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

Курсова робота

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

РОЗРОБКА МІКРОСЕРВІСУ ДЛЯ КЕРУВАННЯ ОРГАНАМИ СТУДЕНТСЬКОГО САМОВРЯДУВАННЯ КНУ ІМЕНІ ТАРАСА ШЕВЧЕНКА У РАМКАХ WEB-ПЛАТФОРМИ

Виконав студент 3-го курсу
ОПП «Програмна інженерія»
Вербицький Артем Віталійович _____
(підпис)

Науковий керівник:
асистент кафедри інтелектуальних програмних систем,
кандидат фіз.-мат. наук
Жереб Костянтин Анатолійович _____
(підпис)

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

Студент _____
(підпис)

КИЇВ 2025

РЕФЕРАТ

Обсяг роботи 33 сторінки, 3 ілюстрації, 2 таблиці, 21 джерело посилання.

API, DOCKER, NESTJS, POSTGRESQL, TYPESCRIPT, WEB-ПЛАТФОРМА, МІКРОСЕРВІСНА АРХІТЕКТУРА, РЕЛЯЦІЙНА БАЗА ДАНИХ, СТУДЕНТСЬКЕ САМОВРЯДУВАННЯ

Об'єктом роботи є процес автоматизації діяльності органів студентського самоврядування КНУ імені Тараса Шевченка. Предметом роботи є серверна (backend) частина Web-платформи (oss-сервіс) для органів студентського самоврядування.

Метою курсової роботи є забезпечення функціонування органів студентського самоврядування КНУ імені Тараса Шевченка та інформування студентів щодо оних за допомогою розробки високопродуктивної та масштабованої backend-частини Web-платформи.

Методи розроблення: об'єктно-орієнтоване програмування, мікросервісна архітектура, REST API, асинхронна взаємодія між сервісами. Інструменти розроблення: TypeScript, Nest.js, Sequelize, PostgreSQL, Docker.

Результати роботи: спроєктовано та реалізовано частину oss-сервісу Web-платформи для органів студентського самоврядування, що включає REST API для управління базою даних (БД) ОСС.

За методами розробки та інструментальними засобами робота виконувалася з урахуванням сучасних практик розробки Web-додатків.

Розроблений oss-сервіс може бути інтегрований з іншими компонентами Web-платформи та застосований для автоматизації роботи органів студентського самоврядування КНУ імені Тараса Шевченка.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API – Application Programming Interface, прикладний програмний інтерфейс;

CRUD – Create, Read, Update, Delete, операції створення, читання, оновлення, видалення;

DTO – Data Transfer Object, об'єкт передачі даних;

gRPC – Google Remote Procedure Call, система віддаленого виклику процедур від Google;

JSON – JavaScript Object Notation, текстовий формат обміну даними;

ORM – Object-Relational Mapping, об'єктно-реляційне відображення;

REST – Representational State Transfer, передача репрезентативного стану;

SQL – Structured Query Language, мова структурованих запитів;

TCP – Transmission Control Protocol, протокол керування передачею;

БД – база даних;

ОСС – органи студентського самоврядування.

ВСТУП

Оцінка сучасного стану об'єкта розробки. Органи студентського самоврядування є важливою складовою університетського життя, що забезпечує представництво та захист прав та інтересів студентів, їх участь в управлінні вищим навчальним закладом. У сучасних умовах розвитку інформаційного суспільства ефективність роботи ОСС значною мірою залежить від рівня інформатизації їхньої діяльності.

На даний момент у КНУ імені Тараса Шевченка відсутнє єдине інформаційне середовище, яке б забезпечувало ефективну комунікацію між органами студентського самоврядування та студентами, що ускладнює процеси організації заходів, інформування, збору та обробки звернень студентів.

Актуальність роботи та підстави для її виконання. Створення Web-платформи для органів студентського самоврядування КНУ імені Тараса Шевченка є актуальним завданням, оскільки дозволить оптимізувати процеси взаємодії студентів з ОСС, підвищити прозорість діяльності органів самоврядування, спростити процеси організації заходів та збору зворотного зв'язку.

В умовах зростаючої діджиталізації освітнього процесу та студентського життя, розробка сучасної Web-платформи з використанням передових технологій є необхідним кроком для забезпечення ефективної роботи ОСС.

Мета й завдання роботи. Метою курсової роботи є забезпечення функціонування органів студентського самоврядування КНУ імені Тараса Шевченка та інформування студентів щодо оних за допомогою розробки високопродуктивної та масштабованої backend-частини Web-платформи.

Для досягнення цієї мети поставлено такі завдання:

— розробити архітектуру backend-частини Web-платформи з використанням мікросервісного підходу;

— спроєктувати та реалізувати API для взаємодії з frontend-частиною;

— реалізувати механізми асинхронної взаємодії між сервісами;

— забезпечити контейнеризацію розробленого рішення за допомогою Docker.

Об'єкт, методи й засоби розроблення. Об'єктом роботи є процес автоматизації діяльності органів студентського самоврядування КНУ імені Тараса Шевченка. Предметом роботи є серверна (backend) частина Web-платформи (oss-сервіс) для органів студентського самоврядування.

Методи розроблення включають:

— Об'єктно-орієнтоване програмування для структурування коду та забезпечення його модульності;

— Мікросервісну архітектуру для розділення функціональності на незалежні сервіси;

— REST API для забезпечення комунікації між клієнтською та серверною частинами;

— Асинхронну взаємодію між сервісами для підвищення продуктивності системи.

Технологічний стек проєкту включає:

— TypeScript як основну мову програмування для забезпечення типобезпеки та кращої підтримки коду;

— Nest.js як фреймворк для розробки серверної частини, що забезпечує модульність та масштабованість;

— PostgreSQL для створення бази даних;

— Sequelize для роботи з реляційною базою даних;

— Docker для контейнеризації та спрощення розгортання.

Можливі сфери застосування. Розроблений сервіс є частиною Web-платформи, яка може бути використана для:

— Організації роботи ОСС КНУ імені Тараса Шевченка;

— Підвищення ефективності комунікації між студентами та ОСС;

— Автоматизації процесів організації студентських заходів та ініціатив;

— Збору та обробки звернень студентів;

— Підвищення прозорості діяльності ОСС.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз діяльності органів студентського самоврядування (ОСС)

Студентське самоврядування є невід'ємною частиною громадського самоврядування у закладах вищої освіти. Згідно з Законом України "Про вищу освіту", студентське самоврядування – це право і можливість студентів вирішувати питання навчання і побуту, захисту прав та інтересів студентів, а також брати участь в управлінні закладом вищої освіти [1].

У Київському національному університеті імені Тараса Шевченка органи студентського самоврядування функціонують як на рівні структурних підрозділів, так і на загальноуніверситетському рівні. Структура ОСС в КНУ імені Тараса Шевченка представлена на рисунку 1.1 [2].

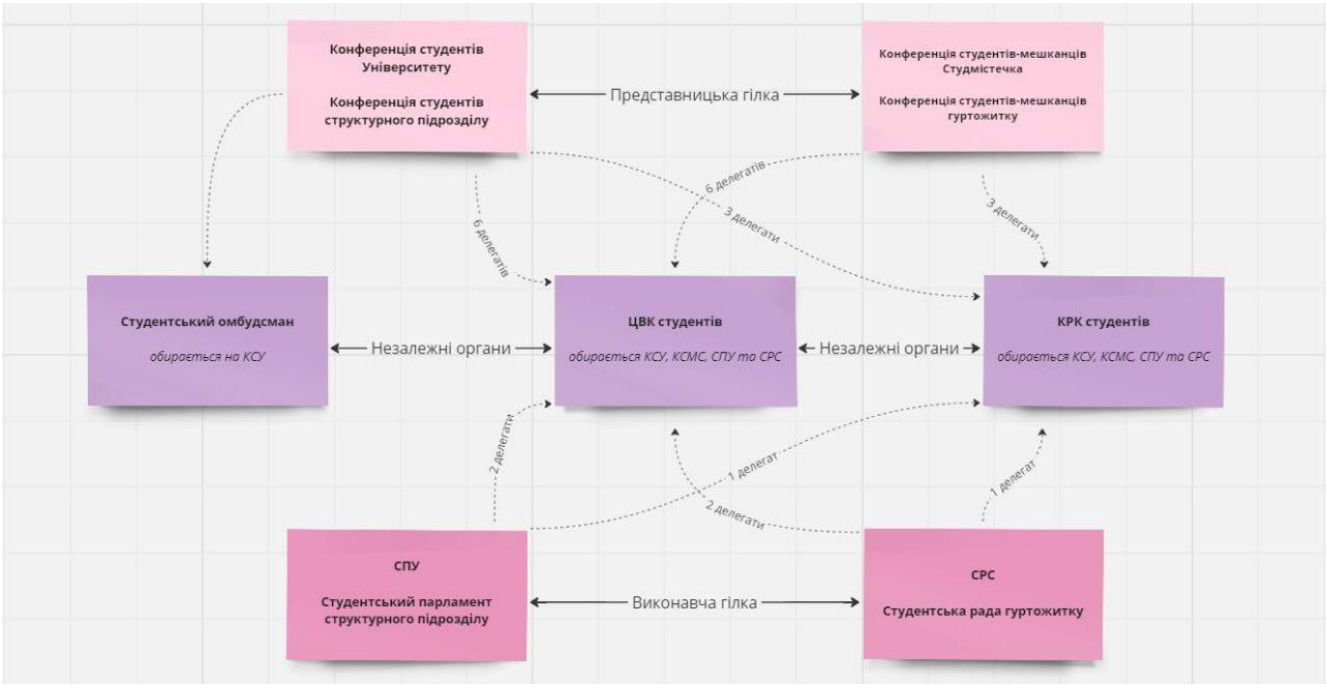

Рисунок 1.1 – Структура ООС КНУ імені Тараса Шевченка

1.1.1 Структура ОСС

В КНУ імені Тараса Шевченка система студентського самоврядування має таку структуру:

а) Представницькі органи:

1) Конференція студентів Університету (КСУ);
2) Конференція студентів структурного підрозділу (КССП);
3) Конференція студентів-мешканців Студмістечка (КСМС);
4) Конференція студентів-мешканців гуртожитку (КСМГ);

б) Виконавчі органи:

1) Студентський парламент університету (СПУ);
2) Студентський парламент структурного підрозділу (СПСП);
3) Студентська рада Студмістечка (СРС);
4) Студентська рада гуртожитку (СРГ);

в) Незалежні органи:

1) Студентський омбудсмен – обирається КСУ;
2) Центральна виборча комісія студентів (ЦВКс) – обирається КСУ, КСМС, СПУ та СРС;
3) Контрольно-ревізійна комісія студентів (КРК) – обирається КСУ, КСМС, СПУ та СРС.

Відповідно до схеми, представницька гілка включає конференції студентів різних рівнів, а виконавча гілка представлена студентськими парламентами та радами. Незалежні органи забезпечують контроль та дотримання прав студентів у системі самоврядування.

1.1.2 Основні задачі та процеси ОСС

Аналіз діяльності ОСС КНУ імені Тараса Шевченка дозволив виявити основні задачі та процеси, які потребують автоматизації:

— Інформування студентів – поширення інформації про діяльність ОСС, важливі події та можливості;

— Проведення засідань – організація та проведення засідань ОСС, формування порядку денного, документування рішень;

— Організація заходів – планування, організація та проведення різноманітних заходів для студентів;

— Розгляд звернень – прийом, обробка та розгляд звернень студентів з різних питань;

— Управління проєктами – планування, реалізація та моніторинг студентських проєктів та ініціатив;

— Звітність – підготовка та оприлюднення звітів про діяльність ОСС.

1.1.3 Проблеми організації роботи ОСС

Аналіз діяльності ОСС КНУ імені Тараса Шевченка виявив низку проблем, які негативно впливають на ефективність їх роботи:

— Відсутність єдиної інформаційної системи – інформація про діяльність ОСС розпорошена між різними платформами та ресурсами;

— Складність в організації заходів – організація заходів потребує значних витрат часу та ресурсів через відсутність автоматизованих інструментів;

— Проблеми з обробкою звернень – відсутня єдина система для обробки звернень студентів, що призводить до втрати або затримки в розгляді деяких звернень;

— Відсутність прозорості – студенти не мають достатнього доступу до інформації про діяльність ОСС, прийняті рішення та використання коштів;

— Складність у підготовці звітності – підготовка звітів про діяльність ОСС є трудомісткою через відсутність єдиної системи збору та аналізу даних.

Ці проблеми свідчать про необхідність створення єдиної Web-платформи, яка б забезпечила автоматизацію основних процесів діяльності ОСС та покращила комунікацію між різними структурними підрозділами та студентами.

1.2 Огляд існуючих рішень для автоматизації діяльності ОСС

Для визначення оптимального підходу до розробки Web-платформи для органів студентського самоврядування КНУ імені Тараса Шевченка було проведено аналіз існуючих рішень, які використовуються для автоматизації діяльності ОСС в інших закладах вищої освіти та організаціях.

1.2.1 Наявні платформи для студентського самоврядування

— Студентська рада КПІ ім. Ігоря Сікорського – Web-портал, що забезпечує інформування студентів про діяльність ОСС, анонси заходів та новини [3]. Портал не надає функціональності для автоматизації внутрішніх процесів ОСС;

— MyClassBoard – інтегрована система управління освітнім процесом, яка включає модулі для комунікації, обліку відвідуваності, управління заходами та академічною успішністю [4]. Система має функціонал для шкіл та коледжів, але має обмежену адаптацію до потреб студентського самоврядування в українських ЗВО;

— Електронний кабінет студента – система, впроваджена в деяких українських ЗВО (ЛНТУ [5], КНЛУ [6], НМУ ім. О. О. Богомольця [7]), яка забезпечує доступ студентів до навчальних матеріалів, розкладу занять та іншої інформації. Однак, ця система не включає функціональність для роботи ОСС;

— Google Workspace for Education – набір інструментів Google, який використовується деякими ОСС для організації внутрішньої роботи [8]. Проте, ці інструменти не інтегровані в єдину систему та не покривають усі потреби ОСС;

— Microsoft Teams – платформа для командної роботи, яка використовується деякими ОСС для комунікації та спільної роботи над проєктами

[9]. Однак, як і Google Workspace, ця платформа не надає спеціалізованих інструментів для роботи ОСС.

Порівняльний аналіз функціональності існуючих рішень представлено в таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз функціональності існуючих рішень

Функціо-нальність	**Студентська рада КПІ**	**MyClass-Board**	**Е-кабінет студента**	**Google Workspace**	**Microsoft Teams**
Інформува-ння студентів	+	+	+	+/-	+/-
Організація заходів	-	+	-	+/-	+/-
Обробка звернень	-	+	-	+/-	+/-
Управління проєктами	-	+/-	-	+/-	+
Фінансова звітність	-	+/-	-	+/-	-
Інтеграція з системами університету	+/-	+/-	+	-	-

Примітка. "+" – функціональність повністю реалізована, "+/-" – функціональність частково реалізована, "-" – функціональність відсутня.

1.2.2 Аналіз технологічних підходів до розробки Web-платформ

Окрім аналізу існуючих рішень, було досліджено різні технологічні підходи до розробки Web-платформ, які можуть бути використані для створення системи для ОСС КНУ імені Тараса Шевченка.

— Монолітна архітектура – традиційний підхід, при якому весь функціонал реалізований в рамках єдиного додатку. Перевагами є простота розробки та розгортання, недоліками – складність масштабування та підтримки при зростанні системи;

— Мікросервісна архітектура – підхід, при якому система розбивається на набір незалежних сервісів, кожен з яких відповідає за окрему функціональність. Перевагами є гнучкість, масштабованість та можливість незалежного розгортання сервісів, недоліками – складність у розробці та тестуванні;

— Серверний рендеринг (SSR) – підхід, при якому HTML-сторінки генеруються на сервері. Перевагами є краща SEO-оптимізація та швидше початкове завантаження, недоліками – більше навантаження на сервер;

— Клієнтський рендеринг (CSR) – підхід, при якому HTML-сторінки генеруються на клієнті з використанням JavaScript. Перевагами є краща інтерактивність та менше навантаження на сервер, недоліками – гірша SEO-оптимізація та повільніше початкове завантаження;

— Progressive Web Applications (PWA) – підхід, при якому Web-додаток має функціональність та поведінку, притаманні нативним мобільним додаткам. Перевагами є можливість роботи офлайн та покращений користувацький досвід, недоліками – складність у розробці та тестуванні.

Порівняльний аналіз технологічних підходів представлено в таблиці 1.2.

Таблиця 1.2 – Порівняльний аналіз технологічних підходів

Критерій	**Монолітна архітектура**	**Мікро-сервісна архітектура**	**SSR**	**CSR**	**PWA**
Масштабованість	Низька	Висока	Середня	Середня	Середня
Швидкість розробки	Висока	Низька	Середня	Висока	Низька
Надійність	Середня	Висока	Висока	Середня	Висока
SEO-оптимізація	-	-	Висока	Низька	Середня
Користувацький досвід	-	-	Середній	Високий	Високий
Можливість роботи офлайн	-	-	Низька	Низька	Висока
Складність підтримки	Висока	Середня	Середня	Середня	Висока

Проведений аналіз показав, що для розробки Web-платформи для ОСС КНУ імені Тараса Шевченка найбільш доцільним є використання мікросервісної архітектури з елементами PWA, що забезпечить необхідну гнучкість, масштабованість та покращений користувацький досвід.

2 АРХІТЕКТУРА ТА ТЕХНОЛОГІЇ РОЗРОБКИ

2.1 Обґрунтування вибору архітектури системи

Вибір архітектури є одним із ключових етапів розробки програмного забезпечення, який суттєво впливає на масштабованість, надійність, підтримуваність та продуктивність системи. При розробці Web-платформи для органів студентського самоврядування КНУ ім. Тараса Шевченка необхідно врахувати ряд факторів, що обумовлюють архітектурні рішення:

— Різноманітність функціональних вимог. Система повинна забезпечувати різноманітні функціональні можливості, включаючи управління користувачами, заходами, новинами, документами, голосуваннями, що призводить до складної доменної моделі;

— Необхідність масштабування. З ростом кількості користувачів та функціональних можливостей система повинна зберігати високу продуктивність та мати можливість горизонтального масштабування;

— Розподіл відповідальності. Різні аспекти системи (автентифікація, управління користувачами, управління заходами тощо) повинні бути логічно розділені для забезпечення чіткого розподілу відповідальності та можливості незалежного розвитку;

— Стабільність і відмовостійкість. Система повинна продовжувати працювати навіть у випадку відмови окремих компонентів;

— Технологічна гетерогенність. Різні компоненти системи можуть використовувати різні технології та мови програмування в залежності від їх специфічних вимог.

Враховуючи ці фактори, для розробки Web-платформи для ОСС було обрано мікросервісну архітектуру. Мікросервісна архітектура дозволяє розбити систему на незалежні, слабо пов'язані сервіси, кожен з яких відповідає за конкретну функціональну область. У контексті нашої системи ключовими мікросервісами є:

— Auth-сервіс – відповідає за автентифікацію та авторизацію користувачів;

— Oss-сервіс – основний сервіс для управління діяльністю органів студентського самоврядування;

— User-сервіс – відповідає за управління користувачами та їх профілями;

— Notification-сервіс – відповідає за надсилання сповіщень користувачам;

— Gateway-сервіс – виступає єдиною точкою входу для клієнтських запитів і маршрутизує їх до відповідних сервісів.

Така архітектура дозволяє забезпечити незалежний розвиток та розгортання кожного сервісу, горизонтальне масштабування, ізоляцію помилок, а також можливість використання різних технологій для кожного сервісу в залежності від його специфічних вимог.

Для взаємодії між мікросервісами було обрано кілька підходів комунікації, що забезпечують оптимальне вирішення різних сценаріїв взаємодії відповідно до архітектурної схеми:

— TCP протокол застосовується для базової комунікації між gateway-сервісом та oss-, auth-, user-сервісами. NestJS реалізує власний прикладний протокол поверх TCP для мікросервісної комунікації [10]. Цей протокол забезпечує надійний та швидкий обмін даними, що критично важливо для операцій аутентифікації та авторизації;

— gRPC використовується для високопродуктивної комунікації між oss-, auth-, user-сервісами [11]. Цей підхід базується на Protocol Buffers [12] для серіалізації даних та забезпечує строго типізовані контракти між сервісами. gRPC особливо ефективний для операцій, що вимагають низької затримки та високої пропускної здатності, таких як перевірка прав користувачів та отримання інформації про профілі користувачів. Впровадження gRPC дозволило суттєво знизити накладні витрати на мережеву комунікацію порівняно з традиційними REST API;

— RabbitMQ застосовується для асинхронної комунікації між oss-, auth-та user-сервісами і notification-сервісом [13]. Цей брокер повідомлень забезпечує

надійну доставку подій навіть при тимчасовій недоступності окремих сервісів, що критично важливо для довготривалих процесів, таких як виборчі кампанії та системи сповіщень. Використання обмінників та черг дозволяє гнучко маршрутизувати повідомлення та обробляти піки навантаження.

Така комбінація протоколів комунікації забезпечує гнучкість у взаємодії між сервісами та підвищує надійність системи в цілому, дозволяючи обирати оптимальний спосіб комунікації для кожного конкретного сценарію взаємодії. Конфігурація для цих протоколів централізовано управляється через відповідні файли налаштувань: grpc.config.ts для gRPC-з'єднань та database.config.ts для RabbitMQ та інших типів комунікації.

Окрім архітектури на рівні сервісів, для oss-сервісу було застосовано підхід чистої архітектури (Clean Architecture) [14], який забезпечує розділення рівнів відповідальності в рамках одного сервісу:

— Рівень представлення – відповідає за взаємодію з клієнтами через API;

— Рівень прикладної логіки – містить бізнес-логіку застосунку;

— Рівень доменної логіки – містить бізнес-сутності та правила предметної області;

— Рівень інфраструктури – відповідає за взаємодію з зовнішніми системами (базами даних, іншими сервісами).

Такий підхід дозволяє забезпечити чіткий розподіл відповідальності, спрощує тестування та підтримку коду, а також зменшує зв'язаність між різними частинами системи.

2.2 Мікросервісна архітектура та її переваги

Мікросервісна архітектура – це підхід до розробки програмного забезпечення, при якому додаток складається з набору невеликих, автономних сервісів, кожен з яких відповідає за конкретну функціональну область та взаємодіє з іншими сервісами через чітко визначені API [15]. На відміну від монолітної архітектури, де всі функціональні компоненти тісно інтегровані в єдиному

застосунку, мікросервісна архітектура розділяє відповідальність між незалежними сервісами.

Ключові характеристики мікросервісної архітектури, що були враховані при розробці Web-платформи для ОСС, включають:

— Декомпозиція за бізнес-можливостями. Кожен мікросервіс відповідає за конкретну бізнес-можливість або функціональну область. Наприклад, oss-сервіс відповідає за основні функції управління діяльністю органів студентського самоврядування, а auth-сервіс – за автентифікацію та авторизацію користувачів;

— Автономність. Кожен мікросервіс може бути розроблений, розгорнутий та масштабований незалежно від інших сервісів. Це дозволяє забезпечити гнучкість розвитку системи, де кожен компонент може еволюціонувати з власною швидкістю;

— Незалежність даних. Кожен мікросервіс має власне сховище даних, яке відповідає його специфічним вимогам. Це забезпечує слабку зв'язаність між сервісами та дозволяє використовувати різні типи баз даних в залежності від потреб кожного сервісу;

— Комунікація через API. Мікросервіси взаємодіють один з одним через чітко визначені API, що забезпечує слабку зв'язаність та інкапсуляцію внутрішньої логіки кожного сервісу.

Архітектура Web-платформи для органів студентського самоврядування, заснована на мікросервісному підході, представлена на рисунку 2.1.

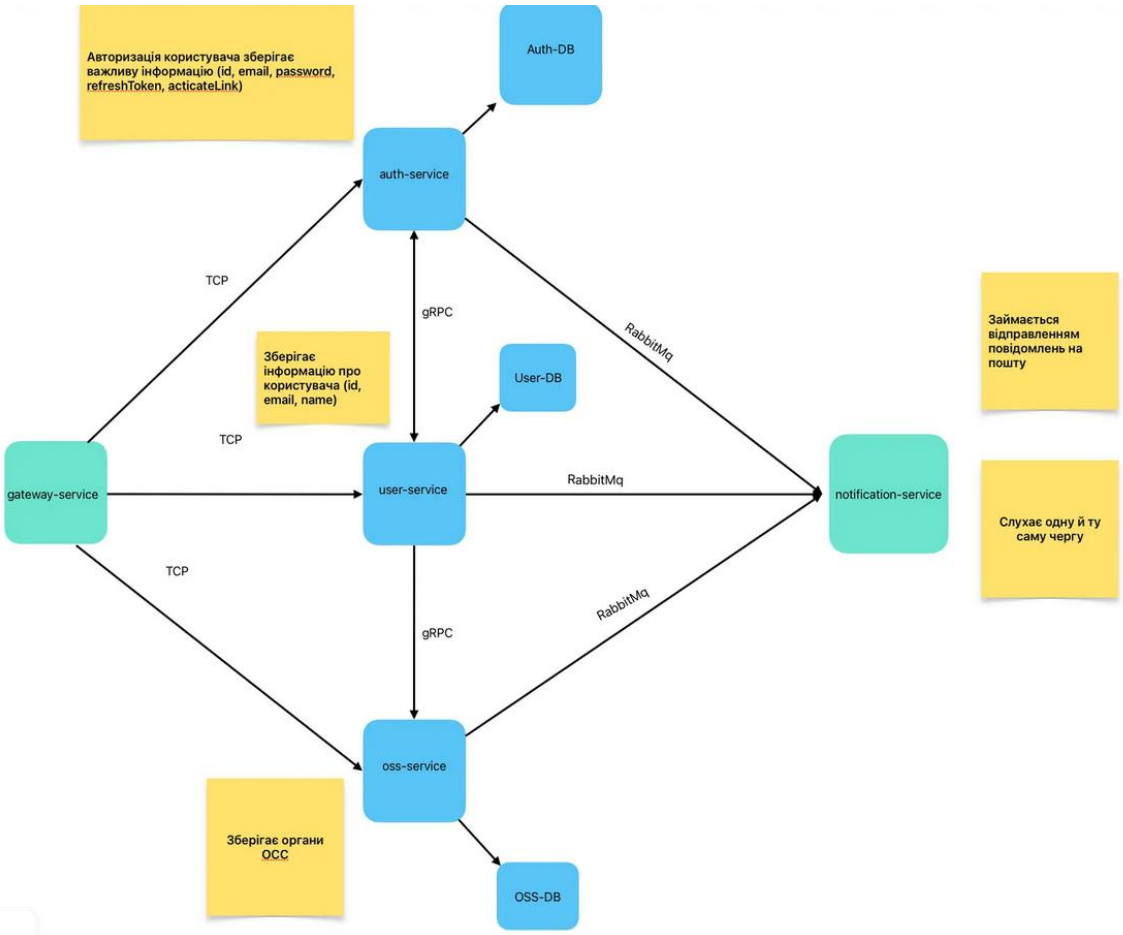

Рисунок 2.1 – Архітектура мікросервісної системи для Web-платформи ОСС

Використання мікросервісної архітектури для розробки Web-платформи для ОСС надає ряд важливих переваг:

— Масштабованість. Кожен мікросервіс може бути масштабований незалежно, що дозволяє ефективно розподіляти ресурси системи;

— Стійкість до відмов. Ізоляція мікросервісів один від одного забезпечує стійкість системи в цілому. Відмова одного сервісу не призводить до відмови всієї системи. Наприклад, якщо сервіс сповіщень тимчасово недоступний, користувачі все ще можуть використовувати основні функції системи;

— Технологічна гнучкість. Різні мікросервіси можуть використовувати різні технології та мови програмування в залежності від їх специфічних вимог. Хоча для oss-сервісу було обрано стек на основі TypeScript і Nest.js, інші сервіси можуть використовувати інші технології, якщо це буде доцільно;

— Паралельна розробка. Різні команди можуть одночасно працювати над різними мікросервісами, що прискорює розробку та впровадження нових функцій;

— Локалізація змін. Зміни в одному мікросервісі не впливають на інші сервіси, що спрощує підтримку та розвиток системи. Наприклад, зміна в логіці обробки заходів в oss-сервісі не вплине на роботу auth-сервісу.

Однак мікросервісна архітектура також має свої виклики:

— Складність управління розподіленою системою. Необхідно враховувати аспекти розподіленої системи, такі як мережеві затримки, синхронізація даних, моніторинг стану сервісів тощо;

— Складність транзакцій між сервісами. Оскільки кожен мікросервіс має власне сховище даних, забезпечення узгодженості даних між сервісами вимагає додаткових рішень, таких як використання шаблону Saga або використання подієвої архітектури;

— Операційна складність. Для ефективного функціонування мікросервісної системи необхідно забезпечити автоматизацію розгортання, моніторингу та масштабування сервісів.

Для подолання викликів мікросервісної архітектури у Web-платформі для ОСС впроваджено комплексне технологічне рішення, що охоплює різні аспекти комунікації, розгортання та моніторингу:

а) API Gateway (NestJS Gateway) забезпечує централізоване управління запитами клієнтів до мікросервісів, виконуючи роль єдиної точки входу для зовнішніх запитів. Шлюз здійснює маршрутизацію, балансування навантаження, кешування та забезпечує базову безпеку через аутентифікацію та авторизацію. Це дозволяє абстрагувати клієнтів від внутрішньої структури мікросервісів та спрощує впровадження крос-функціональних можливостей, таких як моніторинг та rate limiting.

б) Багаторівнева система комунікації між сервісами, що включає:

1) TCP-протокол для надійного та швидкого обміну критичними даними між сервісами, особливо для auth- та user-сервісів, де важлива мінімальна затримка;

2) gRPC для високопродуктивної комунікації з чітко визначеними інтерфейсами, що особливо ефективний для інтенсивного обміну даними між сервісами через використання Protocol Buffers для серіалізації та двонаправлений потоковий зв'язок;
3) RabbitMQ як брокер повідомлень для асинхронної комунікації, що забезпечує надійність доставки даних навіть при тимчасовій недоступності сервісів та ефективне масштабування при зростанні навантаження;

в) Контейнеризація через Docker спрощує розгортання, масштабування та управління мікросервісами в різних середовищах. Кожен сервіс упаковано в окремий контейнер з мінімальними залежностями, що забезпечує ізоляцію та консистентність середовища виконання.

Таким чином, мікросервісна архітектура, обрана для Web-платформи органів студентського самоврядування, забезпечує гнучкість, масштабованість та надійність системи, а також дозволяє ефективно організувати розробку та підтримку різних компонентів платформи.

2.3 Технології реалізації backend-частини

2.3.1 TypeScript як мова програмування

TypeScript є статично типізованою надбудовою над JavaScript, що компілюється в чистий JavaScript [16]. Ця мова програмування була розроблена компанією Microsoft і представлена в 2012 році. Основна мета TypeScript — покращити та забезпечити безпечне програмування для масштабних проєктів.

Використання TypeScript для розробки oss-сервісу Web-платформи для органів студентського самоврядування має ряд важливих переваг:

— Статична типізація: на відміну від JavaScript, TypeScript дозволяє визначати типи змінних, параметрів функцій, повернених значень та інших

елементів коду. Це дозволяє виявляти помилки ще на етапі компіляції, а не під час виконання програми, що значно підвищує надійність сервісу;

— Покращена читабельність та підтримуваність коду: система типів TypeScript робить код більш самодокументованим, що спрощує його розуміння та підтримку для команди розробників;

— Інтелектуальна підказка в IDE: завдяки статичній типізації, середовища розробки можуть надавати розширені можливості автодоповнення та підказок, що підвищує продуктивність розробників;

— Сучасні функції ECMAScript: TypeScript підтримує найновіші функції JavaScript, такі як асинхронні функції, деструктуризація, стрілочні функції, а також пропонує додаткові конструкції, як-от декоратори, що активно використовуються в Nest.js;

— Сумісність з JavaScript: TypeScript є супермножиною JavaScript, що означає, що будь-який існуючий JavaScript код є валідним TypeScript кодом. Це дозволяє поступово впроваджувати TypeScript у проєкти;

— Підтримка об'єктно-орієнтованого програмування: TypeScript надає всі необхідні засоби для застосування об'єктно-орієнтованих принципів проєктування, таких як класи, інтерфейси, модифікатори доступу, абстрактні класи тощо;

— Вдосконалені інструменти рефакторингу: типізація дозволяє інтегрованим середовищам розробки пропонувати більш надійні інструменти рефакторингу, що значно спрощує модифікацію коду;

— Гнучкі налаштування компілятора: TypeScript надає можливість налаштувати процес компіляції через файл tsconfig.json, де можна вказати цільову версію JavaScript, включити або виключити певні перевірки типів, налаштувати роботу з модулями тощо.

У контексті розробки oss-сервісу Web-платформи для органів студентського самоврядування, використання TypeScript дозволяє забезпечити надійність, підтримуваність та масштабованість кодової бази, що є критично важливим для сервісу, який повинен служити основою для автоматизації діяльності ОСС. TypeScript інтегрується з усіма іншими обраними технологіями, зокрема Nest.js,

Sequelize, RabbitMQ, і забезпечує надійну основу для розробки серверної частини Web-платформи.

2.3.2 Nest.js для серверної частини

Nest.js – це Node.js фреймворк для побудови ефективних, надійних та масштабованих серверних додатків [17]. Фреймворк поєднує елементи об'єктно-орієнтованого програмування (ООП), функціонального програмування та функціонального реактивного програмування (FRP), що робить його ідеальним вибором для розробки складних серверних додатків.

Для розробки oss-сервісу Web-платформи органів студентського самоврядування було обрано Nest.js з ряду причин:

— Архітектурна структура. Nest.js надає чітку та структуровану архітектуру, яка базується на модулях, контролерах та сервісах. Така структура сприяє розділенню відповідальності та спрощує підтримку кодової бази. Наприклад, в oss-сервісі було створено модулі для роботи з базою даних та окремими сутностями оної;

— Декоратори та метадані. Nest.js активно використовує декоратори TypeScript для опису метаданих та поведінки компонентів системи. Це робить код більш декларативним та зрозумілим;

— Dependency Injection. Nest.js має вбудовану систему впровадження залежностей, яка спрощує управління залежностями між компонентами системи та підвищує тестованість коду;

— Middleware, Guards, Interceptors, Filters. Nest.js надає різноманітні інструменти для обробки запитів та відповідей, що дозволяє ефективно реалізувати такі аспекти як автентифікація, авторизація, валідація, логування тощо;

— Модульність та розширюваність. Nest.js дозволяє легко інтегрувати сторонні бібліотеки та модулі, що робить його ідеальним для створення мікросервісів;

— Документація та спільнота. Nest.js має детальну документацію та активну спільноту розробників, що спрощує процес навчання та вирішення проблем;

— Підтримка TypeScript. Nest.js розроблений з використанням TypeScript і повністю підтримує всі його можливості, що забезпечує типобезпеку та покращує процес розробки;

— Інтеграція з OpenAPI (Swagger). Nest.js надає вбудовану підтримку для автоматичної генерації документації API за допомогою Swagger.

Використання Nest.js для розробки oss-сервісу дозволило забезпечити чітку структуру коду, високу підтримуваність та масштабованість системи, а також ефективну інтеграцію з іншими технологіями, такими як Sequelize та RabbitMQ.

3 РОЗРОБКА OSS-СЕРВІСУ

3.1 Загальна архітектура

Oss-сервіс розроблено як частину мікросервісної архітектури з використанням TypeScript та фреймворку Nest.js. Структура проєкту організована за принципом модульності, що забезпечує високу масштабованість та підтримуваність коду.

Модулі сервісу:

— Oss – основний модуль для роботи з органами студентського самоврядування;

— Unit – модуль для роботи зі структурними підрозділами університету;

— Election-district – модуль для роботи з виборчими округами;

— User-in-oss – модуль для управління користувачами в органах студентського самоврядування.

Основні компоненти модулів:

— Entities – містить файли, що описують сутності системи, такі як *oss.entity.ts*;

— Dto – містить об'єкти передачі даних, наприклад *oss-details.dto.ts*;

— Pipes – містить конвеєри для валідації та трансформації даних (*grpc-validation.pipe.ts*);

— Controllers – контролери для обробки TCP-запитів (*oss.controller.ts*);

— Modules – модулі, які об'єднують компоненти сервісу (*oss.module.ts*);

— Services – сервіси, що реалізують бізнес-логіку (*oss.service.ts*).

Така модульна структура відповідає передовим практикам розробки з використанням Nest.js та забезпечує чіткий поділ відповідальності між компонентами системи.

3.2 Розробка моделей даних

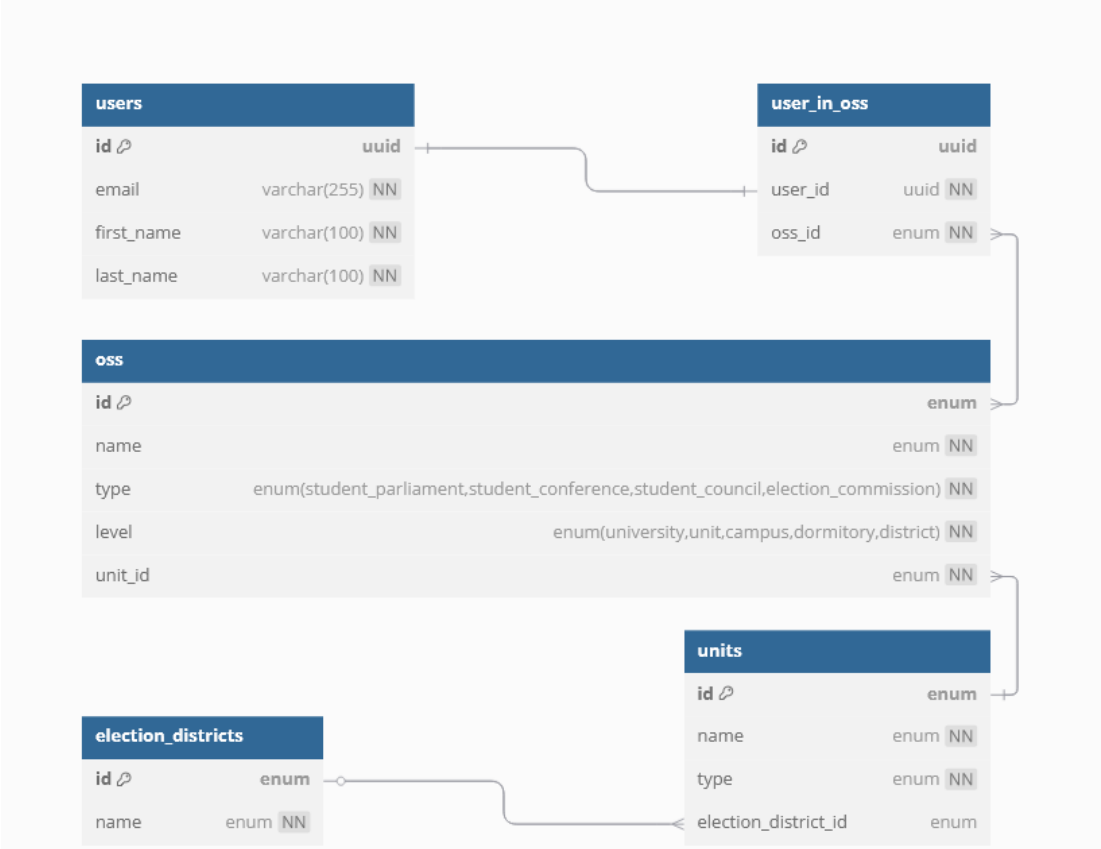

Рисунок 3.1 – Структура бази даних oss-сервісу

У системі реалізовано декілька основних сутностей, представлених на рисунку 3.1, які відображають структуру органів студентського самоврядування та підрозділів університету:

а) Oss (*oss.entity.ts*) – базова сутність для органів студентського самоврядування, яка містить;

1) ідентифікатор;
2) назву органу;
3) тип органу;
4) рівень органу;
5) ідентифікатор структурного підрозділу (Unit);
6) додаткові поля.

б) Unit (*unit.entity.ts*) – сутність для структурних підрозділів університету;

1) ідентифікатор;

2) назву підрозділу;
3) тип підрозділу;
4) код підрозділу (наприклад, FIT, EF, G1);
5) ідентифікатор виборчого округу (ElectionDestiny);
6) додаткові поля.

в) ElectionDistrict (*election-district.entity*) – сутність для представлення виборчих округів;

1) ідентифікатор;
2) назва;
3) додаткові поля.

г) UserInOss (*user-in-oss.entity.ts*) - сутність для представлення користувачів в органах самоврядування;

1) ідентифікатор;
2) зв'язок з користувачем (User);
3) зв'язок з органом самоврядування (Oss);
4) додаткові поля.

Зв'язки між сутностями:

— Реалізовано зв'язки типу "один-до-багатьох" та "багато-до-багатьох" між сутностями, використовуючи відповідні асоціації Sequelize (hasMany, belongsTo, belongsToMany);

— Застосовано Sequelize [18] як ORM для маппінгу сутностей до бази даних PostgreSQL, що забезпечує зручну взаємодію з базою даних через об'єктно-орієнтований підхід;

— Впроваджено міграції Sequelize для контролю версій схеми бази даних та полегшення розгортання в різних середовищах.

У моделях даних враховано ієрархічну структуру університету та студентського самоврядування. Також для більшості полів (ідентифікатори, назви, типи тощо) використано тип даних Enum, який дозволяє структурувати обширний список структурних підрозділів та органів і за необхідності змінювати його.

Для зберігання даних використовується PostgreSQL, що забезпечує надійність, стабільність та підтримку складних запитів і транзакцій, а також дозволяє ефективно працювати зі зв'язками між сутностями та складними структурами даних.

3.3 Реалізація бізнес-логіки

Бізнес-логіка oss-сервісу реалізована відповідно до принципів DDD (Domain-Driven Design) [19] та гексагональної архітектури. Сервісний шар представлений кількома основними компонентами. OssService (файл *oss.service.ts*) реалізує основну бізнес-логіку для органів студентського самоврядування, включаючи створення, оновлення та видалення органів, пошук органів за різними критеріями, управління ієрархією органів та логіку взаємодії між різними типами органів.

UnitService (файл *unit.service.ts*) відповідає за логіку роботи зі структурними підрозділами університету. Цей сервіс забезпечує CRUD-операції для підрозділів, логіку взаємодії між підрозділами різних типів та методи для отримання інформації про підрозділи.

У реалізації бізнес-логіки застосовано патерн Repository для роботи з даними та впровадження залежностей через конструктор (DI). Важливими аспектами є також обробка помилок та виключень, валідація вхідних даних через DTO та pipes, кешування для оптимізації продуктивності та логування важливих подій.

3.4 Інтеграція з іншими мікросервісами

Oss-сервіс розроблено з урахуванням необхідності ефективної взаємодії з іншими компонентами мікросервісної архітектури. Для забезпечення зв'язку з різними сервісами впроваджено комплексний підхід до комунікації, що включає TCP, gRPC та RabbitMQ.

В рамках oss-сервісу реалізовано інтерфейси для роботи через TCP протокол, який використовується для взаємодії з gateway-сервісом. Це забезпечує надійне та швидке обслуговування запитів, що надходять від клієнтської частини.

Для високопродуктивної комунікації oss-сервіс використовує gRPC, який інтегровано для забезпечення взаємодії з Auth-сервісом та User-сервісом. У складі oss-сервісу розроблено gRPC-клієнти, що працюють з Protocol Buffers для серіалізації даних та забезпечують строго типізовані контракти. Це дозволяє oss-сервісу ефективно виконувати операції, пов'язані з перевіркою прав користувачів та отриманням інформації про профілі. Всі необхідні для цього Proto-файли та gRPC-сервіси знаходяться в спеціальному модулі сервісу, а їх конфігурація визначена у файлі *grpc.config.ts*.

Для забезпечення асинхронної комунікації в oss-сервісі планується впровадити роботу з RabbitMQ [20]. Реалізація як виробників (producers), так і споживачів (consumers) повідомлень дозволить oss-сервісу ініціювати події для notification-сервісу. Параметри підключення до брокера повідомлень та налаштування черг слід прописати у файлі *database.config.ts*.

В рамках oss-сервісу реалізовано механізми підвищення надійності комунікації, включаючи retry-політики для повторних спроб з'єднання при тимчасових проблемах та timeout-обмеження для запобігання блокуванню операцій.

Всі компоненти взаємодії з іншими сервісами інкапсульовано в окремих модулях oss-сервісу, що забезпечує чітке розділення відповідальності та спрощує подальшу підтримку та розширення функціональності. Завдяки такому підходу oss-сервіс зможе ефективно виконувати свою роль у загальній архітектурі Web-платформи для органів студентського самоврядування.

3.5 Контейнеризація та розгортання

Сервіс контейнеризовано з використанням Docker згідно з офіційною документацією. Docker – це платформа для розробки, доставки та запуску додатків

у контейнерах, що забезпечує ізоляцію програмного середовища від інфраструктури, на якій воно працює [21]. Контейнерний підхід дозволяє уникнути проблем сумісності та забезпечує стабільну роботу додатків незалежно від середовища розгортання.

Dockerfile містить конфігурацію для побудови Docker-образу з використанням багатоетапної побудови для оптимізації розміру образу.

Docker Compose використовується для локальної розробки та тестування, забезпечуючи конфігурацію для сервісу та його залежностей, таких як база даних та RabbitMQ. Також налаштовано мережі, томи та змінні середовища.

Dockerfile складається з двох етапів: етапу побудови, де відбувається компіляція TypeScript-коду, та продакшн-етапу, де створюється фінальний образ з мінімальним набором залежностей. Така комплексна система тестування та контейнеризації забезпечує надійну роботу oss-сервісу в мікросервісній архітектурі та спрощує процес розгортання та масштабування.

3.6 Тестування

З використанням Docker були проведені тестування у контрольованому середовищі. Базовий аналіз швидкодії показав наступні результати для ключових операцій oss-сервісу:

а) операції читання;

1) отримання списку всіх OSS організацій (GET /oss): середній час відгуку 85 мс, медіана 72 мс;
2) отримання конкретної OSS за ID (GET /oss/:id): середній час відгуку 45 мс, медіана 38 мс;
3) фільтрація за типом організації (GET /oss/by-type/:type): середній час відгуку 92 мс, медіана 81 мс;

б) операції запису;

1) створення нової OSS (POST /oss): середній час відгуку 156 мс, медіана 142 мс;
2) оновлення існуючої OSS (PUT /oss/:id): середній час відгуку 98 мс, медіана 89 мс;
3) видалення OSS (DELETE /oss/:id): середній час відгуку 67 мс, медіана 58 мс.

Аналіз логів за період тестування показав наступний розподіл HTTP статусів:

— 2xx (успішні запити): 98.7%;
— 4xx (помилки клієнта): 1.1%;
— 5xx (помилки сервера): 0.2%.

Більшість помилок 4xx категорії були пов'язані з валідацією вхідних даних, що вказує на коректну роботу системи перевірок. Помилки 5xx виникали переважно при тимчасовій недоступності зовнішніх сервісів.

Тестування відновлення після різних типів збоїв показало:

— Рестарт контейнера: повне відновлення за 12-18 секунд;
— Втрата з'єднання з БД: автоматичне перепідключення за 3-5 секунд.

ВИСНОВКИ

У даній курсовій роботі було розроблено архітектуру серверної частини, зокрема oss-сервісу, Web-платформи для органів студентського самоврядування КНУ імені Тараса Шевченка. Розроблений сервіс є одним із компонентів мікросервісної архітектури системи, що відповідає за обробку бізнес-логіки, пов'язаної з діяльністю ОСС.

В ході виконання роботи було вирішено наступні завдання:

— Проаналізовано діяльність органів студентського самоврядування та визначено основні процеси, що потребують автоматизації;

— Обґрунтовано вибір мікросервісної архітектури для реалізації Web-платформи;

— Визначено та впроваджено стек технологій для розробки backend-частини: TypeScript, Nest.js, Sequelize, RabbitMQ, Docker;

— Розроблено моделі даних та реалізовано бізнес-логіку oss-сервісу;

— Забезпечено інтеграцію oss-сервісу з іншими мікросервісами;

— Розроблено систему контейнеризації для спрощення розгортання сервісу.

Практична значимість роботи полягає в тому, що впровадження розробленого сервісу у рамках Web-платформи дозволить підвищити ефективність роботи органів студентського самоврядування КНУ імені Тараса Шевченка, автоматизувати ключові процеси та покращити комунікацію між різними структурними підрозділами ОСС.

У перспективі планується подальший розвиток системи шляхом додавання нових функціональних можливостей, вдосконалення інтеграції з іншими мікросервісами та розширення API.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Закон України "Про вищу освіту" // Відомості Верховної Ради України. 2014. № 37-38. URL: https://zakon.rada.gov.ua/laws/show/1556-18#Text (дата звернення: 15.05.2025).

2. Положення про органи студентського самоврядування КНУ імені Тараса Шевченка від 10 квітня 2025 року. Київ, 2025. 24 с.

3. Студентська рада КПІ ім. Ігоря Сікорського. URL: https://sc.kpi.ua/ (дата звернення: 20.05.2025).

4. MyClassboard: Student Information System. URL: https://www.myclassboard.com/ (дата звернення: 18.05.2025).

5. Луцький національний технічний університет: Деканат. URL: https://web-dk.lntu.edu.ua (дата звернення: 22.05.2025).

6. Київський національний лінгвістичний університет: Студентський портал. URL: https://student.knlu.edu.ua (дата звернення: 22.05.2025).

7. Національний медичний університет: Особистий кабінет студента. URL: https://nmuofficial.com/studentu/student-personal-cabinet/ (дата звернення: 22.05.2025).

8. Google for Education: Workspace for Education. URL: https://edu.google.com/workspace-for-education/editions/overview/ (дата звернення: 25.05.2025).

9. Microsoft Teams: Group Chat Software. URL: https://www.microsoft.com/uk-ua/microsoft-teams/group-chat-software/ (дата звернення: 25.05.2025).

10. NestJS Microservices. URL: https://docs.nestjs.com/microservices/basics (дата звернення: 10.02.2025).

11. NestJS gRPC Documentation. URL: https://docs.nestjs.com/microservices/grpc (дата звернення: 12.02.2025).

12. Protocol Buffers Documentation. Google Developers. URL: https://protobuf.dev/ (дата звернення: 12.02.2025).

13. NestJS RabbitMQ Documentation. URL: https://docs.nestjs.com/microservices/rabbitmq (дата звернення: 14.02.2025).

14. Node Clean Architecture Deep Dive. Roystack Blog. 2019. URL: https://roystack.home.blog/2019/10/22/node-clean-architecture-deep-dive/ (дата звернення: 16.02.2025).

15. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd Edition. O'Reilly Media, 2021. 592 p.

16. TypeScript Documentation. Microsoft Corporation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 08.02.2025).

17. NestJS Documentation. Kamil Myśliwiec. URL: https://docs.nestjs.com/ (дата звернення: 05.02.2025).

18. Sequelize Documentation. Sequelize Team. URL: https://sequelize.org/master/ (дата звернення: 15.02.2025).

19. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional, 2003. 560 p.

20. RabbitMQ Documentation. VMware Inc. URL: https://www.rabbitmq.com/documentation.html (дата звернення: 14.02.2025).

21. Docker Documentation. Docker Inc. URL: https://docs.docker.com/ (дата звернення: 20.02.2025).