Міністерство освіти і науки України Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «СТВОРЕННЯ ПРОТОТИПУ РЕЄСТРУ ЗДОБУТКІВ ВИКЛАДАЧІВ УНІВЕРСИТЕТУ»

Виконав: студент 4-го року навчання,

Спеціальності 122

Комп’ютерні науки

Біловербенко Ілля Ігорович

Керівник Олецький О. В.

доцент

Рецензент _____

(прізвище та ініціали)

Кваліфікаційна робота захищена з оцінкою

_____

Секретар ЕК

«_____»_____ 20 ___ р.

Київ – 2024

ЗМІСТ

ВСТУП

Обґрунтування вибору теми, актуальності її і наукового і практичного значення

У сучасному освітньому середовищі, де існує конкуренція серед викладачів та науковців, наявність реєстру для зберігання та відстеження їхніх досягнень стає корисним і важливим інструментом. Розглянемо, наприклад, ситуацію, коли університет шукає викладача для нового проекту. Маючи доступ до інформації про досягнення викладачів, адміністрація може здійснити обґрунтований та оптимальний вибір, враховуючи попередні успіхи та напрацювання у різних напрямках діяльності. Наприклад, викладач, який має значні досягнення у вивченні або розвитку певної теми, може бути ідеальним кандидатом для співпраці над конкретним проектом. Подібна система значно полегшить пошук кандидата.

Подальше використання застосунку також може бути важливим елементом у розвитку академічного співтовариства. Студенти, які шукають наукових керівників для своїх досліджень, або інші науковці можуть використовувати цю систему для пошуку викладачів, які мають відповідні здобутки та досвід у певній галузі. Такий підхід сприяє розвитку мережі академічних зв’язків і сприяє співпраці між вченими як з різних установ, так і в межах одного навчального закладу.

Крім того, зберігання і відстеження здобутків викладачів є важливим елементом внутрішнього управління університетом. Адміністрація може використовувати цю інформацію для оцінки роботи викладачів, розподілу

ресурсів та планування подальшого розвитку освітнього процесу. Наприклад, аналіз здобутків може допомогти ідентифікувати потреби викладачів у підвищенні кваліфікації або впровадженні нових педагогічних методик.

Уже висловлені аргументи щодо необхідності системи зберігання та відстеження досягнень викладачів свідчать про актуальність теми. Більш того, що в сучасному освітньому середовищі існує велика кількість можливостей для викладачів та науковців, таких як публікації у наукових журналах, виступи на конференціях, проведення самостійних досліджень та інші активності. Проте, ця різноманітність можливостей часто ускладнює процес відстеження та оцінки досягнень кожного окремого викладача.

Крім того, з огляду на швидкі темпи розвитку наукових та педагогічних технологій, важливо мати ефективні засоби для оцінки та підтримки розвитку викладачів. Система, яка дозволяє адміністрації університету аналізувати досягнення викладачів, стає ключовим інструментом у забезпеченні якості освіти та відповідності університету вимогам сучасності.

Щодо наукового та практичного значення, то розробка нових методів зберігання і відстеження досягнень викладачів може стати кроком у напрямку подальшого розвитку освіти та педагогіки. Ці методи можуть допомогти у створенні ефективних інструментів для аналізу, оцінки та підтримки роботи викладачів, сприяючи покращенню якості навчання студентів. Подальше дослідження у цьому напрямку може вивести на нові

методи і підходи до співпраці з викладачами, що стануть основою для удосконалення університетської середи.

Додатково, висвітлення та аналіз функціональних потреб університетської спільноти стосовно зберігання та відстеження досягнень викладачів може виявити потребу у розробці більш адаптивних та інтегрованих інформаційних систем. Це сприятиме покращенню управління навчальними процесами, підвищенню професійної компетентності викладачів і підтримці наукових досліджень. Такий підхід може забезпечити університети необхідними інструментами для адаптації до змін у сучасному освітньому середовищі.

Більш того, розробка та застосування нових методів зберігання та відстеження досягнень викладачів може мати позитивний вплив на створення сприятливої наукової атмосфери в університетах. Це може сприяти розвитку наукових досліджень, стимулювати потенціал викладачів та сприяти дослідження різних галузей. Такі кроки можуть стати важливими у розвитку освітньої сфери.

Аргументація вибору об'єкту та визначення предмету дослідження

Об'єктом дослідження обрано реєстр досягнень викладачів, оскільки він є ключовим інструментом для зберігання та відстеження академічних досягнень. Предметом дослідження визначено університетську систему, оскільки саме в університетському середовищі такий реєстр найбільш актуальний та потрібний для ефективного управління академічним процесом.

Визначення мети та конкретного завдання дослідження

Метою даного дослідження є створення прототипу реєстру, спрямованого на зберігання та відстеження досягнень викладачів університету. Основні завдання включають:

- Розробку функціональності реєстру: Створення механізму, що дозволить викладачам додавати, редагувати та видаляти свої досягнення у системі. Це включає створення форм для введення різноманітних видів досягнень, таких як наукові публікації, участь у проектах, виступи на конференціях та інші активності.
- Інтеграцію з базою даних Scopus: Реалізація інтеграції з базою даних Scopus для автоматичного отримання даних про наукові публікації викладачів. Це дозволить автоматизувати процес заповнення деяких видів досягнень у реєстрі.
- Розробку структури університету в системі: Створення ієрархічної структури, що відображає університетські підрозділи (факультети, кафедри) та викладачів, які на них працюють. Це передбачає можливість редагування даних про викладачів на різних рівнях адміністрації.
- Розробку зручного інтерфейсу користувача для неструктурованих даних: Реалізація інтерфейсу, що дозволить користувачам зручно вводити та редагувати неструктуровані дані про свої досягнення. Це включає створення зручних інтерфейсів для введення текстової інформації, виведення списків та інші елементи для зручного взаємодії з даними.

Коротка характеристика джерел, які були використані у роботі

У цьому підрозділі наведений огляд ключових джерел, які використовувалися під час розробки та реалізації нашого проекту.

Документації MongoDB та пов’язаних технологій: MongoDB надає докладну та зрозумілу документацію, що охоплює всі аспекти використання бази даних, включаючи інсталяцію, конфігурацію, CRUD операції та розробку схем даних. Документація MongoDB є важливим джерелом інформації для розробників, яка допомагає зрозуміти основні концепції та методики роботи з цією базою даних, що є ключовим для успішного використання MongoDB. Крім того, значний внесок у роботу внесли документація MongoDB Atlas, яка надала важливу інформацію щодо хмарних послуг та взаємодії з MongoDB в хмарному середовищі, та документація Mongoose, яка спростила використання об'єктно-документного моделювання в нашому проекті.

Elsevier Scopus APIs: Ці прикладні програмні інтерфейси надають програмний доступ до абстрактів та даних про цитування з усіх наукових журналів, які індексуються Scopus. Вони також дозволяють отримувати різноманітні дані, такі як відображення метрик публікацій, проведення пошуків та аналіз динаміки публікацій та заповнення систем поточної інформації про дослідження.

Документація Nuxt.js 3: Документація Nuxt.js 3 містить широкий спектр інформації щодо створення веб-додатків з використанням цього фреймворку. Вона охоплює різні аспекти розробки, які включають конфігурацію, роутинг, компоненти, проміжне програмне забезпечення та інші важливі аспекти. Ця документація є надійним джерелом інформації для розробників, яке допомагає зрозуміти принципи роботи фреймворку та

надає потрібні інструкції для розробки веб-застосунків з його використанням.

Документація бібліотеки nuxtjs/auth включає опис основних концепцій, приклади використання, інструкції з налаштування та інші корисні матеріали. Вона допомагає зрозуміти принципи роботи аутентифікації в Nuxt.js та ефективно використовувати бібліотеку для створення безпечних та надійних систем аутентифікації.

Документація бібліотеки jsonwebtoken також була важливим ресурсом у процесі розробники, бо була використана аутентифікація з використанням JSON Web Tokens. Джерело надає детальний огляд функцій та методів, які пропонує ця бібліотека для генерації і перевірки JWT-токенів, а також інструкції з їх правильного використання. Документація включає приклади коду, пояснення кращих практик та інші корисні матеріали, що допомогли ефективно впровадити JWT-аутентифікацію в додаток.

Методологічна основа

Методологічна основа дослідження ґрунтується на принципах системного аналізу та аналізу потреб користувачів. Було розглянуто дослідження як комплексний процес, що включає в себе вивчення потреб користувачів, аналіз функціональних вимог та вибір оптимальних технологічних рішень для досягнення мети проекту. Такий підхід дозволяє нам розробляти продукт, що відповідає реальним потребам користувачів та забезпечує ефективність та зручність використання.

РОЗДІЛ 1

1.1. Постановка задачі

У межах кваліфікаційної роботи були визначені наступні завдання:

1. Аналіз аналогічних платформ: Проведення аналізу різних платформ, спрямованих на академічних працівників та університетську спільноту, аби зрозуміти їхні можливості, їхній функціонал та важливі аспекти. Важливим на цьому етапі є зібрати інформацію про існуючі рішення і виявлення їхніх переваг і недоліків.
2. Визначення основних вимог: Формулювання основних функціональних і технічних вимог до платформи на основі отриманих від аналізу аналогічних програм висновків. Це включає в себе визначення можливостей, які платформа повинна надавати.
3. Вибір оптимальних технологій для розробки додатку та дослідження їхнього можливого використання, включаючи кращі практики та підходи до розробки: Дослідження різних технологій, які можна використовувати для розробки реєстру; вибір доцільних технологій для втілення поставлених функціональних вимог.
4. Формулювання завдань з розробки: Визначення конкретних завдань, які потрібно виконати під час розробки платформи, з урахуванням виявлених функціональних вимог і вибраного стеку технологій. Цей етап передбачає чітке формулювання завдань, які мають бути виконані під час розробки і визначення основних напрямків роботи.
5. Розробка реєстру: Цей етап є ключовим у цій роботі, тому що саме ця система буде центральним інструментом. Реєстр повинен

забезпечувати зручний інтерфейс виконання необхідних операцій над даними згідно функціональних вимог. Він також повинен мати можливості аутентифікації та авторизації користувачів, забезпечувати безпеку та конфіденційність даних. Крім того, реєстр має бути масштабованим, щоб відповідати зростаючим потребам та обсягам даних з часом.

1.2. Аналіз аналогічних програм

Для розуміння потреб, які програма має задовольняти, та очікувань від неї, потрібно провести аналіз аналогічних програм, спрямованих на викладачів та академічне середовище. Розглянемо такі платформи як Academia, ResearchGate та Scopus, щоб визначити їхні переваги, можливості та недоліки, і звернемо особливу увагу на їхню функціональність та відстеження наукового впливу користувачів. Це допоможе визначити ключові аспекти, які необхідно врахувати при розробці нової програми для академічної спільноти.

Отже, почнемо з платформи, що має назву Academia. Вона створена для академічних досліджень та взаємодії між викладачами, студентами та дослідниками. Основною перевагою цієї платформи є можливість безкоштовного публікування та поширення наукових праць, таких як статті, монографії, презентації й інші матеріали. Також важливою функцією є можливість відстеження наукового впливу. Користувачі можуть переглядати статистику цитування своїх робіт, а також інформацію про загальний вплив їхніх публікацій. Більше того, Academia надає інструменти для пошуку колег за інтересами та співпраці над спільними проектами.

Платформа ResearchGate націлена на спільноту науковців і дослідників, сприяння обміну науковими знаннями, публікаціями та дослідженнями. Ця платформа дозволяє науковцям ділитися своїми науковими публікаціями, спілкуватися з колегами, отримувати огляди своєї роботи та взаємодіяти з іншими дослідниками у своїй галузі. На ній користувачі можуть створювати свої профілі, додавати публікації, дослідження та інші наукові матеріали, а також відслідковувати цитування своєї роботи та спілкуватися з іншими учасниками академічної спільноти.

Scopus - це велика база даних наукових публікацій, яка охоплює широкий спектр наукових сфер. Ця платформа надає доступ до мільйонів академічних статей та понад трьох сотень тисяч книг. За допомогою неї науковці можуть знаходити інформацію для своїх досліджень. Крім того, Scopus надає інструменти для аналізу наукових даних, таких як вимірювання впливу публікацій та визначення ключових авторів та інституцій у певній галузі. Одною з основних переваг Scopus є його широке охоплення наукових видань. Платформа включає публікації від провідних наукових журналів та конференцій по всьому світу, що робить її дуже цінним інструментом для наукового дослідження та оцінки наукової продуктивності.

У розглянутих платформах можна виділити два найбільших недоліки, які цілком можна розглядати скоріше як особливості їхньої роботи, а не прості недопрацювання. Однак ці особливості можуть вплинути на те, наскільки ефективно користуватися платформою у науковому середовищі.

По-перше, слів виділити зосередженість цих платформ на наукових досягненнях та дослідженнях. Ця специфіка може викликати обмеження для викладачів, які хочуть показати широкий спектр своєї академічної активності, яка часом також включає в себе участь у наукових заходах, експертних групах і наукове керівництво над студентами.

По-друге, слід врахувати, що підтримка актуальності профілю на цих платформах лежить в основному на викладачах, але було б зручно, якби ці функції можна було надати методистам кафедр чи факультетів, що дозволило б викладачам більше часу приділяти своїй основній діяльності.

Результатом проведеного аналізу аналогічних платформ стали дві ключові функціональні можливості, які можна врахувати при розробці реєстру:

1. Розширення типажу здобутків: Програма має включати додаткові категорії здобутків для викладачів, які відображають їхню участь не тільки у написанні наукових публікацій, а й інші активності.

2. Керівництво акаунтом: Важливою функціональністю є можливість делегувати функції управління профілем іншим співробітникам університету, таким як методисти кафедр чи адміністрація факультету або університету.

1.3. Визначення основних функціональних вимог

Розглянемо функціонал, який має включати програма:

1. Аутентифікація та авторизація: Реалізація реєстрації користувачів у реєстрі та механізму входу в систему з можливістю визначення їхніх ролей та прав доступу до функцій системи.
2. CRUD-операції над профілями користувачів: Користувачі повинні мати можливість створювати, переглядати, оновлювати і видаляти свої профілі у системі, змінювати пароль до акаунту.
3. CRUD-операції над досягненнями: Користувачі повинні мати змогу додавати, переглядати, редагувати і видаляти свої досягнення, а також переглядати досягнення інших користувачів.
4. Співавторство у досягненнях: Реалізація можливості додавання та видалення інших користувачів як співавторів досягнень.
5. Пошук викладачів та їхніх досягнень: Система повинна надати можливість здійснювати пошук викладачів за різними критеріями, такими як прізвище, ім'я й унікальне ім’я користувача.
6. Імпортування досягнень із платформи Scopus: Реалізація інтеграції з платформою Scopus для імпортування наукових публікацій та інших досягнень викладачів.
7. Реєстрація та управління університетами: Система повинна дозволяти університетам реєструватися у системі, а також забезпечити можливість управління інформацією про них, такою як контактні дані та опис.
8. Університетська ієрархічна структура та управління нею: Система повинна підтримувати структуру університету, яка включає факультети і кафедри, а також забезпечити можливість управління цією структурою. Адміністрація кожної кафедри, представлена методистами, повинна мати можливість виконувати CRUD-операції

над досягненнями викладачів, які працюють на цій кафедрі. Адміністрація факультету повинна мати аналогічні можливості для викладачів, що відносяться до факультету. Також, адміністрація університету повинна мати права доступу до факультетів і кафедр всередині університету, щоби за необхідності виконувати відповідні операції над даними.

1.4. Визначення основних технічних вимог

1. Авторизація і аутентифікація: Забезпечення безпеки даних та захист від несанкціонованого доступу до системи є критично важливим аспектом розробки. Впровадження авторизації і аутентифікації дозволяє перевіряти ідентичність користувачів і контролювати доступ до різних функцій. Захист API шляхів, зокрема від спроб виконання несанкціонованих операцій, є дуже важливим для забезпечення безпеки додатку і даних користувачів.
2. Role Based Access Control на рівні БД: Використання RBAC на рівні бази даних дозволяє точно налаштовувати доступ до даних з урахуванням різних рівнів прав доступу. Забезпечити необхідний рівень контролю над даними та їхнього захисту в базі можна за допомогою двох адміністративних користувачів для бази даних - одного з доступом до конкретних колекцій, іншого - з повним доступом до кластера.
3. Роль адміністратора системи: Адміністратор відіграє ключову роль у забезпеченні безпеки та ефективності роботи платформи. Однією з основних функцій адміністратора є моніторинг та управління контентом платформи, зокрема видалення некоректного або

небажаного контенту. Крім того, адміністратор може здійснювати підтримку користувачів у випадку виникнення труднощів або проблем з функціонуванням системи. Такий підхід дозволяє забезпечити надійну та безперебійну роботу системи для всіх її учасників.

4. Надійне зберігання паролів: Паролі користувачів мають зберігатися в зашифрованому вигляді в базі даних. Це забезпечить безпеку особистої інформації користувачів, запобігаючи можливому доступу до їхніх облікових записів з боку третіх осіб, навіть у випадку витоку даних.

У ході цього розділу було визначено ключові завдання, які мають бути вирішені в ході розробки реєстру. Основною метою є створення веб-додатку, спрямованого на полегшення роботи викладачів університету з управління їхніми науковими досягненнями та співпраці в цій сфері. Було детально описано, які функції він має надавати і як він має працювати.

Основні завдання включають управління користувачами, які можуть бути викладачами або адміністраторами, контроль доступу до системи, можливість додавати та редагувати наукові досягнення, і створення ієрархічної структури університету. Також було згадано про необхідність захисту конфіденційності даних, зокрема, паролів користувачів.

Сформульовані вимоги є основою для подальшого розвитку програми і гарантують, що вона буде відповідати потребам користувачів і буде безпечною для використання.

РОЗДІЛ 2

2.1. Обґрунтування вибору технологій

Під час розробки програмного продукту особливу увагу потрібно звернути на вибір технологій, які будуть використані. Технології повинні бути стабільними, надійними та досить прогресивними, аби забезпечити продукту довготривалу та успішну роботу. Більш того, важливо, щоб вибрані технології забезпечували зручність у розробці, дозволяли ефективно втілювати необхідний функціонал і не перенасичували процес розробки надмірною складністю. Для забезпечення успішності реалізації проекту, кожна з обраних технологій повинна бути досить документованою, мати активну спільноту розробників та бути підтримуваною відповідними організаціями або компаніями. У цьому розділі ми розглянемо кожну з обраних технологій, проаналізуємо їх переваги та недоліки, і обґрунтуємо, чому саме вони вибрані для нашого проекту.

При виборі бази даних для проекту була врахована специфіка даних, які будуть зберігатися. Оскільки досягнення викладачів мають складну та неструктуровану суть, було обрано документну базу даних. Це дозволяє ефективно зберігати різноманітні формати даних без жорсткої структури. MongoDB відома своєю здатністю опрацьовувати неструктуровані дані, що робить її ідеальним вибором для проекту. Також вона відома своєю гнучкістю і можливістю масштабування, а це є важливими аспектами для

проекту, який передбачає зберігання потенційно великої кількості даних про досягнення викладачів. Крім того, активна спільнота користувачів та наявність широкої документації роблять MongoDB фаворитом серед подібних баз даних, адже вони дозволяють швидко знаходити відповіді на технічні питання та вирішувати проблеми під час розробки. Також враховуючи попередній досвід використання MongoDB в подібних проектах, можна впевнено стверджувати в її надійності та ефективності.

Задля ефективної та зручної роботи з MongoDB було обрано MongoDB Atlas, який являє собою набір сервісів, зосереджений навколо хмарної бази даних, розроблений для прискорення та спрощення роботи з даними. Він дозволяє швидко та легко створювати, масштабувати та керувати базами даних MongoDB у хмарних середовищах. Однією з головних переваг використання MongoDB Atlas є його зручність у керуванні – можна легко налаштувати та керувати базами даних без необхідності встановлення та налаштування серверів. Також Atlas є дуже надійним, про що свідчить його популярність. Можна бути впевнененим, що дані будуть збережені в безпечному і надійному місці, оскільки Atlas надає різні функції забезпечення безпеки, включаючи автоматичне резервне копіювання та шифрування даних. Окрім цього, MongoDB Atlas пропонує багато інших зручностей, таких як можливість миттєво масштабувати бази даних угору або вниз залежно від потреб додатку, використання різних типів серверів для оптимізації продуктивності та ефективності, а також доступ до усіх переваг хмарної інфраструктури без необхідності управління інфраструктурою на рівні сервера.

Отже, обравши ці технології для роботи з даними реєстру можна бути спокійним не тільки за стабільність цього аспекту програми, але й за майбутні кроки її розвитку.

Щодо клієнтської сторони, то одним із ключових виборів для розробки користувацького інтерфейсу є вибір фреймворку, який забезпечить ефективну та зручну роботу з інтерфейсом застосунку. Серед багатьох існуючих рішень, було обрано фреймворк Vue. Він є доволі прогресивним фреймворком, дозволяє створювати користувацькі інтерфейси без надмірних складнощів. Цей фреймворк отримав широку популярність завдяки своїй простоті, ефективності та зручності використання. Однією з його ключових переваг є невеликий розмір, який дозволяє швидше завантажувати сторінки та зменшує час відгуку. Vue також є гнучким і простим у навчанні. Також він надає широкий спектр можливостей для розробки інтерактивних інтерфейсів, включаючи компонентну архітектуру, реактивність та шаблони. Крім того, Vue має активну спільноту розробників, яка постійно розширюється та підтримується. Це означає, що завжди є багато ресурсів, документації та готових рішень, які допомагають швидше втілювати нові концепції і вирішувати проблеми.

Разом з Vue доцільним є використання іншого фреймворку, який побудований на базі Vue і розширює його функціональність – Nuxt 3. Однією з його ключових переваг є підтримка Server-Side Rendering (рендеринг на стороні сервера), що дозволяє генерувати сторінки на сервері перед їх відправленням клієнту, і це значно покращує продуктивність та SEO-показники веб-додатка. Крім того, Nuxt надає

зручний механізм для організації серверної логіки через server директорію, що спрощує розгортання веб-застосунків. Це далеко не усі переваги фреймворку, але це ключові аспекти, які мали великий вплив на вибір цієї технології.

Також розглянемо TypeScript як наступну ключову технологію. Ця ідеально підходить для веб-розробки через її здатність працювати як на боці клієнта, так і на сервері. Порівняно з JavaScript, найпопулярнішою мовою у цій галузі, TypeScript пропонує ряд переваг. Наприклад, строгу типізацію, за допомогою якої можна виявляти помилки на етапі розробки, що підвищує надійність написаного коду. Редактори коду активно підтримують використання цієї мови, що полегшує процес створення програмної частини проекту. Не в останню чергу інтеграція TypeScript з фреймворком Vue робить її привабливим вибором для розробки.

Наступною важливою технологією є Mongoose – бібліотека для моделювання об'єктів для MongoDB у середовищі Node.js. Вона є досить важливою у контексті даного проекту, оскільки дозволяє зручно і ефективно працювати з даними, які мають складну структуру або потребують валідації. У проекті, де зберігаються дані користувачів, університетів та їхні взаємозв'язки, є доречним мати засіб для опису цих даних у вигляді схем та моделей. Mongoose надає можливість створювати зручні схеми даних з валідацією. Це спрощує роботу з базою даних і допомагає уникнути помилок вводу. Більш того, бібліотека забезпечує можливість виконувати різноманітні операції з даними, такі як пошук, оновлення та видалення, що є ключовими для роботи з даними у будь-якому проекті та чудово інтегрується з фреймворком Vue та мовою

TypeScript. Такий підхід дозволяє ефективно організувати роботу з даними та забезпечити їхню цілісність.

2.2. Основи розробки з використанням MongoDB, MongoDB Atlas і Mongoose

1. Ознайомлення з MongoDB і MongoDB Atlas

MongoDB є документною базою даних, яка відрізняється від традиційних реляційних баз даних, таких як MySQL чи PostgreSQL. У MongoDB дані зберігаються у вигляді документів, які можуть бути структуровані у різний спосіб, що дає більшу гнучкість у роботі з ними.

Основними поняттями у MongoDB є база даних, яка є контейнером для колекцій; колекції, які містять документи; та документи, які є основною одиницею даних у MongoDB. Документи представляють собою JSON-подібні об'єкти зі змінним набором полів.

Для роботи з MongoDB у роботі використовується хмарна платформа MongoDB Atlas, яка забезпечує зручну і надійну роботу з базою даних. MongoDB Atlas дозволяє створювати бази даних у хмарі, керувати доступом до них, а також забезпечує автоматичне резервне копіювання даних для підвищення безпеки.

2. Використання Mongoose

Mongoose є об'єктно-документним моделювальником (ODM) для MongoDB, що надає простий і зручний спосіб взаємодії з цією базою

даних за допомогою JavaScript. Його використовують для визначення схем даних, виконання запитів до бази та валідації даних.

Основною перевагою використання Mongoose є те, що він дозволяє вам визначати схему вашої бази даних за допомогою простого JavaScript-синтаксису. Це полегшує розробку, оскільки ви можете точно визначити структуру даних і забезпечити їхню цілісність.

Крім того, Mongoose надає багато корисних функцій, таких як валідація даних перед збереженням, виконання складних запитів до бази даних та підтримка middleware. Middleware - це функції, які виконуються перед або після певних операцій з базою даних, що дозволяє реалізувати різноманітні логічні операції.

Використання Mongoose у проекті спрощує роботу з базою даних MongoDB, забезпечуючи чітку структуру даних та потужні інструменти для роботи з ними. Його простий та ефективний інтерфейс робить його відмінним вибором для будь-якого веб-додатка, який використовує MongoDB.

3. Розробка API з використанням Mongoose

1) Створення сервера та налаштування середовища розробки:

Першим кроком у розробці API з використанням Mongoose є створення сервера та налаштування середовища розробки. Для цього можна скористатися популярними бібліотеками і фреймворками, такими

як Express.js або Nuxt, які дозволяють швидко налаштувати та розгорнути сервер, або роблять це автоматично.

2) Створення моделей за допомогою Mongoose

важливим кроком є створення схем та моделей даних для взаємодії з базою даних MongoDB. Однією з кращих практик є організація коду, де для кожної колекції даних має бути окремий файл схеми в директорії "dbModels". Це дозволяє підтримувати чистоту коду та зручність у роботі зі схемами.

У директорії "dbModels" можуть знаходитися файли, які називаються відповідно до назв колекцій даних (наприклад, "User.js", "Achievement.js" тощо). Кожен з цих файлів містить опис схеми даних за допомогою Mongoose. Наприклад, для колекції користувачів може бути створено файл "User.js", де визначаються поля та їхні типи даних.

Після визначення схем даних необхідно створити моделі для цих схем. Для зручності краще створити окремий файл "index.js" у директорії "dbModels", де імпортуються всі схеми та створюються моделі за допомогою методу mongoose.model(). Це дозволяє уникнути дублювання коду та підтримувати структурованість проекту.

Цей підхід дозволяє з легкістю імпортувати всі моделі з одного файлу та забезпечує зручність у роботі з ними в усьому проекті.

3) Розробка маршрутів для API:

Після створення сервера потрібно розробити маршрути для обробки запитів до API. Це включає створення маршрутів для різних типів запитів (GET, POST, PUT, DELETE) для кожної сутності, з якою ми працюємо у нашому додатку. Наприклад, можуть бути маршрути для отримання списку користувачів, створення нового користувача, оновлення інформації про користувача тощо.

4) Написання контролерів:

Далі необхідно написати контролери для обробки різних типів запитів. Контролери відповідають за логіку обробки запитів та взаємодію з базою даних через Mongoose. Наприклад, у контролері може бути функція, яка отримує дані з бази даних та повертає їх у відповідь на запит з методом GET.

5) Валідація та перевірка запитів:

Однією з важливих складових розробки API є валідація та перевірка вхідних даних. Це допомагає забезпечити цілісність та безпеку даних. Ми можемо реалізувати валідацію на рівні контролерів, де перевіряємо вхідні дані Mongoose перед їх обробкою та збереженням у базі даних.

6) Тестування та налагодження:

Останнім етапом розробки API є тестування та налагодження. Бажано провести тестування API з використанням спеціальних інструментів, таких як Mocha, Chai або Jest, аби переконатися правильній роботі написаного коду. Після цього потрібно виправити помилки у

реалізації API, якщо такі виникли. Тестувати API рекомендується після кожної зміни у його коді.

2.3. Основи розробки додатків з використанням фреймворку Nuxt 3

1. Огляд фреймворку Nuxt 3

Nuxt 3 - це фреймворк для розробки веб-додатків на базі Vue.js. Він дозволяє розробляти як статичні, так і динамічні веб-додатки. Однією з ключових переваг цього фреймворку є його висока продуктивність та швидкість завантаження сторінок, що робить його чудовим вибором для проектів будь-якої складності.

Він має широкий набір модулів та розширень, що дозволяють легко інтегрувати різноманітні функціональність та інструменти, такі як сторонні бібліотеки або сервіси, чи навіть використовувати інші проекти з використанням Nuxt 3 як шаблон.

Nuxt 3 є потужним інструментом для створення сучасних веб-додатків, оскільки він підтримує різні стратегії веб-рендерингу, включаючи SPA (Single Page Applications) та SSR (Server Side Rendering). SPA дозволяє створювати додатки, які відображаються у браузері як єдине HTML-документ, що забезпечує швидкий та зручний інтерфейс для користувачів. З іншого боку, SSR дозволяє заванатажувати сторінки на сервері перед їхнім відправленням на клієнт, що підвищує продуктивність та SEO-оптимізацію додатків.

2. Основи налаштування фреймворку Nuxt 3 включають:

1. Встановлення: Спочатку потрібно встановити Nuxt 3 у проекті. Це можна зробити за допомогою менеджеру пакетів, наприклад, npm, yarn або pnpm, командою “npm install nuxt”, “yarn add nuxt” чи“pnpm add nuxt”.
2. Конфігурація: Фреймворк Nuxt 3 має файл конфігурації nuxt.config.js, де налаштовуються такі параметри додатку, як заголовок сторінки, метатеги, налаштування сервера, тощо.
3. Маршрутизація: Фреймворк дозволяє налаштовувати маршрути додатку у файлі nuxt.config.js або за допомогою директорії pages для автоматичного визначення маршрутів на основі файлів у ній.
4. Плагіни: Плагіни є чудовим інструментом для розширення функціональності додатку. Їх можна налаштувати у файлі nuxt.config.js або у директорії plugins.
5. Проміжне програмне забезпечення: Nuxt 3 дозволяє використовувати проміжне програмне забезпечення для обробки запитів до сервера перед їх обробкою. Його можна налаштувати як у файлі nuxt.config.js, так і у директорії middleware як на клієнстькій стороні, так і на серверній.
6. Серверна сторона: для належного налаштування фреймворку важливо розмістити серверний код у відповідній директорії з назвою ”server”, де можна розмістити API у директорії “api” та інші серверні скрипти. Це забезпечує належну організацію проекту і значно полегшує управління серверною частиною додатку завдяки автоматичному роутингу у цих директоріях.

РОЗДІЛ 3

3.1. Проектування архітектури додатку та бази даних

3.1.1. Структура бази даних

Нижче описані основні колекції та структури бази даних, які використовуються для зберігання та обробки даних у додатку. Основні сутності, представлені у базі даних, включають досягнення, кафедри, факультети, університети та користувачів. Для кожної сутності визначені відповідні схеми, що забезпечують їхню цілісність і взаємозв'язки одне між одним.

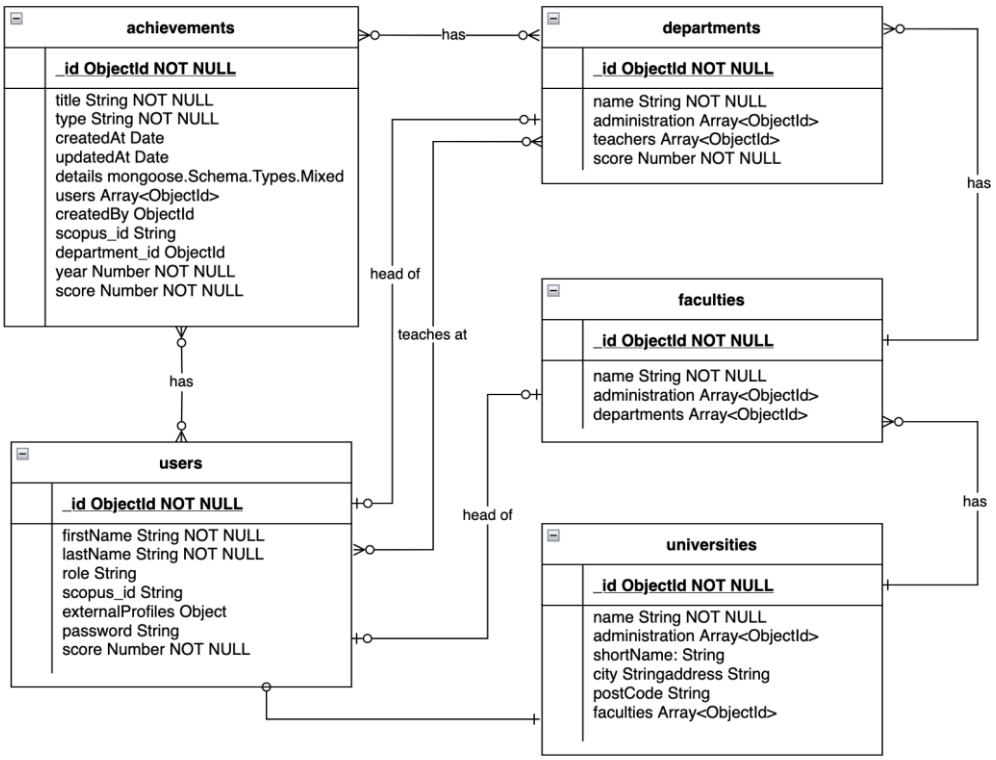

Рис. 3.1.1. ER-модель бази даних

Колекції бази даних:

Achievement (Досягнення):

title: Назва досягнення.

type: Тип досягнення, відповідає одному із значень переліку achievementTypeNames.

createdAt, updatedAt: Дати створення та оновлення досягнення.

details: JS-об'єкт, що містить детальну інформацію про досягнення. Цей об'єкт не має чітко визначеної структури і може містити різноманітні дані, що є ключовим аспектом для зберігання неструктурованої інформації. При визначенні Mongoose цього поля було використано тип Mixed, який призначений саме для таких випадків.

users: Посилання на користувачів, які є авторами досягнення. Це поле являє собою масив ідентифікаторів користувачів типу ObjectId.

createdBy: Поле зберігає ідентифікатор користувача, який створив досягнення.

scopus_id: Ідентифікатор досягнення на платформі Scopus.

department_id: Ідентифікатор кафедри, до якої відноситься досягнення.

year: Рік досягнення.

score: Бали, якими оцінюється досягнення.

Department (Кафедра):

name: Назва кафедри.

administration: Ідентифікатори членів адміністрації кафедри.

teachers: Викладачі, які працюють на кафедрі.

score: Сума балів досягнень, які відносяться до кафедри.

Faculty (Факультет):

name: Назва факультету.

administration: Ідентифікатори членів адміністрації факультету.

departments: Ідентифікатори кафедр, які входять до складу факультету.

University (Університет):

name: Назва університету.

shortName: Скорочена назва (часто аббревіатура) університету.

city, address, postCode: Місто, адреса та поштовий індекс університету.

administration: Ідентифікатори членів адміністрації університету.

faculties: Ідентифікатори факультетів, які входять до складу університету.

User (Користувач):

role: Роль користувача, яка може бути “user” або “admin”.

username: Ім'я користувача.

password: Пароль; зберігається у зашифрованому вигляді.

firstName, lastName, patronymic: Ім'я, прізвище та по батькові користувача.

scopus_id: Ідентифікатор користувача на платформі Scopus.

score: Сума балів усіх досягнень, у яких користувач зазначений серед авторів.

3.1.2. Використання перелічуваних типів даних

Для забезпечення консистентності даних у базі даних використовуються перелічувані типи даних для деяких полів. Це допомагає обмежити можливі значення полів і зменшує ймовірність помилок і забезпечує часткову структурованість даних.

Одним із таких полів є тип досягнення (type) у колекції Achievement. Для цього поля використовується перелік achievementTypeNames, який містить обмежений набір допустимих значень. Це дозволяє частково структурувати досягнення за їхнім типом. Це потрібно для подальшого аналізу та обчислення балів за досягнення.

const achievementTypes = ["Article", "Conference Paper", "Books or Book Chapters", "Technical Reports", "Conference Presentation", "Invited Talk

or Seminar", "Supervision of Graduate or Undergraduate Students", "Postdoctoral Supervision", "Research Grant", "Fellowship", "Award", "Workshops Attended or Conducted", "Certification", "Patents Filed or Granted", "Intellectual Property Contributions", "Editorial Board Memberships", "Reviewing for Journals or Conferences", "Outreach Activities", "Public Lectures or Media Appearances", "Academic Award", "Honorary Membership", "Developing of Software Tools", "Contributed to Open-source Projects", "Other"]

Використання цього переліку гарантує, що поле type у документі досягнення може містити лише одне із зазначених значень.

Також використовується перелічуваний тип даних для ролей користувачів. Це реалізовано за допомогою переліку role. Використання такого переліку допомагає забезпечити чітке розмежування прав і обов'язків різних типів користувачів, а також полегшує керування доступом до різних функцій системи.

const roles = [“admin”, “teacher”];

Використання цього переліку гарантує, що кожен користувач має одну з визначених ролей, що спрощує реалізацію механізмів аутентифікації та авторизації.

Переваги використання перелічуваних типів даних

- Консистентність даних: Використання перелічуваних типів даних дозволяє обмежити набір можливих значень для певних полів, що унеможливлює введення некоректних даних.

- Структурованість: Завдяки обмеженню можливих значень полів досягнень і ролей користувачів, дані стають більш структурованими та впорядкованими.

3.1.3. Використання Mongoose

Проект має на меті підтримувати зручне керування даними через Mongoose. Кожна сутність має свою схему, яка дозволяє визначати взаємозв'язки між колекціями даних та забезпечувати їхню консистентність. Моделі для кожної колекції організовані у окремих файлах для кращої структуризації та зручності підтримки.

```
import mongoose from 'mongoose'  839.1k (gzipped: 225.7k)
const { Schema } = mongoose

const SchemaMain = new Schema({
  name: {
    type: String,
    required: true
  },
  shortName: String,
  city: String,
  address: String,
  postCode: String,
  administration: [{
    type: Schema.Types.ObjectId,
    ref: 'user'
  }],
  faculties: [{
    type: Schema.Types.ObjectId,
    ref: 'faculty'
  }]
})

export default SchemaMain
```

Рис. 3.1.1. Схема університету

3.2. Розробка серверної частини

Для реалізації серверної частини проекту було використано Nuxt 3 та його директорію server для створення API. Це дозволило інтегрувати серверну логіку безпосередньо в Nuxt-додаток, що спрощує розробку та підтримку коду.

3.2.1 Структура директорії server/api

У директорії server/api розміщені наступні маршрути (routes):

Achievements

- Створення, читання, оновлення та видалення досягнень.
- Видалення автора з досягнення.
- Пошук за параметрами:
  - createdBy: пошук досягнень за ідентифікатором користувача, який їх створив.
  - user: пошук досягнень за ідентифікатором користувача (автора).

Departments

- Отримання інформації про кафедри.
- Отримання досягнень кафедри.
- Отримання досягнень і списку, де ключами є ідентифікатори викладачів, а значеннями - суми балів за досягнення, які відносяться до факультету і у яких викладач є автором.

Faculties

- Отримання інформації про факультети.

Universities

- Створення університету.
- Пошук університетів.

Також реалізовано вкладену структуру маршрутів для PUT та POST запитів:

universities/:universityId/faculties/:facultyId/departments/:departmentId

Це дозволяє здійснювати операції з факультетами та кафедрами у контексті конкретного університету. Для того, аби, наприклад, змінити дані про кафедру, необхідно також ввести у параметри universityId і facultyId. Це пов'язано з тим, що права доступу працюють таким чином, що адміністрація університету має доступ до операцій над факультетами університету і їхніми кафедрами. Аналогічно, адміністрація факультету має доступ операцій над усіма кафедрами факультету. Коли здійснюється запит для зміни даних про кафедру, система перевіряє права доступу користувача, використовуючи ID університету (universityId) і ID факультету (facultyId). Це забезпечує контроль доступу таким чином, що лише адміністрація відповідного університету або факультету може здійснювати операції над відповідними даними.

Scopus

- author/[id].get.ts: повертає досягнення користувача за його Scopus ID, а також розділяє досягнення на вже імпортовані та ті, що ще не були імпортовані.

Users

- Створення, читання, оновлення та видалення користувачів.
- Пошук користувачів.
- Зміна пароля користувача.

3.3. Авторизація і аутентифікація

3.3.1. Аутентифікація

Для аутентифікації було використано бібліотеку @nuxtjs/auth. У директорії server/api/auth реалізовано чотири маршрути (routes):

- signup: реєстрація нового користувача.
- signin: вхід користувача в систему.
- user: отримання поточного користувача за допомогою refresh token.
- refresh: оновлення токена.

Використана стратегія local, яка базується на введенні username (унікального імені користувача) та пароля. Токени використовуються у форматі JWT (JSON Web Tokens), для роботи з якими було використано бібліотеку jsonwebtoken. Це забезпечує безпеку передачі даних та зручність в управлінні сесіями. При вході в систему інформація про користувача, така як ім'я, прізвище, роль, ID, та username, шифрується у токені, що забезпечує безпеку передачі даних.

3.3.2. Авторизація

Авторизація у системі базується на ролях користувачів. Кожен користувач має роль, яка визначає його права та доступ до певних функцій системи. Ролі користувачів не можна змінювати через інтерфейс користувача та це заборонено Mongoose схемою – значенням поля “immutable” встановлено “true”. Змінити роль користувача може лише особа з доступом до бази даних, яка може вручну внести зміни. Така реалізація обрана задля гарантування безпеки та запобігання небажаній зміні ролей користувачів, щоб користувачі не отримували адміністративні права без відповідного дозволу.

3.3.3. Захист паролів

Для забезпечення безпеки користувачів у системі було реалізовано надійний механізм збереження та перевірки паролів. Це важливо для захисту особистих даних користувачів і запобігання несанкціонованому доступу. Для цієї мети була використана бібліотека mongoose-bcrypt, яка дозволяє хешувати паролі та здійснювати їх перевірку.

У схемі користувача поле password визначено з використанням бібліотеки mongoose-bcrypt, що забезпечує автоматичне хешування паролів перед їх збереженням у базі даних:

password: { type: String, bcrypt: true, required: true }

Це означає, що кожен пароль, який зберігається у базі даних, буде зашифровано за допомогою алгоритму bcrypt, що значно підвищує рівень безпеки збережених даних.

Для перевірки паролів було додано метод до схеми користувача, який використовує функцію compareSync з бібліотеки bcrypt:

```
SchemaMain.methods.verifyPasswordSync = function (password) {
  return bcrypt.compareSync(password, this.password);
}
```

Він дозволяє порівняти пароль, який введений користувачем, з хешованим паролем, що зберігається у базі даних. Якщо паролі співпадають, користувач отримує доступ до системи і може, наприклад, встановити новий пароль. Зокрема, зміна паролю передбачає таку перевірку поточного паролю користувача. Спочатку перевіряється правильність введеного поточного паролю за допомогою методу verifyPasswordSync:

```
if (this.verifyPasswordSync(currentPassword)) {
  this.password = newPassword;
  this.save();
}
```

Якщо поточний пароль правильний, новий пароль встановлюється і зберігається у зашифрованому вигляді.

3.4. Розробка проміжного програмного забезпечення

Проміжне програмне забезпечення (middleware) відіграє ключову роль у забезпеченні безпеки та управлінні доступом у веб-додатках. Для

даного проекту було реалізовано декілька важливих middleware, які забезпечують захист від неавторизованого доступу та контролюють права користувачів на виконання певних дій.

3.4.1. Middleware для аутентифікації

У серверній частині у папку middleware було додано файл auth.ts, який відповідає за аутентифікацію користувачів. Цей middleware перевіряє наявність та дійсність JWT токенів у запитах, використовуючи бібліотеку jsonwebtoken. Токени надходять у HTTP заголовках, і middleware auth.ts здійснює їхню валідацію перед наданням доступу до захищених маршрутів. Отримані за допомогою токену дані про викладача додаються до контексту запиту і можуть далі бути використані в API:

event.context.user = user

3.4.2. Middleware для авторизації

Для забезпечення надійної авторизації було створено проміжне ПЗ у файлі isAdmin.ts, яке перевіряє, чи має користувач роль адміністратора. Це, у свою чергу, забезпечує захист від несанкціонованих змін даних у системі.

У системі також реалізовано функцію, яка перевіряє, чи має користувач права доступу до університету, факультету або кафедри. Це забезпечує додатковий рівень захисту для ієрархічної системи управління.

3.4.3. Передача інформації про права доступу у токені

При аутентифікації, у токені шифрується також і ідентифікатори університетів, факультетів і кафедр, де користувач має роль адміністратора. У клієнтській частині, коли потрібно перевірити, чи має користувач адміністраторські права на певну сутність, перевіряється наявність ID цієї сутності серед тих, що зашифровані у токені.

Передавання цієї інформації у токені є безпечним, оскільки вона не може бути підроблена. Якщо навіть користувач спробує надіслати запит, на який не має прав, проміжне ПЗ на серверній частині захистить маршрути від небажаних дій.

3.5. Інтеграція з зовнішніми сервісами

Для покращення функціональності та зручності використання додатку було реалізовано інтеграцію з зовнішнім сервісом Scopus для автоматичного додавання наукових публікацій. Це дозволяє викладачам легко додавати свої публікації до бази даних додатку, що значно спрощує процес ведення реєстру їхніх досягнень.

Для інтеграції з зовнішніми сервісами було використано API Elsevier, яке надає доступ до бази даних Scopus. Це API дозволяє отримувати інформацію про наукові публікації, пов'язані з конкретним автором за його Scopus ID.

3.5.1. Налаштування інтеграції зі Scopus

Першим кроком для налаштування інтеграції з Scopus було згенерування API ключа. Це можна зробити за посиланням: https://dev.elsevier.com/apikey/create. Отриманий API ключ було додано до

файлу .env у змінну SCOPUS_API_KEY, звідки потім зручно отримувати цей ключ.

Для імпортування документів з Scopus користувачеві потрібно зазначити ідентифікатор свого профілю на цій платформі у налаштуваннях акаунту. За ним здійснюється імпортування публікацій.

3.5.2. Процес імпортування

Для імпортування наукових публікацій було реалізовано маршрут /server/scopus/author/[id] з методом “GET”. Він обробляє запити на отримання публікацій для конкретного автора за його Scopus ID.

1. Отримання Scopus API ключа: Спочатку перевіряється наявність API ключа для доступу до Scopus API.
2. Формування запиту до Scopus API: Створюється URL запиту для отримання публікацій автора за його Scopus ID. Використовуються параметри au-id і apiKey для зазначення ідентифікатора користувача та ключ API.
3. Конвертація даних: Інформація про публікації, отримана з відповіді від Elsevier API, конвертується у формат, зручний для збереження у базі даних додатку.
4. Перевірка наявності публікацій: Перевіряється, чи вже існують ці публікації у базі даних, щоб уникнути дублювання. Проте вже існуючі публікації у вигляді здобутків групуються окремо. Інформація про них може бути корисною, наприклад, для оновлення даних про здобутки.

3.6. Розробка клієнтської частини

Для розробки клієнтської частини було використано Nuxt 3, що дозволило створити зручний та інтуїтивно зрозумілий інтерфейс користувача з підтримкою сучасних веб-технологій. Nuxt 3 забезпечує ефективну інтеграцію з серверною частиною та спрощує реалізацію багатьох аспектів додатку.

3.6.1. Структура маршрутизації

У клієнтській частині була використана схожа структура маршрутизації, як і у серверній частині, що стосується університетської структури: /universities/:universityId/faculties/:facultyId/departments/:departmentId. Ця структура забезпечує логічну організацію навігації та дозволяє користувачам легко знаходити необхідну інформацію, пов'язану з університетами, факультетами та кафедрами.

3.6.2. Обмеження доступу

Для забезпечення безпеки та управління доступом до певних сторінок було реалізовано обмеження доступу для неавторизованих користувачів. Наприклад, сторінки /account і /achievement/scopus/import, які надають графічний інтерфейс для, відповідно, змінення даних про поточного авторизованого користувача і імпортування публікацій зі Scopus як здобутків, доступні лише для авторизованих користувачів. Якщо ж відвідувач сторінки не авторизований, то йому буде показана сторінка помилки.

Перевірка статусу користувача відбувається за допомогою функції useAuth з бібліотеки @nuxtjs/auth:

```
const { data: user, status } = useAuth()

if (status.value === 'unauthenticated') {

  throw createError({ statusCode: 401, message: 'You must be authenticated to access this page.' })

}
```

3.6.3. Реалізація сторінок

Було додано декілька ключових сторінок для забезпечення основної функціональності додатку:

- Сторінка авторизації: Дозволяє користувачам увійти до системи, ввівши свої облікові дані.
- Сторінка реєстрації: Дозволяє новим користувачам зареєструватися у реєстрі.
- Сторінка профілю викладача: Відображає профіль викладача з його досягненнями та університетами, в яких він працює.
- Сторінка здобутку: Відображає деталі конкретного досягнення.
- Сторінка пошуку університету: Дозволяє користувачам шукати університети за назвою та короткою її версією.
- Сторінка пошуку користувачів: Дозволяє шукати користувачів системи за прізвищем, іменем та унікальним іменем користувача.

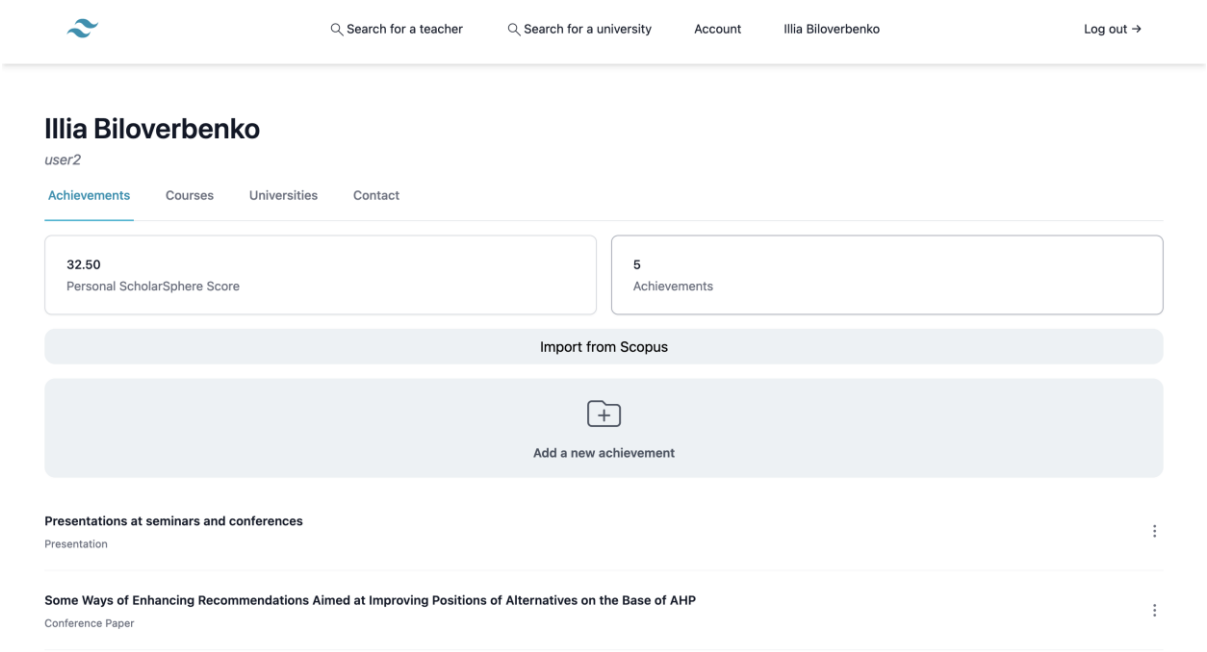

Рис. 3.6.1. Сторінка користувача.

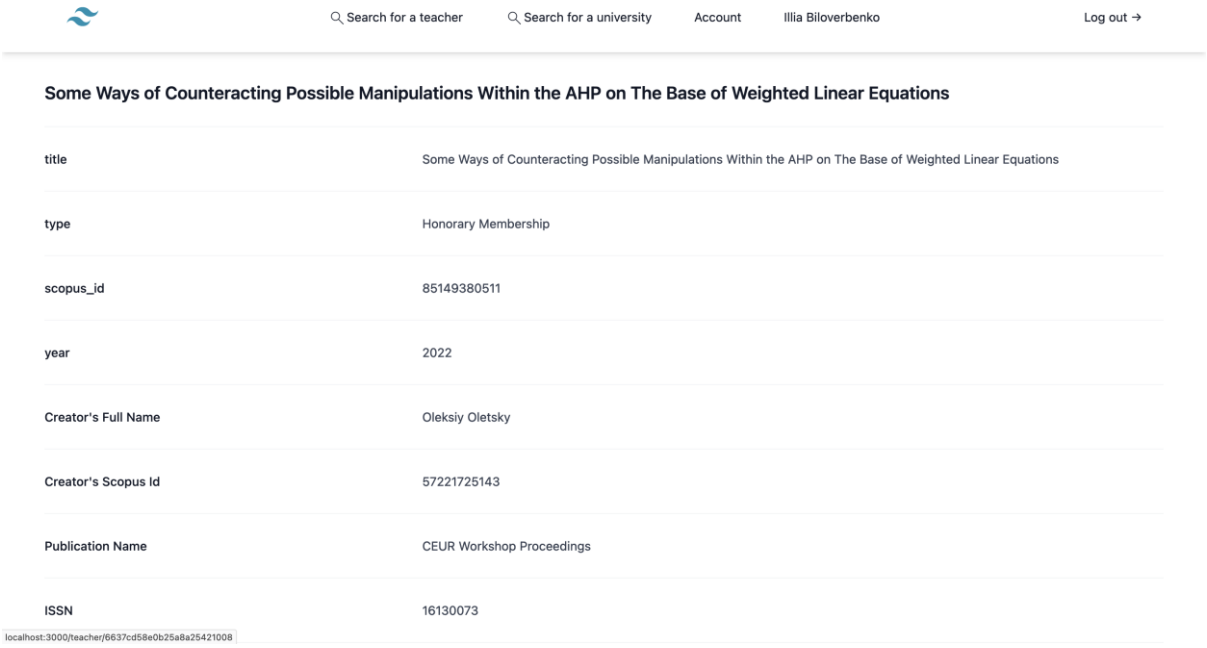

Рис. 3.6.2. Сторінка перегляду досягнення викладача.

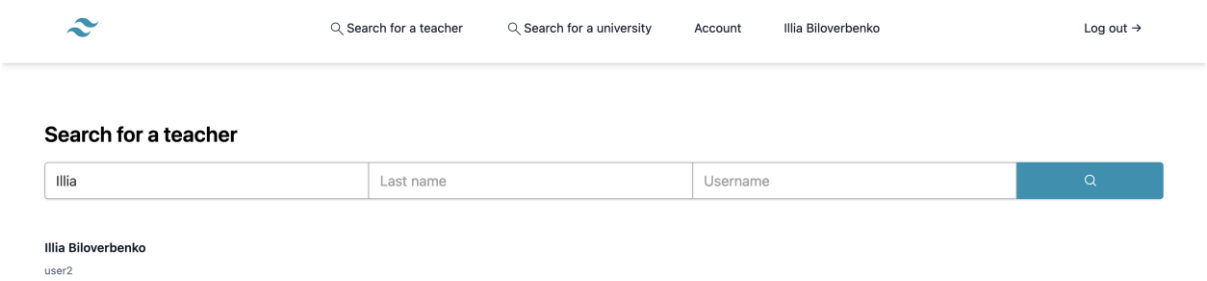

Рис. 3.6.3. Сторінка пошуку викладачів.

3.6.4. Управління досягненнями адміністрацією кафедри

Адміністрація кафедри має можливість додавати досягнення, що стосуються їхньої кафедри, і зазначати авторів цих досягнень. Це забезпечує доступ адміністрації до виконання CRUD операцій над досягненнями викладачів, але лише для тих досягнень, які відносяться до кафедри. Індивідуальні досягнення викладачів, які не прив'язані до кафедри, не можуть бути змінені адміністрацією кафедри, що гарантує збереження особистих даних та результатів роботи викладачів.

3.6.5. Управління доступом до компонентів інтерфейсу

Хоча майже всі сторінки доступні для всіх ролей користувачів, деякі компоненти інтерфейсу доступні лише тим, хто має відповідні права. Таким чином забезпечується більш гнучке та безпечне управління

додатком і запобігаються небажані дії. Наприклад, кнопка видалення досягнення доступна лише адміністраторам додатку та тим, хто додав його. Якщо ж користувач не додавав досягнення, але є серед його авторів, кнопка буде не видаляти документ зі здобутком із бази даних, а лише видаляти користувача з авторів досягнення.

3.6.6. Створення важливих Vue компонентів

Для підвищення функціональності та зручності користування додатком було створено кілька важливих Vue компонент:

1. AutoComplete: компонент AutoComplete був створений на основі однойменного компонента з бібліотеки Vercel, який, у свою чергу, базується на Combobox з Headless UI. Цей компонент дозволяє користувачам шукати та обирати потрібні елементи зі списку. Наприклад, він використовується у полі для авторів здобутку, що дозволяє легко додавати декількох авторів. Це значно спрощує процес введення даних та забезпечує зручність користування системою.
2. DynamicForm: компонент DynamicForm представляє собою форму з динамічними полями, де користувач може видаляти існуючі поля (якщо це дозволено) та додавати нові. Це особливо корисно для введення інформації про здобутки, які часто є дуже неструктурованими. Така гнучкість дозволяє користувачам налаштовувати форму відповідно до своїх потреб, забезпечуючи можливість зберігання різноманітної інформації у зручному вигляді.

3.7 Система балів для викладачів і кафедр

У системі реєстрації досягнень викладачів кожне досягнення має певну "вагу" або кількість балів, які воно додає до рахунку його авторів та, за наявності department_id (якщо досягнення стосується певної кафедри), до рахунку відповідної кафедри. Початково, автори та кафедри мають нульовий рахунок, який зростає з додаванням нових досягнень. Ці бали зберігаються у полях score, які користувачі не можуть змінювати.

3.7.1. Оновлення рахунків авторів і кафедр

Для автоматичного оновлення рахунків авторів і кафедр було створено MongoDB Atlas тригер, який реагує на створення, оновлення або видалення досягнень. Цей тригер забезпечує коректне оновлення балів, базуючись на типі досягнення та його специфічних характеристиках.

3.7.2. Алгоритм роботи MongoDB Atlas Trigger

- При створенні досягнення:
  1. Розраховується кількість балів на основі типу досягнення.
  2. Оновлюється поле score для досягнення.
  3. Бали додаються до рахунків авторів та відповідної кафедри (якщо вказано department_id).
- При оновленні досягнення:
  1. Перевіряється, чи змінився рахунок.
  2. Оновлюється поле score для досягнення.

3. Оновлюються рахунки авторів та відповідної кафедри на основі різниці між старим і новим рахунком.

- При видаленні досягнення:

1. Вираховується кількість балів, яку потрібно відняти.
2. Оновлюються рахунки авторів та відповідної кафедри.
3. Коефіцієнти для розрахунку балів

Для розрахунку балів застосовуються два коефіцієнти:

1. Коефіцієнт за роки:

Зменшує кількість балів за досягннення, які старші за 3 роки, крім певних категорій, таких як нагороди, професійний розвиток, патенти та інтелектуальна власність.

Формула: якщо досягнення старше 4 років, коефіцієнт зменшується починаючи з 0.95 на 0.01 за кожен додатковий рік, але не нижче 0.85.

```
let coefficient = 1.0;
if (!exemptCategories.includes(category) && ageInYears > 3) {
  coefficient = 0.95 - (ageInYears - 4) * 0.01;
  coefficient = Math.max(0.85, coefficient);
}
```

2. Коефіцієнт за цитування:

Застосовується до здобутків, які є публікаціями і збільшує кількість балів за них на основі кількості цитувань.

Формула: коефіцієнт збільшується на 0.05 за кожні 5 цитувань, але не перевищує 1.5.

```
function getCitationsCoefficient(achievement) {
  const category = achievementTypes[achievement.type].category
  let coefficient = 1
  if (category === 'Publications' && achievement?.details?.Citations) {
    coefficient += Math.floor(Number(achievement?.details?.Citations) / 5) / 20
  }
  return Math.min(1.5, coefficient)
}
```

Розрахунок загальної кількості балів:

Загальний коефіцієнт розраховується як сума всіх коефіцієнтів мінус кількість коефіцієнтів плюс 1.

Формула: score = базовий бал досягнення * загальний коефіцієнт.

```
function getScore(achievement) {
  const coefficients = [
```

```
    getCitationsCoefficient(achievement),
    getYearCoefficient(achievement)
  ]
  const coefficient = coefficients.reduce((accumulator, currentValue) =>
accumulator + currentValue, 0) - coefficients.length + 1
  return achievementTypes[achievement.type]?.score * coefficient
}
```

3.8. Висновки

Цей розділ описав ключові етапи розробки системи реєстру досягнень викладачів, охоплюючи проектування бази даних, серверну та клієнтську частини, а також інтеграцію з зовнішніми сервісами та систему балів.

Було спроектовано структуру бази даних, яка забезпечує ефективне зберігання та обробку даних про досягнення викладачів. Вибір MongoDB як основного сховища даних дозволив забезпечити гнучкість та масштабованість системи.

Для реалізації серверної частини було використано Nuxt 3, що дозволило інтегрувати серверну логіку безпосередньо у додаток. Це спростило управління проектом та забезпечило високу продуктивність та надійність системи.

Для аутентифікації користувачів було використано бібліотеку @nuxtjs/auth з підтримкою JWT токенів, що забезпечило безпечний доступ до системи. Авторизація базувалася на ролях користувачів, що гарантувало правильний розподіл прав доступу до різних функцій системи.

Паролі зберігалися у зашифрованому вигляді за допомогою бібліотеки mongoose-bcrypt, що забезпечувало їх захист від несанкціонованого доступу. Процедура зміни паролів включала перевірку поточного паролю, що додатково підвищувало рівень безпеки.

Було реалізовано декілька важливих middleware, які забезпечували аутентифікацію та авторизацію користувачів, а також перевірку прав доступу до різних ресурсів системи. Це дозволило створити надійний захист від небажаних дій.

Клієнтська частина була розроблена з використанням Nuxt 3, що забезпечило створення сучасного та зручного інтерфейсу користувача. Структурована маршрутизація та управління доступом до компонентів інтерфейсу гарантували високий рівень зручності та безпеки для користувачів.

Інтеграція з Scopus дозволила автоматично отримувати наукові публікації та додавати їх до бази даних додатку. Це значно спростило процес ведення реєстру досягнень викладачів та забезпечило актуальність даних.

Розроблена система балів дозволила ефективно оцінювати досягнення викладачів та кафедр. Використання коефіцієнтів для різних

категорій досягнень забезпечило певну обґрунтовану логіку оцінки здобутків, яка позитивно реагує на цитованість публікацій та негативно – на дещо застарілі досягнення. Використаний MongoDB Atlas тригер, який автоматично оновлює рахунки авторів та кафедр при додаванні, оновленні або видаленні досягнень.

Усі ці аспекти разом забезпечили створення доволі ефективної, надійної та зручної системи реєстру досягнень викладачів, яка відповідає необхідним функціональним та нефункціональним вимогам.

ВИСНОВКИ

У ході виконання дипломної роботи було успішно реалізовано всі завдання, визначені на початковому етапі. Проведений аналіз аналогічних платформ дозволив визначити основні функціональні та технічні вимоги до розробки реєстру досягнень викладачів. На основі отриманих даних були вибрані оптимальні технології для реалізації проекту, що включали MongoDB, Mongoose, Vue, Nuxt, TypeScript та інші сучасні веб-технології.

Формулювання завдань з розробки включало чітке визначення всіх необхідних етапів роботи, що дозволило ефективно організувати процес розробки та забезпечити виконання всіх функціональних вимог. Було розроблено систему, яка відповідає вимогам зручності використання, безпеки і конфіденційності даних.

Реєстр досягнень викладачів був реалізований з урахуванням усіх визначених вимог. Система забезпечує зручний та інтуїтивно зрозумілий інтерфейс для виконання необхідних операцій над даними. Функції аутентифікації та авторизації користувачів реалізовані з використанням бібліотеки @nuxtjs/auth та JWT токенів, що забезпечує надійний захист даних. Було розроблено проміжне програмне забезпечення, що включає важливі елементи аутентифікації та авторизації користувачів, а також перевірки прав доступу до ресурсів системи. Це дозволило створити надійний захист від несанкціонованих дій та забезпечити правильний розподіл прав доступу.

Клієнтська частина системи була розроблена з використанням Nuxt 3, що забезпечило створення сучасного та зручного інтерфейсу користувача. Було реалізовано кілька важливих Vue компонентів, таких як AutoComplete та DynamicForm, що підвищило зручність використання системи.

Інтеграція зі Scopus дозволила автоматично отримувати наукові публікації та додавати їх до бази даних додатку, що значно спростило процес ведення реєстру досягнень викладачів. Система балів для викладачів та кафедр була реалізована з використанням коефіцієнтів для різних категорій досягнень, що забезпечило точність та справедливість розрахунків.

Отже, у ході виконання дипломної роботи було успішно реалізовано систему реєстру досягнень викладачів, яка відповідає всім визначеним вимогам та забезпечує позитивний користувацький досвід. Розроблена система є надійною, масштабованою та зручною у використанні, що відповідає сучасним вимогам до веб-додатків.

Подальші покращення системи можуть включати ускладнення алгоритмів розрахунку балів, інтеграцію з додатковими зовнішніми сервісами, розширення функціональності для створення звітів і аналітики, покращення користувацького інтерфейсу для підвищення зручності використання, а також впровадження додаткових заходів безпеки та конфіденційності, таких як двофакторна аутентифікація.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. MongoDB Manual. https://www.mongodb.com/docs/manual/

2. MongoDB Atlas Documentation. https://www.mongodb.com/docs/atlas/

3. Elsevier API Documentation. https://dev.elsevier.com/

4. Nuxt.js Documentation. https://nuxt.com/docs/getting-started/introduction

5. Nuxt.js Auth Module Documentation. https://auth.nuxtjs.org/

6. Auth0 JSON Web Token Documentation. https://github.com/auth0/node-jsonwebtoken#readme

7. Mongoose Documentation. https://mongoosejs.com/docs/

8. Academia.edu. https://www.academia.edu/

9. ResearchGate. https://www.researchgate.net/

10. Scopus. https://www.scopus.com/

11. Стаття "Solving the Many-to-Many Relationship Challenge in MongoDB". https://medium.com/mongodb-tutorial/solving-the-many-to-many-relationship-challenge-in-mongodb-125b0421b6f0

12. NextAuth.js Guides. https://github.com/nextauthjs/next-auth/tree/main/docs/pages/guides

13. Create Elsevier API Key. https://dev.elsevier.com/apikey/create

14. Nuxt 3 Tailwind Kit: AutoComplete Component. https://nuxt3-tailwind-kit.vercel.app/docs/components/autocomplete

15. Headless UI Vue Combobox. https://headlessui.com/v1/vue/combobox