НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____Олександр КОВАЛЬ
«_____»_____202_р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці і веб-технологій»
спеціальності 121 Інженерія програмного забезпечення
на тему: «Створення системи автоматизованого планування бізнес-процесів для контингенту кафедри»

Виконала:
студентка IV курсу, групи ТВ-91
Левкун Дар'я Павлівна _____ _____
(прізвище, ім'я, по батькові) (підпис)
Керівник:
ст. викл. Бандурка О.І. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
Професор кафедри ЦТЕ, д.т.н., проф. Отрох С.І _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____
(підпис)

Київ – 2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер - фізичних систем в енергетиці і веб-технологій»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
(підпис)
«____»_____ 202_р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Левкун Дар'ї Павлівні_____
(прізвище, ім'я, по батькові)

1. Тема роботи
Створення системи автоматизованого планування бізнес-процесів для контингенту кафедри_____
керівник роботи Бандурка Олена Іванівна, старший викладач_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "29" __травня__ 2023 року № 2039-с___
2. Строк подання студентом роботи__12.06.2023_____
3. Вихідні дані до роботи _мова програмування PHP, фреймворк Laravel, середовище розробки PhpStorm, система керування базами даних MySQL, мова програмування JavaScript_____
_____
_____
_____
4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити)_розробити веб-систему для моніторингу студентської активності та планування часу; спроектувати модель бази даних; описати програмну реалізацію; проаналізувати переваги програмних засобів, що використовуються; порівняти програмний продукт з його аналогами розробити користувацький інтерфейс; протестувати програмний продукт_____
_____

5. Перелік ілюстративного матеріалу_схема обміну даними, діагарама_____
прецедентів, концептуальна модель бази даних, логічна модель бази даних,_____
фізична модель бази даних, діагарами послідовності_____
_____
_____

6. Дата видачі завдання «30» вересня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	30.09.2022	Виконано
2	Дослідження предметної області	15.11.22-31.02.23	Виконано
3	Постановка вимог до проєктування системи	03.03.23 – 24.03.23	Виконано
4	Дослідження існуючих рішень	24.03.23 – 15.04.23	Виконано
5	Розробка програмного продукту	17.04.23 – 08.05.23	Виконано
6	Тестування	08.05.23 – 15.05.23	Виконано
7	Захист програмного продукту	16.05.23	Виконано
8	Оформлення дипломної роботи	22.05.23 – 04.06.23	Виконано
9	Передзахист	05.06.22	Виконано
10	Захист	19.06.22-30.06.22	Виконано

Студент _____ Дар’я ЛЕВКУН
(підпис) (ім’я, прізвище)

Керівник роботи _____ Олена БАНДУРКА
(підпис) (ім’я, прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 65 сторінок, 15 рисунків, 2 додатки та 12 посилань.

Метою роботи є полегшення та покращення організації навчального процесу, скорочення витрат часу та зусиль на планування та контроль. Завдання полягає у вчасному інформування користувачів про заплановані події та їхні результати, залучення батьків до спостерігання за успішністю їхніх дітей, що навчаються в університеті. Ця система розроблена з метою поліпшення організації навчального процесу на кафедрі. Вона спрямована на забезпечення ефективного контролю та покращення взаємодії між викладачами, студентами та батьками.

Практичне значення одержаних результатів полягає в поліпшенні організації навчального процесу, ефективному контролі та зручному доступу до інформації для колективів, що збираються використовувати даний продукт.

Апробація результатів дипломної роботи

Основні положення роботи доповідалися та обговорювалися на конференції:

1. Левкун Д.П., Бандурка О.І., Свинчук О.В. Інформаційна система моніторингу успішності студентів. Матеріали XXIII Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів «Стан, досягнення та перспективи інформаційних систем і технологій – 2023». Одеса, 20-21 квітня 2023 р. С.155-156.

Ключові слова: навчальний процес, веб-система, кафедра, контролер, представлення, модель даних, інтерфейс.

ABSTRACT

Structure and scope of the thesis. The work contains 65 pages, 15 figures, 2 appendices and 12 references.

The purpose of the work is to facilitate and improve the organization of the educational process, to reduce time and effort spent on planning and control. The task is to timely inform users about planned events and their results, involve parents in monitoring the progress of their children that study at the university. This system was developed in order to improve the organization of the educational process at the department. It is aimed at ensuring effective control and improving interaction between teachers, students and parents.

The practical significance of the obtained results lies in the improvement of the organization of the educational process, effective control and convenient access to information for teams that are going to use this product.

Approbation of the results of the thesis

The main provisions of the work were reported and discussed at the conference:

1. Levkun D.P., Bandurka O.I., Svynchuk O.V. Information system for monitoring students' progress. Materials of the 23rd All-Ukrainian scientific and technical conference of young scientists, graduate students and students "State, achievements and prospects of information systems and technologies - 2023". Odesa, April 20-21, 2023. P.155-156.

Keywords: educational process, web system, department, controller, representation, data model, interface.

ЗМІСТ

ВСТУП

Автоматизовані системи планування бізнес-процесів давно стали необхідним інструментом для ефективного управління діяльністю різних організацій і навчальних закладів.

Навчальний процес вимагає зберігання великої кількості різноманітних даних. Ці дані можна по різному класифікувати. Крім цього, різні групи даних постійно взаємодіють один з одним. Для систематизації та зберігання цих даних зручно використовувати програмні системи, які зберігають всю інформацію в пам’яті комп’ютера. Хороша програмна система не тільки є джерелом цих даних, вона грамотно ними оперує та подає потрібно інформацію в зручній та зрозумілій формі, керує діями користувача та допомагає йому досягти його мети.

Метою системи автоматизованого планування бізнес-процесів для контингенту кафедри є систематизація навчальних досягнень студента, відслідковування та зберігання завдань, лабораторних та практичних робіт, планування часу студента та надання доступу про дані студента батькам. Головна мета програми - зробити процес навчання простішим, більш організованим та контрольованим. Уникнути ситуацій з загубленими завданнями та забутими оцінками.

Системою можуть користуватись різні люди, яким потрібно виконувати різного роду завдання. Але при цьому вони всі об’єднані спільною метою. В даному випадку - організацією навчального процесу.

Щоб розрізняти дії кожної групи користувачів та обмежити користування деякими функціями, кожному користувачу присвоюється роль. Роль визначає, до яких даних можна отримувати доступ та які дії можна виконувати.

Що стосується даної роботи, то варто відмітити залучення такої ролі як батьки. Це не типово для більшості програм, в яких беруть участь тільки викладачі та студенти. З одного боку це розуміється, як щось цілком логічне. З іншої сторони, батьки часто цікавляться успіхами дітей, проте не завжди розуміють як

відбувається процес навчання. Це пояснюється тим, що система освіти постійно змінюється, покращується, шукаються нові методики та підходи до навчання спеціалістів. Зрозуміло, що процес навчання відрізняється від того, яким він був тридцять років назад. Тому залучення батьків до спостерігання за навчальними процесами - є ще одним рішенням у зміні процесу освіти. Перші курси університету ще можуть бути вразливими до змін, не розуміти, що вони хочуть та навіщо вони поступили у вищий навчальний заклад. Тому підтримка батьків в такій ситуації може стати заспокійливим фактором. Батьки не прийматимуть безпосередню участь в процесах кафедри та університеті, але вони можуть спостерігати зі сторони за прогресом їхньої дитини.

Варто зазначити, що мета системи не полягає в встановленні контролю за студентом. Система лише забезпечує прозорість бізнес-процесів кафедри.

Програма розроблялась на основі власного досвіду та реальних проблем. Принцип ідеї полягає в тому, що “усе знаходиться під рукою”. Система нагадує цифровий аналог щоденника-планера, тільки програма сама все організовує за користувача, йому лишається тільки ввести свої дані.

Розробка інтерфейсу займає суттєвий відрізок часу, адже це настільки ж важливо, як і розробка бекенд частини. Дизайн веб-інтерфейсу повинен бути зручним та інтуїтивно зрозумілим для користувачів. Він повинен мати зрозумілу структуру, з якою користувач може легко орієнтуватися, швидко знаходити потрібну інформацію та виконувати необхідні дії. Саме на етапі дизайну варто створити умови для забезпечення високої продуктивності та захисту даних системи.

У наступних розділах буде детально описано функціонал системи та засоби його створення, описано структуру бази даних та реалізацію основних функцій.

1 МЕТА ТА ЗАВДАННЯ СИСТЕМИ

Завдання роботи полягає в створенні системи автоматизованого планування бізнес-процесів для контингенту відділу. Основна мета системи – полегшити та покращити організацію навчального процесу, скоротити витрати часу та зусиль на планування та контроль.

1.1 Мета функціоналу програми

Для досягнення поставленої мети необхідно розробити програмний продукт, який матиме наступний функціонал:

- реєстрація користувачів (студентів, викладачів і батьків) і введення їх основних даних;
- перегляд розкладу занять, включаючи дисципліни, час, місце проведення та викладачів;
- можливість додавати завдання викладачів і виставляти за них оцінки;
- можливість створювати нотатки та встановлювати нагадування для користувачів;
- можливість контролю за успішністю студентів батьками.

Основною метою розробки даної системи є вдосконалення організації навчального процесу на кафедрі, забезпечення ефективного контролю та покращення взаємодії між викладачами, студентами та батьками.

1.2 Цільові користувачі

Система розроблялась для кафедри університету та має обмежений доступ. Адміністратори, які також є користувачами системи, належать до цієї кафедри та керують основними процесами, що стосуються користування системою.

Адміністратором може бути уповноважений працівник з кафедри: викладач, завідуючий, секретар тощо.

Цільовими користувачами є учасники навчального процесу: студенти та викладачі. Студенти навчаються в університеті, на заочній чи денній формі навчання, можуть мати будь-який освітній ступінь. Вони мають пряме відношення до інформації, що надає кафедра, а саме належать до групи, мають доступ до розкладу, отримують завдання та оцінки.

Викладачі обізнані в правилах виконання роботи на кафедрі. Вони мають доступ до системи, де бачать власний розклад, дисципліни, які вони ведуть, списки груп. Викладач є джерелом вхідних даних, так як він вносить інформацію про завдання та оцінки.

Ще однією групою користувачів є батьки. Вони мають відношення до дітей, які є студентами даної кафедри. Батьки в основному отримують вихідну інформацію у вигляді завдань та оцінок студента, який є їхньою дитиною.

1.3 Огляд аналогів

В мережі інтернет існує безліч систем для організації навчання. Серед них найпопулярнішою є Google Classroom. Він дозволяє вчителям і учням спілкуватися, співпрацювати та вести навчання в онлайн-середовищі. Це платформа для управління класом, де вчителі можуть створювати віртуальні класи, додавати учнів і надавати їм завдання, матеріали та засоби для спільної роботи. В програмного продукту, що був описується в даній роботі, є схожі властивості та функціонал. Викладачі можуть створювати завдання та виставляти оцінки, а студенти їх переглядати.

Своїм підходом до виставлення оцінок та організації інформації програма нагадує університетський Електронний Кампус. Кампус є університетською системою, де зберігається більшість інформації про студентів. В системі виставляються оцінки за завдання та сесію, зберігаються списки груп тощо. Крім

цього, там можна знайти оголошення, що стосуються студентського життя або зареєструватись на вступ до магістратури.

Всі ці системи об’єднує декілька речей. По-перше, цільова аудиторія – це викладачі та студенти. По-друге, є можливість викладання завдань, матеріали та виставлення оцінок студентам, за допомогою чого оцінюється успішність студентів. Викладачі в курсі власних активностей та обов’язків, що стосуються роботі в університеті.

Так як розроблений програмний продукт має функціонал створення нотаток, не можна не згадати про безліч застосунків, які мають в основі списки справ або «чек-листи». Це зазвичай невелика програма, в які користувач записує свої справи, щоб в разі чого швидко переглянути свої записи.

Головною перевагою розробленої програми є те, що вона поєднує в собі деякі функції перерахованих раніше застосунків.

По-перше, система дозволяє додавати до процесу батьків студентів. При цьому в ній вже передбачена така роль, яка відрізняється від інших. Тому не потрібно додатково визначати права для батьків, потрібно лише вказати їхню роль.

Система є вузько направленою та створювалась суто для потреб факультету або навіть і кафедри. Це дозволяє підключити багато ресурсів, що стосуються конкретного університету, таким чином додати застосунок в загальну систему університету, що відкриває більше можливостей для її функціональності.

Система створювалась за принципом «все під рукою». Тому користувачі мають доступ до всіх складових, що потрібні під час навчання. Вони можуть переглянути розклад і відразу створити нотатку на цей день. Створюючи нотатку студент може прикріпити вже готове завдання створене викладачем і поля форми автоматично заповняться потрібною інформацією.

Висновки до розділу 1

Мета системи спростити та покращити рівень навчального процесу на кафедрі. Користувачі взаємодіють між собою та обмінюються інформацією, яку потім використовують для навчання та організації свого часу. Система буде спрямована на автоматизацію рутинних завдань, забезпечення швидкого доступу до необхідної інформації та полегшення взаємодії між різними учасниками навчального процесу. В результаті використання такої системи очікується підвищення ефективності роботи, а також зниження помилок та витрат ресурсів.

Додаток має достатньо аналогів в інтернеті, проте існують деякі переваги та додаткові функції, яких бракує цим аналогам для виконання поставленої задачі.

2 ІНФОРМАЦІЙНІ ПОТОКИ ДАНИХ

Система оперує даними, які подані їй на вхід. Програма вибирає потрібну їй інформацію, приводить її до відповідного вигляду та зберігає у своєму сховищі (в даному випадку - в базі даних). Дані, які приходять в систему, потрібно попередньо обробити, так як вони можуть не відповідати правильному формату, містити помилкову інформацію, що може спричинити збої системи в майбутньому. Тому застосовуються різні методи та операції обробки даних.

Спершу їх потрібно очистити, видалити непотрібну або неправильно інформацію, привести до потрібного формату або типу.

Наступним важливим кроком є фільтрація даних. Дані з зовнішніх джерел мають якусь структуру, яка може не відповідати схемі даних основної програми. Тому варто відфільтрувати та відкинути поля, які в системі не будуть використовуватись.

Після цього проміжним етапом можу бути сортування даних. У відсортованому вигляді дані краще сприймаються, а пошук здійснюється швидше та зручніше. Беручи до уваги, що система працює зі списками студентів, оцінок та завдань, можемо зазначити, що використання сортування є дуже доречним та навіть може позитивно вплинути на ефективність та швидкість роботи системи.

На рисунку 2.1 зображено схему обміну даними в системі.

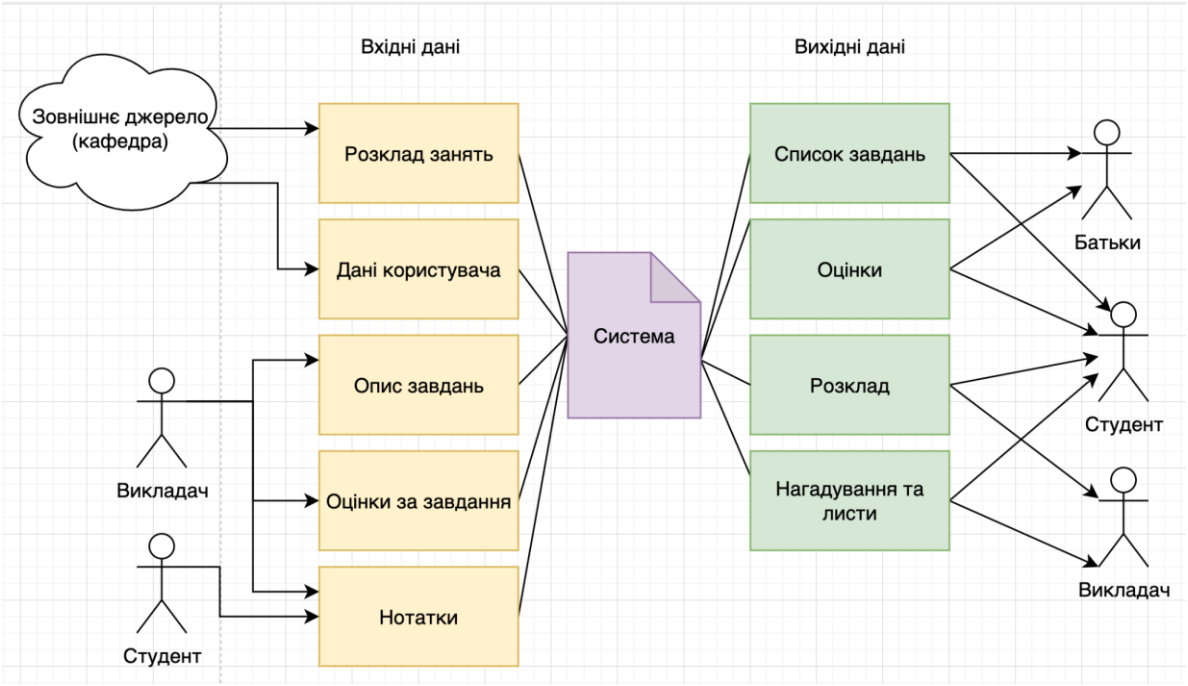

Рисунок 2.1 - Схема обміну даними в системі

У кінці кінців дані групуються та зберігаються в таблицях бази даних. Сюди входить і визначення зв’язків між сутностями, які по різному взаємодіють в системі.

В даному випадку в систему надходять такі дані:

- розклад занять викладачів та груп студентів в електронному вигляді;
- інформація про викладачів, студентів та батьків, така як їхнє ім'я, прізвище, електронна адреса та інші контактні дані;
- інформація про завдання, які повинні бути виконані студентами, такі як опис завдання, термін виконання та бали, що можуть бути отримані за виконання завдання;
- інформація про оцінки, які виставляють викладачі за виконання завдань студентами;
- нотатки створені користувачами системи.

У свою чергу ця інформація трансформується у вихідні дані:

- розклад занять викладачів та груп студентів;

- список завдань, які повинні виконати студенти, з описом завдання, терміном виконання та очікуваними балами;
- оцінки, виставлені викладачами за виконання завдань студентами;
- нотатки та нагадування, створені користувачами системи;
- повідомлення, які автоматично відправляються користувачам на їхню електронну пошту та в застосунку, наприклад, нагадування про завдання, що мають бути виконані або про попередньо зазначений термін виконання.

Висновки до розділу 2

Більша частина інформації стосується даних, які вже є в основних базах університету. Тому вхідні дані, що стосуються розкладу, дисциплін та контактної інформації користувачів надходить з зовнішніх джерел. У свою чергу, викладачі також є постачальниками вхідних даних, так як надають інформацію про завдання та оцінки, які споживають студенти та батьки. Ця інформація для них є вихідною. На виході ми отримуємо записи нотаток та повідомлення, що отримують користувача в курсі процесів, що відбуваються на кафедрі. У результаті відбувається постійний обмін даними між різними користувачами системи.

3 ЗАСОБИ РОЗРОБКИ

Код програмного продукту складається з двох великих частин: бекенд та фронтенд. Бекенд частина відповідає за передавання, обробку даних та зберігання даних, перенаправляє запити, формує структури даних. Фронтенд відповідає за відображення даних та UX, те як користувач взаємодіє з системою.

3.1 Технології розробки бекенду

Бекенд реалізована мовою програмування PHP з використанням фреймворку Laravel. PHP — це скриптова мова програмування, призначена для створення динамічних веб-сторінок і взаємодії з базами даних. PHP є однією з найпопулярніших мов програмування для розробки веб-додатків, особливо для розробки на стороні сервера.

PHP в основному використовується для створення динамічних веб-сторінок, які можуть змінюватися залежно від дій користувача. Крім того, PHP дозволяє розробникам створювати веб-додатки, які можуть працювати з різними базами даних, включаючи MySQL, PostgreSQL і Oracle. Використання PHP дозволяє створювати веб-програми, які забезпечують високу продуктивність, безпеку та зручний інтерфейс.

Скриптова мова програмування — це мова програмування, призначена для написання сценаріїв (програм-сценаріїв), які виконують короткі, невеликі та швидкі завдання, зазвичай із часом виконання до кількох хвилин. Мови сценаріїв не вимагають компіляції перед виконанням і використовуються для автоматизації різних завдань. Такі мови мають легший синтаксис. Їх зручно використовувати для коротких, швидких задач, наприклад, формування запиту на сервер або перенаправлення на іншу сторінку. Тому мова програмування PHP добре підходить для написання веб-систем.

Також для створення системи було використано фреймворк Laravel. Він має добре налаштовану систему маршрутизації, зручний підхід до взаємодії з базою даних (ORM), підтримує багато програмних пакетів та бібліотек. Фреймворк працює у зв’язці з пакетним менеджером Composer, що також є дуже популярним та якісним інструментом. Він підтримує багато бібліотек, допомагає встановлювати різноманітні залежності та підключити власні кастомні пакети. Laravel працює за принципом MVC (Model-View-Controller), який є досить ефективним методом побудови архітектури додатка.

У якості бази даних було обрано MySQL, так як вона злагоджено та зрозуміло працює разом з PHP та Laravel. Наявність багатьох сутностей та зв’язків у системі приводить до використання MySQL, тому що вона має надійний та зрозумілий функціонал по оперуванням різними типами зв’язків (операція, відома як JOIN).

3.2 Технології розробки фронтенду

Для фронтенд частини використовувалась мова програмування JavaScript, а в якості шаблонів - компоненти Laravel. В основі компонентів шаблонізатор Blade.

Blade Templates — це генератор шаблонів для Laravel, який забезпечує зручний і ефективний спосіб роботи з HTML-кодом і вставлення даних у шаблони.

Шаблони Blade підтримують широкий спектр функцій, таких як умовні вирази, цикли, зв’язування дочірніх шаблонів, імітація тощо. Шаблони можна вкладати один в одного, дозволяючи створювати багатошарові макети та динамічно налаштовувати їхній вміст.

Висновки до розділу 3

Для написання коду програми застосовуються досить популярні, але водночас надійні засоби. Інструменти легкі в застосуванні, пристосовані до взаємодії один з одним, мають велику аудиторію, що полегшує пошук рішень для реалізації функціоналу та розв’язання проблем.

Стек технологій Laravel, MySQL, JavaScript є широко застосованим. PHP та Laravel мають розвинені засоби взаємодії з MySQL. Фреймворк Laravel має свій шаблонізатор Blade, який є досить практичним. В основі Laravel закладена архітектура MVC, що ізолює кожну частину системи та дозволяє розширювати кожну не змінюючи при цьому іншу. У такий спосіб код легко редагувати, додавати новий функціонал та підтримувати роботу системи.

4 ОПИС ОБЛАСТІ ЗАСТОСУВАННЯ

Предметна область системи автоматизованого планування бізнес-процесів контингенту кафедри пов’язана з управлінням навчальним процесом в університеті. Ця система охоплює такі аспекти, як планування навчального процесу, контроль успішності студентів, взаємодія викладачів і студентів, організація комунікацій між викладачами, студентами і батьками.

Система розрахована на використання в університеті і має на меті полегшити процес навчання та забезпечити ефективне спілкування між учасниками навчального процесу. Вона може бути корисною для студентів, які зможуть відстежувати свій розклад і результати навчання, викладачів, які зможуть створювати та оцінювати завдання, та батьків, які зможуть переглядати успішність своїх дітей.

Предметна область також включає в себе планування та організацію навчального процесу, що може бути досить складною задачею для університетської адміністрації. Дана система може допомогти у забезпеченні ефективної організації та планування навчального процесу, зменшенні часових затрат та підвищенні якості навчання. Отже, ця система може забезпечити значний внесок у поліпшення якості навчання та забезпечити успішні результати університетського навчального процесу.

Система планування бізнес-процесів може бути частиною великої платформи, яка в охоплює всі сфери студентського життя та університетських процесів. У перспективі можливо розширити програму, щоб це був модуль, який взаємодіє з іншими сервісами. Тоді це буде злагоджене навчальне середовище, яке не тільки збільшить якість освіти, а також покращить умови роботи викладачів та завідуючих.

На рисунку 4.1 зображена діаграма прецедентів. Вони описує найголовніші функції системи. Адміністратор, який є працівником кафедри, має доступ до усієї

інформації про користувачів. Він має право реєструвати батьків в системі, так як дані про них початково не знаходяться в базах університету.

Рисунок 4.1 – Діаграма прецедентів

Викладач створює завдання та виставляє оцінки, що є вихідною інформацією для батьків та студентів, що спостерігають за цими змінами. На основі цієї інформації студенти планують свій час, в курсі нових задач готових до виконання та своєї успішності. Батьки в свою чергу обізнані про прогрес та успішність своєї дитини.

Студенти та викладачі є головними акторами системи, тому що безпосередньо приймають участь в навчальному процесі. Їм доступний функціонал створення нотаток для кращого сприйняття інформації. Нагадування та сповіщення сприяють тому, що учасники навчального процесу завжди в курсі усіх подій і з меншою ймовірністю пропустять якусь подію. Таким чином, ефективність та продуктивність роботи викладачів та студентів значно збільшується.

Висновки до розділу 4

Розглянута система автоматизованого планування бізнес-процесів контингенту кафедри охоплює широкий спектр аспектів управління навчальним процесом в університеті. Ця система спрямована на полегшення планування, контролю успішності студентів, взаємодії між учасниками навчального процесу та організацію комунікацій.

Вона відповідає потребам університетського середовища, спрощує процес навчання та сприяє ефективній комунікації між студентами, викладачами і батьками. Система може бути корисною для всіх учасників навчального процесу, сприяючи відстеженню розкладу, успішності та взаємодії.

5 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

5.1 Структура бази даних

Основа будь-якої програми - це дані. Інформація надходить в систему, обробляється там, модифікується, відображається і звісно зберігається. База даних обов’язковий елемент будь-якої система. Це набір структурованих даних, які організовуються відповідно до завдання предметної області та цілей, які має виконувати програма. Найважливішим етапом в розробці програмного забезпечення є проектування бази даних. Проектування допомагає уникнути помилок у майбутньому, розбіжностей та неконсистентності даних. Добре спроектована база даних дозволяє легко та швидко розширяти систему та її базу відповідно.

Для даного програмного продукту була обрана реляційна база даних. Така база складається з таблиць, які пов’язані між собою ідентифікаторами або ключами (foreign keys). Формат таблиці зрозумілий для використання, тому така база проста в створенні, редагуванні, пошуці та сортуванні даних.

Реляційна база забезпечує високу надійність та цілісність даних. Завдяки таким інструментам як зовнішні та primary ключі, індекси, обмеження, зручно слідкувати за цілісністю даних та контролювати видалення чи редагування. Прикладом може бути встановлення таких обмежень як cascade on update/delete, restrict on update/delete і т.д. Ці конструкції не дозволяють або обмежують видалення або заміну дані, так як це може порушити зв’язки і призвести до подальших збоїв та помилок в системі.

Використання індексів може збільшити швидкість обробки та сортування даних, а з тим і ефективність роботи програми. Також, реляційні бази достатньо популярні, через що сумісні з багатьма мовами програмування, фреймворками та бібліотеками.

У таких баз звісно є свої мінуси. З їх допомогою важко оперувати великими обсягами даних та даними, які постійно змінюють свою структуру. Для таких випадків краще обрати NoSQL бази даних, які зберігають інформацію у форматі JSON масивів. Прикладами таких БД є MongoDB, Redis.

У даному програмному продукті використання реляційної БД цілком доречне, адже кількість даних помірна і не буде розростатись до величезних розмірів (наприклад, тому що в нас є визначене число студентів, дисциплін, завдань). Велика кількість зв’язків передбачає постійне з’єднання таблиць. Для реляційних баз є чіткий визначений алгоритм та зручні інструменти для таких операцій [11].

Для проектування бази використовується ряд схем та діаграм для опису структури бази даних. Проектування баз даних можна поділити на декілька етапів.

Першим етапом є створення концептуальної моделі даних. Вона полягає в загальному описі представлення предметної області та відображенні основних вимог до продукту.

На двох наступних етапах враховуються особливості СКБД, яка використовується. При проектуванні логічної базі даних застосовуються правила нормалізації. Її ціллю є усунення надлишковості даних, уникнення повторень та загальна оптимізація структури даних. Завершальним етапом виконується опис фізичної моделі даних, яка повністю залежить від засобів СКБД, що застосовується для керування процесами з даними [6].

Концептуальна модель даного програмного продукту зображена на рисунку 5.1. Діаграма показує основні сутності системи та як вони взаємодіють одна з одною, а також найголовніші їхні властивості.

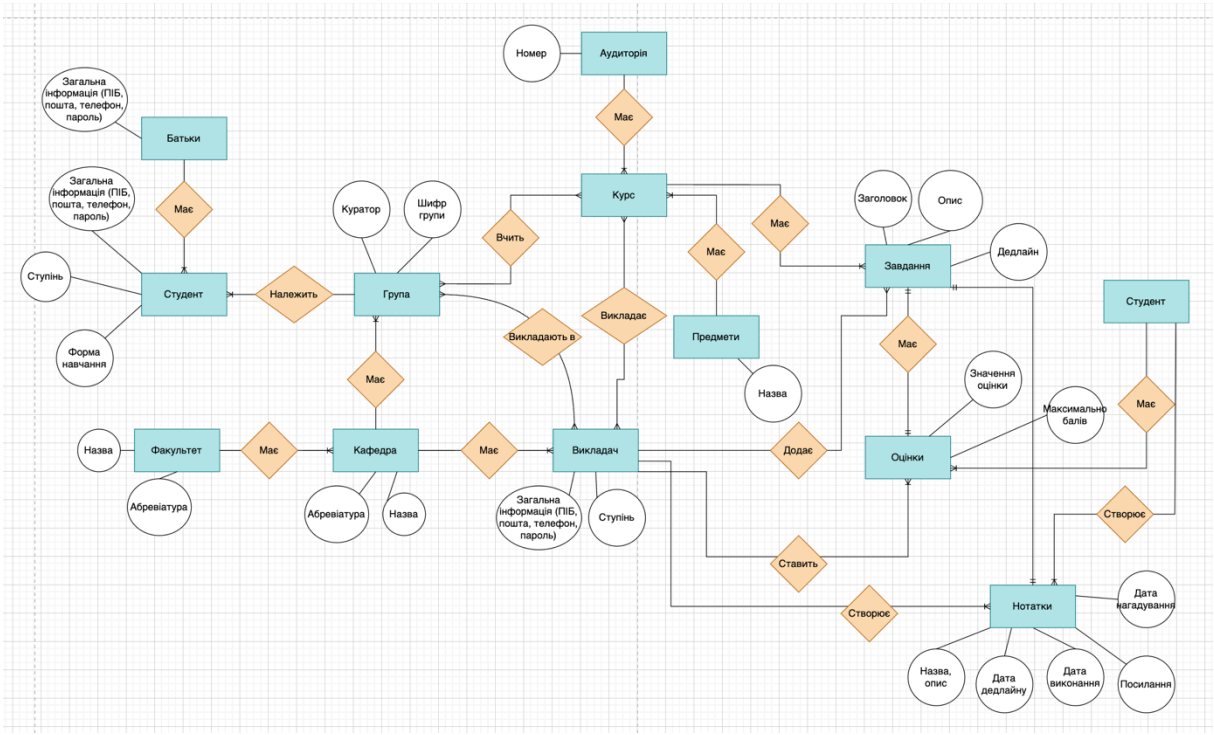

Рисунок 5.1 – Концептуальна модель бази даних

В логічній моделі детальніше розписуються властивості, вводяться такі поняття як зовнішні та первинні ключі. Позначення зв’язків наближене до вигляду реляційних баз даних, як можна бачити з рисунка 5.2. Модель бази даних не відповідає особливостям якоїсь конкретної СКБД. Ця властивість притаманна фізичній моделі бази даних. На рисунку 5.3 можна бачити діаграму для фізичної моделі. Вона більш розширена в порівнянні з логічною та відповідає вимогам обраної СКБД, наприклад, визначенню конкретних даних або загально прийнятого найменуванню змінних.

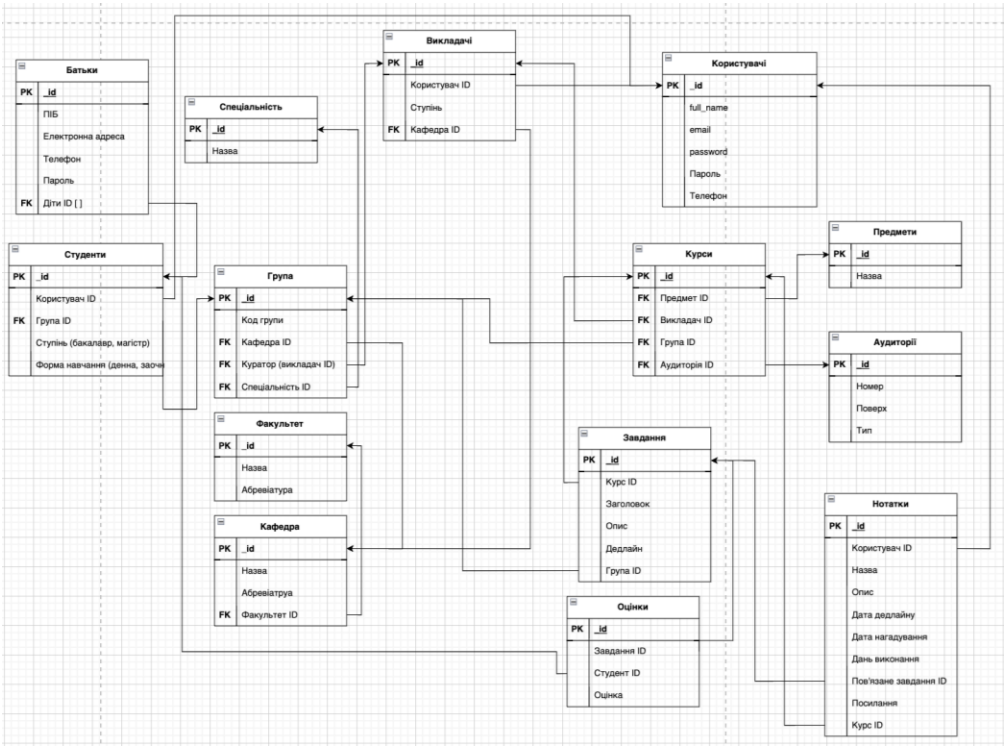

Рисунок 5.2 – Логічна модель бази даних

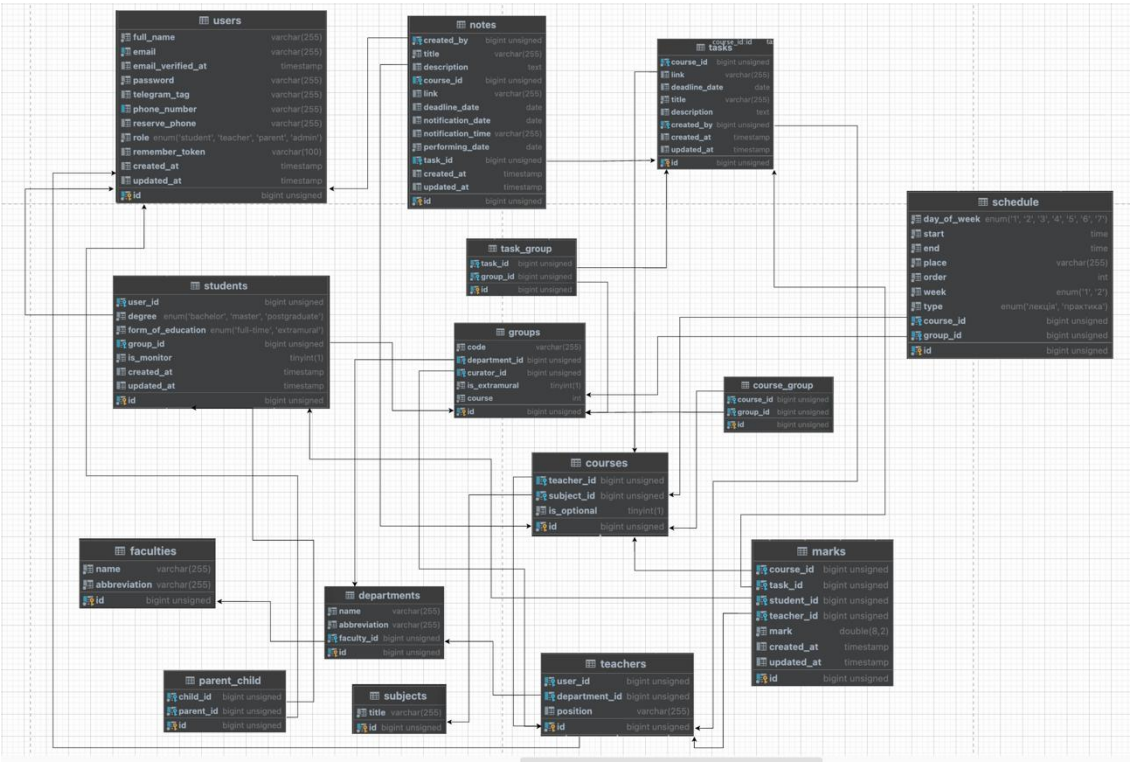

Рисунок 5.3 – Фізична модель бази даних

Ознайомимося зі схемою бази даних програмного продукту.

Основною таблицею є таблиця users. В ній зберігаються дані про всіх користувачів системи, їхня контактна інформація, пароль та логін, а також роль. Роль є дуже важливим полем, адже воно визначає доступи користувача, тобто дозволяє перегляд та оперування одними даними та забороняє доступ до інших.

В системі існує 4 типи доступів:

- студенти - можуть переглядати створені для них завдання та свої оцінки за ці завдання. Створюю нотатки та ставлять нагадування, які приходять у вигляді повідомлень в додатку та на електронну пошту. Бачать розклад занять, аудиторії та викладачів, що ведуть дисципліни;
- викладачі - створюють завдання для студентів, виставляють оцінки за ці завдання, також бачать свій розклад та можуть створювати особисті нотатки з нагадуваннями;
- батьки - мають доступ до системи, прив’язані до своїх дітей та можуть переглядати їхні завдання та оцінки за них;
- адміністратор - бачить у всіх користувачів системи та дані про них. Так як батьки не є офіційними учасниками навчального процесу, тому в базах університету про них може зовсім не буде інформації. Адміністратор відповідальний за створення профілів для батьків, генерування паролів та прив’язування акаунтів дітей до своїх батьків.

Похідними сутностями від користувача є студенти та викладачі. Дані про них зберігаються в таблицях students та teachers. Студенти та викладачі мають головний зв’язок з таблицею користувачів - кожен студент та викладач обов’язково мають відношення до одного користувача.

Головним полем в таблиці студентів є група. Це поле зберігає ідентифікатор групи, який посилається на відповідну таблицю groups. Студент належить до групи, а група може мати багато студентів таким чином формується зв’язок Один до багатьох (One-to-Many). Група у свою чергу пов’язана з кафедрою та факультетом.

Група має куратора, а таблиця зберігає ідентифікатор викладача, який є куратором відповідної групи.

Таблиця teachers по аналогії зі студентами має ідентифікатор користувача, до якого прив’язаний кожен викладач. Викладач також прив’язаних до кафедри та факультета, а також має поле, яке зазначає його посаду.

Варто відзначити розроблене рішення щодо організації таблиць для факультетів та кафедр. Зв'язок між цими двома сутностями визначений як Один до багатьох, оскільки факультет може мати декілька кафедр, а кожна кафедра належить до одного факультету. Ця інформація використовується для визначення належності викладача та групи (а з цього випливає, що і студента) до деякого факультету. Замість зберігання ідентифікатора кафедри та факультету в таблицях groups та teachers, прийнято рішення зберігати значення кафедри та, через наявні зв'язки з факультетом, діставатись до інформації про нього. Таким чином, уникнено можливості внесення неправильної комбінації кафедри та департаменту в таблицю, що може призвести до плутанини або помилок у роботі системи.

Дисципліни зберігаються у таблиці courses. По суті це є комбінацією викладача та курсу, який він може викладати. Список назв курсів зберігається в таблиці subjects. Дисципліни та групи пов’язані зв’язком Багато до багатьох (Many-to-Many). Це викликано ситуацією, в якій група має багато дисциплін та одна дисципліна може викладатись багатьом групам. Для реалізації такого відношення створюється додаткова таблиця course_group, в якій зберігаються ідентифікатори курсів та відповідні ідентифікатори груп, в яких викладається кожна дисципліна.

Наступною великою таблицею є таблиця для зберігання розкладу - schedule. Кожен запис таблиці відповідає одній парі. Пара має такі поля, як день тижня, час початку та завершення, дисципліна, яка викладається, група, до якої відноситься розклад, тип (лекція чи практика), аудиторія (або онлайн, якщо дистанційно), номер по порядку та номер тижня (перший чи другий), якщо розклад складається на два тижні. Поле групи додано безпосередньо до запису про пару для швидшого доступу до інформації, адже фільтрування вибірки по групі - це найпоширеніша операцію і

з такою структурою це робити легше та швидше. В той самий час як зв’язок “курс - course_group - група“ є дещо складнішим, ускладнюватиме код та буде займати час на з’єднання таблиць.

Таблиця tasks зберігає завдання створені викладачами для виконання студентами. Кожне завдання прив’язане до дисципліни, має обов’язкове поле “заголовок” та деякі додаткові поля, такі як опис завдання, посилання на зовнішній ресурс, кінцева дата здачі. Кожне завдання обов’язково прив’язується до того, хто його створив. В даному випадку це буде викладач, бо тільки користувач з такою роллю має право створювати завдання. Так як завдання зазвичай видається на групу, тому доречним було б вказати групу. Зв’язок визначається як Багато до багатьох, так як передбачено, що завдання можна створити для кількох груп (наприклад, для всіх груп потоку), а група має багато завдань. По аналогії з дисциплінами було створено проміжну таблицю task_group.

За кожне завдання викладач повинен виставити оцінку. Оцінки зберігаються в таблиці marks. Кожен запис містить в собі id студента, викладача, завдання та саму оцінку.

Одна з ключових функціональностей системи - це можливість створювати нотатки. Усі нотатки зберігаються в таблиці notes. Запис закріплюється за користувачем незалежно від його ролі. Нотатка може містити інформацію про заголовок, опис того, що треба зробити, посилання на зовнішній ресурс, кінцеву дату здачі, дату та час нагадування. Додатковою функцією є те, що користувач може прикріплювати вже створені завдання та інформація з них автоматично заповниться в нотатку. Після цього користувач зможе відредагувати деякі параметри. Це зручно при підготовці до контрольних або до виконання домашніх завдань. Таким чином, не потрібно дублювати інформацію або довго шукати на інших ресурсах. Потрібно лише знайти завдання в списку та зберегти нотатку з вже заповненими даними.

Користувач з ролю Батьки повинен мати тільки інформацію про свою дитину. Тому всі дані, що стосуються дисциплін, завдань та оцінок, мають діставатись через

зв’язок з дитиною. Інформація про батьків зберігається лише в таблиці users, оскільки такі користувачі не мають більше специфічних даних, крім контактної інформації. Програма передбачає випадок, що батьки можуть мати декілька дітей, які вчаться в одному університеті. Хоч це і трапляється рідко, проте є цілком можливим. Також на платформі можуть зареєструватись двоє батьків дитини. Враховуючи усі сценарії було прийнято рішення про створення відношення Багато до багатьох між дітьми та батьками (який по суті можна охарактеризувати як декілька до декількох). Виходячи з цього створюється проміжна таблиці parent_child, яка визначає, що студент прив’язаний до своїх батьків. Варто зазначити, що child_id у таблиці parent_child вказує на запис про студента. Це зумовлене тим, що більшість інформації про навчальні процеси стосується саме студента. В той час, коли сутність Користувач зберігає в собі контактні дані та дані для авторизації і аутентифікації.

В базі даних також присутні допоміжні таблиці, які не стосуються предметної області. Таблиця migrations - таблиця, яку використовує Laravel для відстеження та керування версіями бази даних. Кожен раз, коли ми запускаємо міграцію в перший раз, вона додається в таблицю migrations. Якщо міграція вже була запущена, Laravel її ігнорує. Це дозволяє запобігти дублюванню таблиць та даних в базі даних.

Крім того, таблиця migrations зберігає поточний стан бази даних, що дозволяє змінювати базу даних у майбутньому, додавати нові міграції та відкочувати зміни, які було зроблено раніше. Це робить процес розгортання та керування базою даних в Laravel більш простим та гнучким.

Таблиця notifications потрібна для зберігання даних про нагадування та повідомлення. Це також допоміжна таблиця Laravel, яка працює разом з функціоналом Notification [1]. В ній зберігається клас відповідальний за кожен тип повідомлення та дані які передаються та використовуються для формування повідомлення або нагадування.

Таблиця jobs створюється при для відслідковування та зберігання черг. Черги використовуються для обробки завдань в асинхронному режимі або на фоні. Черга може обробляти декілька задач одночасно. Відповідальними за виконання таких задач є Jobs. Дані про них зберігаються в таблиці jobs, а саме клас, який виконує задачу, канал, дані, якими оперує цей клас [1].

5.2 Опис архітектури програми за шаблоном MVC

В даній роботі використовувався архітектурний шаблон проектування програмного забезпечення MVC (Model-View-Controller). Laravel є прикладом фреймворку, який побудований по цьому шаблону. Підхід полягає у розділенні програми на три частини:

- модель (Model) - це компонент, який зберігає дані та їхню структуру. Зазвичай через цей компонент здійснюються звернення до бази даних. Модель не залежить від інших компонентів та може бути перевикористана в різних місцях системи;
- представлення (View) - компонент, який відповідає за відображення даних. Прикладом представлень є шаблони, HTML-сторінки, Vue компонента і т.д. Представлення залежить від моделі, оскільки відображає дані, які вона містить в собі;
- контролер (Controller) - компонент, що відповідає за скерування запитів та їх обробку. Він залежить від моделі, бо взаємодіє з її даними та формує масиви з даними в потрібному форматі. Також контролер залежить від представлення, але не від його форми. В цілому задача контролера обробляти запит та керувати іншими діями, такими як звернення до сервісів, репозиторіїв і т.д.

Застосування шаблону MVC дозволяє зменшити зв'язок між компонентами програми, що полегшує розробку та підтримку коду. Крім того, цей шаблон

дозволяє відокремити бізнес-логіку від логіки відображення, що покращує тестування та зручність обслуговування коду [3].

5.3 Реалізація моделей в системі

Для кожної сутності програми прийнято створювати модель. З її допомогою зручно робити запити в базу даних використовуючи компонент Laravel - ORM (object-relational mapper). Принцип полягає в використанні об’єктів для взаємодії з базою даних та виконанні базових операцій таких як, вставка, редагування, видалення, без написання SQL запитів. Такий підхід є досить зручним, забезпечує читабельність коду та зменшує його кількість [1].

Чи є такий підхід універсальним? На жаль, ні. В такий спосіб є можливість писати невеликі та прості запити, що стосуються зазвичай однієї таблиці. Для запитів з двома і більше таблиць, де потрібно використовувати JOIN, використовувались більш низькорівневі інструменти, такі як, QueryBuilder.

Перед тим, як перейти до використання моделей для діставання даних, варто описати процес створення зв’язків між різними моделями.

5.3.1 Типи зв’язків між сутностями та їх реалізація

Зв’язки в Laravel — це спосіб встановлення зв’язку між таблицями в базі даних. Laravel надає кілька типів зв’язків: один до одного, один до багатьох і багато до багатьох.

Кожна модель в Laravel успадковує клас Model , який в свою чергу має декілька трейтів та інтерфейсів. Одним з таких трейтів є HasRelationships, який відповідає за реалізацію зв’язків.

Прикладом зв’язку один до одного є відношення користувача до таблиці його ролі: студента або викладача. Користувач може тільки бути або студентом, або

викладачем. Користувач має один акаунт в системі. Тому студенту або викладачу є тільки один відповідник в таблиці users.

У Laravel такий зв’язок реалізується за допомогою метода HasOne. Для того, щоб пов’язати дві моделі, потрібно в обох класах визначити метод, що діставатиме дані про протилежний об’єкт. Наприклад, модель Студент діставатиме екземпляр відповідного йому користувачу через власний метод user(), а модель Користувач навпаки - через власний метод student().

Варто зазначити, що все залежить від побудови таблиці, тобто від її проектування, а саме від визначення зовнішніх ключів та додаткових таблиць. Якщо все зроблено правильно, фреймворк сам автоматично пов’яже дані після визначення зв’язку в моделях. Проте можливі ситуації, коли потрібно відхилитись від звичного найменування зовнішніх ключів. У цьому випадку варто лише прописати нові назви зовнішніх та primary ключів в методі hasOne. Надалі така логіка стосуватиметься всіх типів зв’язків та методів, що їх реалізують.

Наступний тип зв’язку, один до багатьох, є одним з найпоширеніших. Так пов’язані сутності Кафедра та Факультет, бо Кафедра належить до одного факультету, в той час як Факультет може мати декілька кафедр. За аналогією до зв’язку один до одного, в трейті HasRelationships існують методи hasMany та belongsTo, які прописуються відповідно у класі, що відповідає за єдину сутність, та в класі, що представляє більшість екземплярів. Наприклад, Факультет матиме hasMany, а Кафедра - belongsTo.

Тип зв’язку багато до багатьох часто використовується в даній системі. Це, наприклад, зв’язок між групами та курсами, батьками та дітьми. Цей зв’язок є дещо складнішим в реалізації. Для цього потрібно створити проміжну таблицю, де потрібно зберігати різні комбінації між сутностями. Після цього за аналогією до інших зв’язків використовуються допоміжні методи. В даному випадку це belongsToMany.

Крім базових зв’язків, Laravel пропонує ще низку методів, що спрощує роботу, якщо потрібно позначити складніше структуровані відношення.

Прикладом може бути зв’язок між викладачем та розкладом. Викладач безпосередньо немає відношення до нього. Проте він викладає деяку дисципліну, що прив’язана до розкладу. Як було зрозуміло з описаних раніше реалізацій зв’язків, такий підхід зручно використовувати, якщо дві сутності взаємодіють безпосередньо одна з одною. У випадку з викладачем та розкладом, сутності взаємодіють через ще одну проміжну - Дисципліна. В такому випадку Laravel пропонує функціонал зв’язку hasManyThrough. Він полягає в зв’язку одних класів через третій проміжний. Варто зазначити, що, для того, щоб зв’язок працював потрібно коректно побудувати структура таблиць та зовнішніх ключів. Враховуючи це, курс повинен мати зовнішній ключ, що вказує на викладача, а розклад повинен бути пов’язаний з курсом.

5.3.2 Управління даними за допомогою репозиторіїв та моделей

У програмі використовується архітектурне рішення, що включає в себе використання репозиторіїв.

Репозиторії в Laravel — це шаблон проектування, який використовується для відділення логіки роботи з даними від логіки програми. Використовуючи репозиторії, ви можете зменшити кількість коду в контролерах і моделях, що робить програму більш структурованою та зручною для обслуговування.

Зазвичай використовуються два типи сховищ: для роботи з базою даних і для роботи з зовнішніми API. Репозиторії баз даних містять методи для створення, читання, оновлення та видалення записів бази даних. Репозиторії для роботи із зовнішніми API містять методи взаємодії з API, наприклад, для отримання, створення, оновлення та видалення ресурсів.

Основна ідея репозиторіїв полягає в тому, що вони виконують логіку доступу до даних в окремому місці, тому вони є більш гнучкими та легшими для тестування. Крім того, репозиторії можна повторно використовувати в різних частинах

програми, що дозволяє уникнути дублювання коду та зменшити його складність [5].

Базові методи ORM схожі на звичайні операції SQL запитів. Найбільше хотілось би відмітити використання метода with().

Метод with() у Laravel використовується для попереднього завантаження, пов’язаного з даними моделі. Це означає, що замість того, щоб робити окремі запити до баз даних для отримання пов’язаних даних, ви можете завантажити їх усі одночасно, що може значно підвищити продуктивність вашої програми.

Завантаження зв’язаних даних виконується, тому що Laravel використовує «ледаче» завантаження даних за замовчуванням, що означає, що зв’язані дані не будуть завантажені, доки на них не буде посилання в коді. Це може призвести до проблеми з продуктивністю, особливо у випадках, коли потрібно виконати багато запитів до бази даних.

Використання методу with() дозволяє попередньо завантажувати дані, пов’язані з пам’яттю, що може зменшити кількість запитів до бази даних і збільшити швидкість вашої програми. Далі метод with() дозволяє завантажувати пов’язані дані в одному запиті до бази даних за допомогою класу Eloquent.

5.4 Авторизація та аутентифікація

Авторизація та аутентифікація є двома поняттями, пов'язаними з ідентифікацією та доступом користувачів до системи.

Аутентифікація - це процес перевірки і підтвердження ідентичності користувача. Під час аутентифікації користувач надає свої ідентифікаційні дані (наприклад, ім'я користувача і пароль), і система перевіряє ці дані, щоб з'ясувати, чи вони вірні. Успішна аутентифікація дозволяє користувачу увійти до системи і мати доступ до обмежених ресурсів або функцій.

Наприклад, в контексті даної системи, аутентифікація відбувається, коли студенти, викладачі або батьки надають свої облікові дані (наприклад, ім'я користувача та пароль) для входу до системи.

Авторизація - це процес визначення та надання прав доступу користувачеві після успішної аутентифікації. Після того, як користувач успішно пройшов аутентифікацію, система визначає його рівень доступу та дозволяє виконувати певні операції, переглядати або змінювати дані відповідно до його ролі або повноважень.

В даній системі після аутентифікації студенти матимуть доступ до функцій, таких як перегляд розкладу, створення нотаток, перегляд завдань та оцінок. Викладачі матимуть доступ до функцій, таких як створення завдань, редагування завдань, виставлення оцінок тощо. Батьки матимуть обмежений доступ, що дозволяє їм переглядати завдання та оцінки своїх дітей.

Нотатки можуть переглядати тільки користувачі, що їх створили. Студенти можуть переглядати завдання, які були створені для їхньої групи, викладачі – завдання, створені ними, а батьки – завдання, що створені для їхніх дітей. Це все також є прикладом обмеження доступів, що є задачею авторизації. На рисунку 5.4 зображено діаграму послідовності дій системи при авторизації та аутентифікації користувача.

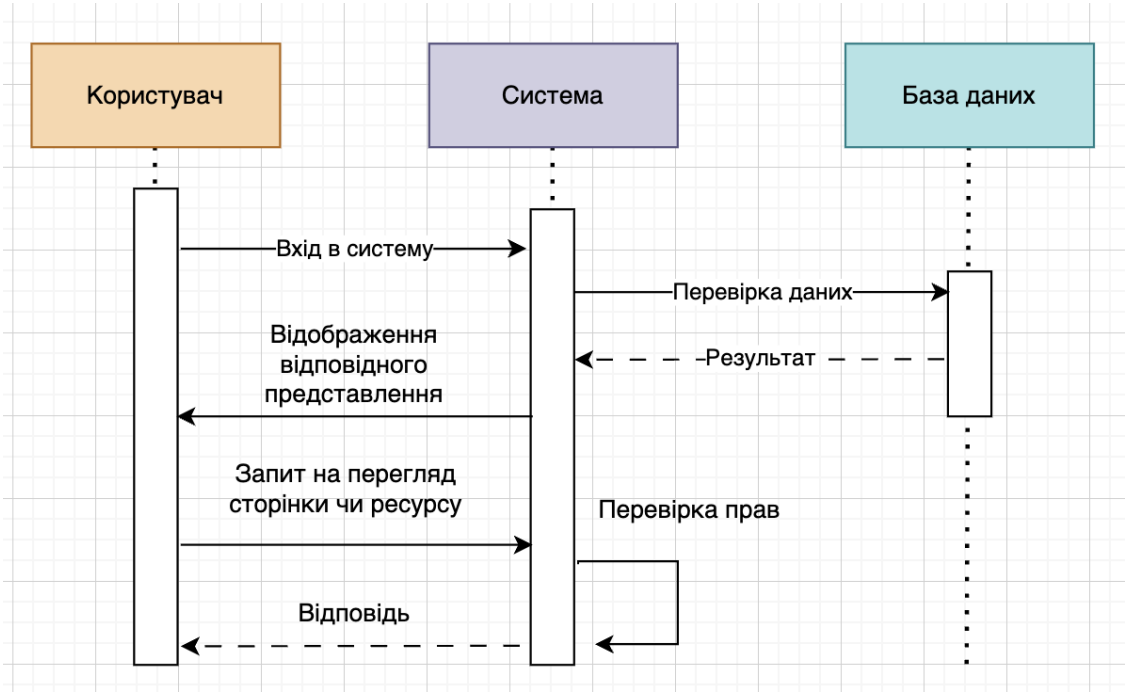

Рисунок 5.4 – Діаграма послідовності процесів авторизації та аутентифікації

Таким чином, аутентифікація перевіряє ідентичність користувача, а авторизація надає права доступу після успішної аутентифікації. Ці два процеси є важливими для забезпечення безпеки та контролю доступу до системи та її ресурсів.

Для реалзіації аутентифікації та авторизації використовувався вбудований пакет фреймворка Laravel - Auth. Цей пакет дозволяє легко встановити і налаштувати систему аутентифікації для різних типів користувачів. Auth забезпечує механізми для аутентифікації користувачів з використанням різних джерел ідентифікації, таких як база даних, Eloquent моделі, API-токени тощо. Auth дозволяє визначати права доступу для різних ролей користувачів. Є можливість встановлення правил авторизації для обмеження доступу до певних роутів або контролерів в залежності від ролей користувачів. Auth забезпечує можливість зберігання ідентифікатора сесії користувача та збереження інформації про аутентифікацію в сесіях. Це дозволяє визначати стан авторизації користувача і зберігати дані, пов'язані з аутентифікацією, наприклад, ім'я користувача чи роль.

Після налаштування Auth стають доступними функції, такі як Auth::check() для перевірки авторизації користувача, Auth::user() для отримання поточного аутентифікованого користувача, Auth::logout() для виходу з системи тощо.

Auth в Laravel - потужний інструмент для роботи з аутентифікацією та авторизацією в вашому додатку, який забезпечує безпеку та контроль доступу для різних типів користувачів.

5.5 Використання middleware

Middleware - це спеціальні класи, яки виконують деякі дії перед виконання запитів та передачі даних до контролера. Це може бути авторизація користувача та перевірка його прав, перевірка введених даних або захист від атак. Особливість middleware в Laravel полягає в тому, що існує можливість додати декілька класів до одного маршруту, що дозволяє провести комплексну перевірку.

У програмі middlewares в основному використовуються для того, що б перевірити чи має користувач доступи до деяких частин програми. Це забезпечує деякий шар безпеки, оскільки система контролює, чи зареєстрований користувач, чи має він дозволи на перегляд окремих даних.

Authenticate middleware перевіряє чи зареєстрований користувач. Він є вбудованим класом Laravel та працює у зв’язці з методами авторизації та аутентифікації, які надає фреймворк [1].

В додаток до нього ще були створені класи для визначення чи є користувач студентом,викладачем або адміністратором, щоб обмежити доступ для тих користувачів, хто немає потрібною ролі для перегляду окремих сторінок. Наприклад, студент не повинен мати доступ до сторінки з оцінками або створення завдань. Це може призвести не тільки до виникнення помилок в програмі, але й до витоку даних, їх небажаної зміни чи видалення. Так само як звичайний користувач не повинен мати доступу на панелі адміністратора, так як там зберігаються важливі дані про всіх користувачів системи.

Було створено ще один додатковий Middleware на перевірку прав доступу до завдань. Це створено для того, що користувач могли бачити лише завдання, які належать їм.

5.6 Застосування класів Request та валідація даних

У Laravel запити — це класи, які дозволяють перевіряти та обробляти дані, надіслані на сервер. Запити допомагають забезпечити правильну та безпечну обробку даних, отриманих від користувачів.

Основна мета запитів — забезпечити захист від збирання даних із форм і перевірити введені користувачем дані. Запити також забезпечують більшу гнучкість і легкість рефакторингу коду, оскільки код перевірки та обробки вхідних даних відокремлений від коду контролера та моделі.

У Laravel запити можуть мати різні правила перевірки, які можуть перевіряти наявність даних, правильне форматування даних, їх тип, довжину та інші параметри. Зазвичай це застосовується при отриманні даних з форми надісланих POST методом [1]. Якщо дані не проходять перевірку, програма повертає всі дані та масив з помилками. Після цього їх можна відобразити на сторінці представлення та повідомити користувача, що введені ним дані не відповідають визначеним умовам чи надіслані в неправильній формі. Laravel дозволяє прописувати власні повідомлення на кожне правило валідації, і таким чином інформувати про наявні помилки.

Запити можна використовувати в контролерах, де їх можна приймати в параметрах методів, які отримують дані від користувача. Запити також можна використовувати в маршрутах, де їх можна використовувати для перевірки та обробки даних, надісланих через форми.

Крім того, у Laravel Requests можна використовувати для захисту від атак Cross-Site Request Forgery (CSRF). Фреймворк автоматично додає захист CSRF до

всіх запитів POST, які отримує ваша програма. Це захищає вашу програму від атак CSRF, які можуть призвести до її зламу.

5.7 Реалізація сповіщень та повідомлень

Одна із задач системи - інформувати користувача про дії, що відбуваються в додатку. Для цього використовуються оповіщення, яке надсилається користувачу в самому додатку та на електронну адресу. В системі існують такі типи сповіщень:

- повідомлення про створення нового завдання викладачем;
- повідомлення про виставлення оцінки за завдання;
- нагадування, якщо таке було встановлене при створенні нотатки.

5.7.1 Використання Listeners та Events для створення сповіщень

Одним із широко застосованих методів надсилання сповіщень є створення подій та оброблення її за допомогою слухача. Застосування цього способу полягає в відслідковуванні потрібної дії та виконувати деякі функції після цього.

Після дії, яка повинна спричинити надсилання повідомлення, потрібно викликати подію (Event). У Laravel події — це спосіб ініціювати певні дії чи поведінку, коли в програмі відбуваються певні речі, наприклад створення нового користувача або виконання певного завдання.

Події в Laravel зазвичай визначаються як клас, який розширює базовий клас Illuminate\Foundation\Events\Event. Вони містять інформацію про подію, що сталася, і можуть використовуватися для передачі додаткових даних між різними частинами програми.

Слухачі подій (Listeners) — це класи, які обробляють події. Ці прослуховувачі зареєстровані в класі EventServiceProvider і визначають дії, які слід виконати, коли подія запускається [2].

Загалом події в Laravel забезпечують потужний механізм для відокремлення різних частин вашої програми та надання їм можливості гнучкіше та модульніше взаємодіяти.

Після того, як викладач створює завдання або виставляє оцінку, викликається подія, яка в свою чергу має відповідний слухач, який виконує деякі операції. На рисунку 5.4 можна переглянути діаграму послідовності, яка демонструє процес створення викладачем завдання.

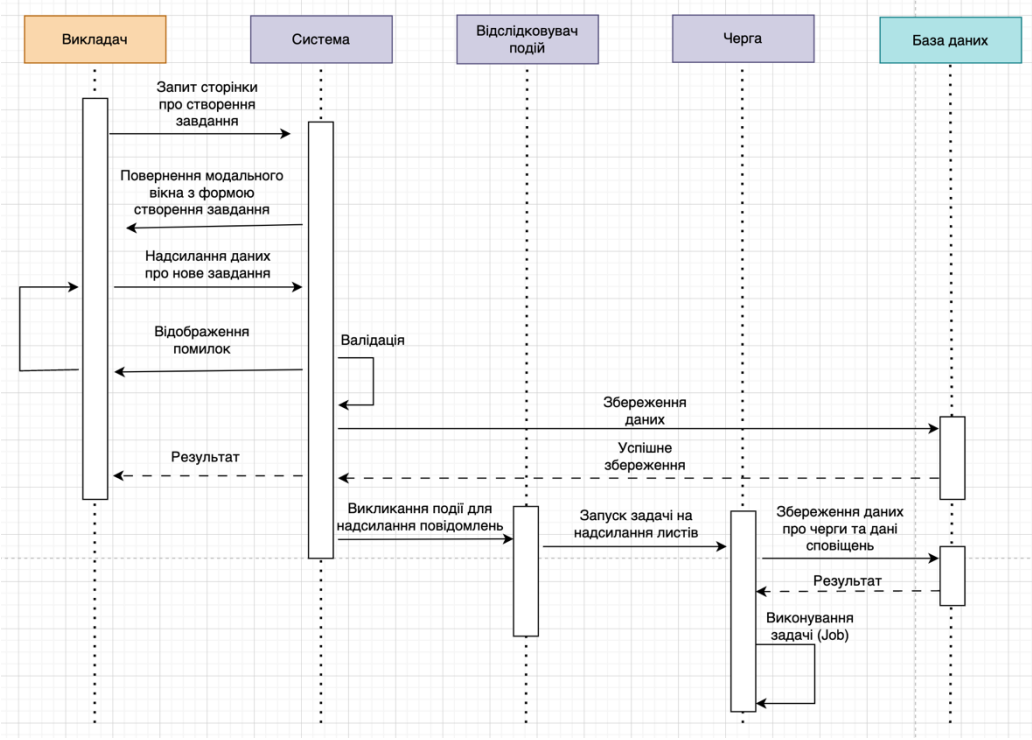

Рисунок 5.5 – Діаграма послідовності для створення домашнього завдання

Отже, після визначення слухача, пропишемо тут код, який виконуватиме розсилання повідомлень.

У програмі надсилання сповіщень кільком користувачам електронною поштою чи іншим способом може бути трудомістким завданням, яке може уповільнити роботу користувача. Якщо сповіщення потрібно надіслати великій кількості користувачів, це може зайняти багато часу, і навіть може призвести до збою програми або тайм-ауту.

Використовуючи чергу для керування надсиланням сповіщень, програма може надсилати сповіщення у фоновому режимі, не впливаючи на роботу користувача. Сповіщення додаються до черги, яка є списком завдань, які потрібно виконати. Потім робочий процес може обробити чергу, надсилаючи сповіщення одне за одним, звільняючи програму для виконання інших завдань [4]. У Laravel цей процес досить легко налаштовується та реалізується за допомогою Jobs. Але в чергу можна додати будь-який процес, що займає певну кількість часу, як Job або Notification [1].

У Laravel Job (або завдання) — це одиниця роботи, яка може виконуватися асинхронно у фоновому режимі веб-додатку. Це дозволяє довгострокові завдання, такі як надсилання електронних листів або обробка великих обсягів даних, виконувати поза основним циклом запит/відповідь, що може допомогти покращити продуктивність і швидкість реагування програми.

Завдання зазвичай визначаються як класи PHP, які реалізують інтерфейс Illuminate\Contracts\Queue\ShouldQueue, який дозволяє додавати їх до черги. Коли завдання відправляється, воно серіалізується та додається до черги, а потім процесор завдань отримує та виконує його асинхронно в окремому процесі [1].

У Laravel черга — це спосіб відкласти обробку трудомісткого завдання або роботи на потім, щоб звільнити ресурси для інших завдань, які потрібно виконати негайно.

Коли завдання додається до черги, воно зберігається в системі черги повідомлень, яка керує обробкою завдань у фоновому режимі. Тоді система черги забере завдання з черги та виконає його пізніше, як правило, за допомогою окремого робочого процесу.

Використання черги може допомогти підвищити продуктивність і надійність програми, гарантуючи, що довгострокові завдання не зв’язують ресурси, необхідні для інших запитів, і забезпечуючи спосіб автоматичного повторного виконання невдалих завдань.

У Laravel є можливість створити декілька каналів черг. У системі є два типи сповіщень: у самому додатку та на електронну пошту. Відповідно для кожного виду сповіщень визначимо з’єднання, в яке має попасти сповіщення. Це дозволить розгрузити потік процесів та зменшити навантаження, адже лист може надсилатись кільком десяткам людей одночасно.

У системі існує з’єднання за замовчування, або «default», та канали «app» та «emails», відповідно для надсилання листів в додатку і на пошту. На локальному сервері запускаємо слухач черг, який відслідковує та виконує процеси, що запускаються в програмі. Для цього використовується команда Artisan “php artisan queue:work --queue=default,app,emails”. Уся інформація про процеси зберігається у таблиці бази даних “jobs”.

5.7.2 Надсилання електронних листів

Фреймворк Laravel надає зручний та гнучкий інструмент для створення та надсилання електронних листів. З його допомогою можна надсилати листи на фоні з використанням черг, легко розширювати наявний код та легко формувати структуру листа і створювати ієрархію для окремих частин програми.

Щоб почати використовувати інструмент створення електронних листів, потрібно налаштувати змінні у .env файлі. Там можна вказати різні драйвери та параметри з’єднання для надсилання листів.

Для тестування надсилання листів використовувався онлайн-сервіс Mailtrap. У .env файлі проекту встановлюються параметри для підключення. Наприклад, драйвер, хост, пароль та логін для авторизації в сервісі. Також можна визначити електронну пошту за замовчуванням, з якої будуть приходити усі листи. У даному випадку це загальна пошта університету, що буде прив’язана до системи.

SMTP (Simple Mail Transfer Protocol) – це протокол надсилання електронних листів через мережу. Він дозволяє встановити з'єднання з сервером електронної пошти за допомогою протоколу SMTP і передавати електронні листи через це

з'єднання. Наприклад, до сервісу перехоплення листів, як Mailtrap [12]. Використання драйвера SMTP у Laravel дозволяє надсилати електронні листи з вашого додатка, використовуючи сервер електронної пошти за вказаними параметрами. Драйвер SMTP забезпечує надійну доставку листів до поштових серверів отримувачів.

Після налаштування з'єднання та драйверів, можна переходити до створення листів. Клас Mailable має багато методів, за допомогою яких зручно складати лист. Базовими властивостями для листа є тема, відправник та отримувач. Отримувач визначається шляхом передачі моделі користувача до метода Mailable - to(). При цьому обов'язковою умовою є наявність поля email з коректним значення адреси для кожного користувача.

Для власного шаблону листа використовуються blade-шаблони Laravel. Визначити вид кожного листа можна за допомогою метода view. Передати параметри до шаблона можна використовуючи метод with().

При надсиланні електронного листа задача скеровується до з'єднання "emails". Таким чином, відокремлюючись від інших процесів, що забезпечує надійність передачі, більшій швидкості виконання та швидший пошук помилки в разі її виникнення.

Концепція роботи з листами в Laravel дуже схожа на інші процеси, такі як створення представлень та передавання в них даних з контролера. Тому опанування способами надсилання листів є надзвичайно простим, зрозумілим та швидким.

5.7.3 Написання консольних команд та їх запуск за розкладом

У системі передбачено надсилання нагадувань про виконання нотаток та сповіщення студентів, що наближається строк здачі завдання. Таким чином, задача полягає в постійному відстеженні чи не наблизилась потрібна дата. У даній системі цей процес виконується за допомогою консольних команд.

Створити консольну команду можна за допомогою Artisan за анлогією до створення усіх програмних структур, що використовуються в програмі: “php artisan make:command ComandName”. Обов’язковим параметром в команді є signature, яка відповідає за назву команди. Ця назва визначає, яку команду потрібно запустити. У методі handle виконується основний код.

Головною особливістю команди є можливість запланувати її виконання в потрібний момент або періодично. Для цього Laravel має вбудований функціонал, що відповідає за виконання команд за деяким розкладом. Час можна виставити будь-який – від виконання кожну хвилину до запуску команди один раз на день, тиждень чи місяць.

Розглянемо наявний функціонал на прикладі команди, що повідомляє про початок роботи над деякою нотаткою. Нагадування про здачу завдання буде працювати аналогічно. Алгоритм роботи такий: команда запускається кожну хвилину, щоб звіряти час визначений в записі окремої нотатки. Код команди перевіряє чи є заданий час пізніше поточного або дорівнює йому і якщо так надсилає повідомлення користувачу. Такий підхід допоможе не пропустити ні одне сповіщення.

5.7.4 Синхронізація з API університету

Спершу варто розібратись, що таке API.

API (Application Programming Interface) - це набір правил та протоколів, які визначають, як різні програмні компоненти мають взаємодіяти між собою. API визначає набір функцій, методів, класів та структурних правил, які дозволяють програмним компонентам обмінюватися даними та взаємодіяти один з одним.

Через використання API, розробники можуть використовувати готовий код, функції та можливості, що надаються іншими програмними компонентами, замість написання всього з нуля. API дозволяє розширювати функціональність програм та

створювати інтеграції між різними системами для забезпечення більш гнучкої та ефективної розробки програмного забезпечення.

Одне із завдань системи полягає в використанні вже існуючих університетських даних з офіційних джерел та актуалізації інформації у власній системі. Так як програма отримує на вхід дані про розклад, в системі існує реалізація постійного стягування свіжих даних. Цей процес можна описати одним словом – «синхронізація». Синхронізація також реалізується запланованою командою, тільки виконується рідше. Це зумовлене тим, що дані про розклад є більш статичними на відміну від нагадувань та рідко потребують актуалізації. Проте виконувати його потрібно періодично, щоб уникнути використання застарілих даних. Сам процес синхронізації може бути довготривалим, так як задача полягає в отриманні великої кількості даних. Тому цей процес варто виконувати на фоні основної програми, щоб він не блокував виконання інших задач та був непомітним для звичайного користувача.

Отримання даних здійснюється з API маршруту: schedule.kpi.ua/api/schedule/lessons. Щоб отримати дані про розклад групи, потрібно передати обов’язковий параметр groupName або groupId.

5.8 Обробка виключень та логування помилок

Обробка помилок є обов’язковою та дозволяє реагувати на виникнення збоїв чи проблем. Основою такої обробки становлять виключення (Exceptions) та логування. Обробка виключень дозволяє відловлювати помилки в місцях найімовірнішого їхнього виникнення та взяття відповідальності на себе. Це включає в себе відображення сторінки з помилкою, інформування про можливі причини збою, відміна поточних змін, якщо мова йде про запит до БД, а також логування.

Логування - це процес збирання та записування інформації про різні події, що відбуваються в програмі. Логування є важливим інструментом для розробників,

щоб зрозуміти, що відбувається в системі, особливо якщо виникають проблеми. Використання логів може допомогти вам виявити та виправити помилки, а також зрозуміти, як користувачі взаємодіють із системою. Існують різні рівні логування для різних типів повідомлення, такі як ‘error’, ‘info’, ‘warning’. Це дозволяє легко розрізняти повідомлення та не тільки бачити, де виникла помилка, але і які дії користувача призвели до неї.

5.9 Реалізація фронтенду

Фронтенд в системі відіграє важливу роль у забезпеченні зручного та естетичного інтерфейсу для користувачів. Задача фронтенда полягає в тому, щоб зробити веб-додаток або веб-сайт доступним, привабливим та легким у використанні. Цей аспект розробки фокусується на тому, як користувачі взаємодіють з інформацією, яка надається системою, і як їм комфортно користуватись її функціоналом. Важливість фронтенду в системі полягає в тому, що він впливає на перший враження користувачів та їхню спроможність ефективно взаємодіяти з системою.

За допомогою зручного та інтуїтивно зрозумілого інтерфейсу, користувачі легко зможуть знайти необхідну інформацію та виконати потрібні дії. Це покращує їх задоволення від використання системи та забезпечує більш високу ймовірність повернення до неї у майбутньому.

Наявність привабливого та зручного фронтенду є важливим фактором в конкурентному середовищі. Він може надати перевагу вашій системі порівняно з іншими, залучити більше користувачів та сприяти позитивному іміджу вашої компанії.

Добре структурований фронтенд забезпечує легкість розширення та модифікації системи у майбутньому. Це дозволяє забезпечити гнучкість та швидкість реагування на зміни вимог користувачів або бізнес-процесів.

5.9.1 Використання Bootstrap

Bootstrap – це найпопулярніший фреймворк для HTML розмітки та CSS стилів. Принцип роботи з Bootstrap полягає в використанні готових рішень, стилів та JavaScript реалізацій у власному проекті. Він реалізовує велику кількість компонентів, що часто зустрічаються у фронтенд розробці, наприклад, акордеони, меню-бургер, модальні вікна, вкладки, слайдери та ін. Усі компоненти та стилі вже мають властивість адаптивності за замовчуванням, тобто вони пристосовані до мобільних та планшетних версій. За допомогою Bootstrap швидко та легко створювати розмітку сторінок, так як він має широкий вибір параметрів для налаштування сіток та контейнерів.

Можна подумати, що сам по собі цей інструмент є не зовсім гнучким, обмежує створення власних стилів та є складним в розширенні, проте це не так.

Однією з головних переваг використання фреймворка є зменшення кількості роботи та зусиль. Базові компоненти, що реалізовані в Bootstrap, широко застосовуються у веб-програмуванні і є невідмінною частиною користувацького інтерфейсу. Тому замість того, що витрачати час на створення рішення, що вже існує, доцільно використовувати готовий шаблон.

По-друге, Bootstrap надає зручні можливості для розширення та налаштування, що дозволяє розробникам вносити зміни в стилі і компоненти, а також створювати власні стилі на його основі. Вони можуть кастомізувати свої налаштування перед збирання фреймворку, а саме обрати необхідні компоненти, налаштувати кольори, сітку та інші параметри. Стилі Bootstrap також можна перезаписати, так як фреймворк має чітку структуру класів. Класи можна перезаписувати або додавати свої і таким чином, змінювати вигляд компонентів. Для цього можна використовувати як звичайний CSS, так і його препроцесори SASS та LESS, які дозволяють розробникам писати стилізований код більш ефективно та організовано. Вони розширюють можливості звичайного CSS,

додаючи нові функції та синтаксичні конструкції. При розробці програмного продукту використовувався препроцесор LESS [7].

Усі ці можливості роблять Bootstrap гнучким фреймворком, який дозволяє легко вносити зміни, розширювати його та створювати власні стилі на основі його компонентів.

Для створення цілісної та усюди однакової системи контейнерів та компонування у програмі було використано Bootstrap контейнери для сітки. За допомогою класів визначаються параметри та розміри екранів. За допомогою особливих класів створюються складніші компоненти, такі як вкладки та акордеон. На рисунку 5.5 представлена реалізація вкладок за допомогою Bootstrap на сторінці «Повідомлення», а на рисунку 5.6 – реалізація акордеону для списку предметів викладача. Обидва компоненти стилізовані відповідно до решти елементів користувацького інтерфейсу. Це доводить, що застосування Bootstrap не обмежує розробника в зміні стилів та їх кастомізації.

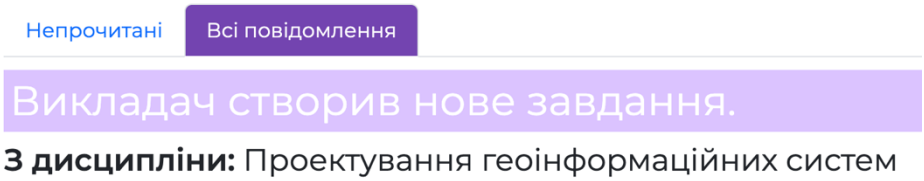

Рисунок 5.6 – Вигляд компонента «Вкладки»

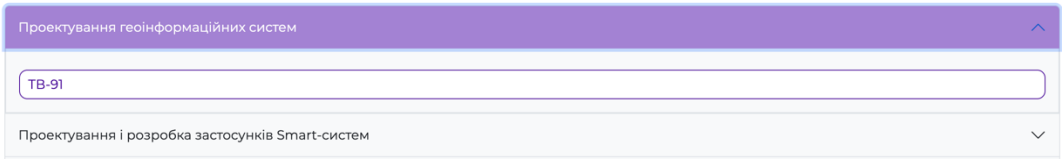

Рисунок 5.7 – Вигляд компонента «Акордеон»

5.9.2 Використання значків та іконок

Іконки можуть служити як універсальний мовний засіб, що сприяє легкому сприйняттю та розумінню інтерфейсу користувачем. Вони можуть швидко передати певні дії, об'єкти або концепції, зменшуючи необхідність у додаткових

поясненнях або текстових описах. Використання іконок у панелі навігації або меню дозволяє створити компактну та інтуїтивну навігацію. Користувачі швидко розпізнають іконки та можуть ефективно переміщатись між різними частинами системи або веб-сайту. Іконки можуть додати естетичний ефект до дизайну інтерфейсу, зробити його більш привабливим та професійним.

Використання іконок в UI допомагає поліпшити взаємодію користувача з системою, забезпечує швидкість сприйняття інформації та додає естетичний аспект до дизайну. Важливо вибирати адекватні та зрозумілі іконки, а також забезпечувати консистентність в їх використанні для досягнення максимальної ефективності та задоволення користувачів. З точку зору дизайну доцільно використовувати іконки, які стилізовані однаково. Тоді вигляд сторінок буде цілісним та охайнішим.

FontAwesome є популярною бібліотекою іконок, яка надає широкий вибір векторних іконок для веб-розробки. Вона пропонує набір іконок у вигляді шрифту та CSS-класів, що дозволяє легко використовувати їх у вашому проекті.

Бібліотека легко підключається за допомогою CDN, що дозволяє лише додати потрібне посилання в хедері головного шаблону. Після цього відразу можна використовувати іконки додаючи їх в HTML код у вигляді тега. Додатково обрану іконку можна продовжити стилізувати: додавати власні стилі, змінювати колір, розмір. FontAwesome надає широкий вибір іконок на будь-яку тему в декількох варіантах, наприклад, різної товщини ліній, значок із заливкою або тільки з контуром.

На рисунку 5.7 зображено блок, що відображає додаткову інформацію про нотатку. Серед інших елементів блоків можна помітити іконки, які передають різного роду інформацію, наприклад, розташування дати та часу нагадування або дедлайну. Використання іконок економить місце, інтуїтивно зрозумілі користувачу та однаково стилізовані як між собою, так і з рештою елементів.

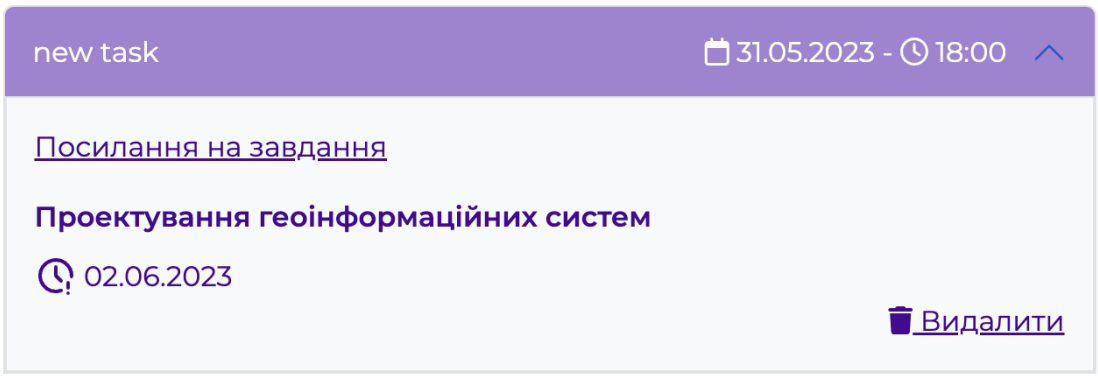

Рисунок 5.8 – Приклад застосування іконок в користувацькому інтерфейсі

5.9.3 Реалізація представлення та його компонентів

Laravel має потужну систему шаблонів, відому як Blade, яка дозволяє легко впорядковувати та відображати веб-сторінки. Blade забезпечує простий і елегантний синтаксис для роботи з кодом PHP у шаблонах, що дозволяє швидко та зручно створювати сторінки HTML. Крім того, Laravel надає можливість використовувати компоненти для створення повторно використовуваних блоків коду. У Laravel шаблони використовуються для структурування та відображення веб-сторінок. Вони можуть відокремити логіку відображення від логіки програми. Шаблони мають розширення .blade.php і складаються з HTML-коду разом із вставками PHP-коду. Використовуючи шаблони, можна визначати макети для сторінок, спільних елементів інтерфейсу та інших повторюваних блоків.

Blade — це механізм шаблонів, вбудований у Laravel. Він пропонує зручний і виразний синтаксис для вставки PHP-коду та виконання умов, циклів, відображення даних тощо. Blade дозволяє використовувати директиви (@if, @foreach, @extends тощо) та компоненти для організації шаблонів.

Компоненти — це повторно використовувані блоки коду, які можна використовувати в різних місцях програми. Вони дозволяють створювати ізольовані блоки коду, які містять власну логіку та відображення. Компоненти можна використовувати з параметрами, що робить їх більш гнучкими та налаштовуваними.

Усі шаблони зберігаються в папці «resources/views» та згруповані логічно по папкам, що відповідають задачі або області шаблонів.

Для компонентів є окрема папка. Для кожного компонента є відповідний клас PHP, який визначає, які параметри може приймати компонент та яких типів. Потім в коді вставляється компонент у вигляді тега, де ім’я тега – це ім’я компонента.

Використання компонентів веб-розробки має багато переваг, особливо коли маємо справу зі складними та масштабованими програмами. Компоненти дозволяють створювати блоки коду, які можна використовувати на різних сторінках або в різних частинах програми. Це дозволяє повторно використовувати код, що спрощує розробку та підвищує продуктивність.

Використання компонентів допомагає структурувати код програми в логічні блоки. Компоненти можуть бути незалежними самодостатніми модулями, які легко зрозуміти та підтримувати. Це полегшує командну роботу над проектом і дозволяє швидко знаходити та вносити зміни.

Також це прискорює розробку, оскільки можна використовувати готові компоненти замість того, щоб писати код з нуля. Це зменшує кількість коду, необхідного для розробки, і дозволяє зосередитися на бізнес-логіці та основній функціональності програми.

Використання компонентів допомагає створити єдиний стиль та вигляд веб-додатку. Можна стилізувати та налаштовувати компоненти відповідно до своїх потреб, забезпечуючи послідовність у всій програмі.

Компоненти роблять код більш зрозумілим і зручним для обслуговування. Вони дозволяють зосередитися на певних частинах програми, що полегшує розробку нових функцій і внесення змін. Крім того, можна легко замінити або розширити компоненти, не впливаючи на інші частини коду.

Загалом використання компонентів допомагає створювати більш упорядкований код, який можна багаторазово використовувати та легко підтримувати, що прискорює процес розробки та забезпечує вищу якість веб-програми.

5.9.4 Створення полів вводу за допомогою бібліотеки Select2

Select2 — це популярний плагін JavaScript для стилізації та вдосконалення вибраних елементів веб-форм. Він надає багато додаткових функцій і можливостей, які розширюють стандартний елемент вибору HTML.

Бібліотеку Select2 було обрано, тому що вона надає можливість швидкого пошуку елементів у великому списку та дозволяє вводити текст і фільтрувати елементи, що допомагає швидко і легко знайти потрібний варіант.

Select2 дозволяє вибирати багато елементів одночасно, дозволяючи вибирати кілька значень зі списку. Це корисно, коли потрібно вибрати кілька варіантів або зробити множинний вибір. Наприклад, з такою ціллю Select2 застосовується в таких частинах програми, де потрібно обрати декілька груп для завдання або додати декілька дітей для батьків. На рисунку 5.8 зображено приклад такого поля форми. У такому полі можна обирати декілька опцій одночасно та здійснювати пошук.

Рисунок 5.9 – Вигляд поля форми з типом мульти-селект

Бібліотека надає ряд інструментів та властивостей, за допомогою яких налаштування відбувається швидко та зрозуміло. Бібліотека має методи для кастомізування своїх компонентів, що дозволяє створювати свої класи та додавати їх до елементів поля, у такий спосіб змінюючи вигляд списку, обраних елементів та самого поля.

Бібліотека дозволяє робити запити AJAX для динамічного завантаження даних у список. Це особливо корисно, коли у вас є великий обсяг даних для вибору з віддаленого сервера.

Select2 надає широкі можливості для налаштування та стилю. Він дозволяє змінити зовнішній вигляд елементів списку, додати власні класи та стилі CSS, а також налаштувати поведінку компонента відповідно до ваших потреб.

5.9.5 Створення таблиць за допомогою бібліотеки DataTables

DataTables — це потужна бібліотека JavaScript для створення інтерактивних таблиць у веб-додатках. Він надає широкі можливості для організації, сортування, фільтрації та візуалізації даних у вигляді таблиць з багатьма додатковими функціями. Вона додає багато додаткових функцій до стандартних таблиць HTML, таких як сортування, фільтрація, розбивка на сторінки, пошук тощо. Це дозволяє користувачам взаємодіяти з даними в таблиці, роблячи її більш функціональною та зручною у використанні.

DataTables надає велику кількість параметрів і параметрів, які дозволяють змінювати зовнішній вигляд, поведінку та функціональність таблиці відповідно до потреб. Є можливість налаштувати відображення колонок, стилі, додаткові елементи керування та багато іншого [8].

Таблиці можуть ефективно обробляти великі обсяги даних, навіть тисячі записів, за допомогою сторінкового та відкладеного завантаження. Це дозволяє підтримувати продуктивність і швидкість роботи таблиці незалежно від обсягу даних.

DataTables підтримує адаптивний дизайн, що означає, що таблиця адаптується до різних розмірів екрана та пристроїв. Він може автоматично адаптувати свою структуру та функціональність для оптимального відображення на мобільних пристроях і планшетах.

Загалом DataTables дозволяє працювати з даними в зручний і ефективний спосіб, надаючи користувачам багато можливостей для взаємодії з даними в таблиці.

Для створення таблиці потрібно застосувати потрібні налаштування для конкретного елемента , який зазвичай позначається ідентифікатором. Для таких цілей найзручніше використовувати бібліотеку jQuery.

Коротко переглянемо основні властивостями для створення таблиць.

Властивість «info» вказує, чи відображати інформацію про сторінку таблиці, наприклад кількість записів, що відображаються на поточній сторінці, загальну кількість записів тощо.

Властивість «language» змінює локалізацію DataTables. Можна налаштувати тексти повідомлень, які відображаються в таблиці, як-от повідомлення про завантаження даних, повідомлення про порожні таблиці, текст кнопок тощо. За допомогою «language» можна налаштувати ці тексти відповідно до своїх потреб.

Так як за замовчуванням усі тексти таблиці написані англійською мовою, виникла потреба перейменувати все українською. Для цього потрібно зайти на сайт з документацією DataTables у розділ з локалізацією та знайти потрібну мову (в даному випадку, українську). Бібліотека надає ресурси з перекладами для таблиць. Це посилання потрібно скопіювати та передати як значення до властивості «language.url». після цього переклад автоматично буде застосовано до всіх написів таблиці.

Властивість «columnDefs» дозволяє налаштувати поведінку та властивості окремих стовпців таблиці. Для кожного стовпця можна додати різні властивості, як-от видимість, сортування, фільтрування, класи стилів тощо. Це дозволяє контролювати вигляд поведінки кожного стовпця в таблиці.

Властивість «paging» визначає, чи вмикати розбиття на сторінки для таблиці. Властивість «searching» визначає, чи вмикати пошук у таблиці. За замовчуванням встановлено значення true, що дозволяє користувачам шукати записи в таблиці за допомогою текстового поля пошуку. Властивість «ordering» визначає, чи дозволяти сортування даних у таблиці.

У системі таблиці застосовуються для структурування та відображення інформації. Для адміністратора це таблиці з усіма користувачами, де подається вся

інформація про них. Табличний вигляд легко сприймається та якісно відображає дані. У таблиці відразу видно різницю між записами та список властивостей, що описують сутність. До того ж таблицю можна сортувати та здійснювати пошук, що значно полегшує та пришвидшує роботу.

Дані можна відразу внести в таблицю в HTML файлі, а можна підгрузити за допомогою AJAX. DataTables мають властивість, якій потрібно передати маршрут до контролера і з цим шляхом таблиці підтягнуть потрібні дані.

5.9.6 Використання асинхронних запитів

Асинхронні запити - це методологія взаємодії між клієнтською та серверною сторонами програмного забезпечення, де запити можуть бути відправлені та оброблені незалежно один від одного без блокування основного потоку виконання [10].

Звичайно, при традиційних синхронних запитах клієнт відправляє запит до сервера і блокується до отримання відповіді. У цей час неможливо взаємодіяти з іншими елементами сторінки або виконувати інші дії.

Асинхронні запити дають змогу відправляти запити на сервер без блокування основного потоку. Клієнтська сторона може відправити запит і продовжувати виконання інших операцій, таких як зміна вмісту сторінки, відображення анімації чи взаємодія з користувачем. Коли сервер повертає відповідь на запит, спеціальна функція (зазвичай називається callback) виконується для обробки цієї відповіді [10].

Асинхронні запити використовуються широко для веб-розробки, особливо при отриманні або відправленні даних на сервер без перезавантаження сторінки. Це дозволяє створювати більш динамічні та реактивні веб-додатки, покращує швидкодію та користувацький досвід. AJAX (асинхронний JavaScript і XML) є одним з популярних підходів до реалізації асинхронних запитів на веб-сторінках.

Ajax (Asynchronous JavaScript and XML) - це технологія, яка дозволяє взаємодіяти з сервером без перезавантаження сторінки. Вона використовує

JavaScript для асинхронного відправлення та отримання даних з сервера, а також для динамічного оновлення вмісту веб-сторінки без повного перезавантаження.

Axios - це JavaScript-бібліотека, яка дозволяє зручно виконувати HTTP-запити з використанням технології Ajax. Вона підтримує відправку різних типів запитів, таких як GET, POST, PUT, DELETE, та дозволяє встановлювати заголовки запиту, передавати параметри, обробляти відповіді сервера та багато іншого.

Основні переваги Axios:

- має простий та зрозумілий інтерфейс, що дозволяє швидко виконувати HTTP-запити зі сторони клієнта;
- повертає проміси, що дозволяє використовувати синтаксис обіцянок (promises) або async/await для обробки результатів запитів;
- можливість легко встановлювати заголовки запиту, такі як заголовки авторизації або заголовки з додатковою інформацією;
- має вбудовану підтримку обробки помилок, що дозволяє зручно визначати та обробляти помилки під час виконання запитів [9].

AJAX є основою для багатьох UI бібліотек. Як описувалось раніше, такі бібліотека, як Select2 та DataTables підтримують використання AJAX для збору даних шляхом відправки запиту на потрібний маршрут. Для цього у системі було створена окрема група маршрутів – API маршрути. Вони відрізняються від веб маршрутів тим, що не повертають представлення, а лише дані. API виклики є універсальними та зазвичай створюються для використання різними платформами, бо не залежать від мови чи фреймворка. Їхня мета - надати дані.

У систем такі маршрути окремо згруповані та використовуються в AJAX запитах. Найпростішим прикладом є API маршрут, який надає масив даних з інформацією про усіх студентів, які користуються системою. Кожен маршрут має відповідний йому контролер, який через модель взаємодіє з базою даних та повертає відповідь з даними або кодом помилки, якщо така виникла в системі.

Для оновлення розкладу та нотаток при зміні тижня також використовується AJAX. Це дозволяє не перезавантажувати кожен раз сторінку, а лише отримувати

оновлену інформацію та заміняти її в шаблоні. Для запитів на отримання розкладу та нотаток передається ще додатковий параметр «weekNumber». За допомогою моделі обираються записи, дата яких відповідає номеру тижня в році, та підставляються у представлення.

Форма створення нотаток та завдання знаходиться у модальному вікні. При проведені валідації, бекенд може повернути помилку. При цьому сторінка буде перезавантажена, а вікно закриється, що може заплутати користувача. У такому випадку необхідно використовувати AJAX запит та за його допомогою повертати помилки валідації. Такий підхід допоміг знайти вирішення проблеми та зробити роботу програми гнучкою та зручною для користувача.

Висновки до розділу 5

Фреймворк Laravel надає широкий вибір інструментів для реалізації різних потреб системи. Він сумісний з багатьма типами баз даних та СКБД, бібліотек, UI фреймворків. З використанням фреймворку створення додатку виходить швидким, код чистим та добре розширюваним. У системі особлива увага приділялась безпеці та коректності введених даних, так як від цього залежить цілісність даних та правильна робота системи.

6 ВЗАЄМОДІЯ З КОРИСТУВАЦЬКИМ ІНТЕРФЕЙСОМ

Користувач починає свій маршрут з головної сторінки, на якій коротко розповідається в чому мета та задача системи та присутня кнопка Увійти. Тут варто зазначити, що система не передбачає самостійну реєстрацію користувачів. Програма сама синхронізується з університетськими даними про студентів та викладачів та генерує для них паролі. Куратори або старости надають ці паролі групам, а відповідальні на кафедрі викладачам, після чого вони можуть увійти в систему та користуватись її функціоналом. Вигляд сторінки з формою авторизації можна бачити на рисунку 6.1.

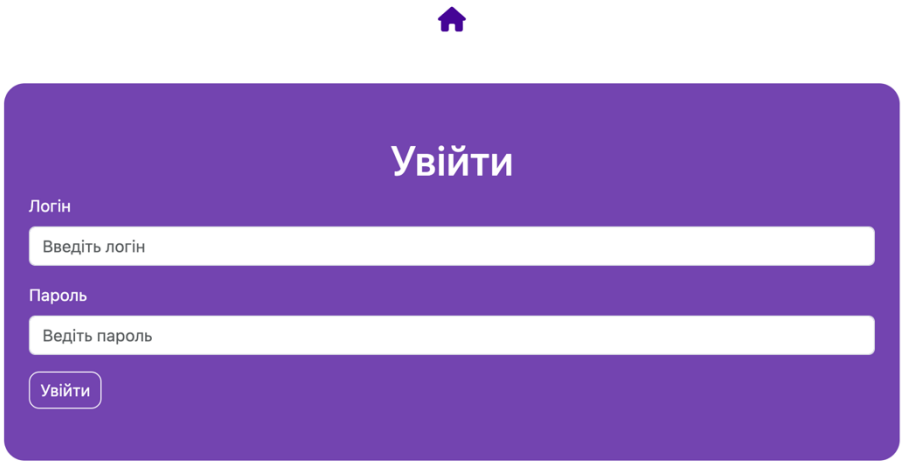

Рисунок 6.1 – Вигляд форми авторизації

Після того, як користувач здійснив вхід в систему, його автоматично перенаправляє на сторінку з розкладом. Тут він може переглянути розклад для його групи. Якщо користувач зайшов в систему в перший раз, справа, де розташований блок з нотатками, буде пусто. Користувач бачить лише кнопки “Створити нотатку” на кожен день тижня.

Розклад розбитий по тижням, який можна змінити користуючись випадаючим списком зверху. Він відображає пронумеровані тижні року. Користувач бачить, який зараз тиждень та може змінити, якщо потрібно

запланувати щось заздалегідь або переглянути минулі завдання. Загальний вигляд сторінки з розкладом та нотатками показаний на рисунку 6.2.

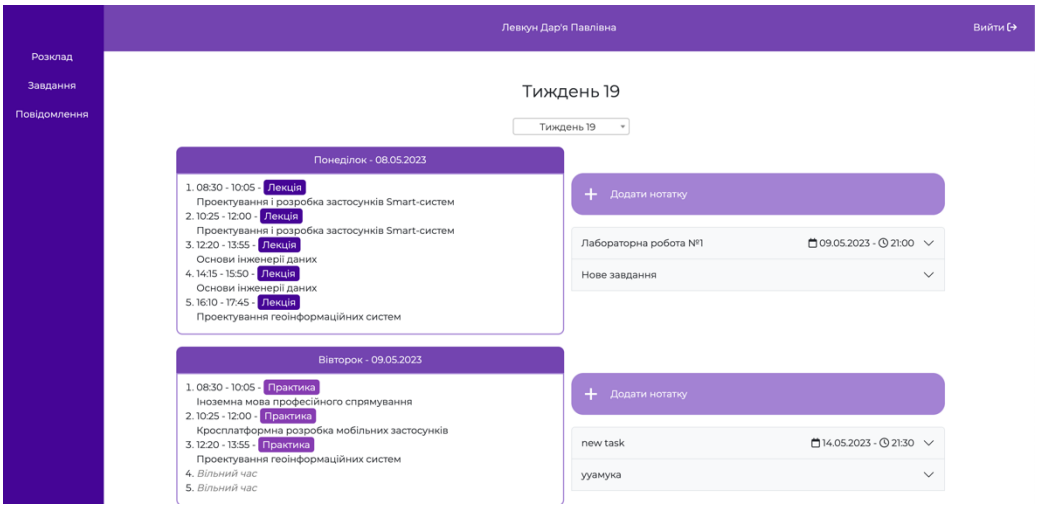

Рисунок 6.2 – Вигляд сторінки розкладу для студента

Натиснувши на кнопку “Створити нотатку” користувач бачить перед собою форму для її створення. Вона зображена на рисунку 6.3.

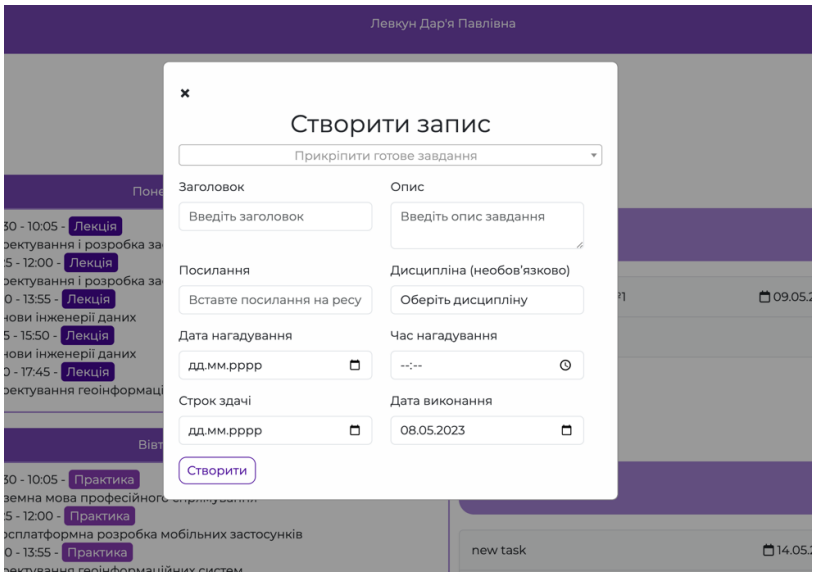

Рисунок 6.3 – Вигляд форми для створення нотаток

Тут присутні такі поля:

- заголовок - як короткий опис того, що потрібно зробити. Це поле є обов’язковим;

- опис - детальніший опис задачі. Можна лишити пустим;
- посилання - необов’язкове поле, використовується в разі того, якщо потрібно прикріпити якийсь ресурс;
- дата нагадування - дата, коли система має надіслати користувачу нагадування про поточне завдання;
- час нагадування - час, коли система має надіслати користувачу нагадування про поточне завдання;
- строк здачі - дата, до моменту якої потрібно виконати це завдання;
- дата виконання - день, до якого прив’язана нотатка;
- дисципліна - назва дисципліна, яка може бути ціллю завдання описаного в нотатці.

У системі передбачено функціонал, коли можна прикріпити до нотатки готове завдання, яке до цього було створене викладачем для цієї групи. Для цього існують ще деякі додаткові поля в формі для створення нотатки:

- завдання - дозволяє обрати з випадаючого списку готове завдання. Після цього ключові поля завдання, такі як заголовок, опис, посилання, строк здачі та дисципліна, автоматично заповняться у поля нотатки. Користувач зможе відредагувати текст або додати щось свого, та зберегти нотатку;
- дисципліна - може бути обрана не прив’язуючись до завдання. Автозаповнюється, якщо користувач вирішив прикріпити завдання.

Студент може перейти на вкладку Завдання. Там перед ним відобразяться усі дисципліни, які в нього зараз викладаються. Список відображається у вигляді “акордеону”. Натиснувши на дисципліну, з’явиться список з завданнями. Біля кожного завдання студент може бачити оцінку або прочерк, якщо її ще не поставили. Студент може натиснути на завдання, перейти на окрему сторінку та переглянути деталі.

У вкладці Повідомлення Студент бачить свої сповіщення. На сторінці є дві вкладки: в одній відображаються ще не прочитані повідомлення, а в другій усі

сповіщення, які колись приходили користувачу. Приклад одного з повідомлень зображено на рисунку 6.4.

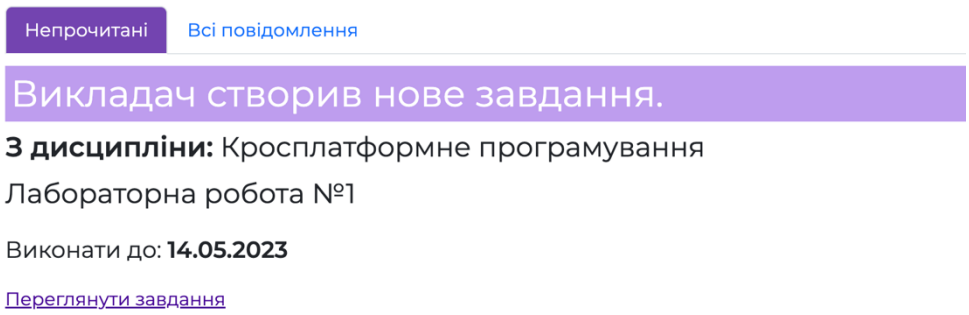

Рисунок 6.4 – Вигляд сторінки зі сповіщеннями

Викладач, так само як і Студент, після входу в систему бачить сторінку з розкладом. Тут відображається власний розклад викладача. Функціонал створення нотаток - аналогічний створенню нотатки для Студента.

Викладач має право виставляти оцінки студентам за завдання. Для цього потрібно перейти на вкладку Оцінки. Відобразиться список дисциплін, які веде викладач. Цей список виконаний у вигляді “акордеона”. Натиснувши на назву дисципліни, “акордеон” відкриє блок, що стосується цієї дисципліни, в якому міститься список груп, в яких викладач веде цей предмет. Натиснувши на групу, викладачу відкриється сторінка з таблицею. В таблиці присутній список студентів обраної групи та список усіх завдань цієї дисципліни. Викладач виставляє оцінки навпроти кожного студента та відповідного завдання.

Для створення завдань викладач має перейти на вкладку “Завдання для груп”. Відобразиться сторінка з дисциплінами та групами у вигляді акордеону, як у вкладці з оцінками. Викладач обирає групу та може переглянути список завдань або створити нове. Для цього на сторінці присутня кнопка “Створити завдання”. При натисканні на неї з’являється вікно для створення завдання. У формі присутні поля, такі як “Заголовок”, “Опис”, “Посилання” та “Дата виконання”.

У списку завдань навпроти кожного завдання є кнопка відредагувати. Натиснувши на неї викладач бачитиме форму з заповненими старими даними. Старі дані можна відредагувати та зберегти.

Вкладка Повідомлення виконує такий самий функціонал для викладача, як і для Студента.

Батьки при вході в систему відразу бачать завдання, які створені їхніх дітей. Якщо в батька чи матері в системі зареєстровані двоє чи більше дітей, спочатку вони будуть бачити список, де можуть обрати, інформацію про яку дитину вони хочуть переглянути. Після цього їм відображається список з завдань аналогічний тому, що бачать студенти в себе на сторінці. Батько чи мати можуть переглянути завдання та оцінку, яку дитина отримала.

Адміністратор при вході в систему бачить таблиці з усіма користувачами: студентами, батьками, викладачами. Інформація, яка відображається в таблицях, включає в себе імена, електронні адреси, групи, до яких належать студенти, факультети та кафедри, посади викладачів і т.д.

У вкладці Додати користувача розташована форма. Її функціонал полягає в додаванні батьків, так як вони не є частиною університету з самого початку і їхні дані не знаходяться в університетських базах. Адміністратор записує основну інформацію про нового користувача: ім’я, електронна адреса, телефон і т.д. Найголовніше - це обрати дітей серед студентів, що є у випадаючому списку та таким чином прикріпити їх до батька чи матері.

Висновки до розділу 6

Інтерфейс системи інтуїтивно зрозумілий, супроводжується надписами, які допомагають користувачу орієнтуватись в застосунку. Дизайн має приємну кольорову гаму, весь текст видно та зручно читати. Значки коротко позначають дії або передають якусь інформацію в стислій формі, економлять місце та вписуються в загальний вигляд системи.

Функціонал системи виконує завдання системи та керує програмними задачами. Зручна навігація допомагає легко пересуватись сайтом. Користувач має широкий вибір інструментів, якими може користуватись. Для створення нотаток та завдань існує достатньо параметрів, як обов’язкові, так і на вибір користувача.

ВИСНОВКИ

У даній роботі було створено систему автоматизованого планування бізнес-процесів для контингенту кафедри. Для цього було створено веб-систему, яка спрямована на забезпечення прозорого сприйняття навчального процесу усіма його суб’єктами: викладачами, студентами та батьками. Система була ретельно спроектована для виконання базових задач. Серед них створення нотаток, завдань, виставлення оцінок та системи сповіщень, яка пов’язує між собою різні процеси та інформує користувачів про їх виконання.

Було розроблено зручний та зрозумілий інтерфейс, який допомагає швидко дослідити та використовувати систему.

Основна мета - покращення ефективності навчального процесу, залучення батьків до спостерігання за ним та вдосконалення успішності студентів за рахунок організації часу.

При створені системи були враховані фактори безпеки та захисту даних, спроектовано базу даних, для якої написані запити, що швидко дістають інформацію та відображають її.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тейлор О. Laravel Documentation. [Електронний ресурс]. - Режим доступу: https://laravel.com/docs/8.x (дата звернення: 12.05.2023).

2. Толетино А. Laravel: Рефакторинг до чистого коду. Київ: Видавничий дім "КМ-Букс", 2020. 416 с.

3. Жарова Л. Ю., Бариленко В. В. Веб-програмування та його технології. Київ: ВПЦ "Київський університет", 2019. 412 с.

4. Котеров, Д. PHP. Об'єктно-орієнтоване програмування. 4-е видання. Київ: Видавництво "Наш Формат", 2022. 512 с.

5. Денисенко І. В. Шаблон репозиторію в мові PHP. Харків: Видавничий дім "Ін Юре", 2014. 120 с.

6. Толстохатько, В. А., Поморцева, О. Є., Патракеєв, І. М. Бази даних: проектування та використання для обліку нерухомого майна. Харків: ХНУМГ, 2014. 175 с.

7. Bootstrap Documentation. [Електронний ресурс]. – Режим доступу: https://getbootstrap.com/docs/4.6/getting-started/introduction/ (дата звернення: 25.05.2023).

8. DataTables Documentation. [Електронний ресурс] / The DataTables Project. – Режим доступу: https://datatables.net/manual/ (дата звернення: 25.05.2023).

9. Axios Docs. [Електронний ресурс]. — Режим доступу до ресурсу: https://axios-http.com/docs/intro/ (дата звернення 24.05.2023).

10. Сучасний підручник з JavaScript. [Електронний ресурс]. - Режим доступу: https://uk.javascript.info/ (дата звернення 24.05.2023).

11. Ульман Дж. Д. Введення в системи баз даних: [монографія] / Дж. Д. Ульман, Дж. Уідом; пер. з англ. - М .: Лорі, 2000. - 374 с.

12. Mailtrap Documentation: [Електронний ресурс] / Mailtrap. – Режим доступу: https://mailtrap.io/docs (Дата звернення: дата звернення на ресурс).

ДОДАТОК А

Створення системи автоматизованого планування бізнес-процесів для контингенту кафедри

Лістинг програмного модуля

Аркушів 15

Київ – 2023

Контролер для роботи з профілем адміністратора

```
class AdminController extends Controller
{

    public function __construct(
        protected StudentRepository $studentRepository,
        protected TeacherRepository $teacherRepository,
        protected UserRepository $userRepository
    ) {}

    public function index(): Factory|View|Application
    {
        $students = $this->studentRepository->getAll();
        $teachers = $this->teacherRepository->getAll();
        $parents = $this->userRepository->getParents();

        return view('admin.index', compact('students', 'teachers', 'parents'));
    }

    public function create(): Factory|View|Application
    {
        $roles = [RolesEnum::ADMIN, RolesEnum::PARENT];
        return view('admin.create-user', compact('roles'));
    }

    private function generatePassword($length = 8): string
    {
        $chars = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_';
        $password = '';
        for ($i = 0; $i < $length; $i++) {
            $index = rand(0, strlen($chars) - 1);
            $password .= substr($chars, $index, 1);
        }
        return $password;
    }

    public function save(StoreUserRequest $request)
    {
        try {
            DB::beginTransaction();

            $password = $this->generatePassword(12);

            $parentId = DB::table('users')->insertGetId([
                'full_name' => $request->get('full_name'),
                'email' => $request->get('email'),
```

```
            'telegram_tag' => $request->get('username'),
            'phone_number' => $request->get('phone_number'),
            'reserve_phone' => $request->get('reserve_phone'),
            'role' => $request->get('role'),
            'password' => $password
        ]);

        $childrenData = $this->mapChildrenIds($parentId, $request->get('children'));

        DB::table('parent_child')->insert($childrenData);

        DB::commit();
    } catch (\Exception $e) {
        DB::rollBack();
        logger()->error($e->getMessage());

        return back()->withInput()->with('error', 'Щось пішло не так :(');
    }

    return back()->with('success', 'Нового користувача успішно створено')->with('password',
$password);
  }


  private function mapChildrenIds(int $parentId, array $childrenIds): array
  {
    return array_map(function ($childId) use ($parentId) {
        return [
            'parent_id' => $parentId,
            'child_id' => (int)$childId
        ];
    }, $childrenIds);
  }
}
```

Контролер для роботи зі створенням, збереженням та відображенням завдань

```
class TasksController extends Controller
{

  public function __construct(
    protected CourseRepository $courseRepository,
    protected GroupRepository $groupRepository,
    protected TaskRepository $taskRepository,
    protected StudentRepository $studentRepository,
    protected MarksRepository $marksRepository
  ){}
```

```
public function indexTeacher(): Factory|View|Application
{
    $teacher = auth()->user()->teacher;
    $courses = $this->courseRepository->getCoursesByTeacher($teacher->id);

    return view('tasks.teacher.courses', compact('courses'));
}

public function indexStudent(): Factory|View|Application
{
    $student = auth()->user()->student;
    $courses = $this->courseRepository->getCoursesForGroup($student->group_id);
    $mappedCourses = $this->mapCoursesWithTasks($courses, $student->group_id, $student->id);

    return view('tasks.student.courses', ['courses' => $mappedCourses]);
}

private function mapCoursesWithTasks($courses, int $groupId, int $studentId): array
{
    $mappedCourses = [];
    foreach ($courses as $course) {
        $mappedCourses[] = json_decode(json_encode($course), true);
    }

    return array_map(function ($course) use ($groupId, $studentId) {
        $tasks = $this->taskRepository->getTasksForCourse($course['id'], $groupId);
        $tasks = $tasks->map(function ($task) use ($studentId) {
            $mark = $this->marksRepository->getMarkForTask($task->id, $studentId);
            $task->mark = $mark?->mark;

            return $task;
        });

        return Arr::set($course, 'tasks', json_decode(json_encode($tasks), true));
    }, $mappedCourses);
}

public function tasksForGroup(Request $request, $courseId): Factory|View|Application
{
    $params = $this->validateParameters(
        collect(['courseId' => $courseId, 'groupId' => $request->query('groupId')])
    );
    $teacherId = auth()->user()->teacher->id;

    $currentGroup = $this->groupRepository->getById($params->get('groupId'), ['tasks']);
```

```
    $data = [
        'course' => $this->courseRepository->getById($params->get('courseId')),
        'group' => $currentGroup,
        'tasks' => $this->taskRepository->getTasksByTeacher(
            $params->get('courseId'),
            $params->get('groupId'),
            $teacherId
        ),
    ];

    return view('tasks.teacher.group-tasks', compact('data'));
}

private function validateParameters(Collection $params): Collection
{
    return $params->map(function ($item) {
        return (int)htmlspecialchars($item);
    });

}

public function createTask(StoreTaskRequest $request): JsonResponse
{
    $validator = Validator::make($request->all(), [
        'title' => 'required|string|min:3|max:45',
        'description' => 'string|max:255|nullable',
        'link' => 'active_url|nullable',
        'deadline_date' => 'date|after:today|nullable',
        'courseId' => 'required|exists:App\Models\Course,id',
        'groups' => 'required|array',
    ]);

    if ($validator->fails()) {
        return response()->json([
            'errors' => $validator->errors(),
        ]);
    }

    $courseId = (int)$request->get('courseId');
    $groups = $request->get('groups');

    $task = Task::create(
        [
            'title' => $request->get('title'),
            'description' => $request->get('description'),
```

```
            'link' => $request->get('link'),
            'deadline_date' => $request->get('deadline_date'),
            'course_id' => $courseId,
            'created_by' => auth()->user()->teacher->id,
        ]
    );

    $task->save();

    foreach ($groups as $groupId) {
        TaskGroup::create(
            [
                'task_id' => $task->id,
                'group_id' => $groupId,
            ]
        )->save();
    }

    TaskCreated::dispatch($task, $groupId);

    return response()->json([
        'success' => true,
        'redirect' => route('group.tasks', ['courseId' => $courseId, 'groupId' => $groupId])
    ]);
}

/**
 * Show task details
 *
 * @param int $taskId
 * @return Factory|View|Application
 */
public function showTask(int $taskId): Factory|View|Application
{
    $params = $this->validateParameters(collect(['taskId' => $taskId]));

    $task = $this->taskRepository->getById($params->get('taskId'));
    $course = $this->courseRepository->getById($task->course_id, ['subject']);

    if (auth()->user()->role == RolesEnum::STUDENT) {
        $studentId = auth()->user()->student->id;
        $markModel = $this->marksRepository->getMarkForTask($task->id, $studentId);
        $task->mark = $markModel?->mark;
    }

    return view('tasks.one-task', compact('task', 'course'));
```

```
    }

    public function editTask(Request $request, int $taskId): Factory|View|Application
    {
        $params = $this->validateParameters(collect(['taskId' => $taskId, 'groupId' => $request->query('groupId')]));
        $task = $this->taskRepository->getById($params->get('taskId'), ['course.subject']);

        $data = [
            'task' => $task,
            'course' => $task->course,
            'groups' => $task->groups,
            'currentGroup' => $this->groupRepository->getById($params->get('groupId'))
        ];

        return view('tasks.edit', compact('data'));
    }

    public function updateTask(StoreTaskRequest $request): RedirectResponse
    {
        $params = $this->validateParameters(collect(['taskId' => $request->query('taskId')]));

        $taskModel = Task::find($params->get('taskId'));

        $taskModel->update([
            'title' => $request->get('title'),
            'description' => $request->get('description'),
            'link' => $request->get('link'),
            'deadline_date' => $request->get('deadline_date'),
            'course_id' => $request->get('courseId'),
            'created_by' => auth()->user()->teacher->id,
        ]);

        $taskModel->save();

        TaskGroup::where('task_id', $params->get('taskId'))->delete();
        $groupIds = $request->get('groups');

        foreach ($groupIds as $groupId) {
            TaskGroup::create(
                [
                    'task_id' => $params->get('taskId'),
                    'group_id' => $groupId,
                ]
            )->save();
        }
```

```
        return back()->with('success', 'Завдання успішно оновлене');
    }

    public function indexParent(): Factory|View|Application
    {
        $parent = auth()->user();
        $children = $parent->children;

        if ($children->count() == 1) {
            $child = $children->first();
            $courses = $this->courseRepository->getCoursesForGroup($child->group_id);
            $mappedCourses = $this->mapCoursesWithTasks($courses, $child->group_id, $child->id);

            return view('tasks.student.courses', ['courses' => $mappedCourses, 'childName' => $child-
>user->full_name]);
        } else {
            return view('tasks.parent.children-list', compact('children'));
        }
    }

    /**
     * Get student's task
     * @param int $childId
     * @return Factory|View|Application
     */
    public function getChildTasksPage(int $childId): Factory|View|Application
    {
        $student = $this->studentRepository->getById($childId, ['user']);

        $courses = $this->courseRepository->getCoursesForGroup($student->group_id);
        $mappedCourses = $this->mapCoursesWithTasks($courses, $student->group_id, $childId);

        return view('tasks.student.courses', ['courses' => $mappedCourses, 'childName' => $student-
>user->full_name]);
    }

    /**
     * Get tasks list for note dropdown
     *
     * @param Request $request
     * @return JsonResponse
     */
    public function tasks(Request $request): JsonResponse
    {
        $searchTerm = $request->get('search');
```

```
        $role = auth()->user()->role;

        $tasks = match ($role) {
            RolesEnum::TEACHER => $this->taskRepository->getTasksForTeacher(auth()->user()->teacher->id, $searchTerm),
            RolesEnum::STUDENT => $this->taskRepository->getTaskForGroup(auth()->user()->student->group_id, $searchTerm),
        };

        return response()->json($tasks);
    }

    /**
     * Get task details for filling note
     *
     * @param int $taskId
     * @return JsonResponse
     */
    public function getTaskInfo(int $taskId): JsonResponse
    {
        $task = $this->taskRepository->getById($taskId);

        return response()->json($task);
    }

    /**
     * Delete task
     *
     * @param int $taskId
     * @return RedirectResponse
     */
    public function deleteTask(int $taskId): RedirectResponse
    {
        try {
            DB::beginTransaction();

            $this->taskRepository->delete($taskId);

            DB::commit();
        } catch (\Exception $e) {
            DB::rollBack();
            logger()->error($e->getMessage());

            return back()->with('error', 'Щось пішло не так :(');
        }
```

```
        return back()->with('success', 'Завдання успішно видалено');
    }
}

Контролер для роботи з рокзладом
class ScheduleController extends Controller
{

    public function __construct(
        protected ScheduleRepository $scheduleRepository,
        protected CourseRepository $courseRepository,
        protected NotesRepository $notesRepository
    )
    {
    }

    public function index(): Factory|View|Application
    {
        $userRole = auth()->user()->role->value;
        $schedule = [];
        $courses = [];

        $weekNumber = $this->getWeekNumber();
        $currentDate = Carbon::parse(Carbon::now());
        $week = ($weekNumber % 2 == 0) ? 2 : 1;

        if ($userRole == 'student') {
            $groupId = auth()->user()->student->group_id;
            $repository = new StudentRepository();
            $schedule = $repository->getSchedule();

            $courses = $this->courseRepository->getCoursesForGroup($groupId);
        } else if ($userRole == 'teacher') {
            $repository = new TeacherRepository();
            $schedule = $repository->getSchedule();

            $teacherId = auth()->user()->teacher->id;
            $courses = $this->courseRepository->getCoursesByTeacher($teacherId);
            $courses = $courses->map(function ($course) {
                $course->title = $course->subject->title;
                return $course;
            });
        }

        // filter by week
```

```
        $schedule = $schedule->filter(function ($item) use ($week) {
            return $item->week == $week;
        });

        $filteredSchedule = [];

        $schedule->each(function ($item) use (&$filteredSchedule) {
            $filteredSchedule[(int)$item->day_of_week][] = $item;
        });

        $filteredSchedule = collect($filteredSchedule)->map(function ($lessons, $day) {
            return collect($lessons)->mapWithKeys(function ($lesson) {
                return [$lesson->order => $lesson];
            });
        });

        $startOfWeek = Carbon::create(Carbon::now()->year)->startOfWeek()->addWeeks($weekNumber);
        $endOfWeek = $startOfWeek->copy()->endOfWeek();

        $notes = $this->notesRepository->getNotesByDate([$startOfWeek, $endOfWeek], auth()->user()->id, ['task', 'course']);
        $notes = $this->mapNotes($notes);

        return view('schedule.index', compact('filteredSchedule', 'weekNumber', 'courses', 'currentDate', 'notes'));
    }

    private function mapNotes($notes): array
    {
        $mappedNotes = [];

        $notes->each(function ($note) use (&$mappedNotes) {
            $dayOfWeek = Carbon::parse($note->performing_date)->dayOfWeek;
            $mappedNotes[$dayOfWeek][] = $note;
        });

        return $mappedNotes;
    }

    private function getWeekNumber(): int
    {
        $currentDate = new DateTime();

        return (int)$currentDate->format("W");
    }
```

```
    public function getScheduleForWeek(int $weekNumber): JsonResponse
    {
        $userRole = auth()->user()->role;
        $schedule = collect();

        if ($userRole == RolesEnum::STUDENT) {
            $repository = new StudentRepository();
            $schedule = $repository->getSchedule();
        } else if ($userRole == RolesEnum::TEACHER) {
            $repository = new TeacherRepository();
            $schedule = $repository->getSchedule();
        }

        $week = ($weekNumber % 2 == 0) ? 2 : 1;

        $schedule = $schedule->filter(function ($item) use ($week) {
            return $item->week == $week;
        });

        $filteredSchedule = [];

        $schedule->each(function ($item) use (&$filteredSchedule) {
            $filteredSchedule[(int)$item->day_of_week][] = $item;
        });

        $filteredSchedule = collect($filteredSchedule)->map(function ($lessons, $day) {
            return collect($lessons)->mapWithKeys(function ($lesson) {
                $lesson->title = $lesson->course->subject->title;
                $lesson->group_code = auth()->user()->role == RolesEnum::TEACHER ? $lesson->group-
>code : null;

                return [$lesson->order => $lesson];
            });
        });


        return response()->json($filteredSchedule);
    }
}
```

Команда для надсилання листів та сповіщень про те, що скоро строк здачі завдання

```
class SendTaskDeadlineReminder extends Command
{
    /**
     * The name and signature of the console command.
```

```
     *
     * @var string
     */
    protected $signature = 'send:task-deadline';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Send reminder that task deadline is soon';

    /**
     * Execute the console command.
     *
     * @return void
     */
    public function handle(): void
    {
        $now = Carbon::now();
        $tasks = Task::select(['id', 'title', 'description', 'course_id', 'link', 'deadline_date'])
            ->where('deadline_date', '=', $now->toDateString())
            ->get();

        foreach ($tasks as $task) {
            $groups = $task->groups;
            foreach ($groups as $group) {
                $students = (new StudentRepository())->getByGroup($group->id);

                SendTaskDeadlineReminderJob::dispatch($students, $task);
            }
        }
    }
}
```

Код представлення для блока з нотаткою

```
<div class="note-item">
    <div class="add-btn create_note_btn" data-id="{{ $day }}">
        <span class="plus-icon">+</span>
        <span> Додати нотатку </span>
    </div>
    <div class="notes accordion" id="{{ "accordion_$day" }}">
        @foreach($notes as $note)
            <div class="accordion-item">
                <h2 class="accordion-header">
```

```
            <button id="{{ "accordion_button_{$note->id}" }}" class="accordion-button collapsed"
type="button" data-bs-toggle="collapse" data-bs-target="#collapse_{{$note->id}}" aria-
expanded="true" aria-controls="collapse_{{$note->id}}">
                <span id="{{ "note_title_{$note->id}" }}">{{ $note->title }}</span>
                @if ($note->notification_date)
                  <span id="{{ "note_time_{$note->id}" }}" class="note-time">
                    <i class="fa-regular fa-calendar"></i>
                    {{ \Carbon\Carbon::parse($note->notification_date)->format('d.m.Y') }}
                    -
                    <i class="fa-regular fa-clock"></i>
                    {{ $note->notification_time }}
                  </span>
                @endif
              </button>
            </h2>
            <div id="collapse_{{$note->id}}" class="accordion-collapse collapse" aria-
labelledby="heading_{{$note->id}}" data-bs-parent=".accordion">
              <div class="accordion-body">
                @if($note->task_id)
                <p><a class="note-task-link" id="{{"note_task_link_{$note->id}"}}" href="{{
route('task.show', ['taskId' => $note->task_id]) }}">Посилання на завдання</a></p>
                @endif
                <p id="{{ "note_description_{$note->id}" }}"><i>{{$note->description}}</i></p>
                <div id="{{"note_course_{$note->id}"}}" class="note-subject">{{ $note->course?-
>subject->title }}</div>
                <p id="{{"deadline_date_{$note->id}"}}" class="note-deadline">
                  <img src="{{asset('/img/deadline.svg')}}" alt="deadline">
                  {{ \Carbon\Carbon::parse($note->deadline_date)->format('d.m.Y') }}
                </p>
                <p id="{{"note_delete_$note->id"}}" class="note-delete">
                  <a href="{{ route('notes.delete', ['noteId' => $note->id]) }}" class="dark-link">
                    <i class="fa-solid fa-trash"></i> Видалити
                  </a>
                </p>
              </div>
            </div>
          </div>
        @endforeach
          <div class="accordion-item hidden">
            <h2 class="accordion-header">
              <button id="accordion_button_0" class="accordion-button collapsed" type="button" data-
bs-toggle="collapse" data-bs-target="#collapse_0" aria-expanded="true" aria-controls="collapse_0">
                <span id="note_title_0"></span>
                <span id="note_time_0" class="note-time"></span>
              </button>
            </h2>
```

```
            <div id="collapse_0" class="accordion-collapse collapse" aria-labelledby="heading_0" data-bs-parent=".accordion">
                <div class="accordion-body">
                    <p><a class="note-task-link" id="note_task_link_0" href="">Посилання на завдання</a></p>
                    <p id="note_description_0"></p>
                    <div id="note_course_0" class="note-subject"></div>
                    <p id="deadline_date_0" class="note-deadline">
                        <img src="{{asset('/img/deadline.svg')}}" alt="deadline">
                    </p>
                    <p id="note_delete_0" class="note-delete">
                        <a href="" class="dark-link">
                            <i class="fa-solid fa-trash"></i> Видалити
                        </a>
                    </p>
                </div>
            </div>
        </div>
    </div>
</div>
@include('modals.create-note-modal')

Ініціалізація поля форми для груп
$(document).ready(function() {
    $('#task-groups').select2({
        placeholder: 'Додати групи, яким надіслати завдання',
        templateSelection: formatSelection,
        ajax: {
            url: '/api/groups',
            dataType: 'json',
            delay: 250,
            processResults: function (data) {
                return {
                    results: $.map(data, function (item) {
                        return {
                            text: item.code,
                            id: item.id
                        }
                    })
                };
            },
            data: function (params) {
                return {
                    search: params.term
                };
            },
```

```
        },
    });

    function formatSelection(option, container) {
        $(container).addClass('groups-selections');
        return option.text;
    }

    const groupIdEls = $('[name=groupId]');
    let groupIds = [];
    for (let groupIdEl of groupIdEls ) {
        groupIds.push($(groupIdEl).val())
    }

    $('#task-groups').val(groupIds).trigger('change');
});
```

ДОДАТОК Б

Створення системи автоматизованого планування бізнес-процесів для контингенту кафедри

Апробація наукових досліджень

Аркушів 4

Київ – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Одеський національний технологічний університет
Університет Інформатики і прикладних знань, м.Лодзь, Польща
Національний технічний університет України «Київський політехнічний інститут»
Навчально-науковий інститут комп'ютерних систем і технологій «Індустрія 4.0» ім. П.М. Платонова

XXIII Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів

«СТАН, ДОСЯГНЕННЯ ТА ПЕРСПЕКТИВИ ІНФОРМАЦІЙНИХ СИСТЕМ І ТЕХНОЛОГІЙ»

Матеріали конференції

Одеса

20-21 квітня 2023 р.

студентів. Виявивши будь-які проблеми чи питання щодо контенту та вживши заходів для їх вирішення, можна забезпечити максимальну ефективність та результативність дистанційних курсів в системі дистанційного навчання ОНТУ.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Learning Management System (LMS) Market Overview // MARKET RESEARCH FUTURE: [Веб-сайт]. URL: https://www.marketresearchfuture.com/reports/ learning-management-system-market-1858 (viewed on: 28.02.2023).
2. LMS Market Size and Spending Statistics // TrustRadius: [Веб-сайт]. URL: https://www.trustradius.com/vendor-blog/lms-statistics-trends (viewed on: 28.02.2023).

УДК 004.9

СИСТЕМА АВТОМАТИЗОВАНОГО ПЛАНУВАННЯ БІЗНЕС-ПРОЦЕСІВ ДЛЯ КОНТИНГЕНТУ КАФЕДРИ

ЛЕВКУН Д.П., БАНДУРКА О.І., СВИНЧУК О.В. (levkun.dasha@gmail.com),
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Головною метою системи є планування бізнес-процесів для контингенту кафедри, що полягає в систематизації навчальних досягнень студента, створенні комплексу задач для планування часу як студента, так і викладача, а також зберігання та організація практичних завдань, запам'ятовування важливих навчальних активностей. Система дозволить підвищити ефективність управління освітнім процесом. Необхідною складовою системи є web-дизайн, який повною мірою забезпечує правильне керування програмою.

Розробка та впровадження системи автоматизованого планування бізнес-процесів для контингенту кафедри може стати важливим інструментом підвищення ефективності управління освітнім процесом. Така програма може виконувати наступні завдання:

- відслідковування та зберігання завдань, лабораторних та практичних робіт;
- планування часу студента та викладача;
- надання доступу про дані студента батькам для контролювання його активності;
- встановлення нагадувань про виконання задач.

Розроблена система дозволяє зробити навчальний процес простішим, більш організованим та контрольованим; уникати ситуацій із загубленими завданнями та забутими оцінками. Хоча система є аналогічною до паперових аналогів (наприклад, журнали, щоденники, планери, журнали відвідування і т.д.), але головною відмінністю її є автоматизація процесу та тривале зберігання з можливістю доступного перегляду в будь-який час.

Час, який витрачається на створення користувацького інтерфейсу, має бути не меншим, ніж час витрачений на проектування архітектури системи. Перш за все, дизайн веб-інтерфейсу повинен бути зручним та інтуїтивно зрозумілим для користувачів [1]. Це означає, що інтерфейс повинен мати зрозумілу структуру, з якою користувач може легко орієнтуватися, швидко знаходити потрібну інформацію та виконувати необхідні дії. Тому для системи для планування навчальними процесами в університеті найкращим рішенням було б зробити інтерфейс схожим на справжні предмети, які використовуються в навчальному процесі, такі як журнали виставлення оцінок чи відвідування, папки з роботами кожної групи і т.д. Функціонал створення нотаток розміщується безпосередньо біля

розкладу, щоб відразу орієнтуватись в часі та задачах [2].

Для покращення ефективності роботи з системою планування важливо забезпечити високу швидкість завантаження веб-сторінок та підвищення їхньої продуктивності. Також, користувачам може бути надана можливість персоналізувати свій досвід використання системи, встановлюючи власні налаштування та фільтри для відображення інформації.

Однією з важливих задач дизайну веб-інтерфейсу для системи планування бізнес-процесів є забезпечення належного рівня безпеки. Оскільки в системі може зберігатися конфіденційна інформація, важливо забезпечити її захист від несанкціонованого доступу.

Також, важливо забезпечити зручність та зрозумілість процесу взаємодії користувача з системою. Наприклад, можна передбачити функцію відстеження статусу бізнес-процесу, яка буде дозволяти користувачам відслідковувати хід виконання завдань та отримувати повідомлення про їх статус [3].

Оскільки більшість користувачів використовує мобільні пристрої, важливо забезпечити оптимізацію системи для маленьких екранів, швидкий доступ до інформації та просту навігацію на телефоні чи планшеті [4]. При відвідуванні занять, студент частіше користується телефоном. Тому створення мобільної версії програми ще більше полегшує навчальний процес та допомагає відслідковувати нові події.

Важливо забезпечити користувачів докладними інструкціями та поясненнями про те, як користуватися системою. Наприклад, в системі можна застосовувати підказки для кожного поля форми, щоб пояснити, що потрібно заповнити в цьому полі.

Отже, дизайн веб-інтерфейсу повинен бути зручним та інтуїтивно зрозумілим для користувачів. Він повинен мати зрозумілу структуру, з якою користувач може легко орієнтуватися, швидко знаходити потрібну інформацію та виконувати необхідні дії. Саме на етапі дизайну варто створити умови для забезпечення високої продуктивності та захисту даних системи.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. В.Г. Коржик, І.А. Хоменко, *Інтерфейс користувача web-системи: дизайн та розробка*. К.: Видавництво НТУУ "КПІ", 2012.

2. С.І. Скороходько, В.А. Толочко, Л.В. Лихоліт, *Моделювання бізнес-процесів: системний підхід*. К.: Кондор, 2011.

3. І.Ю. Білоус, *Веб-дизайн. Експеримент.* Книга 2. К.: АртЕк, 2017.

4. J. Preece, Y. Rogers, H. Sharp, *Interaction design: beyond human-computer interaction*. John Wiley & Sons, 2015.

УДК 655.533

ОСОБЛИВОСТІ ПІДГОТОВКИ ПРЕДМЕТНОЇ ФОТОГРАФІЇ ДЛЯ НАВЧАЛЬНИХ ПОСІБНИКІВ ТА РОЗДАТКОВИХ МАТЕРІАЛІВ

ЛИПОВИЙ А.Є.
(arsenlipoviy@gmail.com),
НЕРОДА Т. В.
(tetyana.neroda@uad.edu.ua)
Українська академія друкарства

У роботі представлений опис вимог особливостей при підготовці предметної фотографії для навчальних посібників та роздаткових матеріалів та розглянуті відповідні варіанти для отримання цифрової сцени найкращої якості.