МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА СИСТЕМИ ОБЛІКУ НАВЧАЛЬНОГО НАВАНТАЖЕННЯ»**

Виконав: студент 4 курсу, групи 6.1219-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Д.Р. Таран
(ініціали та прізвище)

Керівник декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ 07 ” 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Тарану Дмитру Романовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка системи обліку навчального навантаження

керівник роботи Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Проєктування.
3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.02.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.02.2023	
2.	Збір вихідних даних.	15.02.2023	
3.	Обробка методичних та теоретичних	01.03.2023	
	джерел.		
4.	Розробка першого та другого розділу.	03.04.2023	
5.	Розробка третього розділу.	15.05.2023	
6.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	21.06.2023	

Студент _____ Д.Р. Таран
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка системи обліку навчального навантаження»: 73 с., 24 рис., 10 джерел, 4 додатки.

ANGULAR, API, FRAMEWORK, LARAVEL, MVC, RXJS, TYPESCRIPT, UML.

Об’єкт дослідження – система, інструменти для взаємодії Angular та Laravel, засоби взаємодії системи з користувачем.

Мета роботи – розробити систему обліку навчального навантаження.

Методи дослідження – моделювання, проєктування, програмний, аналітичний.

Системи для обліку навчального навантаження стають невід’ємною частиною в забезпеченні інформації та функціоналу щодо контролю та моніторингу навчального навантаження закладу вищої оствіти. Знайдеться небагато закладів, котрі б не користувалися послугами таких систем. За допомогою них користувачі мають можливість вести облік навчального навантаження, формувати звіти, перегляди детальну статистику та оптимізувати свою роботу.

Як правило, подібні системи мають різний функціонал, який охоплює різні аспекти взаємодії з навчальним навантаженням. Але відсутність або обмеженість потрібного функціоналу не є виключенням. Одним з прикладів такої проблеми є неможливість керування власними записами про навчальне навантаження.

Таким чином, за результатами роботи створено зручну та ефективну систему обліку навчального навантаження з використанням Angular та Laravel.

SUMMARY

Bachelor’s qualifying paper «Development of a Study Hours Accounting System»: 73 pages, 24 figures, 10 references, 4 supplements.

ANGULAR, API, FRAMEWORK, LARAVEL, MVC, RXJS, TYPESCRIPT, UML.

The object of the study is the system, tools for Angular and Laravel interaction, means of system interaction with the user.

The aim of the study is to develop an educational workload accounting system.

The methods of research are modeling, design, programming, analytical.

Educational workload accounting systems have become an integral part of providing information and functionality for monitoring and controlling the educational workload of higher education institutions. There are few institutions that do not use such systems. With their help, users have the ability to track educational workload, generate reports, view detailed statistics, and optimize their work.

Typically, such systems have different functionality that covers various aspects of interaction with educational workload. However, the absence or limitation of necessary functionality is not uncommon. One example of such a problem is the inability to manage one’s own records of educational workload.

Thus, as a result of the study, a convenient and efficient educational workload accounting system has been developed using Angular and Laravel.

ЗМІСТ

ВСТУП

У сучасному світі вищої освіти, університети знаходяться перед постійним викликом – ефективно керувати та розподіляти навчальне навантаження серед своїх викладачів. Для вирішення цієї проблеми стає все більш актуальним впровадження ефективної системи обліку навчального навантаження.

Однією з основних переваг системи обліку навчального навантаження є можливість забезпечення справедливого розподілу навчального навантаження серед викладачів. Вона дозволяє враховувати кваліфікацію, досвід, академічні обов’язки та інші фактори, що впливають на обсяг роботи, і гарантує, що навантаження розподіляється рівномірно та справедливо.

Система обліку навчального навантаження допомагає університетам оптимізувати використання своїх ресурсів, зокрема людських та матеріальних. Вона дозволяє виявляти можливості для раціонального розподілу навантаження, уникнення перевантаження або недостатку роботи, та забезпечує максимальну ефективність роботи викладачів.

Виходячи з цього, було вирішено створити систему, котра би мала зрозумілий інтерфейс та надавала гнучкий функціонал користувачам.

Актуальність дослідження: актуальність теми зумовлена необхідністю оптимізації та уніфікації процесу обліку навчального навантаження, а також можливості взаємодії з ним на різних рівнях.

З огляду на це, можна виділити наступні цілі і задачі нашого дослідження:

Мета: розробити систему обліку навчального навантаження.

Задачі:

1) сформувати вимоги до системи;
2) спроєктувати та побудувати архітектуру системи;
3) реалізувати систему обліку навчального навантаження;
4) протестувати роботу системи.

Об’єкт дослідження: процес обліку навантаження користувачем,

інструменти для реалізації системи навчального навантаження, функціонал системи, необхідний користувачам.

Предмет дослідження: створення системи, що дозволяє користувачам вести облік навчального навантаження.

Методи дослідження: моделювання, проєктування, програмний, аналітичний.

Перший розділ присвячено збору та аналізуванню вимог до системи, огляду інструментів розробки та опису системи.

У другому розділі розглянуто етапи проєктування системи, наведено детальний опис прецедентів.

Третій розділ присвячено реалізації та тестуванню роботи системи, наведено детальне керівництво користувача, яке описує процес роботи з системою обліку навчального навантаження.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Терміни та визначення

1.1.1 Загальні терміни

Система – веб-сайт створений на основі Angular та Laravel.

Angular – це фреймворк для створення веб-додатків, який базується на TypeScript. Angular дозволяє створювати потужні та масштабовані додатки з використанням компонентної архітектури та забезпечує розробникам ряд інструментів для зручної роботи.

Laravel – PHP-фреймворк, призначений для розробки веб-додатків відповідно до шаблона model–view–controller.

MySQL – реляційна система керування базами даних.

ДВІ – Діаграма Варіантів Використання чи Use Case Diagram.

ДД – Діаграма Діяльності.

ДП – Діаграма Послідовності.

Користувач – людина, котра зареєстрована у системі та має певні права доступу.

Адміністратор – користувач, котрий має права на редагування системи.

1.1.2 Технічні терміни

ІС – інформаційна система.

БД – база даних, місце збереження інформації ІС.

MVC – архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення.

1.2 Функціональні вимоги

1.2.1 Призначення і цілі створення системи

Функціональне призначення системи – реалізувати можливість обліку навчального навантаження.

Експлуатаційне призначення системи: система може експлуатуватися адміністратором і користувачем системи на різних рівнях доступу.

Мета створення системи – розробка системи обліку навчального навантаження.

1.2.2 Загальні функціональні можливості системи

Система має надавати користувачам такі можливості:

- створення/редагування/перегляд/видалення/прив’язка дисциплін;
- створення/редагування/перегляд/видалення тем;
- створення/редагування/перегляд/видалення навантаження;
- створення звіту навантаження;
- вхід/вихід до/з системи.

Система має надавати адміністраторам такі можливості:

- створення/редагування/перегляд/видалення користувачів;
- створення/редагування/перегляд/видалення кафедр;
- створення/редагування/перегляд/видалення факультетів.

1.3 Нефункціональні вимоги

1.3.1 Інтерфейс користувача

Система повинна відображати коректно інтерфейс користувача на будь-якому пристрої.

1.3.2 Підтримка браузерів

Система повинна працювати для наступних браузерів останніх версій:

- Mozilla Firefox;
- Google Chrome;
- Safari;
- Opera.

1.3.3 Вимоги до продуктивності

Система повинна відображати будь-яку сторінку не довше, ніж за 2 секунди.

Система повинна відправляти повідомлення не довше, ніж 2 секунди.

1.3.4 Вимоги до безпеки

Система не повинна дозволяти не адміністраторам фізичний доступ до інтерфейсу адміністратора.

Система не повинна надавати доступ неавторизованим користувачам доступ до даних системи.

1.4 Опис предметної області

Предметною областю є розробка системи для обліку навчального навантаження. Дана система повинна надавати користувачам можливість ведення обліку власного навчального навантаження та керувати ним. Процес взаємодії з системою проходить наступним чином. Користувач, повинен ввести адресу сайту в браузері та ввести дані в форму входу до системи. Якщо користувач вперше на сайті, він повинен спочатку отримати свій логін та пароль у адміністратора системи.

Після входу до системи користувач має можливість взаємодіяти з такими розділами як: дисципліни, теми, навантаження, форма, звіти. Перед початком внесення навчального навантаження користувач повинен створити дисципліни та теми.

Процесс внесения даних навачального навантаження проходить наступним чином. Користувач обирає розділ «Форма», система відображає інтерфейс взаємодії з даним розділом. Для створення нового запису користувач заповнює форму, яка складається з 3-х розділів: інформація про викладача, інформація про заняття і про групу. Після введення даних користувач натискає кнопку «Зберегти», система зберігає введені дані, у розділі «Навантаження» з’являється новий запис навчального навантаження користувача. Ці дані надалі будуть використовуватися для відображення статистичних даних та при формуванні звітів.

Окрім основних функцій система надає адміністраторам можливість керувати обліковими записами користувачів, факультетами, кафедрами та переглядати звіти за обраними налаштуваннями.

1.5 Опис системи

Ведення звітності є ключовою та невід'ємною частиною функціонування вищого навчального закладу, одна з основних категорій звітності – облік

начального навантаження. Зазвичай він ведеться в паперовому вигляді, кожною кафедрою окремо, що призводить до багатьох проблем, наприклад: неможливість фізично бути присутнім у навчальному закладі, немає уніфікованості записів, тощо. У зв’язку з цим велику актуальність набуває розробка систем, які дозволяють вести облік начального навантаження.

Системи для обліку навчального навантаження стають невід’ємною частиною в забезпеченні інформації та функціоналу щодо контролю та моніторингу навчального навантаження закладу вищої оствіти. Знайдеться небагато закладів, котрі б не користувалися послугами таких систем. За допомогою них користувачі мають можливість вести облік навчального навантаження, формувати звіти, перегляди детальну статистику та оптимізувати свою роботу.

Однією з таких систем є «Облік навчального навантаження». Нова система для ведення обліку навчального навантаження.

Можливості системи:

- основні:
  - керування дисциплінами/темами/навантаженням;
  - формування звітів;
  - детальна статистика;
- консоль адміністратора:
  - керування обліковими записами/кафедрами/факультетами.

1.6 Огляд інструментів розробки

1.6.1 Angular

Angular – це потужний та високопродуктивний фреймворк для розробки веб-додатків з використанням HTML, CSS та TypeScript. Angular заснований на принципах компонентної архітектури, де кожен елемент веб-додатку

представлений компонентом, який містить свою логіку та представлення. Компоненти Angular є самодостатніми, що дозволяє їх використовувати в різних частинах додатку, що підвищує його масштабованість та підтримку.

У фреймворку Angular є низка переваг. Розглянемо основні з них.

Декларативність: Angular використовує декларативні шаблони для відображення даних, що дозволяє розробникам зосередитися на функціональному програмуванні та забезпечити легку та читабельну кодову базу.

Широкі можливості: Angular надає вбудовані директиви, такі як *ngIf та *ngFor, які дозволяють керувати відображенням даних на сторінці. Крім того, Angular надає розробникам можливість використовувати підключені бібліотеки, такі як RxJS, для роботи з асинхронними операціями та роботи зі стрімами даних.

Ін’єкція залежностей: Angular має потужну систему ін'єкції залежностей, яка дозволяє забезпечити легку тестовість додатку та збільшити його модульність. Розробникам не потрібно прямо включати залежності в клас, вони можуть бути включені за допомогою ін’єкції.

Маршрутизація: Angular має потужну систему маршрутизації, яка дозволяє розробникам змінювати сторінки без перезавантаження сторінки. Це дозволяє забезпечити користувачам більш зручний та швидкий досвід взаємодії з додатком.

Підтримка мобільних пристроїв: Angular дозволяє розробникам створювати мобільні додатки з використанням іонів (Ionic), який є фреймворком для розробки гібридних мобільних додатків на базі Angular.

Узагальненість: Angular дозволяє розробникам створювати універсальні додатки, які можуть працювати як на клієнтському боці, так і на серверному. Це дозволяє розробникам забезпечувати більшу кількість можливостей для оптимізації додатку та його підтримки.

Angular є популярним вибором для розробки веб-додатків, особливо для більш складних та масштабних проєктів. За допомогою Angular розробники можуть створювати високопродуктивні додатки, які мають легку та читабельну

кодову базу, а також підтримку мобільних пристроїв та інші переваги, які забезпечують більш зручний та ефективний досвід взаємодії з додатком.

1.6.2 Laravel

Laravel – це безкоштовний та відкритий фреймворк для розробки веб-додатків, написаний на мові програмування PHP. Він заснований на принципах модульності, масштабованості та легкості використання. Laravel надає готові рішення для низки завдань, які часто виникають при розробці веб-додатків, таких як маршрутизація, обробка запитів, робота з базами даних та автентифікація.

У фреймворку Laravel є низка переваг. Розглянемо основні з них.

Елегантний та інтуїтивно зрозумілий синтаксис: Laravel надає простий та легкий для використання синтаксис, що дозволяє розробникам швидко створювати функціонал веб-додатків.

Архітектура MVC: Laravel використовує архітектуру Model-View-Controller (MVC), що дозволяє відділити логіку додатку від представлення та бізнес-логіки.

Потужний маршрутизатор: Laravel має потужний маршрутизатор, який дозволяє легко налаштовувати маршрути для додатку.

Підтримка баз даних: Laravel має вбудовану підтримку для різних типів баз даних, таких як MySQL, PostgreSQL та SQLite, що дозволяє легко працювати з базами даних та виконувати різноманітні операції.

Розширюваність: Laravel має велику кількість розширень та пакетів, що дозволяє розробникам швидко додавати новий функціонал до своїх додатків.

Безпека: Laravel має вбудовану підтримку для різних методів аутентифікації та захисту від атак, що дозволяє забезпечити безпеку додатку та його користувачів.

1.6.3 MySQL

MySQL – це система керування базами даних (СКБД), що дозволяє зберігати, організовувати та забезпечувати доступ до великої кількості даних у веб-додатках та інших програмах.

MySQL є безкоштовним та відкритим програмним забезпеченням, що дозволяє розробникам використовувати його безкоштовно та редагувати вихідний код [9].

Основні переваги MySQL:

- висока продуктивність: MySQL може ефективно обробляти великі обсяги даних та забезпечувати швидкий доступ до них;
- надійність та стабільність: MySQL є дуже стабільною та надійною СКБД, яка дозволяє зберігати дані безпечно та без втрати;
- підтримка стандартів: MySQL підтримує багато стандартів, таких як SQL та ODBC, що дозволяє легко інтегрувати його з різними програмами та додатками;
- масштабованість: MySQL може бути легко масштабований, що дозволяє збільшувати його продуктивність залежно від потреб;
- зручний для використання: MySQL має простий та інтуїтивно зрозумілий інтерфейс користувача, що дозволяє легко створювати та змінювати бази даних;
- безпека: MySQL надає вбудовану підтримку для шифрування та аутентифікації, що забезпечує безпеку даних у базах даних.

2 ПРОЄКТУВАННЯ

2.1 Використання UML під час розробки системи

Використання UML (Unified Modeling Language) під час розробки системи є ключовим елементом для успішного проєктування та впровадження програмного забезпечення. UML надає стандартизований набір графічних засобів та нотацій, які допомагають команді розробників зрозуміти, визначити та моделювати систему перед її реалізацією.

Однією з переваг використання UML є його універсальність. Він може бути використаний для розробки різних типів систем, включаючи програмне забезпечення, апаратне забезпечення, бізнес-процеси та інші. Це дозволяє команді розробників зрозуміти систему в цілому та виявити взаємозв'язки між її складовими частинами.

Одним з основних елементів UML є діаграми, які візуалізують різні аспекти системи. Наприклад, діаграма класів дозволяє визначити структуру системи, класи, атрибути та методи, а також взаємозв'язки між ними. Діаграма компонентів допомагає виділити компоненти системи та їх залежності. Діаграма послідовності відображає послідовність виконання повідомлень між об'єктами та систему в цілому.

Використання UML сприяє чіткому розумінню вимог до системи та специфікацій. Він дозволяє команді розробників визначити функціональні та нефункціональні вимоги, ідентифікувати ключові сценарії використання та пріоритети. Завдяки цьому, всі учасники проєкту мають однакове уявлення про систему та вимоги до неї.

UML також допомагає виявляти та усувати проблеми та помилки ще на ранніх етапах розробки. Шляхом використання діаграм UML команда розробників може виявити потенційні проблеми в архітектурі системи, взаємозв'язках між компонентами, а також ідентифікувати можливі конфлікти та

несупадання між вимогами та реалізацією.

У процесі розробки системи, діаграми UML допомагають команді розробників узгодити свої рішення та спільно працювати над вирішенням технічних проблем. Вони створюють спільну мову та зрозумілість між всіма учасниками проєкту, включаючи аналітиків, розробників, тестувальників та замовників.

Діаграми UML також допомагають команді розробників зрозуміти процеси та поведінку системи. Наприклад, діаграма послідовності відображає послідовність взаємодії між об'єктами та дозволяє виявити можливі помилки або неочікувані становища. Діаграма станів допомагає моделювати різні стани об'єктів та їх переходи, що сприяє виявленню потенційних проблем з логікою системи.

Уміння користуватись UML також сприяє покращенню комунікації між командою розробників та замовником. Графічне представлення системи у вигляді діаграм дозволяє замовнику краще зрозуміти, яким чином система буде функціонувати та які можливості вона надає. Це сприяє виявленню можливих розбіжностей між очікуваннями замовника та побудованою системою ще на ранніх етапах розробки.

2.2 Діаграма варіантів використання

Діаграма варіантів використання (Use Case Diagram) є одним із ключових інструментів методології UML (Unified Modeling Language), який допомагає моделювати та розуміти взаємодію системи з її зовнішнім середовищем. Цей графічний засіб відображає акторів (користувачів або зовнішні системи) та варіанти використання, які представляють сценарії взаємодії між акторами та системою.

Діаграма варіантів використання є потужним засобом для визначення та уточнення вимог до системи. Вона дозволяє команді розробників програмного

забезпечення зрозуміти, які функції повинна виконувати система, які сценарії взаємодії мають бути підтримані та які актори взаємодіють з системою.

На діаграмі варіантів використання актори зображуються у вигляді піктограм, таких як людина, зовнішня система або інший компонент, що взаємодіє з системою. Варіанти використання відображаються у вигляді овалів або прямокутників і представляють сценарії дій акторів з системою. Зв'язки між акторами та варіантами використання показують способи взаємодії.

Переваги використання діаграми варіантів використання очевидні. По-перше, вона дозволяє збагнути контекст системи та її основні функції, що допомагає уточнити вимоги до програмного забезпечення. Вона служить основою для подальшого аналізу та проєктування системи.

По-друге, діаграма варіантів використання сприяє виявленню потенційних проблем та протиріч у системі, а також допомагає прорахувати можливі ризики та виробити ефективні стратегії для їх управління. Вона дає змогу команді розробників зосередитись на основних функціях системи та забезпечити відповідність до вимог користувачів.

Крім того, діаграма варіантів використання є зручним засобом комунікації між розробниками, замовниками та іншими зацікавленими сторонами. Вона допомагає уточнити сподівання та очікування стосовно функціональності системи і забезпечує загальне розуміння проєкту.

Однак, важливо зазначити, що діаграма варіантів використання не є вичерпною та деталізованою моделлю системи. Вона слугує вихідним пунктом для подальшого аналізу та проєктування. Для отримання повної картини системи можуть бути потрібні інші діаграми UML, такі як діаграма класів, діаграма послідовностей та діаграма станів.

На рисунку 2.1 представлена діаграма варіантів використання.

На діаграмі представлено актора «Адміністратор», який безпосередньо є користувачем з розширеними права на взаємодію з системою. Також представлений актор «Користувач», який відображає спільні функціональні можливості інших акторів.

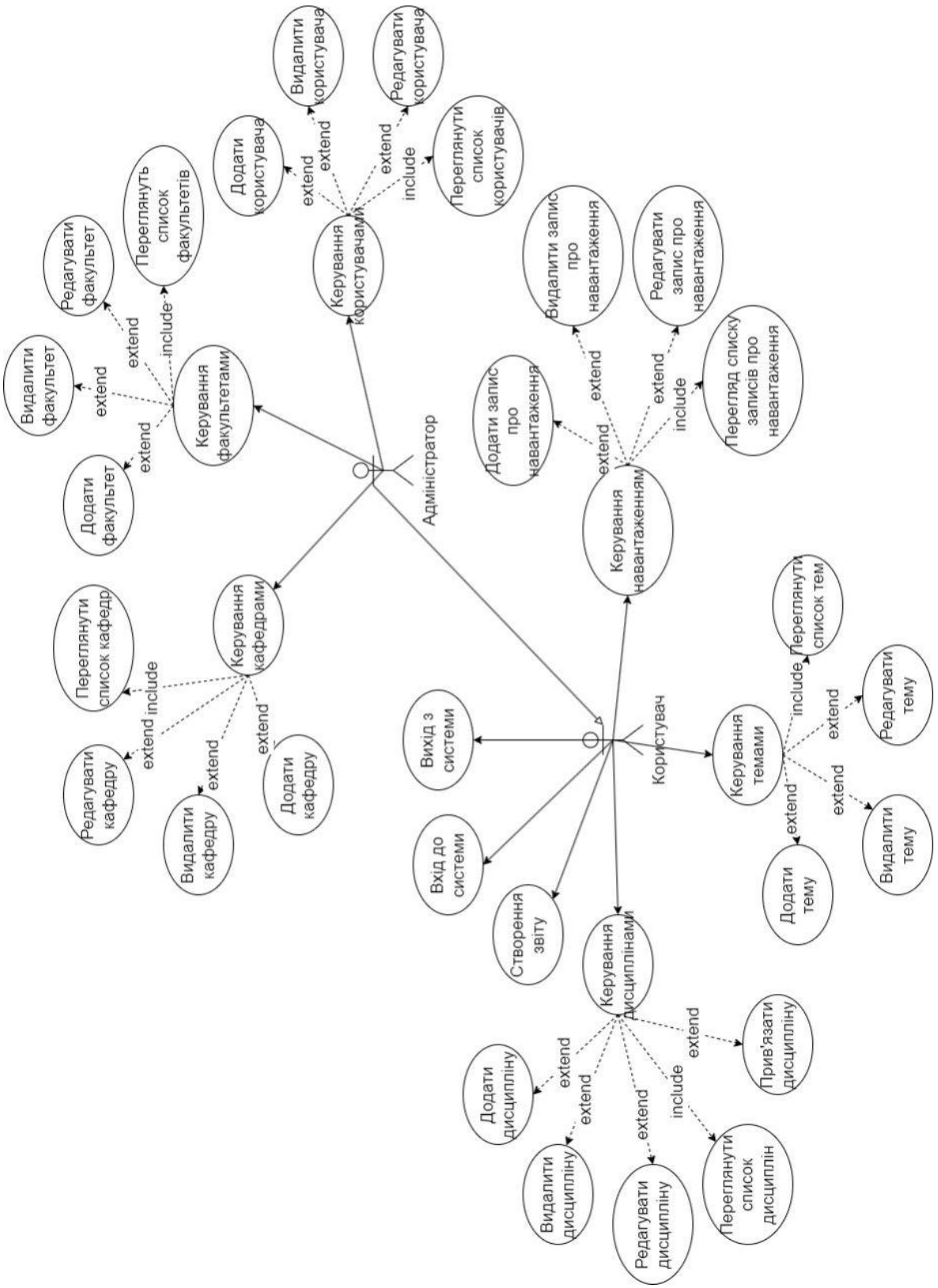

Рисунок 2.1 – Діаграма варіантів використання

Виділено 1 основний варіант використання – «керування навантаженням». Після того, як користувач авторизується у системі та перейде у розділ «Форма», система відображає форму для введення даних щодо навчального навантаження. Форма динамічно реагує на зміну ключових полів та видозмінюється залежно від обраного виду навчального навантаження. Після заповнення форми та збереження даних, користувач має можливість перейти до розділу «Записи», де відображається інформація про всі записи навчального навантаження користувача. Також на даній сторінці користувач має можливість застосовувати різні фільтри для відображення інформації, а також здійснювати такі дії як: додавання, видалення, редагування записів. На кожну з цих дій система посилає запит до БД і повідомляє користувача про результат дії.

Варіанти використання визначають функціональні можливості. Кожен з них представляє певний спосіб використання. Таким чином, кожен варіант використання відповідає послідовності дій для того, щоб користувач міг отримати певний результат (див. рис. 2.2 – 2.3).

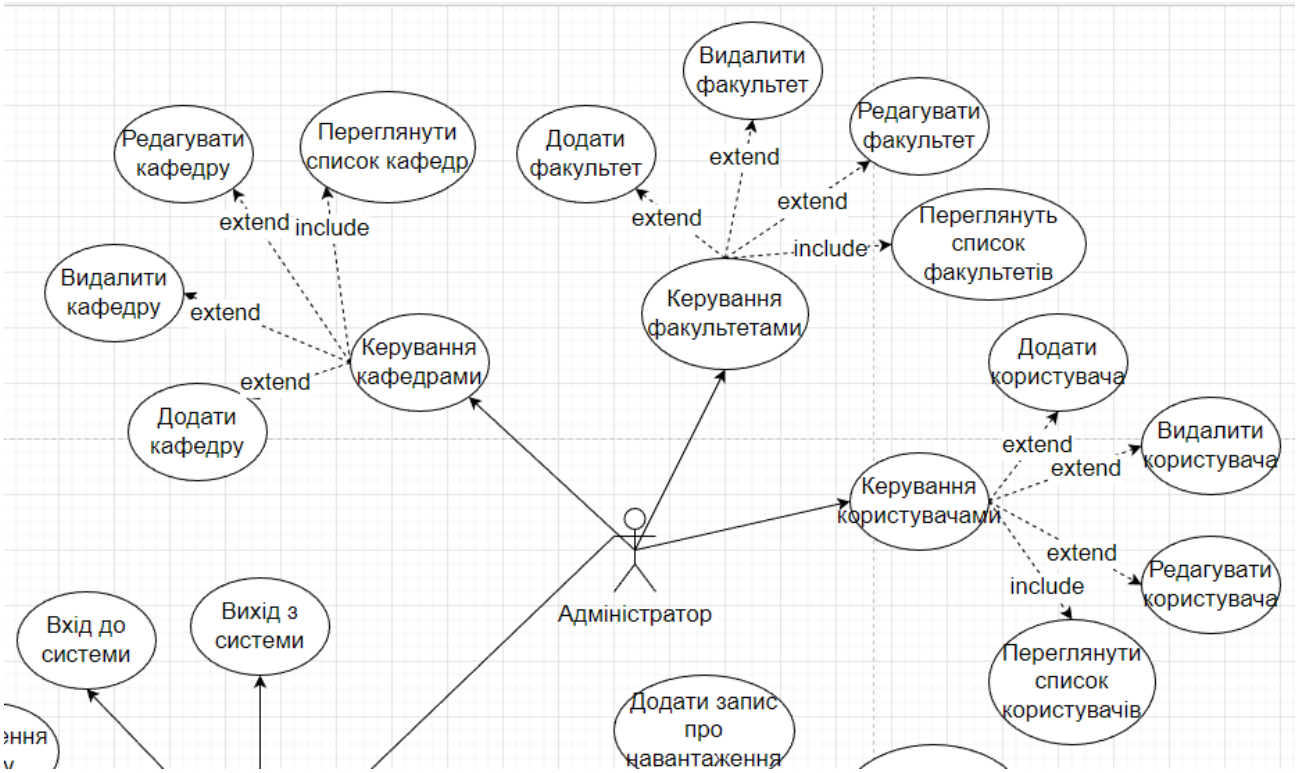

Рисунок 2.2 – Діаграма варіантів використання «Адміністратор»

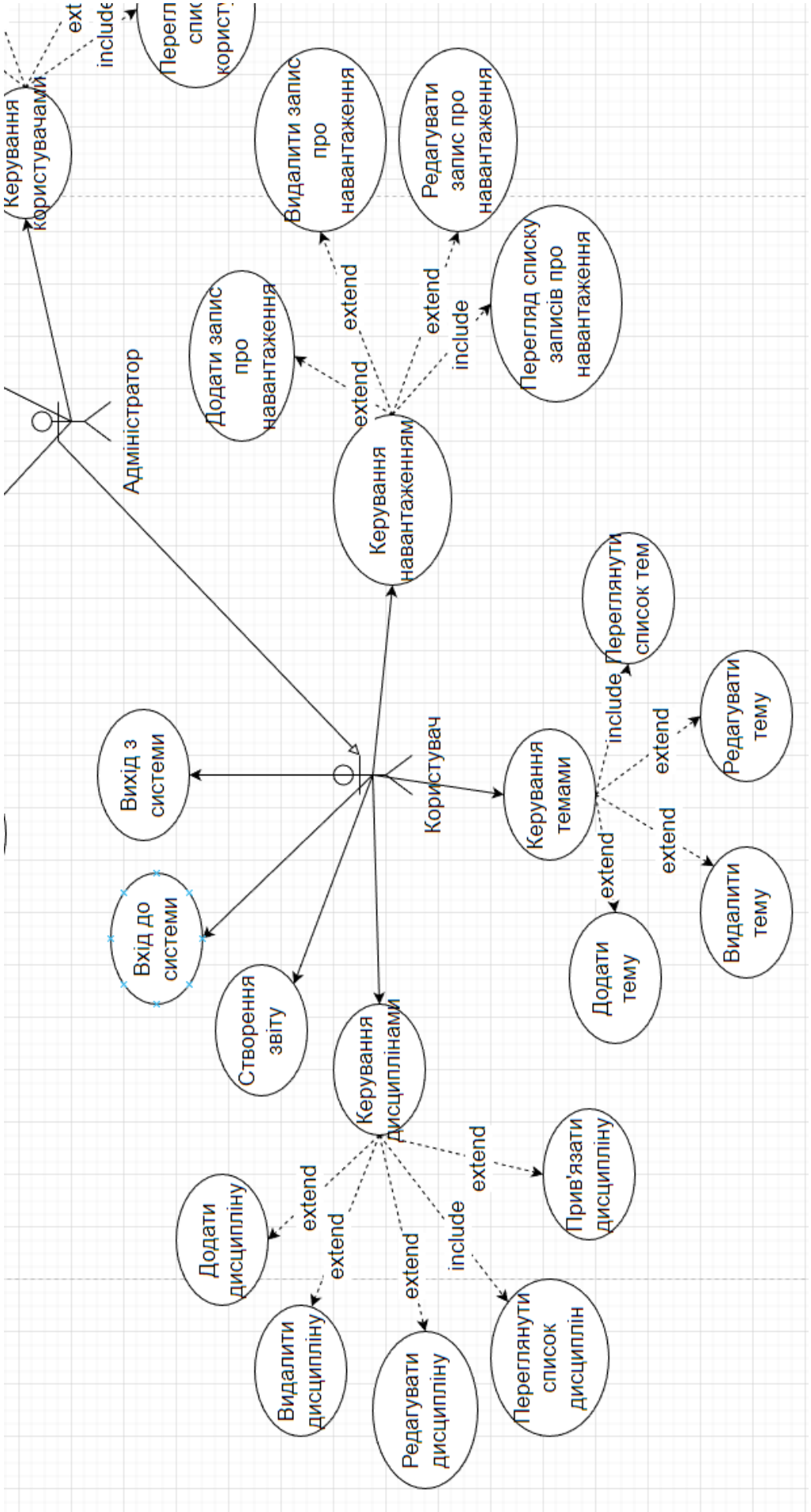

Рисунок 2.3 – Діаграма варіантів використання «Користувач

2.2.1 Опис варіантів використання

Прецедент «Вхід до системи»

Призначення: даний варіант використання надає можливість зареєстрованому користувачу увійти у систему для використання.

Основний потік подій: даний варіант використання починає виконуватися, коли зареєстрованому користувачу потрібно авторизуватися. Система пропонує ввести логін та пароль. Після того, як користувач ввів їх, система перевіряє правильність введених даних, і у випадку вдачі надає йому функціонал.

Альтернативний потік: якщо логін чи пароль невірні – система сповіщає про це користувача. Користувач може спробувати ще раз пройти авторизацію.

Передумова: перед початком виконання даного варіанта використання користувач повинен бути зареєстрований у системі.

Прецедент «Вихід з системи»

Призначення: даний варіант використання надає можливість користувачу вийти з системи.

Основний потік подій: даний варіант використання починає виконуватися, коли користувачу потрібно вийти з системи. Користувач натискає на значок профіля та у списку дій обирає «Вихід». Після того, як користувач вийшов, система відображає головну сторінку.

Передумова: перед початком виконання даного варіанта використання користувач повинен авторизуватися в системі.

Прецедент «Керування дисциплінами»

Призначення: даний варіант використання надає можливість користувачу взаємодіяти з дисциплінами згідно з системною роллю.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Дисципліни» в особистому кабінеті. Система відображає список дисциплін та інструменти взаємодії з ними згідно з системною роллю.

Альтернативний потік подій: якщо жодної дисципліни ще не було додано,

система сповістить про це у вигляді напису «Немає жодного запису».

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Додати дисципліну»

Призначення: даний варіант використання надає можливість користувачу додавати нові дисципліни.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає кнопку «Додати», система відображає форму створення нової дисципліни. Після заповнення форми валідними даними, користувач натискає кнопку «Зберегти», система зберігає нову дисципліну та відображає сторінку «Дисципліни».

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Дисципліни».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Користувач може змінити дані.

Виняткова ситуація 2: користувач натиснув кнопку «Скасувати» – система відобразить сторінку «Дисципліни».

Прецедент «Видалити дисципліну»

Призначення: даний варіант використання надає можливість користувачу видалити дисципліну.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає піктограму кошику в рядку дисципліни, система видаляє дисципліну.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Дисципліни».

Прецедент «Редагувати дисципліну»

Призначення: даний варіант використання надає можливість користувачу редагувати дисципліну.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на піктограму олівця в рядку дисципліни,

система відображає інтерфейс редагування дисципліни. Після заповнення форми валідними даними, користувач натискає кнопку «Зберегти», система зберігає оновлені дані та відображає сторінку «Дисципліни».

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Дисципліни».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Користувач може змінити дані.

Виняткова ситуація 2: користувач натиснув кнопку «Скасувати» – система відобразить сторінку «Дисципліни».

Прецедент «Переглянути список дисциплін»

Призначення: даний варіант використання надає можливість користувачу переглядати список дисциплін згідно з системною роллю.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Дисципліни» в особистому кабінеті. Система відображає список дисциплін згідно з системною роллю.

Альтернативний потік подій: якщо жодної дисципліни ще не було додано, система сповістить про це у вигляді напису «Немає жодного запису».

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Прив’язати дисципліну»

Призначення: даний варіант використання надає можливість користувачу прив’язати існуючу дисципліну до облікового запису.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи вводить ідентифікатор дисципліни у поле та натискає кнопку «Прив’язати», система зв’язує дисципліну з обліковим записом та відображає сторінку «Дисципліни».

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Дисципліни».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Користувач може змінити дані.

Виняткова ситуація 2: введено неіснуючий ідентифікатор – система відобразить відповідне повідомлення. Користувач може змінити дані.

Прецедент «Керування темами»

Призначення: даний варіант використання надає можливість користувачу взаємодіяти з темами згідно з системною роллю.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Теми» в особистому кабінеті. Система відображає список тем та інструменти взаємодії з ними згідно з системною роллю.

Альтернативний потік подій: якщо жодної теми ще не було додано, система сповістить про це у вигляді напису «Немає жодного запису».

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Додати тему»

Призначення: даний варіант використання надає можливість користувачу додавати нові теми.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає кнопку «Додати», система відображає форму створення нової теми. Після заповнення форми валідними даними, користувач натискає кнопку «Зберегти», система зберігає нову тему та відображає сторінку «Теми».

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Теми», а також повинна бути створена мінімум одна дисципліна.

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Користувач може змінити дані.

Виняткова ситуація 2: користувач натиснув кнопку «Скасувати» – система відобразить сторінку «Теми».

Виняткова ситуація 3: користувач не створив або не прив’язав жодної дисципліни – система блокує селектор з переліком дисциплін і кнопку

«Зберегти».

Прецедент «Видалити тему»

Призначення: даний варіант використання надає можливість користувачу видалити тему.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає піктограму кошику в рядку теми, система видаляє тему.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Теми».

Прецедент «Редагувати тему»

Призначення: даний варіант використання надає можливість користувачу редагувати тему.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на піктограму олівця в рядку теми, система відображає інтерфейс редагування теми. Після заповнення форми валідними даними, користувач натискає кнопку «Зберегти», система зберігає оновлені дані та відображає сторінку «Теми».

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Теми».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Користувач може змінити дані.

Виняткова ситуація 2: користувач натиснув кнопку «Скасувати» – система відобразить сторінку «Теми».

Прецедент «Переглянути список тем»

Призначення: даний варіант використання надає можливість користувачу переглядати список тем згідно з системною роллю.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Теми» в особистому кабінеті. Система відображає список тем згідно з системною роллю.

Альтернативний потік подій: якщо жодної теми ще не було додано,

система сповістить про це у вигляді напису «Немає жодного запису».

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Керування навантаженням»

Призначення: даний варіант використання надає можливість користувачу взаємодіяти з записами про навчальне навантаження згідно з системною роллю.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Записи» в особистому кабінеті. Система відображає список записів про навчальне навантаження та інструменти взаємодії з ними згідно з системною роллю.

Альтернативний потік подій: якщо жодного запису ще не було додано, система сповістить про це у вигляді напису «Немає жодного запису».

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Додати запис про навантаження»

Призначення: даний варіант використання надає можливість користувачу додавати нові записи про навчальне навантаження.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає кнопку «Додати» у розділі «Записи» або натискає на кнопку «Форма» в боковому меню, система відображає форму створення нового запису про навчальне навантаження. Після заповнення форми валідними даними, користувач натискає кнопку «Зберегти», система зберігає новий запис та відображає сторінку «Форма».

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Записи».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Користувач може змінити дані.

Виняткова ситуація 2: користувач натиснув кнопку «Скасувати» – система відобразить сторінку «Записи».

Прецедент «Видалити запис про навантаження»

Призначення: даний варіант використання надає можливість користувачу видалити запис про навантаження.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає піктограму кошику в рядку запису навчального навантаження, система видаляє запис.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Записи».

Прецедент «Редагувати запис про навантаження»

Призначення: даний варіант використання надає можливість користувачу редагувати запис про навчальне навантаження.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на піктограму олівця в рядку запису навчального навантаження, система відображає інтерфейс редагування запису. Після заповнення форми валідними даними, користувач натискає кнопку «Зберегти», система зберігає оновлені дані та відображає сторінку «Записи».

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Записи».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Користувач може змінити дані.

Виняткова ситуація 2: користувач натиснув кнопку «Скасувати» – система відобразить сторінку «Записи».

Прецедент «Перегляд списку записів про навантаження»

Призначення: даний варіант використання надає можливість користувачу переглядати список записів про навчальне навантаження згідно з системною роллю.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Записи» в особистому кабінеті. Система відображає список записів згідно з системною роллю.

Альтернативний потік подій: якщо жодного запису ще не було додано, система сповістить про це у вигляді напису «Немає жодного запису».

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Керування факультетами»

Призначення: даний варіант використання надає можливість адміністратору взаємодіяти з факультетами.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає на кнопку «Факультети» в особистому кабінеті. Система відображає список факультетів та інструменти взаємодії з ними.

Альтернативний потік подій: якщо жодного факультету ще не було додано, система сповістить про це у вигляді напису «Немає жодного запису».

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи з правами адміністратора.

Прецедент «Додати факультет»

Призначення: даний варіант використання надає можливість адміністратору додавати нові факультети.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає кнопку «Додати», система відображає форму створення нового факультету. Після заповнення форми валідними даними, адміністратор натискає кнопку «Зберегти», система зберігає новий факультет та відображає сторінку «Факультети».

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Факультети».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Адміністратор може змінити дані.

Виняткова ситуація 2: адміністратор натиснув кнопку «Скасувати» – система відобразить сторінку «Факультети».

Прецедент «Видалити факультет»

Призначення: даний варіант використання надає можливість адміністратору видалити факультет.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає піктограму кошику в рядку факультету, система видаляє факультет.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Факультети».

Прецедент «Редагувати факультет»

Призначення: даний варіант використання надає можливість адміністратору редагувати факультет.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає на піктограму олівця в рядку факультету, система відображає інтерфейс редагування факультету. Після заповнення форми валідними даними, адміністратор натискає кнопку «Зберегти», система зберігає оновлені дані та відображає сторінку «Факультети».

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Факультети».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Адміністратор може змінити дані.

Виняткова ситуація 2: адміністратор натиснув кнопку «Скасувати» – система відобразить сторінку «Факультети».

Прецедент «Переглянути список факультетів»

Призначення: даний варіант використання надає можливість адміністратору переглядати список факультетів.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає на кнопку «Факультети» в особистому кабінеті. Система відображає список факультетів.

Альтернативний потік подій: якщо жодного факультету ще не було додано, система сповістить про це у вигляді напису «Немає жодного запису».

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи з правами адміністратора.

Прецедент «Керування кафедрами»

Призначення: даний варіант використання надає можливість адміністратору взаємодіяти з кафедрами.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає на кнопку «Кафедри» в особистому кабінеті. Система відображає список кафедр та інструменти взаємодії з ними.

Альтернативний потік подій: якщо жодної кафедри ще не було додано, система сповістить про це у вигляді напису «Немає жодного запису».

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи з правами адміністратора.

Прецедент «Додати кафедру»

Призначення: даний варіант використання надає можливість адміністратору додавати нові кафедри.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає кнопку «Додати», система відображає форму створення нової кафедри. Після заповнення форми валідними даними, адміністратор натискає кнопку «Зберегти», система зберігає нову кафедру та відображає сторінку «Кафедри».

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Кафедри».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Адміністратор може змінити дані.

Виняткова ситуація 2: адміністратор натиснув кнопку «Скасувати» – система відобразить сторінку «Кафедри».

Прецедент «Видалити кафедру»

Призначення: даний варіант використання надає можливість адміністратору видалити кафедру.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає піктограму кошику в рядку кафедри, система видаляє кафедру.

Передумова: перед початком виконання даного варіанта використання

адміністратор повинен знаходитися в розділі «Кафедри».

Прецедент «Редагувати кафедру»

Призначення: даний варіант використання надає можливість адміністратору редагувати кафедру.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає на піктограму олівця в рядку кафедри, система відображає інтерфейс редагування кафедри. Після заповнення форми валідними даними, адміністратор натискає кнопку «Зберегти», система зберігає оновлені дані та відображає сторінку «Кафедри».

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Кафедри».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Адміністратор може змінити дані.

Виняткова ситуація 2: адміністратор натиснув кнопку «Скасувати» – система відобразить сторінку «Кафедри».

Прецедент «Переглянути список кафедр»

Призначення: даний варіант використання надає можливість адміністратору переглядати список кафедр.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає на кнопку «Кафедри» в особистому кабінеті. Система відображає список кафедр.

Альтернативний потік подій: якщо жодної кафедри ще не було додано, система сповістить про це у вигляді напису «Немає жодного запису».

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи з правами адміністратора.

Прецедент «Керування користувачами»

Призначення: даний варіант використання надає можливість адміністратору взаємодіяти з користувачами.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає на кнопку «Користувачі» в особистому

кабінеті. Система відображає список користувачів та інструменти взаємодії з ними.

Альтернативний потік подій: якщо жодного користувача ще не було додано, система сповістить про це у вигляді напису «Немає жодного запису».

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи з правами адміністратора.

Прецедент «Додати користувача»

Призначення: даний варіант використання надає можливість адміністратору додавати нових користувачів.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає кнопку «Додати», система відображає форму створення нового користувача. Після заповнення форми валідними даними, адміністратор натискає кнопку «Зберегти», система зберігає нового користувача та відображає сторінку «Користувачі».

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Користувачі».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Адміністратор може змінити дані.

Виняткова ситуація 2: адміністратор натиснув кнопку «Скасувати» – система відобразить сторінку «Користувачі».

Прецедент «Видалити користувача»

Призначення: даний варіант використання надає можливість адміністратору видалити користувача.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає піктограму кошику в рядку користувача, система видаляє користувача.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Користувачі».

Прецедент «Редагувати користувача»

Призначення: даний варіант використання надає можливість

адміністратору редагувати користувача.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає на піктограму олівця в рядку користувача, система відображає інтерфейс редагування користувача. Після заповнення форми валідними даними, адміністратор натискає кнопку «Зберегти», система зберігає оновлені дані та відображає сторінку «Користувачі».

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Користувачі».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Адміністратор може змінити дані.

Виняткова ситуація 2: адміністратор натиснув кнопку «Скасувати» – система відобразить сторінку «Користувачі».

Прецедент «Переглянути список користувачів»

Призначення: даний варіант використання надає можливість адміністратору переглядати список користувачів.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає на кнопку «Користувачі» в особистому кабінеті. Система відображає список користувачів.

Альтернативний потік подій: якщо жодного користувача ще не було додано, система сповістить про це у вигляді напису «Немає жодного запису».

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи з правами адміністратора.

Прецедент «Створення звіту»

Призначення: даний варіант використання надає можливість користувачу створювати звіт на основі записів про навчальне навантаження.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Звіти» в особистому кабінеті. Система відображає інтерфейс налаштування параметрів звіту. Після обрання параметрів, користувач натискає на кнопку «Згенерувати», система завантажує pdf-файл зі звітом.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

2.3 Діаграма діяльності

Діаграма діяльності (Activity Diagram) є одним з ключових інструментів методології UML (Unified Modeling Language), який допомагає моделювати послідовності дій або процеси в системі.

Цей графічний засіб відображає стан та послідовність різних дій або діяльностей, що відбуваються в системі або у конкретному процесі. Діаграма діяльності використовується для візуалізації послідовностей кроків або дій, що відбуваються в процесі.

Вона дозволяє розробникам програмного забезпечення аналізувати та моделювати процеси, виявляти потенційні проблеми та протиріччя, а також оптимізувати послідовність дій для досягнення більш ефективного виконання процесу.

На діаграмі діяльності дії або діяльності зображуються у вигляді прямокутників, а стрілки показують послідовність виконання цих дій. Різні типи стрілок та символи використовуються для вказівки різних видів рішень, умов, розгалужень, злиття та інших аспектів процесу. Одна з переваг використання діаграм діяльності полягає у зрозумілості та простоті представлення процесу.

Графічне зображення діаграми діяльності дозволяє всім зацікавленим сторонам легко розуміти послідовність дій та взаємозв'язки між ними. Вона також допомагає виявляти можливі помилки або недоліки в процесі та забезпечує загальне розуміння проєкту.

Діаграма діяльності також дозволяє виявити та оптимізувати зайві або непотрібні кроки в процесі. Це допомагає покращити ефективність та продуктивність процесу, зменшити його тривалість та витрати ресурсів.

Окрім того, діаграма діяльності дозволяє враховувати паралельні та

альтернативні шляхи в процесі. Вона допомагає виявити різні можливості та сценарії, які можуть виникнути під час виконання дій, і дозволяє встановити правильні пріоритети та послідовність кроків.

Діаграма діяльності є також зручним засобом комунікації між розробниками, замовниками та іншими зацікавленими сторонами. Вона допомагає уточнити та узгодити розуміння процесу, виявити можливі незгоди або недорозуміння та забезпечує однозначність та спільне бачення проєкту.

Наведемо діаграму діяльності, що описує модель поведінки варіанта використання «Замовити переклад». Діаграма представлена на рисунках 2.4 – 2.5.

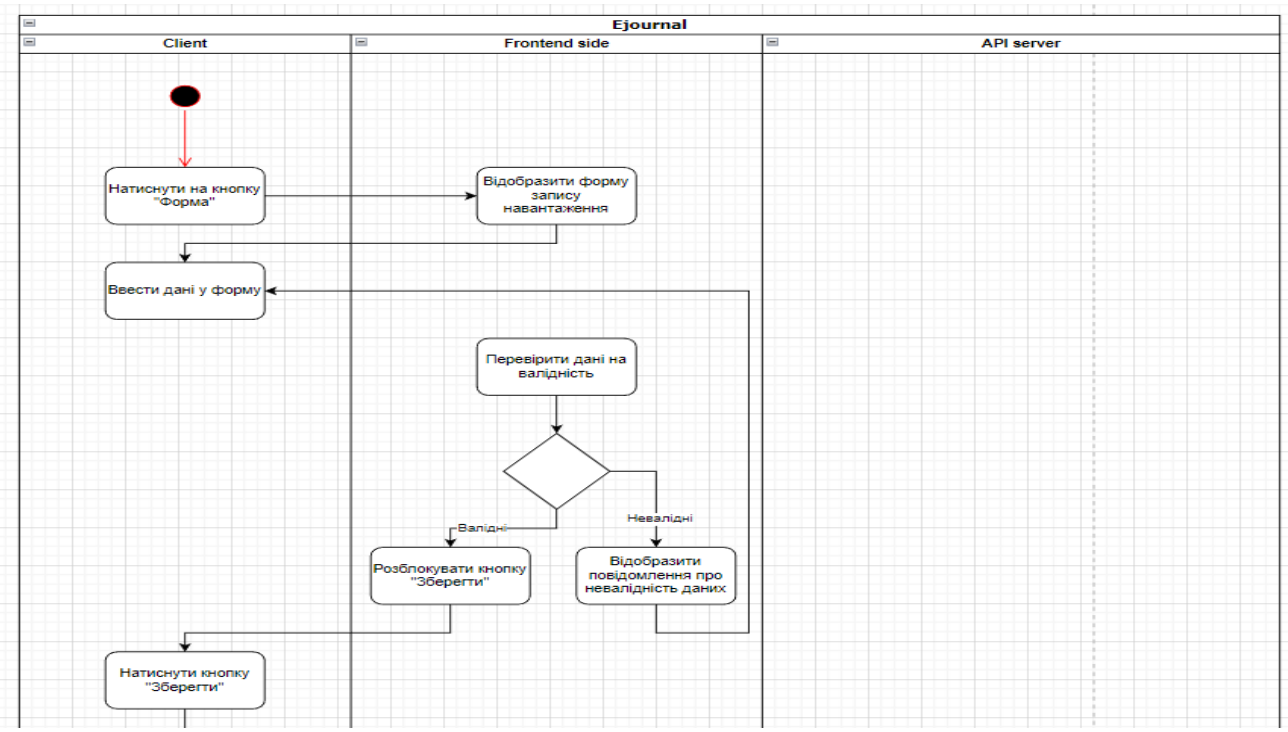

Рисунок 2.4 – Діаграма діяльності

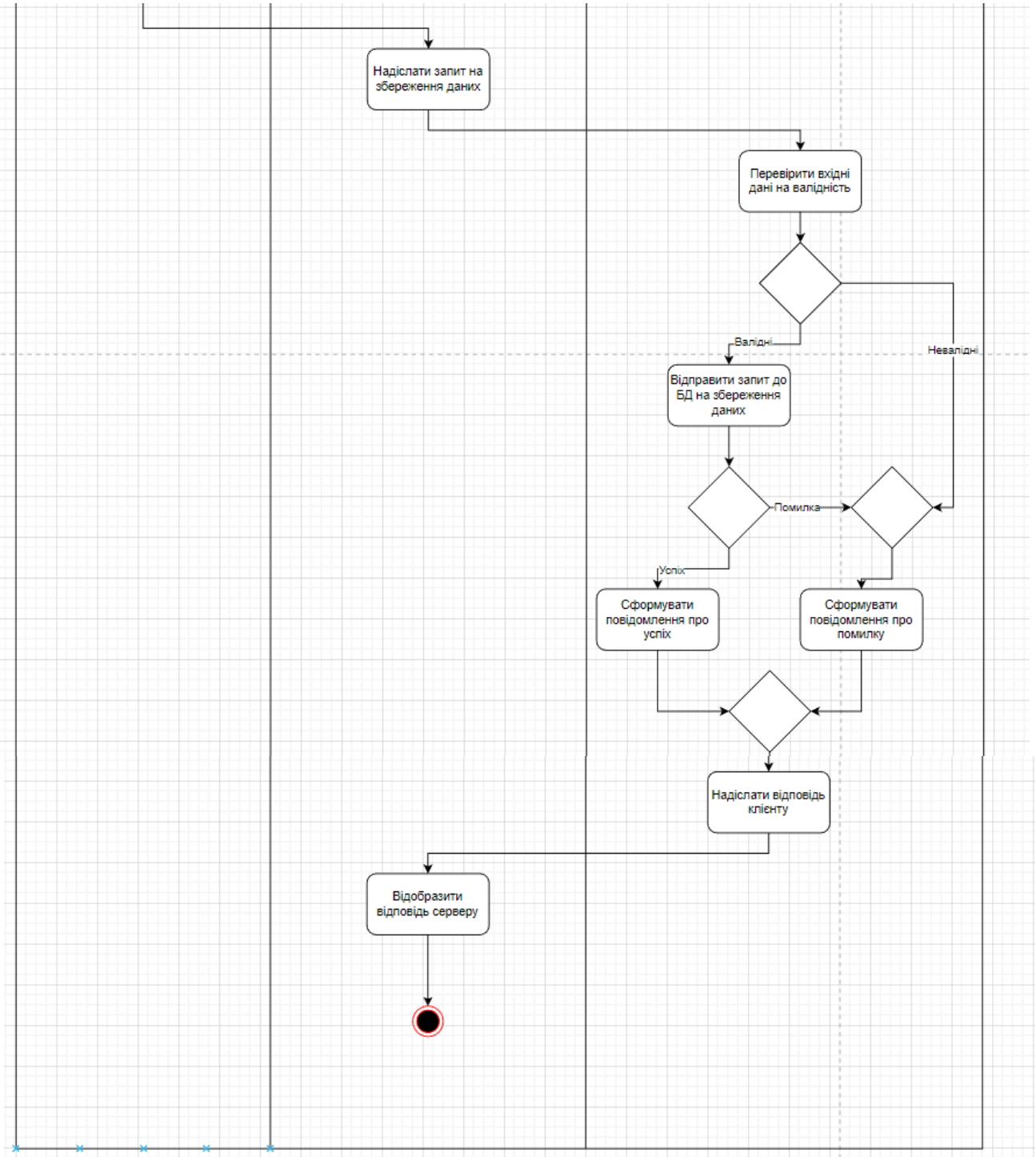

Рисунок 2.5 – Діаграма діяльності

2.4 Діаграма послідовності

Діаграма послідовності (Sequence Diagram) є одним з найпоширеніших та корисних інструментів методології UML (Unified Modeling Language), який

допомагає моделювати та візуалізувати взаємодію об'єктів або компонентів в системі. Цей графічний засіб відображає послідовність повідомлень, що передаються між об'єктами протягом певного часу, відображаючи тим самим виконання процесу або сценарію.

Діаграма послідовності є потужним інструментом для аналізу та моделювання взаємодії об'єктів у системі. Вона дозволяє розробникам програмного забезпечення визначати та уточнювати поведінку системи, виявляти взаємозв'язки між об'єктами, визначати порядок виконання дій та виявляти потенційні проблеми або недоліки в системі.

На діаграмі послідовності об'єкти зображуються у вигляді прямокутників, а повідомлення, що передаються між об'єктами, показуються у вигляді стрілок. Крім того, на діаграмі можуть бути вказані умови, цикли, розгалуження та інші структурні елементи, що визначають послідовність виконання повідомлень.

Одна з переваг використання діаграм послідовності полягає у зрозумілості та простоті моделювання процесу. Графічне зображення діаграми послідовності дозволяє всім зацікавленим сторонам легко розуміти взаємодію об'єктів та порядок виконання дій. Вона сприяє чіткому розумінню взаємозв'язків та логіки виконання процесу, що є важливим для команди розробників, замовників та інших зацікавлених сторін.

Однією з ключових особливостей діаграми послідовності є показ взаємодії між об'єктами у конкретний момент часу. Вона дозволяє показати, як об'єкти взаємодіють один з одним шляхом відправлення повідомлень та отримання відповідей. Це допомагає уточнити та зрозуміти виконання процесу або сценарію, ідентифікувати можливі проблеми зі зв'язками між об'єктами, а також покращити взаємодію і оптимізувати систему.

Діаграма послідовності може бути використана на різних етапах розробки програмного забезпечення. Вона допомагає при аналізі та проєктуванні системи, уточнює специфікацію вимог, визначає поведінку системи та забезпечує взаєморозуміння між розробниками та замовниками.

Окрім того, діаграма послідовності може бути використана для

моделювання тестових сценаріїв. Вона дозволяє визначити послідовність дій та взаємодію об'єктів для перевірки коректності роботи системи та виявлення можливих помилок.

На рисунку 2.6 описана діаграма послідовності прецедента «Додати запис про навантаження».

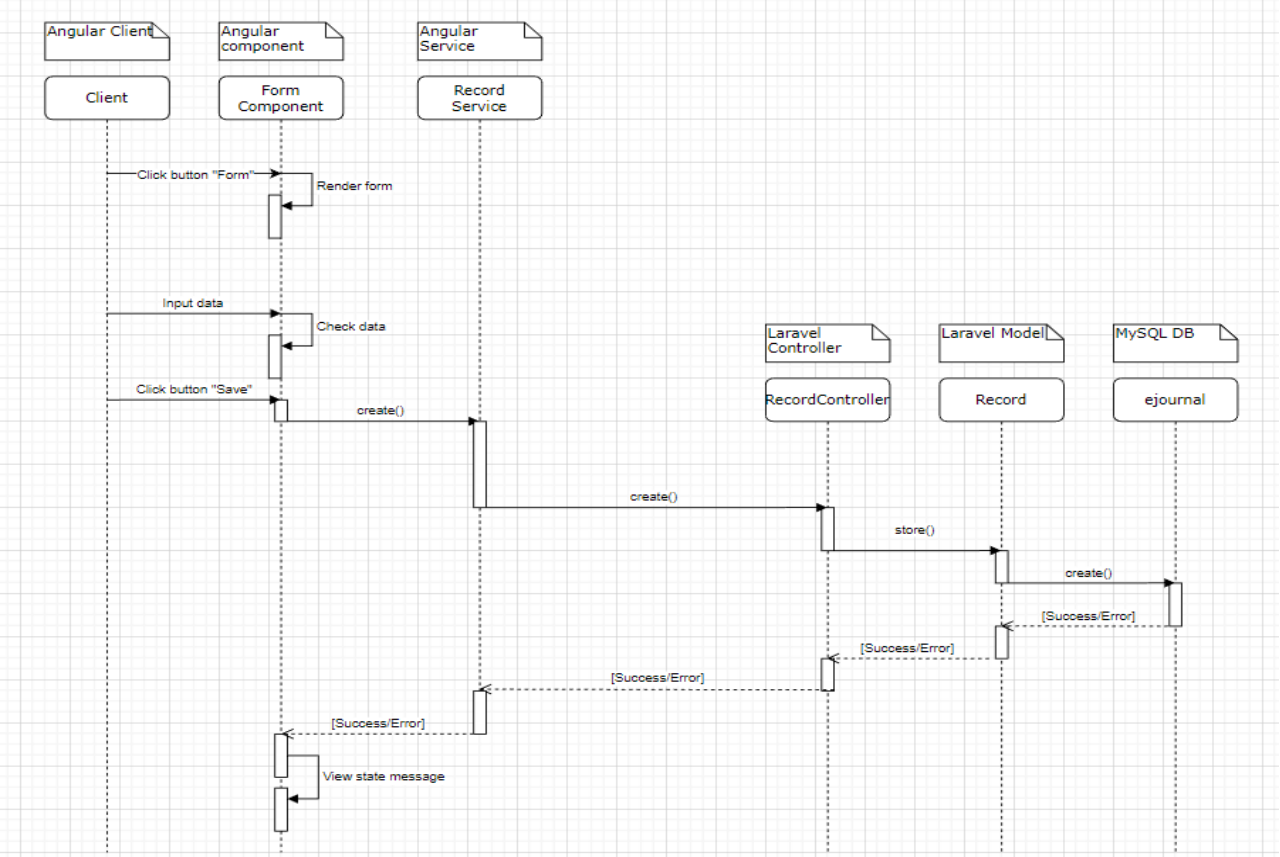

Рисунок 2.6 – Діаграма послідовності

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Для реалізації були використані php-фреймворк Laravel та front-end фреймворк Angular.

RxJS – це бібліотека, що реалізує принципи реактивного програмування JavaScript. Заснована на об'єктах типу Observable, вона спрощує написання та контроль асинхронного та подійного коду.

Angular Material – це бібліотека, що включає 30 сучасних компонентів, виготовлених згідно специфікації Google Material Design.

3.2 Процес створення Angular-компонента

Є два шляхи створення компоненту в Angular:

- консольна команда;
- вручну.

Процес створення компонента за допомогою консольної команди:

- відкрити консоль (термінал) у корні проєкту;
- ввести команду «ng g c %path_to_component/%%component_name% --skip-tests».

Розглянемо аргументи команди:

- g – скорочення від generate, генерує та/або змінює файли на основі схеми;
- c – скорочення від component, схема для створення;
- %path_to_component/% – шлях, по якому буде згенеровано компонент, якщо відсутній, то створюється у директорії src/app;
- %component_name% – назва компоненту;

– skip-tests – опціональний аргумент, не створює тестовий файл для компоненту.

Після виконання команди, система створить новий компонент з базовими файлами шаблону, стилів та оброблювача. Також система зареєструє новий компонент у масиві declarations, файла app.module.ts [1 - 3].

Процес створення компонента вручну:

– відкрити проєкт у зручному редакторі коду (у мене WebStorm);
– створити теку з ім’ям компоненту в теці app, це буде контейнер для файлів компонента;
– створюємо у теці компоненту файли app.%component_name%.html|.scss|.ts;
– у файлі app.%component_name%.ts визначаємо декоратор @Component, у якому підключаємо файли шаблону та стилів;
– експортуємо класс компонента, для можливості його використання у інших частинах проєкту;
– імпортуємо компонент у файл app.module.ts та реєструємо його у масиві declarations.

Візуалізація процесу та приклад коду компонента наведено у Додатку А.

3.3 Процес створення Angular-сервісу

Є два шляхи створення сервісу в Angular:

– консольна команда;
– вручну.

Процес створення сервісу за допомогою консольної команди:

– відкрити консоль (термінал) у корні проєкту;
– ввести команду «ng g s %path_to_service/%%service_name% --skip-tests».

Розглянемо аргументи команди:

- s – скорочення від service, схема для створення;
- %path_to_service/% – шлях, по якому буде згенеровано сервіс, якщо відсутній, то створюється у директорії src/app;
- %service_name% – назва сервісу;
- skip-tests – опціональний аргумент, не створює тестовий файл для сервісу.

Після виконання команди, система створить новий сервіс [1 - 3].

Процес створення сервісу вручну:

- відкрити проєкт у зручному редакторі коду (у мене WebStorm);
- створити теку з назвою «services» в теці app, це буде контейнер для файлів сервісів;
- створюємо у теці «services» файл app.%service_name%.ts;
- у файлі app.%service_name%.ts визначаємо декоратор @Injectable;
- експортуємо класс сервісу, для можливості його використання у інших частинах проєкту.

Візуалізація процесу та приклад коду сервісу наведено у Додатку Б.

3.4 Процес створення Laravel-моделі

Є два шляхи створення моделі в Laravel:

- консольна команда;
- вручну.

Процес створення моделі за допомогою консольної команди:

- відкрити консоль (термінал) у корні проєкту;
- ввести команду «php artisan make:model %path_to_model/%%model_name% [params]».

Розглянемо аргументи команди:

- make:model – створює новий клас Eloquent моделі;
- %path_to_model/% – шлях, по якому буде згенеровано модель, якщо

відсутній, то створюється у директорії app/Models;

- %model_name% – назва моделі;
- params – можно вказати додаткові аргументи для автоматичного створення контролера, міграції, тощо.

Після виконання команди, система створить нову модель [8, 10].

Процес створення моделі вручну:

- відкрити проєкт у зручному редакторі коду (у мене PhpStorm);
- створюємо у теці «app/Models» клас %model_name%.php;
- у файлі %model_name%.php вказуємо, що вона буде розширяти базову модель «Illuminate\Database\Eloquent\Model».

Візуалізація процесу та приклад коду моделі наведено у Додатку В.

3.5 Процес створення Laravel-контролеру

Є два шляхи створення контролеру в Laravel:

- консольна команда;
- вручну.

Процес створення контролеру за допомогою консольної команди:

- відкрити консоль (термінал) у корні проєкту;
- ввести команду «php artisan make:controller %path_to_controller/%%controller_name% [params]».

Розглянемо аргументи команди:

- make:controller – створює новий клас контролера;
- %path_to_controller/% – шлях, по якому буде згенеровано контроллер, якщо відсутній, то створюється у директорії app/Http/Controllers;
- %controller_name% – назва контролеру;
- params – можно вказати додаткові аргументи для автоматичного ресурсів та прив’язки до моделі.

Після виконання команди, система створить новий контролер [8, 10].

Процес створення контролеру вручну:

– відкрити проєкт у зручному редакторі коду (у мене PhpStorm);
– створюємо у теці «app/Http/Controllers» клас %controller_name%.php;
– у файлі %controller_name%.php вказуємо, що він буде розширяти базовий контролер «App\Http\Controllers\Controller».

Візуалізація процесу та приклад коду моделі наведено у Додатку Г.

3.6 Основні класи системи

Так як основним прецедентом системи є «Керування навантаженням», то основними класами системи будуть ті, що надають можливість створювати, змінювати, видаляти та переглядати дані.

Головними класами зі сторони Angular є:

– RecordService – відповідає за дії пов’язані з Laravel API-сервером, а саме містить запити на створення, видалення, зміну та відображення даних про навчальне навантаження;
– RecordComponent – обробляє та відображає дані, отримані від методів RecordService (містить методи для управління відображенням прив’язаного до компоненту шаблона, отримує дані від шаблону та передає їх у сервіс (наприклад видалення)).

Головними класами зі сторони Laravel є:

– Record – модель, що пов’язана з таблицею «records» (містить список полів для запису до таблиці, їх типи, зв’язки з іншими таблицями);
– RecordController – групує логіку обробки запитів пов’язаних з моделлю «Record».

Нижче приведено приклади функцій контролеру та сервісу (див. рис. 3.1).

```
        getByUserId(id: number): Observable<any> {
          return this.http.get(
            this.apiUrl + '/record/' + id,
            {headers: this.headers}
          );
        }

        public function getByUserId($id): JsonResponse
        {
        $reports = Report::with(['faculties', 'departments', 'users', 'disciplines.semesters', 'topics', 'disciplineKinds', 'courses', 'forms'])
        ->select('reports.*', DB::raw('group_concat(g.name) as group_name'))
        ->leftJoin('groups as g', DB::raw('FIND_IN_SET(g.id, reports.group)'),'>',DB::raw('"0"'))
        ->where('user_id', $id)->groupBy('reports.id')->get();
      return response()->json(['records' => $reports, 'message' => 'Success'], 200);
        }
```

Рисунок 3.1 – Приклади функцій контролеру та сервісу

3.7 Тестування проєкту

3.7.1 Unit-тест

Модульне тестування, або юніт-тестування – процес у програмуванні, що дозволяє перевірити на коректність окремі модулі вихідного коду програми.

Ідея полягає в тому, щоб писати тести для кожної нетривіальної функції чи методу. Це дозволяє досить швидко перевірити, чи не привела чергова зміна коду до регресії, тобто до появи помилок у вже відтестованих місцях програми, а також полегшує виявлення та усунення таких помилок [5]. Приклади такого тестування наведено на рисунках 3.2–3.3.

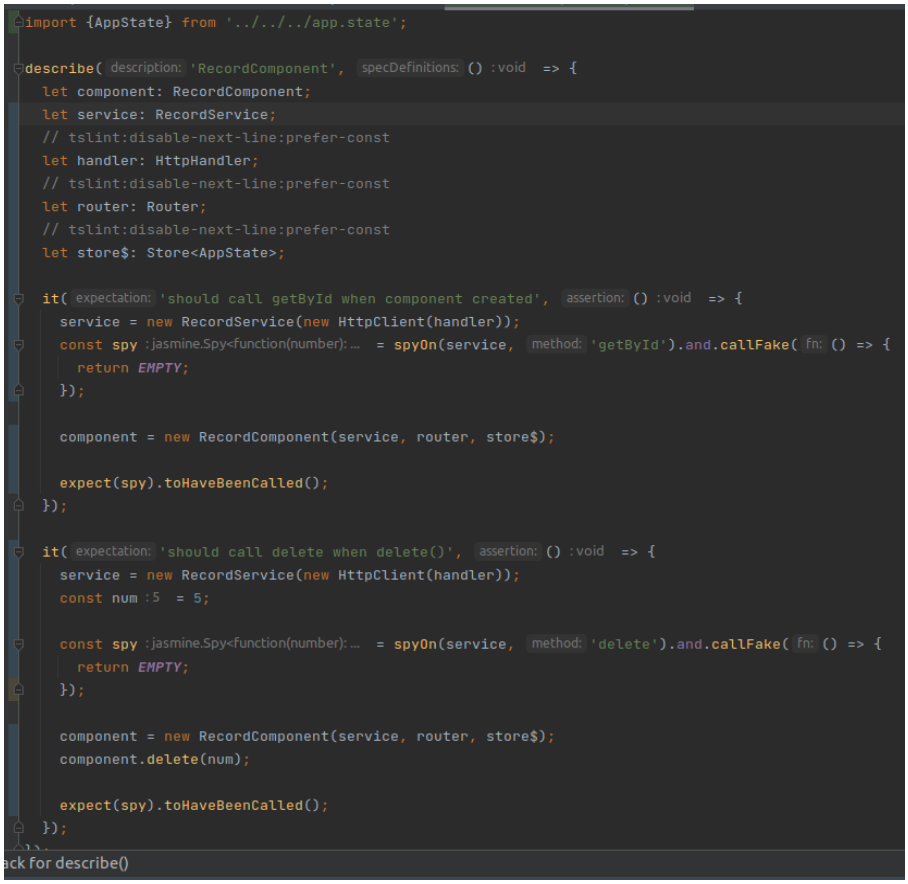

Рисунок 3.2 – Тестування RecordComponent

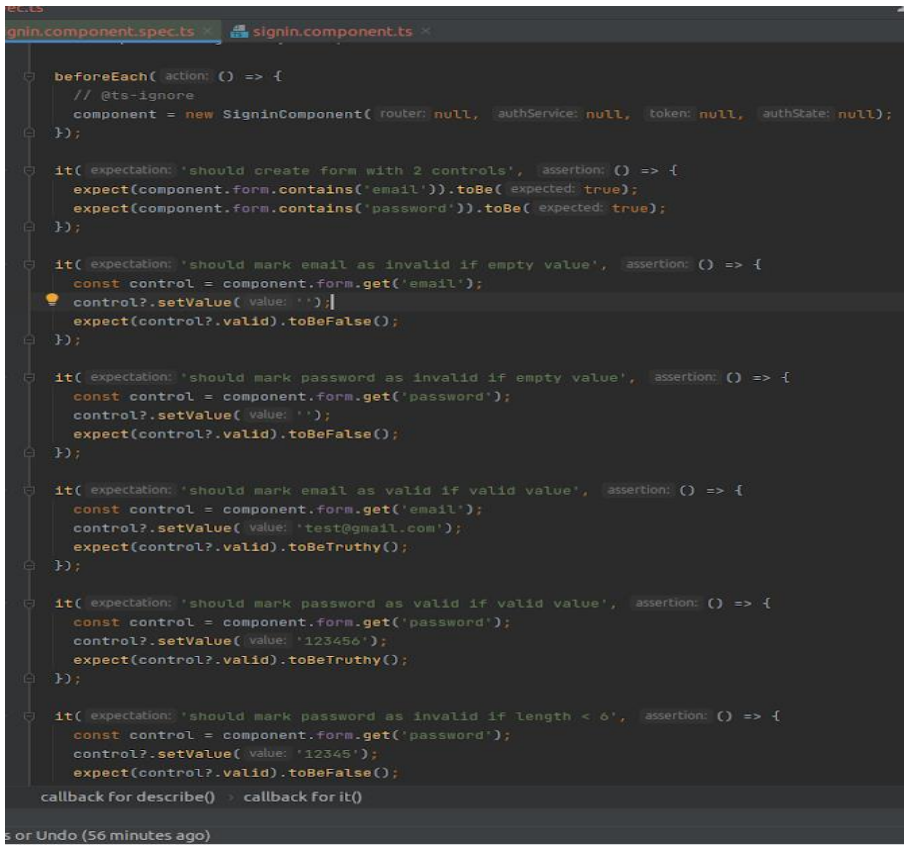

Рисунок 3.3 – Тестування SigninComponent

3.7.2 Integration-тест

Інтеграційне тестування проводиться для тестування модулів/компонентів, коли вони інтегровані, щоб перевірити, чи працюють вони належним чином [4, 6].

На рисунку 3.4 зображено тестування на створення сервісу.

```
import { TestBed } from '@angular/core/testing';

import { RecordService } from './record.service';
import {HttpClientModule} from '@angular/common/http';

describe( description: 'RecordService', specDefinitions: () :void => {
  let service: RecordService;

  beforeEach( action: () :void => {
    TestBed.configureTestingModule( moduleDef: {
      imports: [
        HttpClientModule
      ]
    });
    service = TestBed.inject(RecordService);
  });

  it( expectation: 'should be created', assertion: () :void => {
    expect(service).toBeTruthy();
  });
});
```

Рисунок 3.4 – Тестування створення сервісу

3.8 Керівництво користувача

3.8.1 Рівень підготовки користувача

Користувач сайту повинен володіти певною кваліфікацією.

Навички користувача для роботи з ПК, та навички роботи з web-браузером.

Знайомство з Керівництвом користувача.

3.8.2 Підготовка до роботи

Запуск системи.

Доступ до сайту здійснюється через мережу Інтернет за допомогою звичайного web-браузера. Адреса сайту в мережі Інтернет: https://ejournal.loc. Для коректної роботи клієнтської частини повинен використовуватися браузер Google Chrome, Mozilla Firefox, Opera, Safari.

При вході на Сайт користувач потрапляє на сторінку входу до системи.

На Сайті розрізняються наступні групи користувачів:

Анонімний користувач (не увійшли або не зареєстровані, мають доступ до сторінок входу).

Користувач – приватна особа, авторизований на сайті (далі Користувач), що має доступ до функцій користувача у особистому кабінеті.

Адміністратор системи (далі Адміністратор) відповідає за підтримку та конфігурування системи.

3.8.3 Вхід до системи

При вході на сайт, система відображає сторінку входу до системи (рис. 3.5).

Рисунок 3.5 – Форма входу до системи

Якщо Ви зареєстровані, то потрібно ввести логін та пароль у відповідні поля та натиснути кнопку «Вхід».

Під час заповнення форми, система автоматично перевіряє валідність, якщо дані валідні, то поля форми будуть мати сірий контур (рис. 3.6), у противному випадку – червоний (рис. 3.7).

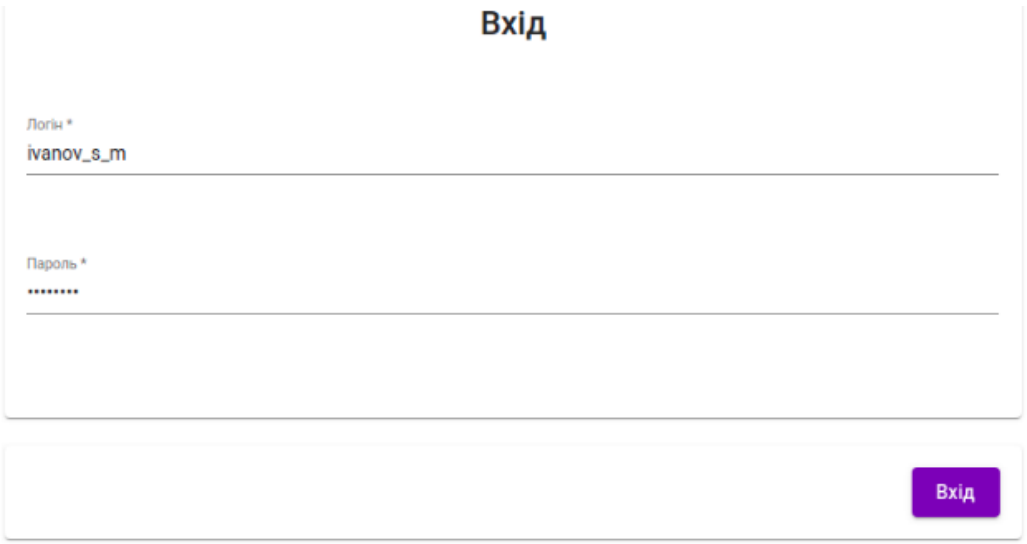

Рисунок 3.6 – Валідні дані

Рисунок 3.7 – Невалідні дані

Якщо користувач з таким логін та/або паролем не існує [7], то система сповістить про це відповідним повідомленням (рис. 3.8).

Рисунок 3.8 – Невірний логін та/або пароль

Після авторизації, система відобразить сторінку з записами навчального навантаження користувача (рис. 3.9).

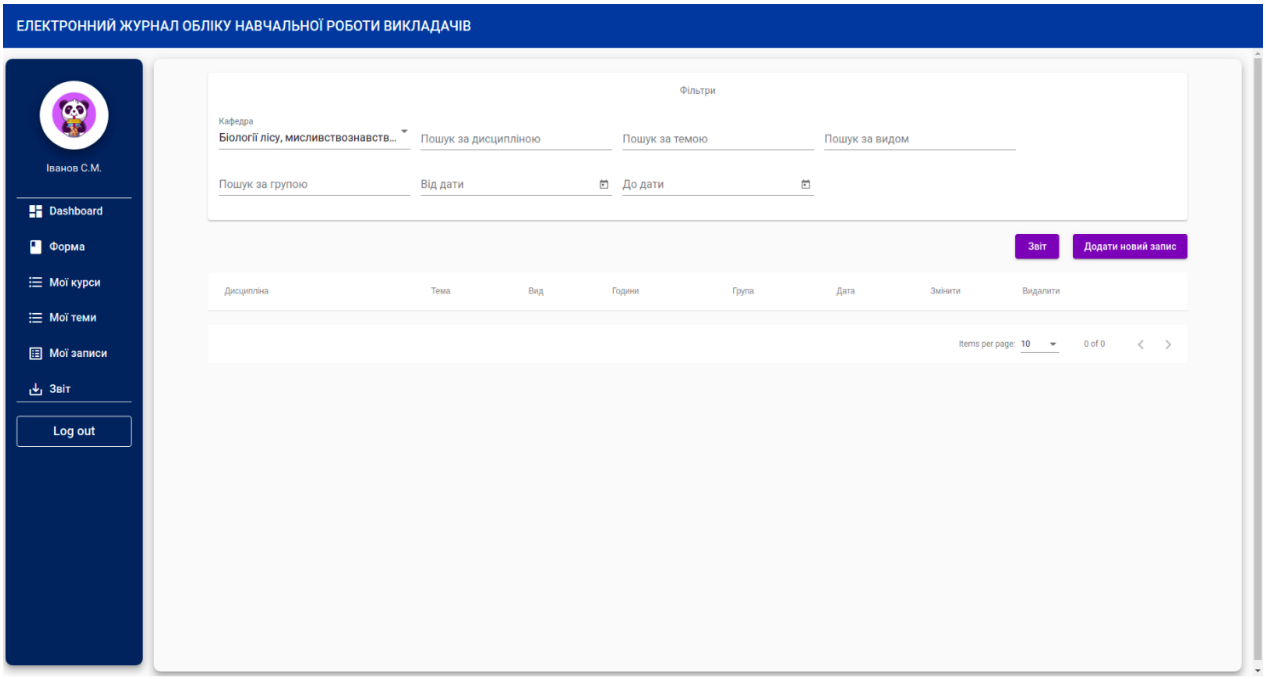

Рисунок 3.9 – Записи навчального навантаження

3.8.4 Взаємодія зі списком дисциплін

Для введення навантаження, потрібно створити дисципліни, на основі яких воно буде формуватися. Переходимо до панелі керування дисциплінами,

обравши пункт «Мої дисципліни» на панелі керування. Інтерфейс керування дисциплінами є інтуїтивно зрозумілим. Для створення дисципліни потрібно натиснути на кнопку «Додати дисципліну» та у модальній формі, що відобразилась ввести назву, кафедру та семестр (рис. 3.10 – 3.12).

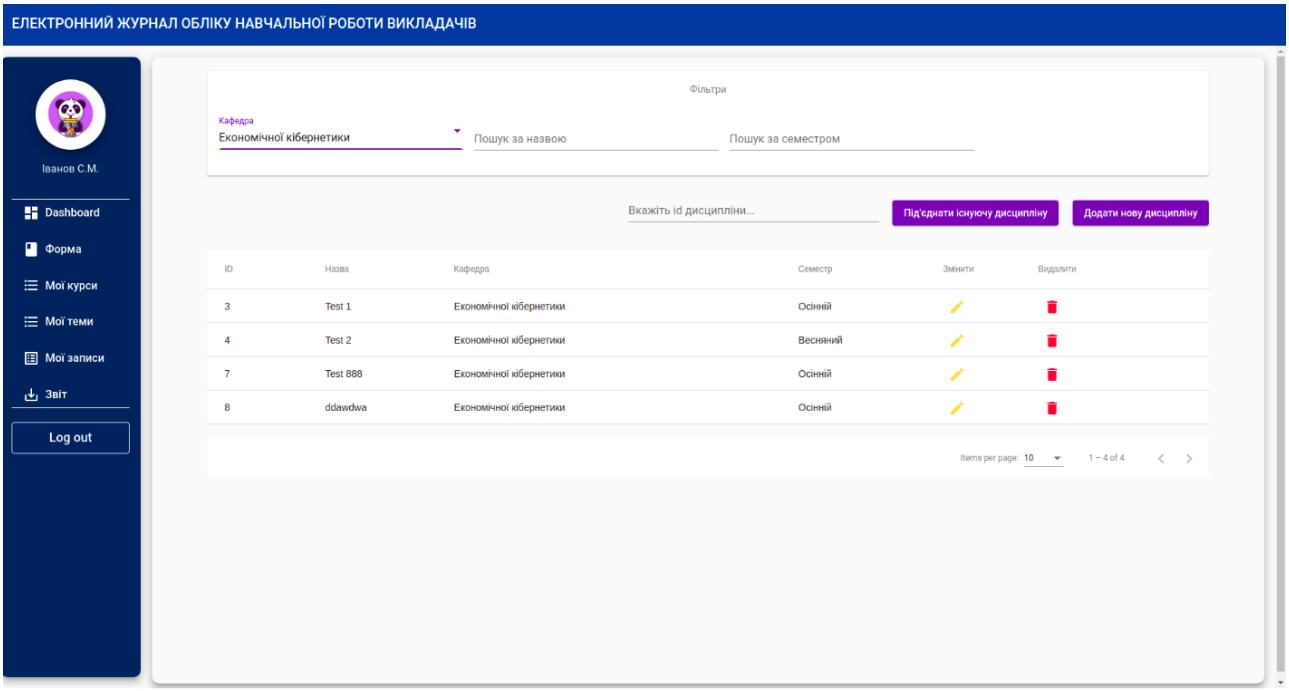

Рисунок 3.10 – Панель керування дисциплінами

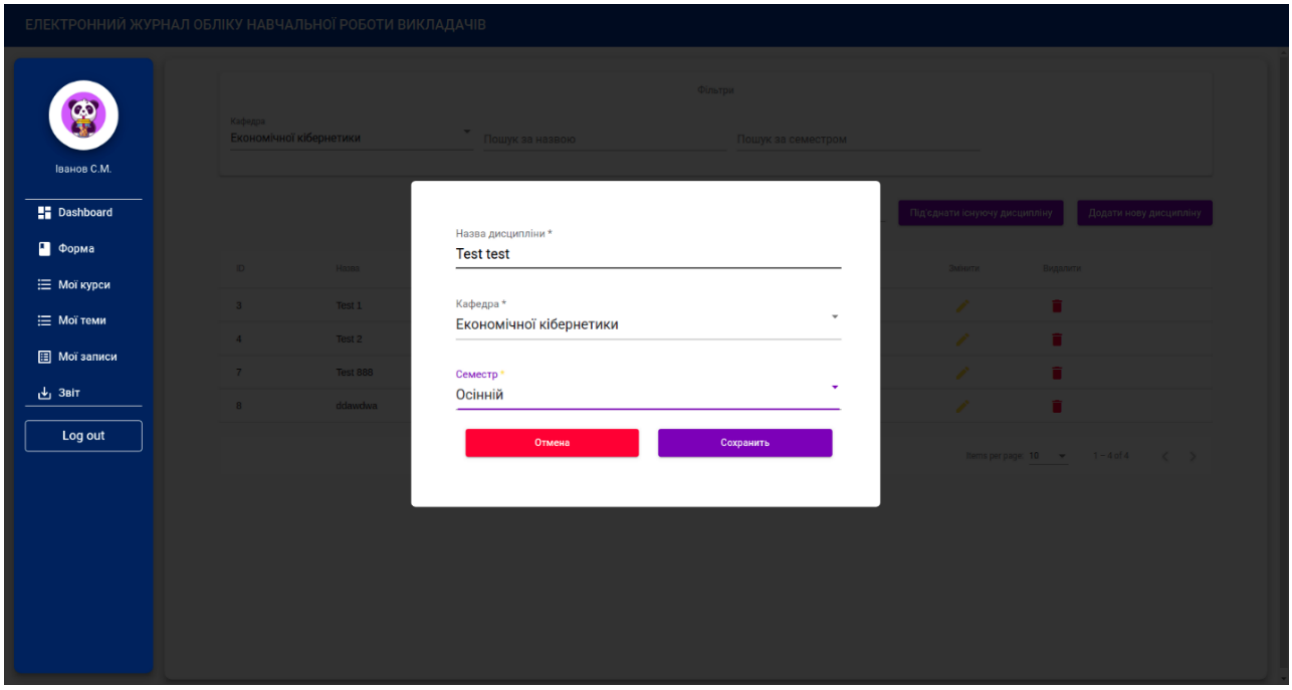

Рисунок 3.11 – Форма створення нової дисципліни

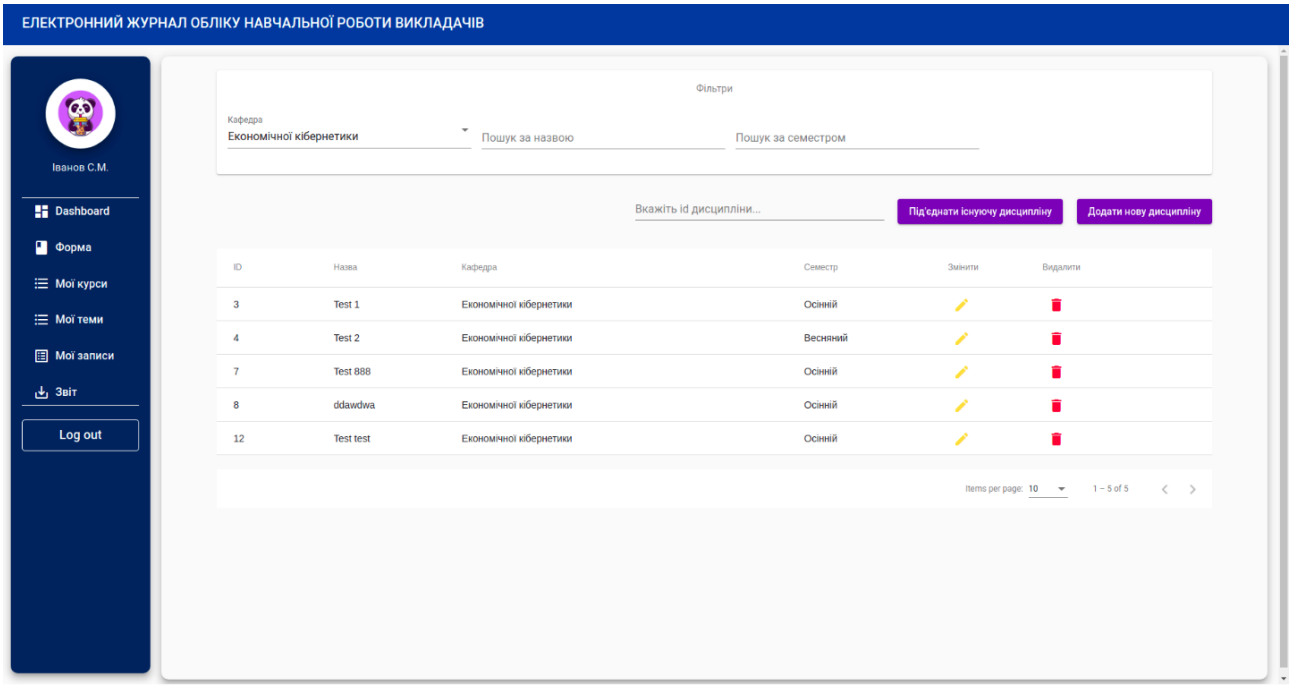

Рисунок 3.12 – Відображення нової дисципліни

Якщо дисципліна створена з помилками, потрібно натиснути на піктограму олівця та у відкритій формі внести зміни. Якщо дисципліна більше непотрібна, то її можна видалити, натиснувши на піктограму смітника.

3.8.5 Взаємодія зі списком тем

Після створення дисципліни, необхідно створити теми. Для цього потрібно обрати пункт «Мої теми» на панелі навігації. Взаємодія з темами збігається з взаємодією з дисциплінами, тому на рисунках 3.13 – 3.15 наведено покроковий алгоритм створення нової теми.

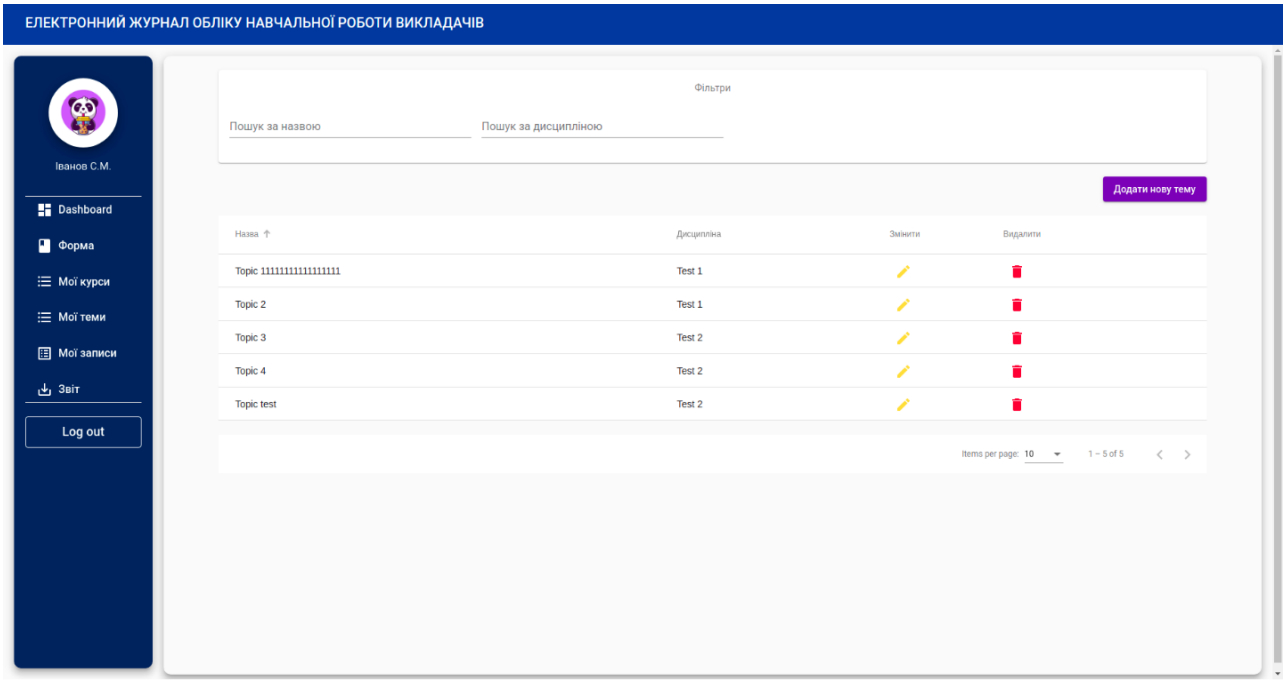

Рисунок 3.13 – Панель керування темами

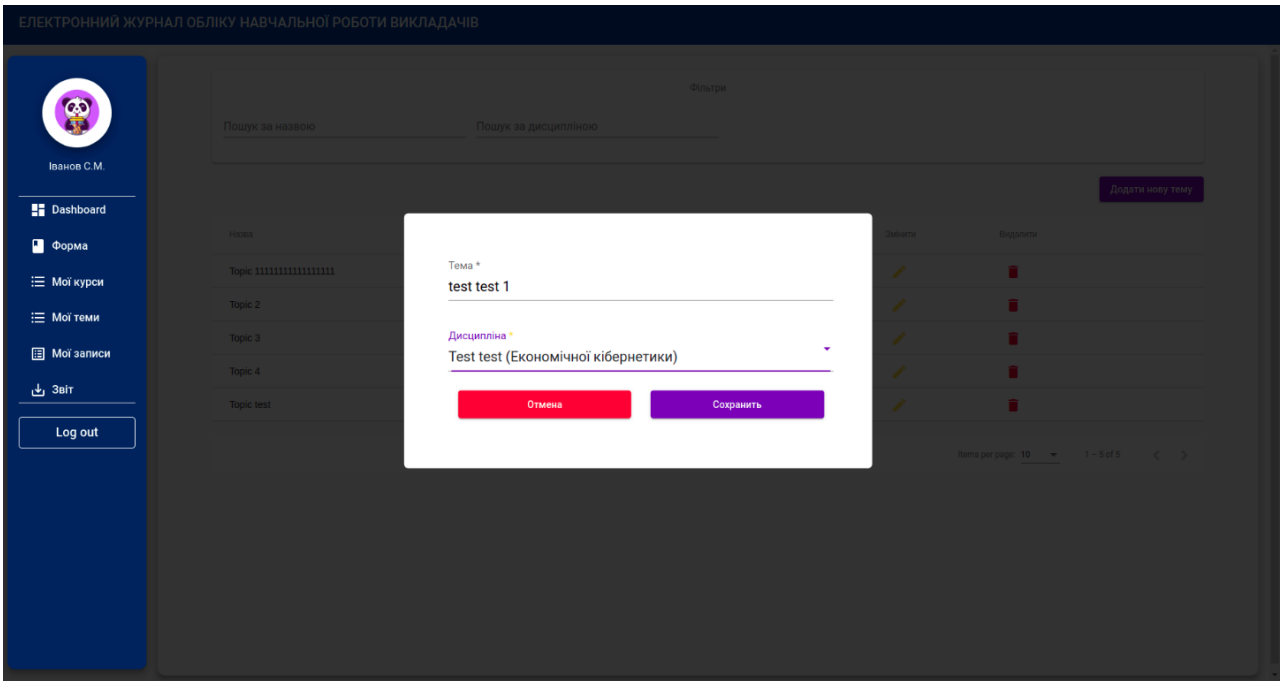

Рисунок 3.14 – Форма створення нової теми

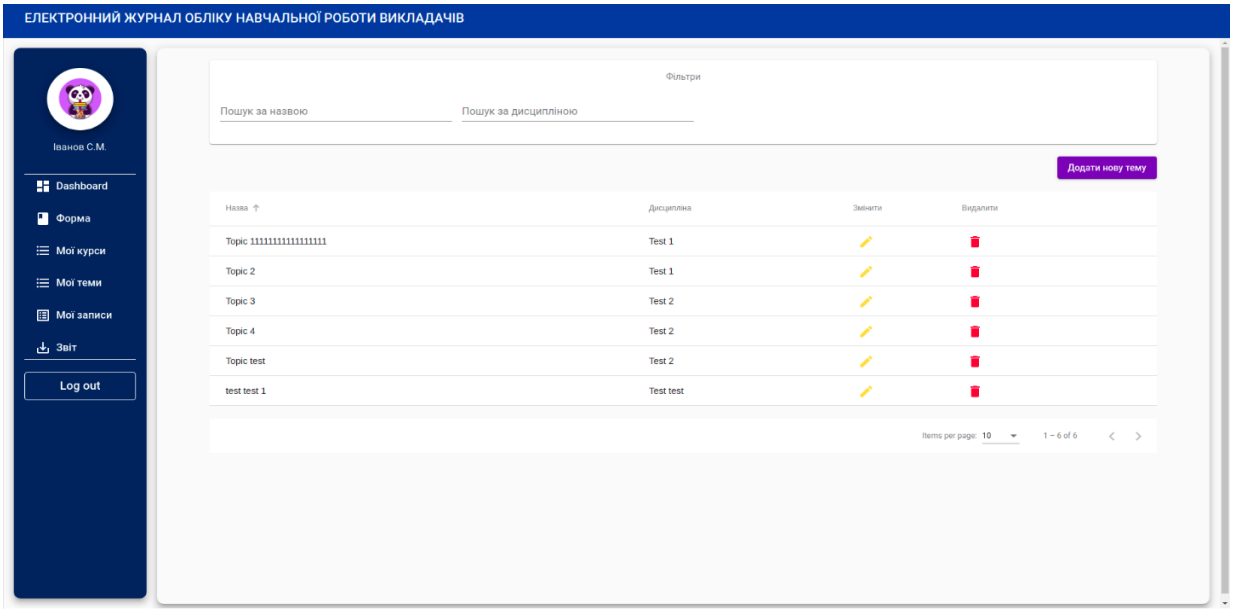

Рисунок 3.15 – Відображення нової теми

3.8.6 Взаємодія з навантаженням

Так як інтерфейс взаємодії з системою уніфікований, то взаємодія зі списком записів навчального навантаження користувача є подібною до взаємодії з дисциплінами та темами (рис. 3.16). Основною відмінністю є форма створення (рис. 3.17) та редагування (рис. 3.18) записів.

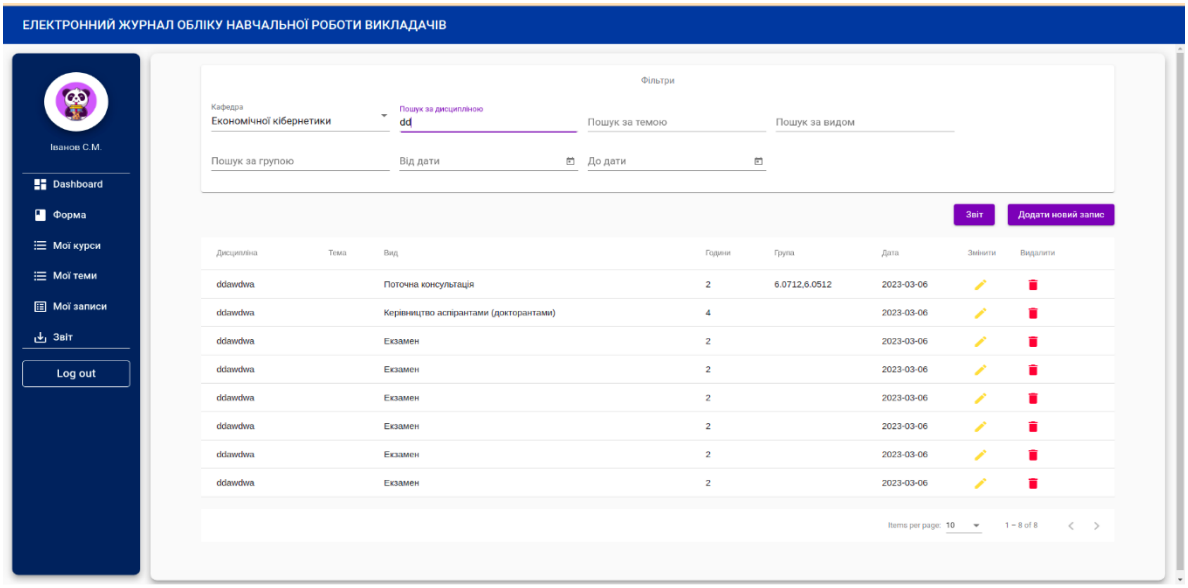

Рисунок 3.16 – Панель керування навантаженням

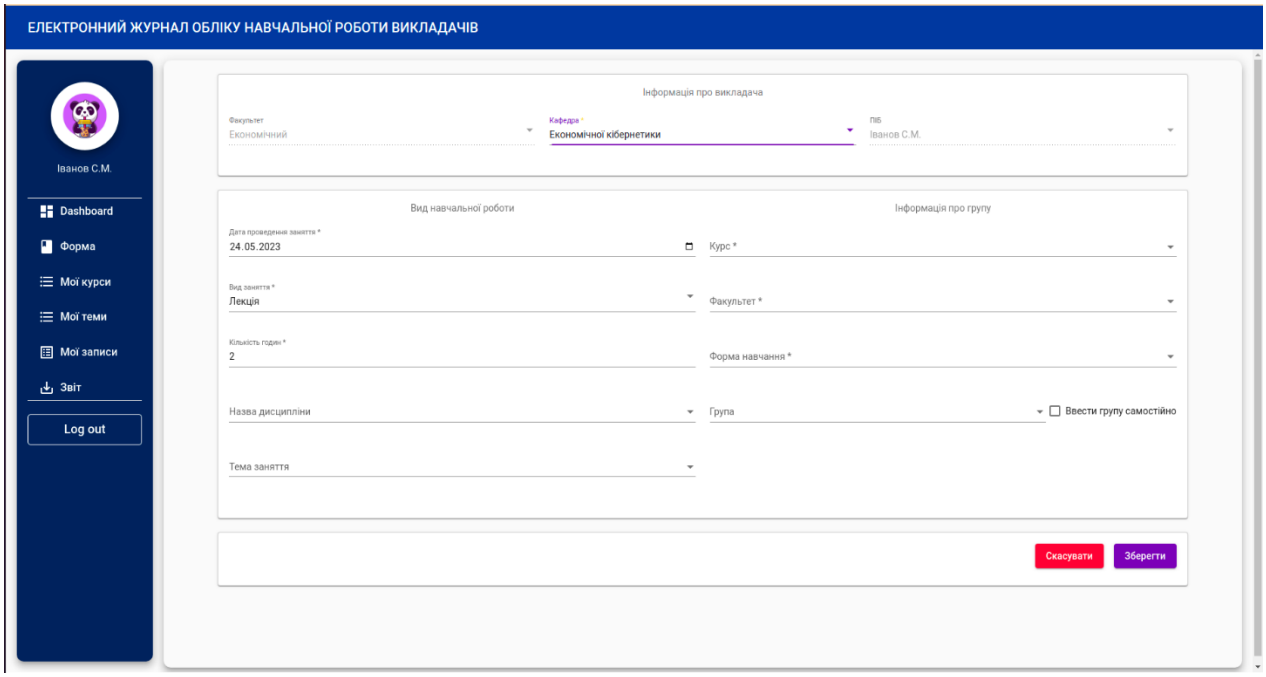

Рисунок 3.17 – Форма створення нового запису

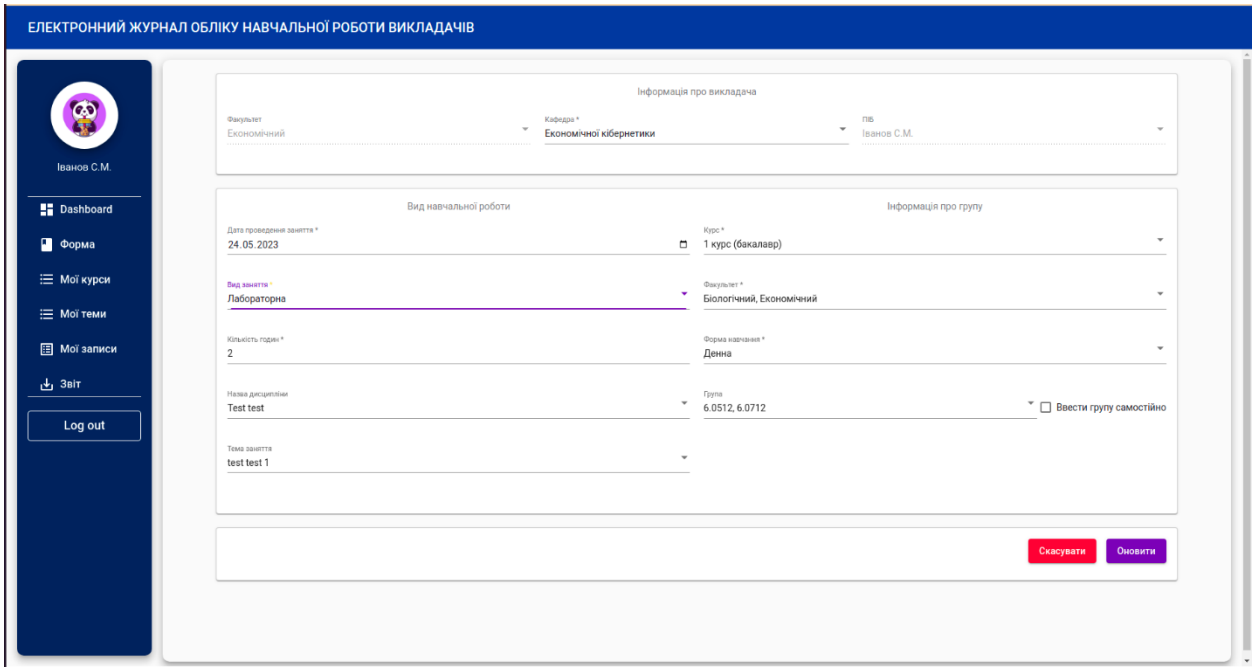

Рисунок 3.18 – Форма редагування запису

Форми створення і редагування записів навчального навантаження є динамічними, тобто видозмінюються залежно від даних, які вносить або корегує користувач.

ВИСНОВКИ

В результаті роботи було написано технічне завдання на розробку системи обліку навчального навантаження. Для створення цієї системи були обрані фреймворки Angular зі сторони клієнта та Laravel зі сторони сервера, за їх широкі можливості у сфері створення web-систем.

У відповідності з метою кваліфікаційної роботи була розроблена система обліку навчального навантаження із застосуванням наступних технологій:

- Laravel для реалізації back end API;
- Angular для реалізації front end частини, а також відправки запитів до серверу.

У відповідності з поставленими задачами були виконані наступні етапи створення системи:

- сформовані вимоги до системи (функціональні та нефункціональні (інтерфейс, кросбраузерність, безпека, продуктивність)) (також проведено огляд предметної області та інструментів розробки);
- спроєктована та побудована структура системи (побудовані діаграми прецедентів, діяльності та послідовності; надано детальний опис прецедентів);
- реалізовано систему обліку навчального навантаження (наведена інструкція по створенню компонентів системи, надано керівництво користувача та структура проєкту);
- протестована робота системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Angular Developer Documentation. URL: https://angular.io/docs/ (дата звернення: 01.03.2023).

2. Bampakos A. Angular Projects: Build modern web apps by exploring Angular 12 with 10 different projects and cutting-edge technologies. Birmingham : Packt Publishing, 2021. 344 p.

3. Bampakos A., Deeleman P. Learning Angular: A no-nonsense guide to building web applications with Angular 15. Birmingham : Packt Publishing, 2023. 446 p.

4. Chebbi L. Reactive Patterns with RxJS for Angular: A practical guide to managing your Angular application's data reactively and efficiently using RxJS 7. Birmingham : Packt Publishing, 2022. 224 p.

5. Fain Y., Moiseev A. Angular Development with TypeScript. New York : Manning Publications, 2019. 560 p.

6. Freeman A. Pro Angular: Build Powerful and Dynamic Web Apps. London : Apress, 2022. 905 p.

7. JSON Web Token Authentication for Laravel & Lumen. URL: https://jwt-auth.readthedocs.io/en/develop/ (дата звернення: 03.04.2023).

8. Laravel 8 Developer Documentation. URL: https://laravel.com/docs/8.x/ (дата звернення: 05.04.2023).

9. MySQL Developer Documentation. URL: https://dev.mysql.com/doc/ (дата звернення: 11.03.2023).

10. Stauffer M. Laravel: Up & Running: A Framework for Building Modern PHP Apps : O’Reilly Media, 2019. 544 p.

ДОДАТОК А

Angular-компонент

```
import {AfterViewInit, Component, OnInit, ViewChild} from '@angular/core';
import {MatSort} from '@angular/material/sort';
import {MatPaginator} from '@angular/material/paginator';
import {MatTableDataSource} from '@angular/material/table';
import {Record} from '../../../core/models/record';
import {DatePipe} from '@angular/common';
import {FormControl} from '@angular/forms';
import {RecordService} from '../../../services/admin/record.service';
import {Unsubscribe} from '../../../helpers/unsubscribe';
import {Department} from '../../../core/models/department';
import {takeUntil} from 'rxjs/operators';
import {Router} from '@angular/router';
import {Store} from '@ngrx/store';
import {AppState} from '../../../app.state';
import {MatSnackBar} from '@angular/material/snack-bar';

@Component({
  selector: 'app-record',
  templateUrl: './record.component.html',
  styleUrls: ['./record.component.scss']
})
export class RecordComponent extends Unsubscribe implements OnInit, AfterViewInit {

  @ViewChild(MatSort) sort!: MatSort;
  @ViewChild(MatPaginator) paginator!: MatPaginator;

  public record = new MatTableDataSource<Record>();
  public displayedColumns = ['discipline', 'topic', 'kind', 'hour', 'group', 'date',  'update', 'delete'];
  pipe!: DatePipe;
  userId!: number;
  departments!: Department[];
  selected = 0;
  disciplineFilter = new FormControl('');
```

```
  topicFilter = new FormControl('');
  kindFilter = new FormControl('');
  groupFilter = new FormControl('');
  departmentFilter = new FormControl('');
  fromDate = new FormControl();
  toDate = new FormControl();

  filterValues = {
    discipline: '',
    topic: '',
    kind: '',
    group: '',
    department: '',
    fromDate: Date,
    toDate: Date,
  };

  constructor(
    public recordService$: RecordService,
    public router: Router,
    private store$: Store<AppState>,
    private message: MatSnackBar
  ) {
    super();
    this.pipe = new DatePipe('en');
    this.store$.select('userinfo').pipe(takeUntil(this.ngUnsubscribe)).subscribe((userdata) => {
      this.departments = userdata[0].departments;
      this.selected = this.departments[0].id;
      this.userId = userdata[0].id;
    });

    this.recordService$.getByUserId(this.userId).pipe(takeUntil(this.ngUnsubscribe)).subscribe((data: any) =>
{
      this.record.data = data.records.sort((a: any, b: any) => {
        return (new Date(b.date) as any) - (new Date(a.date) as any);
      }) as Record[];
    });
    this.record.filterPredicate = (data, filter) => {
```

```
    const searchTerms = JSON.parse(filter);
    if (searchTerms.fromDate && searchTerms.toDate) {
      return
data?.disciplines?.name.trim().toLowerCase().indexOf(searchTerms.discipline.trim().toLowerCase()) !== -1
        &&
data?.topics?.name.toString().trim().toLowerCase().indexOf(searchTerms.topic.trim().toLowerCase()) !== -1
        &&
data?.discipline_kinds?.name.toString().trim().toLowerCase().indexOf(searchTerms.kind.trim().toLowerCas
e()) !== -1
        &&
data?.group_name?.toString().trim().toLowerCase().indexOf(searchTerms.group.trim().toLowerCase()) !== -
1
        && data?.user_department_id === searchTerms.department
        && new Date(new Date(data.date).setHours(0, 0, 0, 0)) >= new Date(searchTerms.fromDate)
        && new Date(new Date(data.date).setHours(0, 0, 0, 0)) <= new Date(searchTerms.toDate);
    } else {
      return
data?.disciplines?.name.trim().toLowerCase().indexOf(searchTerms.discipline.trim().toLowerCase()) !== -1
        &&
data?.topics?.name.toString().trim().toLowerCase().indexOf(searchTerms.topic.trim().toLowerCase()) !== -1
        &&
data?.discipline_kinds?.name.toString().trim().toLowerCase().indexOf(searchTerms.kind.trim().toLowerCas
e()) !== -1
        &&
data?.group_name?.toString().trim().toLowerCase().indexOf(searchTerms.group.trim().toLowerCase()) !== -
1
        && data?.user_department_id === searchTerms.department;
    }
  };
 }

 ngOnInit(): void {
  this.disciplineFilter.valueChanges.pipe(takeUntil(this.ngUnsubscribe)).subscribe(
    discipline => {
      this.filterValues.discipline = discipline;
      this.record.filter = JSON.stringify(this.filterValues);
    });
  this.topicFilter.valueChanges.pipe(takeUntil(this.ngUnsubscribe)).subscribe(
```

```
    topic => {
      this.filterValues.topic = topic;
      this.record.filter = JSON.stringify(this.filterValues);
    }
  );
  this.kindFilter.valueChanges.pipe(takeUntil(this.ngUnsubscribe)).subscribe(
    kind => {
      this.filterValues.kind = kind;
      this.record.filter = JSON.stringify(this.filterValues);
    }
  );
  this.groupFilter.valueChanges.pipe(takeUntil(this.ngUnsubscribe)).subscribe(
    group => {
      this.filterValues.group = group;
      this.record.filter = JSON.stringify(this.filterValues);
    }
  );
  this.departmentFilter.valueChanges.pipe(takeUntil(this.ngUnsubscribe)).subscribe(
    department => {
      this.filterValues.department = department;
      this.record.filter = JSON.stringify(this.filterValues);
    }
  );
  this.fromDate.valueChanges.pipe(takeUntil(this.ngUnsubscribe)).subscribe(
    from => {
      this.filterValues.fromDate = from;
      this.record.filter = JSON.stringify(this.filterValues);
    }
  );
  this.toDate.valueChanges.pipe(takeUntil(this.ngUnsubscribe)).subscribe(
    to => {
      this.filterValues.toDate = to;
      this.record.filter = JSON.stringify(this.filterValues);
    }
  );
}

ngAfterViewInit(): void {
```

```
  this.record.sort = this.sort;
  this.record.paginator = this.paginator;
}

delete(id: number): void {
  this.recordService$.delete(id).subscribe(
    result => {
      this.record.data = this.record.data.filter(item => item.id !== id);
      this.record._updateChangeSubscription();
      this.message.open('Видалено', '', {
        duration: 5000,
        announcementMessage: 'Видалено',
        panelClass: 'success'
      });
    },
    error => {
      this.message.open('Помилка', '', {
        duration: 5000,
        announcementMessage: 'Помилка',
        panelClass: 'error'
      });
    }
  );
}

edit(id: number): void {
  this.router.navigate(['app/form'], {
    queryParams: {
      state: 'edit',
      id
    }
  });
}

add(): void {
  this.router.navigate(['app/form'], {
    queryParams: {
      state: 'add'
```

```
    }
  });
}
```

ДОДАТОК Б

Angular-сервіс

```
import {Injectable} from '@angular/core';
import {Observable} from 'rxjs';
import {environment} from '../../../environments/environment.prod';
import {HttpClient, HttpHeaders} from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class RecordService {

  apiUrl = environment.apiUrl;
  headers: any;

  constructor(private http: HttpClient) {
    this.headers = new HttpHeaders({
      Accept: 'application/json',
      'Content-Type': 'application/json',
      'Access-Control-Allow-Origin': '*',
      'Access-Control-Allow-Headers': 'Content-Type',
      'Access-Control-Allow-Methods': 'GET,POST,OPTIONS,DELETE,PUT',
    });
  }

  getByUserId(id: number): Observable<any> {
    return this.http.get(
      this.apiUrl + '/record/' + id,
      {headers: this.headers}
    );
  }

  getById(id: number): Observable<any> {
    return this.http.get(
      this.apiUrl + '/record/elem/' + id,
```

```
      {headers: this.headers}
    );
  }

  create(value: any): Observable<any> {
    return this.http.post(
      this.apiUrl + '/record',
      value,
      {headers: this.headers}
    );
  }

  update(value: any, id: number): Observable<any> {
    return this.http.patch(
      this.apiUrl + '/record/' + id,
      value,
      {headers: this.headers}
    );
  }

  delete(id: number): Observable<any> {
    return this.http.delete(
      this.apiUrl + '/record/' + id,
      {headers: this.headers}
    );
  }
}
```

ДОДАТОК В

Laravel-модель

```
<?php

namespace App\Models;

use App\Traits\Date;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class Record extends Model
{
    use HasFactory, Date;

    protected $table = 'reports';
    protected $primaryKey = 'id';

    const CREATED_AT = 'created_at';
    const UPDATED_AT = 'updated_at';

    protected $fillable = [
        'id',
        'user_faculty_id',
        'user_department_id',
        'user_id',
        'discipline_id',
        'topic_id',
        'kind_id',
        'hour',
        'semester',
        'date',
        'group_faculty',
        'course_id',
        'group',
```

```
    'form_id',
    'is_custom_group'
];

public function getId() {
    return $this->id;
}

/**
 * @return BelongsTo
 */
public function faculties(): \Illuminate\Database\Eloquent\Relations\BelongsTo
{
    return $this->belongsTo(Faculty::class, 'user_faculty_id');
}

/**
 * @return BelongsTo
 */
public function departments(): BelongsTo
{
    return $this->belongsTo(Department::class, 'user_department_id');
}

/**
 * @return BelongsTo
 */
public function users(): BelongsTo
{
    return $this->belongsTo(User::class, 'user_id');
}

/**
 * @return BelongsTo
 */
public function disciplineKinds(): BelongsTo
{
    return $this->belongsTo(DisciplineKind::class, 'kind_id');
```

```
    }

    /**
     * @return BelongsTo
     */
    public function courses(): BelongsTo
    {
        return $this->belongsTo(Course::class, 'course_id');
    }

    /**
     * @return BelongsTo
     */
    public function forms(): BelongsTo
    {
        return $this->belongsTo(Form::class, 'form_id');
    }

    /**
     * @return BelongsTo
     */
    public function disciplines(): BelongsTo
    {
        return $this->belongsTo(Discipline::class, 'discipline_id');
    }

    public function topics(): BelongsTo
    {
        return $this->belongsTo(Topic::class, 'topic_id');
    }
}
```

ДОДАТОК Г

Laravel-контролер

```
<?php

namespace App\Http\Controllers\Api;


use App\Models\Report;
use Exception;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use Validator;

class RecordController
{

    /**
     * @param Request $request
     * @return JsonResponse
     */
    public function store(Request $request): JsonResponse
    {
        $data = $request->all();

        $validator = Validator::make($data, [
            'user_faculty_id' => 'required',
            'user_department_id' => 'required',
            'user_id' => 'required',
            'discipline_id' => '',
            'topic_id' => '',
            'kind_id' => 'required',
            'hour' => 'required',
            'date' => 'required',
            'group_faculty' => 'string',
```

```
        'course_id' => '',
        'form_id' => '',
        'group' => '',
        'is_custom_group' => ''
    ]);

    if($validator->fails()){
        return response()->json($validator->errors(), 400);
    }

    $report = Report::create($validator->validated());

    return response()->json(['record' => $report, 'message' => 'Success'], 200);
}

/**
 * @param Request $request
 * @param $id
 * @return JsonResponse
 */
public function update(Request $request, $id): JsonResponse
{
    $data = $request->all();
    try {
        Report::where('id', intval($id))->update($data, ['upsert'=>true]);
    } catch (Exception $exception) {
        return response()->json(['error' => $exception, 'message' => 'Error'], 200);
    }

    return response()->json(['record' => Report::find($id), 'message' => 'Retrieved successfully'], 200);
}

/**
 * @param $id
 * @return JsonResponse
 */
public function destroy($id): JsonResponse
{
```

```
        try {
            Report::where('id', intval($id))->delete();
        } catch (Exception $exception) {
            return response()->json(['error' => $exception, 'message' => 'Error'], 200);
        }

        return response()->json(['message' => 'Success'], 200);
    }

    /**
     * @param $id
     * @return JsonResponse
     */
    public function getByUserId($id): JsonResponse
    {
        $reports = Report::with(['faculties', 'departments', 'users', 'disciplines.semesters', 'topics',
'disciplineKinds', 'courses', 'forms'])
            ->select('reports.*', DB::raw('group_concat(g.name) as group_name'))
            ->leftJoin('groups as g', DB::raw('FIND_IN_SET(g.id, reports.group)'),'>',DB::raw('"0"'))
            ->where('user_id', $id)->groupBy('reports.id')->get();
        return response()->json(['records' => $reports, 'message' => 'Success'], 200);
    }

    public function getById($id): JsonResponse
    {
        $report = Report::with(['faculties', 'departments', 'users', 'disciplines.semesters', 'topics',
'disciplineKinds', 'courses', 'forms'])
            ->where('id', $id)->get();
        return response()->json(['record' => $report, 'message' => 'Success'], 200);
    }

}
```