UNICHECK

Ім'я користувача:
Недашківський Олексій Леонідович

ID перевірки:
1013220443

Дата перевірки:
06.12.2022 21:27:57 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
07.12.2022 15:26:17 EET

ID користувача:
100004440

Назва документа: **MishynAA_TVz11MP_magistr_2022**

Кількість сторінок: **62** Кількість слів: **10703** Кількість символів: **89529** Розмір файлу: **2.06 MB** ID файлу: **1012965767**

0%
Схожість

Збіги відсутні

0% Цитат

Не знайдено жодних цитат

Вилучення списку бібліографічних посилань вимкнене

0.36%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за **10 слів** та **10**...

Немає вилучених Інтернет-джерел

0.36% Вилученого тексту з Бібліотеки **1** ..... Сторінка 64

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_____

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

Магістерська дисертація

За освітньою програмою «Інженерія програмного 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 Недашківський О. Л._____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент_____ _____
(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 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; дослідіть 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

Робота містить 82 сторінки, 27 рисунків, 22 таблиці та 18 посилань.

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 безпосередньої 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.

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

ABSTRACT

The work contains 82 pages, 27 pictures, 22 tables and 18 links.

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.

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 викликі 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 важливою 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 актуальною потребою. Впровадження системи вирішить ряд проблем: покращить процес взаємодії 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

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 фрейсворку 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.

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.

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 малюнку 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 які 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.

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 наюанси 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 зпростити 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 сервер 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 зраучні у використанні бібліотеки.

Висновки до розділу

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 архітетуро з використанням REST API. Основні засрби розробки мова Java та фреймворк Spring Boot.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Огляд архітектури додатку

Перед початком розробки було розроблено проект архітектури для побудовисистем. 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 сисетмі, від публічних – які будуть використані кілєнтами 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 рахуно створення копій, це зумовлено відсутністю стану шлюзу;

2. Підвищена безбека. Шлюз робить аутентифікацію запитів користувача і відкриває лише частину доступного API із системи, завдяки чому значно зменшує кількість можливих варінтів 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 занчного 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 такий 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, відкладуючи обробку поміщених подій. Таким чином, ActiveMQ ідеально підходить для тривалих завдань чи при лімітії 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.

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 впровадженні 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;

– 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 у 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

Інформації, що зберігається у базі даних зосереджена навколо таких сутностей як обліковий запис користувача, його ролі, контактну інформацію та відношення до кфедр та факультетів. Схему бази даних наведено на рисунку 3.5.

Рисунок 3.5 – Схема бази даних Account сервісу

3.2.3 Огляд API Account сервісу

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

Рисунок 3.6 – Swagger документація API для створення облкового 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

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 файлів, 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 сторонніс файловим сховищем для надання додаткового функціоналу: контрою доступу для окреми 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, безпецкі 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 зберігатютья 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 частинох світу.

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

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

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 делігувати 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

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 повдомлень 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. Таким чином, все, що може бути змодельовано як дерево DOM, може бути ефективно оброблено як шаблон Thymeleaf.

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 згенерованого повідомлення слід використати текстові шаблони. Коли повідомлення відправляється автоматично можна використовувати шаблони на основі HTML.

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

3.4.3 Огляд API Mail сервісу

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

3.5 Process сервіс

Process сервіс відповідає за збереження схем процесів, запуск процесів та контроль їх виконання на усіх етапах, відповідними учасниками. Схеми процесів зберігаться 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 документно-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 заздалегіть. 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 знахоиться у MonogoDB та зберігає лише одну колекцію зі схемами бізнес процесів. Приклад документу зі схемою бізне процесу зображено на рисунку 3.17.

Рисунок 3.17 – Приклад схеми бізнес процесу збереженого у MongoDB

Інша частина даних зберігається безпосереньо у реляційній базі даних так як має структуру. Сутності які зберігаються у реляційній базі даних це записи

запущених бізнес процесів та коментарі до деяких кроків бізнес процесу. Схему бази даних наведено на рисунку 3.18.

Рисунок 3.18 – Схема бази даних Process сервісу

3.5.3 Огляд API Process сервісу

API Process сервісу передбачає створення схем та запуск процесів. Приклад створення бізнес процесу, за схемою на рисунку 3.19, зображено на рисунку 3.20.

Рисунок 3.20 – Схема бізнес процесу складання предмету

Рисунок 3.20 – Приклад запиту для створення

Як можна побачити бізнес процес представлений JSON об’єктом зі значеннями ключ-значення. Кожне значення має колекцію вказівників на наступні кроки залежно від результату виконання поточного етапу.

Висновки до розділу

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 бази 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 тесутвання 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 умовах 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

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

ВИСНОВКИ

У ході роботи було розроблено інформаційну систему для автоматизації роботи кафедри. Система значно зменшує кількість ресурів, 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 відпоівності до потреб, без змін у коді системі чи потреби залучення тезнічного спеціаліста – як результат значно збільшує гнучкість організації освтнього 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 св'язності між компонентами, надає великий потенціал для впровадження механізмів відмовостійкості системи, масштабування для обробки підвищенного 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 вілкриттям нових навчальних закладів, як на державні основі так і приватнихю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Thinking in Java 4th edition / Bruce Eckel / Pearson Education (US) 2006.
2. Spring Boot [Електронний ресурс] – Режим доступу до ресурсу (дата звернення: 11.05.2021): https://spring.io/projects/spring-boot
3. Spring Security [Електронний ресурс] – Режим доступу до ресурсу (дата звернення: 15.05.2021): https://spring.io/projects/spring-security
4. Spring Data JPA [Електронний ресурс] – Режим доступу до ресурсу (дата звернення: 8.05.2021): https://spring.io/projects/spring-data-jpa
5. Spring Web [Електронний ресурс] – Режим доступу до ресурсу (дата звернення: 9.05.2021): https://easyjava.ru/spring/spring-web-mvc/
6. Spring Cloud [Електронний ресурс] – Режим доступу до ресурсу (дата звернення: 16.05.2021): https://spring.io/projects/spring-cloud
7. FeignClient [Електронний ресурс] – Режим доступу до ресурсу (дата звернення:16.05.2021):https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-feign.html
8. MapStruct [Електронний ресурс] – Режим доступу до ресурсу (дата звернення: 8.05.2021): https://mapstruct.org/documentation/reference-guide/
9. Effective Java 3rd edition / Joshua Bloch / Addison Wesley Professional 2017.
10. Maven: The Definitive Guide 2nd edition / Brian Jackson / O'Reilly Media 2015.
11. Swagger [Електронний ресурс] – Режим доступу до ресурсу (дата звернення: 05.05.2021): https://swagger.io/docs/
12. Garofolo E. Practical Microservices: Build Event-Driven Architectures with Event Sourcing and CQRS / Ethan Garofolo — Pragmatic Bookshelf — 1st edition, 2020 — 292c.
13. Khan A. Developing Microservices Architecture on Microsoft Azure with Open Source Technologies (IT Best Practices - Microsoft Press) / Ovais Mehboob Ahmed Khan, Arvind Chandaka - Microsoft Press – 1st edition, 2020 – 304c.
14. Newman S. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith / Sam Newman — O'Reilly Media — 1st edition, 2019 — 272c.

15. Matsinopoulos P. Practical Test Automation: Learn to Use Jasmine, RSpec, and Cucumber Effectively for Your TDD and BDD / Panos Matsinopoulos — Apress — 1st edition, 2020 — 334c.
16. Vitillo R. Understanding Distributed Systems: What every developer should know about large distributed applications / Roberto Vitillo — Apress — 1st edition, 2021 — 253c.
17. Petrov A. Database Internals: A Deep Dive into How Distributed Data Systems Work / Alex Petrov — O'Reilly Media — 1st edition, 2019 — 376c.
18. Adkins H. Building Secure and Reliable Systems: Best Practices for Designing, Implementing, and Maintaining Systems / Heather Adkins, Betsy Beyer, Paul Blankinship, Piotr Lewandowski, Ana Oprea, Adam Stubblefield - O'Reilly Media – 1st edition, 2020 – 558c.

Вилучення

Вилучення по Бібліотеці акаунту	1

Студентська робота	ID **файлу:** 1008971258	**Навчальний заклад:** National Technical University of Ukraine "Kyiv Po...	0.36%