм інформування користувачів;	Високий рівень свободи користувачів може призвести до непередбачуваної поведінки; Додаткове тестування на етапі розробки;

Таблиця 5.12 (продовження)

Можливості (О):	Загрози (Т):
Відсутність повноцінних альтернатив Вихід на нові ринки Розширення клієнтської бази	Поява нових конкурентів Зниження репутації компанії Новий функціонал від

Після SWOT-аналізу виконано аналіз альтернатив ринкової поведінки. Проаналізовано оптимальний час виходу проекту на ринок та його ринкової реалізації з урахуванням потенційних проектів конкурентів. Альтернативи впровадження проекту наведено нижче (табл. 5.13).

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Терміни реалізації
Орієнтація поточної моделі на ринок бухгалтерії	70%	3 місяці
Орієнтація поточної моделі на організацію промислових підприємств	15%	12 місяців
Орієнтація поточної моделі на державні структури	20%	6 місяців

Найбільш перспективною альтернативою є “Орієнтація поточної моделі на ринок бухгалтерії”. Існуючий функціонал дуже близький та може легко пристосуватися. Ймовірність отримання ресурсів 70%, що значно перевищує показники для інших альтернатив.

5.4 Розроблення ринкової стратегії проекту

При створенні ринкової стратегії необхідно визначити стратегію охоплення ринку. Проаналізувавши демографічні дані цих груп, творці вирішують, якій групі вони пропонуватимуть свій продукт, і створюють стратегію охоплення ринку. Цей процес передбачає визначення цільових груп потенційних споживачів, результати аналізу наведено нижче (табл. 5.14).

Таблиця 5.14 – Вибір цільових груп потенційних споживачів

Опис цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в сегменті	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
Державні навчальні заклади	Потребують	Попит є	Незначна	Просто
Приватні навчальні заклади	Потребують	Попит є	Незначна	Помірно

Оскільки компанія хоче почати заробляти гроші якнайшвидше та враховуючи, що бажаним результатом є масовий маркетинг, найкраще використовувати обидві цільові групи при створенні стандартизованої програми для обох груп – завдяки тому, що відмінності між двома групами незначні. Хоча більшу увагу слід приділити саме державним навчальним закладам, за дотримання правильної стратегії існує можливість отримати монополію на ринку та фінансування від держави. Слід також урахувати альтернативи розвитку як наприклад впровадження у інші галузі.

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку, яка визначається нижче (табл. 5.15).

Таблиця 5.15 – Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Орієнтація поточної моделі на ринок приватних навчальних закладів	Стратегія концентрованого маркетингу	Можливість налаштовувати гнучкі бізнес процеси для організації роботи кафедри та навчального процесу	Стратегія диференці-ації
Орієнтація поточної моделі ринок на державних навчальних закладів	Стратегія концентрованого маркетингу	Можливість налаштовувати гнучкі бізнес процеси для організації роботи кафедри та навчального процесу	Стратегія диференці ації

Перш ніж створити ринкову стратегію, компанія повинна створити план охоплення ринку, який окреслює потенційних споживачів.

Після виявлення потенційних споживачів було проведено аналіз маркетингового середовища. Це призвело до створення списку ринкових можливостей і загроз. Список створено на основі висновків щодо факторів загрози та факторів можливостей у маркетинговому середовищі.

Слід визначити чи існують інші продукти на ринку, які характеристики конкурентів можна використати, чи слід шукати нових користувачів або забирати користувачів у конкурентів а також стратегію конкурентної поведінки.

Отже наступним кроком є вибір стратегії конкурентної поведінки (табл 5.16).

Таблиця 5.16 – Визначення базової конкурентної поведінки

Чи є проект «першопроходьцем» на ринку	Ні
Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Так
Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Так, весь потенціально вигідний функціонал буде адаптовано та покращено для домінації продукту на ринку
Стратегія конкурентної поведінки	Стратегія заняття конкурентної ніші

Стартап-компанія повинна створити стратегію позиціонування на основі базової стратегії розвитку та конкурентної поведінки і потреб споживачів певних груп.

Ця стратегія позиціонування повинна включати товарний знак або дизайн, які споживачі повинні ідентифікувати, щоб створити унікальну для них позицію на ринку. Стратегія позиціонування наведена нижче (табл. 5.17).

Таблиця 5.17 – Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту

Таблиця 5.17 (продовження)

Легкість розуміння, зручний інтерфейс, кастомізація.	Стратегія диференціації	Легкість і простота у використанні. Вирішення важливих поставлених задач швидко	Гнучкість; Швидкість; Простота;

5.5 Розроблення маркетингової програми стартап-проекту

Було виконано попередній аналіз конкурентоспроможності продукту стартапу разом із формуванням його маркетингової концепції. Цей крок необхідний для розробки маркетингової програми кожного стартапу. Результати наведено нижче (табл. 5.18).

Таблиця 5.18 – Визначення ключових переваг концепції потенційного товару

Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
Зручність застосування	Не вимагається специфічних технічних знань	Інтуїтивний інтерфейс для автоматизації будь-яких процесів
Автоматизація процесів керування кафедрою та навчальним процесом	Можливість створювати та використовувати бізнес-процеси для автоматизації роботи; Мінімальна витрата часу від кожного учасника;	Синхронізація усіх учасників проекту

Концепції маркетингу майбутнього використовують трирівнева систему. Перший рівень включає ідею продукту або послуги, тоді як другий рівень включає його фізичні компоненти. Третій рівень обговорює конкретні процедури, які використовуються для надання продукту чи послуги.

1-й рівень Перший рівень абстракції має справу з пошуком проблем і потреб, які вирішуються продуктом. Ці запитання допомагають визначити, які аспекти проекту потрібно задокументувати. Це призводить до створення технічного завдання при створенні конструкторської документації на виріб.

2-й рівень Цей рівень деталізує передбачуваний спосіб продажу продукту; міркування включають дизайн, характеристики, ціну та упаковку.

3-й рівень Товари з додатковими послугами зазвичай є другорядними перевагами для споживача. Вони створюються шляхом створення продукту та способу його виконання. Це можна побачити в продуктах із посиленими матеріалами або функціями. Опис наведено нижче (табл 5.19).

Таблиця 5.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
Товар за задумом	Програмне забезпечення для автоматизації процесів функціонування кафедри
Товар у реальному виконанні	Властивості / характеристики
	Реалізовано ПЗ для оптимізації роботи кафедри на основі веб-додатку для автоматизації бізнес-процесів
Товар із підкріпленням	До продажу: стандартна розроблена система
	Після продажу: додані додаткові можливості та підтримка

Створення маркетингової моделі продукту має зосередитися на з’ясуванні того, як проект буде захищено від копіювання іншими людьми. Захищені методи захисту відрізняються від інтелектуальної власності, знань і атрибутів, що знаходяться на другому та третьому рівнях продукту.

Перш ніж сформувати ціну на потенційний продукт, споживачі повинні проаналізувати ціни на подібні продукти та рівень доходів своїх потенційних клієнтів. Нижче пояснено процес ціноутворення шляхом визначення цінових обмежень (табл. 5.20).

Таблиця 5.20 – Визначення меж встановлення ціни

Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни
Низький	Сердній	Низький

Слід визначити найкращу модель продажів для подальшого використання. Потрібно вирішити, чи використовувати зовнішні чи власні канали продажів. Далі потрібно вибрати оптимальну глибину каналу, вирішити, яку підкатегорію третіх сторін слід залучити, і визначитися з конкретним типом третьої сторони, яку вони повинні використовувати у своїй системі. Інформацію про формування системи збуту наведено нижче (табл. 5.21).

Таблиця 5.21 – Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Можливість використання демо версії продукту без оплати	Легкість використання, доступність у хмарі	Постачальник - Користувач.	Збут силами посередника

Розробка концепції маркетингових комунікацій – останній крок у створенні маркетингової програми. Цей процес передбачає розробку ідеї маркетингових комунікацій як описано нижче (табл. 5.22).

Таблиця 5.22 – Концепція маркетингових комунікацій

Поведінка цільових клієнтів	Канали комунікацій цільових клієнтів	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення
Клієнт отримує інформацію про нові продукти з реклами в інтернеті, соціальних мереж	Корпоративна електронна пошта, служба підтримки в телефонному режимі та чаті	Унікальний гнучкий функціонал за доступними цінами	Простота використання продукту для вирішення повсякденних питань

Результат аналізу – маркетингова програму, яка поєднує концепції продукту, продажі, просування та аналіз ціноутворення. Це базується на потребах і цінностях потенційних клієнтів, конкурентних перевагах ідеї та поточному стані ринку. Крім того, це залежить від того, чи відбувається реалізація проекту в динамічному ринковому середовищі чи середовищі з усталеними цінностями.

Отже, проаналізувавши можливі аспекти впровадження стартап-проекту та розглянувши конкуренцію на ринку, фактори ризику, переваги, цільову аудиторію можна зробити висновок, що в даного стартап-проекту є потенціал на комерціалізацію.

Висновки до розділу 5

У розділі розглянуто стартап проект. Його основними моментами були опис ідеї проекту, аналіз ринкових можливостей для запуска проекту, технологічний аудит проекту та розробка маркетингової стратегії проекту. Додатково розглядається розробка ринкової стратегії проекту та маркетингових ідей. Остаточний старт-ап проект готовий до презентації.

ВИСНОВКИ

У ході роботи було розроблено інформаційну систему для автоматизації роботи кафедри. Система значно зменшує кількість ресурсів, що витрачаються на організацію роботи та навчального процесу. Система надає можливість налаштовувати бізнес процеси, зберігати файли, відправляти email повідомлення а також зберігати інформацію про організаційну структуру навчального закладу та учасників процесів.

Проведено огляд існуючих підходів, що використовуються при автоматизації роботи підприємств та навчальних закладів. Серед найпоширеніших підходів, таких як CRM та ERP моделі, оптимального варіанту для вирішення специфічних для домену освіти проблем не знайдено. Як можливість задовольнити потреби у функціоналі будо впроваджено систему на основі моделі Low code/no code. Low code/no code це модель, що допомагає налаштовувати програмне забезпечення у відповідності до потреб, без змін у коді системі чи потреби залучення технічного спеціаліста – як результат значно збільшує гнучкість організації освітнього закладу та не потребує додаткових інвестицій.

Підхід low code/no code раніше не використовувався у сфері освіти і продукт є єдиним серед конкурентів на ринку та інновацією у своєму домені.

Для розробки було обрано мікросервісний архітектурний стиль який, за допомогою слабкої зв’язності між компонентами, надає великий потенціал для впровадження механізмів відмовостійкості системи, масштабування для обробки підвищеного навантаження, інтеграції зі сторонніми сервісами та можливість використанні технологій на будь-якій мові програмування.

Було проведено маркетинговий аналіз для розробки стартап проекту, потенціал впровадження системи на ринку є дуже високим завдяки функціоналу який на разі не доступний у конкурентів на ринку. Також кількість потенційних клієнтів зростає у зв’язку з відкриттям нових навчальних закладів, як на державні основі так і приватних.

Проаналізувавши можливі аспекти впровадження стартап-проекту та розглянувши конкуренцію на ринку, фактори ризику, переваги, цільову аудиторію можна зробити висновок, що в даного стартап-проекту є потенціал на комерціалізацію.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems: O'Reilly Media, 1st edition, 2017. 457 c.
2. Design Patterns: Elements of Reusable Object-Oriented Software / Gamma Erich, Richard Helm, Ralph Johnson, John Vlissides: Addison-Wesley Professional, 1st edition, 1994. 416 c.
3. Robert M. Clean Code: A Handbook of Agile Software Craftsmanship: Pearson, 1st edition, 2008, 464c.
4. Refactoring: Improving the Design of Existing Code / Fowler Martin, Kent Beck, John Brant, William Opdyke, Don Roberts: Addison-Wesley Professional, 1st edition, 1999, 431c.
5. Seemann M., Steven van D. Dependency Injection Principles, Practices, and Patterns: Manning, 1st edition, 2019, 552c.
6. Osherove R. The Art of Unit Testing: with examples in Java: Manning, 2nd edition, 2013, 296c.
7. Beck K. Test Driven Development: By Example: Addison-Wesley Professional, 1st edition, 2002, 240c.
8. Feathers M. Working Effectively with Legacy Code (Robert C. Martin Series): Pearson, 1st edition, 2004, 458c.
9. Fowler M. Patterns of Enterprise Application Architecture (Addison-Wesley Signature Series (Fowler)): Addison-Wesley Professional, 1st edition, 2012, 558c.
10. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software: Addison-Wesley Professional, 1st edition, 2003, 563c.
11. Vaughn V. Implementing Domain-Driven Design: Addison-Wesley Professional, 1st edition, 2013, 656c.

12. Robert M. Clean Architecture: A Craftsman's Guide to Software Structure and Design (Robert C. Martin Series): Pearson, 1st edition, 2017, 430c.
13. Buschmann F. Pattern-Oriented Software Architecture Volume 1: A System of Patterns: Wiley, 1st edition, 1996, 476c.
14. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development: Pearson, 3rd edition, 2004, 736c.
15. Прилипко Н.О. Вдосконалення системи електронного документообігу в органах державної влади // Н.О. Прилипко: зб. наук. пр. Донецького державного університету управління. Серія: Державне управління. 2014, С. 155-164.
16. Асєєв Г.Г. Управління сучасним документообігом: теорія, структура, методи // Г.Г. Асєєв: Вісник Книжкової палати. 2004, С. 17-19.
17. Проектування інформаційних систем. / Пономаренко В.С., Пушкар О.І., Журавльова І.В., Мінухін С.В.: 2002.
18. ActiveMQ documentation. URL: https://www.rabbitmq.com/documentation.html. (дата звернення: 10.10.2022)
19. Fowler M. Microservices/Martin. URL: http://martinfowler.com/articles/microservices.html. (дата звернення: 09.25.2022)
20. Docker overview. URL: https://docs.docker.com/get-started/overview/ (дата звернення: 10.25.2022)

ДОДАТОК А

Лістинг

Моделі та методи розробки Back End для задач створення інформаційного порталу кафедри

УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ ТВ-з11мп

Аркушів 12

2022

```
@Component
@RequiredArgsConstructor
public class JwtUtil {

    private final JwtProperties jwtProperties;

    private Algorithm algorithm;

    @PostConstruct
    void setAlgorithm() {
        this.algorithm = Algorithm.HMAC256(jwtProperties.getSecret());
    }

    public String generateAccessToken(Account account) {
        Function<Role, String> roleMapper = role -> "ROLE_" + role.getName();
        return JWT.create()
                .withSubject(account.getUsername())
                .withIssuedAt(Instant.now())
               .withExpiresAt(Instant.now().plusMillis(jwtProperties.getAccessToken
Expiry()))
                .withClaim("scp",
account.getRoles().stream().map(roleMapper).collect(Collectors.toList()))
                .withClaim("account_id", account.getId())
                .withClaim("user_id",
Optional.ofNullable(account.getUser()).map(AppUser::getId).orElse(null))
                .sign(algorithm);
    }

    public String generateRefreshToken(Account account) {
        return JWT.create()
                .withSubject(account.getUsername())
                .withIssuedAt(Instant.now())
                .withExpiresAt(Instant.now().plusMillis(jwtProperties.getRefreshTo
kenExpiry()))
                .sign(algorithm);
    }

    public Authentication getAuthenticationFromRawToken(String rawToken) {
        DecodedJWT decodedJWT = validateAndParseToken(rawToken);

        String username = decodedJWT.getSubject();
        Long userId = decodedJWT.getClaim("user_id").asLong();
        Long accountId = decodedJWT.getClaim("account_id").asLong();
        Collection<SimpleGrantedAuthority> authorities =
decodedJWT.getClaim("scp").asList(String.class)
                .stream()
                .map(SimpleGrantedAuthority::new)
                .collect(Collectors.toSet());

        UserPrincipal userPrincipal = new UserPrincipal(accountId, userId,
username);
        return new UsernamePasswordAuthenticationToken(userPrincipal, null,
authorities);
    }

    public DecodedJWT validateAndParseToken(String token) {
        return JWT.require(algorithm).build().verify(token);
    }

}
```

```
@Slf4j
@RequiredArgsConstructor
public class JwtAuthorizationFilter extends OncePerRequestFilter {


   private final static String BEARER_SCHEMA = "Bearer ";


   private final ObjectMapper objectMapper;
   private final JwtUtil jwtUtil;
   private final Set<String> whitelistEndpoints;


   @Override
   protected boolean shouldNotFilter(HttpServletRequest request) throws
ServletException {
       return whitelistEndpoints != null &&
whitelistEndpoints.contains(request.getServletPath());
   }


   @Override
   protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
response, FilterChain filterChain) throws ServletException, IOException {
       String jwt = resolveToken(request);
       if (StringUtils.hasText(jwt)) {
           try {
               Authentication authentication =
jwtUtil.getAuthenticationFromRawToken(jwt);
               SecurityContextHolder.getContext().setAuthentication(authentication
);
           } catch (JWTVerificationException exception) {
               response.setStatus(UNAUTHORIZED.value());
               response.setContentType(APPLICATION_JSON_VALUE);
               objectMapper.writeValue(response.getOutputStream(),
Map.of("error_message", exception.getMessage()));
           }
       }
       filterChain.doFilter(request, response);
   }
```

```
   private String resolveToken(HttpServletRequest request) {
       String authorizationHeader = request.getHeader(AUTHORIZATION);
       if (StringUtils.hasText(authorizationHeader) &&
authorizationHeader.startsWith(BEARER_SCHEMA)) {
           return authorizationHeader.substring(BEARER_SCHEMA.length());
       }
       return null;
   }
}


@AllArgsConstructor
public class UserDetailsAdapter implements UserDetails {


    private Account account;


    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return account.getRoles().stream()
                .map(Role::getName)
                .map(SimpleGrantedAuthority::new)
                .collect(Collectors.toSet());
    }


    @Override
    public String getPassword() {
        return account.getPassword();
    }


    @Override
    public String getUsername() {
        return account.getUsername();
    }


    @Override
    public boolean isAccountNonExpired() {
```

```
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return account.getActive();
    }
}

@Data
@AllArgsConstructor
public class UserPrincipal implements Principal {

    private Long accountId;
    private Long appUserId;
    private String name;

}

@Configuration
@EnableJpaAuditing(auditorAwareRef = "auditorProvider")
@EnableJpaRepositories(basePackages = "com.masterswork.account.repository")
public class PersistenceContextConfig {
    @Bean
```

```
    public AuditorAware<String> auditorProvider() {
        return () ->
Optional.ofNullable(SecurityContextHolder.getContext().getAuthentication())
                        .map(Principal::getName)
                        .or(() -> Optional.of("system"));
    }
}
@Configuration
@AutoConfigureOrder(Ordered.HIGHEST_PRECEDENCE)
public class RestExceptionHandlerConfig {


    @Bean
    public RestExceptionHandler<Exception> messageSourceExceptionHandler(
            MessageSource messageSource, SpelMessageInterpolator
spelMessageInterpolator) {
        return new MessageSourceExceptionHandler<>(messageSource,
spelMessageInterpolator);
    }


    @Bean
    public ConstraintViolationExceptionHandler
constraintViolationExceptionHandler(
            MessageSource messageSource, SpelMessageInterpolator
spelMessageInterpolator) {
        return new ConstraintViolationExceptionHandler(messageSource,
spelMessageInterpolator);
    }


    @Bean
    public MethodArgumentNotValidExceptionHandler
methodArgumentNotValidExceptionHandler(
            MessageSource messageSource, SpelMessageInterpolator
spelMessageInterpolator) {
        return new MethodArgumentNotValidExceptionHandler(messageSource,
spelMessageInterpolator);
    }


    @Bean
    public MessageSource messageSource() {
```

```
        ReloadableResourceBundleMessageSource messageSource = new
ReloadableResourceBundleMessageSource();

        messageSource.setBasenames("classpath:default-exception-messages",
"classpath:jwt-exception-messages",

                "classpath:spring-exception-messages", "classpath:exception-
messages");

        messageSource.setDefaultEncoding(StandardCharsets.UTF_8.name());

        return messageSource;

    }



    @Bean

    public SpelMessageInterpolator spelMessageInterpolator() {

        return new SpelMessageInterpolator();

    }



    @Bean

    public RestExceptionHandlerResolver<Exception> restExceptionHandlerResolver(

            RestExceptionHandler<Exception> messageSourceExceptionHandler,

            ConstraintViolationExceptionHandler
constraintViolationExceptionHandler,

            MethodArgumentNotValidExceptionHandler
methodArgumentNotValidExceptionHandler) {



        RestExceptionHandlerResolver<Exception> restExceptionHandlerResolver = new
RestExceptionHandlerResolver<>();

        restExceptionHandlerResolver.addHandler(MethodArgumentNotValidException.cl
ass,

                methodArgumentNotValidExceptionHandler);

        restExceptionHandlerResolver.addHandler(ConstraintViolationException.class
,

                constraintViolationExceptionHandler);

        restExceptionHandlerResolver.addHandler(Exception.class,
messageSourceExceptionHandler);

        return restExceptionHandlerResolver;

    }

}



@Configuration

@EnableGlobalMethodSecurity(prePostEnabled = true)
```

```
public class SecurityConfig {


    private final static String[] swaggerEndpoints = {
            "/v3/api-docs/**",
            "/configuration/ui",
            "/swagger-resources/**",
            "/configuration/security",
            "/swagger-ui/**",
            "/swagger-ui.html/**"
    };


    private final static String loginEndpoint = "/login";
    private final static String signUpEndpoint = "/signup";
    private final static String refreshTokenEndpoint = "/token-refresh";


    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http, JwtUtil jwtUtil,
ObjectMapper objectMapper) throws Exception {
        http.csrf().disable()
                .cors().configurationSource(corsConfigurationSource())
                .and()
                .httpBasic().disable()
                .sessionManagement((session) ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authorizeHttpRequests((authorize) -> authorize
                        .antMatchers(loginEndpoint, signUpEndpoint,
refreshTokenEndpoint).permitAll()
                        .antMatchers(swaggerEndpoints).permitAll()
                        .anyRequest().authenticated()
                )
                .exceptionHandling()
                .authenticationEntryPoint((request, response, authException) -> {
                    response.setStatus(UNAUTHORIZED.value());
                    response.setContentType(APPLICATION_JSON_VALUE);
                    objectMapper.writeValue(response.getOutputStream(),
Map.of("error_message", "No access token provided"));
                });
```

```
        JwtAuthorizationFilter jwtAuthorizationFilter = new
JwtAuthorizationFilter(objectMapper, jwtUtil, getAllWhitelistEndpoints());

        http.addFilterBefore(jwtAuthorizationFilter,
UsernamePasswordAuthenticationFilter.class);



        return http.build();

    }



    @Bean

    public RoleHierarchy roleHierarchy() {

        RoleHierarchyImpl roleHierarchy = new RoleHierarchyImpl();

        String hierarchy = "ROLE_ADMIN > ROLE_USER";

        roleHierarchy.setHierarchy(hierarchy);

        return roleHierarchy;

    }



    @Bean

    public AuthenticationManager authenticationManager(AuthenticationConfiguration
authenticationConfiguration) throws Exception {

        return authenticationConfiguration.getAuthenticationManager();

    }



    @Bean

    public UserDetailsService userDetailsService(AccountRepository userRepository)
{

        return username -> userRepository.findFirstByUsername(username)

                .map(UserDetailsAdapter::new)

                .orElseThrow(() -> new UsernameNotFoundException(username));

    }



    @Bean

    public PasswordEncoder passwordEncoder() {

        return new BCryptPasswordEncoder();

    }
```

```
    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        CorsConfiguration corsConfiguration = new
CorsConfiguration().applyPermitDefaultValues();
        corsConfiguration.setAllowedMethods(List.of("HEAD", "GET", "POST", "PUT",
"DELETE", "PATCH"));
        source.registerCorsConfiguration("/**", corsConfiguration);
        return source;
    }

    private Set<String> getAllWhitelistEndpoints() {
        Set<String> whitelist = new HashSet<>(Arrays.asList(swaggerEndpoints));
        whitelist.add(loginEndpoint);
        whitelist.add(signUpEndpoint);
        whitelist.add(refreshTokenEndpoint);
        return whitelist;
    }
}

@Configuration
public class SwaggerConfig {
    private final static String JWT_AUTHORIZATION_NAME = "bearer token";

    @Bean
    public OpenAPI api() {
        return new OpenAPI()
                .addSecurityItem(new
SecurityRequirement().addList(JWT_AUTHORIZATION_NAME))
                .components(new Components()
                    .addSecuritySchemes(JWT_AUTHORIZATION_NAME, new
SecurityScheme()
                            .name(JWT_AUTHORIZATION_NAME)
                            .type(SecurityScheme.Type.HTTP)
                            .scheme("bearer")
                            .bearerFormat("JWT")
```

```
                    ));
    }
}


@RestController
@RequestMapping("/accounts")
@RequiredArgsConstructor
public class AccountController {


    private final AccountService accountService;
    private final AppUserService appUserService;
    private final RoleService roleService;
    @Operation(summary = "Create new appUser and assign to account")
    @PreAuthorize("hasRole('ADMIN') or #accountId ==
authentication.principal.accountId")
    @PostMapping(path = "/{accountId}/app-users", consumes = "application/json",
produces = "application/json")
    public ResponseEntity<AppUserResponseDTO>
createAppUserForAccount(@PathVariable Long accountId, @Valid @RequestBody
AppUserCreateDTO body) {
        return ResponseEntity.ok(appUserService.createAppUserForAccount(accountId,
body));}
    @Operation(summary = "Update account by accountId")
    @PreAuthorize("hasRole('ADMIN')")
    @PutMapping(path = "/{accountId}", consumes = "application/json", produces =
"application/json")
    public ResponseEntity<AccountResponseDTO> updateAccount(@PathVariable Long
accountId, @Valid @RequestBody AccountUpdateDTO body) {
        return ResponseEntity.ok(accountService.updateAccount(accountId, body)); }
    @Operation(summary = "Patch account by accountId")
    @PreAuthorize("hasRole('ADMIN')")
    @PatchMapping(path = "/{accountId}", consumes = "application/json", produces =
"application/json")
    public ResponseEntity<AccountResponseDTO> patchAccount(@PathVariable Long
accountId, @RequestBody AccountUpdateDTO body) {return
ResponseEntity.ok(accountService.patchAccount(accountId, body));}
    @Operation(summary = "Add role to account")
    @PreAuthorize("hasRole('ADMIN')")
    @PutMapping(path = "/{accountId}/roles/{roleId}", produces =
"application/json")
```

```
    public ResponseEntity<AccountResponseDTO> addRole(@PathVariable Long
accountId, @PathVariable Long roleId) {

        return ResponseEntity.ok(accountService.addRoleToAccount(accountId,
roleId));

    }

    @Operation(summary = "Remove role from account")

    @PreAuthorize("hasRole('ADMIN')")

    @DeleteMapping(path = "/{accountId}/roles/{roleId}", produces =
"application/json")

    public ResponseEntity<AccountResponseDTO> removeRole(@PathVariable Long
accountId, @PathVariable Long roleId) {

        return ResponseEntity.ok(accountService.removeRoleFromAccount(accountId,
roleId));

    }

    @Operation(summary = "Assign appUser to account")

    @PreAuthorize("hasRole('ADMIN')")

    @PutMapping(path = "/{accountId}/app-users/{appUserId}", produces =
"application/json")

    public ResponseEntity<AccountResponseDTO> addAppUser(@PathVariable Long
accountId, @PathVariable Long appUserId) {

        return ResponseEntity.ok(accountService.addAppUserToAccount(accountId,
appUserId));

    }

    @Operation(summary = "Unassign appUser from account")

    @PreAuthorize("hasRole('ADMIN')")

    @DeleteMapping(path = "/{accountId}/app-users", produces = "application/json")

    public ResponseEntity<AccountResponseDTO> unassignAppUser(@PathVariable Long
accountId) {

        return
ResponseEntity.ok(accountService.unassignAppUserFromAccount(accountId));

    }

    @Operation(summary = "Get all accounts")

    @PreAuthorize("hasRole('ADMIN')")

    @GetMapping(produces = "application/json")

    public ResponseEntity<Page<AccountResponseDTO>> getAllAccounts(

            @ParameterObject @PageableDefault(sort = "id") Pageable pageable) {

        return ResponseEntity.ok(accountService.getAllAccounts(pageable));

    }

    @Operation(summary = "Get account of the current user")

    @PreAuthorize("hasRole('USER')")
```

```
    @GetMapping(path = "/current", produces = "application/json")
    public ResponseEntity<AccountResponseDTO> getMyAccount(Authentication
authentication) {
        return
ResponseEntity.ok(accountService.getAccountByUsername(authentication.getName()));
    }
    @Operation(summary = "Get account by username")
    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping(path = "/username/{username}", produces = "application/json")
    public ResponseEntity<AccountResponseDTO> getAccountByUsername(@PathVariable
String username) {
        return ResponseEntity.ok(accountService.getAccountByUsername(username));
    }
    @Operation(summary = "Get account by accountId")
    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping(path = "/{accountId}", produces = "application/json")
    public ResponseEntity<AccountResponseDTO> getAccountById(@PathVariable Long
accountId) {
        return ResponseEntity.ok(accountService.getAccountById(accountId));
    }
    @Operation(summary = "Get roles assigned to account by accountId")
    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping(path = "/{accountId}/roles", produces = "application/json")
    public ResponseEntity<Page<RoleResponseDTO>> getAllRolesForAccount(
            @PathVariable Long accountId, @ParameterObject @PageableDefault(sort =
"id") Pageable pageable) {
        return ResponseEntity.ok(roleService.getAllRolesByAccountId(accountId,
pageable));
    }
    @Operation(summary = "Get roles assigned to account by username")
    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping(path = "/username/{username}/roles", produces =
"application/json")
    public ResponseEntity<Page<RoleResponseDTO>> getAllRolesForAccountByUsername(
            @PathVariable String username, @ParameterObject @PageableDefault(sort
= "id") Pageable pageable) {
        return ResponseEntity.ok(roleService.getAllRolesByUsername(username,
pageable));
    }
}
```