НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

До захисту допущено:
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційне забезпечення робототехнічних систем»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Вебресурс «Освітня платформа» з особистим кабінетом студента за допомогою фреймворку Laravel»

Виконав:
студент IV курсу, групи ІК-92
Кондратенко Віталій Ігорович _____

Керівник:
к.т.н., доц.,
Ткач Михайло Мартинович _____

Рецензент:
доцент каф. ІПІ, к.т.н., доцент
Ліхоузова Тетяна Анатоліївна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 126 «Інформаційні системи та технології»
Освітньо-професійна програма «Інформаційне забезпечення робототехнічних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

ЗАВДАННЯ
на дипломний проєкт студенту
Кондратенку Віталію Ігоровичу

1. Тема проєкту «Вебресурс «Освітня платформа» з особистим кабінетом студента за допомогою фреймворку Laravel», керівник проєкту доц., Ткач Михайло Мартинович, затверджені наказом по університету від «31» травня 2023 р. № 2101-с
2. Термін подання студентом проєкту: «12» червня 2023 року
3. Вихідні дані до роботи Існуючі модулі в загальній системі кафедри. Мова програмування серверної частини — PHP 8.0.28. Фреймворк — Laravel 9.52.8. Технології програмування клієнтської частини — HTML, JavaScript, CSS, Laravel Blade. Середовище програмування – PhpStorm 2023.1.2
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
   - Аналіз існуючих схожих систем;
   - Розробка архітектурних та функціональних вимог;
   - Проєктування системи;
   - Аналіз та вибір технологій та інструментів розробки;
   - Розробка серверної частини застосунку;
   - Реалізація веб-додатку;
   - Детальний опис організації взаємодії користувачів з системо
5. Перелік графічного матеріалу матеріалу «Схожі системи», «Технології реалізації», «Діаграма прецедентів», «Діаграма класів», «Знімки кінцевого продукту».
6. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Затвердження теми роботи	10.03.2023	Виконано
2.	Аналіз та осмислення теми	20.03.2023	Виконано
3.	Розробка структури та загальної архітектури системи	01.04.2023	Виконано
4.	Розробка архітектури окремих підсистем	15.04.2023	Виконано
5.	Програмна реалізація проєкту	20.05.2023	Виконано
6.	Написання пояснювальної записки	27.05.2023	Виконано

Студент Віталій КОНДРАТЕНКО

Керівник Михайло ТКАЧ

АНОТАЦІЯ

Обсяг роботи: Проєкт містить 88 с. тексту, 35 рисунків, 3 таблиці, посилання на 10 літерату-рні джерела, 2 додатки.

Мета роботи: Основним напрямом вебресурсу «Освітня платформа» з особистим кабінетом студента за допомогою фреймворку Laravel було обрано систему інформування викладачів/студентів. Це є інноваційним рішенням, яке дозволяє забезпечити ефективну комунікацію, обмін інформацією та надання актуальних даних між викладачами та студентами.

У ході виконання проєкту було створено систему модуля кабінету для студентів та викладачів. Проведено аналіз вже існуючого, подібного програмного забезпечення. Здійснено реалізацію застосунку, його бази даних, розроблено веб-систему, використовуючи технології Blade.php, Laravel фреймворку.

Ключові слова: викладач, студент, пост, коментар, HTML, JavaScript, PHP, Blade, Laravel, MySQL, Eloquent.

ABSTRACT

Scope of work: The project contains 88 p. text, 35 figures, 3 tables, references to 10 literary sources, 2 appendices.

Purpose of the work: The main direction of the web resource "Educational platform" with a student's personal account using the Laravel framework was the system of informing teachers/students. This is an innovative solution that allows for effective communication, information exchange and provision of up-to-date data between teachers and students.

As a result of the work, a cabinet module system was created for students and teachers. An analysis of existing, similar software was carried out. The design of the application, its database, and the web system were developed using Blade.php technologies and the Laravel framework.

Keywords: teacher, student, post, comment, application, JavaScript, PHP, Blade.php, Laravel, MYSQL, Eloquent.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІК92.140БАК.005 ПЗ	Пояснювальна записка	60		
6	А3	ІК92.140БАК.005 Д1	Вебресурс «Освітня	1		
7			платформа» з особистим			
8			кабінетом студента за			
9			допомогою фреймворку			
10			Laravel. Діаграма варіантів			
11			створення особистого			
12	А3	ІК92.140БАК.005 Д2	Вебресурс «Освітня	1		
13			платформа» з особистим			
14			кабінетом студента за			
15			допомогою фреймворку			
16			Laravel. Діаграма			
17			функціонування кабінету			
18	А3	ІК92.140БАК.005 Д3	Вебресурс «Освітня	1		
19			платформа» з особистим			
20			кабінетом студента за			
21			допомогою фреймворку			
22			Laravel. Діаграма функціоналу			
23			кабінету викладача			
24	А3	ІК92.140БАК.005 Д4	Вебресурс «Освітня	1		
25			платформа» з особистим			
26			кабінетом студента за			
27			допомогою фреймворку			
28			Laravel. ER-модель бази даних			

					ІК92.140БАК.005 ТП		
Зм.	Аркуш	№ докум.	Підпис	Дата			
Розроб.		Кондратенко В.І.			Вебресурс «Освітня платформа» з особистим кабінетом студента за допомогою фреймворку Laravel. Відомість проєкту	Літ. Т	Аркуш 1 / Аркушів 1
Керівн.		Ткач М. М.				КПІ ім. Ігоря Сікорського Група ІК-92	
Затв.							

Пояснювальна записка
до дипломного проєкту
на тему: «Вебресурс «Освітня платформа» з особистим кабінетом студента за допомогою фреймворку Laravel»

Київ – 2023 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 4
ВСТУП 6
1 АНАЛІЗ ПРОЮЛЕМИ НЕСВОЄЧАСНОГО ОТРИМАННЯ ІНФОРМАЦІЇ СТУДЕНТАМИ/ВИКЛАДАЧАМИ 8
1.1 Аналогічні системи 9
1.1.1 Кабінет студента КПІ в «Кампусі» 9
1.1.2 Силабус факультету ФІОТ 12
1.2 Необхідність системи інформування викладачів/студентів 16
1.3 Недоліки сучасних систем інформування викладачів/студентів 16
2 ЗАСОБИ РОЗРОБКИ 19
2.1 Серверна частина 19
2.1.1 Laravel 19
2.1.2 Routing 22
2.1.3 Middleware 23
2.1.4 MySQL 24
2.1.5 Eloquent ORM 27
2.1.6 PHP 8 29
2.2 Середовище розробки 33
2.2.1 PHPStorm 33
2.3 Веб-ресурс 34
2.3.1 HTML 34
2.3.2 JavaScript 37
2.3.3 jQuery JavaScript Framework 39
2.3.4 Laravel Blade 40

3 ОПИС ПРОГРАМНОЇ ЧАСТИНИ 43
3.1 Діаграма прецедентів 43
3.2 Діаграма класів системи 45
3.3 API 48
3.4 Вебсайт 52
4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ 55
ВИСНОВКИ 59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 61

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

База даних (БД) — це поіменована, структурована сукупність взаємопов’язаних даних, які належать до певної предметної області. Бази даних створюють з певною метою, і для зручності опрацювання даних використовуються комп’ютери та спеціальні програми. Сукупність атрибутів об’єкта та їх значень є даними конкретної бази даних, які мають певну логічну структуру, тобто описуються деякою інформаційною моделлю даних. Визначення об’єктів та їх атрибутів залежить від мети побудови бази даних: сфери застосування та потреб користувачів.

HTML (Hypertext Markup Language — Мова гіпертекстової розмітки) — це мова опису структури сторінок документів, яка дозволяє звичайний текст форматувати в абзаци, заголовки, списки та інші структури, створювати посилання на інші сторінки. Це текстова мова, в якій інструкції з форматування, що називаються тегами, вбудовані в розділи документа, які містять конкретну інформацію. Теги повідомляють браузерам, як форматувати і представляти інформацію на екрані.

CSS (Cascading Style Sheets або каскадні таблиці стилів) називається мова візуального представлення вмісту HTML-документа. Він має на увазі додавання кольору, редагування відступів, шрифтів, розташування кожного блоку на сторінці.

Blade — це простий, але потужний механізм створення шаблонів, який входить до складу Laravel. На відміну від деяких механізмів створення шаблонів PHP, Blade не обмежує використання простого коду PHP у ваших шаблонах.

CRUD — Стандартна абревіатура, що означає Create, Read, Update, Delete (Створення, Читання, Оновлення, Видалення). Використовується для позначення операцій, які можна виконувати з моделями або ресурсами.

API — Application Programming Interface (Інтерфейс програмування додатків). Використовується для позначення елементів, пов'язаних з розробкою API, таких як маршрути, контролери або запити.

MVC — Model-View-Controller (Модель-Представлення-Контролер). Використовується для позначення елементів, пов'язаних з архітектурою MVC, наприклад, моделі, представлення або контролери.

ORM — Object-Relational Mapping (Об'єктно-реляційне відображення). Використовується для позначення елементів, пов'язаних з роботою з базою даних за допомогою ORM, таких як моделі, міграції або запити.

JWT — JSON Web Token (Токен доступу на основі JSON). Використовується для позначення елементів, пов'язаних з аутентифікацією та авторизацією, які використовують JWT, такі як маршрути, контролери або middleware.

SPA — Single-Page Application (Односторінковий додаток). Використовується для позначення елементів, пов'язаних з розробкою односторінкових додатків, таких як маршрути, контролери або компоненти.

UI — User Interface (Користувацький інтерфейс). Використовується для позначення елементів, пов'язаних з розробкою інтерфейсу користувача, таких як шаблони, компоненти або стилі.

ACL — Access Control List (Список керування доступом). Використовується для позначення елементів, пов'язаних з керуванням доступом до ресурсів, таких як ролі, дозволи або політики.

HTTP (Hypertext Transfer Protocol) – протокол передачі гіпер-текстових документів.

ВСТУП

Основним напрямом вебресурсу «Освітня платформа» з особистим кабінетом студента за допомогою фреймворку Laravel було обрано систему інформування викладачів/студентів.

У сучасному світі, де швидкість і доступ до інформації стали невід'ємною частиною нашого повсякденного життя, системи інформування в реальному часі стають все більш популярними і важливими. Одним зі сфер, які отримали значні переваги від таких систем, є сфера освіти.

Система інформування викладачів/студентів у реальному часі є інноваційним рішенням, яке дозволяє забезпечити ефективну комунікацію, обмін інформацією та надання актуальних даних між викладачами та студентами. Ця система базується на фреймворку Laravel, який є одним з найпопулярніших фреймворків розробки веб-додатків на сьогоднішній день.

Використання фреймворку Laravel дозволяє розробникам створювати потужні, гнучкі та масштабовані системи інформування, що працюють в реальному часі. Завдяки багатому функціоналу Laravel, розробники можуть швидко реалізувати різноманітні функції, такі як миттєве повідомлення про оновлення розкладу, нагадування про дедлайни, повідомлення про зміни в матеріалах, оголошення та багато іншого.

Ця система не тільки полегшує комунікацію між викладачами та студентами, але й забезпечує відстеження актуальної інформації в режимі реального часу. Вона дозволяє студентам бути в курсі всіх найважливіших подій, оновлень та змін у навчальному процесі, що сприяє підвищенню їх продуктивності та успішності.

Окрім того, цей сервіс на базі фреймворку Laravel є зручною та легко налаштовуваною. Вона може бути адаптована до потреб конкретного навчального закладу, дозволяючи налаштовувати різні рівні доступу, функціональні можливості та інтерфейс користувача.

Загалом, своєчасне отримання інформації відкриває нові можливості для поліпшення навчального процесу та сприяє ефективній комунікації між всіма учасниками освітнього середовища. Це дозволяє забезпечити швидкий доступ до актуальної інформації та сприяє покращенню якості навчання та результатів студентів.

Для розробки серверної частини застосунку було обрано PHP фреймворк — Laravel, для клієнтської частини — прогресивний реактивний двигун Blade шаблонів, Javascript та допоміжні бібліотеки. В якості системи управління базами даних було обрано MySQL, та ORM Eloquent.

1 АНАЛІЗ ПРОЮЛЕМИ НЕСВОЄЧАСНОГО ОТРИМАННЯ ІНФОРМАЦІЇ СТУДЕНТАМИ/ВИКЛАДАЧАМИ

У контексті карантину або дистанційного навчання, проблема несвоєчасного отримання інформації студентами та викладачами стає особливо актуальною. Традиційні методи комунікації, такі як особисті зустрічі, оголошення на дошці або розповсюдження паперової документації, можуть бути недостатньо ефективними або недоступними під час дистанційного навчання.

Одна з основних проблем полягає в тому, що інформація може надходити затримкою або бути неактуальною, оскільки існує багато каналів комунікації та джерел інформації. Студенти та викладачі можуть втратити час, шукаючи актуальну інформацію в різних джерелах, що може призвести до плутанини і незрозуміння.

Додатковою проблемою є обмежена можливість взаємодії та обміну інформацією між студентами та викладачами у режимі реального часу. Відсутність ефективних засобів комунікації може призвести до зниження якості навчання, ускладнити процес вирішення питань або отримання допомоги, а також знизити мотивацію студентів через відчуття відокремленості та недостатньої підтримки.

Окрім того, проблема безпеки та конфіденційності також важлива. Недостатньо захищені канали комунікації можуть призвести до витоку особистої інформації, порушення приватності студентів або можливості зловживання даними.

Враховуючи ці проблеми, розробка системи інформування викладачів та студентів у реальному часі, заснованої на фреймворку Laravel, може стати вирішенням. Ця система дозволить студентам та викладачам отримувати актуальну інформацію, сповіщення та повідомлення у режимі реального часу, а також забезпечить зручний доступ до необхідних матеріалів та інструментів для навчання та спілкування.

Використання фреймворку Laravel дозволить забезпечити стабільність, безпеку та розширюваність системи, забезпечуючи ефективну комунікацію та обмін інформацією у часи карантину або дистанційного навчання.

1.1 Аналогічні системи

1.1.1 Кабінет студента КПІ в «Кампусі»

Кабінет студента в кампусі Київського політехнічного інституту (КПІ) є важливою складовою студентського життя і навчання. Цей кабінет надає студентам доступ до різноманітних ресурсів, послуг та інформації, що допомагають їм забезпечити успішне навчання та зручне перебування в університетському середовищі. Ось деякі ключові аспекти та функції, які можуть бути присутніми в кабінеті студента КПІ:

Особиста інформація та профіль: Кабінет студента може містити особисту інформацію, таку як дані студента, контактна інформація, фотографія та інші деталі. Студенти можуть оновлювати свої профілі, включаючи зміну паролю та інших налаштувань облікового запису.

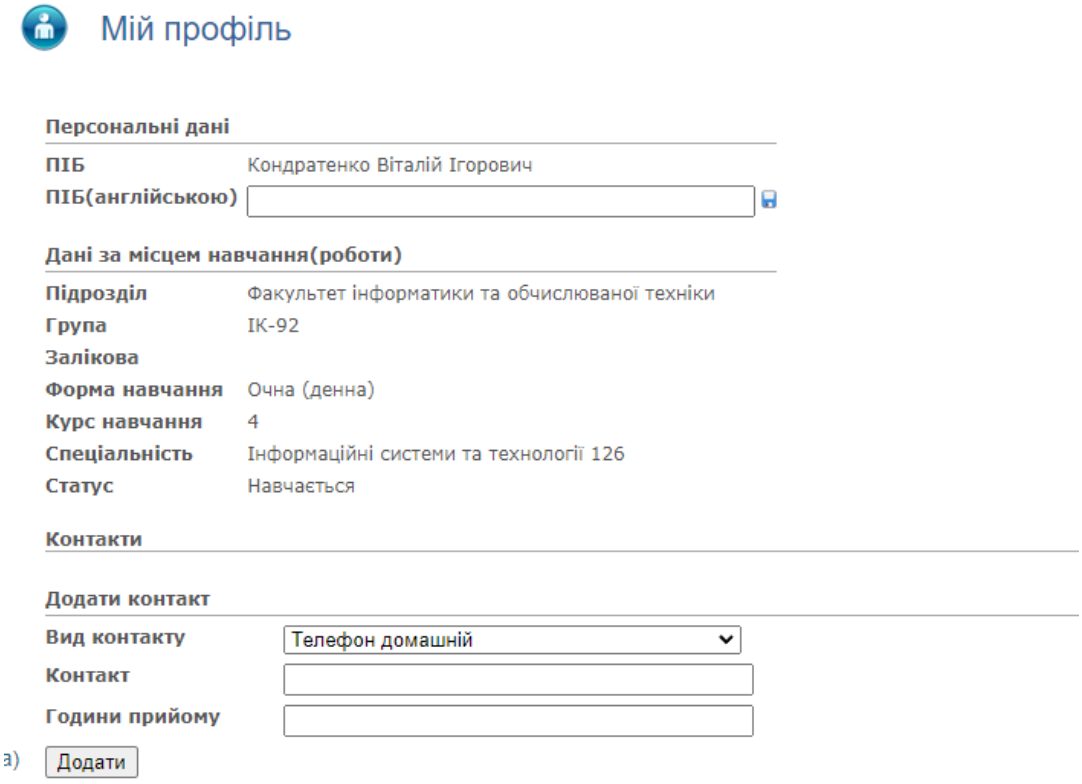
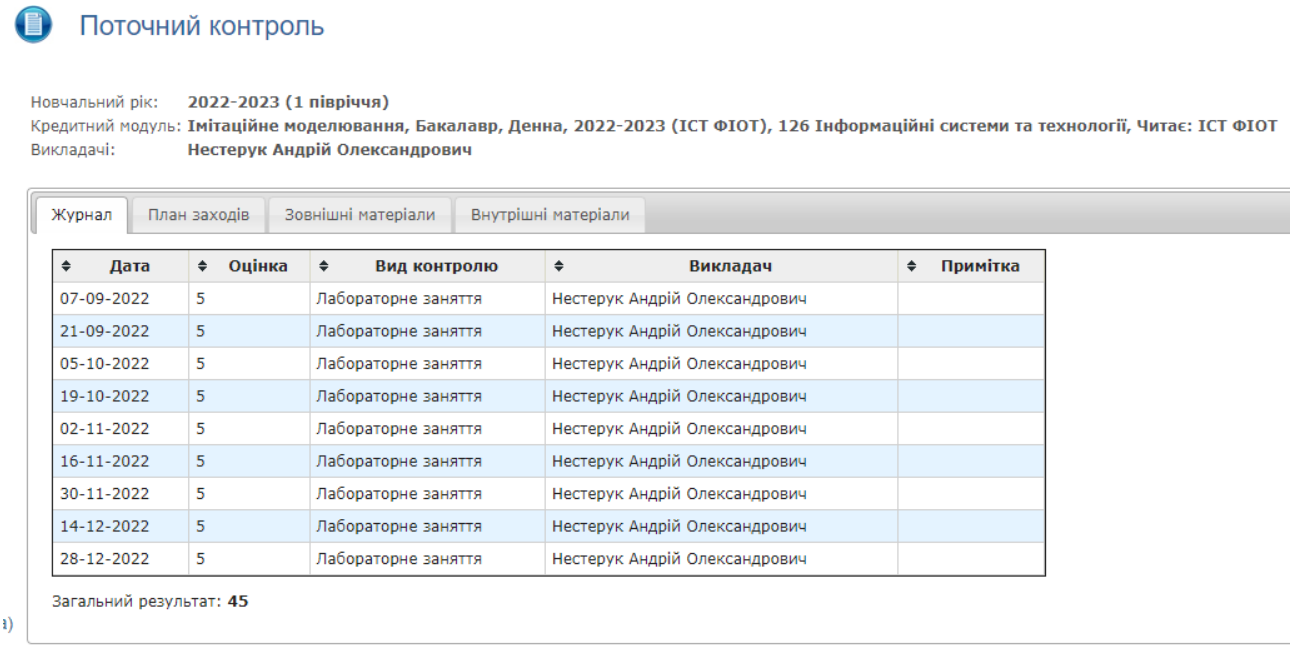


Рисунок 2.1.1.1 – Профіль студенту

Оцінки та академічна інформація: Студенти можуть переглядати свої академічні досягнення, включаючи оцінки за курси, результати іспитів та заліків. Кабінет може також надавати доступ до академічних календарів, правил та положень, а також інших ресурсів, що стосуються навчання.



Дата	Оцінка	Вид контролю	Викладач	Примітка
07-09-2022	5	Лабораторне заняття	Нестерук Андрій Олександрович	
21-09-2022	5	Лабораторне заняття	Нестерук Андрій Олександрович	
05-10-2022	5	Лабораторне заняття	Нестерук Андрій Олександрович	
19-10-2022	5	Лабораторне заняття	Нестерук Андрій Олександрович	
02-11-2022	5	Лабораторне заняття	Нестерук Андрій Олександрович	
16-11-2022	5	Лабораторне заняття	Нестерук Андрій Олександрович	
30-11-2022	5	Лабораторне заняття	Нестерук Андрій Олександрович	
14-12-2022	5	Лабораторне заняття	Нестерук Андрій Олександрович	
28-12-2022	5	Лабораторне заняття	Нестерук Андрій Олександрович	

Рисунок 2.1.1.2 – Поточний контроль

Електронна пошта та повідомлення: Кабінет може мати вбудований електронний поштовий ящик або систему повідомлень, яка дозволяє студентам отримувати офіційні сповіщення від університету, викладачів та інших студентів. Це може включати оголошення, нагадування про дедлайни, сповіщення про зміни в розкладі тощо (рис. 2.1.1.3).

Бібліотека та навчальні матеріали: Кабінет може мати доступ до онлайн бібліотеки, де студенти можуть шукати та замовляти книги, переглядати електронні журнали, статті та інші ресурси. Крім того, студенти можуть мати можливість завантажувати навчальні матеріали, лекції, презентації та інші документи, які пов'язані з їх курсами.

Фінансова інформація: Кабінет може надавати доступ до фінансової інформації, такої як рахунки за навчання, оплати, стипендії та фінансова допомога. Студенти можуть переглядати свої фінансові статуси, оплачувати рахунки та отримувати важливу інформацію щодо фінансових питань.

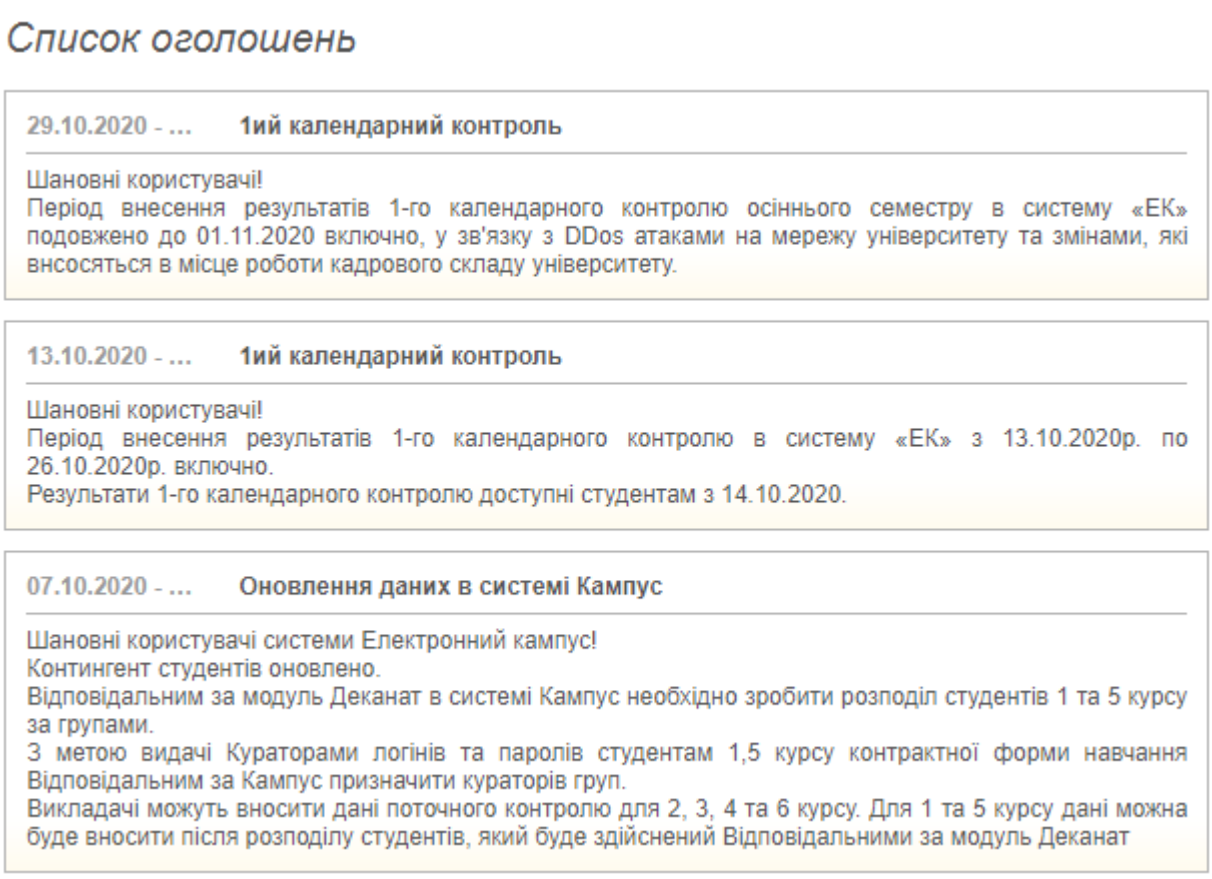


Рисунок 2.1.1.3 – Список оголошень

Заяви та реєстрація на курси: Студенти можуть мати можливість подавати заяви на різні послуги, такі як зміни в розкладі, заявки на переведення або реєстрацію на певні курси. Кабінет може забезпечувати простий та зручний процес заповнення та подання заявок.

Також кабінет студента «Кампус» має вагомі ознаки інформаційної платформи для студентів, але все і одразу в ньому не знайти. Крім того, ця система зовсім не може похизуватися сучасними системами безпеки контактної інформації користувачів, частотою оновлювання постів або оголошень.

Тому першу чергу, студенти використовують «Кампус», в першу чергу, для відслідковування своєї успішності в навчанні та дошки оголошень щодо конкретних заліків, екзаменів, сесій і т.д.

Сесія

Результати сесії

Якщо виникли питання, пишіть в бот Деканату => Телеграм

#	Дата заходу	Назва дисципліни	Вид контролю	Тип відомості	Викладач	Оцінка	Статус
1		Дипломне проектування	Екзамен	Основна	Не визначен деканатом		-
2	2023-05-25 00:00:00	Переддипломна практика	залік	Основна	Анікін Володимир Костянтинович	95	-
3	2023-04-18 00:00:00	Управління проектами	залік	Основна	Мелкумян Катерина Юріївна	73	-
4	2023-04-14 00:00:00	Іноземна мова професійного спрямування. Практичний курс для професійного спілкування II	Екзамен	Основна	Гайденко Юлія Олексіївна	60	-
5	2023-04-07 00:00:00	Екологічна та природно-техногенна безпека	залік	Основна	Прахович Наталія Артурівна	72	-
6	2023-04-06 00:00:00	Системи автоматизованого проектування комп'ютерних систем	залік	Основна	Сергієнко Анатолій Михайлович	60	-
7	2023-04-06 00:00:00	Статистичні методи машинного навчання	залік	Основна	Олещенко Любов Михайлівна	80	-
8	2023-04-05 00:00:00	Інтелектуальні транспортні системи	залік	Основна	Сокульський Олег Євгенович	75	-
9	2023-01-20 00:00:00	Проектування інформаційних систем	Екзамен	Основна	Мелкумян Катерина Юріївна	63	-
10	2023-01-20 00:00:00	Управління технічними системами	Екзамен	Основна	Цьопа Наталія Володимирівна	60	-
11	2023-01-20 00:00:00	Імітаційне моделювання	залік	Основна	Сокульський Олег Євгенович	75	-
12	2023-01-20 00:00:00	Інтелектуальні технології в робототехніці	залік	Основна	Олійник Володимир Валентинович	67	-
13	2023-01-20 00:00:00	Економіка і підприємництво	залік	Основна	Семенченко Наталія Віталіївна	60	-
14	2023-01-20 00:00:00	Курсова робота з Інтелектуальних технологій в робототехніці	залік	Основна	Олійник Володимир Валентинович	61	-
15	2023-01-20 00:00:00	Основи комп'ютерного моделювання	залік	Основна	Іваніщев Богдан Вячеславович	60	-
16	2023-01-20 00:00:00	Проектування та реалізація операційних систем	залік	Основна	Сімоненко Андрій Валерійович	75	-
17	2022-07-07 11:00:00	Курсовий проект з робототехнічних систем та комплексів	Курсовий проект	Основна	Остапченко Костянтин Борисович	70	-
18	2022-07-04 00:00:00	Робототехнічні системи та комплекси-3. Алгоритмізація та верифікація управління в РТС	залік	Основна	Остапченко Костянтин Борисович	81	-
19	2022-06-29 00:00:00	Іноземна мова професійного спрямування 2.	залік	Основна	Гайденко Юлія Олексіївна	60	-
20	2022-06-29 00:00:00	Моделювання технічних ситем	залік	Основна	Чумаченко Олена Іллівна	95	-
21	2022-06-29 00:00:00	Основи WEB-програмування/WEB - технології	залік	Основна	Крилов Євген Володимирович	60	-
22	2022-06-29 00:00:00	Розробка ігрових застосувань	залік	Основна	Катін Павло Юрійович	61	-
23	2022-06-29 00:00:00	Теорія систем та системний аналіз	залік	Основна	Чумаченко Олена Іллівна	100	-
24	2022-06-29 00:00:00	Технології програмування на мові Python	залік	Основна	Шевело Олексій Павлович	70	-
25	2022-06-29 00:00:00	Технологія блокчейн	залік	Основна	Яланецький Валерій Анатолійович	62	-
26	2022-01-14 14:00:00	Операційні системи мобільних роботів	Екзамен	Основна	Резніков Сергій Анатолійович	95	-
27	2022-01-10 14:00:00	Технології розроблення програмного забезпечення	Екзамен	Основна	Мелкумян Катерина Юріївна	60	-
28	2022-01-04 14:00:00	Безпека інформаційних систем	Екзамен	Основна	Корнага Ярослав Ігорович	95	-
29	2021-12-30 12:00:00	Основи Front-end та Backend розробок	залік	Основна	Проскура Світлана Леонідівна	95	-
30	2021-12-30 08:30:00	Робототехнічні системи та комплекси-2. Проектування РТС	залік	Основна	Ткач Михайло Мартинович	67	-
31	2021-12-29 14:15:00	Аналіз даних з використанням мови Python	залік	Основна	Тимофєєва Юлія Сергіївна	70	-
32	2021-12-29 12:20:00	Ігровий штучний інтелект	залік	Основна	Головченко Максим Миколайович	100	-
33	2021-12-28 12:20:00	Курсова робота з технологій розроблення програмного забезпечення	Курсова робота	Основна	Мелкумян Катерина Юріївна	75	-
34	2021-06-16 14:00:00	Робототехнічні системи та комплекси - 1. Проектування компонентів РТС	Екзамен	Основна	Поліщук Михайло Миколайович	90	-
35	2021-06-12 14:00:00	Технології інтернету речей	Екзамен	Основна	Жураковський Богдан Юрійович	95	-
36	2021-06-08 14:00:00	Комп'ютерна електроніка та мікропроцесорні системи - 1. Комп'ютерна електроніка	Екзамен	Основна	Пархомей Ігор Ростиславович	80	-

Рисунок 2.1.1.4 – Профіль успішності студенту

Переваги:

– Простий в засвоєнні

– Гарний сервіс, з точки зору, слідкування успішності студента.

– Інтуїтивний інтерфейс

– Має великий базу викладачів

Недоліки:

– Відсутність своєчасного оновлення інформації

– Уся контактна інформація про викладачів відсутня

– Пошук усієї інформацію, окрім успішності, переводить на сторонні ресурси

1.1.2 Силабус факультету ФІОТ

Силабус факультету ФІОТ – це особистий кабінет студента, з контактною інформацією користувача, текою студента, індивідуальним навчальним планом,

функціональним розкладом, можливістю звернення до деканату, вибір дисциплін конкретного року, курсу та сертифікатні програми.

Кабінет поділений на декілька секцій, детально розберемо кожну з них:

– Однією з сильних сторін цього силабусу є «Особистий профіль студента», який можна редагувати та видозмінювати.

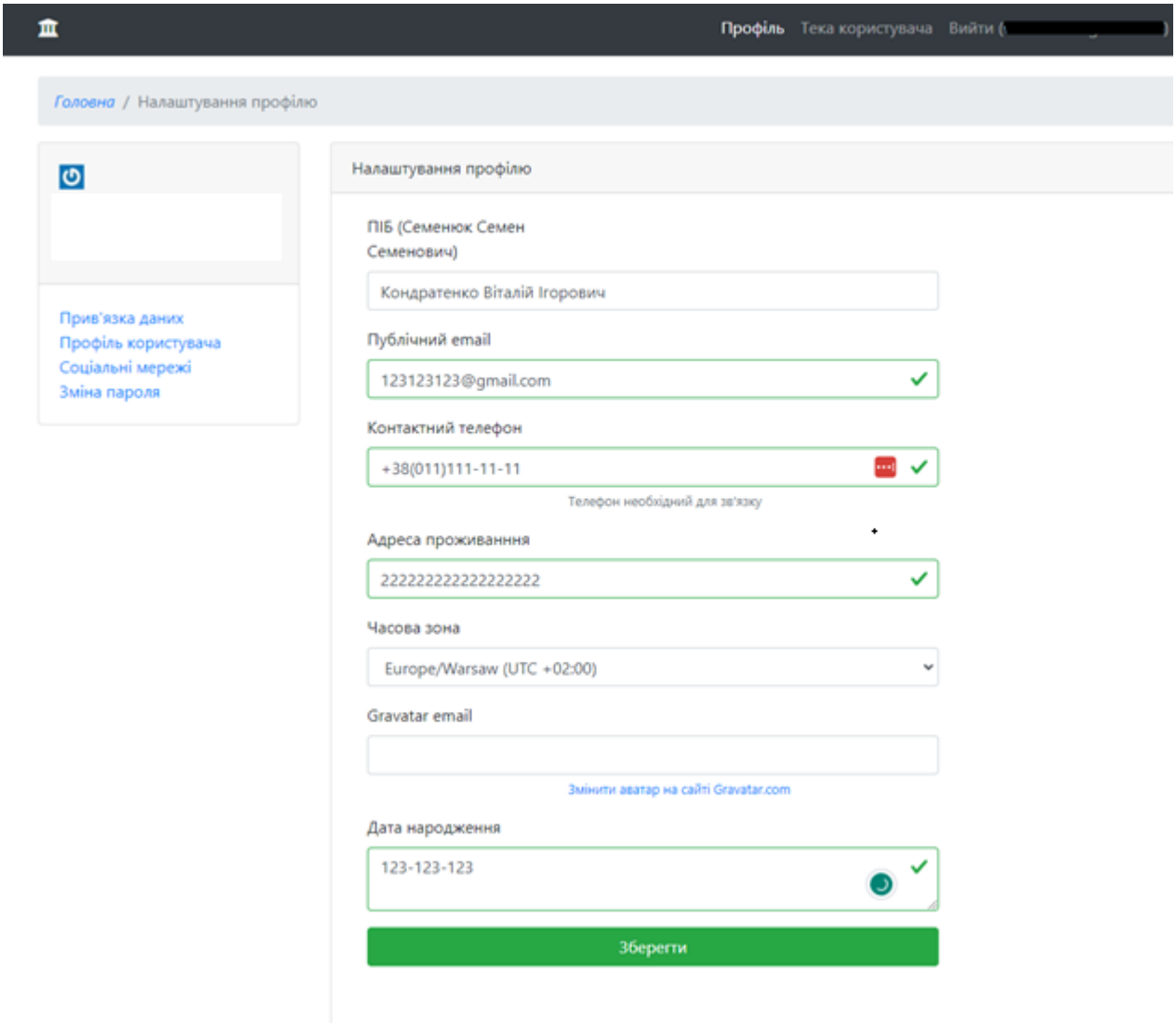


Рисунок 2.1.2.1 – Профіль студенту

– Не менш вагомим пунктом є тека користувача, яка увібрала основний функціонал цього сервісу. Основним плюсом якого є «Індивідуальний навчальний план» – це обрані предмети користувачем, конкретного курсу та року навчання.

Індивідуальний навчальний план

ІК-92

2022 - 4-й курс

PDF

№	Н/Д (О/К)	Кафедра	Сем	Кред	Год	Лек	Прак	Лаб	СРС	Контр.	МКР	Ін.звд
					Нормативні							
1	Економіка і підприємництво	ЕК	7	3.0	90	18	18	0	54	Залік	1	-
2	Іноземна мова професійного спрямування	АМГСЗ	7	1.5	45	0	36	0	9	-	1	-
3	Проєктування інформаційних систем	ІСТ	7	5.0	150	36	36	0	78	Екзамен	1	-
4	Управління технічними системами	ІСТ	7	4.0	120	36	18	0	66	Екзамен	1	-
5	Інтелектуальні технологій в робототехніці	ІСТ	7	3.5	105	36	18	0	51	Залік	1	-
6	Інтелектуальні технологій в робототехніці. Курсова робота	ІСТ	7	1.0	30	0	0	0	30	Залік	-	КР
7	Екологічна та природно-техногенна безпека	ОПЦБ	8	2.0	60	18	18	0	24	Залік	1	-
8	Переддипломна практика	ІСТ	8	6.0	180	0	0	0	180	Залік	-	-
9	Управління проєктами	ІСТ	8	3.0	90	18	18	0	54	Залік	1	-
10	Дипломне проектування	ІСТ	8	6.0	180	0	0	0	180	Захист	-	-
11	Іноземна мова професійного спрямування	АМГСЗ	8	1.5	45	0	18	0	27	Екзамен	-	-
					Обрані							
12	Основи комп'ютерного моделювання (Освітній компонент-10. Ф-Каталог)	ОТ	7	4.0	120	36	18	0	66	Залік	1	-
13	Імітаційне моделювання (Освітній компонент-11. Ф-Каталог)	ІСТ	7	4.0	120	36	18	0	66	Залік	1	-
14	Проєктування та реалізація операційних систем (Освітній компонент-12. Ф-Каталог)	ОТ	7	4.0	120	36	18	0	66	Залік	1	-

Рисунок 2.1.2.2 – Індивідуальний план

– Також в цьому додатку є вибір дисциплін на конкретний період часу

Але все одно ми повертаємось до проблеми надходження інформації з затримкою або бути неактуальною, оскільки існує багато каналів комунікації та джерел інформації. Студенти та викладачі можуть втратити час, шукаючи актуальну інформацію в різних джерелах, що може призвести до плутанини і незрозуміння.

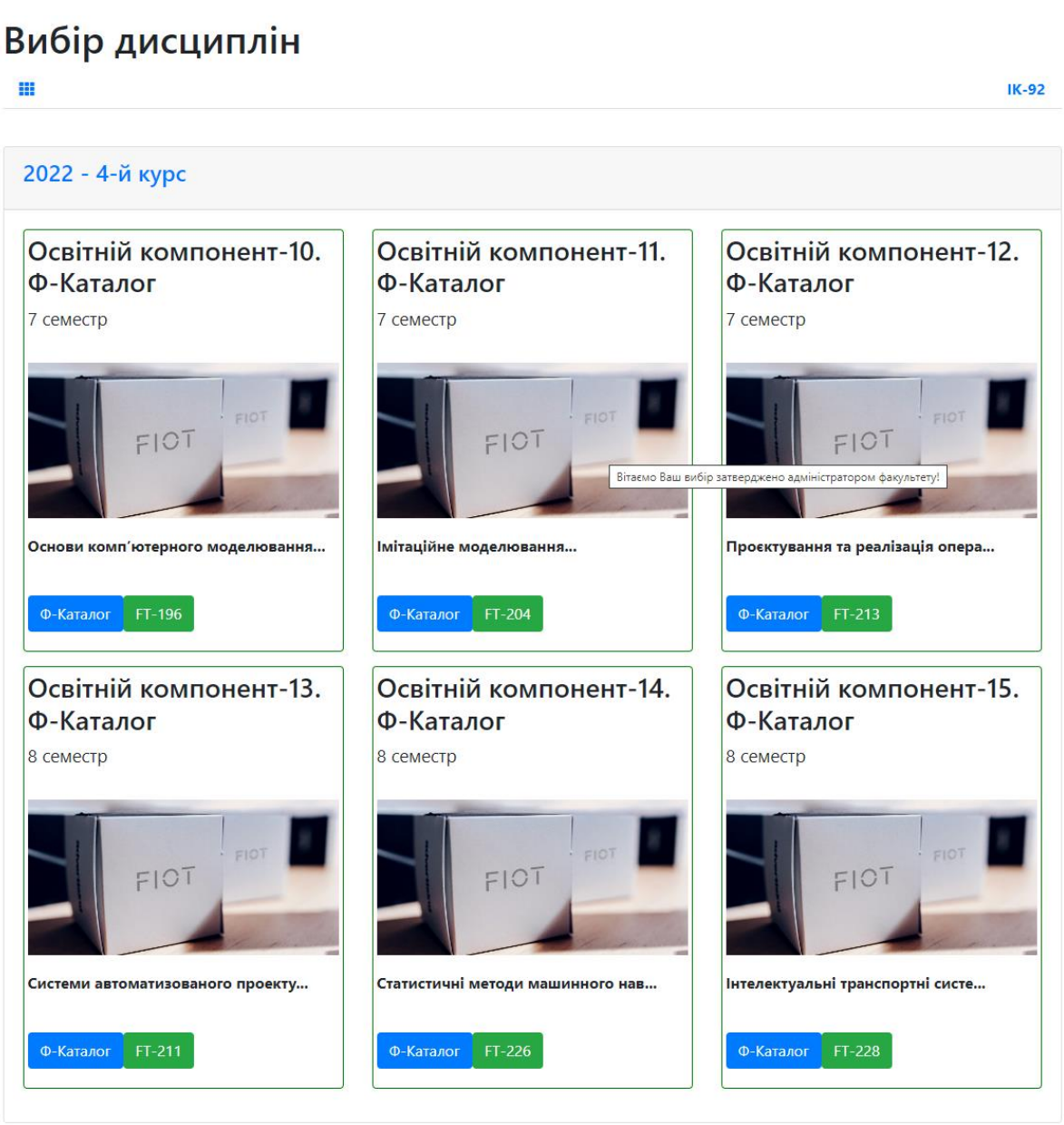


Рисунок 2.1.2.3 – Вибір дисциплін

В цій системі-силабусі не вистачає багатьох компонентів для коректного навчання студента сьогоденних реаліях.

Переваги:

– Простий в засвоєнні

– Інтуїтивний інтерфейс

– Індивідуальний план студента

Недоліки:

– Відсутність актуального розкладую.

– Немає можливості контакту з деканатом.

– Незрозуміло використовувати цей сервіс для обрання ІНП. Оскільки немає гарної інформаційної системи

1.2 Необхідність системи інформування викладачів/студентів

Розглянувши вже існуючі варіанти реалізації подібних до задуманої систем, стала зрозуміла необхідність запровадження власної, яка б цілком була побудована для потреб інформування студентів та персоналу навчального закладу, адже аналогічні системи спрямовані більше на, власне, навчальний процес, або розкладу сесій, заліків і тому подібне.

Система інформування у реальному часі дозволяє викладачам та студентам ефективно спілкуватися між собою. Вона забезпечує зручний канал обміну повідомленнями, де можна швидко обговорювати питання, вирішувати проблеми і взаємодіяти на різних рівнях навчального процесу

Також, усі з вище наведених систем є дуже застарілими, отже побудова власної, адаптованої до вимог системи є найкращим вирішенням.

В результаті аналізу наведених вище систем, було зроблено висновки з того, як повинен виглядати користувацький інтерфейс — має бути інтуїтивним, вміщати якомога більше інформації на одному екрані, мати можливість обговорення того чи іншого аспекту.

Також були обрані технології та інструменти розробки:

– PHP

– Бекенд — фреймворк Laravel

– Фронтенд — Blade, Javascript

– СУБД — MySQL

– ORM Eloquent

– Середовище розробки — PHPStorm.

1.3 Недоліки сучасних систем інформування викладачів/студентів

Незважаючи на те, що сучасні системи інформування викладачів та студентів мають багато переваг, таких як полегшення спілкування, збереження часу та покращення доступу до інформації, вони також мають свої недоліки. Ось деякі з них:

1. Складність використання: Деякі системи інформування можуть бути складними у використанні, особливо для користувачів, які не мають досвіду з технологіями. Це може вимагати додаткової навчання та підтримки для ефективного використання системи.

2. Технічні проблеми: Системи інформування можуть стикатися з технічними проблемами, такими як відмови серверів, перебої в мережі, проблеми зі сумісністю браузерів тощо. Це може призводити до збоїв у роботі системи та обмежувати доступ до необхідної інформації.

3. Залежність від Інтернету: Більшість систем інформування потребують постійного з'єднання з Інтернетом для доступу до функціональності та отримання оновлень. Це може створювати проблеми у випадку відсутності стабільного з'єднання або відсутності доступу до Інтернету.

4. Проблеми з конфіденційністю та безпекою: Використання систем інформування вимагає обміну особистими даними та конфіденційною інформацією. Це може ставити під загрозу приватність та безпеку, особливо у випадку недостатньої захищеності системи або недостатньої уваги до кібербезпеки.

5. Надмірна інформація: Іноді системи інформування можуть надавати надмірну кількість інформації, що може бути важко переглянути та зрозуміти. Це може призводити до перевантаження користувачів та зниження ефективності використання системи.

6. Відсутність особистого контакту: Використання систем інформування може знижувати особистий контакт та взаємодію між викладачами та студентами. Вони можуть втратити можливість обговорювати питання, отримувати негайний фідбек та встановлювати ближчі взаємини.

7. Потреба у підтримці та оновленнях: Системи інформування потребують постійної підтримки, оновлень та вдосконалення. Це може вимагати часу, ресурсів та фахових знань для забезпечення правильної роботи та задоволення потреб користувачів.

Це лише кілька недоліків, які можуть бути пов'язані з сучасними системами інформування викладачів та студентів. Важливо збалансувати переваги та недоліки при виборі та використанні системи, а також забезпечити належну підтримку та захист інформації.

2 ЗАСОБИ РОЗРОБКИ

У якості середовища розробки було використано PHPStorm, так як архітектура застосунку — монолітна, та уся робота проходила в одному репозиторії, а PHPStorm запроваджує статичний аналіз коду як для мови PHP, так і для JavaScript, HTML та CSS, та шаблонів Blade.

Серверною мовою програмування було обрано PHP, а конкретніше — фреймворк Laravel.

У якості СУБД використано MySQL.

Для побудови клієнтської частини було використано механізм створення шаблонів Blade.

2.1 Серверна частина

2.1.1 Laravel

Laravel [1] - це високопродуктивний, елегантний і простий у використанні фреймворк для розробки веб-додатків з використанням мови програмування PHP. Laravel надає розробникам потужні інструменти та компоненти, які допомагають в реалізації широкого спектру завдань, таких як маршрутизація, обробка запитів, робота з базою даних, аутентифікація користувачів та багато іншого.

Основні особливості Laravel:

1. Елегантний синтаксис та зрозумілість: Laravel має чистий та елегантний синтаксис, що дозволяє розробникам швидко створювати код, який легко читати та розуміти. Це сприяє збереженню часу при розробці та підтримці проекту.

2. Маршрутизація та обробка запитів: Laravel надає потужну систему маршрутизації, яка дозволяє легко визначати URL-шаблони та обробляти запити. Ви можете визначати маршрути з параметрами, обробляти RESTful запити та використовувати різноманітні HTTP методи.

3. ORM Eloquent: Laravel поставляється з Eloquent ORM, який надає простий та елегантний спосіб роботи з базами даних. Eloquent дозволяє визначати моделі та взаємодіяти з базою даних, використовуючи розширені можливості, такі як створення запитів, зв'язки між моделями та міграції.

4. Міграції та база даних: Laravel має вбудовану підтримку міграцій, що дозволяє легко керувати структурою бази даних. Ви можете визначати міграції для створення таблиць, індексів, зв'язків та виконувати зміни схеми бази даних безпосередньо з коду. Laravel також підтримує різні системи керування базами даних, такі як MySQL, PostgreSQL, SQLite та інші.

5. Аутентифікація та авторизація: Laravel має потужну систему аутентифікації, що дозволяє легко реалізувати функціональність реєстрації, входу користувача, відновлення паролю та багатокористувацьку авторизацію. Ви можете використовувати готові шаблони вигляду та контролери для швидкого впровадження цих функцій.

6. Кешування та оптимізація продуктивності: Laravel надає інструменти для кешування даних, що дозволяє зберігати часто використовувані дані у швидкодіючому сховищі та підвищувати продуктивність додатка. Ви також можете використовувати інші методи оптимізації, такі як попередня завантаження класів (eager loading), оптимізація автозавантаження (auto-loading) та кешування конфігурації.

7. Розширюваність: Laravel дозволяє легко розширювати функціональність за допомогою пакетів. Ви можете використовувати пакети, які надають додаткові функції та інтеграцію з іншими сервісами, такими як платіжні шлюзи, обробка зображень, інтеграція з соціальними мережами та багато іншого.

8. Тестування: Laravel має вбудовану підтримку для тестування, що дозволяє легко створювати та запускати тести для перевірки правильності роботи вашого додатка. Ви можете тестувати маршрути, контролери, моделі та інші складові системи Laravel.

9. Система шаблонів Blade: Laravel використовує свою власну систему шаблонів під назвою Blade. Blade надає зручний та експресивний спосіб включати PHP-код в шаблони та робити складніші речі, такі як умовні вирази, цикли та наслідування шаблонів.

10. Спільнота та документація: Laravel має активну та дружню спільноту розробників, яка надає підтримку, пакети, приклади коду та багато корисної інформації. Laravel також має добре написану документацію, яка допомагає вам швидко ознайомитися з фреймворком та використовувати його ефективно.

11. Роутинг: Laravel має потужну систему роутингу, яка дозволяє легко визначати маршрути для вашого додатку. Ви можете визначити GET, POST, PUT, DELETE та інші типи маршрутів, а також параметри маршруту і обробники, які виконуються при виклику маршруту. Це дозволяє вам організувати логіку вашого додатку та керувати потоком запитів.

12. Middleware: Middleware в Laravel - це шар проміжного програмного забезпечення, яке дозволяє обробляти запити перед тим, як вони потрапляють до маршруту або після того, як вони виходять з маршруту. Ви можете використовувати middleware для авторизації, перевірки даних, обробки сесій та багатьох інших завдань. Middleware дозволяє вам зручно розділяти логіку додатку на кілька етапів і забезпечує повну гнучкість в керуванні обробкою запитів.

13. Шаблонізатор Blade: Laravel використовує свій власний шаблонізатор Blade, який дозволяє зручно створювати та відображати HTML-шаблони. Blade має простий та зрозумілий синтаксис, який дозволяє використовувати умови, цикли, наслідування шаблонів, компоненти та багато інших корисних функцій. Використання шаблонів Blade спрощує розробку та підтримку вашого інтерфейсу.

14. Система міграцій: Laravel надає інтуїтивно зрозумілу та потужну систему міграцій, яка дозволяє легко керувати структурою бази даних. Ви можете використов

Laravel є одним з найпопулярніших фреймворків PHP завдяки своїй простоті, потужності та гнучкості. Він є ідеальним вибором для розробки веб-додатків будь-якої складності, від невеликих сайтів до великих корпоративних рішень.

2.1.2 Routing

Laravel Routing [11] - це механізм маршрутизації, який надається фреймворком Laravel. Він дозволяє визначати, як HTTP-запити до вашого веб-додатку повинні бути оброблені. Маршрутизація Laravel дуже гнучка і дозволяє вам легко визначати шляхи, оброблювати параметри маршруту, застосовувати фільтри та багато іншого.

Основні компоненти маршрутизації Laravel включають:

1. Маршрути: Маршрути визначають шаблони URL-адрес, які ваш додаток може обробляти. Вони визначаються за допомогою методу `Route::` і доступні в файлі `routes/web.php` або `routes/api.php`. Маршрути можуть мати параметри, наприклад, `{id}`, для отримання даних з URL.

2. Обробники маршрутів: Кожен маршрут пов'язується з обробником, який визначає, що робити, коли відповідний маршрут збігається з URL-адресою. Обробники маршрутів можуть бути анонімними функціями (`Closure`) або контролерами Laravel.

3. Групи маршрутів: Laravel дозволяє групувати маршрути, що мають спільні атрибути, в одному місці. Це допомагає організувати ваші маршрути та застосовувати фільтри (middleware) до всіх маршрутів у групі.

4. Фільтри маршрутів (middleware: Фільтри маршрутів, також відомі як middleware, дозволяють вам виконувати операції перед або після обробки запиту до маршруту. Наприклад, ви можете використовувати middleware для перевірки автентифікації користувача перед доступом до певної сторінки.

5. Іменовані маршрути: Laravel дозволяє надавати імена маршрутам, що дозволяє легко посилатися на них в вашому додатку. Це зручно, коли ви хочете

згенерувати URL на підставі імені маршруту, а не на основі конкретного шляху.

6. Генерування URL: Laravel надає зручні методи для генерації URL на основі імені маршруту або шляху маршруту. Це дозволяє легко створювати посилання на інші сторінки в вашому додатку.

Основним файлом для визначення маршрутів є `routes/web.php` для веб-додатків та `routes/api.php` для API-додатків. У цих файлах ви можете визначати свої маршрути, пов'язані обробники, фільтри та інші налаштування.

2.1.3 Middleware

Laravel Middleware [12] - це механізм, який дозволяє виконувати фільтрацію та обробку HTTP-запитів перед тим, як вони досягнуть обробників маршрутів у вашому Laravel додатку. Middleware може виконувати певні дії перед або після обробки запиту, такі як перевірка автентифікації, перевірка прав доступу, логування, маніпуляція даними запиту і багато іншого.

Основні особливості Laravel Middleware:

– Функціональність: Middleware може бути використано для будь-яких додаткових операцій, які ви бажаєте виконати перед або після обробки запиту. Це можуть бути перевірки безпеки, перенаправлення запитів, зміна даних запиту, запис до журналу тощо.

– Глобальні та маршрутні Middleware: У Laravel існує можливість використовувати глобальне Middleware, яке застосовується до кожного запиту до вашого додатку, або маршрутне Middleware, яке застосовується тільки до певних маршрутів.

– Структуровані Middleware: Ви можете організовувати ваші Middleware в групи або структурувати їх за певними принципами. Це допомагає зробити код більш організованим та легким для керування.

– Послідовність Middleware: Middleware виконуються в заданому порядку. Ви можете налаштувати порядок виконання Middleware в файлі

app/Http/Kernel.php, де ви можете визначити глобальні Middleware та групи Middleware.

– Своя Middleware: Ви також можете створювати власні Middleware в Laravel. Для цього ви можете використовувати Artisan-команду make:middleware, яка згенерує новий клас Middleware, де ви зможете визначити свою логіку фільтрації.

Приклад використання Middleware в Laravel (рис 3.1.3.1)

```php
<?php

namespace App\Http\Middleware;

use Closure;

class Authenticate
{
    public function handle($request, Closure $next)
    {
        if (!auth()->check()) {
            return redirect('/login');
        }

        return $next($request);
    }
}
```

Рисунок 3.1.3.1 -- Приклад використання Middleware в Laravel

2.1.4 MySQL

MySQL[2] - це реляційна база даних (RDBMS), яка є однією з найпопулярніших та широко використовуваних систем керування базами даних. Вона пропонує потужні функціональні можливості для зберігання та керування даними, що дозволяє розробникам створювати ефективні та масштабовані

додатки.

Основні характеристики MySQL:

1. Реляційна структура: MySQL базується на реляційній моделі даних, що дозволяє зберігати дані у вигляді таблиць зі зв'язками між ними. Це спрощує організацію та управління даними, а також забезпечує цілісність даних.

2. Мова запитів SQL: MySQL використовує структуровану мову запитів SQL (Structured Query Language) для взаємодії з базою даних. SQL дозволяє виконувати операції вибірки, вставки, оновлення та видалення даних, а також визначати структуру таблиць, індексів та зв'язків.

3. Підтримка транзакцій: MySQL підтримує транзакції, що дозволяють групувати кілька операцій у логічну одиницю роботи. Транзакції забезпечують атомарність, консистентність, ізольованість та стійкість (ACID властивості), що робить базу даних надійною та безпечною.

4. Широка підтримка типів даних: MySQL надає багатий набір вбудованих типів даних, таких як цілі числа, десяткові числа, рядки, дата і час, булеві значення та багато інших. Крім того, ви можете визначати власні типи даних та функції.

5. Індексування: MySQL дозволяє створювати індекси на таблицях для підвищення швидкодії запитів. Індекси дозволяють ефективно знаходити, сортувати та фільтрувати дані, що знижує час виконання запитів.

6. Масштабованість та висока продуктивність: MySQL може працювати з великими обсягами даних і забезпечує високу продуктивність завдяки ефективному оптимізатору запитів, кешуванню, паралельному виконанню та іншим оптимізаціям.

7. Безпека: MySQL надає різні механізми безпеки, такі як ролі та дозволи, шифрування даних, автентифікація користувачів та механізми перевірки цілісності даних. Це дозволяє забезпечувати конфіденційність, цілісність та доступ до даних.

8. Реплікація та висока доступність: MySQL підтримує реплікацію, що дозволяє створювати копії бази даних для підвищення доступності та надійності.

Реплікація дозволяє розподілити навантаження та забезпечити забезпечення резервних копій.

MySQL є відкритим програмним забезпеченням та підтримується широкою спільнотою розробників. Він використовується у багатьох веб-додатках, від невеликих сайтів до великих підприємственних систем, завдяки своїй надійності, продуктивності та гнучкості.

Додатково параметри про MySQL:

– Хешування паролів : MySQL має вбудовану функціональність для хешування паролів користувачів. Це дозволяє зберігати паролі у захищеному вигляді, що підвищує безпеку системи.

– Транзакційна підтримка: MySQL підтримує транзакції, які дозволяють виконувати групу зв'язаних запитів як єдину логічну операцію. Це забезпечує цілісність даних та можливість відновлення до попереднього стану в разі помилок або відміни операцій.

– Реплікація та шардування: MySQL дозволяє налаштовувати реплікацію, що дозволяє створювати декілька копій бази даних для забезпечення високої доступності та резервного копіювання. Також, він підтримує шардування, яке дозволяє горизонтально розподіляти дані між декількома серверами для забезпечення масштабованості.

– Повний текстовий пошук: MySQL надає можливості для повнотекстового пошуку, що дозволяє виконувати швидкі та ефективні пошукові запити на основі повнотекстових індексів. Це особливо корисно для додатків, які вимагають пошуку по ключовим словам або фразам.

– Хмарні сервіси та інтеграція: MySQL працює в екосистемі хмарних сервісів, таких як Amazon Web Services (AWS), Microsoft Azure, Google Cloud і багато інших. Це дозволяє розробникам розгортати та масштабувати свої бази даних на хмарних платформах зі забезпеченням високої доступності та масштабованості.

– Підтримка тригерів та хранилищ: MySQL підтримує тригери, які дозволяють автоматично виконувати дії на базі даних при виконанні певних подій. Також, він має можливість визначати та використовувати хранилища (stored procedures) - набори SQL-коду, що можуть бути викликані з програми або запиту до бази даних.

MySQL є відкритим програмним забезпеченням, що означає, що ви можете звільнено використовувати його, модифікувати та розповсюджувати відповідно до умов ліцензії. Він активно підтримується та оновлюється спільнотою розробників, що робить його стабільним та надійним рішенням для багатьох проектів.

2.1.5 Eloquent ORM

Eloquent ORM (Object-Relational Mapping)[3] - це один з ключових компонентів фреймворка Laravel, який надає зручний спосіб роботи з базою даних. Він дозволяє виконувати операції з базою даних, такі як вибірка, вставка, оновлення та видалення записів, використовуючи об'єктно-орієнтований підхід. Eloquent ORM спрощує взаємодію з базою даних і дозволяє вам працювати з даними у зручний та елегантний спосіб.

Основні компоненти Eloquent ORM:

1. Моделі (Models): Моделі в Laravel - це класи, які представляють таблиці в базі даних. Кожна модель відповідає одній таблиці і містить властивості та методи для роботи з цими даними. Моделі використовуються для вибірки, вставки, оновлення та видалення даних з бази даних.

2. Запити (Queries): Eloquent дозволяє виконувати потужні запити до бази даних, використовуючи ланцюжки методів (method chaining). Ви можете легко виконувати запити на вибірку даних, додавати умови, сортувати, групувати та обмежувати результати запиту.

3. Відносини (Relationships): Eloquent надає зручні механізми для визначення та роботи з відносинами між моделями. Ви можете визначати взаємозв'язки типу один до одного, один до багатьох та багато до багатьох. Це дозволяє легко отримувати пов'язані дані без необхідності вручну писати складні SQL-запити.

4. М'який видалення (Soft Deleting): Eloquent підтримує м'яке видалення записів, що означає, що записи не фізично видаляються з бази даних, а лише помічаються як видалені. Це дозволяє відновлювати видалені записи та зберігати історію дій.

5. Масове заповнення (Mass Assignment): Eloquent надає можливість масового заповнення властивостей моделі з масиву даних. Це дозволяє швидко створювати або оновлювати записи, передаючи дані в одному масиві.

6. Мутатори та аксесори (Mutators and Accessors): Eloquent дозволяє визначати мутатори та аксесори для властивостей моделі. Мутатори дозволяють автоматично змінювати значення властивості перед збереженням, а аксесори - змінювати значення під час отримання.

7. Кешування запитів (Query Caching): Eloquent підтримує кешування запитів, що дозволяє зберігати результати запитів у кеші, зменшуючи навантаження на базу даних та прискорюючи виконання запитів.

Eloquent ORM забезпечує зручний та ефективний спосіб роботи з базою даних у Laravel. Він полегшує розробку та підтримку веб-додатків, дозволяючи розробникам працювати на вищому рівні абстракції та сконцентруватися на бізнес-логіці своїх додатків.

Додаткові характеристики про Eloquent ORM:

– Міграції (Migrations): Laravel має вбудований механізм міграцій, який спрощує керування структурою бази даних. Ви можете визначати міграції для створення, зміни або видалення таблиць та їхніх колонок. Eloquent інтегрується з міграціями, дозволяючи автоматично створювати моделі, відповідні до таблиць, що значно спрощує роботу з базою даних.

– Запити з об'єктами (Query Builder): Крім використання Eloquent моделей, Laravel також надає запити з об'єктами (Query Builder), що дозволяють складати запити до бази даних за допомогою методів. Це дозволяє писати складні запити з використанням ланцюжків методів, які автоматично генерують SQL-запити.

– Події (Events) та Сповіщення (Notifications): Eloquent підтримує систему подій та сповіщень, що дозволяє визначати та обробляти події, пов'язані з моделями. Ви можете визначати події, які відбуваються перед або після збереження, оновлення або видалення моделей. Сповіщення дозволяють надсилати повідомлення або повідомлення електронною поштою на основі подій, пов'язаних з моделями.

– Колекції (Collections): Eloquent повертає результати запитів у вигляді колекцій, що розширюють функціональність масивів. Ви можете використовувати методи колекцій для фільтрації, сортування, групування та інших операцій над результатами запитів.

– Локальні області (Local Scopes): Ви можете визначати локальні області для моделей, що дозволяють вам зручно виконувати повторювані запити з певними умовами. Локальні області дозволяють групувати часто використовувані умови в одному методі, що полегшує читабельність та підтримку коду.

Eloquent ORM володіє багатьма іншими корисними функціями та можливостями, що роблять роботу з базою даних в Laravel простою та зручною. Він забезпечує потужність та гнучкість при роботі з даними, дозволяючи розробникам швидко та ефективно створювати складні веб-додатки.

2.1.6 PHP 8

PHP (Hypertext Preprocessor) [4] - це мова програмування загального призначення, яка часто використовується для розробки веб-додатків. PHP дозволяє створювати динамічні та інтерактивні веб-сторінки, обробляти форми, доступатися до баз даних, керувати сесіями та багато іншого. PHP є однією з

найпопулярніших мов програмування для веб-розробки, і вона має велику активну спільноту розробників.

Основні особливості PHP:

1. Синтаксис: Синтаксис PHP схожий на синтаксис C і інших мов програмування з подібними конструкціями. PHP використовує `;` як роздільник кінця кожного оператора.

2. Вбудована підтримка HTML: PHP розроблена для вбудованої роботи з HTML. Ви можете включати HTML-код безпосередньо в PHP-скриптах і використовувати спеціальні теги `<?php ?>` для вставки PHP-коду в HTML-сторінку.

3. Розширені можливості: PHP має багатий набір вбудованих функцій і розширень, що дозволяють виконувати різноманітні завдання, такі як робота з базами даних, робота з файлами, маніпулювання рядками, робота з мережами та багато іншого.

4. Підтримка різноманітних баз даних: PHP підтримує багато різних систем управління базами даних, таких як MySQL, PostgreSQL, SQLite, Oracle і багато інших. Це дає можливість легко підключатися до баз даних і виконувати операції з ними.

5. Можливості роботи з файлами: PHP дозволяє взаємодіяти з файловою системою сервера, створювати, читати, записувати і видаляти файли та каталоги, а також виконувати операції з файловими шляхами.

6. Обробка форм: PHP має вбудовану підтримку для обробки форм, отримання даних з полів форми та їх обробку. Ви можете перевіряти дані, фільтрувати їх та зберігати в базі даних або використовувати для інших операцій.

7. Сесії та кукі: PHP дозволяє працювати з сесіями і кукі для збереження стану користувача між різними запитами. Це дозволяє зберігати дані про користувача, такі як ідентифікатор сесії або вибрані налаштування, і використовувати їх на протязі відвідування сайту.

8. Об'єктно-орієнтоване програмування: PHP підтримує об'єктно-орієнтоване програмування, що дозволяє створювати класи, об'єкти, наслідування, інтерфейси та інші об'єктно-орієнтовані конструкції.

PHP 8 є останньою стабільною версією PHP, яка була випущена у листопаді 2020 року. Вона включає ряд значних покращень та нових функціональних можливостей:

1. Вдосконалення продуктивності: PHP 8 має вдосконалену продуктивність та оптимізацію в порівнянні з попередніми версіями. Впроваджені оптимізації дозволяють покращити швидкодію виконання коду і зменшити споживання пам'яті.

2. Оновлення синтаксису: PHP 8 включає нові синтаксичні можливості, такі як оператори null coalescing (??=) і spaceship (<=>), що полегшують написання коду та роблять його більш зрозумілим.

3. Статична типізація: PHP 8 підтримує статичну типізацію, що дозволяє вказувати типи аргументів функцій, типи повернення та типи змінних. Це допомагає зменшити кількість помилок під час виконання програми та полегшує розуміння коду.

4. Attributes: PHP 8 включає нову функціональність, відому як атрибути (attributes). Атрибути дозволяють встановлювати метадані до класів, методів, властивостей та інших елементів коду. Це може бути корисно для автоматичної генерації документації, створення анотацій або впровадження деяких механізмів на основі цих метаданих.

5. Match-вирази: PHP 8 додає новий тип виразу під назвою match. Match-вирази дозволяють виконувати кілька порівнянь одночасно та повертати результат в залежності від виконаної умови.

6. JIT (Just-In-Time) компіляція: PHP 8 має підтримку JIT-компіляції, що дозволяє покращити швидкодію деяких типів коду. JIT-компіляція дозволяє перетворити PHP-код у машинний код на льоту під час виконання програми, що поліпшує швидкодію та реактивність додатків.

PHP 8 пропонує численні поліпшення та нововведення, які покращують продуктивність, безпеку та розширюють можливості мови для розробників. Використання PHP 8 дозволяє отримати більше переваг та покращити якість та ефективність ваших проектів.

Окрім основних функцій та нововведень, PHP 8 має ще кілька важливих характеристик:

– Union Types: PHP 8 включає підтримку об'єднаних типів (union types), що дозволяють вказувати, що змінна може мати один з кількох можливих типів.

Наприклад, ви можете вказати, що параметр функції може бути або цілим числом, або рядком.

– Named Arguments: PHP 8 дозволяє передавати аргументи функції не за позицією, а за їхніми іменами. Це полегшує розуміння та читабельність коду, особливо коли функція має багато аргументів зі значеннями за замовчуванням.

– Constructor Property Promotion: PHP 8 додає новий синтаксис для визначення властивостей об'єктів в конструкторі класу. Тепер ви можете оголошувати та ініціалізувати властивості безпосередньо в параметрах конструктора, що спрощує створення об'єктів.

– Nullable Return Types: PHP 8 дозволяє вказувати, що функція може повертати значення або `null`. Це корисно, коли функція може не мати валідного результату, і ви хочете явно показати цю можливість.

– Throw Expressions: PHP 8 додає новий синтаксис для викидання виключень. Ви можете викидати виключення прямо виразом, що дозволяє скоротити код та полегшити використання виключень.

– Weak Maps: PHP 8 представляє нову структуру даних - Weak Maps, яка дозволяє зберігати об'єкти у вигляді ключів без утримання посилань на них. Це допомагає уникнути проблем з витіком пам'яті та полегшує управління об'єктами.

PHP 8 є значним оновленням мови з важливими функціональними змінами та покращеннями продуктивності. Він надає розробникам більше можливостей

для створення потужних, ефективних та безпечних веб-додатків.

2.2 Середовище розробки

2.2.1 PHPStorm

PHPStorm [5] - це інтегроване середовище розробки (IDE) для роботи з мовою програмування PHP. Розроблено компанією JetBrains, PHPStorm надає розширені можливості для розробки, налагодження та керування PHP-проєктами.

PHPStorm має потужний редактор коду з підсвіткою синтаксису, автодоповненням, рефакторингом та вбудованою підтримкою шаблонів коду. Він спрощує написання та редагування PHP-коду, забезпечуючи швидкість та точність.

Дане середовище розробки надає розширені засоби для налагодження PHP-коду. Він підтримує встановлення точок зупину, крокування в коді, спостереження за змінними та інші налагоджувальні можливості, що допомагають знайти та виправити помилки в програмі.

JetBrains IDE дозволяє легко керувати PHP-проєктами. Він надає можливість підключати та настроювати версійний контроль, використовувати інструменти для автоматичного завершення завдань, створювати конфігурації для різних середовищ розробки та багато іншого.

Окрім всього, PHPStorm підтримує різні фреймворки для тестування PHP-коду, такі як PHPUnit. Він надає зручний інтерфейс для написання, запуску та аналізу тестів, що допомагає забезпечити якість коду та виявити проблеми швидше.

Також PHPStorm інтегруватися з різними зовнішніми інструментами, такими як бази даних, системи збірки, засоби аналізу коду тощо. Це дозволяє зручно працювати з іншими компонентами проєкту та забезпечує зручну роботу з інструментарієм розробника.

Загалом, PHPStorm є потужним інструментом для розробки PHP-програм, який надає широкий набір функцій та можливостей для підвищення продуктивності та якості роботи розробників.

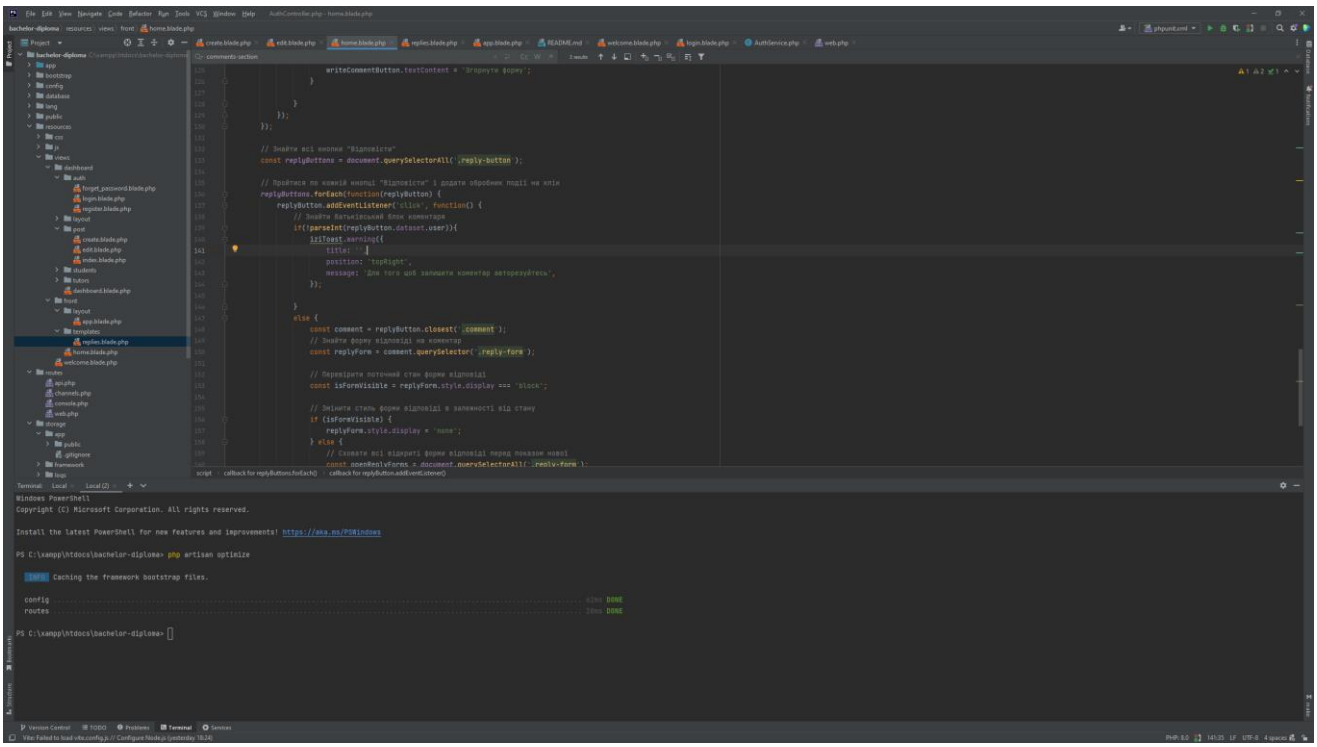


Рисунок 2.2.1.1 – Інтерфейс застосунку PHPStorm

2.3 Веб-ресурс

2.3.1 HTML

HTML (HyperText Markup Language)[6] є стандартною мовою розмітки для створення веб-сторінок. Використовуючи HTML, ви визначаєте структуру та семантику веб-сторінки, розташовуючи різні елементи на сторінці та надаючи їм значення.

Основні поняття та функції HTML:

1. Теги: HTML використовує теги для визначення різних елементів на сторінці. Теги складаються з відкриваючого `<tag>` та закриваючого `</tag>` тегів, які обрамлюють вміст елемента. Наприклад, `<p>` - тег абзацу, `<h1>` - тег заголовка першого рівня, `<a>` - тег посилання тощо.

2. Атрибути: Теги можуть мати атрибути, які надають додаткові властивості елементам. Атрибути розташовуються всередині відкриваючого тега і задаються у вигляді `ім'я="значення"`.

Наприклад, `<a href="https://example.com">` - атрибут `href` вказує адресу, до якої веде посилання.

3. Структура сторінки: HTML сторінки складаються з різних елементів, таких як `<head>`, `<body>`, `<header>`, `<footer>`, `<nav>` та інші, які визначають різні частини сторінки та їх значення. Наприклад, `<head>` містить мета-інформацію про сторінку, таку як заголовок, метатеги, посилання на зовнішні ресурси, а `<body>` містить основний вміст, який відображається на сторінці.

4. Списки: HTML має можливість використання списків для структурування вмісту. Існують два типи списків: впорядковані (`<ol>`) і невпорядковані (`<ul>`). Елементи списку визначаються за допомогою `<li>` тегу.

5. Таблиці: HTML дозволяє створювати таблиці за допомогою `<table>`, `<tr>` (рядок таблиці) та `<td>` (комірка таблиці) тегів. Це дозволяє організовувати дані у вигляді табличної структури.

6. Форми: HTML надає можливість створювати форми за допомогою `<form>` та різних елементів вводу, таких як `<input>`, `<select>`, `<textarea>`. Форми використовуються для збору даних від користувача.

7. Зображення: HTML дозволяє вставляти зображення на сторінку за допомогою `<img>` тега. Ви можете вказати шлях до зображення та додаткові атрибути, такі як `alt` (альтернативний текст для зображення) і `width`/`height` (ширина/висота зображення).

8. Посилання: HTML дозволяє створювати посилання на інші сторінки або документи за допомогою `<a>` тега. Ви можете вказати адресу посилання за допомогою атрибуту `href`.

Це лише кілька основних понять та функцій HTML. Існує багато інших можливостей, таких як робота з CSS для стилізації сторінок, використання

JavaScript для додавання інтерактивності, вбудовування відео та аудіо, робота з форматуванням тексту та багато іншого. HTML є основою для розробки веб-сторінок і відіграє важливу роль у створенні змісту та структури в Інтернеті.

Додатково, в HTML також існує кілька інших важливих можливостей та концепцій:

1. Вкладеність: Елементи HTML можуть бути вкладені один в одного, утворюючи ієрархічну структуру. Наприклад, можна вкласти елемент `<em>` (курсивний текст) всередину елемента `<p>` (абзац).

Це курсивний текст.

2. Коментарі: HTML дозволяє додавати коментарі у коді, які не відображатимуться на сторінці. Вони корисні для розуміння і позначення різних частин коду.

<!-- Це коментар -->

Це текст на сторінці.

3. Мета-теги: `<meta>` теги використовуються у розділі `<head>` для надання мета-інформації про сторінку. Наприклад, мета-тег `<meta charset="UTF-8">` визначає кодування символів сторінки.

4. Шаблони: HTML дозволяє використовувати шаблони для повторного використання коду. Ви можете створити окремий файл з фрагментами HTML та підключити їх до різних сторінок за допомогою тега `<include>` або шаблонізаторів, таких як Twig або Handlebars.

5. Валідація: Існують інструменти та служби, які допомагають валідувати HTML-код, перевіряючи його на відповідність стандартам і помилки. Це дозволяє забезпечити правильну структуру та форматування сторінки.

HTML є основою для будь-якого веб-документа і надає структуру та семантику вмісту. Важливо знати основні поняття та можливості HTML, а також дотримуватись найкращих практик для створення доступних, семантичних та добре структурованих веб-сторінок.

2.3.2 JavaScript

JavaScript[7] є однією з найпопулярніших мов програмування, використовуваних для розробки веб-додатків. Вона була створена для надання можливості динамічного взаємодії з веб-сторінками. Основна мета JavaScript полягає у покращенні користувацького досвіду, наданні можливості взаємодії з користувачем та динамічного змісту на веб-сторінках.

Основні риси JavaScript:

1. Клієнтський скрипт: JavaScript виконується безпосередньо в браузері користувача, що дозволяє створювати інтерактивні веб-сторінки, відповідаючи на події та змінюючи зміст сторінки під час її завантаження або після нього.

2. Синтаксис: Синтаксис JavaScript схожий на синтаксис інших мов програмування, таких як C, Java та C++. Це робить його досить легким для вивчення для розробників, які вже знайомі з цими мовами.

3. Взаємодія з HTML і CSS: JavaScript може змінювати зміст і стилі веб-сторінки, динамічно маніпулювати HTML-елементами та CSS-стилями, додавати нові елементи до сторінки або приховувати наявні.

4. Обробка подій: JavaScript може реагувати на події, такі як натискання кнопок, наведення курсора на елементи сторінки, завантаження сторінки тощо. За допомогою обробників подій, розробники можуть виконувати певні дії при виникненні певних подій.

5. Об'єктно-орієнтований підхід: JavaScript підтримує об'єктно-орієнтований підхід до програмування, що дозволяє створювати класи, об'єкти та використовувати наслідування.

6. Підтримка сторонніх бібліотекі: У JavaScript є велика кількість сторонніх бібліотек і фреймворків, таких як React, Angular і Vue.js, які полегшують розробку веб-додатків і роблять їх більш потужними та ефективними.

JavaScript застосовується не лише на клієнтській стороні веб-розробки, але й на серверній стороні. Наприклад, існують популярні серверні платформи, такі

як Node.js, які дозволяють розробникам використовувати JavaScript для написання серверних додатків.

За останні роки JavaScript став надзвичайно популярним і широко використовується веб-розробниками для створення різноманітних веб-додатків, включаючи односторінкові додатки, ігри, соціальні мережі та багато іншого. Його активно розвивають та оновлюють, додаючи нові функції і можливості, що робить його сучасним та потужним інструментом для веб-розробки.

JavaScript також має ряд додаткових особливостей:

1. Динамічна типізація: JavaScript використовує динамічну типізацію, що означає, що змінні можуть змінювати свій тип під час виконання програми. Наприклад, змінна, яка спочатку містить число, може пізніше містити рядок або об'єкт.

2. Асинхронне програмування: JavaScript підтримує асинхронний підхід до програмування, що дозволяє виконувати деякі операції паралельно і не блокувати виконання інших частин програми. Це особливо корисно для обробки мережевих запитів та взаємодії з сервером без зупинки виконання іншого коду.

3. Робота з DOM: JavaScript дозволяє взаємодіяти з Document Object Model (DOM), який представляє структуру HTML-сторінки. Ви можете змінювати вміст елементів, додавати або видаляти елементи, змінювати їх атрибути та стилі за допомогою JavaScript.

4. Можливості HTML5: JavaScript підтримує нові можливості, які надає стандарт HTML5, такі як геолокація, мультимедіа, локальне сховище (LocalStorage) та багато іншого. Ці можливості дозволяють створювати більш багатофункціональні та потужні веб-додатки.

5. Маніпулювання зображеннями: JavaScript має можливості маніпулювання зображеннями, такі як обрізка, масштабування, фільтри та інші операції. Це дозволяє створювати фоторедактори, галереї зображень та інші візуальні ефекти на веб-сторінках.

6. Розробка мобільних додатків: За допомогою фреймворків, таких як React Native і Ionic, JavaScript може використовуватись для розробки мобільних додатків для платформ Android та iOS. Це дозволяє розробникам використовувати свої знання JavaScript для побудови переносних мобільних додатків.

JavaScript є потужним і гнучким інструментом, який знаходить застосування в широкому спектрі веб-розробки. Завдяки своїм можливостям та великому екосистемі бібліотек і фреймворків, JavaScript залишається основним інструментом для створення інтерактивних та високоефективних веб-додатків.

2.3.3 jQuery JavaScript Framework

jQuery є популярною бібліотекою JavaScript, яка спрощує взаємодію з HTML-документами, обробку подій, анімацію, роботу з AJAX та інші завдання веб-розробки. Вона надає потужні функції та методи для легкого та ефективного маніпулювання DOM-деревом, що дозволяє зробити сторінки більш динамічними та взаємодійними.

Основні риси та можливості jQuery:

– Вибірка елементів: jQuery забезпечує простий та потужний синтаксис для вибору елементів з використанням CSS-подібного селектора. Наприклад, з допомогою `$()` можна вибрати всі елементи з певним класом, тегом або ідентифікатором.

– Маніпуляція DOM: jQuery дозволяє зручно маніпулювати DOM-деревом, змінюючи атрибути, стилі, текстовий вміст або додавати/видаляти елементи. Це дозволяє динамічно змінювати вигляд сторінки під час роботи на стороні клієнта.

– Обробка подій: З jQuery можна легко призначати обробники подій для елементів, таких як клік, наведення миші, введення тексту тощо. Це дозволяє реагувати на дії користувача та виконувати певні дії відповідно до цих подій.

– AJAX: jQuery має вбудовану підтримку AJAX, що спрощує взаємодію з сервером без перезавантаження сторінки. Вона надає методи для асинхронного

відправлення запитів на сервер, отримання та обробки відповідей, що дозволяє створювати більш інтерактивні веб-додатки.

– Анімація: jQuery має підтримку для анімації елементів, таких як зміна розміру, положення, прозорості тощо. Вона дозволяє створювати плавні переходи та анімаційні ефекти, що зроблюють сторінку більш привабливою та змістовною.

– Розширення: Існує велика кількість розширень (плагінів) для jQuery, які додають додаткові функціональні можливості до бібліотеки. Це дозволяє легко використовувати готові рішення для таких завдань, як каруселі, валідація форм, слайдери тощо.

jQuery є досить легкою для вивчення та використання бібліотекою, що робить її популярним вибором для багатьох веб-розробників. Однак, варто зазначити, що останні версії JavaScript (ECMAScript) також набули значних покращень у вбудованих функціях та API, що можуть замінити деякі функціональності, які надає jQuery.

2.3.4 Laravel Blade

Laravel Blade [8] є шаблонним механізмом у фреймворку Laravel, який дозволяє розробникам будувати розширювані та повторно використовувані шаблони для генерації HTML-сторінок. Blade надає зручний та елегантний спосіб для впровадження логіки в шаблони безпосередньо на рівні перегляду (view) у Laravel.

Особливості Laravel Blade:

1. Простота та зрозумілість: Синтаксис Blade простий та зрозумілий, що дозволяє легко створювати шаблони без необхідності вчитися новій мові. Він надає зрозумілі теги та директиви, які дозволяють впроваджувати логіку, цикли, умовні оператори та інші функції прямо у шаблоні.

2. Наслідування шаблонів: Blade підтримує концепцію наслідування

шаблонів, що дозволяє створювати основний шаблон, який визначає загальну структуру сторінки, і далі розширювати його у дочірніх шаблонах. Це дозволяє уникнути дублювання коду та забезпечує більшу організацію та повторне використання коду.

3. Умовні оператори та цикли: Blade дозволяє використовувати умовні оператори, такі як `if`, `else`, `elseif`, а також цикли, наприклад, `foreach` та `for`, безпосередньо в шаблонах. Це дозволяє генерувати динамічний вміст залежно від умов та даних.

4. Включення та компоненти: Blade підтримує можливість включати інші шаблони у шаблони. Це дозволяє розбити код на більш маленькі та універсальні компоненти, які можна повторно використовувати у різних частинах додатка. Включення дозволяє розділяти логіку та представлення на окремі частини.

5. Кешування шаблонів: Laravel Blade має вбудовану підтримку кешування шаблонів. Це означає, що шаблон може бути компільований в оптимізований PHP-код та збережений у кеші, що покращує швидкодію додатка.

6. Робота з формамиBlade надає зручні та експресивні способи роботи з формами. Він дозволяє створювати форми швидко та просто, автоматично генеруючи потрібні HTML-елементи та роблячи обробку даних з форми зручною та безпечною.

Laravel Blade є потужним інструментом для розробки шаблонів в Laravel. Він спрощує створення динамічних та гнучких HTML-сторінок, розбиваючи їх на частини, додаючи логіку та забезпечуючи повторне використання коду.

Окрім основних функцій, Laravel Blade має ще кілька корисних можливостей:

– Передача даних до шаблону: Blade дозволяє передавати дані з контролера до шаблону для їх відображення. Ви можете передати змінні, масиви або об'єкти і використовувати їх у шаблоні для генерації відповідного вмісту.

– Секції (Sections): Секції дозволяють визначати розділи коду в шаблоні, які можна заповнювати з інших частин додатку. Наприклад, ви можете визначити секцію для заголовка сторінки, а потім заповнити її з окремого

шаблону. Це дає змогу гнучко організовувати розмітку та структуру сторінки.

– Наслідування шаблонів з даними: Blade дозволяє передавати дані з батьківського шаблону до дочірніх шаблонів. Це дозволяє уникнути дублювання даних і забезпечити спадкування даних між шаблонами.

– Умовне відображення (Conditional Display): Blade має зручну синтаксичну конструкцію для умовного відображення вмісту на основі значень змінних або умовних виразів. Наприклад, ви можете використовувати `@if`, `@else`, `@elseif` для визначення умови відображення певного блоку коду.

– Форматування даних: Blade має вбудовані директиви для форматування даних. Ви можете використовувати `@datetime` для форматування дати та часу, `@number` для форматування чисел і `@currency` для форматування грошових значень.

Laravel Blade надає розробникам зручний та потужний інструмент для роботи з шаблонами у фреймворку Laravel. Він допомагає підтримувати чистий та організований код.

3 ОПИС ПРОГРАМНОЇ ЧАСТИНИ

3.1 Діаграма прецедентів

Діаграма прецедентів [9]– є одним із видів діаграм у мові моделювання UML (Unified Modeling Language). Вона використовується для візуального представлення функціональності системи з точки зору зовнішніх акторів (користувачів, систем або інших систем) і взаємодій між ними.

Основною метою діаграми прецедентів є опис сценаріїв взаємодії між акторами та системою. Вона дозволяє ідентифікувати основні функції, які повинна виконувати система, та уточнює взаємодію між користувачами та системою.

Основні елементи діаграми прецедентів:

– Актори. Представляють ролі або зовнішніх учасників, які взаємодіють з системою. Актори зображаються у вигляді піктограм або простих фігур, наприклад, людей або інших систем. Актор може бути користувачем, зовнішньою системою, або навіть іншою частиною самої системи.

– Прецеденти. Представляють функціональність або дії, які можуть бути виконані системою. Кожен прецедент описує конкретну дію або сценарій, який актор може запитати у системи. Прецеденти зображаються у вигляді овалів із назвою.

– Взаємодії. Відображають взаємодію між акторами та прецедентами. Вони показують, як актори використовують прецеденти для досягнення певних цілей. Взаємодії зображаються у вигляді стрілок, що виходять від акторів до прецедентів.

– Успадкування прецедентів. Використовується, коли один прецедент успадковує функціональність іншого прецеденту. Це вказує на те, що більш конкретний прецедент має всі функції батьківського прецеденту, а також додаткові.

– Розширення прецедентів. Використовується, коли один прецедент може розширити або модифікувати функціональність іншого прецеденту. Це вказує на

те, що деякі додаткові дії можуть бути виконані залежно від умов або варіацій сценарію.

Діаграма прецедентів дозволяє зосередитися на функціональних вимогах системи та взаємодії з її користувачами. Вона допомагає розуміти потреби акторів, ідентифікувати основні прецеденти та встановлювати основні взаємозв'язки між ними. Це цінний інструмент для аналізу вимог та проєктування системи перед її реалізацією.

Зв'язки між акторами та прецедентами на діаграмі прецедентів можуть бути наступними:

1. Виклик прецеденту («Uses» або «Include»): Актор викликає певний прецедент для виконання певної дії. Наприклад, актор «Клієнт» може викликати прецедент «Оформити замовлення» для здійснення покупки.

2. Участь актора («Extends»): Актор бере участь у виконанні певного прецеденту на додатковому рівні функціональності. Наприклад, прецедент «Редагувати профіль» може мати варіант, в якому актор «Адміністратор» бере участь для підтвердження змін.

3. Загальна взаємодія (зв'язок): Це зв'язок між актором та прецедентом, який показує, що актор може взаємодіяти з прецедентом. Наприклад, актор «Менеджер» може взаємодіяти з прецедентом «Генерувати звіт» для отримання необхідних звітів.

4. Залежність (пунктирний зв'язок): Це зв'язок, що показує, що один прецедент залежить від іншого прецеденту для свого виконання. Наприклад, прецедент «Оформити замовлення» може залежати від прецеденту «Авторизуватися» для перевірки користувача перед здійсненням покупки.

Діаграма прецедентів також може містити додаткові елементи, такі як системні границі, коментарі та акторів-спадкоємців, які розширюють функціональність базових акторів.

Надіюся, що ця додаткова інформація буде корисною для вас. Якщо у вас є ще якісь питання, будь ласка, не соромтеся запитати.

3.2 Діаграма класів системи

Діаграма класів [10]– є одним із видів структурних діаграм у мові моделювання UML (Unified Modeling Language). Вона використовується для візуального представлення класів, їх атрибутів, методів та взаємозв'язків між класами в системі.

Основною метою діаграми класів є моделювання структури системи з точки зору класів, які утворюють її основу. Діаграма надає зручний спосіб описати класи, їх властивості та поведінку, а також взаємодії між ними.

Основні елементи діаграми класів:

– Класи. Представляють основні елементи системи і зображуються у вигляді прямокутників з поділом на три розділи. Верхній розділ містить назву класу, середній розділ - атрибути класу (змінні, стани), а нижній розділ - методи класу (функції, операції).

– Атрибути. Представляють дані, що належать класу. Вони зображаються у середньому розділі прямокутника класу. Атрибути можуть мати ім'я, тип даних та значення за замовчуванням.

– Методи. Представляють функції або операції, які можуть бути виконані класом. Вони зображаються у нижньому розділі прямокутника класу. Методи можуть мати ім'я, параметри, типи повернутих значень та модифікатори доступу.

– Взаємозв'язки. Відображають взаємодію між класами і показують зв'язки, які існують між ними. Наприклад, асоціація, композиція, агрегація, спадкування тощо.

– Класи-специфікації. Використовуються для уточнення абстрактних класів і надають більш детальний опис їх властивостей і методів.

– Залежності. Відображають залежності між класами, коли зміна в одному класі може вплинути на інший, але не є прямим зв'язком взаємодії.

– Інтерфейси. Відображають контракти, які реалізуються класами. Інтерфейси визначають методи, які повинні бути реалізовані класами, що їх

реалізують.

Додатковою елементи щодо діаграми класів є такі аспекти:

– Стереотипи: Стереотипи використовуються для надання додаткової інформації про класи або взаємозв'язки. Наприклад, можуть бути використані стереотипи, що позначають ролі або шаблони проектування, такі як "контролер", "сутність" чи "фабрика".

– Коментарі: Коментарі можуть бути додані до діаграми класів, щоб надати додаткові пояснення або деталізацію. Вони можуть містити описи класів, їхніх взаємозв'язків, призначення та будь-яку іншу корисну інформацію.

– Залежності: Залежності показують взаємозв'язки між класами, коли один клас використовує або залежить від іншого, але не є прямим взаємозв'язком. Залежності можуть бути тимчасовими або контекстуальними і можуть відображати, наприклад, використання одного класу в методі іншого класу.

– Асоціації: Асоціації вказують на зв'язки між класами та можуть мати ролі, множинність та напрямок. Вони використовуються для показу взаємозв'язку між об'єктами різних класів і можуть мати атрибути або пов'язані операції.

– Композиція та агрегація: Композиція та агрегація є специфічними типами асоціацій, які показують, що один клас є складовою частиною іншого класу. Композиція показує сильну залежність, де класи не можуть існувати окремо, а агрегація показує більш слабку залежність, де класи можуть існувати незалежно один від одного.

– Інтерфейси: Інтерфейси використовуються для опису поведінки, яку класи мають реалізувати. Вони вказують на методи, які класи повинні мати, але не визначають реалізацію самого методу.

Діаграма класів може бути використана не тільки для аналізу та проектування систем, але і для генерації коду, визначення архітектури системи, виявлення залежностей та з'ясування взаємодій між класами.

Якщо у вас є більше питань або потреба у конкретних прикладах, будь ласка, повідомте мене.

Діаграма класів є потужним інструментом для аналізу, проєктування та розуміння архітектури системи. Вона дозволяє відобразити взаємозв'язки між класами і використовується як основа для подальшого розвитку програмного продукту.

Діаграма класів моделей для серверної частини системи наведена на рисунку 4.2.1.1.

На рисунку 4.2.2 зображено схему бази даних, вона відображає представлення даних з моделей у нашій СУБД.

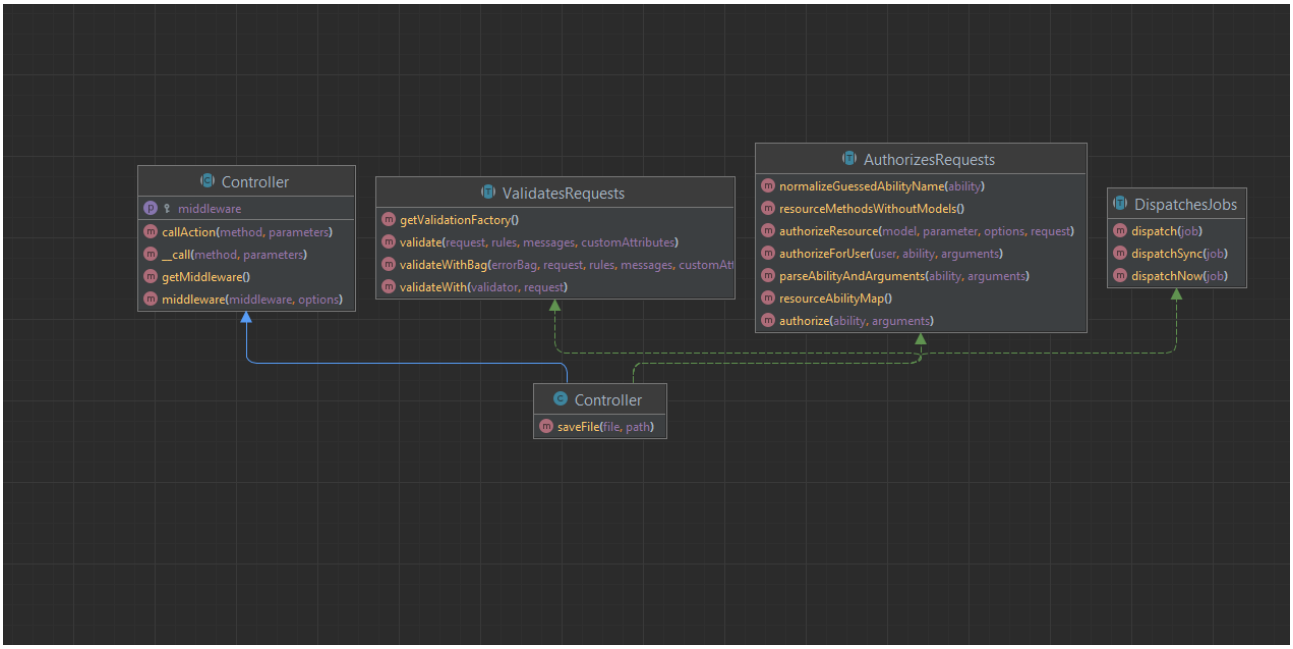


Рисунок 4.2.1.1 — Діаграма класів моделей

На рис. 4.2.2 наведено всі поля, присутні на кожному з класів моделей.

Діаграма схеми бази даних (Database Schema Diagram) відображає структуру бази даних, включаючи таблиці, колонки, взаємозв'язки та обмеження. Вона надає візуальне уявлення про логічну або фізичну організацію даних у базі даних. Діаграма схеми бази даних зазвичай використовується для моделювання та проектування бази даних, а також для спільного розуміння структури даних між розробниками, аналітиками та іншими зацікавленими сторонами.

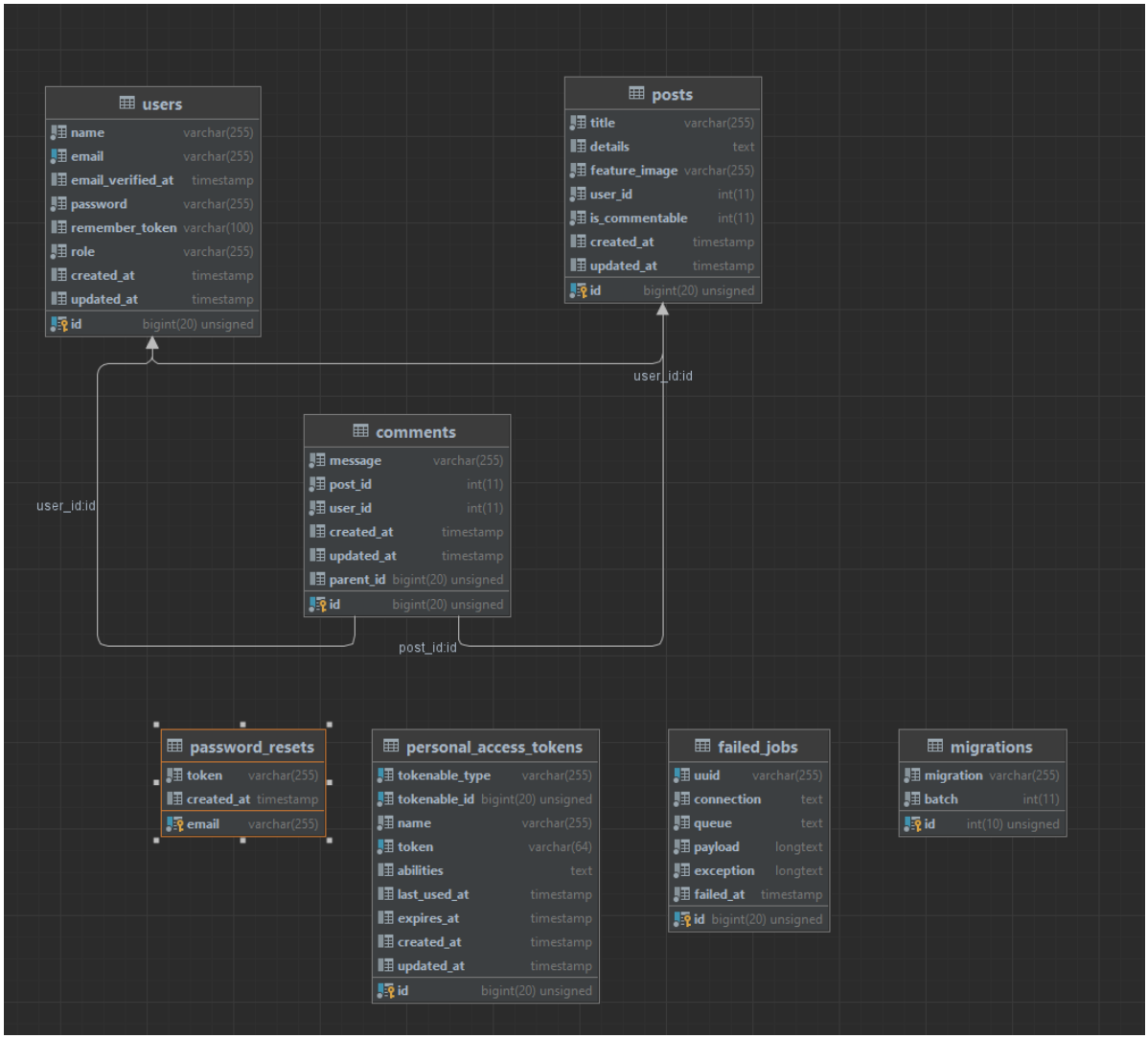


Рисунок 4.2.2 — Діаграма схеми бази даних

Наведені діаграми класів демонструють загальну структуру даних, як вони пов’язані між собою, інтерфейси взаємодії.

3.3 API

API (Application Programming Interface) у Laravel - це спосіб взаємодії з додатком Laravel зовнішніми системами, такими як мобільні додатки, веб-сервіси, клієнтські додатки та інші системи. Laravel надає потужні інструменти для розробки API, що дозволяють створювати надійні та масштабовані інтерфейси для обміну даними.

Основні можливості та функціонал, які надає Laravel для розробки API, включають:

1. Маршрутизація: Laravel має зручний механізм для визначення маршрутів API. Ви можете використовувати `Route::apiResource` або `Route::group` для організації маршрутів та визначення дій, які повинні виконуватися для кожного маршруту.

2. Контролери: Ви можете створити контролери для обробки різних запитів API. Контролери дозволяють вам групувати логіку, пов'язану з обробкою запитів, у методах контролера.

3. Серіалізація даних: Laravel надає різні зручні способи для серіалізації даних в формати, такі як JSON або XML. Ви можете використовувати різні класи серіалізаторів, такі як `Fractal`, для структурування та форматування вихідних даних.

4. Аутентифікація та авторизація: Laravel надає вбудовану підтримку для аутентифікації та авторизації API. Ви можете використовувати маркери доступу (access tokens), JWT (JSON Web Tokens) або інші методи аутентифікації для захисту своїх API-шляхів.

5. Перевірка валідності даних: Laravel має потужну систему валідації, яка дозволяє перевіряти вхідні дані запитів API на коректність та валідність. Ви можете використовувати правила валідації, які надаються Laravel, або створювати власні правила валідації.

6. Обробка помилок: Laravel має вбудований механізм обробки помилок, що дозволяє зручно відображати та обробляти помилки API. Ви можете налаштувати власні обробники помилок та повертати зрозумілі відповіді з відповідними HTTP-статусами.

7. Тестування: Laravel надає засоби для написання автоматизованих тестів для вашого API. Ви можете перевіряти правильність роботи маршрутів, контролерів та інших компонентів API за допомогою вбудованих функцій тестування.

Laravel забезпечує зручний та потужний фреймворк для розробки API. Він спрощує процес розробки, забезпечує безпеку, масштабованість та широкі можливості взаємодії з додатками та сервісами.

Виділяють такі запити:

– GET: для запиту ресурсу.

– POST: для створення нового ресурсу.

– PUT: для оновлення існуючого ресурсу.

– DELETE: для видалення існуючого ресурсу. Розглянемо маршрутизацію запитів у нашому застосунку.

Розглянемо маршрутизацію запитів у нашому застосунку.

```
/* Tutors routes */
Route::get( uri: '/tutors/all', action: 'UserController@getTutors')->name( name: 'admin_tutors');
Route::get( uri: '/tutor/create', action: 'UserController@createTutor')->name( name: 'admin_tutor_create');
Route::post( uri: '/tutor/store', action: 'UserController@storeTutor')->name( name: 'admin_tutor_store');
Route::get( uri: '/tutor/delete/{id}', action: 'UserController@deleteTutor')->name( name: 'admin_tutor_delete');
```

Рисунок 4.3.1 — API методи для класу "User" для ролі викладач

```
/* Students routes */
Route::get( uri: '/students/all', action: 'UserController@getStudents')->name( name: 'admin_students');
Route::get( uri: '/student/delete/{id}', action: 'UserController@deleteStudent')->name( name: 'admin_student_delete');
```

Рисунок 4.3.2 — API методи для класу "User" для ролі студент

```
/* Posts routes */
Route::get( uri: '/posts/all', action: 'PostController@index')->name( name: 'admin_posts');
Route::get( uri: '/post/create', action: 'PostController@create')->name( name: 'admin_post_create');
Route::post( uri: '/post/store', action: 'PostController@store')->name( name: 'admin_post_store');
Route::get( uri: '/post/delete/{id}', action: 'PostController@delete')->name( name: 'admin_post_delete');
Route::get( uri: '/post/edit/{id}', action: 'PostController@edit')->name( name: 'admin_post_edit');
Route::post( uri: '/post/update/{id}', action: 'PostController@update')->name( name: 'admin_post_update');
```

Рисунок 4.3.3 — API методи для класу "Пост"

```
Route::group(['namespace' => 'App\Http\Controllers\Dashboard\Auth'], function () {
    Route::get( uri: '/register', action: 'AuthController@register')->name( name: 'admin_register');
    Route::post( uri: '/register/submit', action: 'AuthController@registerCreate')->name( name: 'admin_register_create');
    Route::get( uri: '/login', action: 'AuthController@login')->name( name: 'admin_login');
    Route::get( uri: '/logout', action: 'AuthController@logout')->name( name: 'admin_logout');
    Route::post( uri: '/login/submit', action: 'AuthController@login_submit')->name( name: 'admin_login_submit');

    Route::get( uri: '/forget-password', action: 'AuthCOntroller@forget_password')->name( name: 'admin_forget_password');
    Route::post( uri: '/forget-password/submit', action: 'AuthController@forget_password_submit')->name( name: 'admin_forget_password_submit');
});
```

Рисунок 4.3.4 — API методи реєстрації та авторизації

```
Route::group(['namespace' => 'App\Http\Controllers\Front'], function () {
    Route::get( uri: '/', action: 'HomeController@index')->name( name: 'home');

    Route::post( uri: '/comment/store', action: 'CommentController@store')->name( name: 'comment_store');
    Route::post( uri: '/reply-comment/store', action: 'CommentController@replyStore')->name( name: 'comment_reply_store');
    Route::get( uri: '/comment/delete/{id}', action: 'CommentController@delete')->name( name: 'comment_delete');
});
```

Рисунок 4.3.5 — API методи фронтенд частини

Кожен з наведених вище API методів визначає обробник виклику HTTP запиту.

На рисунку 4.3.3 зображені API методи для класу "Post", розглянемо синтаксис запису:

Route::get('/posts/all', 'PostController@index')->name('admin_posts');

Route::{вид HTTP запиту}(‘{суфікс посилання, за яким здійснюється запит}/{id відповідного запису з бази даних}’, ‘{назва контролеру, який буде обробляти запит}{метод з контролеру}’]);

Далі розглянемо структуру контролера PostController:

– Функція index — повертає шаблон Blade та передає змінну у вигляді масиву. Ця змінна дістається з допоміжного класу PostService. У методах цього класу виконуються усі «брудні» операції. Також клас PostService має допоміжний контролер PostRepository, який має прямий контакт з класами моделями. У класі PostRepository виконують запроси до БД за допомогою ORM Eloquent. На даний момент, нас цікавить метод get(), який повертає усі пости викладача або студента, у випадку адміна – метод повертає усі пости на сайті

```
public function index()
{
    $posts = [];
    if(Auth::user()->role == 'admin'){
        $posts = $this->postService->get();
    }
    else{
        $posts = Auth::user()->posts;
    }
    return view( view: 'dashboard.post.index', [
        'posts' => $posts,
    ]);
}

public function create()
{
    return view( view: 'dashboard.post.create');
}

public function store(PostRequest $request)
{

    $postArray = $request->except( keys: '_token');
    $postArray['user_id'] = Auth::user()->id;
    if (isset($request->feature_image)) {
        $postArray['feature_image'] = self::saveFile($request->feature_image, path: PostEnum::POST_FEATURE_IMAGE_PATH);
    }

    $result = $this->postService->store($postArray);

    return $result ? redirect()->route( route: 'admin_posts')->with('success', 'Post successfully added') : back()->with('error', 'Error db connection');
}
```

Рисунок 4.3.6 — Методи класу PostController

– Функція create – повертає Blade шаблон для відображення HTML елементів

– Функція store – отримує POST об’єкт, попередньо оброблений класом PostRequest(це прототип класу Validator для постів). Перевіряє чи було додана фотографія поста, якщо так, то проводить додавання файлу у папку storage/assets/img/posts. Далі передає змінну у вигляді масиву до класу PostService, який в свою чергу проводить перевірку інформацію і викликає метод create() класу PostRepository, який додає інформацію у базу даних.

– Функція edit – отримує ID посту, який юзер хоче редагувати. Викликає метод getById() класу PostService, який за допомогою методу findOrFail() класу PostRepository, отримує об’єкт посту або повертає false, якщо такого запису немає в БД. Кінцевим пунктом є повернення шаблону Blade з об’єктом поста.

– Функція update – отримує POST об’єкт, попередньо оброблений класом PostRequest(це прототип класу Validator для постів). Перевіряє чи було додана фотографія поста, якщо так, то проводить додавання файлу у папку storage/assets/img/posts. Далі передає змінну у вигляді масиву до класу PostService, який в свою чергу проводить перевірку інформацію, і якщо бачить ,що є нова фотографія, то видаляє стару. Викликає метод update() класу PostRepository, який обновляє інформацію у базу даних.

– Функція delete – отримує ID посту, який юзер хоче видалити. Викликає метод delete() класу PostService, який за допомогою методу findOrFail() класу PostRepository, отримує об’єкт посту або повертає false, якщо такого запису немає в БД. Якщо такий запис є в БД, то проходить перевірка фотографії поста, якщо перевірка завершилася успішно видаляє фотографію. Кінцевим пунктом є виклик методу delete() класу PostRepository, який видаляє інформацію з бази даних.

Інші контролери (AuthContoller, UserController, CommentController, HomeController) працюють аналогічно до PostController.

3.4 Вебсайт

Для розробки клієнтського інтерфейсу було використано двигун шаблонів

Blade:

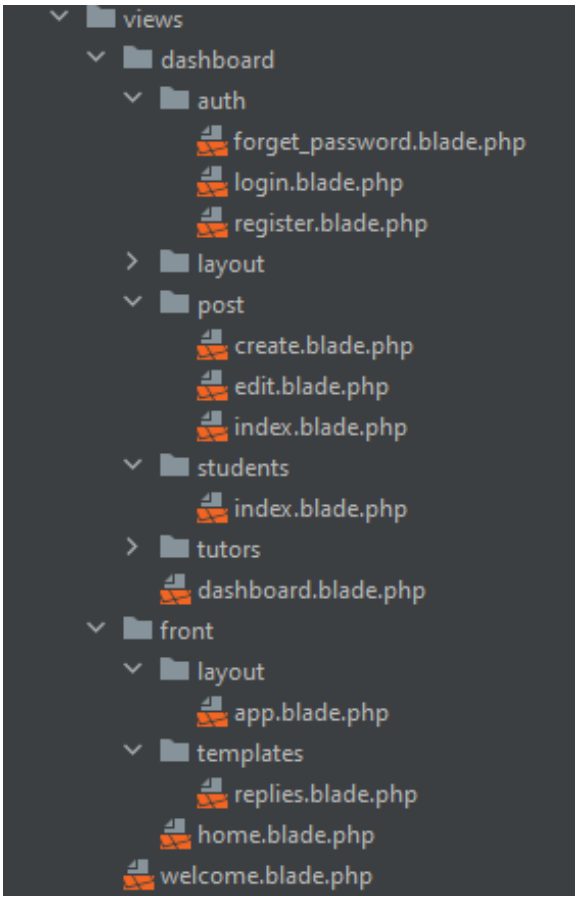


Рисунок 4.4.1 — файли двигуна Blade

Розглянемо внутрішню структуру одного з шаблонів на прикладі home.blade.php. Це шаблон UI(User Inteface) – те що бачить юзер, зайшовши на сайт.

– Синтаксис. Blade має зрозумілий та інтуїтивний синтаксис, що дозволяє розробникам легко вбудовувати PHP-код у шаблони. Наприклад, використання подвійних фігурних дужок **{{ }}** для відображення змінних або результатів виразів.

– Наслідування шаблонів. Blade підтримує механізм наслідування шаблонів, що дозволяє створювати основний шаблон зі спільною структурою та вбудовувати в нього окремі сторінки зі змінним вмістом. Це спрощує управління макетом сторінок та полегшує повторне використання коду.

– Секції. Blade дозволяє визначати секції (наприклад, блоки контенту) в шаблонах, які можуть бути перевизначені в конкретних сторінках, що

наслідують шаблон. Це дає можливість динамічно вставляти вміст у специфічні частини шаблону.

– Включення шаблонів. Blade підтримує включення інших шаблонів, що дозволяє розробникам розділити шаблони на більш малі і циклічні компоненти. Це забезпечує легку модульність та підтримку компонентного підходу до розробки.

– Керування потоком. Blade надає умовні оператори, цикли та інші засоби керування потоком, що дозволяють розробникам динамічно змінювати вміст сторінок залежно від умов.

– Кешування шаблонів. Laravel має можливість кешувати Blade-шаблони, що поліпшує продуктивність шляхом збереження компіляції шаблонів та уникнення повторного компілювання при кожному запиті.

4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Розглянемо сценарій роботи з системою "адміна". Акаунт адміна додає розробник, тому що адмін сайту може бути тільки один, і немає потреби для реєстрації багато адмін акаунтів. Далі розробник передає людині доступи. Стандартний процес входу в акаунт (рис. 5.1)

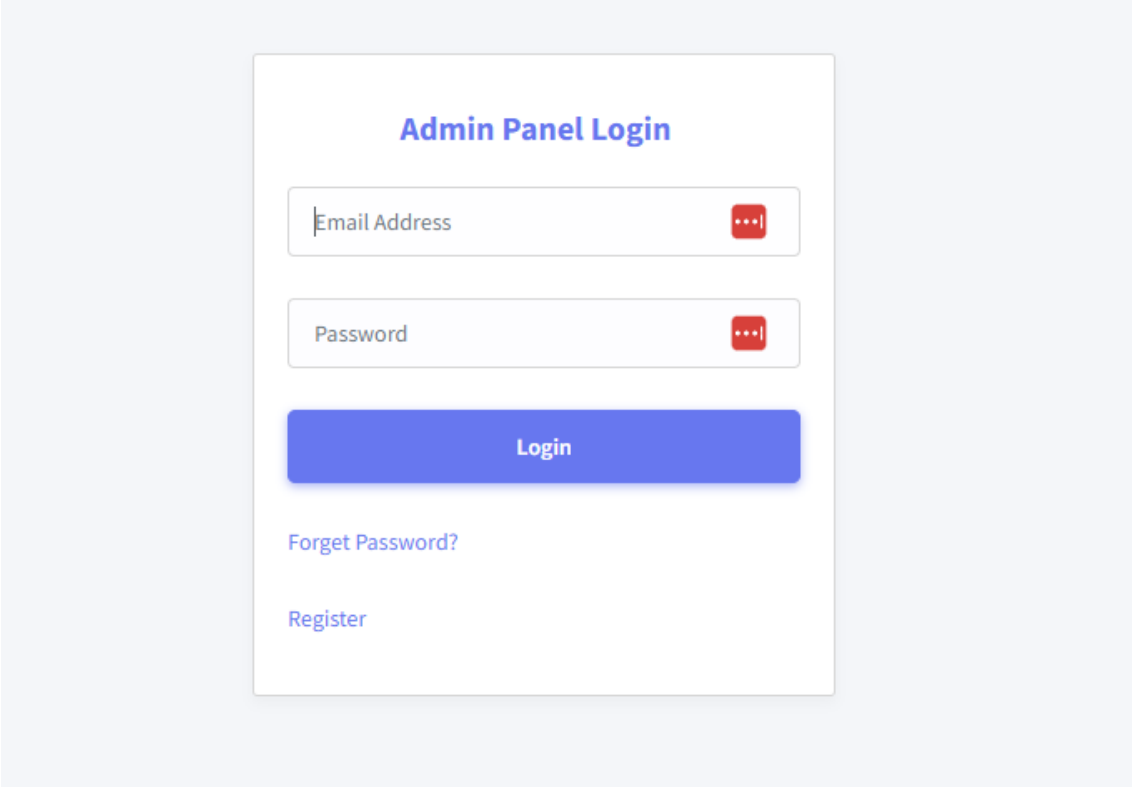


Рисунок 5.1 – Вхід в особистий кабінет

Після успішного входу в акаунт, юзер потрапляє адмін панелі (рис. 5.2)

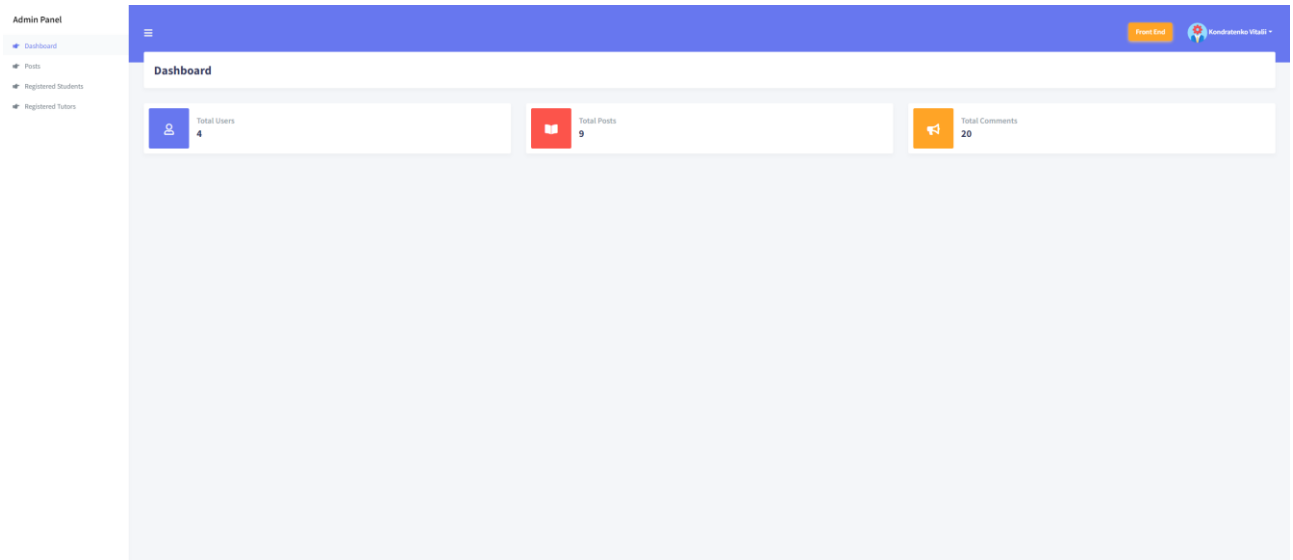


Рисунок 5.2 – особистий кабінет

В особистому кабінеті адмін може виконувати наступні кроки:

– Бачити пости усіх користувачів, редагувати чи видаляти будь-який пост, та додавати свої пости(рис. 5.3)

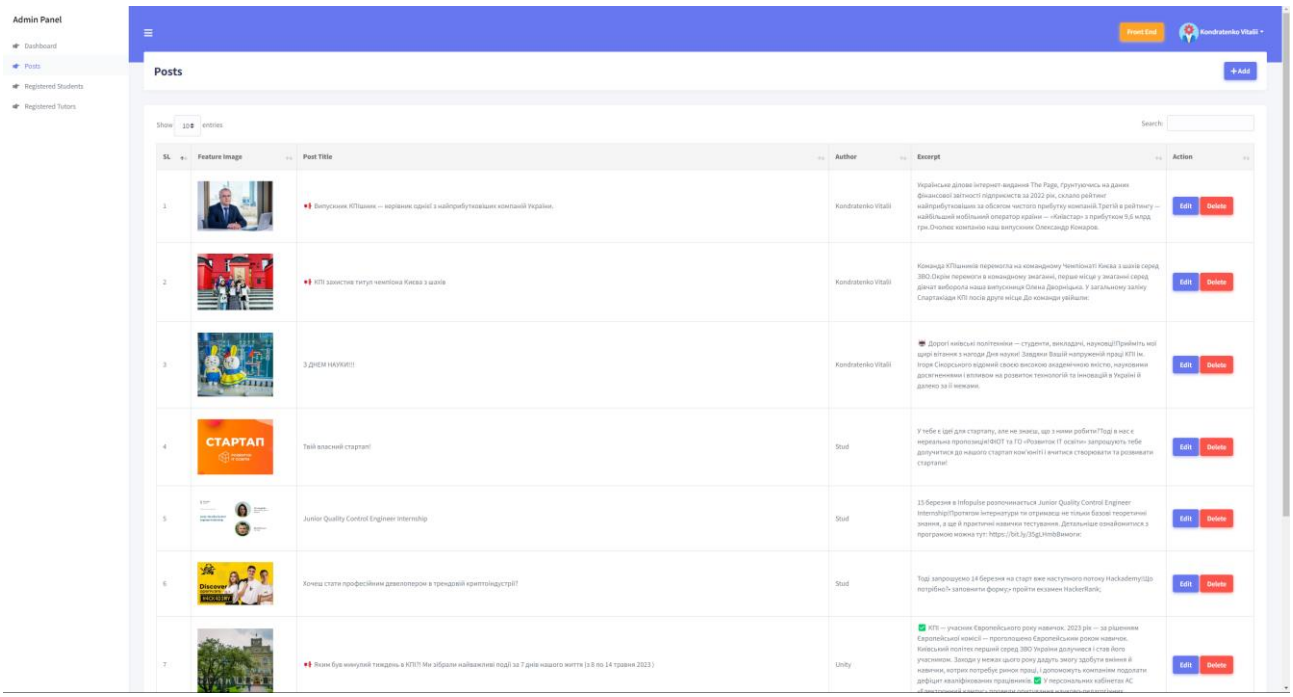


Рисунок 5.3 – Сторінка маніпуляції постами

– Наступним пунктом є сторінка усіх студентів зареєстрованих на цьому ресурсі. Також адмін має право на видалення акаунту будь-якого студента(рис. 5.4)

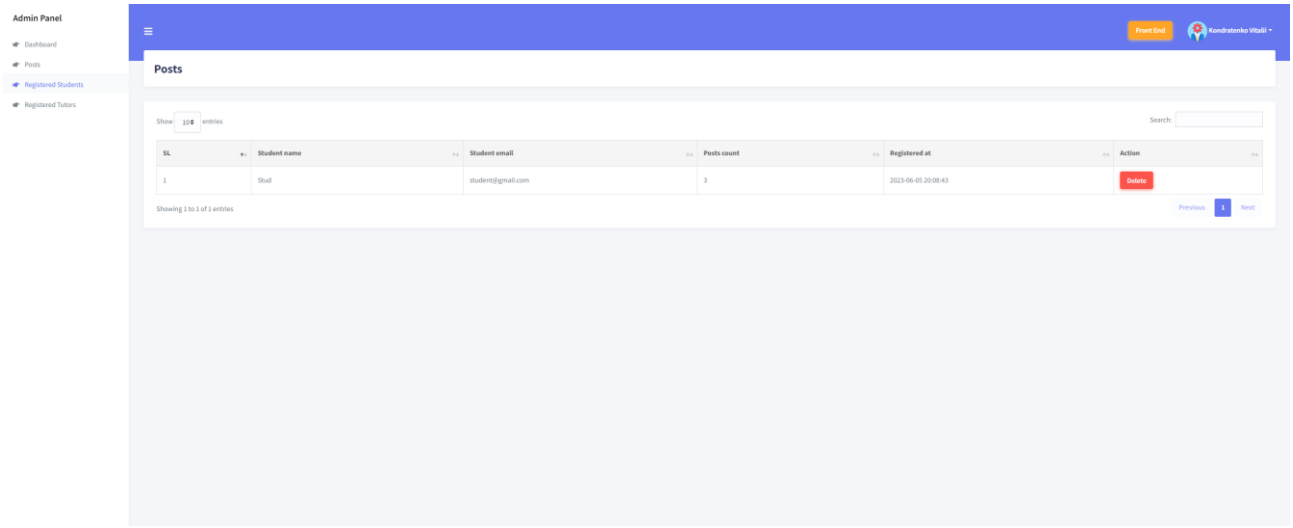


Рисунок 5.4 – Сторінка маніпуляції акаунтами студентів

– Основною, на мою думку, є наступна функція – сторінка додавання та видалення особистих кабінетів викладачів. Тут адмін сайту може створювати

кабінет викладача та передавати доступи безпосередньо викладачу, або адмін може видалити непотрібний акаунт викладача(рис. 5.5, 5.6)

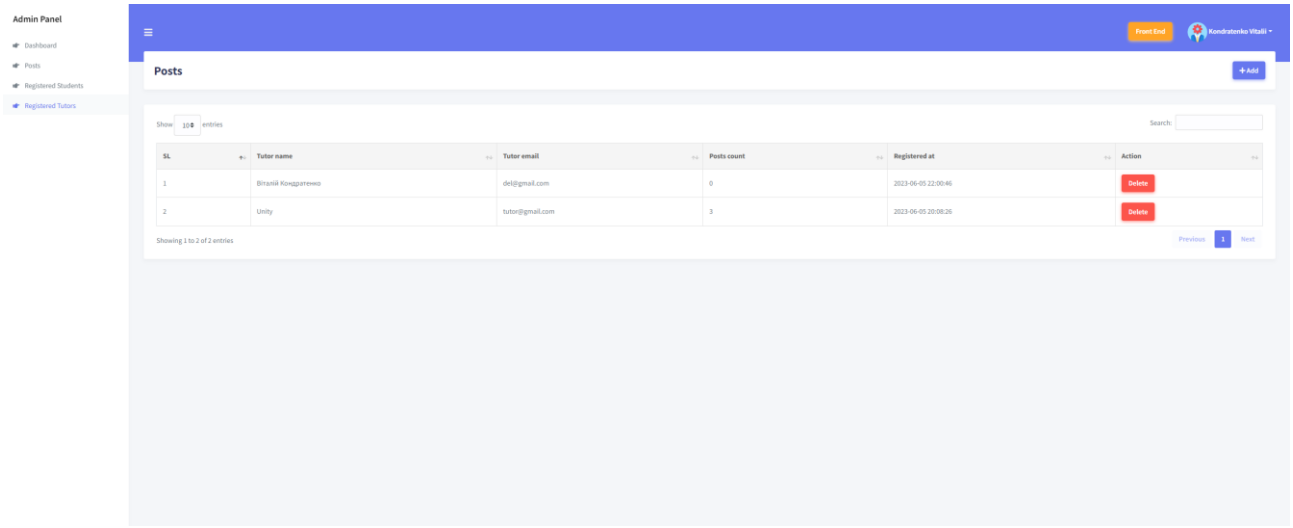


Рисунок 5.5 – Сторінка маніпуляції акаунтами викладачів

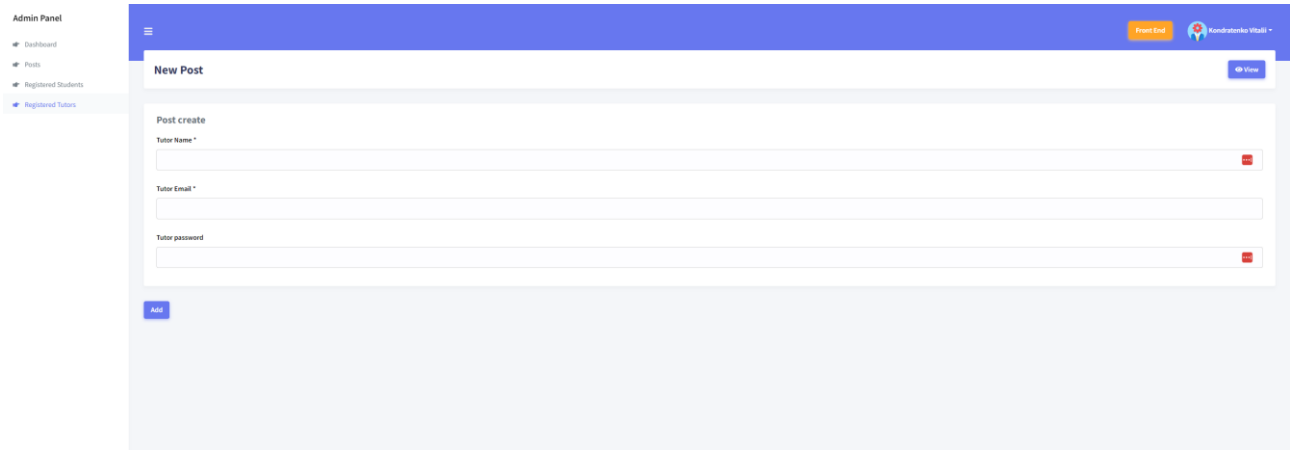


Рисунок 5.6 – Сторінка маніпуляції акаунтами викладачів

Наступним етапом розглянемо фронтенд даної системи. Тут на разі є сторінка постів з header, в якому розташовані: назва вед-додатку, кнопка реєстрації та входу в особистий кабінет.(рис.5.7)

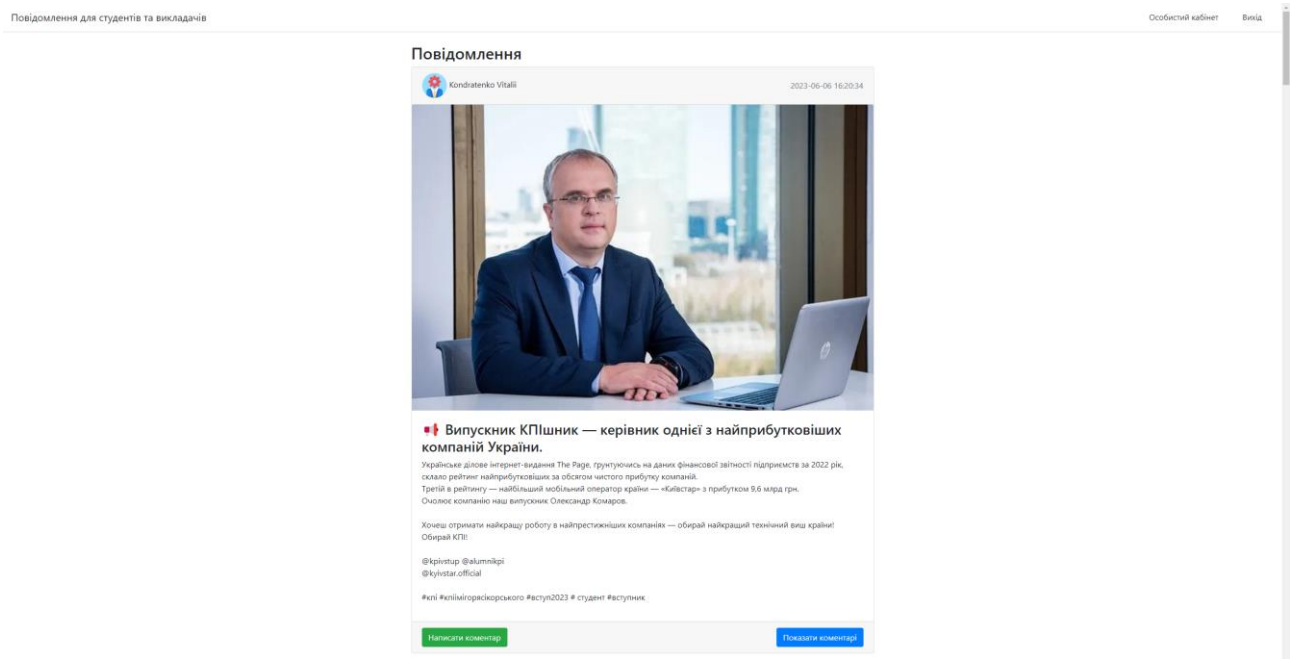


Рисунок 5.7 – Фронтед веб-ресурсу

Також розглянемо систему додавання коментарів під кожен пост. Зокрема, адмін може додавати коментарі, додавати відповіді на коментарі, та видаляти будь-який коментар. В свою чергу, викладач може видаляти також будь-який коментар, та писати їх. А студент може видалити тільки свій коментар, та додавати чи відповідати на коментарі. (рис.5.8)

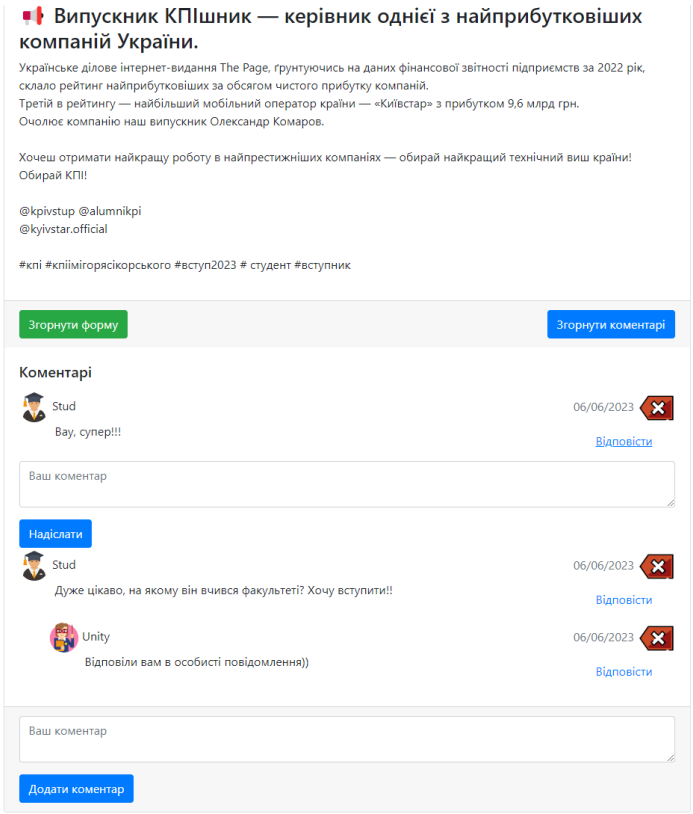


Рисунок 5.8 – Система коментування поста

ВИСНОВКИ

В рамках даної дипломної роботи було розроблено систему інформування викладачів та студентів у реальному часі з використанням програмного фреймворка Laravel. Висновки, отримані в результаті дослідження та розробки, можуть бути наступними:

1. Ефективність системи інформування: Розроблена система забезпечує швидку та ефективну комунікацію між викладачами та студентами. Завдяки використанню реального часу, інформація передається миттєво, що сприяє оперативному реагуванню на події та вирішенню проблем.

2. Зручність та доступність: Розроблена система базується на веб-технологіях, що робить її доступною з будь-якого пристрою з підключенням до Інтернету. Користувачі можуть отримувати повідомлення, оновлення та іншу інформацію в режимі реального часу без необхідності встановлення додаткових програм.

3. Покращення комунікації: Система інформування дозволяє викладачам та студентам зберігати постійний зв'язок, обмінюватися документами, відправляти повідомлення та отримувати оновлення щодо навчального процесу. Це сприяє поліпшенню комунікації та співпраці між всіма учасниками освітнього процесу.

4. Застосування Laravel: Використання фреймворка Laravel дозволяє розробникам швидко і ефективно створити систему інформування. Laravel забезпечує широкий функціонал та зручний інтерфейс програмування додатків (API), що сприяє швидкому розгортанню та масштабуванню системи.

5. Перспективи розвитку: Розроблена система інформування може постійно вдосконалюватися та розширюватися, залежно від потреб викладачів та студентів. Наприклад, можна додати нові функції, які допоможуть в організації та плануванні навчального процесу, а також інтегрувати систему з іншими освітніми платформами.

У загальному, розробка системи інформування викладачів та студентів у реальному часі на основі програмного фреймворка Laravel є актуальним та перспективним напрямом. Така система сприяє покращенню комунікації, ефективності та доступності освітнього процесу, сприяючи підвищенню якості навчання та співпраці між викладачами та студентами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Laravel [Електронний ресурс] — Режим доступу: https://laravel.com/docs

2. MySQL [Електронний ресурс] — Режим доступу: http://www.mysql.ru/docs/man/

3. Eloquent ORM [Електронний ресурс] — Режим доступу: https://laravel.com/

4. PHP 8 [Електронний ресурс] – Режим доступу: https://www.php.net/releases/8.0/ru.php

5.PHPStorm [Електронний ресурс] – Режим доступу: https://itpro.ua/product/jetbrains

6.HTML [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/ru/docs

7.JavaScript [Електронний ресурс] – Режим доступу: https://learn.javascript.ru/intro

8.Laravel Blade [Електронний ресурс] – Режим доступу: https://laravel.com/docs/10.x/blade#main-content

9.Діаграми прецедентів [Електронний ресурс] – Режим доступу: https://evergreens.com.ua/ru/articles

10.Діаграми класів [Електронний ресурс] – Режим доступу: https://www.ua5.org/oop/392-diagrami-klasiv.html

11. Laravel Routing – Режим доступу: https://laravel.com/docs/10.x/routing

12.Laravel Middleware – Режим доступу: https://laravel.com/docs/10.x/middleware

13. Laravel HTTP Request – Режим доступу: https://laravel.com/docs/10.x/requests

14. HTTP Session – Режим доступу: https://laravel.com/docs/10.x/session

15. Error Handling Laravel– Режим доступу: https://laravel.com/docs/10.x/errors

Додаток А

Вебресурс Освітня платформа з особистим кабінетом студента за допомогою фреймворку Laravel

Специфікація

ПЗБП_ІСТ_КПІ_2023_КВІ

Аркушів 3

Київ – 2023

Позначення	Найменування	Примітки
	Документація	
ПЗБП_ІСТ_КПІ_2023_КВІ	Записка Кондратенко В.І.ІК-92.docx	Текстова частина дипломної роботи
	Компоненти	
ПЗБП_ІСТ_КПІ_2023_КВІ	AuthController.php, PostController.php, UserContreller.php, CommentController.php, HomeController.php, AuthService.php, PostService.php, UserService.php, CommentService.php, HomeService.php, AuthRepository.php, PostRepository.php, UserRepository.php, CommentRepository.php, HomeRepository.php	Оброблювач запитів до серверу
ПЗБП_ІСТ_КПІ_2023_КВІ	Comment.php, Post.php, User.php	Представлення даних БД в коді
ПЗБП_ІСТ_КПІ_2023_КВІ	Routes	Оголошення маршрутів

Додаток Б

Вебресурс Освітня платформа з особистим кабінетом студента за допомогою фреймворку Laravel

Лістнинг програми

ПЗБП_ІСТ_КПІ_2023_КВІ

Аркушів 22

Київ – 2023

AuthController.php

```php
<?php

namespace App\Http\Controllers\Dashboard\Auth;

use App\Services\Dashboard\PostService;
use App\Services\Dashboard\UserService;
use App\Services\Front\CommentService;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use App\Http\Requests\Dashboard\Auth\AuthRequest;
use App\Http\Requests\Dashboard\Auth\RegisterRequest;
use App\Services\Dashboard\Auth\AuthService;
use Illuminate\Support\Facades\Auth;


class AuthController extends Controller
{

    /**
     * @var AuthService
     */
    protected $authService;
    /**
     * @var PostService
     */
    protected $postService;
    /**
     * @var UserService
     */
    protected $userService;
    /**
     * @var CommentService
     */
    protected $commentService;

    public function __construct(AuthService $authService,
PostService $postService, UserService $userService, CommentService
$commentService)
    {
        $this->authService = $authService;
        $this->userService = $userService;
        $this->postService = $postService;
        $this->commentService = $commentService;
    }

    public function index(){
        return view('dashboard.dashboard', [
            'users' => count($this->userService->get()),
            'posts' => count($this->postService->get()),
            'comments' => count($this->commentService->get()),
        ]);
    }
```

```
    public function login(){
        return view('dashboard.auth.login');
    }

    public function register(){
        return view('dashboard.auth.register');
    }

    public function registerCreate(RegisterRequest $request){
        $res = $this->authService->create($request->except('_token'));

        return $res ? redirect()->route('admin_dashboard') : back()->with('error', "Something went wrong");
    }

    public function login_submit(AuthRequest $request){

        // dd($request->except('_token'));
        $res = $this->authService->attempt($request->except('_token'));

        return $res ? redirect()->route('admin_dashboard') : back()->with('error', "Email or Password is incorrect");
    }

    public function logout(){
        Auth::guard('web')->logout();

        return redirect()->route('home');
    }

    public function forget_password(){
        return view('dashboard.auth.forget_password');
    }

    public function forget_password_submit(Request $request){
        $request->validate([
            'email' => 'required|email',
        ]);

        $res = $this->authService->forgetPassword($request->email);

        return $res ?  redirect()->route('admin_login')->with('success', "Mail has been sent to your mailbox. Check it out!") : back()->with('error', 'There is no user with such email!');
    }


}
```

PostController.php

```php
<?php


namespace App\Http\Controllers\Dashboard;

use App\Http\Controllers\Controller;
use App\Enums\Dashboard\PostEnum;
use App\Http\Requests\Dashboard\PostRequest;
use Illuminate\Http\Request;
use App\Services\Dashboard\PostService;
use Illuminate\Support\Facades\Auth;


class PostController extends Controller
{
    /**
     * @var PostService
     */
    private $postService;


    public function __construct(PostService $postService)
    {
        $this->postService = $postService;

    }

    public function index()
    {
        $posts = [];
        if(Auth::user()->role == 'admin'){
            $posts = $this->postService->get();
        }
        else{
            $posts = Auth::user()->posts;
        }
        return view('dashboard.post.index', [
            'posts' => $posts,
        ]);
    }

    public function create()
    {
        return view('dashboard.post.create');
    }

    public function store(PostRequest $request)
    {

        $postArray = $request->except('_token');
        $postArray['user_id'] = Auth::user()->id;
        if (isset($request->feature_image)) {
            $postArray['feature_image'] = self::saveFile($request-
>feature_image, PostEnum::POST_FEATURE_IMAGE_PATH,);
        }
```

```
        $result = $this->postService->store($postArray);

        return $result ? redirect()->route('admin_posts')-
>with('success', 'Post successfully added') : back()-
>with('error', 'Error db connection');
    }

    public function edit($id)
    {
        return view('dashboard.post.edit', [
            'post' => $this->postService->getById($id),
        ]);
    }

    public function update($id, PostRequest $request)
    {
        $postArray = $request->except('_token');

        if (isset($request->feature_image)) {
            $postArray['feature_image'] = self::saveFile($request-
>feature_image, PostEnum::POST_FEATURE_IMAGE_PATH);
        }

        $result = $this->postService->update($id, $postArray);

        return $result ? redirect()->route('admin_posts')-
>with('success', 'Post updated') : back()->with('error', 'Error db
update');
    }

    public function delete($id)
    {
        $result = $this->postService->delete($id);
        return $result ? back()->with('success', 'Post deleted
permanently') : back()->with('error', 'Post not found');
    }
```

UserController.php

```
<?php

namespace App\Http\Controllers\Dashboard;

use App\Http\Controllers\Controller;
use App\Http\Requests\Dashboard\TutorRequest;
use App\Services\Dashboard\UserService;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\DB;

class UserController extends Controller
{
    /**
```

```
     * @var UserService
     */
    protected $userService;

    public function __construct(UserService $userService)
    {
        $this->userService = $userService;
    }

    public function getStudents(){
        return view('dashboard.students.index', [
            'students' => $this->userService->getStudents(),
        ]);
    }
    public function deleteStudent(int $id)
    {
        if($this->userService->deleteStudent($id))
            return back()->with('success', 'Student deleted
permanently');

        return back()->with('error', 'Some kind of error
occurred');
    }

    public function getTutors(){
        return view('dashboard.tutors.index', [
            'students' => $this->userService->getTutors(),
        ]);
    }
    public function deleteTutor(int $id)
    {
        if($this->userService->deleteTutor($id))
            return back()->with('success', 'Tutor deleted
permanently');

        return back()->with('error', 'Some kind of error
occurred');
    }

    public function createTutor(){
        return view('dashboard.tutors.create');
    }

    public function storeTutor(TutorRequest $request){

        $result = $this->userService->store($request-
>except('_token'));

        if($result){
            return redirect()->route('admin_tutors')-
>with('success', 'Tutor added successfully!');
        }
        return redirect()->route('admin_tutors')->with('error',
'Something went wrong(');
```

```
    }
}
```

CommentController.php

```
<?php

namespace App\Http\Controllers\Front;

use App\Http\Controllers\Controller;
use App\Http\Requests\Dashboard\CommentRequest;
use App\Services\Front\CommentService;

class CommentController extends Controller
{
    /**
     * @var CommentService
     */
    protected $commentService;

    public function __construct(CommentService $commentService)
    {
        $this->commentService = $commentService;
    }

    public function store(CommentRequest $request)
    {
        $comment = $this->commentService->store($request-
>except('_token'));

        if($comment){
            return back()->with('success', 'Comment stored
successfully');
        }
        return back()->with('error', 'Something went wrong(
OOOOPppps');
    }
    public function replyStore(CommentRequest $request)
    {
        $comment = $this->commentService->store($request-
>except('_token'));

        if($comment){
            return back()->with('success', 'Comment stored
successfully');
        }
        return back()->with('error', 'Something went wrong(
OOOOPppps');
    }

    public function delete($id){
        $result = $this->commentService->delete($id);
```

```
        if($result){
            return back()->with('success', 'Comment was
successfully deleted!');
        }
        return back()->with('error', 'Something went wrong(
OOOOPppps');

    }
}
```

HomeController.php

```
<?php

namespace App\Http\Controllers\Front;

use App\Http\Controllers\Controller;
use App\Services\Dashboard\PostService;
use Illuminate\Http\Request;

class HomeController extends Controller
{
    /**
     * @var PostService
     */
    protected $postService;

    public function __construct(PostService $postService)
    {
        $this->postService = $postService;
    }

    public function index(){
        return view('front.home', [
            'posts' => $this->postService->get(),
        ]);
    }
}
```

CommentRequest.php

```
<?php

namespace App\Http\Requests\Dashboard;

use Illuminate\Foundation\Http\FormRequest;

class CommentRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
```

```
    }

    /**
     * @return array
     */
    public function rules()
    {
        $rules = [
            'message' => ['required', 'string'],
            'user_id' => ['integer'],
            'post_id' => ['integer'],
            'parent_id' => ['nullable'],
            ];

        return $rules;
    }
}
```

PostRequest.php

```
<?php

namespace App\Http\Requests\Dashboard;

use Illuminate\Foundation\Http\FormRequest;

class PostRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * @return array
     */
    public function rules()
    {
        $rules = [
            'title' => ['required', 'string', 'max:255'],
            'details' => ['nullable','string'],
            'feature_image' => ['nullable', 'file'],
            'is_commentable' => ['nullable', 'string'],
        ];

        return $rules;
    }
}
```

TutorRequest.php

```php
<?php


namespace App\Http\Requests\Dashboard;

use Illuminate\Foundation\Http\FormRequest;

class TutorRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * @return array
     */
    public function rules()
    {
        $rules = [
            'name' => ['required', 'string', 'max:255'],
            'email' => ['email', 'string'],
            'password' => ['required', 'string'],
        ];

        return $rules;
    }
}
```

Comment.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Comment extends Model
{
    use HasFactory;

    protected $fillable = [
        'message', 'user_id', 'post_id', 'parent_id',
'created_at', 'updated_at',
    ];

    public function user(){
        return $this->belongsTo(User::class, 'user_id');
    }
```

```
    public function replies(){
        return $this->hasMany(Comment::class, 'parent_id');
    }
}
```

Post.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    use HasFactory;

    protected $fillable = [
        'title', 'details', 'is_commentable', 'feature_image',
'user_id', 'created_at', 'updated_at',
    ];

    public function user(){
        return $this->belongsTo(User::class, 'user_id');
    }

    public function comments(){
        return $this->hasMany(Comment::class, 'post_id')-
>where('parent_id', 0);
    }
}
```

User.php

```
<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'email',
```

```
        'password',
        'role',
        'remember_token',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    public function posts(){
        return $this->hasMany(Post::class);
    }
}
```

PostRepository.php

```
<?php

namespace App\Repositories\Dashboard;

use App\Models\Post;
use App\Repositories\BaseRepository;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Storage;


class PostRepository extends BaseRepository
{
    /**
     * @var Post
     */
    protected $eloquent = Post::class;



    public function create(array $postArray): ?Post
    {
        return $this->model()
            ->create($postArray);
    }
```

```
    public function update($id, array $postArray): ?Post
    {
        $post = $this->model()->findOrFail($id);
        if(isset($postArray['feature_image']) &&
!preg_match('/noImage.png/', $post->feature_image)){
            Storage::disk('feature_image')-
>delete(last(explode('/', $post->feature_image)));
        }
        if($post){
            $post->update($postArray);
            return $post;
        }
        return false;

    }


    public function getById($id): ?Post
    {
        return $this->model()->findOrFail($id);
    }
}
```

UserRepository.php

```
<?php

namespace App\Repositories\Dashboard;


use App\Models\User;
use App\Repositories\BaseRepository;

class UserRepository extends BaseRepository
{
    /**
     * @var $eloquent
     */

    protected $eloquent = User::class;

    public function getStudents(){
        return $this->model()->where('role', 'student')->latest()-
>get();
    }

    public function getTutors(){
        return $this->model()->where('role', 'tutor')->latest()-
>get();
    }
}
```

CommentRepository.php

```
<?php

namespace App\Repositories\Front;
```

```php
use App\Models\Comment;
use App\Repositories\BaseRepository;

class CommentRepository extends BaseRepository
{
    /**
     * @var Comment
     */
    protected $eloquent = Comment::class;

    public function get(){
        return $this->model()->all();
    }

    public function delete(int $id): bool
    {
        $comment = $this->model()->findOrFail($id);

        if($comment){
            if($comment->parent_id){
                return $comment->delete();
            }
            else{
                $comment->replies()->delete();
                return $comment->delete();
            }
        }
        return false;
    }
}
```

BaseRepository.php

```php
<?php

namespace App\Repositories;



class BaseRepository
{
    protected $eloquent;

    protected function model()
    {
        if (!$this->eloquent) {
            throw new Exception('No eloquent model set for' .
get_class($this));
        }

        return new $this->eloquent;
    }

    public function create(array $arr){
        return $this->model()->create($arr);
```

```
    }

    public function all(){
        return $this->model()->latest()->get();
    }

    public function findOrFail(int $id){
        return $this->model()->findOrFail($id);
    }

    public function delete(int $id): bool
    {
        return (bool) $this->model()
            ->findOrFail($id)
            ->delete();
    }
}
```

AuthService.php

```
<?php

namespace App\Services\Dashboard\Auth;

use App\Services\BaseService;
use App\Repositories\Dashboard\Auth\AuthRepository;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;

class AuthService extends BaseService
{

    /**
     * @var AuthRepository
     */
    protected $authRepository;

    public function __construct(AuthRepository $authRepository)
    {
        $this->authRepository = $authRepository;
    }

    public function attempt(array $creds)
    {
        if (Auth::guard('web')->attempt($creds)) {
            return true;
        }
        return false;
    }

    public function create(array $userArray){
        $userArray['remember_token'] = hash('sha256', time());
        $userArray['role'] = 'student';
        $userArray['password'] =
Hash::make($userArray['password']);
        return $this->authRepository->create($userArray);
```

```
    }
    public function forgetPassword($email)
    {

        $res = $this->authRepository->forgetPassword($email);

        if ($res) {

            //Mail send

            // $subject = 'Reset password';
            // $url = url('admin/reset-password/'.$admin-
>id.'/'.$token);
            // $body = 'Click <a href="'.$url.'">here to reset
your password</a>';

            // \Mail::to($admin->email)->send(new
Webmail($subject, $body));

            return true;
        }

        return false;
    }
}
```

PostService.php

```
<?php

namespace App\Services\Dashboard;

use App\Repositories\Dashboard\PostRepository;
use App\Services\BaseService;

class PostService extends BaseService
{
    /**
     * @var PostRepository
     */
    private $postRepository;

    public function __construct(PostRepository $postRepository)
    {
        $this->postRepository = $postRepository;
    }

    public function get()
    {
        return $this->postRepository->all();
    }

    public function store(array $postArray)
    {

        if($postArray['is_commentable'] == 'on'){
```

```
            $postArray['is_commentable'] = 1;
        }
        else{
            $postArray['is_commentable'] = 0;
        }

        return $this->postRepository->create($postArray);

    }

    public function update($id, array $postArray): bool
    {


        if(isset($postArray['is_commentable'])){
            $postArray['is_commentable'] = 1;
        }
        else{
            $postArray['is_commentable'] = 0;
        }



        $post = $this->postRepository->update($id, $postArray);
        if($post){
            return true;
        }
        return false;
    }

    public function delete($id): bool
    {
        $post = $this->postRepository->findOrFail($id);

        if(!preg_match('/noImage.png/', $post->feature_image)){
            self::deleteFile(last(explode('/', $post-
>feature_image)));
        }

        return $this->postRepository->delete($id);
    }

    public function getById(int $id)
    {
        return $this->postRepository->getById($id);
    }

}
```

UserService.php

```
<?php

namespace App\Services\Dashboard;
```

```php
use App\Repositories\Dashboard\UserRepository;
use App\Services\BaseService;
use Illuminate\Support\Facades\Hash;

class UserService extends BaseService
{
    /**
     * @var UserRepository
     */
    protected $userRepository;



    public function __construct(UserRepository $userRepository)
    {
        $this->userRepository = $userRepository;
    }

    public function store(array $tutorArray){
        if(!isset($tutorArray['role'])){
            $tutorArray['role'] = 'tutor';
        }
        if(isset($tutorArray['password'])){
            $tutorArray['password'] =
Hash::make($tutorArray['password']);
        }
        if(isset($tutorArray['password'])){
            $tutorArray['remember_token'] = hash('sha256',
time());
        }
        return $this->userRepository->create($tutorArray);
    }

    public function get(){
        return $this->userRepository->all();
    }

    public function getStudents(){
        return $this->userRepository->getStudents();
    }

    public function deleteStudent($id): bool
    {
        $student = $this->userRepository->findOrFail($id);

        if($student){
            return self::userDelete($student);
        }
        return false;
    }

    public function getTutors(){
        return $this->userRepository->getTutors();
    }
```

```
    public function deleteTutor($id): bool
    {
        $tutor = $this->userRepository->findOrFail($id);

        if($tutor){
           return self::userDelete($tutor);
        }
        return false;
    }

    public function userDelete($user){
        foreach ($user->posts as $post){
            if(!preg_match('/noImage.png/', $post->feature_image))
{
                self::deleteFile(last(explode('/', $post-
>feature_image)));
            }
        }
        $user->posts->map->delete();
        $user->delete();

        return true;
    }
}
```

CommentService.php

```
<?php

namespace App\Services\Front;

use App\Repositories\Front\CommentRepository;
use App\Services\BaseService;

class CommentService extends BaseService
{
    /**
     *@var CommentRepository
     */
    protected $commentRepository;

    public function __construct(CommentRepository
$commentRepository)
    {
        $this->commentRepository = $commentRepository;
    }


    public function get()
    {
        return $this->commentRepository->get();
    }

    public function store(array $commentArray)
    {
```

```
        return $this->commentRepository->create($commentArray);
    }
    public function delete(int $id){
        return $this->commentRepository->delete($id);
    }
}
```

BaseService.php

```
<?php

namespace App\Services;

use Illuminate\Support\Facades\Storage;

class BaseService
{
    public function deleteFile($fileName){
        Storage::disk('feature_image')->delete($fileName);
    }
}
```

routes.php

```
<?php

use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/
Route::group(['namespace' => 'App\Http\Controllers\Front'], function () {
    Route::get('/', 'HomeController@index')->name('home');

    Route::post('/comment/store', 'CommentController@store')->name('comment_store');
    Route::post('/reply-comment/store', 'CommentController@replyStore')->name('comment_reply_store');
    Route::get('/comment/delete/{id}', 'CommentController@delete')->name('comment_delete');
});

Route::group(['prefix' => 'admin'], function () {
    Route::group(['middleware' => 'auth', 'namespace' => 'App\Http\Controllers\Dashboard'], function () {
```

```
        Route::get('/', 'Auth\AuthController@index')->name('admin_dashboard');


        /* Tutors routes */
        Route::get('/tutors/all', 'UserController@getTutors')->name('admin_tutors');
        Route::get('/tutor/create', 'UserController@createTutor')->name('admin_tutor_create');
        Route::post('/tutor/store', 'UserController@storeTutor')->name('admin_tutor_store');
        Route::get('/tutor/delete/{id}', 'UserController@deleteTutor')->name('admin_tutor_delete');

        /* Students routes */
        Route::get('/students/all', 'UserController@getStudents')->name('admin_students');
        Route::get('/student/delete/{id}', 'UserController@deleteStudent')->name('admin_student_delete');

        /* Posts routes */
        Route::get('/posts/all', 'PostController@index')->name('admin_posts');
        Route::get('/post/create', 'PostController@create')->name('admin_post_create');
        Route::post('/post/store', 'PostController@store')->name('admin_post_store');
        Route::get('/post/delete/{id}', 'PostController@delete')->name('admin_post_delete');
        Route::get('/post/edit/{id}', 'PostController@edit')->name('admin_post_edit');
        Route::post('/post/update/{id}', 'PostController@update')->name('admin_post_update');
    });



});
Route::group(['namespace' => 'App\Http\Controllers\Dashboard\Auth'], function () {
    Route::get('/register', 'AuthController@register')->name('admin_register');
    Route::post('/register/submit', 'AuthController@registerCreate')->name('admin_register_create');
    Route::get('/login', 'AuthController@login')->name('admin_login');
    Route::get('/logout', 'AuthController@logout')->name('admin_logout');
    Route::post('/login/submit', 'AuthController@login_submit')->name('admin_login_submit');

    Route::get('/forget-password', 'AuthCOntroller@forget_password')->name('admin_forget_password');
    Route::post('/forget-password/submit',
```

```
'AuthController@forget_password_submit')-
>name('admin_forget_password_submit');
});
```