Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра
на тему: «Розробка back-end частини веб-додатку для створення онлайн-конференцій»

Виконав: студент групи ІП321-2
Спеціальність 121 «Інженерія програмного забезпечення»
Винник Олександр Володимирович
(прізвище та ініціали)
Керівник к.т.н., доц. Ульяновська Ю.В.
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент Університет митної справи та фінансів
(місце роботи)
Доцент кафедри кібербезпеки та інформаційних технологій
(посада)
к.т.н., доцент Флоров С.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Винник О.В. Розробка back-end частини веб-додатку для створення онлайн-конференцій.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавра за спеціальністю 121 «Інженерія програмного забезпечення» – Університет митної справи та фінансів, Дніпро, 2025.

У процесі дослідження здійснено глибокий аналіз існуючих програмних рішень для організації онлайн-конференцій, зокрема таких платформ, як Zoom Events, Microsoft Teams та Google Meet. Встановлено, що ці сервіси, попри свою популярність, мають суттєві обмеження щодо використання в академічному середовищі, зокрема недостатню підтримку процесів рецензування, слабку інтеграцію з науковими базами даних та обмежену можливість кастомізації під специфічні потреби наукових заходів.

На підставі порівняльного аналізу архітектурних підходів було обґрунтовано доцільність застосування мікросервісної архітектури, яка забезпечує гнучкість, масштабованість та спрощує супровід окремих компонентів системи. Реалізацію серверної частини виконано із застосуванням технологій Node.js, Express.js та MongoDB, що дало змогу створити ефективне, високопродуктивне та безпечне серверне середовище, орієнтоване на потреби наукової спільноти.

Проведене тестування підтвердило коректність функціонування серверної частини, її стійкість до навантажень та відповідність функціональним вимогам, що засвідчує готовність системи до реального використання.

Ключові слова: веб-застосунок, онлайн-конференція, Node.js, Express.js, MongoDB, REST API, мікросервісна архітектура, JWT, backend.

ABSTRACT

Vynnyk O.V. Development of the Back-End Part of a Web Application for Hosting Online Conferences

Bachelor's thesis for obtaining a degree in Software Engineering, specialty 121. – University of Customs and Finance, Dnipro, 2025.

In the course of the research, a comprehensive analysis of existing software solutions for hosting online conferences was carried out, including platforms such as Zoom Events, Microsoft Teams, and Google Meet. It was determined that, despite their popularity, these services have significant limitations in the academic context, such as insufficient support for peer-review processes, weak integration with scientific databases, and limited customization capabilities for the specific needs of scholarly events.

Based on a comparative analysis of architectural approaches, the use of a microservice architecture was substantiated as the most appropriate solution, ensuring flexibility, scalability, and simplified maintenance of individual system components. The back-end part was implemented using Node.js, Express.js, and MongoDB, which enabled the creation of an efficient, high-performance, and secure server environment tailored to the needs of the academic community.

Testing confirmed the correct operation of the server component, its resistance to high loads, and compliance with functional requirements, demonstrating the system's readiness for real-world use.

Keywords: web application, online conference, Node.js, Express.js, MongoDB, REST API, microservice architecture, JWT, backend.

ЗМІСТ

ВСТУП

Актуальність проблеми. У сучасному цифровому суспільстві академічна та професійна комунікація стрімко трансформується під впливом інформаційно-комунікаційних технологій. Проведення наукових конференцій дедалі частіше переходить у віртуальне середовище. Це зумовлено не лише глобалізацією наукового простору, а й практичними потребами: зменшенням витрат на поїздки, необхідністю оперативного доступу до досліджень, а також прагненням до інклюзивності та доступності для учасників з різних країн.

Ринок програмних рішень для організації онлайн-конференцій представлений широким спектром платформ, що можна умовно розділити на універсальні засоби відеоконференц-зв'язку та більш спеціалізовані платформи для управління подіями.

Проте більшість наявних інструментів для проведення онлайн-заходів, такі як Zoom, Microsoft Teams чи Google Meet, орієнтовані на загальні бізнес-потреби й не враховують специфічних вимог академічної спільноти. Їм бракує функціоналу для подачі та рецензування статей, інтеграції з науковими базами даних та зручного керування конференціями. Саме тому розробка спеціалізованої платформи для онлайн-конференцій є актуальним напрямком у сфері прикладного програмування та цифровізації науки.

Метою роботи є розробка backend-частини веб-застосунку для онлайн-конференцій.

Методи дослідження: проектування та розробка веб-додатків.

У відповідності до поставленої мети сформульовано наступні завдання:

1. Дослідити предметну область і проаналізувати обмеження наявних платформ онлайн-конференцій.
2. Дослідити сучасні архітектурні підходи до побудови веб-застосунків.
3. Сформулювати та реалізувати функціональні вимоги.
4. Реалізувати backend-частини платформи.

5. Провести тестування отриманої результату.

Основні завдання проєкту:

1) розробити архітектуру веб-платформи;

2) реалізувати механізми реєстрації та автентифікації користувачів з використанням сучасних стандартів;

3) забезпечити можливості адміністрування конференцій та обробки наукових статей;

4) інтегрувати функціонал для завантаження файлів та роботи з базою даних;

5) підібрати сучасні інструменти розробки back-end частини, що забезпечать високу продуктивність та гнучкість.

Об'єктом дослідження є процеси організації та підтримки науково-комунікативних заходів.

Предметом дослідження є апаратно-програмне забезпечення для розробки веб-додатків.

Робота складається зі вступу, 3-х розділів, висновків, списку використаних джерел з 15 найменувань. Обсяг роботи 60 сторінок кваліфікаційної роботи, 37 рисунків, 4 таблиці.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.

1.1. Опис предметної області.

Історично сформовані практики проведення конференцій, що передбачають фізичну присутність учасників у визначеному місці та часі, зазнають кардинальних трансформацій під впливом стрімкого розвитку інформаційно-комунікаційних технологій (ІКТ). Ця еволюція інтегрується в ширший контекст глобальної цифрової трансформації, що пронизує усі сфери суспільства, вимагаючи нових, ефективніших підходів до формування та поширення знань.

Незважаючи на усталеність традиційних форм академічного обміну, сучасна предметна область виявляє низку системних проблем, які актуалізують необхідність розробки інноваційних цифрових рішень. До ключових проблем належать:

- Географічні та логістичні обмеження традиційних конференцій. Проведення заходів у фізичному просторі створює непереборні бар'єри для багатьох потенційних учасників. Обмеження мобільності, зумовлені як геополітичними, так і індивідуальними обставинами, виключають значну частину зацікавленої аудиторії з міжнародного наукового обігу. Це призводить до фрагментації наукової спільноти та перешкоджає інклюзивному доступу до знань.
- Висока економічна вартість організації та участі.

Проведення офлайн-конференцій вимагає значних фінансових ресурсів, що включають оренду приміщень, технічне забезпечення, логістику для спікерів та учасників, а також адміністративні витрати. Ці витрати неминуче відображаються у високих реєстраційних зборах, що стає серйозною перешкодою для дослідників з обмеженим фінансуванням, студентів та молодих вчених, особливо з країн, що розвиваються. Така ситуація обмежує академічну мобільність та рівний доступ до професійного розвитку.

- Недостатня оперативність поширення наукових результатів.

У традиційній парадигмі наукової комунікації, інтервал між представленням дослідження на конференції та його доступністю для широкого загалу може бути значним, особливо з урахуванням тривалих циклів публікації збірників тез або повних матеріалів. Це уповільнює процес обміну знаннями, що є критичним в умовах швидкого науково-технічного прогресу, де оперативність є запорукою своєчасного впровадження інновацій.

- Обмежена адаптованість та інтеграція існуючих комерційних платформ. Хоча ринок пропонує низку функціональних комерційних рішень для відеоконференцій та вебінарів (як-от Zoom Events, Microsoft Teams), вони переважно орієнтовані на загальні бізнес-потреби. Ці платформи часто демонструють недостатні можливості для глибокої кастомізації під специфічні потреби академічних та наукових конференцій. Типовими проблемами є висока вартість ліцензування, відсутність гнучких інструментів для організації багатоетапного рецензування наукових статей, обмежені можливості інтеграції з академічними бібліографічними базами даних та репозитаріями, а також неадекватний функціонал для структурованого управління науковими матеріалами. Це створює "прогалину" в інструментарії для організації наукових подій, яка не покривається універсальними комерційними продуктами.

Наслідком зазначених проблем є об'єктивна необхідність у розробці спеціалізованих платформ, які здатні забезпечити повний життєвий цикл академічного заходу в режимі онлайн.

1.2. Аналіз існуючих платформ онлайн-конференцій.

Ринок програмних рішень для організації онлайн-конференцій представлений широким спектром платформ, що можна умовно розділити на універсальні засоби відеоконференц-зв'язку та більш спеціалізовані платформи для управління подіями. Для глибокого розуміння предметної області та обґрунтування необхідності розробки власного рішення, доцільно провести аналіз трьох найбільш поширених комерційних платформ, які часто

використовуються для проведення конференцій та вебінарів: Zoom Events, Microsoft Teams та Google Meet.

1. Zoom Events Zoom Events є розширеною версією популярної платформи Zoom Meetings, призначеною для управління віртуальними подіями. Вона пропонує інтегровані функції для реєстрації, продажу квитків, створення віртуальних лобі та розширених сесій.

- Переваги: Висока якість відео- та аудіозв'язку, інтуїтивно зрозумілий інтерфейс, широкі можливості для проведення інтерактивних сесій та кімнат для обговорень, надійність з'єднання, підтримка великої кількості учасників.
- Недоліки (у контексті академічних конференцій): Обмежені можливості глибокої кастомізації під специфічні потреби наукових заходів, висока вартість ліцензування для розширених функцій, відсутність вбудованих механізмів для подачі та рецензування наукових статей, слабка інтеграція з академічними бібліографічними та репозитарними сервісами.

2. Microsoft Teams Microsoft Teams є інтегрованою платформою для спільної роботи, що входить до екосистеми Microsoft 365. Вона пропонує функції чату, відеоконференцій, спільного доступу до файлів та інтеграцію з іншими додатками Microsoft (наприклад, SharePoint, OneNote).

Переваги:

Глибока інтеграція з іншими продуктами Microsoft, що є перевагою для організацій, які вже використовують екосистему Microsoft; функції спільної роботи над документами; можливість створення команд та каналів для довготривалої взаємодії.

Недоліки:

Інтерфейс може бути перевантаженим для користувачів, не знайомих з екосистемою Microsoft; відсутність спеціалізованих інструментів для академічного менеджменту (наприклад, автоматизованого рецензування, керування абстрактами), складнощі з кастомізацією візуального стилю та робочих процесів конференції, потенційно висока вартість за повний набір функцій Microsoft 365.

3. Google Meet Google Meet – це сервіс відеоконференцій, що є частиною Google Workspace. Він пропонує базові та розширені можливості для онлайн-зустрічей, інтегруючись з Google Календарем, Gmail та Google Документами.

Переваги:

Простота використання та доступність (частина Google Workspace), зручна інтеграція з календарем та поштою, базові функції для проведення зустрічей та вебінарів, доступність для широкого кола користувачів з обліковим записом Google.

Недоліки:

Обмежений функціонал для організації складних, багатопотокових конференцій; відсутність інструментів для продажу квитків, розширеної аналітики подій; мінімальні можливості кастомізації інтерфейсу; повна відсутність спеціалізованих інструментів для академічного документообігу та інтеграції з науковими базами даних.

Для наочного порівняння переваг та недоліків зазначених платформ у контексті вимог до платформи онлайн-конференцій, призначеної для академічного середовища, наведена Таблиця 1.1.

Таблиця 1.1

Порівняльний аналіз існуючих платформ онлайн-конференцій

Критерій	Монолітна архітектура	Мікросервісна архітектура	Serverless-архітектура
Структура	Один цілісний застосунок	Розподілена система з незалежних сервісів	Функції, що виконуються за подією (Function-as-a-Service)
Масштабованість	Обмежена	Висока (масштабуються окремі сервіси)	Автоматична, гнучка
Простота реалізації	Висока на початковому етапі	Складна (потрібна координація сервісів)	Залежить від платформи (AWS Lambda, Google Cloud)

Час розгортання	Швидкий	Складніший, потребує оркестрації	Швидкий
Вимоги до інфраструктури	Сервер/хостинг	Контейнери (Docker, Kubernetes)	Хмарні сервіси
Відмовостійкість	Низька (помилка впливає на весь додаток)	Висока (відмова одного сервісу не зупиняє систему)	Висока, але залежить від хмари
Гнучкість у розробці	Низька	Висока (можна використовувати різні мови, бази)	Середня
Вартість розгортання	Низька	Вища (через складність інфраструктури)	Може бути високою за великого навантаження
Сумісність з REST API	Повна	Повна	Обмежена (залежить від обробників подій)
Відповідність проекту	Обмежена (менш гнучка та масштабована)	Оптимальна (обрана архітектура для проєкту)	Невиправдана складність і залежність від хмари

Таким чином, попри загальну популярність та базову функціональність, що забезпечується такими платформами, як Zoom Events, Microsoft Teams та Google Meet, вони виявляються недостатньо ефективними для повноцінного задоволення специфічних потреб академічного середовища. Їхні ключові недоліки полягають в обмежених можливостях глибокої кастомізації під унікальні вимоги наукових заходів, надмірній вартості ліцензування для некомерційних установ, а також суттєвій відсутності інтеграції зі спеціалізованими бібліографічними та репозитарними сервісами. Більш того, ці універсальні платформи не пропонують вбудованого функціоналу для складних, багатоетапних академічних процесів, таких як подача та прозоре рецензування наукових статей чи комплексне управління програмними комітетами, що

вимагає використання неінтегрованих сторонніх рішень як наслідок значно ускладнює та здорожує організацію наукових подій.

1.3. Аналіз методів та підходів реалізації платформи.

На підготовчому етапі розробки платформи для створення онлайн-конференцій важливо здійснити ґрунтовний аналіз архітектурних підходів та технологічних засобів, що дозволяють забезпечити ефективність, масштабованість та безпеку веб-додатку. Вибір відповідних архітектурних рішень впливає на подальшу структуру розробки, супровід системи та можливість її адаптації до зростаючих вимог.

Серед найпоширеніших підходів до архітектури веб-застосунків виокремлюють монолітну, мікросервісну та серверлес архітектури. Також важливим є визначення методу взаємодії з клієнтом (наприклад, REST API чи GraphQL), а також вибір бази даних: реляційної (MySQL, PostgreSQL) чи документо-орієнтованої (MongoDB).

Для прийняття обґрунтованого рішення доцільно здійснити порівняльний аналіз цих підходів, зосередившись на таких критеріях, як гнучкість, масштабованість, простота розгортання, вимоги до ресурсів і відповідність специфіці академічної платформи.

Таблиця 1.2

Порівняльний аналіз архітектур і методів реалізації

Критерій	Монолітна архітектура	Мікросервісна архітектура	Serverless-архітектура
Структура	Один цілісний застосунок	Розподілена система з незалежних сервісів	Функції, що виконуються за подією
Масштабованість	Обмежена	Висока (масштабуються окремі сервіси)	Автоматична, гнучка

Простота реалізації	Висока на початковому етапі	Складна	Залежить від платформи
Час розгортання	Швидкий	Складніший, потребує оркестрації	Швидкий
Вимоги до інфраструктури	Сервер/хостинг	Контейнери	Хмарні сервіси
Відмовостійкість	Низька (помилка впливає на весь додаток)	Висока (відмова одного сервісу не зупиняє систему)	Висока, але залежить від хмари
Гнучкість у розробці	Низька	Висока	Середня
Вартість розгортання	Низька	Вища	Може бути високою за великого навантаження
Сумісність з REST API	Повна	Повна	Обмежена
Відповідність проекту	Обмежена	Оптимальна	Невиправдана складність і залежність від хмари

З огляду на вимоги до гнучкості, розширюваності та майбутньої інтеграції з зовнішніми сервісами, для реалізації платформи було обрано мікросервісну архітектуру. Вона дозволяє розділити систему на незалежні модулі: автентифікацію, керування конференціями, обробку статей.

1.4. Висновок до першого розділу.

У першому розділі було здійснено комплексний аналіз предметної області, що охоплює особливості сучасних академічних конференцій та їхню трансформацію під впливом цифровізації. Виявлено основні проблеми традиційної моделі проведення конференцій: географічні та логістичні обмеження, висока вартість організації та участі, низька швидкість поширення

наукових результатів, а також обмежена функціональність існуючих універсальних платформ, які не адаптовані до специфіки наукової комунікації. Це підтверджує актуальність розробки спеціалізованого веб-застосунку для проведення онлайн-конференцій, який би враховував потреби академічної спільноти.

Було проведено порівняльний аналіз популярних інструментів проведення конференцій, що засвідчив їхні обмеження щодо кастомізації, інтеграції з науковими сервісами та підтримки академічного документообігу. Також проаналізовано сучасні архітектурні підходи до побудови веб-додатків з точки зору масштабованості, продуктивності, гнучкості, вартості розгортання та відповідності вимогам проєкту. Проведене дослідження дало підстави для вибору мікросервісної архітектури як оптимальної для реалізації майбутньої системи.

На основі проведеного аналізу сформульовано такі основні завдання проєкту:

1) розробити архітектуру веб-платформи;

2) реалізувати механізми реєстрації та автентифікації користувачів з використанням сучасних стандартів;

3) забезпечити можливості адміністрування конференцій та обробки наукових статей;

4) інтегрувати функціонал для завантаження файлів та роботи з базою даних;

5) підібрати сучасні інструменти розробки back-end частини, що забезпечать високу продуктивність та гнучкість.

Таким чином, перший розділ заклав теоретичну базу для створення інноваційної платформи онлайн-конференцій, обґрунтував вибір технічного підходу та визначив напрямки розробки.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ BACK-END ЧАСТИНИ ПЛАТФОРМИ ОНЛАЙН-КОНФЕРЕНЦІЙ.

2.1. Вибір архітектури для реалізації проєкту.

Архітектура веб-застосунку визначає логічну структуру системи, способи її взаємодії з користувачами та іншими сервісами, а також правила обробки, зберігання та передачі даних. У сучасних умовах ефективна веб-архітектура повинна відповідати принципам модульності, масштабованості, безпеки, повторного використання компонентів та незалежності частин системи.

Веб-застосунок для організації онлайн-конференцій побудовано за класичною тришаровою архітектурною моделлю:

- Клієнтський рівень – забезпечує інтерфейс користувача, надсилає HTTP-запити до серверу, відображає результати. Може бути реалізований через веб-браузер або мобільний застосунок.
- Серверний рівень – відповідає за бізнес-логіку, обробку запитів, перевірку даних, автентифікацію користувачів та взаємодію з базою даних.
- Рівень даних – реалізує збереження даних користувачів, конференцій, статей, аватарів, історії авторизацій тощо.

Основними принципами, яких дотримано при побудові системи, є:

- Розділення відповідальності – кожен рівень виконує чітко визначену роль.
- Інкапсуляція логіки – бізнес-процеси ізольовані від деталей реалізації інтерфейсу.
- Безперервна інтеграція (CI/CD) – передбачає поступову розробку та розгортання без зупинки сервісу.
- RESTful-взаємодія – клієнт і сервер взаємодіють через HTTP-запити, обмінюючись JSON-даними.

Для реалізації серверної частини застосунку було обрано мікросервісну архітектуру, яка дозволяє створювати незалежні функціональні модулі (сервіси), що взаємодіють між собою через стандартизовані API. Такий підхід забезпечує:

- Масштабованість
- Незалежну розробку та тестування
- Високу відмовостійкість
- Гнучку інтеграцію

2.2. Функціональні вимоги до програмної реалізації.

Функціональні вимоги формують основу технічного завдання та визначають ключовий набір операцій, які має підтримувати серверна частина веб-платформи для організації онлайн-конференцій. Вони забезпечують зв'язок між очікуваннями кінцевого користувача та архітектурними рішеннями, що реалізуються під час розробки.

Система повинна підтримувати повноцінну взаємодію користувачів із сервісом через RESTful API, забезпечуючи всі необхідні функції для створення, адміністрування, управління матеріалами та участі в онлайн-конференціях.

До основних функціональних вимог належать такі:

1. Керування користувачами:

- Надання можливості реєстрації нових користувачів з валідацією імені, електронної пошти, пароля та, за потреби, аватару.
- Реалізація входу користувача із перевіркою облікових даних.
- Механізм генерації та перевірки авторизаційних токенів (JWT).
- Отримання поточних профільних даних авторизованого користувача.
- Захист доступу до приватних API за допомогою middleware авторизації.

2. Адміністрування конференцій:

- Створення конференцій.
- Перегляд списку всіх конференцій.
- Перегляд детальної інформації про окрему конференцію.
- Можливість редагування параметрів конференції.

- Видалення конференції з бази даних.

3. Обробка наукових статей:

- Додавання нових статей до конференції.
- Вивід списку всіх статей з можливістю фільтрації за конференцією.
- Перегляд, оновлення та видалення окремої статті.
- Прив'язка статей до користувачів та конференцій.

4. Завантаження файлів:

- Підтримка завантаження зображень та інших файлів.
- Зберігання файлів у захищеному файловому сховищі з формуванням публічних URL-адрес.
- Забезпечення відповідності імен файлів та структурованого доступу до них.

5. Валідація вхідних даних:

- Перевірка коректності форматів електронної пошти, дат, паролів, описів.
- Обов'язкова наявність необхідних полів у запитах.

6. Інтеграція з базою даних:

- Застосування документно-орієнтованої бази даних для зберігання користувачів, конференцій і статей.
- Визначення структурованих моделей даних з підтримкою timestamps.
- Реалізація CRUD-операцій через ORM або ODM-рішення.

7. Забезпечення безпеки

Таким чином, сформульовані функціональні вимоги визначають необхідний обсяг серверної логіки, яка повинна забезпечити коректну, безпечну та ефективну роботу веб-платформи для онлайн-конференцій. Вони слугують орієнтиром для розробника та основою для подальшого тестування, розширення і супроводу системи.

2.3. Технології для розробки back-end частини.

Вибір технологічного стеку для розробки back-end частини веб-додатку є критично важливим етапом, що визначає його функціональність, продуктивність, масштабованість, безпеку та вартість супроводу. Аналіз засобів реалізації здійснюється з урахуванням обраної мікросервісної архітектури, вимог до високої доступності, обробки значних обсягів даних та забезпечення надійного захисту інформації.

2.3.1. Аналіз середовищ виконання.

При виборі основного середовища виконання для back-end сервісів розглядалися наступні ключові варіанти:

– Node.js – це високопродуктивне серверне середовище виконання JavaScript, побудоване на рушії V8 Google Chrome та асинхронній, неблокуючій моделі вводу-виводу (libuv). Його архітектура дозволяє ефективно обробляти тисячі конкурентних з'єднань з мінімальними витратами системних ресурсів, що є критично важливим для високо-навантажених веб-додатків та мікросервісних архітектур. Node.js володіє великою та активною спільнотою, що сприяє швидкій еволюції екосистеми пакетів та забезпечує довгострокову підтримку (LTS).

– Python є популярною мовою програмування з високою читабельністю коду та великою кількістю готових бібліотек. Фреймворки, такі як Django (повний стек) або Flask (мікрофреймворк), дозволяють швидко розробляти веб-додатки. Однак, Python традиційно використовує синхронну модель обробки запитів (хоча існують асинхронні фреймворки, як FastAPI), що може бути менш ефективним для високонавантажених I/O-операцій.

– Java є зрілою, надійною та широко використовуваною мовою для корпоративних додатків. Фреймворк Spring Boot дозволяє швидко створювати незалежні, готові до виробництва додатки та мікросервіси. Перевагами Java є її висока продуктивність для CPU-інтенсивних завдань, потужна екосистема та

зрілість. Проте, вона часто вимагає більше системних ресурсів (пам'яті), має вищий поріг входу та може бути менш гнучкою в розробці швидких REST API.

Таблиця 2.1

Порівняльна характеристика технологій для розробки back-end

Критерій	Express.js	NestJS	Koa.js
Підхід до архітектури	Мінімалістичний, "unopinionated"	Строгий, "opinionated", модульна структура	Мінімалістичний, низькорівневий
Мова розробки	JavaScript (підтримка TypeScript доступна, але необов'язкова)	TypeScript (використовується за замовчуванням)	JavaScript / TypeScript
Складність освоєння	Низька, підходить для швидкого старту	Вища, вимагає знань концепцій Angular/DI	Середня, вимагає глибшого розуміння асинхронності
Підтримка middleware	Вбудована підтримка middleware (use())	Підтримка через модулі та декоратори	Потужна підтримка через async/await
Продуктивність	Висока (у поєднанні з простотою)	Висока (але більший розмір коду та складність)	Висока (завдяки відсутності "надлишкової логіки")
Структура проєкту	Вільна (розробник сам визначає структуру)	Суворо регламентована (модулі, контролери, DI)	Потрібно формувати власну структуру з нуля
Придатність для великих систем	Помірна, залежить від якості організації коду	Висока, спеціально спроєктований для масштабованих застосунків	Низька без додаткової надбудови
Крива навчання	Пологий старт	Крута, особливо для новачків у TypeScript	Вища, потрібен глибший контроль

Для реалізації back-end частини обрано Node.js. Цей вибір обумовлений його нативною підтримкою асинхронних операцій, що є ключовим для створення ефективних, масштабованих та стійких до навантажень веб-сервісів у мікросервісній архітектурі. Крім того, використання JavaScript як на фронтенді, так і на бекенді, сприяє уніфікації стеку розробки та спрощує обмін даними між клієнтом та сервером.

2.3.2. Аналіз веб-фреймворків

У контексті Node.js, для прискореної розробки веб-сервісів та API, було розглянуто кілька фреймворків:

- Express.js – це мінімалістичний та гнучкий веб-фреймворк для Node.js, який надає базовий набір функцій для створення веб-серверів та REST API. Він спрощує обробку HTTP-запитів, маршрутизацію та роботу з middleware. Його простота та "unopinionated" підхід дають розробнику велику свободу.
- NestJS – більш структурований та "opinionated" фреймворк, побудований на TypeScript, що використовує багато концепцій з Angular. Він ідеально підходить для великих корпоративних додатків, що вимагають високої архітектурної дисципліни. Проте його складність може бути надмірною для поточних потреб проекту.
- Koa.js – розроблений командою Express, Koa.js є більш сучасним та легким фреймворком, що використовує async/await для кращого контролю потоку виконання. Він більш низькорівневий, ніж Express, і вимагає більше зусиль для налаштування, що може уповільнити розробку на початкових етапах.

Таблиця 2.2

Порівняння фреймворків для Node.js

Критерій	Express.js	NestJS	Koa.js
Підхід до архітектури	Мінімалістичний, "unopinionated"	Суворий, "opinionated",	Мінімалістичний, низькорівневий

		модульна структура	
Мова розробки	JavaScript/TypeScript	TypeScript	JavaScript / TypeScript
Складність освоєння	Низька	Висока	Середня
Підтримка middleware	Вбудована підтримка middleware (use())	Підтримка через модулі та декоратори	Підтримка через async/await
Продуктивність	Висока	Висока	Висока
Структура проєкту	Вільна	модулі, контролери, DI	Вільна
Придатність для великих систем	Середня	Висока,	Низька

Для реалізації back-end обрано Express.js. Його мінімалізм, гнучкість, висока продуктивність та популярність для побудови REST API роблять його оптимальним вибором для цього проекту, забезпечуючи швидку та ефективну розробку необхідного функціоналу.

2.3.3. Аналіз систем управління базами даних

Вибір системи управління базами даних (СУБД) є стратегічно важливим етапом проєктування серверної частини платформи, оскільки саме вона визначає спосіб зберігання, організації та доступу до даних. Під час аналізу враховувалися такі критерії, як масштабованість, гнучкість, швидкодія, легкість інтеграції з обраним технологічним стеком, а також підтримка сучасних механізмів роботи з напівструктурованими даними.

У процесі відбору розглядалися два основні типи баз даних:

1) Реляційні бази даних (MySQL, PostgreSQL, MS SQL Server) традиційно використовуються для систем із чітко визначеною схемою та складними зв’язками між сутностями. Вони забезпечують високу цілісність даних,

підтримують транзакції з дотриманням принципів ACID, а також мають розвинені засоби для реалізації складних запитів. Проте реляційні СУБД демонструють обмежену гнучкість при зміні структури даних, значні витрати часу на міграцію схем, а також складнощі з горизонтальним масштабуванням, що може ускладнити розвиток системи в умовах зростаючого навантаження.

2) Нереляційні СУБД – сучасна альтернатива класичним реляційним системам, що охоплює кілька підтипів: документо-орієнтовані, сховища типу "ключ-значення", колоночні та графові бази. Особливістю цих рішень є відмова від фіксованої схеми зберігання, що дозволяє зберігати гнучко структуровані дані з мінімальними витратами на трансформацію моделей.

В якості основної бази даних для зберігання інформації про конференції, користувачів та статті обрано MongoDB. Цей вибір обумовлений її документо-орієнтованою моделлю, яка забезпечує природне відображення об'єктних структур у JavaScript-коді, що є особливо зручним для Node.js. Гнучкість схеми MongoDB і висока швидкість роботи з даними відповідають вимогам сучасних веб-сервісів та мікросервісної архітектури.

2.4. Висновок до другого розділу

У другому розділі проведено комплексний аналіз засобів реалізації back-end частини платформи для онлайн-конференцій, що охоплює архітектурні принципи, функціональні вимоги та вибір технологічного стеку. З метою забезпечення гнучкості, масштабованості, високої доступності та безпеки системи, в основі архітектури платформи закладено мікросервісний підхід із чітким розподілом відповідальності між сервісами. Така структура дозволяє незалежну розробку, тестування та масштабування окремих компонентів, що є критично важливим для підтримки великої кількості користувачів та динамічного розширення функціональності.

Побудова серверної частини здійснюється відповідно до тришарової моделі: клієнтський рівень, рівень бізнес-логіки та рівень збереження даних. В

основі взаємодії між клієнтом і сервером покладено RESTful-підхід, який передбачає стандартизований обмін JSON-даними через HTTP-запити. Такий підхід забезпечує прозору інтеграцію з фронтендом, мобільними клієнтами та зовнішніми сервісами.

У функціональних вимогах визначено ключові можливості системи: керування користувачами, адміністрування конференцій, обробка наукових статей, завантаження файлів, валідація вхідних даних, а також реалізація повноцінного механізму безпеки з використанням JWT-токенів, middleware-аутентифікації та контроль доступу до приватних маршрутів.

Реалізації серверної логіки обрано платформу Node.js з Express.js, яка завдяки неблокуючій моделі вводу/виводу забезпечує високу продуктивність і ефективну роботу з одночасними запитами. Як основну базу даних обрано MongoDB, що природно інтегрується з JavaScript-орієнтованим стеком та надає гнучку модель зберігання, що особливо актуально в умовах змінних вимог і активного зростання системи.

Таким чином, вибрані архітектурні рішення, технології та інструменти забезпечують надійну основу для подальшої розробки, розгортання та супроводу серверної частини платформи онлайн-конференцій. Результати аналізу доводять відповідність обраного підходу сучасним стандартам веб-розробки та забезпечують реалізацію масштабованої, безпечної і функціонально повної системи, здатної задовольнити вимоги академічної спільноти.

РОЗДІЛ 3. РОЗРОБКА BACKEND ЧАСТИНИ ВЕБ-ДОДАТКУ ДЛЯ СТВОРЕННЯ ОНЛАЙН-КОНФЕРЕНЦІЙ.

3.1. Актуальність розробки проекту.

Стрімке розширення інформаційно-комунікаційних технологій та зростання попиту на дистанційні форми наукової комунікації зумовили кардинальну трансформацію традиційного формату конференцій. Глобальні обмеження мобільності, економічні чинники та потреба в оперативному обміні результатами досліджень актуалізували розроблення веб-платформ, здатних забезпечити повний життєвий цикл академічного заходу в режимі онлайн. У цьому контексті створення backend-частини веб-додатку для організації онлайн-конференцій є ключовим, оскільки саме серверна складова відповідає за стабільність, масштабованість і безпеку критичних процесів — від реєстрації користувачів до збереження матеріалів і керування сесіями.

Існуючі комерційні рішення часто мають обмежені можливості кастомізації, високу вартість ліцензування та недостатню інтеграцію з бібліографічними й репозитарними сервісами академічних інституцій. Відтак постає необхідність у розробці спеціалізованого backend-модуля, який би реалізував відкриті API для створення конференцій із гнучким налаштуванням назви, опису та параметрів доступу, а також підтримував завантаження й рецензування статей користувачами відповідно до тематики вибраної конференції. Запровадження такої системи дозволяє забезпечити прозорість редакційних процедур, оптимізувати роботу програмних комітетів і суттєво скоротити організаційні витрати.

З технічного погляду серверна частина повинна гарантувати стійкість до високих навантажень, ідентифікацію та авторизацію користувачів, захист конфіденційних даних відповідно до сучасних стандартів (OWASP, GDPR), а також підтримувати модульність і розширюваність за рахунок мікросервісної архітектури та контейнеризації. Реалізація RESTful забезпечить сумісність з

фронтенд-клієнтом, мобільними застосунками та сторонніми інформаційними системами, що уможливить подальшу інтеграцію з базами даних, системами антиплагіатної перевірки та цифровими бібліотеками.

Таким чином, розробка backend-частини веб-додатку для онлайн-конференцій є стратегічно необхідною для формування стійкої інфраструктури академічних комунікацій у цифрову епоху. Вона відкриває доступ до участі у наукових заходах широкому колу дослідників, сприяє оперативному поширенню знань і формує передумови для подальшої еволюції онлайнових форм співпраці, що відповідає сучасним вимогам відкритої науки та глобальної освітньо-наукової екосистеми.

3.2. Компоненти розробки

Основним компонентом для створення backend- середовища є Node.js.

Node.js (рис. 3.1) — це високопродуктивне серверне середовище виконання JavaScript, побудоване на рушії V8 і асинхронній моделі вводу-виводу libuv, що забезпечує неблокувальне обслуговування тисяч конкурентних з'єднань з мінімальними витратами системних ресурсів. Його модульна система ESM/CJS, вбудовані механізми роботи з потоками та кластеризацією, а також підтримка сучасних API (Streams, Web Crypto, Worker Threads) формують технологічне підґрунтя для розробки масштабованих мікросервісних архітектур. Стратегія довгострокової підтримки (LTS) гарантує стабільність критичних оновлень безпеки, а активне ядро спільноти сприяє швидкій еволюції екосистеми пакетів, що робить Node.js ключовим компонентом серверної частини сучасних веб-додатків.

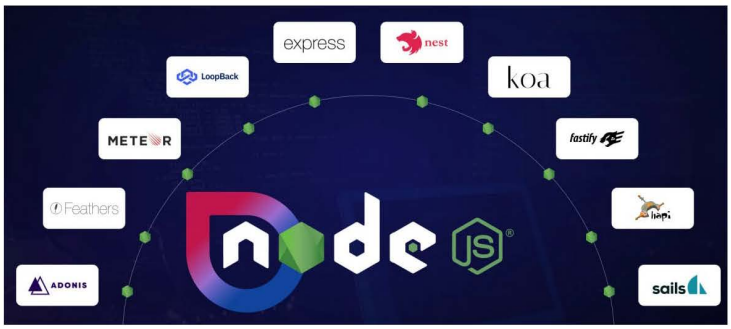

Рисунок 3.1 – Node.js та його компоненти

Express.js — це мінімалістичний і гнучкий веб-фреймворк для Node.js, який дозволяє швидко створювати веб-сервери та REST API. Він спрощує обробку HTTP-запитів, маршрутизацію, роботу з middleware, а також легко інтегрується з базами даних. Express широко використовується для створення бекенд-частини сучасних веб-застосунків завдяки своїй простоті, великій спільноті та розширюваності.

Express-validator — це бібліотека для Node.js, яка забезпечує зручний спосіб перевірки (валідації) та нормалізації вхідних даних у запитах до серверу, написаного з використанням Express.js. Вона побудована на базі популярної бібліотеки validator.js і дозволяє застосовувати різні перевірки (наприклад, перевірку email, дати, URL, чисел тощо) без необхідності писати вручну багато умов.

Бібліотека працює як middleware, що вставляється перед обробником маршруту, і дозволяє задати правила валідації за допомогою зручного ланцюгового API (.isEmail(), .isLength(), .notEmpty(), тощо). Після цього можна перевірити результат через validationResult(req), і в разі помилок — вивести або надіслати детальні повідомлення про помилки клієнту. Це суттєво підвищує безпеку та стабільність API, зменшуючи ризик обробки некоректних чи зловмисних даних.

Рисунок 3.2 – Компонент Express-validator

Nodemon слугує інструментом підвищення продуктивності під час локальної розробки: утиліта відстежує зміни у вихідних файлах і автоматично перезапускає Node-процес, усуваючи потребу в ручному циклі «зупинка — запуск». Такий перезапуск скорочує час зворотного зв'язку, сприяє швидкому виявленню дефектів і підтримує безперервний інтеграційний потік. Гнучкі правила спостереження, конфігурація через nodemon.json та сумісність із Docker-контейнерами дозволяють органічно включати nodemon у CI/CD-конвеєри, зберігаючи при цьому чистоту оточення від зайвих залежностей.

Синергія Node.js, Express.js, express-validator та nodemon формує цілісне середовище, у якому високопродуктивна обробка запитів поєднується з декларативною перевіркою даних і ефективним циклом розробки. Такий набір компонентів дозволяє будувати масштабовані, безпечні та легко супроводжувані серверні застосунки, що відповідають сучасним вимогам до якості програмного забезпечення й швидкості його доставки користувачам.

В якості бази даних для зберігання інформації про конференції, зареєстрованих користувачів та статті, виступає MongoDB.

MongoDB — це документо-орієнтована нереляційна база даних, яка зберігає інформацію у форматі BSON-документів, логічно спорідненому з JSON, що забезпечує природне відображення об'єктних структур в JavaScript-коді без необхідності складних трансформацій. Архітектура «schema-less» дозволяє динамічно корегувати схему даних відповідно до потреб структури моделі, скорочуючи витрати на модифікацію таблиць і міграцію даних, властиві

традиційним реляційним системам. Вбудовані механізми індексування, агрегацій та підтримка транзакцій на рівні документів (multi-document transactions) забезпечують засоби для складного аналітичного та оперативного запиту, одночасно гарантують цілісність даних у критичних бізнес-процесах.

Рисунок 3.3 – База даних MongoDB

У контексті серверної частини, побудованої на Node.js і Express.js, MongoDB інтегрується безпосередньо через офіційний високопродуктивний драйвер **Mongoose**, який надає декларативні схеми, валідатори та middleware-hooks. Така інтеграція дозволяє інкапсулювати логіку доступу до даних, забезпечити централізовану обробку та зберігання при цьому залишаючи переваги асинхронної моделі Node.js.

Інструментарій Atlas або on-premise-кластеризація з optional-TLS/SSL шифруванням та role-based access control сприяють відповідності вимогам безпеки й конфіденційності, зокрема стандартам GDPR (General Data Protection Regulation), що важливо для обробки персональних даних учасників та їхніх наукових статей.

Таким чином, використання MongoDB у зазначеному технологічному стеку формує гнучкий і масштабований шар взаємодій, який поділяє філософію неформалізованої схеми даних, властиву сучасним веб-сервісам, та забезпечує високу швидкодію й прозору інтеграцію з асинхронними обчисленнями Node.js. Це дає змогу реалізувати стійку до навантажень інфраструктуру збереження конференційних метаданих і матеріалів, а також сприяє подальшій еволюції системи без кардинальних змін у базових концептах її зберігання.

MongoDB Compass використовується в проєкті як офіційний графічний інструмент дослідження та обробки даних, що надає розробникам візуальний інтерфейс для безпосередньої взаємодії з колекціями й індексами.

3.3 Розробка backend-серверу

Для початку потрібно створити стартовий конфігураційний файл для завантаження backend. Для цього слугує файл index.js. Даний скрипт представляє собою модуль серверної частини веб-застосунку, збудований на базі Node.js та Express.js, який реалізує REST-архітектурний інтерфейс для керування користувачами, статтями та конференціями.

```
import express from 'express';
import mongoose from 'mongoose';
import multer from 'multer';
import cors from 'cors';

import {registerValidation, loginValidation, postCreateValidation, conference

import checkAuth from './utils/checkAuth.js';

import * as UserControllers from './Controllers/UserController.js'
import * as PostControllers from './Controllers/PostController.js'
import * as ConferensController from './Controllers/ConferenseController.js'

import handleErrorsValidation from './utils/handleErrorsValidation.js';

mongoose.connect('mongodb://localhost:27017/Node').
```

Рисунок 3.4 – Початок коду index.js

На початку підключаються необхідні залежності: Express для створення HTTP-серверу, Mongoose для взаємодії з базою даних MongoDB, Multer для зберігання завантажених файлів, а також CORS для розв'язання проблеми міждоменного доступу. Окремо імпортуються набори правил валідації, допоміжні утиліти для перевірки автентифікації та обробки помилок, а також контролери, що реалізують бізнес-логіку кожного функціонального модуля.

Під'єднання до MongoDB відбувається за допомогою mongoose.connect (рис. 3.5), у випадку успішного з'єднання виводиться повідомлення «DB ok», а в

разі виникнення помилки — «DB error», такимспособом забезпечується базова діагностика стану підлючення. Після попередніх дій створюється екземпляр застосунку Express, конфігуруються глобальні middleware: express.json() для зчитування JSON-тіла запитів і cors() для дозволу зовнішніх звернень.

```
mongoose.connect('mongodb://localhost:27017/Node').
then(()=> console.log('DB ok')
).catch(()=> console.log('DB error', err))
const app = express();
app.use(express.json());
app.use(cors());
const storage = multer.diskStorage({
    destination: (_, __,cb)=>{
        cb(null,'uploads')
    },
    filename:(_,file,cb)=>{
        cb(null,file.originalname)
    }
})

const upload  = multer({storage});
```

Рисунок 3.5 – Ініціалізація MongoDB та Multer

Наступним кроком визначається диск-сховище Multer (рис. 3.5): усі файли, що надходять з frontend середовища, розміщуються в директорії uploads, збергіаючи оригінальне ім'я. Це рішення спрощує подальший доступ до ресурсів і забезпечує передбачуваність шляху. Додатково маршрути express.static('uploads') дозволяють надавати файли з відповідної дерикторії, а POST-маршрут /upload — завантажувати одне зображення та одразу повертати клієнту сформовану URL-адресу.

Блок авторизації включає три маршрути: реєстрацію (/auth/register), вхід (/auth/login) і виведення інофрмації про поточного користувача (/auth/me). Кожна кінцева точка використовує свою схему валідації даних, загальний фільтр handleErrorsValidation для уніфікованого перехоплення помилок вводу та, де потрібно, middleware checkAuth, перевіряє наявність дійсного JWT-токена. Такий підхід чітко розмежовує відповідальність і мінімізує дублювання коду (рис. 3.6).

Рисунок 3.6 – Блок маршрутів реєстрації, авторизації, виведення даних

Для управління статтями використовуються CRUD-маршрути: створення статті у межах конкретної конференції (/article/create/:conID), отримання списку всіх статей, зчитування, оновлення та видалення статті за її ідентифікатором. Аналогічні дії реалізовано для конференцій: створення, перелік, детальний перегляд, часткове оновлення та повне видалення. Усі операції зміни даних (POST, PATCH, DELETE) захищені аутентифікацією й супроводжуються відповідними схемами валідації, що гарантує коректність і цілісність записів у базі.

Рисунок 3.7 – CURD операції для статей та конференцій

Наприкінці викликається app.listen на порту 4444. Функція перевіряє наявність помилки запуску, фіксуючи відповідні записи в консолі. Таким чином, код формує повноцінний сервер, у якому реалізовані ключові компоненти для керування контентом онлайн-конференцій, забезпечено взаємодію з базою даних, зберігання файлів, процедури автентифікації й валідації та базові механізми відслідковування подій.

Розглянемо більш детально CRUD-операції (рис. 3.8). CRUD-операції (Create, Read, Update, Delete) становлять фундаментальний набір дій, якими керується життєвий цикл будь-якого сутнісного об'єкта в інформаційній системі. На етапі Create відбувається ініціалізація нової сутності: сервер приймає дані від клієнта, пропускає їх через каскад валідаційних правил, формує документ згідно зі схемою доменної моделі та зберігає його у базі даних. У контексті REST-архітектури ця операція відповідає методу HTTP POST.

Операція Read забезпечує доступ до наявних записів, реалізовується за допомогою протоколу HTTP GET та виконується у вигляді запитів collection.find() або Model.find()/Model.findById(), які, за потреби, можуть доповнюватися фільтрацією.

Update змінює існуючу сутність. У MongoDB це реалізовано командами updateOne() чи updateMany(), у Mongoose — методами findByIdAndUpdate() або updateOne(). Виконується через протокол HTTP PATCH.

Delete відповідає за безповоротне видалення ресурсу через протокол HTTP DELETE. У MongoDB операція виконується через deleteOne()/deleteMany(), а у Mongoose — findByIdAndDelete() чи deleteOne().

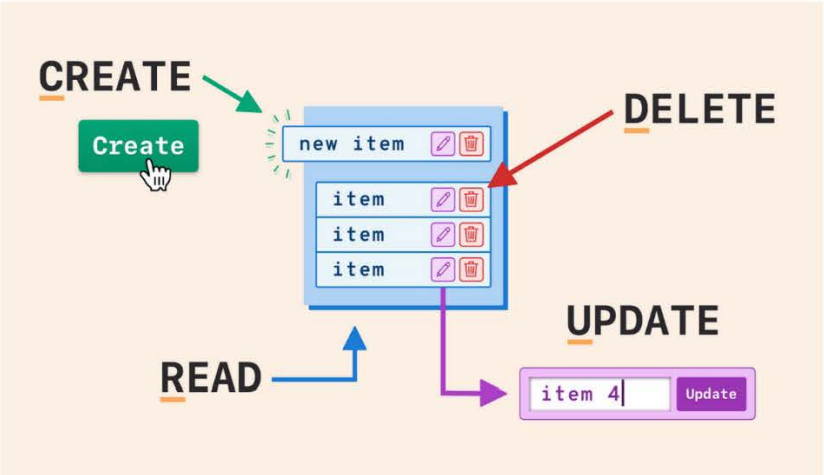

Рисунок 3.8 – CRUD-операції

Разом ці чотири операції формують замкнений набір, який, поєднаний із валідацією, авторизацією та логуванням, забезпечує повний цикл керування

даними й підтримує принципи ACID/BASE залежно від обраної стратегії персистентності.

Один з проміжних етапів, через що проходить виконання CRUD операцій є скрипт checkAuth.js.

Фрагмент даного коду реалізує (рис. 3.9) проміжний компонент (middleware) перевірки автентичності запитів на основі технології JSON Web Token. На початковому етапі модуль імпортує бібліотеку jsonwebtoken, після чого експортує анонімну функцію, призначену для перевірки токену під час обробки HTTP-звернень—фактично, це тонка оболонка, яка контролює доступ до частини запитів за наявності токена авторизованого користувача.

```
import jwt from 'jsonwebtoken';

export default (req,res, next)=>{

    const token = (req.headers.authorization || '').replace(/Bearer\s?/,'');
    if(token)
    {
        try{
            const decodeToken = jwt.verify(token, 'secret123');
            req.userId = decodeToken._id;
            next();
        } catch(e) {
            return res.status(403).json({
            message: "Нема доступу"
            });
        }
    }
    else{
        return res.status(403).json({
            message: "Нема доступу"
        });
    }
```

Рисунок 3.9 – Фрагмент коду checkAuth.js

Алгоритм роботи полягає у витягуванні значення Header Authorization за допомогою регулярного виразу вилучається префікс Bearer і формується чистий токен. Якщо такий рядок існує, виконується спроба його декодування методом jwt.verify, що застосовує симетричний код «secret123». У разі успіху зі структури токена дістається ідентифікатор користувача _id, який зберігається у властивості req.userId, — це забезпечує прозоре передавання даних зареєстрованого суб'єкта до наступних запитів й спрощує авторизацію на рівні контролерів.

Ситуації, коли токен відсутній або не проходить криптографічну перевірку цілісності (підпису й строків дії), обробляються єдиним механізмом: клієнтові повертається відповідь із кодом HTTP 403 Forbidden та уніфікованим повідомленням «Нема доступу». Завдяки такій політиці відхилення «за замовчуванням» захищає від витоку даних реалізації та встановлюються найнижчі за рівнем права отримання даних.

Далі наступний скрипт створює модель User для MongoDB – User.js використовує бібліотеку Mongoose, що виконує роль об'єктно-документного відображення даних (ODM) у середовищі Node.js. Спочатку імпортується ядро Mongoose, після чого формується схема UserSchema, яка визначає структуру документів колекції та характерніатрибути кожного поля. Атрибут fullName описує обов'язковий рядок - повне ім'я в кожному записі; аналогічно параметр email не лише позначений як required (тобто обов'язковий), а й визначений як unique, що гарантує унікальність електронних адрес і запобігає дублюванню акаунтів (рис. 3.10).

```
import mongoose from "mongoose";

const UserSchema = new mongoose.Schema({
    fullName:{
        type: String,
        required: true,
    },
    email:{
        type: String,
        required: true,
        unique: true,
    },
    passwordHash:{
        type: String,
        required: true,
    },
    avatarUrl: String,
},{
    timestamps: true,
},)

export default mongoose.model('User', UserSchema);
```

Рисунок 3.10 – Модель User

Поле passwordHash призначене для зберігання криптографічно хешованого пароля, підкреслюючи вимоги до безпеки: у сховище потрапляє не «чистий» пароль, а його хеш-значення, що зменшує ризик взлому облікових даних.

Додатковий атрибут avatarUrl оголошено простим рядком та не обов'язковим, дозволяючи користувачеві опціонально додати посилання на аватар (власне фото). У другому аргументі конструктора схеми задається об'єкт конфігурації timestamps: true, який автоматично включає поля createdAt і updatedAt, що спрощує аудит змін і дає змогу легко відслідковувати час створення та останнього оновлення документа.

Далі розглянемо скрипт, що відповідає за створення тобто реєстрацію користувача, автентифікації та відображення основних даних про обліковий запис, це UserController.js.

Перша функція register (рис. 3.11) – асинхронна функція реалізує повний цикл реєстрації користувача, поєднуючи криптографічний захист облікових даних, збереження інформації в базі та видачу маркера автентифікації.

```
export const register = async (req,res)=>{

    try{
    const password = req.body.password;
    const salt = await bcrypt.genSalt(10);
    const hash = await bcrypt.hash(password, salt);
    const doc = new UserModel({
        email:req.body.email,
        fullName:req.body.fullName,
        avatarUrl:req.body.avatarUrl,
        passwordHash: hash,
    })
    const  user = await doc.save()
    const token = jwt.sign({
        _id:user._id,
    },
    'secret123',
    {
        expiresIn: "30d",
    },);
    const {passwordHash, ...userData} = user._doc;
    res.json({
        ...userData,
        token
    })
    }
    catch(err){
        console.log(err);
        res.status(500).json({
            message:"Не вдалося зареєструватися",
        });
    }
};
```

Рисунок 3.11 – Функція register

Спочатку вона вилучає пароль із тіла HTTP-запиту, генерує криптографічну «сіль» (алгоритм шифрування) (bcrypt.genSalt) і формує стійкий хеш (bcrypt.hash), замість відкритого пароля в сховищі. Далі функція створює

екземпляр моделі UserModel, передаючи електронну пошту, повне ім'я, опціонально URL-адресу аватара й отриманий хеш, після чого асинхронно зберігає документ у MongoDB через doc.save().

Після успішного запису генерується JWT-токен (jwt.sign) із payload, що містить ідентифікатор створеного користувача, симетричним ключем «secret123» і терміном дії 30 діб. Цей токен слугує засобом подальшої автентифікації запитів. Перед відправленням відповіді функція поділяє об'єкт користувача, вилучаючи поле passwordHash, аби не розкривати його клієнтові через запит, і повертає JSON зі стерильними даними користувача та новоствореним токеном. У разі будь-якого винятку помилки виводиться на серверну консоль, а клієнт отримує статус 500 і стандартизоване повідомлення про неможливість завершити реєстрацію, що забезпечує узгоджену поведінку при виникненні помилки.

Далі функція login реалізує автентифікацію користувача на основі комбінації електронної пошти та пароля з використанням асинхронних операцій Mongoose і криптографічної бібліотеки bcrypt.

На першому етапі виконання відбувається запит до колекції користувачів: метод findOne шукає документ, де поле email збігається з електронною адресою, що отримано з тіла HTTP-запиту (рис. 3.12). Якщо відповідний об'єкт не знайдено, сервер одразу повертає статус 404 Not Found із повідомленням «Користувача не знайдено», тим самим перериваючи подальшу обробку й уникнувши зайвих обчислень.

```
export const login = async (req,res)=>{
    try{
        const user = await UserModel.findOne({email:req.body.email});
        if(!user)
        {
            return res.status(404).json({
                message:'Користувача не знайдено',
            });
        }
```

Рисунок 3.12 – Пера частина функції login

Коли документ успішно знайдено, функція переходить до перевірки достовірності пароля. Для цього використовується bcrypt.compare, який порівнює переданий у запиті «чистий» пароль з хешем, збереженим у полі passwordHash. У разі невідповідності парольних даних формується відповідь зі статусом 400 Bad Request та уніфікованим повідомленням «Невірний логін чи пароль», що відображає логічну помилку користувача, а не збій системи (рис. 3.13).

```
const isValidPassword = await bcrypt.compare(req.body.password, user._doc.passwordHash);
if(!isValidPassword)
{
    return res.status(400).json({
        message:'Невірний логін чи пароль',
    });
}
const token = jwt.sign({
    _id:user._id,
},
    'secret123',
{
    expiresIn: "30d",
},);
const {passwordHash, ...userData} = user._doc;
res.json({
    ...userData,
    token
})
```

Рисунок 3.13 – Друга частина функції login

Після успішної валідації пароля генерується JWT-токен за допомогою jwt.sign. У payload токена вміщується ідентифікатор користувача _id, підпис здійснюється кодуванням «secret123», а строк дії встановлюється на 30 діб. Для забезпечення безпеки, об'єкт користувача очищається від важливої інформації: поле passwordHash вилучається, а решта даних передається клієнтові разом із токеном. Тим самим досягається принцип мінімального розголошення чутливої інформації, а клієнт одержує повноваження на подальші запити.

Блок catch охоплює всі виняткові ситуації, що можуть виникнути на будь-якому кроці процесу, зокрема помилки мережевого доступу до бази чи внутрішні помилки криптографічних операцій. У разі помилки дані виводяться на серверній консолі, а клієнтові повертається статус 500 Internal Server Error з повідомленням «Не вдалося авторизуватися». Таким чином, функція забезпечує

узгоджену й безпечну модель входу в систему, обробляючи кожен етап — від пошуку користувача до надання токена — із чітким розмежуванням успішних і помилкових сценаріїв.

Функція getMe (рис. 3.14) використовується для отримання профілю поточного аутентифікованого користувача й реалізує захищений доступ до персональних даних через асинхронні можливості Mongoose. На початку обробки з об'єкта req вилучається ідентифікатор userId, який зазвичай отримується за допомогою скрипту перевірки токена (middleware checkAuth). Метод UserModel.findById здійснює прямий пошук у колекції users за ідентифікатором.

```
export const getMe = async (req, res)=>{
    try{
        const user = await UserModel.findById(req.userId)
        if(!user)
        {
            return res.status(404).json({
                message:'Користувач не знайдений'
            })
        }
        const {passwordHash, ...userData} = user._doc;
        res.json({
            ...userData,
        })
    } catch(err) {
        console.log(err);
        res.status(500).json({
            message:"Відсутній доступ",
        });
    }
};
```

Рисунок 3.14 – Функція getMe

Якщо жоден документ не відповідає критерію, функція перериває виконання й повертає клієнтові статус 404 Not Found із повідомленням «Користувач не знайдений». Це дає змогу коректно відреагувати на ситуацію, коли токен формально коректний, але відповідного запису вже не існує (наприклад, його було видалено адміністратором).

У випадку успішного пошуку виконується деструктуризація результату: поле passwordHash, що містить чутливий хеш пароля, вилучається, а решта властивостей (_id, email, fullName, avatarUrl, createdAt, updatedAt) формується у

відповідь. Такий підхід дотримується принципу мінімального розголошення й унеможливлює витік криптографічного матеріалу.

Неодноразово можна спостерігати, що в коді використовується JSON Web Token (JWT). Особливість його полягає у тому, що це самодостатній, криптографічно підписаний носій атрибутів, який дає змогу реалізувати гнучку схему безпеки. Кожен токен складається із трьох базово 64-кодованих частин — назви, корисного навантаження (claims) і цифрового підпису — що разом утворюють так званий «паспорт» користувача: після одноразової автентифікації сервер створює підпис, що усуває потребу постійно звертатися до бази даних для перевірки сесії, адже вся необхідна інформація (ідентифікатор, роль, термін дії) уже міститься всередині токена. У коді це принципово спрощує логіку захищених маршрутів: middleware checkAuth зчитує заголовок Authorization, вилучає рядок Bearer <token>, верифікує підпис за симетричним кодом «secret123» і, не виконуючи додаткових запитів до сховища, одразу передає у req.userId достовірний _id, перевірений на цілісність і строк придатності.

Ще одна важлива риса — можливість закладати час життя (claim exp), отже, природну ротацію облікових даних без централізованого відкликання сесій: у функціях login та register створюється токени з параметром expiresIn: "30d", що зобов'язує клієнта автоматично оновити або повторно отримати маркер після тридцяти днів. Оскільки JWT може бути підписаний як симетричним ключем (HMAC), так і парою відкритий/приватний (RSA чи ECDSA), схема легко масштабується до мікросервісних: будь-який компонент, маючи лише публічний ключ, може незалежно використовувати токен, зберігаючи спільну зону довіри без централізованого сховища сесій.

Далі розглянемо роботу зі структурою даних як Conference.js, що використовується для зберігання даних про конференцію в MongoDB.

Схема ConferenceSchema описує обов'язкові атрибути title, description, date та organizer, які забезпечують мінімальний інформаційний каркас події: назву, текстовий опис, дату проведення та організатора. Поле location має стандартне значення Online, що, з одного боку, підтримує модель «за замовчуванням

онлайн», а з іншого — дозволяє легко змінити місце проведення, якщо формат стане гібридним чи офлайн. Колекція articles визначена як масив об'єктів ObjectId, посилаючись на модель Article,— це реалізує логічний зв'язок «одна-багато» між конференцією та науковими статтями й дає змогу здійснювати попереднє завантаження (populate) даних для агрегованих запитів (рис. 3.15).

```
import mongoose from 'mongoose';

const ConferenceSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  date: {
    type: Date,
    required: true,
  },
  location: {
    type: String,
    default: 'Online',
  },
  organizer: {
    type: String,
    required: true,
  },
  articles: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Article',
  }],
  zoomLink: {
    type: String,
  },
  coverImageUrl: {
    type: String,
  },
  isActive: {
    type: Boolean,
    default: true,
```

Рисунок 3.15 – Модель Conference

Додаткові поля zoomLink та coverImageUrl надають можливість інтегрувати зовнішні ресурси: посилання на відеоконференцію та URL обкладинки, забезпечуючи зручний доступ для користувачів і візуальну ідентифікацію події. Булева ознака isActive із визначеним значенням true дозволяє швидко вимикати/архівувати конференції без фізичного видалення документа.

Далі розглянемо основні CURD операції, пов'язані з роботою над конференціями в скрипті ConferenseController.js.

Функція create (рис. 3.16) забезпечує серверний механізм ініціювання нової сутності «Конференція» в базі даних MongoDB, використовуючи модель ConferenceModel, що реалізує схему документно-орієнтованого зберігання. На першому етапі з тіла HTTP-запиту вилучаються всі параметри, релевантні доменній моделі — назва, опис, дата, місце проведення, організатор, посилання на Zoom-кімнату й URL обкладинки. Ці дані інкапсулюються в об'єкті doc, створеному через конструктор моделі, що одночасно запускає валідаційний шар Mongoose і перевіряє відповідність значень вимогам схеми (обов'язковість полів, формат дати тощо).

```
// CREATE
export const create = async (req, res) => {
  try {
    const doc = new ConferenceModel({
      title: req.body.title,
      description: req.body.description,
      date: req.body.date,
      location: req.body.location,
      organizer: req.body.organizer,
      zoomLink: req.body.zoomLink,
      coverImageUrl: req.body.coverImageUrl,
    });

    const conf = await doc.save();
    res.json(conf);
  } catch (err) {
    res.json(err)
    res.status(500).json({ message: 'Не вдалося створити конференцію' });
  }
};
```

Рисунок 3.16 – Функція створення нової конференції

Далі за допомогою асинхронного виклику doc.save() документ фіксується в колекції, а повернене значення conf містить уже згенерований первинний ключ _id, а також автоматичні мітки createdAt та updatedAt. У разі успіху функція одразу надсилає клієнтові JSON-репрезентацію створеної конференції, що забезпечує миттєвий зворотний зв'язок і підтвердження операції.

Функція update (рис. 3.17) використовується для часткового оновлення даних конференції у сховищі MongoDB, спираючись на модель ConferenceModel і метод updateOne. Спершу з параметрів маршруту (req.params.id) вилучається унікальний ідентифікатор конференції confId. Далі викликається updateOne, де

умова відбору { _id: confId } гарантує, що модифікація торкнеться лише документа з відповідним первинним ключем, уникаючи небажаного множинного оновлення. У другому аргументі передається об'єкт нових значень, сформований із тіла запиту: назва, опис, дата, локація, організатор, посилання на Zoom і URL для обкладинки.

```
export const update = async (req, res) => {
  try {
    const confId = req.params.id;
    console.log(confId);
    const newdoc = await ConferenceModel.updateOne(
    {
        _id:confId,
    },{
        title: req.body.title,
        description: req.body.description,
        date: req.body.date,
        location: req.body.location,
        organizer: req.body.organizer,
        zoomLink: req.body.zoomLink,
        coverImageUrl: req.body.coverImageUrl,
      },
    );
    console.log(newdoc)
     res.json({
            success: true,
            message: "данні конференції оновлено",
     });

  } catch (err) {
    console.log(err)
    res.status(500).json({ message: 'Не вдалося оновити' });
  }
};
```

Рисунок 3.17 – Функція update

Результат операції, повернений у змінну newdoc, містить службову інформацію про кількість змінених документів і статус виконання. Після завершення функція повертає клієнтові JSON-повідомлення з ознакою success: true і підтвердженням «данні конференції оновлено».

Функції getAll та getOne відповідають за виведення даних, але мають певні відмінності в кількості отриманні інформації.

Перша функція getAll (рис. 3.18) реалізує операцію читання всієї колекції конференцій. Вона асинхронно викликає метод ConferenceModel.find(), що повертає масив документів без жодних фільтрів або сортування, забезпечуючи повний перелік наявних подій. Отримані дані негайно переводяться у формат JSON і надсилаються клієнтові, що дозволяє фронтенду відобразити каталог конференцій у вигляді списку. Якщо під час виконання запиту виникає виняток

— наприклад, збої мережевого доступу до MongoDB або внутрішня помилка Mongoose — обробник catch повертає відповідь зі статусом 500 Internal Server Error і уніфікованим повідомленням про неможливість отримати список.

```
// READ ALL
export const getAll = async (req, res) => {
  try {
    const confs = await ConferenceModel.find();
    res.json(confs);
  } catch (err) {
    res.status(500).json({ message: 'Помилка при отриманні списку' });
  }
};
```

Рисунок 3.18 – Функція getAll

Друга функція getOne (рис. 3.19) призначена для вибіркового зчитування однієї конференції за первинним ключем, переданим за допомогою параметром маршруту req.params.id. Вона використовує метод findById, який забезпечує прямий доступ до документа за _id, і застосовує ланцюгову операцію populate. Спершу заповнюється поле articles, що містить об’єкти ObjectId, а потім через вкладений populate витягуються дані автора кожної статті; при цьому селектор select: 'fullName email' обмежує вибірку лише до суттєвих атрибутів, мінімізуючи обсяг переданої важливої інформації й дотримуючись принципу мінімального розголошення. Якщо конференцію не знайдено, сервіс повертає статус 404 Not Found з інформативним повідомленням, що унеможливлює помилкову обробку порожнього результату на клієнті.

```
export const getOne = async (req, res) => {
  try {
    const conf = await ConferenceModel.findById(req.params.id).populate({
      path: 'articles',
      populate: { path: 'author', select: 'fullName email' },
    });

    if (!conf) {
      return res.status(404).json({ message: 'Конференцію не знайдено' });
    }

    res.json(conf);
  } catch (err) {
    res.status(500).json({ message: 'Не вдалося отримати конференцію' });
  }
};
```

Рисунок 3.19 – Функція getOne

Функція remove (рис. 3.20) реалізує видалення конференції з бази даних MongoDB, використовуючи асинхронний метод ConferenceModel.findByIdAndDelete, який одразу шукає документ за первинним ключем і, якщо знайдений, фізично виключає його з колекції. У випадку, коли за наданим req.params.id об'єкту не існує, сервер повертає статус 404 Not Found із повідомленням «Не знайдено конференцію»

```
export const remove = async (req, res) => {
  try {
    const conf = await ConferenceModel.findByIdAndDelete(req.params.id);
    if (!conf) return res.status(404).json({ message: 'Не знайдено конференцію' });

    res.json({
          success: true,
          message: "Конференцію видалено",
        });
  } catch (err) {
    res.status(500).json({ message: 'Не вдалося видалити' });
```

Рисунок 3.20 – Функція remove

Наступний скрипт auth.js використовується для валідації чотирьох категорій HTTP-запитів, використовуючи API бібліотеки express-validator. Для кожної операції визначено окремий масив правил body(), які накладають обмеження на поля тіла запиту: registerValidation перевіряє коректність електронної адреси, мінімальну довжину пароля й імені, а також необов'язково перевіряє URL аватар (рис. 3.21).

```
import {body} from 'express-validator';

export const registerValidation = [
    body('email',"вкажіть вашу електронну пошту").isEmail(),
    body('password', "надто короткий пароль").isLength({min:5}),
    body('fullName', "вкажіть ім'я ").isLength({min:3}),
    body('avatarUrl','Не вірне посилання на фото').optional().isURL(),
];
```

Рисунок 3.21 – Перевірка даних при реєстрації

loginValidation фокусує на перевірку лише правильності формату e-mail та наявності пароля необхідної довжини.

У контексті створення статей postCreateValidation вимагає наявності назви й тексту з установленими мінімальними порогами символів, дозволяючи при цьому необов'язковий рядковий параметр imageUrl (рис. 3.22).

```
export const loginValidation = [
    body('email',"вкажіть вашу електронну пошту (невірний формат)").isEmail(),
    body('password', "Вкажіть пароль").isLength({min:5}),
];

export const postCreateValidation = [
    body('title',"Додайте заголовок").isLength({min:5}).isString(),
    body('text', "Додайте текст статті").isLength({min:10}).isString(),
    body('imageUrl','Не вірне посилання на зображення').optional().isString(),
];
```

Рисунок 3.22 – Перевірка даних при автентифікації та створенні статті

Найбільш об'ємним набір conferenceCreateValidation, що має перелік для перевірки: назви, опису, ISO-8601 дати, організатора, а також необов'язкових полів, Zoom-посилання та обкладинки. Кожне правило повертає локалізоване повідомлення про помилку (рис. 3.23).

```
export const conferenceCreateValidation = [
  body('title', 'Введіть заголовок конференції').isLength({ min: 3 }).isString(),
  body('description', 'Введіть опис конференції').isLength({ min: 10 }).isString(),
  body('date', 'Некоректна дата').isISO8601(),
  body('location', 'Введіть місце проведення').optional().isString(),
  body('organizer', 'Організатор обов’язковий').isString(),
  body('zoomLink', 'Некоректне посилання').optional().isURL(),
  body('coverImageUrl', 'Неправильне посилання на зображення').optional().isURL(),
];
```

Рисунок 3.23 – Перевірка даних при створенні конференції

3.4 Тестування backend інфраструктури

Для тестування backend інфраструктури використовується програма Insomnia. Insomnia — це кросплатформний клієнт для моделювання, тестування й налагодження HTTP- та GraphQL-запитів, який надає розробникам інтегроване середовище для перевірки бекенд-інфраструктури без необхідності звертатися до браузерних розширень чи CLI-утиліт.

Спочатку завантажимо backend-сервер, щоб отримати http-адресу, за допомогою якої буде відбуватися звернення до різних функцій додатку. Використовуємо для завантаження вбудований термінал Visual Studio Code (рис. 3.24):

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

> package@1.0.0 start:dev
> nodemon index.js

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Server OK
DB ok
```

Рисунок 3.24 – Завантаження backend-сервер

Отже сервер успішно завантажився, тепер є можливість переходити до наступного етапу, це перевірка запитів (POST, GET, PATCH, DELETE). Відкриємо програму Insomnia для моделювання звернення до серверу:

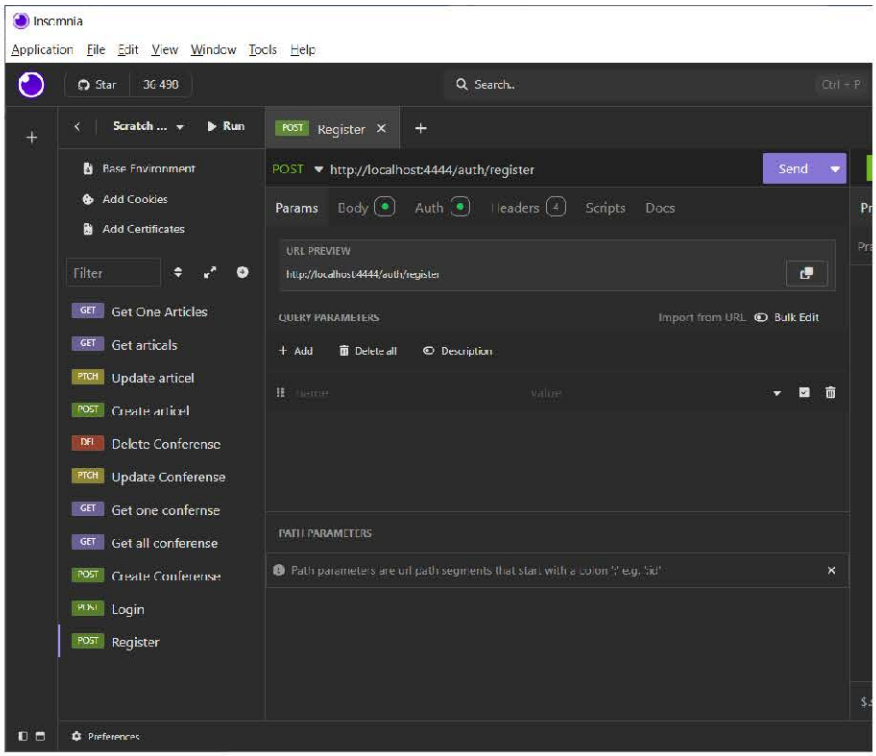

Рисунок 3.25 – Відкриття програми Insomnia

Спробуємо перевірити перший запит – це реєстрація нового облікового запису в базі даних. Обираємо запит виду POST, та вводимо наступний URL-рядок: http://localhost:4444/auth/register

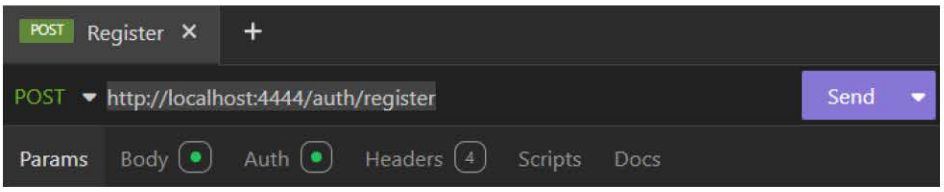

Рисунок 3.26 – Формування адреси запиту до серверу

Після цих дій потрібно визначити в якому форматі потрібно передавати дані. Потрібно обрати формат JSON (рис. 3.27):

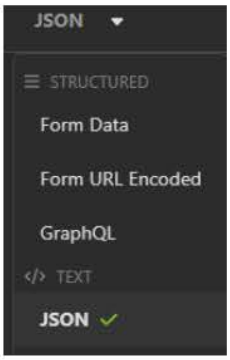

Рисунок 3.27 – Формат передачі даних

Перед користувачем з'явиться поле в якому можна додавати дані для передачі в базу даних, спробуємо додати нового користувача ввівши наступні дані (рис. 3.28):

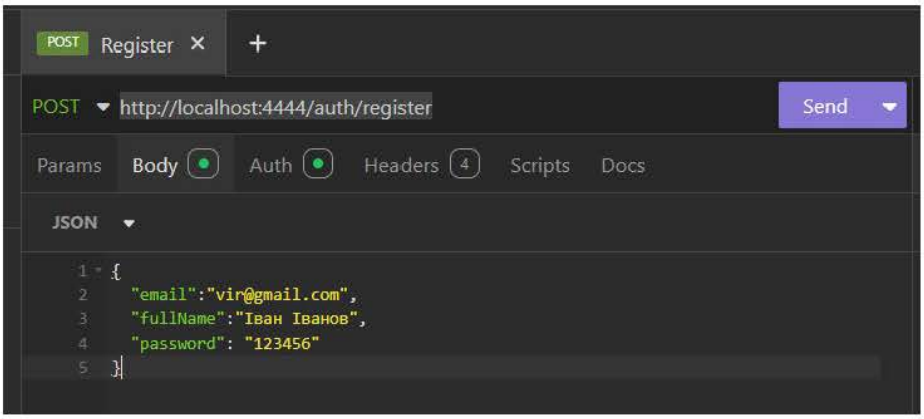

Рисунок 3.28 – Набір даних для реєстрації

Спробуємо відправити дані до бази, натиснувши на кнопку «Send». В результаті отримали успішну відповідь на запит авторизації, повернувши облікові дані та токен (рис. 3.29):

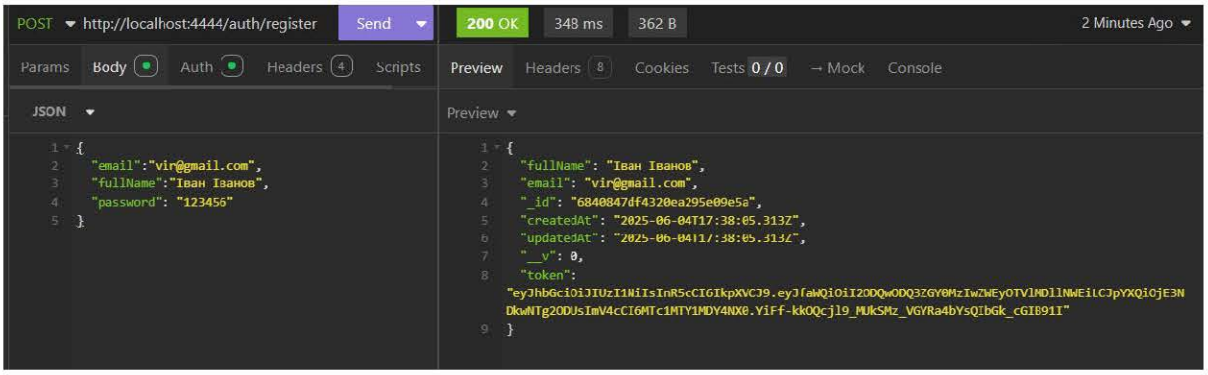

Рисунок 3.29 – Отримання даних при реєстрації

Варто звернути увагу, що при поверненні даних відсутнє поле як passwordHesh, воно не повертається а залишається тільки в базі даних. Щоб упевнитися в цьому відкриємо MongoDB, колекцію під назвою «users» (рис. 3.30):

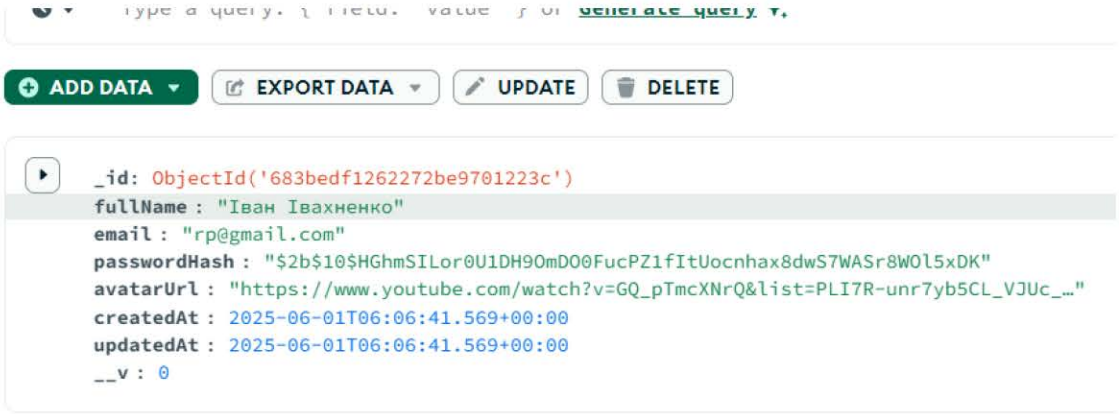

Рисунок 3.30 – Обліковий запис в базі даних

Щоб визначити які дані зберігається у токені, що щойно отримали потрібно використати сайт JWT: https://jwt.io/. Далі додати токен та отримати розшифровані дані (рис. 3.31) (eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2ODQwODQ3ZGY0MzIwZWEyOTVlMDllNWEiLCJpYXQiOjE3NDkwNTg2ODUsImV4cCI6MTc1MTY1MDY4NX0.YiFf-kkOQcjl9_MUkSMz_VGYRa4bYsQIbGk_cGIB91I):

Рисунок 3.31 – Отримані дані після розшифровки

Навіть після розшифровки токена, в результаті отримаємо лише ідентифікатор користувача для інших запитів, важлива інформація залишається в базі даних.

Спробуємо перевірити правила валідації. Для прикладу спробуємо повторно зареєструвати користувача з однаковою поштою (рис. 3.32):

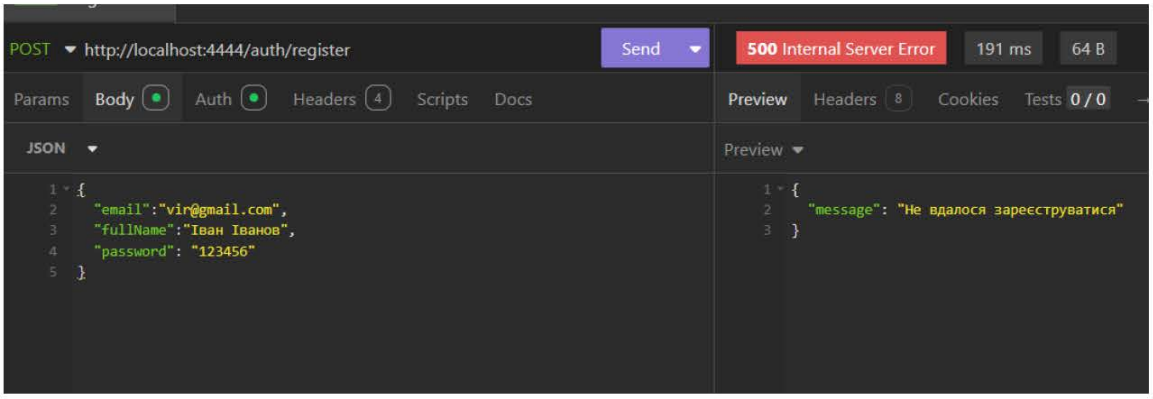

Рисунок 3.32 – Повторна реєстрація користувача

В результаті була отримана помилка 500, так як користувач вже з такими обліковими даними вже зареєстрований. Сервер також реагуватиме при невірному додаванні даних, що супроводжується помилкою «надто короткий пароль» або «вкажіть ім'я» (рис. 3.33):

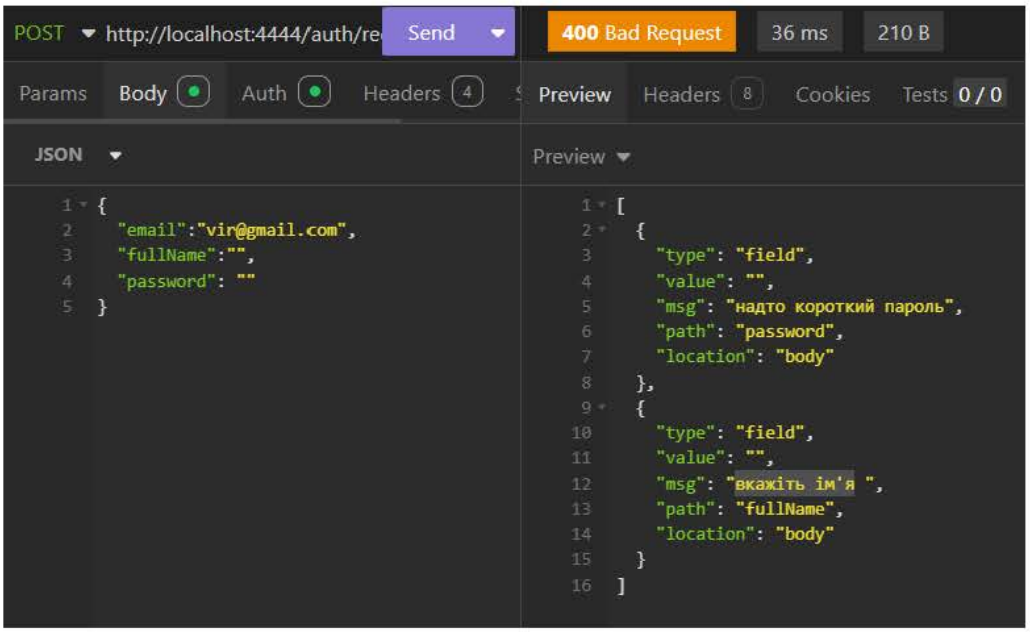

Рисунок 3.33 – Робота валідаційної системи

Є запити які не мають вбудованої перевірки авторизований користувач чи ні, в даному випадку це GET-запити, вони лише виводять дані. Спробуємо вивести всі створені конференції (рис. 3.34):

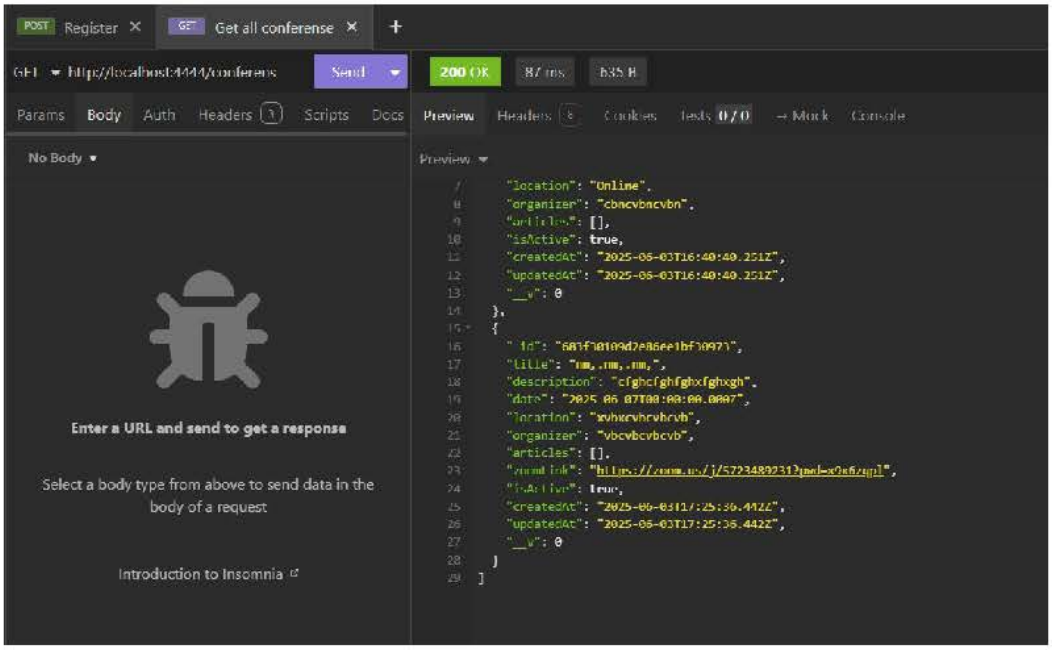

Рисунок 3.34 – Виведення всіх конференцій

В результаті отримуємо всі записи з колекції conferences, може бути використана в подальшому для розгляду конференції та вибору для участі в одній з них.

Розглянемо також запити, що потребують додаткових даних. Для прикладу спробуємо виконати POST-запит для створення нової конференції без токена (рис. 3.35):

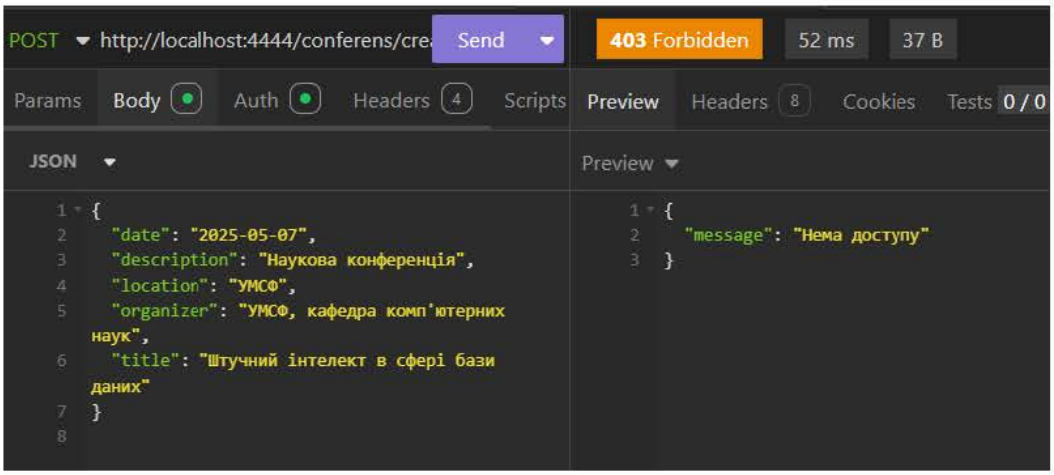

Рисунок 3.35 – Виведення помилки

В результаті спостерігаємо помилку «Нема доступу», це через те, що користувач не увійшов у систему та не отримав токен. Спробуємо ввести токен та знову виконати запит на створення конференції (рис. 3.36):

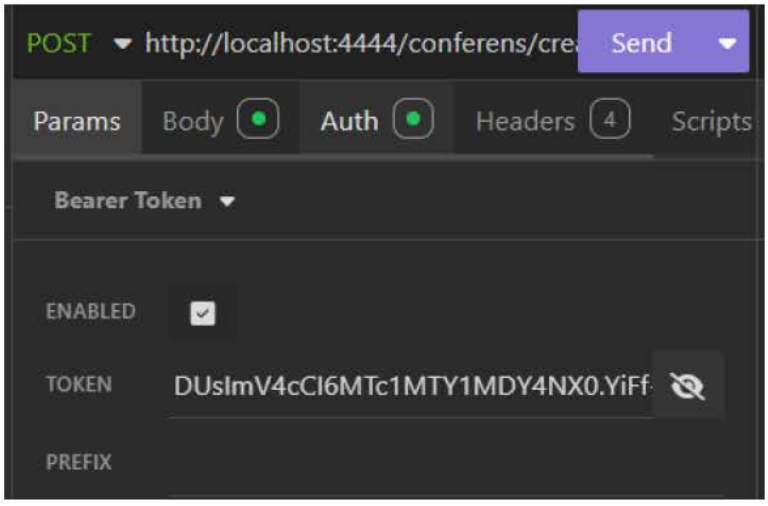

Рисунок 3.36 – Додавання токену

Знову спробуємо відправити запит на створення конференції (рис. 3.37):

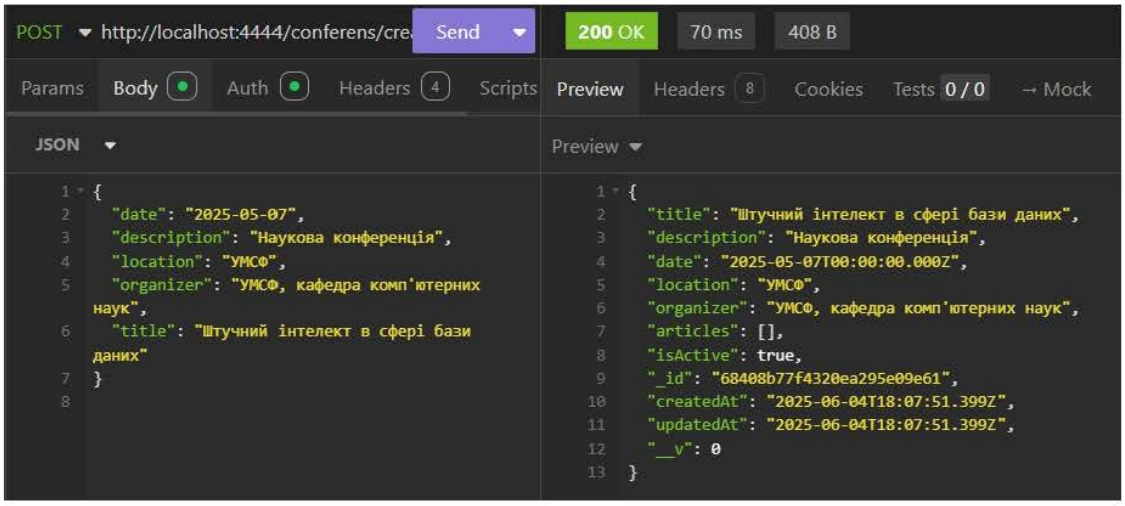

Рисунок 3.37 – Успішне додавання конференції

В результаті після додавання токену, запит відбувся успішно. Створено нова конференції з назвою «Штучний інтелект в сфері бази даних».

3.5 Висновок до третього розділу

У підсумку реалізовано повнофункціональний backend-сервер, що забезпечує реєстрацію й автентифікацію користувачів за допомогою JWT, підтримує CRUD-операції для конференцій та статей, інтегрується з MongoDB через Mongoose і гарантує захист даних завдяки хешуванню паролів і ретельній валідації вхідних запитів. Проведене тестування запитів у середовищі Insomnia підтвердило коректність логіки маршрутизації, стабільність взаємодії з базою даних і відсутність критичних вразливостей: усі сценарії авторизації, створення, оновлення та видалення ресурсів були успішно виконані, а передбачені механізми обробки помилок продемонстрували надійну стійкість сервісу до некоректних введень. Отримані результати засвідчують, що розроблений сервер відповідає вимогам безпеки й функціональності та може слугувати надійною основою для подальшого розширення веб-застосунку.

ВИСНОВОК

Кваліфікаційна робота присвячена розробці веб-платформи для проведення онлайн-конференцій, що є актуальним завданням у контексті цифрової трансформації наукової комунікації. Метою роботи було створення функціонального, масштабованого та безпечного інструменту, який дозволяє організовувати повноцінні академічні заходи в онлайн-середовищі.

У межах дослідження було проведено аналіз предметної області, виявлено ключові проблеми традиційного формату конференцій та обмеження сучасних універсальних платформ.

Серед проблем, які актуалізують необхідність розробки інноваційних цифрових рішень у пробемній області кваліфікаційної роботи є: географічні та логістичні обмеження традиційних конференцій, висока економічна вартість організації та участі, недостатня оперативність поширення наукових результатів, обмежена адаптованість та інтеграція існуючих комерційних платформ.

Наслідком зазначених проблем є об'єктивна необхідність у розробці спеціалізованих платформ, які здатні забезпечити повний життєвий цикл академічного заходу в режимі онлайн.

До основних функціональних вимог застосунку належать наступні: керування користувачами, адміністрування конференцій, обробка наукових статей, завантаження файлів, валідація вхідних даних, інтеграція з базою даних, забезпечення безпеки.

Таким чином, сформульовані функціональні вимоги визначають необхідний обсяг серверної логіки, яка повинна забезпечити коректну, безпечну та ефективну роботу веб-платформи для онлайн-конференцій. Вони слугують орієнтиром для розробника та основою для подальшого тестування, розширення і супроводу системи.

Здійснено порівняльний огляд архітектурних підходів і технологій, в результаті чого обґрунтовано вибір мікросервісної архітектури, технологій Node.js, Express.js та бази даних MongoDB. Особливу увагу приділено реалізації RESTful API, валідації вхідних даних, системі автентифікації користувачів та збереженню файлів.

У процесі реалізації було створено повнофункціональний back-end, який включає механізми керування користувачами, конференціями, науковими статтями та завантаженням матеріалів. Реалізовано систему авторизації на основі JWT, забезпечено захист доступу до маршрутів, валідацію даних та взаємодію з документно-орієнтованою базою даних. Для тестування API-запитів використано середовище Insomnia, що дозволило підтвердити стабільність і коректність роботи серверної частини.

Результати тестування демонструють відповідність системи поставленим вимогам, зручність використання та готовність до інтеграції з клієнтською частиною. Створений програмний продукт дозволяє автоматизувати ключові етапи організації наукових подій, зменшити адміністративні витрати, розширити доступ до академічних заходів та сприяє розвитку відкритої науки.

Отже, розроблена платформа відповідає сучасним технологічним вимогам і має потенціал до подальшого масштабування та впровадження в освітніх та наукових установах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бондаренко А.А. Цифрова трансформація академічної комунікації: виклики та можливості для онлайн-конференцій // Вісник Київського національного університету імені Тараса Шевченка. Серія «Інформаційні технології». 2021. № 1 (97). С. 15–23.

2. Коваленко О.В., Петренко І.С. Асинхронна архітектура у високопродуктивних веб-додатках на Node.js // Матеріали III Міжнародної науково-практичної конференції «Сучасні тенденції розвитку інформаційних технологій». Дніпро, 2020. С. 45–49.

3. Черниш М.Д., Гнатенко В.П. Мікросервісні архітектури: принципи та патерни розробки. Львів: Видавництво Львівської політехніки, 2022. 320 с.

4. Gartner. Top Strategic Technology Trends 2021: Digital Twins. 2021. URL: https://www.gartner.com/smarterwithgartner/top-strategic-technology-trends-2021-digital-twins

5. Клим М.І. REST-архітектура як основа для побудови сучасних веб-сервісів // Науковий вісник Національного університету «Львівська політехніка». Серія «Комп'ютерні науки». 2017. № 876. С. 34–38.

6. Зайцев С.В., Мачула О.А. Основи розробки веб-застосунків. Київ: Ліра-К, 2019. 280 с.

7. Node.js. About Node.js. URL: https://nodejs.org/en/about

8. MongoDB. MongoDB Manual. URL: https://www.mongodb.com/docs/manual/

9. Грицюк Ю.І. Документоорієнтовані бази даних як основа для сучасних інформаційних систем // Вісник Національного університету «Львівська політехніка». Серія «Інформаційні системи та мережі». 2018. № 905. С. 13–18.

10. Express.js Documentation. URL: https://expressjs.com/en/starter/installing.html

11. Mongoose. Mongoose Documentation. URL: https://mongoosejs.com/docs/

12. Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others. URL: https://www.altexsoft.com/blog/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/

13. Basics of MVC, Microservices, and Spring Boot. URL: https://www.educative.io/courses/developing-microservices-with-spring-boot/basics-of-mvc-microservices-and-spring-boot

14. MVC vs. Microservices: Understanding their Architecture. URL: https://wisdomplexus.com/blogs/mvc-vs-microservices/

15. Insomnia. Welcome to Insomnia Docs. URL: https://docs.insomnia.rest