ВІДЗИВ НАУКОВОГО КЕРІВНИКА НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента **Шмалюка Д.В.**
(прізвище та ініціали)

представлену на здобуття освітнього ступеня «бакалавр»
зі спеціальності 123 «Комп'ютерна інженерія»

1. Робота виконана на тему Проект захищеної інформаційної Web-орієнтованої системи для оформлення індивідуального плану навчання.

2. Актуальність. Автоматизація процесів документообігу, знайдення нових інформаційних технологій управління процесами для мінімізації часу та максимізації зручності є важливою технічною задачею сьогодення. Дана робота направлена на підвищення якості надання адміністративних послуг закладом вищої освіти на прикладі факультету інформаційно-комп'ютерних технологій Житомирської політехніки. Перевагами системи є подача онлайн заяви на оформлення індивідуального графіку навчання із можливістю підкріплення всіх необхідних документів. Передбачено систему автентифікації користувачів, отримання згоди викладачів та формування необхідних документів для друку. Створення та впровадження такого роду систем в значній мірі спрощує процедуру оформлення такого роду документів та мінімізує час отримання підписів від викладачів щодо погодження.

(зазначити, які питання теоретичного і практичного характеру визначають актуальність роботи)

3. Основний зміст роботи викладено на 82 сторінках тексту, містить достатню частину ілюстрацій та таблиць. Список використаних літературних джерел налічує 46 найменувань.

4. Зміст розділів роботи *(зазначити за кожним розділом, які питання досліджені та з якою метою)*

Розділ 1. Проведено теоретичне дослідження методів, засобів та технологій побудови веб-додатків, методів захисту інформації, підходів до побудови інформаційних платформ. На підставі проведеного аналізу запропонована архітектура інформаційної платформи.

Розділ 2. Даний розділ присвячено питанням проектування та розробки серверної та клієнтської частини інформаційної платформи, представлено реалізацію основних модулів та модулю для оформлення індивідуального графіку. Реалізовано процедуру автентифікації та реєстрації користувачів, персональний кабінет користувача, сторінка для надання послуг користувачу, перегляд запитів на оформлення документів, глобальні повідомлення та домашня сторінка. Дані дослідження реалізують запропоновану архітектуру інформаційної платформи.

Розділ 3. Проведено тестування веб-додатку з метою перевірки працездатності. Описано процеси автентифікації та реєстрації користувача, процес оформлення індивідуального графіку.

5. Висновки обгрунтовані, повністю відповідають поставленим завданням, мають теоретичне та практичне значення. Практичне значення отриманих результатів полягає у розробці захищеної інформаційної Web-орієнтованої системи для оформлення індивідуального графіку навчання студентів з метою оптимізації, діджиталізації процесів, мінімізації часу співробітників університету та студента та підвищення якості надання послуг.

(обґрунтованість, теоретичне і практичне значення)

6. Список використаних літературних джерел. Джерела опрацьовані у повному обсязі,

(обсяг і повнота опрацювання за змістом роботи)

відповідають змісту роботи.

7. Додатки містять фрагменти коду програмного продукту

(перелік додатків та повнота їх опрацювання за змістом роботи)

8. Таблиці та ілюстрації Ілюстрації носять інформаційний характер, є доповненням до змісту

(призначення таблиць та їх використання для підтвердження досліджень в роботі)

роботи.

9. Якість оформлення роботи на високому рівні

10. Ставлення студента (ки) до виконання кваліфікаційної магістерської роботи відповідально і самостійно.

11. Висновок

В цілому при написанні кваліфікаційної роботи на здобуття освітнього ступеня «бакалавр» студент Шмалюка Д.В показав вміння самостійно підібрати та опрацьовувати необхідну літературу, інформаційну базу, робити відповідні висновки і оформлювати свої дослідження у кваліфікаційній роботі. Поставлені перед студентом завдання виконано в повному обсязі на високому інденерно-технічному рівні. Робота є прикладною, пройшла тестування та планується до використання з 01 вересня 2020 року на факультеті інформаційно-комп'ютерних технологій Житомирської політехніки (акт впровадження додається до кваліфікаційної роботи).

Кваліфікаційна бакалаврська робота студента Шмалюка Д.В написана на високому рівні та може бути допущена до захисту в Екзаменаційній комісії.

Керівник,
доцент кафедри КІтаКБ
Державного університету
«Житомирська політехніка»
канд. техн. наук, доц. _____ Надія Лобанчикова

10.06.2020 р

РЕЦЕНЗІЯ

на кваліфікаційну роботу студента денної форми навчання

спеціальності 123 Комп'ютерна інженерія

освітній ступінь «бакалавр» Д.В.Шмалюк

1. Тема: Проект захищеної інформаційної Web-орієнтованої системи для оформлення індивідуального плану навчання студент

2. Актуальність теми випускної кваліфікаційної роботи: тема роботи є актуальною, адже питання захисту інформації та забезпечення збереження конфіденційних даних в публічних мережах є наразі дуже актуальним, адже вимоги прозорості змушують організації реалізовувати документообіг через глобальні мережі, що становить потенційну небезпеку втрати інформації.

3. Наукова та практична цінність роботи: в роботі реалізовано проект та функціонал захищено інформаційної Web-орієнтованої системи із використанням точки доступу API, веб-складова реалізована на базі бібліотеки React JS.

4. Недоліки: незначні стилістичні та орфографічні помилки, питання тимчасового збереження активних ключів у тимчасовому сховище браузера, становить потенційну можливість для інцидентів, тому потребує доопрацювання. Вказані недоліки не є суттєвими, та не зменшують практичну цінність роботи

5. Загальний висновок:

Вцілому, що кваліфікаційна робота студента Д.В. Шмалюка на тему: Проект захищеної інформаційної Web-орієнтованої системи для оформлення індивідуального плану навчання відповідає профілю підготовки за спеціальністю 123 «Комп'ютерна інженерія», виконана на високому рівні, а студент заслуговує на присвоєння освітнього ступеня бакалавр.

Рецензент, к.т.н., доцент
зав. кафедри КН _____ І.І. Сугоняк

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра
на тему:
«Проект захищеної інформаційної Web-орієнтованої системи для оформлення індивідуального плану навчання студента»

Виконав студент 4-го курсу, групи КІ-2
спеціальності 123 «Комп'ютерна інженерія»
_____ Шмалюк Д.В.
Керівник, декан ФІКТ, канд. техн. наук, доцент
_____ Лобанчикова Н.М.
Професор кафедри ІПЗ, д-р пед. наук, доцент
_____ Вакалюк Т.А.
Рецензент, зав. кафедри КН, канд. техн. наук, доц.
_____ Сугоняк І.І.

Житомир 2020

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА КІБЕРБЕЗПЕКИ
СПЕЦІАЛЬНІСТЬ 123 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерної інженерії
та кібербезпеки
Офіменко А. А.
«06» квітня 2020 р.

ЗАВДАННЯ
на випускну кваліфікаційну роботу

Студента **Шмалюк Дмитра Володимировича**

Тема роботи: **Проект захищеної інформаційної Web-орієнтованої системи для оформлення індивідуального плану навчання студента**
затверджена Наказом по університету від **«06» квітня 2020 р. № 98с**
Термін здачі студентом закінченої роботи – **1 червня 2020 р.**
Вихідні дані для роботи:

1. Аналіз освітніх процесів для діджиталізації
2. Прогресивні веб-додатки
3. Розподілення **server-side** та **client-side** частини
4. Обробка документів за допомогою інформаційної платформи
5. Аналіз інтеграції веб-додатку з наявними сервісами

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Лобанчикова Н.М.	06.04.20	06.04.20
2	Лобанчикова Н.М.	20.04.20	20.04.20
3	Лобанчикова Н.М.	11.05.20	11.05.20

Керівник _____ Лобанчикова Н.М.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	6 квітня 2020 – 7 квітня 2020	Виконано
2	Пошук, огляд та аналіз літературних джерел щодо стеку технологій, для розробки прогресивного веб-додатку	8 квітня 2020 – 11 квітня 2020	Виконано
3	Аналіз технологій для розробки серверної частини, та налаштування периметру захисту	12 квітня 2020 – 16 квітня 2020	Виконано
4	Аналіз можливостей прогресивного веб-додатку	17 квітня 2020 – 19 квітня 2020	Виконано
5	Проектування структури та архітектури client-side та server-side	20 квітня 2020 – 30 квітня 2020	Виконано
7	Розробка веб-додатку	1 травня 2020 – 18 травня 2020	Виконано
11	Тестування роботи веб-додатку	19 травня 2020 – 22 травня 2020	Виконано
12	Аналіз результатів тестування	23 травня 2020 – 24 травня 2020	Виконано
13	Написання пояснювальної записки	25 травня 2020 – 29 травня 2020	Виконано
14	Підготовка презентації та виступу	30 травня 2020 – 31 травня 2020	Виконано
15	Отримання відгуку керівника та рецензії на кваліфікаційну роботу	4 червня 2020 – 8 червня 2020	Виконано
16	Захист	13 червня 2020	Виконано

Студент _____ Шмалюк Д.В.

Керівник _____ Лобанчикова Н.М.

РЕФЕРАТ

Пояснювальна записка до дипломного проекту випускника освітньо-кваліфікаційного рівня бакалавр на тему «Проект захищеної інформаційної Web-орієнтованої системи для оформлення індивідуального графіку навчання» викладена на 121 с., містить 58 рисунків, 46 літературних джерел.

Об'єкт дослідження – технології розробки Web-орієнтованих систем, процеси збереження, обробки, передачі та захисту інформації, процедура оформлення індивідуального графіку в закладах вищої освіти.

Предмет дослідження – методи, засоби та технології розробки захищеної інформаційної Web-орієнтованої системи, процедура оформлення індивідуального графіку в Житомирській політехніці

Мета дослідження – розробка захищеної інформаційної Web-орієнтованої системи для оформлення індивідуального графіку навчання студентів з метою оптимізації, діджиталізації процесів, мінімізації часу співробітників університету та студента, підвищення якості надання послуг.

У першому розділі дипломного проекту представлено результати аналізу методів та засобів, які будуть використовуватися під час створення веб-додатку. За допомогою інформації про наявні процеси, було виявлено ряд недоліків, які будуть мінімізовані у процесі виконання досліджень. У другому розділі дипломного проекту запропоновано архітектури проекту та визначені особливості. Запропонована інформаційна платформа є відкритою, придатною до розширення своїх функцій, адаптована до використання у закладах вищої освіти.

В третьому розділі продемонстрований процес розробки, та процес тестування додатку.

Ключові слова: ДІДЖИТАЛІЗАЦІЯ, ВЕБ-СИСТЕМА, ІНДИВІДУАЛЬНИЙ ГРАФІК, ІНФОРМАЦІЙНА ПЛАТФОРМА, ІНФОРМАЦІЙНА СИСТЕМА.

ABSTRACT

Explanatory note to the diploma project of the graduate of educational qualification level bachelor on the topic "Project of the protected information Web-oriented system for registration of the individual schedule of training" is stated on __ pages., contains ___ pictures, ___ literary sources.

The object of research - technologies for the development of Web-oriented systems, the processes of storage, processing, transmission and protection of information, the procedure for drawing up an individual schedule in higher education.

Subject of research - methods, means and technologies of development of the protected information Web-oriented system, procedure of registration of the individual schedule in Zhytomyr polytechnic

The purpose of the study is to develop a secure information Web-oriented system for the design of individual study schedules of students in order to optimize, digitalize processes, minimize the time of university staff and students, improve the quality of services.

The first section of the thesis project presents the results of the analysis of methods and tools that will be used when creating a web application. With the help of information about the existing processes, a number of shortcomings were identified, which will be minimized in the process of research. The second section of the diploma project offers project architectures and identifies features. The proposed information platform is open, suitable for expanding its functions, adapted for use in higher education institutions.

The third section demonstrates the development process and the application testing process.

Keywords: DIGITALIZATION, WEB SYSTEM, INDIVIDUAL SCHEDULE, INFORMATION PLATFORM, INFORMATION SYSTEM.

ЗМІСТ

РЕФЕРАТ 4

ABSTRACT 5

ЗМІСТ 6

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 8

ВСТУП 9

РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ РОЗРОБКИ ВЕБ ДОДАТКУ 11

1.1 Постановка задачі 11

1.2 Аналіз технологій для створення інформаційної платформи 13

1.2.1 Технології для розробки серверної частини 13

1.2.3 Технології для розробки client частини 17

1.3 Аналіз методів захисту інформації 21

1.3.1 Методи шифрування паролю 21

1.3.2 Механізм зв’язку між клієнтом та сервером 23

1.3.3 Архітектура інформаційної платформи 26

1.3.3.1 Архітектура client частини 26

1.3.3.2 Архітектура серверної частини 29

1.3.3.3 Проектування бази даних 31

Висновки до першого розділу 33

РОЗДІЛ 2. РОЗРОБКА ВЕБ-ДОДАТКУ 34

2.1 Розробка серверної частини 34

2.1.1 Встановлення та налаштування серверу 34

2.1.2 Реалізація основних модулів 40

2.1.2.1 Реалізація модуля для оформлення індивідуального графіку 49

2.2 Розробка client частини 53
2.2.1 Автентифікація та реєстрація користувачів 54
2.2.2 Персональний кабінет користувача 60
2.2.3 Сторінка для надання послуг користувачу 66
2.2.3.1 Послуга оформлення індивідуального графіку навчання 67
2.2.4 Перегляд запитів на оформлення документів 71
2.2.5 Глобальні повідомлення 72
2.2.6 Домашня сторінка 73
Висновки до другого розділу 75
РОЗДІЛ 3. ТЕСТУВАННЯ ВЕБ-ДОДАТКУ 76
3.1 Автентифікація та реєстрація користувача 76
3.2 Особистий кабінет 78
3.3 Процес оформлення індивідуального графіку 82
Висновки до третього розділу 84
ВИСНОВКИ 85
ПЕРЕЛІК ПОСИЛАНЬ 86
ДОДАТКИ 89
Додаток А 90
Додаток Б 110

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

PWA – Progressive Web Application (з англ. прогресивний веб додаток)

API – Application Programming Interface (з англ. програмний інтерфейс додатку)

СУБД – Система Управління Базами Даних

БД – База даних

ОРМ – Об’єктно-реляційне відображення

HTML – Hyper Text Markup Language

IDE – Integrated Development Environment

FTP – File Transfer Protocol

CSS – Cascading Style Sheets

MySQL – вільна система керування реляційними базами даних

SQL – Structured Query Language

ВНЗ – Вищий навчальній заклад

PDF – Portable Document Format

DOM – Document Object Model (з англ. об’єктна модель документа)

ВСТУП

На сьогоднішній день багатьом галузям довелося розпочати трансформацію для адаптації до цифрової ери. Цифрові технології використовуються як в роботі, так і для дозвілля. Цифрові технології існують десятиліттями і продовжують розвиватися настільки інтенсивно, що навіть їх області та створені ними можливості ще не були опановані та окреслені навіть приблизно. Коли цифрові технології поєднуються з фінансовими, соціальними, гуманітарними та іншими технологіями, з'являється можливість реалізації «нової реальності», включаючи «цифрову» освіту. Таким чином, діджиталізація змінює середовище освіти в університетах і змінює самі університети. Зв'язок змін в університетах із загальною діджиталізацією суспільства та самого університету став загальноприйнятою ідеєю. Існують різні обговорення компонентів діджиталізації університетів, але в даній кваліфікаційній роботі, буде розглядатися електронний документообіг.

Діджиталізація процесів в університеті, в нашому випадку документообігу, є важливою, та невід'ємною частиною. Виділимо деякі переваги цього: співробітник або викладач університету, має можливість редагування та аналізу документів в системі, в режимі реального часу; збереження документів, довідок, тощо, позбавляються можливості бути фізично пошкодженими, тож збереження документів, стає більш надійним; пошук потрібних документів, тощо, здійснюється за лічені хвилини; оформлення певних документів стає більш зручним, як для викладача, або співробітника університету, так і для студента. Важливим моментом при розробці інформаційної платформи, є захист даних, отож частину уваги, як при плануванні, так і при розробці архітектури, буде приділено методам захисту інформації. Таким чином, метою даної кваліфікаційної роботи є розробка захищеної інформаційної платформи, яка дасть можливість запустити налагоджений процес діджиталізації, для того щоб, покращити процеси документообігу та якості надання адміністративних послуг.

Об'єкт дослідження – технології розробки Web-орієнтованих систем, процеси збереження, обробки, передачі та захисту інформації, процедура

оформлення індивідуального графіку в закладах вищої освіти.

Предмет дослідження – методи, засоби та технології розробки захищеної інформаційної Web-орієнтованої системи, процедура оформлення індивідуального графіку в Житомирській політехніці

Мета дослідження – розробка захищеної інформаційної Web-орієнтованої системи для оформлення індивідуального графіку навчання студентів з метою оптимізації, діджиталізації процесів, мінімізації часу співробітників університету та студента та підвищення якості надання послуг.

Для досягнення вказаної мети необхідним є вирішення наступних інженерних задач:

- провести аналіз методів засобів та технологій розробки Веб-орієнтованих інформаційних систем та інформаційних платформ;
- провести аналіз методів, засобів та технологій захисту інформації;
- провести аналіз процедури оформлення індивідуального графіку студентів;
- розробити проект захищеної інформаційної Web-орієнтованої системи для оформлення індивідуального графіку навчання;
- реалізувати основні модулі системи та модуль для оформлення індивідуального графіку студента;
- реалізувати процедури захисту інформації для безпечної експлуатації розробленої системи;
- провести розробку модулів взаємодії всіх учасників процесу оформлення індивідуального графіку з належним рівнем захисту;
- провести тестування системи.

Методологія роботи – системний аналіз, методи об’єктно-орієнтованого проектування та програмування, методологія розробки Web-орієнтованих систем, теорія захисту інформації.

Запропонована система доведена до практичного використання та впроваджується на факультеті інформаційно-комп’ютерних технологій Житомирської політехніки.

РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ РОЗРОБКИ ВЕБ ДОДАТКУ

1.1 Постановка задачі

Підставою для розробки захищеної інформаційної Web-орієнтованої системи для оформлення індивідуального графіку навчання є завдання на кваліфікаційну роботу.

Метою створення даної інформаційної платформи є запуск процесу діджиталізації в закладах вищої освіти, на прикладі Державного університету «Житомирська політехніка».

Дана інформаційна система призначена для оформлення індивідуального графіку навчання студентів факультету інформаційно-комп'ютерних технологій Державного університету «Житомирська політехніка» через Web-додаток.

Результатом реалізації поставленого завдання є створення PWA додатку, для оптимізації процесу оформлення індивідуального графіку, впровадження електронного документообігу в університеті, підвищення якості надання адміністративних послуг, мінімізувати витрати часу на оформлення індивідуального графіку навчання.

Умови експлуатації та параметри навколишнього середовища: розроблену інформаційну систему планується використовувати у навчальних закладах на засобах обчислювальної техніки (персональних комп'ютерах, смартфонах та інших сучасних мобільних пристроях), тому система має працювати, якщо працює пристрій. Особливістю використання є можливість багато потокового звернення до сервера в один і той самий час.

Вимоги до технічних характеристик: придатна до використання під операційними системами Windows, Linux, MacOS, Android, з використанням всіх новітніх браузерів таких як, Chrome, Firefox, Edge. Вимоги до оперативної пам'яті пристроїв для клієнтської частини – для комфортного використання новітніх браузерів потрібно більше 8 ГБ. оперативної пам'яті, що цілком буде достатньо для комфортної роботи з інформаційною платформою, вимоги до оперативної пам'яті

серверної частини – 16 ГБ. Наявність встановленого програмного забезпечення для серверної частини: встановлений пакет Node.js

Вимоги до надійності: система відновлювальна, резервована, стійка до зламу, стабільна в роботі, мінімізувати «відмову в обслуговуванні», «відмову у доступі».

Вимоги до інтерфейсу: інтерфейс повинен буди максимально комфортним для використання. Розроблений під стиль державного університету «Житомирська політехніка». Деякі моменти в інтерфейсі повинні мати підказки. При виникненні помилки на сервері, інтерфейс повинен реагувати на цю помилку та повідомляти користувачу деталі помилки.

1.2 Аналіз технологій для створення інформаційної платформи

Метою даного розділу, є аналіз методів, та засобів для створення, в першу чергу захищеної, та масштабованої інформаційної платформи.

Під найменування інформаційна платформа розуміється веб-додаток, який має властивість масштабування. В даній кваліфікаційній роботі буде розглянуто базовий функціонал, такий як, реєстрація, авторизація, аутентифікації, тощо, та функціональність яка буде спрямована на діджиталізацію процесу оформлення індивідуального графіку навчання.

Інформаційна платформа буде складатися з двох частин, API (серверної частини) та клієнту. За допомогою правильного розподілу обов'язків, в нашому випадку, серверна частина відповідає тільки за обробку, захист, та збереження конфіденційної інформації, а клієнтська частина відповідає за відображення, та побудову сторінок для користувача. Побудові такого роду додатків присвячено багато робіт як закордонний так і вітчизняних науковців та програмістів. Зокрема в межах даної кваліфікаційної роботи було проаналізовано ряж наукових та технічних публікацій, основні з яких представлено в переліку посилань [1-46].

1.2.1 Технології для розробки серверної частини

Зростаюча популярність JavaScript принесла з собою багато змін, та процес веб-розробки сьогодні кардинально відрізняється. Те, що ми можемо робити в Інтернеті сьогодні з JavaScript, що працює на сервері, а також у браузері, важко було уявити лише кілька років тому.

Node.js [1-3] – це серверна платформа, побудована на JavaScript Chrome Engine (V8 Engine). Node.js був розроблений Райаном Далом у 2009 році. Node.js був створений для зручного створення швидких та масштабованих мережевих додатків. Node.js використовує керовану подіями модель, що не блокує введення / виведення, що робить її легкою та ефективною, ідеальною для використання в реальному часі додатків.

Node.js має деякі важливі функції, які роблять його першим вибором архітекторів програмного забезпечення.

1. Асинхронний та керований подіями – всі API бібліотеки Node.js асинхронні, тобто не є блокуючими. Це по суті означає, що сервер на базі Node.js ніколи не чекає повернення даних API. Після виклику сервера переходить до наступного API, а механізм сповіщень про події Node.js допомагає серверу отримати відповідь від попереднього виклику API.

2. Дуже швидкий - побудована на V8 JavaScript Engine Google, бібліотека Node.js дуже швидка у виконанні коду.

3. Єдиний потік, але дуже масштабований - Node.js використовує єдину одно поточну модель з зацикленням подій. Механізм подій допомагає серверу реагувати не блокуючим образом і робить сервер високо масштабованим на відміну від традиційних серверів, які створюють обмежені потоки для обробки запитів. Node.js використовує одну потокову програму, і ця сама програма може надавати послуги набагато більшій кількості запитів, ніж традиційні сервери, такі як Apache HTTP Server.

4. Без буферизації - програми Node.js ніколи не буферують будь-які дані. Ці програми просто виводять дані шматками.

5. Ліцензія - Node.js випускається під ліцензією MIT

Отож, використовуючи Node.js як базу, для розробки серверної частини, ми отримуємо велику кількість переваг, але оскільки в даній кваліфікаційній роботі розглядається створення інформаційною платформи для освітнього закладу, потрібно пам’ятати, що про керування та підтримку БД які буде в собі містити велику кількість зв’язків, та таблиць. Для вирішення цього питання будемо використовувати OPM Sequelize.

Sequelize [1, 31] – це бібліотека реляційного відображення об'єктів (OPM) Node.js для Postgres, MySQL, MariaDB, SQLite та Microsoft SQL Server. Взагалі кажучи, структура ORM - це бібліотека додатків, яка дозволяє розробнику робити різні операції з базою даних в рамках програми, не записуючи SQL-запит. Sequelize має суцільну підтримку транзакцій, відносини, читання реплікацій тощо. Sequelize

слідує Semantic Versioning та підтримує Node v10 та вище. Декілька важливих моментів, які відповідають на питання чому саме Sequelize.

По-перше, Sequelize надає можливість керувати БД, за допомогою ОРМ [1-3, 31]. ОРМ - це методика для перетворення даних між несумісними системами типу за допомогою об'єктно орієнтованих мов програмування. Це фактично створює "віртуальну базу даних об'єктів ", яку можна використовувати з мови програмування.

По-друге, Sequelize слідує Semantic Versioning. Підтримка семантичної версії – це простий набір правил і вимог, які диктують, як номери версій присвоюються та збільшуються. Це дозволяє уникнути проблем з масштабуванням додатків які будуть використовувати Sequelize, та зробити обновлення до нової версії більш гнучким.

По-третє, Sequelize надає можливість працювати з такими БД як, Postgres, MySQL, MariaDB, SQLite та Microsoft SQL Server. Це надає можливість обрати БД яка найбільше підходить під вимоги проекту, таким чином, переходимо до вирішення питання, яка БД буде використана при розробці інформаційної платформи.

З висновків попереднього аналізу [1-46], про те що, інформаційна платформа буде мати велику кількість таблиці, та зв’язків, та з наявного списку БД, які підтримує Sequelize, при розробці інформаційної платформи буде використовуватися MySQL.

Багато найбільших та найшвидших у світі організацій, включаючи Facebook, Google, Adobe, Alcatel Lucent та Zappos, онлайн платформа для інтернет магазинів Magento, покладаються на MySQL, щоб заощадити час та гроші, використовуючи свої веб-сайти, бізнес-критичні системи та програмне забезпечення.

MySQL [46] є реляційна системи управління базами даних на основі SQL. Додаток використовується для широкого спектру цілей, включаючи програми зберігання даних, електронну комерцію та реєстрацію журналів. Однак найчастіше для MySQL використовується веб-база даних. З його

допомогою можна зберігати що завгодно, від одного запису інформації до цілого інвентаря доступних товарів для інтернет-магазину.

MySQL була вибрана не тільки за те, що цю БД, використовують найпопулярніші компанії світу, але й за деякими моментами, які важливі в процесі розробки:

1. Масштабованість на вимогу. MySQL надає перевагу неперевершеної гнучкості, яка полегшує ефективне управління глибоко вбудованими додатками, навіть у гігантських центрах обробки даних, у яких зберігається величезна кількість критично важливої інформації. Це дозволяє повне налаштування, щоб задовольнити унікальні вимоги підприємств електронної комерції зі значно меншим слідом. MySQL забезпечує максимальну гнучкість платформи для підприємств, які потребують додаткових функцій та функцій для своїх серверів баз даних.

2. Висока доступність. Постійна доступність є непохитною особливістю MySQL - підприємства, які її розгортають, можуть користуватися цілодобовим режимом роботи. MySQL Незалежно від того, який веб-додаток буде запущений, MySQL призначений для обробки мільйонів запитів і тисяч транзакцій, забезпечуючи унікальні кеші пам'яті, повнотекстові індекси та оптимальну швидкість.

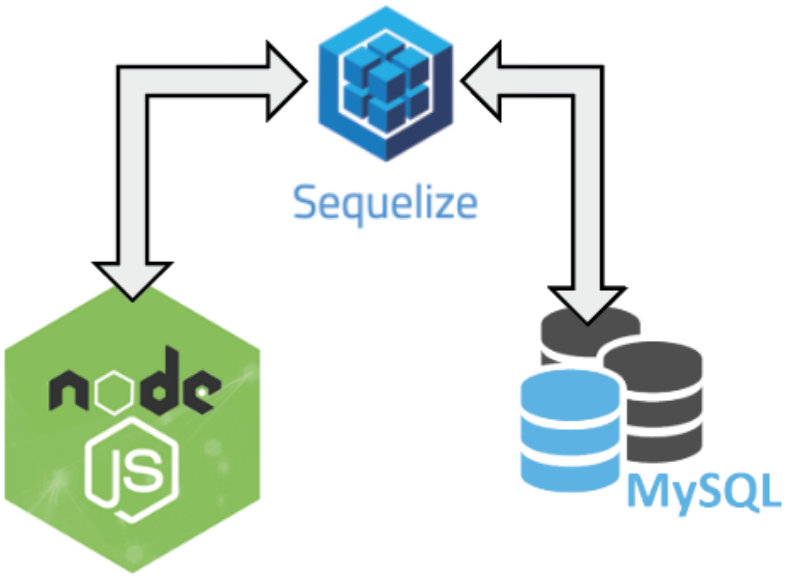


Рисунок 1.1 - Зв'язки між компонентами серверної частини

Таким чином, наша серверна частина складається з Node.js – обробка запитів, та підготовка відповідей, Sequelize – зв’язок з БД. Sequelize буде використовувати об’єкти JavaScript, відображені в таблиці реляційних баз даних (у нашому випадку - MySQL) і безперешкодно перетворює зміни об’єкта JavaScript запит, для зміни даних в БД, MySQL (рисунок 1.1).

Отже, з технологіями, які будуть використовуватися при розробці серверної частини визначалися, в наступному розділі, буде продемонстровано технології для розробки частини, яка відповідає за відображення інформації, та комунікацію з користувачем, client-side.

1.2.3 Технології для розробки client частини

Важливим питанням, яке виникає при розробці client-side, це швидкодія та масштабованість додатку. Нашим вимогам відповідає веб-додаток типу PWA.

PWA є абревіатурою для Progressive Web Apps [32]. Термін вперше був введений Google ще в 2015 році. "Прогресивна" частина, за словами Pete LePage - Google Developer Advocate, можна пояснити як «оскільки користувач з часом поступово вибудовує відносини з додатком, він стає все більш потужним».

В мережі інтернет, скоріш за все буде написано, що PWA в більшості випадків використовується для мобільного застосунку, та PWA не відноситься до стандартних веб-додатків, але це не зовсім так, прогресивні веб-додатки, все ще є веб-сайтами. Вони просто виглядають і поводять себе як звичайний додаток, завдяки сучасним веб-технологіям. Це означає, що користувачі відкривають прогресивні веб-додатки у своєму браузері з URL-адресою так само, як і будь-який веб-сайт, але одразу після того, як вони бачать PWA, вони отримують досвід використання "додатка" прямо у своєму браузері без необхідності завантаження та встановлення.

Для побудови інформаційної платформи на базі PWA будуть використані наступні технології. Почнемо з фундаменту, за основну частину веб-додатку буде відповідати React.

React [4] - це бібліотека JavaScript для створення інтерфейсів користувача. В основі всіх додатків React лежать компоненти. Компонент - це автономний модуль, який надає певну інформацію. За допомогою React можна створити елементи інтерфейсу, як кнопка або поле введення. Компоненти є композиційними та можуть включати в себе один або більше інших компонентів.

В цілому, для створення веб-додатків на базі React створюються компоненти, які відповідають різним елементам інтерфейсу. Потім ці компоненти організовуються в всередині компонентів вищого рівня, які визначають структуру веб-додатку. Важливо, що кожен компонент програми React дотримується суворих принципів управління даними.

Складні, інтерактивні користувацькі інтерфейси часто включають складні дані та стан програми. Для компоновки інтерфейсу, в React компонентах буде використовуватися Blueprint, це інструментарій з готових елементів інтерфейсу, який дозволяю побудувати будь-який компонент з лічені хвилини.

На відміну від багатьох своїх попередників, React оперує не безпосередньо на моделі об'єкта документа (DOM) браузера негайно, віртуальним DOM. Тобто, замість маніпулювання документом браузером після зміни наших даних (що може бути досить повільним), він змінює дані в віртуальному DOM і працює повністю в пам'яті. Після того як віртуальний DOM був оновлений, React інтелектуально визначає, які зміни потрібно внести до фактичної DOM-адреси браузера.

Для забезпечення прозорості, та якісного проекту на базі React, потрібно дотримуватися декількох важливих правил [38], які відносять до створення компонентів та архітектури React додатку, в цілому. Таким чином при реалізації веб-додатку, потрібно дотримуватися наступних правил:

1. Компоненти повинні бути невеликими та функціональними. в React можливе наявність величезних компонентів, які виконують ряд завдань. Але кращим методом проектування компонентів є збереження їх маленькими, щоб один компонент відповідав одній функції.

2. Компонент повинен мати можливість бути повторно використаним. Наприклад, якщо розробник створює компонент, який відображає текстове поле, цей компонент можливо буде використати в іншому місці.

3. Загальне правильно для написання будь яких додатків, яке говорить «Уникайте дублікату коду», це правило також відноситься й до React.

4. Коли проект тільки починається, більшість розробників створює стилі для компонентів в одному файлу, одночасно для всіх компонентів, це є невірно, кожен компонент повинен мати свій файл стилів.

5. Назва файлу, в більшості випадків повинна співпадати з назвою класу, або функції, яка знаходиться в файлі.

6. Використання CamelCase для назв компонентів. Це правило допомагає не тільки покращити візуальний вигляд коду, а також відрізнити звичайний HTML тег, від компоненту, одним реченням, це робиться для того, щоб JSX міг ідентифікувати від тегів HTML за замовчуванням.

7. Усі файли які відносяться до компоненту, повинні знаходитися в одній папці. Тобто, якщо розробник створює компонент з ім’ям Input, до нього повинна бути створена папка Input, а в цій папці сам компонент, та стилі які відносяться до цього компоненту.

Отже, після визначення основних аспектів, та правил написання React компонентів, переходимо до роботи зі станами в React.

Кожен React компонент має певні стани, які відносяться до компоненту, але виникає питання, як забезпечити глобальне збереження даних, по-перше для всіх компонентів, а по друге, для того щоб запобігти лишніх запитів до серверу, щоб вирішити це питання, в процесі розробки будемо використовувати технологію Redux.

Redux [12] – це контейнер для збереження станів, в нашому випадку для PWA додатків. За допомогою Redux, компоненти у веб додатку можуть описувати глобальний стан, наприклад чи користувач є авторизованим, або чи має користувач певні права для виконання операції, тощо. Таким чином, кожен компоненти у веб-

додатку може звертатися до станів іншого компоненту, без навантаженого коду, це важливий момент у проектуванні, та масштабуванні веб-додатку.

Отож, технологія для відображення та збереження певної інформації, для розробки PWA проаналізована, залишається одне не вирішене питання, яким чином, server-side та client-side будуть зв'язані, але це питання вже вирішили розробники Facebook, розробив мову запитів під назвою GraphQL.

GraphQL [14] – це мова запитів для API для виконання запитів із наявними даними. GraphQL надає повний і зрозумілий опис даних у вашому API, дає клієнтам можливість запитувати саме те, що їм потрібно, і нічого більше, полегшує еволюцію API з часом і надає потужні інструменти для розробників.

Як висновок, з пунктів 1.2.1 та 1.2.2, можна запропонувати наступну схему зв'язків (рисунок 1.2).

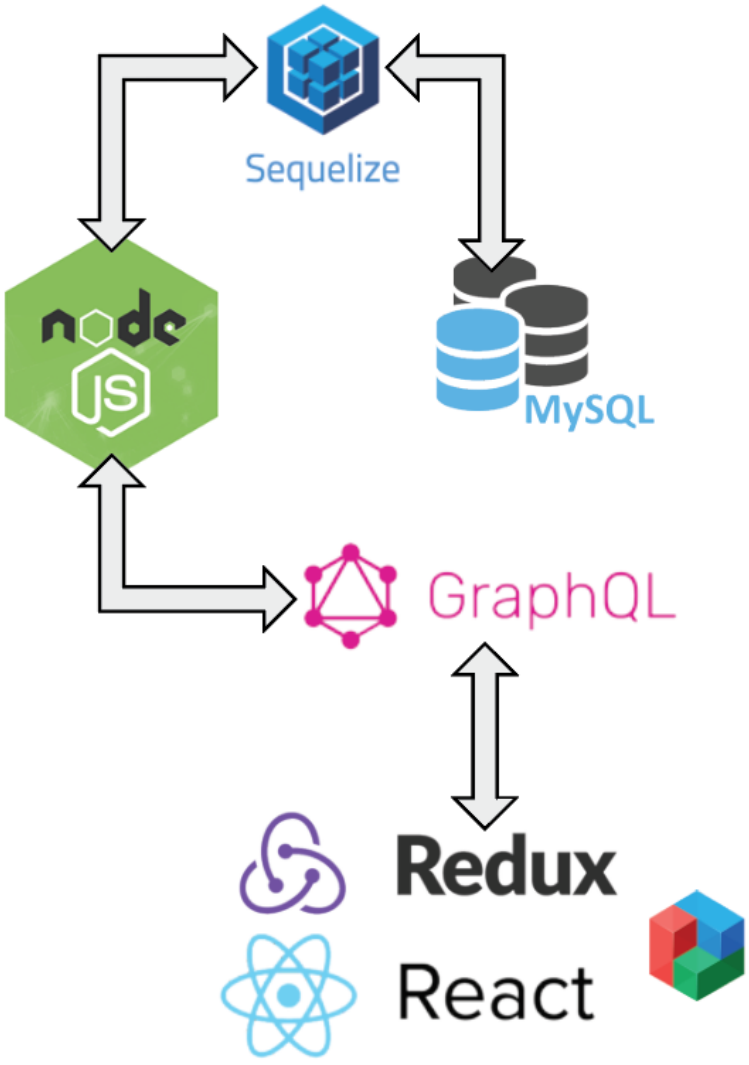


Рисунок 1.2 – Поверхнева схема зв'язку між сервером та клієнтом

1.3 Аналіз методів захисту інформації

В даному розділі будуть розглянуть питання, збереження та шифрування паролю, питання доступу до конфіденційної інформації.

1.3.1 Методи шифрування паролю

При розробці будь-якого веб-додатку, виникає питання про безпечне збереження даних користувачів, в першу чергу, збереження паролю. Для того щоб безпечно зберігати паролю, потрібно використовувати алгоритми шифрування, та в ніякому випадку не зберігати його у відкритому виді, отож Node.js має вбудований модуль криптографічних функцій. Він підтримує обчислення хешей, аутентифікацію з HMAC, шифрами тощо.

Для початку визначимось с тлумаченням базових термінів.

Хеш – це біт фіксованої довжини, який процедурно та детерміновано генерується з якогось довільного блоку вихідних даних. Деякі важливі властивості цих хешей (тип корисний для криптографії) включають:

– Фіксована довжина. Це означає, що, незалежно від введення, довжина хеша однакова. Наприклад, хеші SHA-256 завжди є 256 бітами, незалежно від того, чи є вхідними даними кілька біт або кілька гігабайт.

– Детерміновано: для одного і того ж вводу слід розраховувати, що зможете обчислити точно той же хеш. Це робить хеші корисними для контрольних сум.

– Стійкий до співпадіння: співпадіння – це коли однаковий хеш генерується для двох різних вхідних блоків даних. Алгоритми хешу розроблені таким чином, що вкрай малоймовірно мати співпадіння – наскільки малоймовірним є властивість алгоритму хешу. Важливість цієї властивості залежить від випадку використання.

– Односпрямований: Хороший алгоритм хешування легко застосувати, але важко скасувати. Це означає, що, з урахуванням хешу, немає жодного розумного способу з'ясувати, яким був оригінальний фрагмент даних.

В даній кваліфікаційній роботі буде використано, метод шифрування PBKDF2.

PBKDF2 – це те, що ми називаємо функцією виведення ключа на основі пароля. Якщо отриманий секрет використовується не як ключ, а як хеш-значення, його також називають хешем пароля. Хеші паролів відрізняються від захищених хешей тим, що вони містять salt та кількість робочих коефіцієнтів / ітерацій [11].

Криптографічні хеші, і хеші паролів - це одна з функцій, призначена для створення короткого виводу з фіксованим розміром із заданого входу. У випадку хешу пароля вхідним кодом буде пароль та salt. Розмір salt та кількість ітерацій зазвичай вважаються параметрами конфігурації; обидва, звичайно, впливають на вихід хеша пароля (рисунок 1.3).

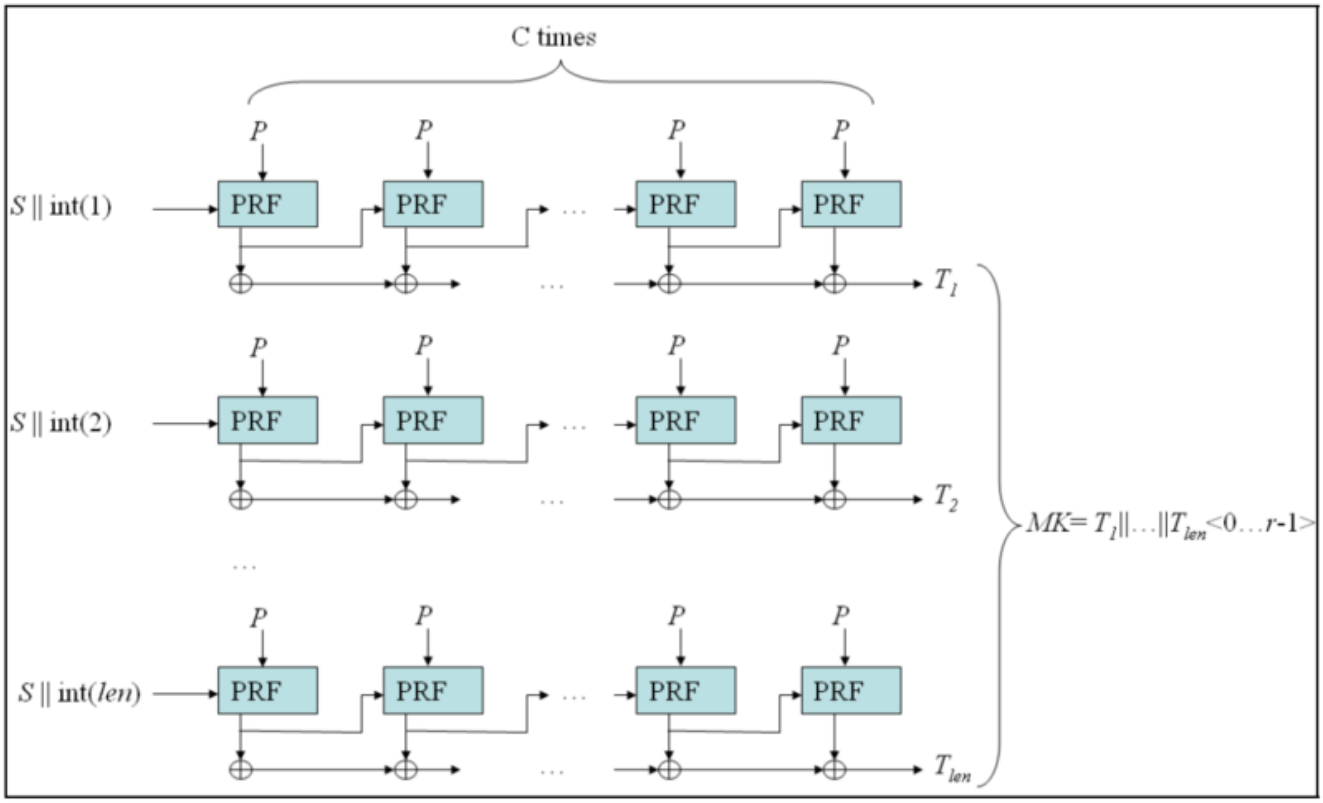


Рисунок 1.3 - Алгоритмічне зображення ітераційного процесу функції виведення ключових ключів на основі PBKDF2

Хеши паролів, як правило, створюються на основі псевдо випадкових функцій або PRF. Звичайною формою PRF є код автентифікації повідомлення на основі HMAC або Hash, який, в свою чергу, використовує хеш всередині країни. Коли використовуються PRF на основі хешу, такі як HMAC, тип / розмір використовуваного хешу часто можна налаштувати. Однак хеші паролів не

потрібно створювати за допомогою хешів. Будь-який інший PRF, наприклад, на основі симетричних шифрів, може це зробити. Наприклад, bcrypt побудований поверх Blowfish, блочного шифру.

Хеші паролів потребують salt, щоб ідентичні паролі не збігалися з одним і тим же хешем. Крім того, вони містять коефіцієнт роботи / кількість ітерацій, так що зловмиснику потрібно виконати більше роботи, щоб обчислити хеш кожного можливого пароля. Це потрібно, оскільки більшість паролів недостатньо випадкові. Без фактора роботи зловмисник зможе перевірити величезну кількість можливих паролів за допомогою грубої сили або нападу словника.

Оскільки було визначено метод шифрування основного компоненту веб-додатку, потрібно визначитися з питанням отримання доступу до конфіденційної інформації. В перших під-розділах було обґрунтовані, та описані технології які будуть використовуватися при розробці, але якщо дивитись з точки зору будування захисту для інформаційної платформи, ми маємо сервер, який працює тільки з отриманням, обробкою, та надсиланням інформації, та клієнт, який бере на себе всі інші функції, отож потрібно забезпечити механізм аутентифікації та авторизації користувача.

1.3.2 Механізм зв’язку між клієнтом та сервером

Для побудування зв’язку між клієнтом та сервером будемо використовувати клієнт-серверну модель. Модель клієнт-сервер - це розподілена структура додатків, яка розділяє завдання або робочі навантаження між постачальниками ресурсу або послуги, що називаються серверами, і запитувачами послуг, що називаються клієнтами. За допомогою обґрунтування аналізів в перших під-розділів, було визначено, що сервер буде побудований за допомогою моделі API. Інтерфейс прикладного програмування (API) являє собою обчислювальний інтерфейс, який визначає взаємодію між декількома проміжними програмним забезпеченням. Він визначає типи викликів чи запитів, які можна здійснювати, як їх здійснювати, формати даних, які слід використовувати, конвенції, які слід дотримуватися тощо. Він також може забезпечити механізми розширення, щоб користувачі могли

розширити наявну функціональність різними способами та різною мірою. Отож, кожен запит до нашої точки доступу, надалі будемо називати API, точкою доступу, має обробляти запити, тільки тих користувачів, які по-перше є аутентифіковані, по-друге є, авторизовані, для цього будемо використовувати унікальний ідентифікатор користувача, та для більш зручного використання інформації платформи, створимо сесію для користувача.

Кожен користувач інформаційної платформи, буде мити свій тип облікового запису, та роль. Інформаційна платформа наразі має три типи облікового запису, це звичайний користувач, студент та викладач, та три ролі, звичайний користувач, модератор та адміністратор.

Отож, з базовими компонентами визначились, переходимо до більш детальних речей, таких як а автентифікація, та авторизація.

Автентифікація – це механізм який є актом доведення в твердження, наприклад, особистість користувача, комп'ютерної системи. На відміну від ідентифікації, акт вказівки особи чи речі, автентифікація – це процес підтвердження цієї особи.

Авторизація - це функція визначення прав доступу / привілеїв до ресурсів, яка пов'язана з інформаційною безпекою та безпекою комп'ютерів загалом та зокрема з контролем доступу.

Таким чином, всі наші запити, до точки доступу, потрібно розділити на дві групи, перша група, запити які не потребують автентифікації, та авторизації користувача, та друга група, запити які потребують всі три механізми захисту, тобто автентифікація, авторизація та перевірка дійсності сесії користувача. До першої групи можемо віднести парольну автентифікацію, та звичайну реєстрацію користувача. До другої групи відносимо всі інші запити, які надають інформацію про користувачів, тощо.

Базуючись на попередньому аналізі, маємо можливість зобразити наступну схему зв’язку (рисунок 1.4).

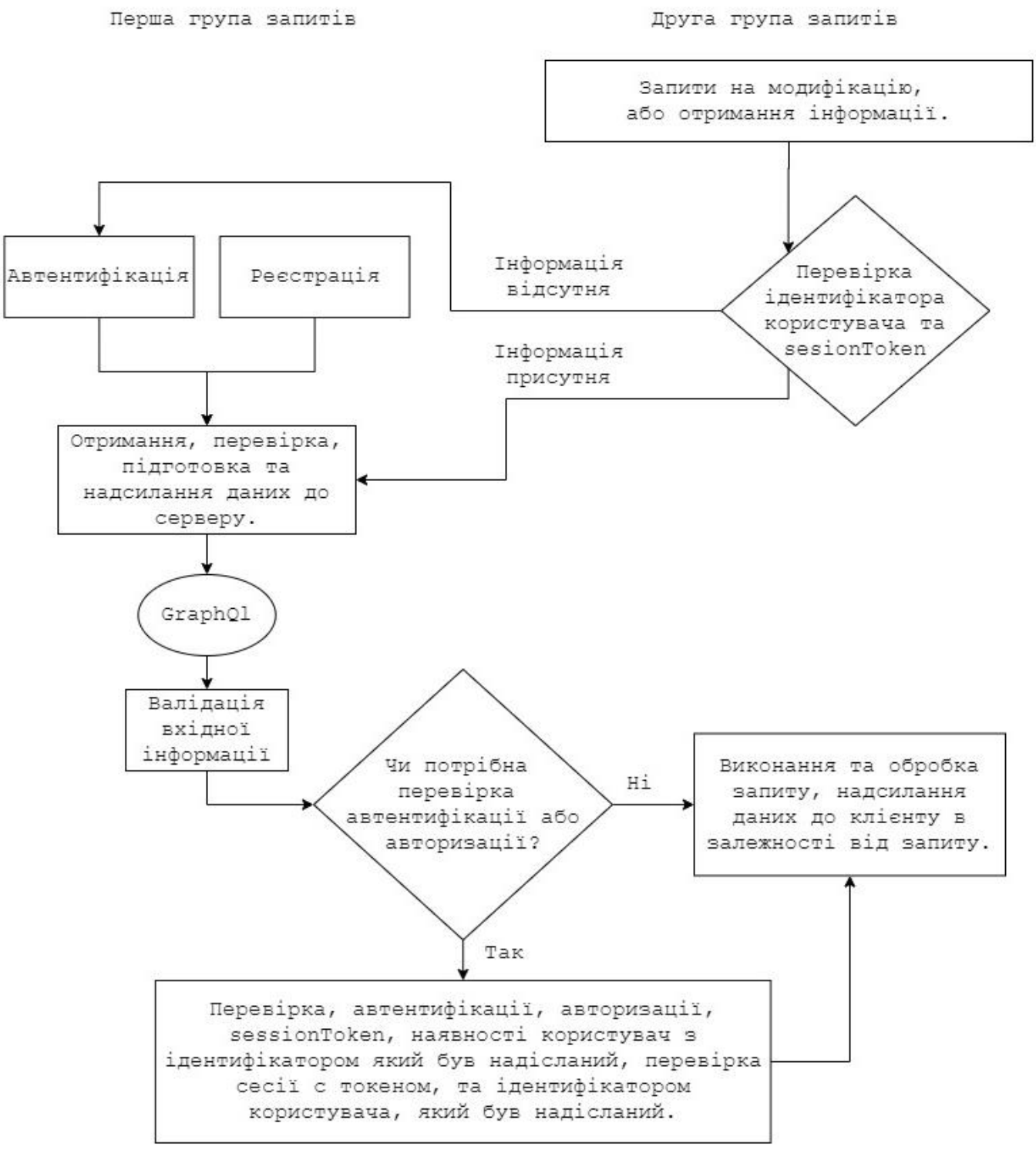


Рисунок 1.4 – Схематичне зображення механізму зв’язку між сервером та клієнтом, для двох типів запитів

Таким чином визначено механізми зв’язку між сервером та клієнтом, для двох типів запитів.

1.3.3 Архітектура інформаційної платформи

Одна із цілей розробки інформації платформи, це можливість масштабувати, та додавати нові функції до додатку, за це питання відповідає архітектура проекту. Архітектура веб-додатку полягає у прийнятті фундаментальних структурних виборів, які дорого змінити після їх впровадження. Вибір архітектури програмного забезпечення включає конкретні структурні варіанти від можливостей розробки програмного забезпечення, тому цей етап є дуже важливим. В попередніх пунктах було описано технології, методи захисту інформації, тому в цьому пункті буде описана теоретична архітектура інформаційної платформи, а вже в другому розділі буде проведена її розробка.

1.3.3.1 Архітектура client частини

Дуже важливим моментом при розробці архітектури було те, що потрібно максимально розгрузити сервер, тобто серверна частина не повинна генерувати HTML, або виконувати інші дії, які допомагають відображати інформацію для користувача. За це буде відповідати клієнт частина, яка буде побудована на базі React, Redux, GraphQl, тощо.

Архітектура клієнт частини складається з компонентів які спілкуються з сервером (actions), з UI компонентів, які використовуються для побудування інтерфейсу веб-додатку (components), з сторінок, які використовуються для навігації користувача, та для відображення певної інформації (pages), та з компонентів які відповідають за сервісні дії, такі як зберігання певних даних, на стороні клієнта, тощо, (рисунок 1.5).

Обов’язково потрібно звернути увагу, як зберігаються дані на клієнтській частині. Це є важливим, тому що клієнт не надсилає повторного запиту на сервер, якщо у клієнта є всі дані. У відмінності від звичайних веб-додатків, наша інформаційна платформа має таку можливість.

Отож збереження інформації на стороні клієнта, працює наступним чином, оголошується єдине сховище даних, інформація зберігається в певній комірці пам’яті, та для того щоб працювати з цією інформацією оголошується редуктори, в нашому випадку оголошується редуктор створення, оновлення або очищення комірки пам’яті з певною інформацією, (рисунок 1.6).

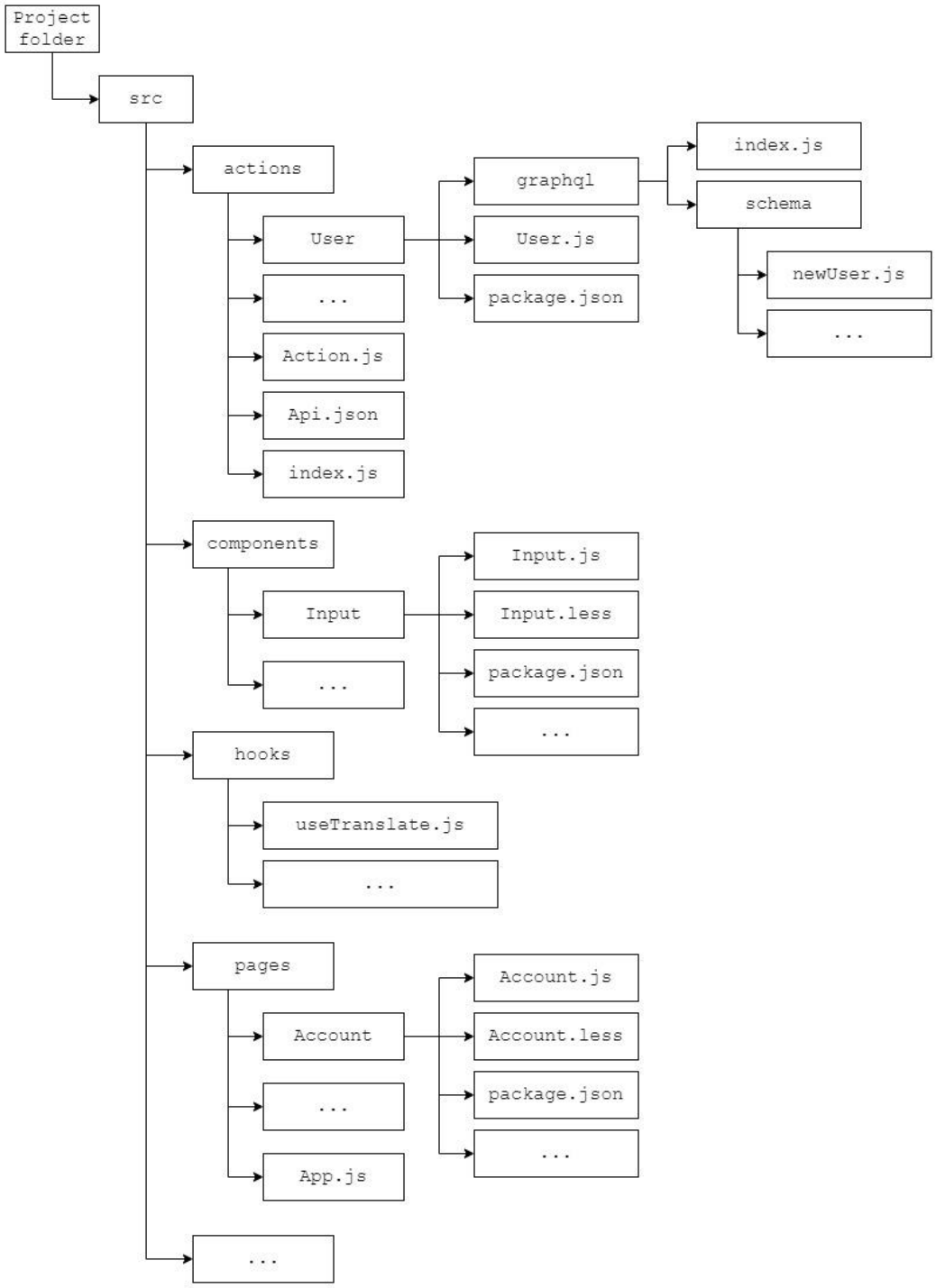


Рисунок 1.5 – Схематичне зображення архітектури клієнт частини, не враховуючи сервісні дії

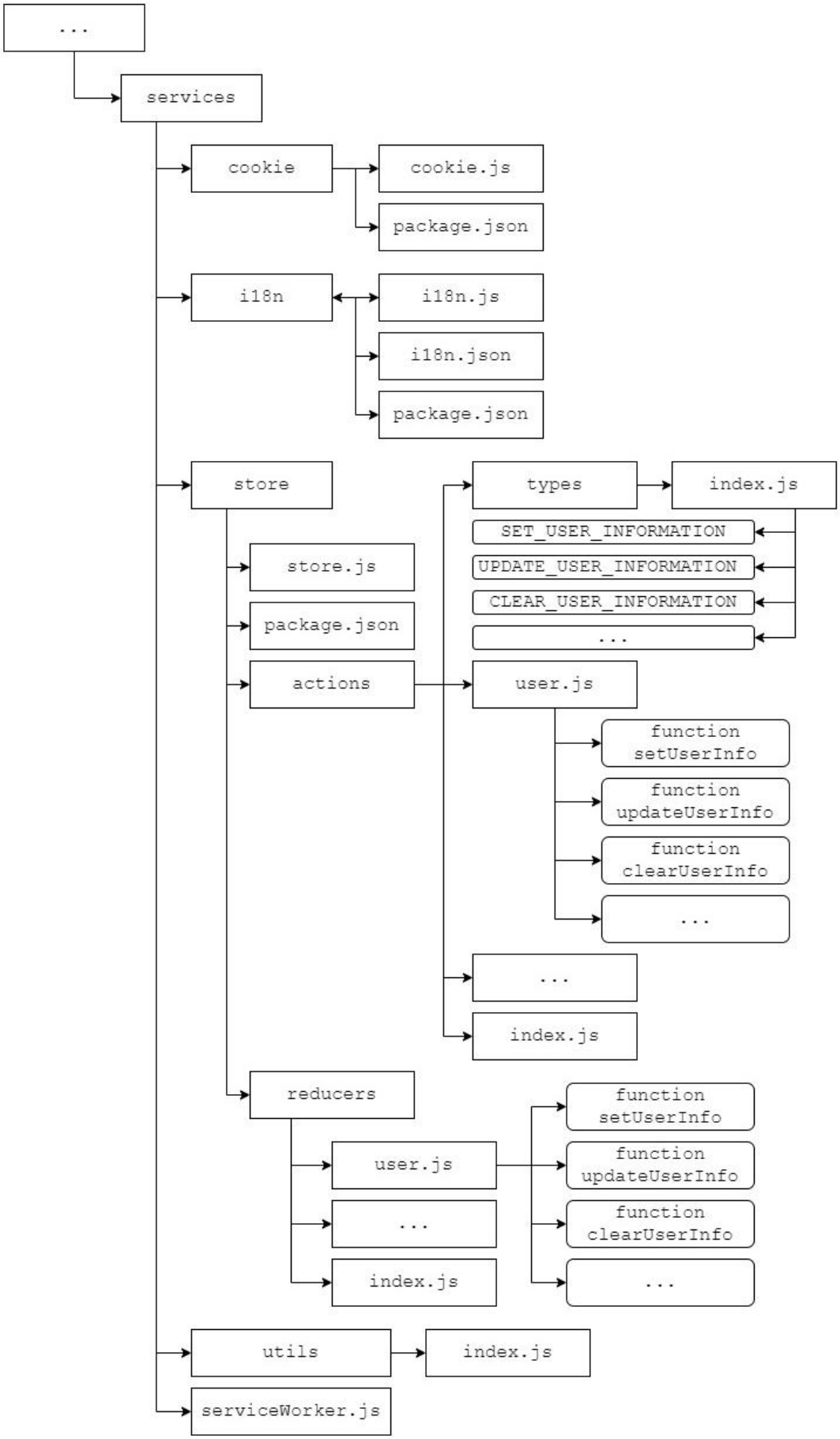


Рисунок 1.6 – Продовження рисунку 1.5, відображення сервісної частини проекту

1.3.3.2 Архітектура серверної частини

Серверна частина складається з стандартних для сервера компонентів, але є декілька моментів, які роблять наш сервер унікальним.

По-перше, сервер не відповідає за маршрутизацію, тобто сервер не має ніяких маршрутів, є єдина точка доступу, за допомогою якої, сервер спілкується з клієнтом.

По-друге, сервер не зберігає файли, які можна згенерувати з наявної інформації, це відноситься більше до оформлення певних документів. Це завдання надано клієнту. Клієнт отримаю всю інформацію, та генерує потрібні файли. Сервер зберігає дані, типу: довідка з місця роботи, студентський квиток, тощо.

По-третє, низька ресурсовитратність. Як вже було проаналізовано, сервер не генерує HTML сторінки, не зберігає файли, які можна згенерувати на базі наявних даних, не відповідає за маршрутизацію, всі ці моменти роблять сервер мінімально ресурсовитратним.

Отож, за реалізацію вище описаних пунктів, та за надання максимальної масштабованості при додаванні нових функцій, відповідає наступна архітектура серверної частини. (рисунок 1.7)

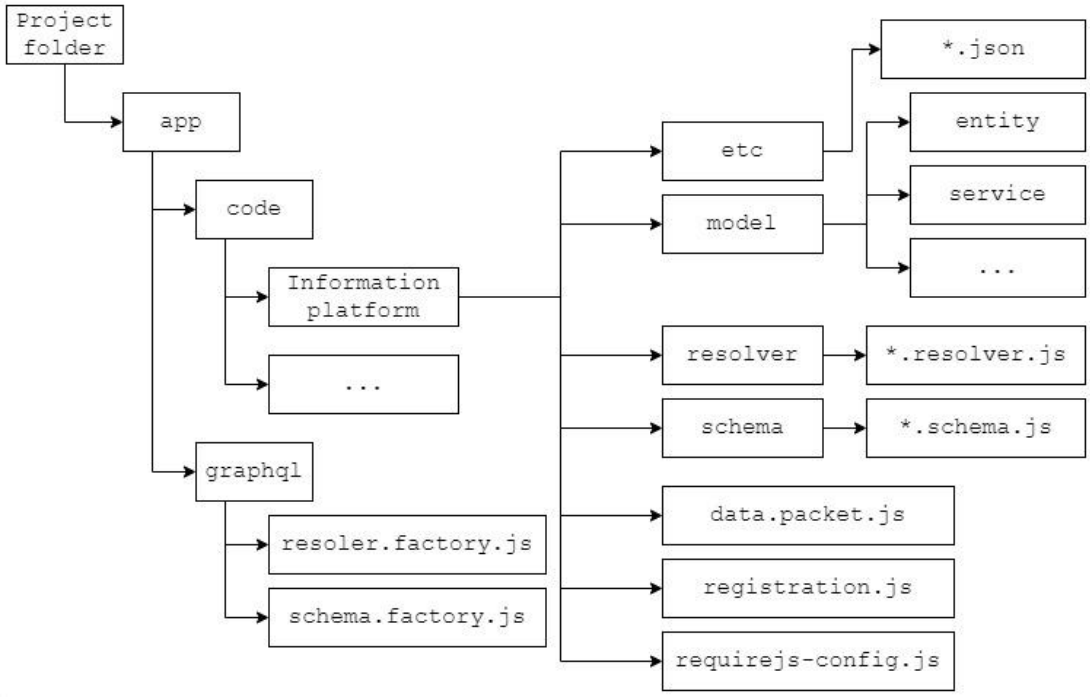


Рисунок 1.7 – Архітектура серверної частини

На рисунку 1.7 відображено який чином, побудована серверна частина. Сервер складається з модулів, які відповідають за певну функціональність, це надає можливість масштабування в майбутньому.

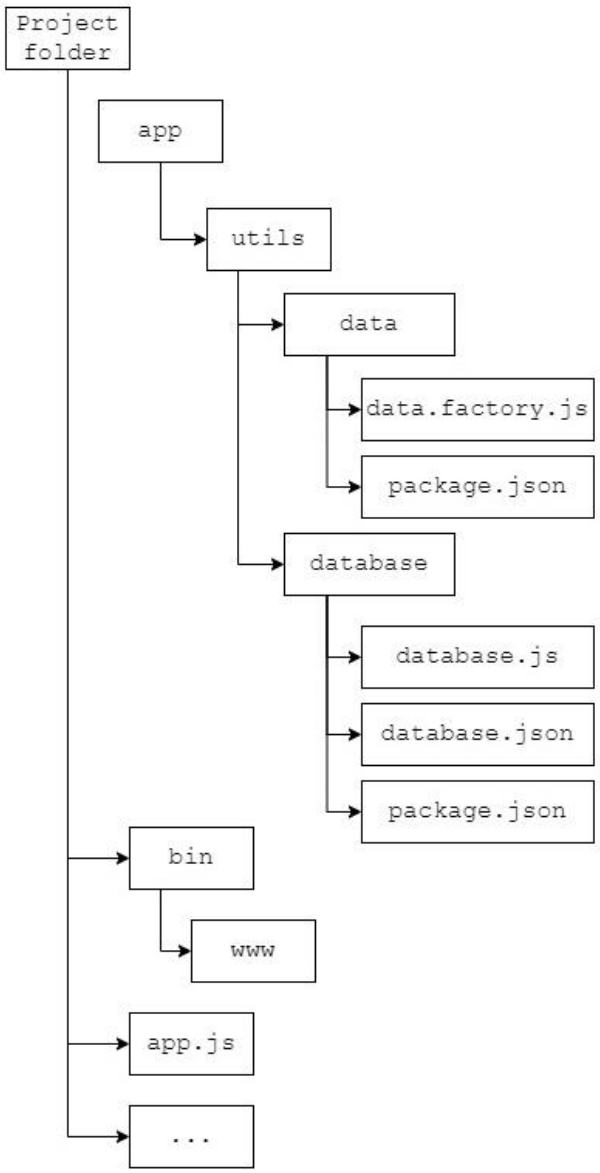


Рисунок 1.8 – Продовження рисунку 1.7

При розробці сервера буде використовуватись RequireJS. RequireJS - це завантажувач файлів і модулів JavaScript [5]. За допомогою RequireJS створюється можливість контролювати місце знаходження файлів, набагато простіше, саме тому, кожен розробник який буде працювати з інформаційною платформою, може добудовувати проект, без зміни архітектури.

Звернемо увагу на те, що при ініціалізації серверу потрібно надати GraphQL схему методів, для вирішення тих, чи інших запитів, тобто іншими словами

GraphQL повинен розуміти до якого метода йому звертатись, якщо наприклад на вхід приходить запит «newUser», відповідно що назва запиту та назва методу для вирішення цього запиту, повинні бути ідентичні. Нажаль GraphQL не надає можливості розділити всі схеми на під схеми, тому спеціально для вирішення цієї задачі створимо «schema.factory.js» та «resolver.factory.js», ці дві фабрики відповідають за збір, та ініціалізацію схем, та методів для вірної маршрутизації запитів. Ініціалізація схем та методів для вирішення запитів знаходяться в кожному модулі. Розробник має можливість створювати необмежовану кількість схем, та методів.

Також, потрібно передбачити можливість ініціалізації першочергових даних, наприклад створити облікові записи для викладачів, для цього буде створено «data.factory.js». За допомогою даної фабрики даних, сервер має можливість зібрати все дані, які мають бути створені, до запуску сервера, але «data.factory.js» тільки збирає дані, а для оголошення методів ініціалізації відбуваються в кожному модулі в пакеті даних, тобто «data.packet.js».

Таким чином, за допомогою такої архітектури проекту, для розробника надається максимальний набір інструментів, для простого створення нового функціоналу. Весь новий функціонал знаходиться в окремому модулі, та має можливість звертатися до вже створених модулів, та використовувати їх функціонал.

Отож, після вирішення та затвердження архітектури клієнтської, та серверної частини, потрібно подбати, про збереження інформації, перейдемо до розробки бази даних.

1.3.3.3 Проектування бази даних

Розпочнемо проектування бази даних з базових речей, таких як користувач. Для того, щоб опрацьовувати різних користувачів, наприклад, студент, або викладач, потрібно створити базову таблицю, де буде зберігатися ідентифікатор користувача, пароль, та інші дані, тому перейдемо до створення такої таблиці (рисунок 1.9).

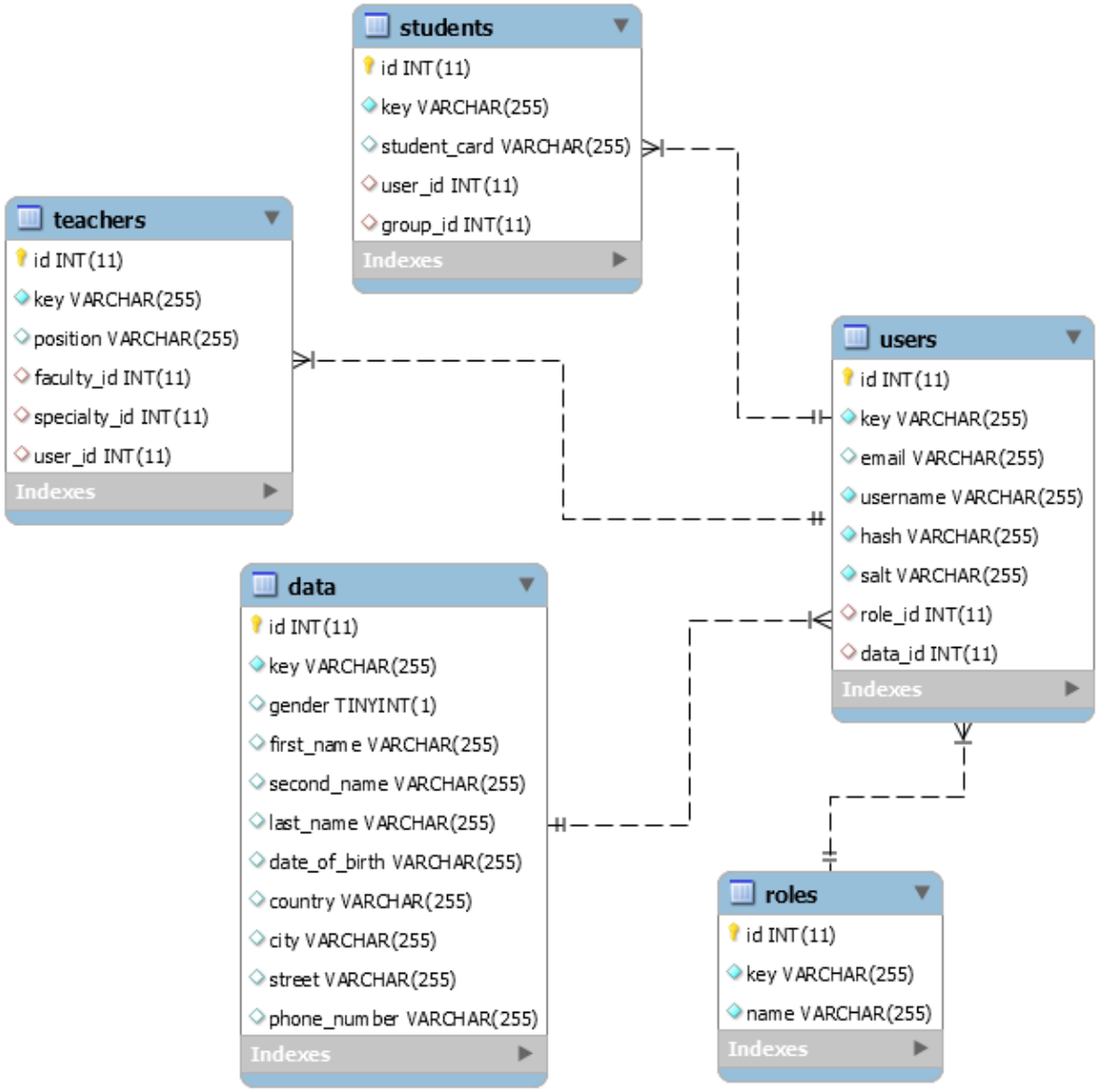


Рисунок 1.9 – Таблиці з БД, та взаємозв’язки

На рисунку, який представлений вище, зображена основна таблиця користувачів, та таблиця для студентів, викладачів, які відноситься до таблиці користувач, також маємо одну таблицю для збереження більш детальної інформації про користувача, і таблиця з ролями, для визначення прав користувача. В таблицях викладач та студент є інші зв’язки між таблицями, але вся побудована БД буде представлена частково в другому розділі, та повністю в третьому розділі під час тестування додатку.

Висновки до першого розділу

1. Проведено аналіз технологій для розробки інформаційної платформи дозволив сформувати вимоги до побудови веб-додатку з метою запуску процесу діджиталізації для забезпечення більш якісного процесу надання адміністративних послуг закладом вищої освіти.

2. Було проведено аналіз методів розробки, методів захисту веб-додатку.

3. За допомогою проведеного аналізу архітектури такого роду систем, було визначено будувати веб-додаток у вигляді двох частин: клієнтської та серверної, що забезпечить якісну підтримку, та безпроблемне масштабування веб-додатку в майбутньому.

РОЗДІЛ 2. РОЗРОБКА ВЕБ-ДОДАТКУ

В даному розділі будуть продемонстровані методи, засоби, та їх використання під час розробки веб-додатку «Інформаційна платформа Житомирська політехніка».

2.1 Розробка серверної частини

Під час опрацювання даного розділу, буде продемонстровано вирішення критичних питань, які виникали під час розробки. Буде продемонстровано лістинг програмного коду до базовий методів, тощо. Розпочнемо з встановлення, та налаштування проекту.

2.1.1 Встановлення та налаштування серверу

Робота з сервером починається з підготовки фундаменту, з налаштування, встановлення потрібних пакетів, та технологій, приведемо приклад встановлення тих пакетів, які були проаналізовані в першому розділі, це: **sequelize**, **mysql**, **graphql** та основний пакет **express**. Таким чином, за допомогою команди **yarn install [назва пакету]**, менеджер пакетів встановлює потрібні пакети останньої версії.

```
"bcryptjs": "^2.4.3",
"express": "~4.16.1",
"express-fileupload": "^1.1.7-alpha.3",
"express-graphql": "^0.9.0",
"graphql": "^15.0.0",
"mysql2": "^2.1.0",
"sequelize": "^5.21.5",
"underscore": "^1.10.2",
"uniqid": "^5.2.0",
"yarn": "^1.22.4"
```

Повертаючись до питання масштабування, кожен розробник може встановлювати потрібний йому пакет, та використовувати його за допомогою **requirejs**, але за замовчуванням, **requirejs** не встановлений, тому потрібно завантажити цей пакет також: "requirejs": "^2.3.6" та "amdefine": "^1.0.1".

Для того щоб розпочати роботу з проектом, потрібно налаштувати **requirejs**, тобто при ініціалізації проекту, **requirejs** потребує вже готовий список файлів, до яких розробник буде мати можливість звертатися, для цього було передбачено створення файлу **requirejs-config.js**, який допоможе з процесом збирання файлів, це виглядає наступним чином: розробник створює власний компонент, наприклад **User.js**, після оголошення цього файлу, потрібно надати йому глобальне ім'я, для того щоб до нього можна було звертатися, отож за допомогою **requirejs-config.js**, цей процес набагато спрощується.

```
module.exports = {
    paths: {
        configurations: {
            ...
        },
        models: {
            'user': 'InformationPlatform/model/entity/User'
        }
    }
};
```

Після оголошення **User.js**, розробник має можливість звернутися до цього файлу за допомогою `requirejs('model/user');`, як бачимо, до нашого ім'я, яку було прописано в **requirejs-config.js** додалася приставка **model**, це особливість яка була закладена при розробці, за допомогою цього покращується підтримка та процес масштабування.

Отож, тепер потрібно звернутися до кожного **requirejs-config.js**, якій знаходиться в окремих модулях, для цього використовуємо стандартні методи node.js для читання файлів. Наведемо невеличкий лістинг програмного коду:

```
for (let name in pathToModels) {
    if (pathToModels.hasOwnProperty(name)) {
        let pathToFile = path.join(__dirname, `./app/code/${
pathToModels[name] }`);
        Object.assign(paths, { [`model/${ name }`]: pathToFile });
    }
}
```

В даному програмного коді наведено збір всіх файлів в єдиний об'єкт, після чого, цей об'єкт буде переданий до **requirejs-config.js**, та розробник зможе звертатися до свої файлів.

Після того як **requirejs** був налаштований та залишалося тільки створити нову структуру файлу, виникла проблема з читанням інших файлів, тому що по-перше, **requirejs** не дозволяє виконувати циклічне звертання, тобто розробник не може звертатися до файлу, якщо той файл звертається до файлу розробник, це є дуже важливим моментом, але це питання вирішується дуже просто, оскільки такого не повинно бути, а якщо є, це проблема в архітектуру проекту, по-друге, під час ініціалізації файлу, **requirejs** використовував для пошуку файлів звичайний **require**, що є не допустим, тому для вирішення цієї проблеми було використано наступну конструкцію на початку кожного компоненту:

```
...
if (typeof define !== 'function') {
    var define = require('amdefine')(module, require('requirejs'));
}
...
```

Данна конструкція по-перше, виконує роль ініціалізації **define**, за допомогою якого, виконується виклик інших файлів, та по-друге, в параметрах до `require('amdefine')` вказується який менеджер файлів використовувати, отож, це вирішує всі проблеми з процесом роботи з файлами, та покращує всі наявні процеси під час розробки веб-додатку.

За допомогою аналізів з першого розділу, можна зробити висновки про велику кількість таблиць, та зв’язків між ними, отож потрібно було знайти найкращий метод підтримки та роботи з БД, для цього було встановлено **sequelize**, як вже було описано в перших розділах **sequelize** надає можливість керувати БД, за допомогою **ORM**.

Sequelize, так як і **requirejs**, потребує список файлів які містять в собі оголошення таблиць, оскільки цей процес все було автоматизовано, залишилось тільки створити таблицю, та надати список файлів до **sequelize**. Також потрібно пам’ятати, що для того, щоб зв’язатися з БД, **sequelize** потрібні параметри, такі як: пароль, ім’я користувача, хост, тощо. Для початку буде використовуватися локальна інсталяція, а вже потім буде налаштовано використання серверної БД, на базі **Azure Server for MySQL**.

Отож, на прикладі таблиці **User**, створимо об’єктне представлення даної таблиці за допомогою **sequelize**.

```
User.init(
    {
        id: {
            type: Sequelize.INTEGER,
            autoIncrement: true,
            allowNull: false,
            primaryKey: true,
        },
        key: {
            type: Sequelize.STRING,
            allowNull: false,
        },
        email: {
            type: Sequelize.STRING,
            allowNull: true,
        },
        username: {
            type: Sequelize.STRING,
            allowNull: false,
        },
        hash: {
            type: Sequelize.STRING,
            allowNull: false,
        },
        salt: {
            type: Sequelize.STRING,
            allowNull: false,
        },
    },
    {
        sequelize,
        modelName: "User",
        tableName: "Users",
        timestamps: false,
        underscored: true,
    }
);

User.belongsTo(Role, {foreignKey: "role_id"});

User.belongsTo(Data, {foreignKey: "data_id"});
```

Таким чином, керування, та створення таблиць стає набагато зручніше, за допомогою методу **init** оголошується поля, їх типи, та вказується ті параметри які надає **sequelize**. Метод `belongsTo` – це асоціація один до одного, в параметрах до цього методу можна вказувати назву `foreignKey` та деякі інші параметри.

Перейдемо до налаштування зв’язку з БД, для цього створимо спеціальний утиліту, (рисунок 2.1).

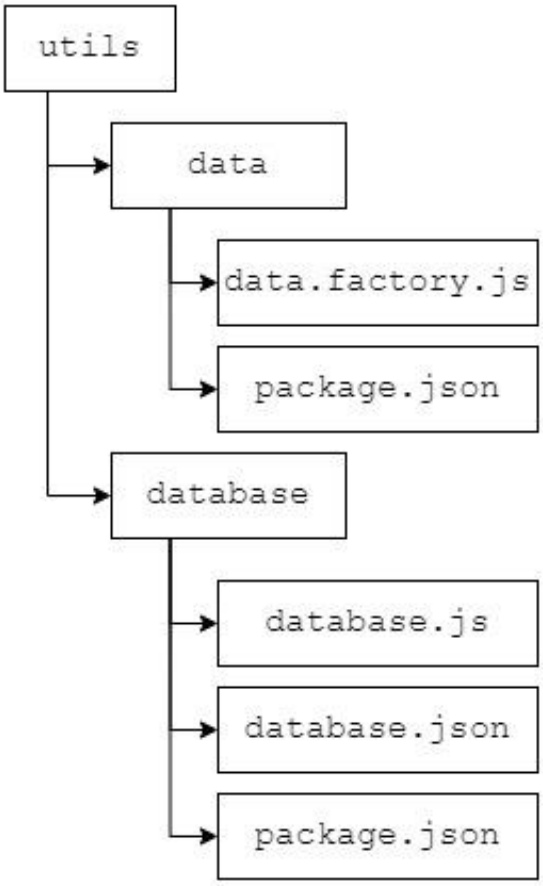


Рисунок 2.1 – Створені серверні утиліти

За допомогою **database** утиліти створюється можливість відділити логіку підключення до БД, від логіки ініціалізації, як вже було сказано, це питання архітектури та майбутнього масштабування. Таким чином, **database.js** збирає інформацію з **database.json** та створює підключення до БД. Структура **database.json** виглядає наступним чином:

```
{
  "host": "localhost",
  "database": "information-platform",
  "username": "root",
  "password": "abcABC123",
  "dialect": "mysql"
}
```

Як висновок з пророблені роботи, ми маємо готовий менеджер файлів, менеджер для ініціалізації таблиць в БД, та потрібно пам’ятати про неменше

важливий момент, такий як початкові дані, на рисунку 2.1 продемонстровано дві утиліти, перша для створення зв'язку з БД, та друга, яка оброблює пакети даних.

Утиліта, яка оброблює пакети даних, цілком зв'язана з **data.packet.js**, з пакетом даних, який оголошується в кожному модулі, отож при ініціалізації проекту, після ініціалізації файлів, створення зв'язку з БД, в дію вступає ініціалізатор початкових даних, він виглядає наступним чином:

```
define([
    'underscore'
], (
    _
) => {

    let dataPacketStorage = {};


    /**
     * @class ResolverFactory
     * @description The factory is initialize graphql resolvers in
project.
     */
    class DataFactory {

        static initialize () {
            for (const key in dataPacketStorage) {
                dataPacketStorage[key]();
            }
        }

        static defineData (initializationPacket) {

            if (_.isObject(initializationPacket) &&
_.isEmpty(initializationPacket)) {
                return;
            }

            try {
                for (const key in initializationPacket) {
                    dataPacketStorage[key] =
initializationPacket[key];
                }
            } catch (e) {
                throw new Error(`Something went wrong! Error: ${
JSON.stringify(e) }`);
            }
        }
    }

    return DataFactory;
});
```

Утиліта **data.factory.js** має метод `defineData` за допомогою якого і відбувається оголошення даних, після чого метод `initialize` маж можливість

створити ці дані, а за алгоритм створення, відповідає метод в пакеті даних, для прикладу наведемо створення ролів користувача:

```
return DataPacket.defineData({
    initializeRoles() {
        Role.new(RoleType.USER);
        Role.new(RoleType.MODER);
        Role.new(RoleType.ADMIN);
    },
});
```

Отож, переходимо до створення модулів, які будуть відповідати за першочерговий функціонал інформаційної платформи.

2.1.2 Реалізація основних модулів

Першим, та основним модулем для інформаційної платформи буде, створено модуль з ім'я **InformationPlatform**, цей модуль буде вмістити в собі, наступну структуру, що представлена на рисунку 2.2.

Модуль **InformationPlatform** відповідає за автентифікацію, авторизацію, надсилання email листів, створення та реєстрацію користувачів, збереження конфіденційних файлів, роботу з паролем та сесіями користувача, тощо.

Почнемо з автентифікації та авторизації. Модель **Auth.js** має три метода, перший метод `authenticate` відповідає за автентифікацію користувача на сайті, перед тим як надати доступ, перевіряє чи існує користувач, якщо так, надає право на автентифікацію, в результаті метод повертає `{key: user.key, sessionToken: session.sessionToken}`. В скороченому варіанті, метод виглядає наступним чином:

```
...
const isExist = await User.isExist({ username: login, email: login });
...
const session = await Session.create({
    sessionToken: Session.generateToken(), user_id: user.id
});
...
```

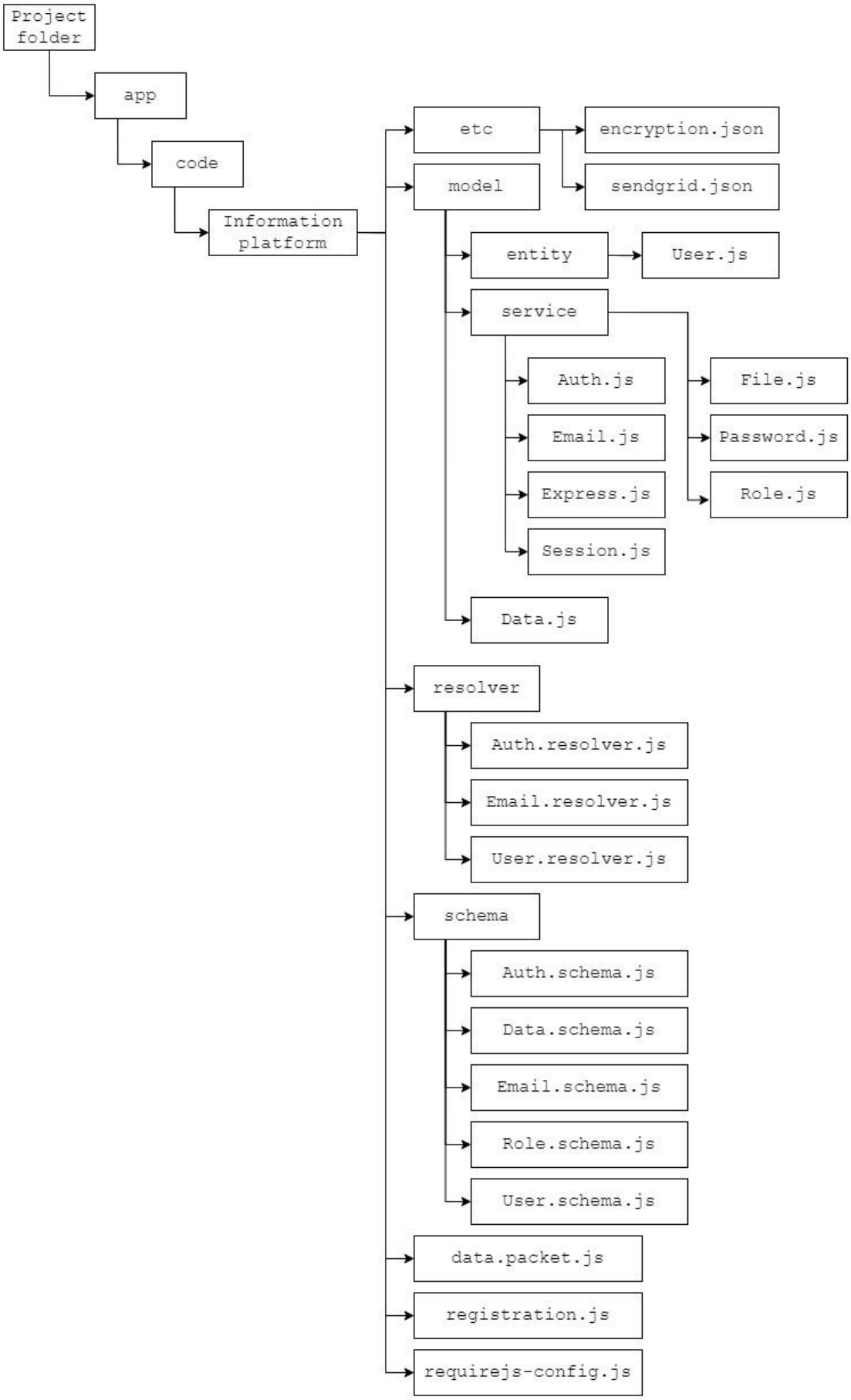


Рисунок 2.2 – Структура модулю, **InformationPlatform**

Отож, реалізацію першого метода, в моделі **Auth.js**, було розглянуто, інші методи такі як, `isAuthenticated` та `isAuthorized` відповідають за перевірку. Переходимо до реалізації методу, для створення облікового запису, тобто реєстрації, в даній реалізації нас цікавлять моменти шифрування паролю, та збереження даних. За даний функціонал буде відповідати модель **User.js**, але перш ніж почати реалізовувати метод для створення облікового запису, потрібно розібратися яким чином відбувається реалізація резолверів (методів для вирішення запитів).

На рисунку 2.2 відображена структура модуля **InformationPlatform**, якщо звернути увагу на структуру, можна побачити резолвери та відповідні схеми. Вся концепція максимально проста, за допомогою схем, здійснюється опис даних, та оголошення методів, для вирішення певних запитів, саме за допомогою схем, розробник має можливість здійснювати зв'язок між сервером та клієнтом, а за допомогою резолверів, розробник отримує можливість опрацьовувати дані. Таким чином для прикладу, проведемо реалізацію методів для вирішення запиту, та схеми, для попередньої моделі **Auth.js**.

Вище було описано реалізацію метода `authenticate`, але це тільки реалізація, для того щоб клієнт мав можливість запросити права на автентифікацію, потрібно оголосити резолвер наступним чином:

```
async authenticate ({ authenticationInput }, request) {
    const { login, password } = authenticationInput;
    if (!login || !password) {
        throw new Error('The parameters are incorrect!');
    }
    if (!validate.isLength(password, { min: 6, max: 24 })) {
        throw new Error('The password must be more 6 and less 24
symbols!');
    }
    return Auth.authenticate(authenticationInput);
}
```

Даний резолвер, приймає на вхід дані з клієнта перевіряє їх на правдивість, та надсилає вище реалізованому методу в моделі **Auth.js**, але все це не буде працювати без схеми, тому наступним кроком має бути оголошення схеми.

```
return Schema.defineSchema({
    types: `
        input AuthenticationInput {
            login: String!
            password: String!
        }
        type AuthenticationOutput {
            key: String!
            sessionToken: String!
        }
    `,
    queries: `
        authenticate(authenticationInput: AuthenticationInput):
AuthenticationOutput!
        isAuthenticated(key: String!, sessionToken: String!):
Boolean!
        isAuthorized(key: String!, role: UserRole!, sessionToken:
String!): Boolean!
    `,
    mutations: ``
}, { name: 'Authentication' });
```

Як було вже вище описано, схема відповідає за оголошення типів даних, та за оголошення резолверів. Резолвери поділяються на два типи, ті які повертають дані, та ті які, модифікують їх, тобто `queries` та `mutations`. Таким чином було оголошено `authenticate query` за допомогою якого, клієнт має можливість запросити автентифікацію користувача.

Отже, повертаючись до реалізації реєстрації користувача, розпочнемо з оголошення схеми та резолвера (в скороченому виді). Схема **User.schema.js**, яка оголошує `mutation::newUser(userInput: UserInput): UserOutput!` та типи `type::UserOutput` та `type::UserInput`, і **User.resolver.js**, який визначає

`mutation::newUser`. На перших кроках, при отриманні даних, резолвер перевіряє їх на правильність, також потрібно згадати, що на стороні клієнта також є перевірка даних, але для забезпечення процесу більш надійним, перевірки дублюються на сервері також. Резолвер перевіряє коректність **email**, **username**, **password**. Далі дії відбуваються в залежності від типу користувача, так як інформаційна платформа має три типи, і один з типів має права модератора, резолвер це визначає та надає потрібні права.

Після того як всі перевірки пройдені, тип прав визначений, відбувається створення користувача, під час цього відбувається перевірка на існування схожого **email** або **username**, якщо ці два типи даних нові, відбувається шифрування паролю.

В перших розділах було проаналізовано методи шифрування паролю, та як результат, було обрано `PBKDF2`. Даний метод було вибрано тому що використання PBKDF2 рекомендовано для нових програм і продуктів. В якості псевдовипадкової може бути обрана криптографічна хеш-функція, шифр, HMAC. Таким чином, при створенні користувача, відбувається звертання до методу `encrypt`, даний метод встановлює **salt** та **hash** для пароля користувача `encrypt` метод спочатку створює унікальну для кожного користувача **salt**, потім хешує **salt** із паролем користувача та створює **hash**, цей **hash** зберігається в базі даних як пароль користувача. Даний метод в коді:

```
...
const salt = crypto.randomBytes(
    randomBytes.size = 16
).toString(encoding = hex);
const hash = crypto.pbkdf2Sync(
    password, salt, iterations = 1000, keylen = 64, digest = sha512
).toString(encoding = hex);
return {
    salt: salt,
    hash: hash
};
...
```

Перевірка вірності паролю відбувається за допомогою метода `validate`, метод перевіряє правильність пароля користувача. Він бере пароль користувача із запиту та **salt** із запису бази даних користувача. Потім він хешує пароль користувача та **salt**, а потім перевіряє, чи генерований хеш дорівнює хешу користувача у базі даних чи ні. Якщо хеш користувача дорівнює генерованому хешу, то пароль правильний. Перевірка вірності паролю в коді, виглядає наступним чином:

```
...
if (!password || !passwordSalt || !passwordHash) {
    return false;
}
const hash = crypto.pbkdf2Sync(
    password, passwordSalt, iterations, keylen, digest
);
return passwordHash === hash.toString(encoding);
...
```

Після отримання зашифрованого паролю, виконується операцію створення нового записку в таблиці користувач, та надсилання листа на електронну пошту для підтвердження **email**, після чого клієнт отримає першу інформацію про користувача. Як результат маємо наступну схему, (рисунок 2.3).

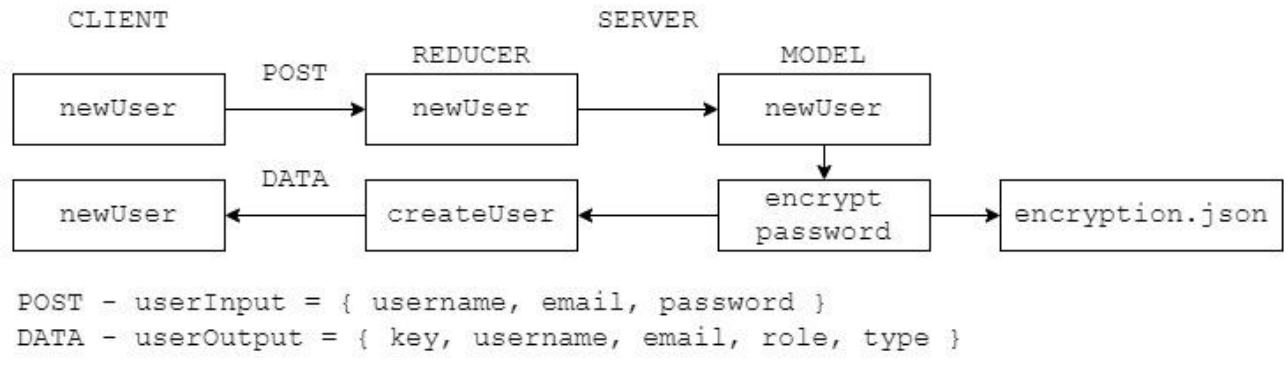


Рисунок 2.3 – Відображення процесу створення користувача в стислому виді

За роботу з електронною поштою відповідає окремий сервіс, під назвою SendGrid [25]. SendGrid надає хмарну послугу, яка допомагає підприємствам з доставкою електронної пошти. Служба управляє різними типами електронної

пошти, включаючи сповіщення про доставку, запити друзів, підтвердження реєстрації та електронні розсилки. Він також обробляє моніторинг постачальника послуг Інтернету (ISP), ключі домену, рамки політики відправника (SPF) та петлі зворотного зв'язку. Крім того, **SendGrid** забезпечує відстеження посилань, звітність за відкритою ставкою. Він також дозволяє компаніям відстежувати відкриття електронної пошти, скасовувати підписки, відмов і звіти про спам. Під час розробки веб-додатку, **SendGrid** демонструє всю активність, яка відбувалася з електронними листами, (рисунок 2.4).

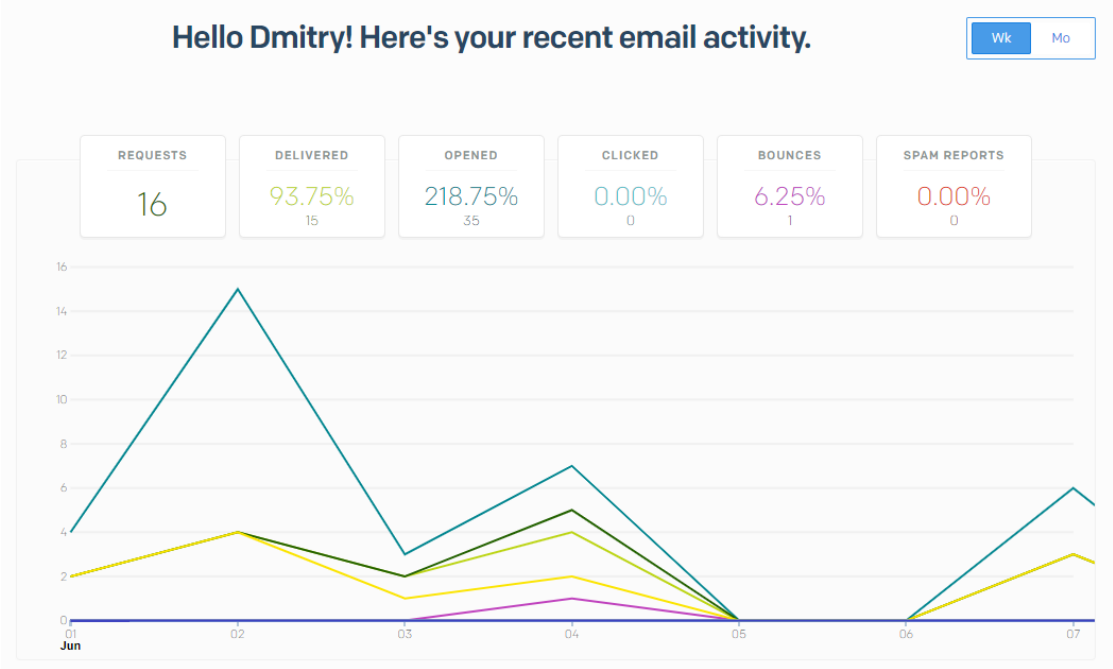


Рисунок 2.4 – Відображення активності з електронними листами

SendGrid надає велику кількість можливостей, а враховуючи те, що **SendGrid** відслідковує відкриття електронних листів, цю функцію можна використовувати в майбутньому для роботи з студентами, викладач буде знати, який студент відкрив електронний лист, а який ні.

В загальному модуль **InformationPlatform** відповідає за основні процеси в веб-додатку, але вище згадувалися такі моменти, як створення облікового записку для студента, або викладача, насправді це зовсім інший модуль, оскільки цей функціонал відноситься до університету, тому переходимо до створення модулю, **UniversityPlatform** який буде доповнювати модуль **InformationPlatform**.

Для початку створимо структуру модулю **UniversityPlatform** (рисунок 2.5)**,** а далі відбудеться реалізація основних методів.

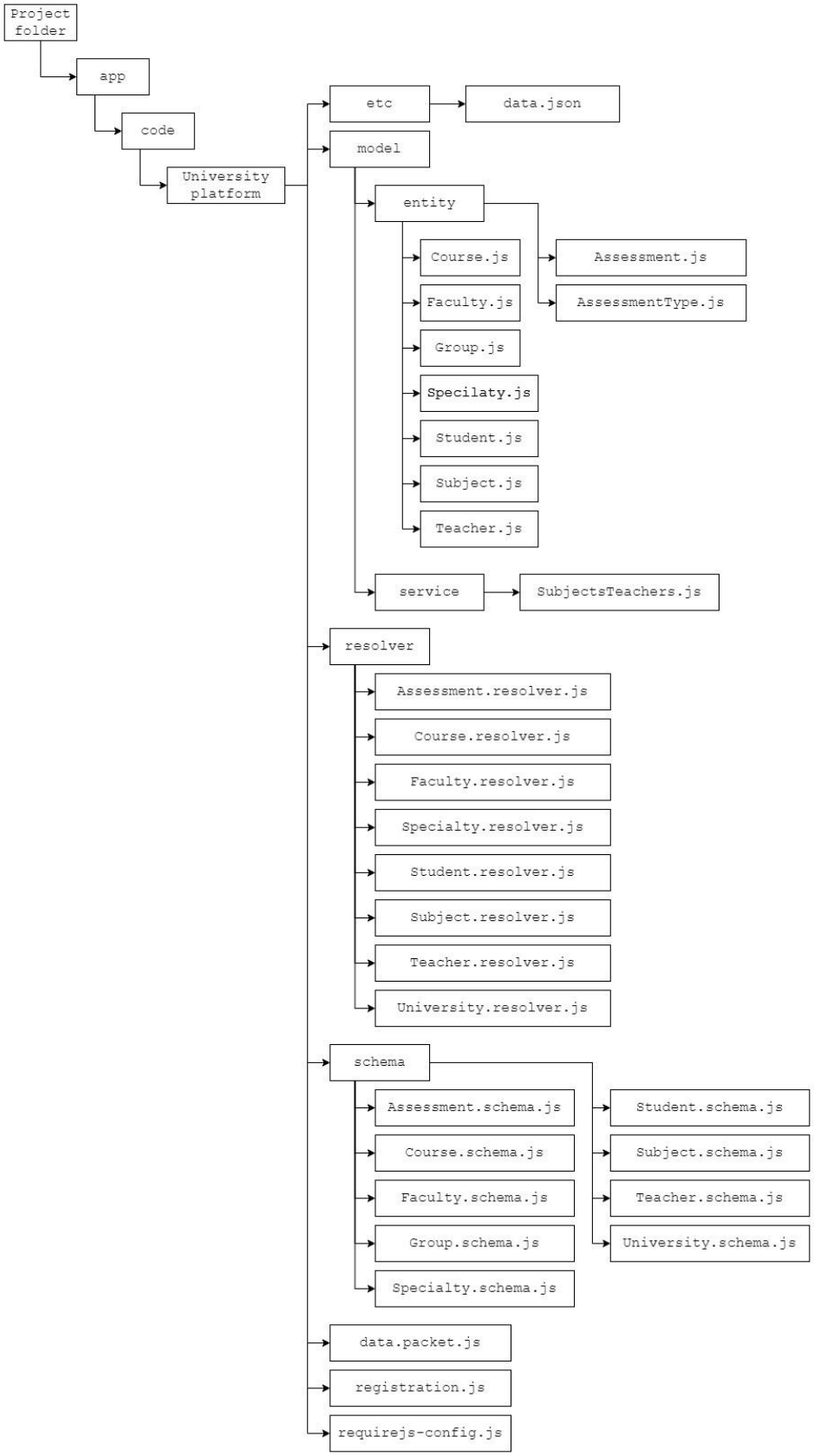


Рисунок 2.5 – Структура модулю, **UniversityPlatform**

Звернемо увагу на файл **data.json** та **data.packet.js**. Оскільки Інформаційна платформа створюється для покращення наявних процесів документообігу, для того щоб все коректно працювало, потрібно надати викладачам моментальний доступ, тобто при ініціалізації серверу, створити всіх наявних викладачів. В файлі **data.json** зібрана інформація про факультети, спеціальності, та ПІБ всіх викладачів. За допомогою **data.packet.js**, кожен викладач має змогу авторизуватися в веб-додатку за наданими логіном та паролем, надалі пароль можливо буде змінити.

При оформленні будь якого документу, чи як в нашому випадку, оформленні індивідуального графіку, потрібно вказати викладача, факультет, спеціальність, тощо. За допомого ініціалізації першочергових даних, ми забезпечуємо більш надійний процес, оскільки студент не в змозі ввести власного викладача, тощо. Під час розробки клієнтської частини, буде продемонстровано вигляд цього функціонала з точки зору користувача.

Однією з багатьох функцій яких виконує модуль **UniversityPlatform** є, обробка та збереження інформації яка відноситься до університету, тобто факультет, спеціальність, тощо. За даний функціонал відповідають два метода, які знаходиться в **University.resolver.js**, це `setUniversityInformation` та `getUniversityInformation`, дані методи відносяться до другої групи запитів (рисунок 1.4), кожен метод перед виконанням певних дій, перевіряє автентифікацію користувача.

Перевірка автентифікації користувача відбувається за допомогою методів, які були вище описані, це `isAuthenticated` та `isAuthorized`. Кожен метод другої групи має обов’язкові дані на вході, це ідентифікатор користувача, та його токен сесії, якщо ці два параметра вірні, та існують в БД, користувач отримає дані, якщо ні, користувач побачить відповідне повідомлення про помилку, в нашому випадку «`You do not have all rights to this request.`». Відповідно до назви модуль, модуль виконає надані йому дії, тому, переходимо до наступного модуля, **IndividualScheduling**.

За для того щоб, реалізувати можливість оформлення індивідуального графіку студента, потрібно було створити один модуль, який відповідає за базовий функціонал, та другий модуль, який відповідає за специфічні до проекту дані, а тепер, використовуючи два вище новостворених модуля можна розробити, модуль, який відповідає за оформлення індивідуального графіку.

2.1.2.1 Реалізація модуля для оформлення індивідуального графіку

Модуль для оформлення індивідуального графіку, буде складатися с двох головних таблиць **Company** та **Schedule**, та з двох сервісних таблиць **ScheduleApproves** та **ScheduleSubjects**. Таблиця або модель **Company** відповідає за збереження даних про працевлаштування користувача. Таблиця **Schedule** відповідає за збереження всіх запитів на оформлення індивідуального графіку. Сервісні таблиці, відповідають за збереження проміжних даних, які використовується для забезпечення коректної роботи оформлення, та використання індивідуального графіку.

Отже, інформаційна платформа буде мати секцію «Послуги», наразі веб-додаток має послугу оформлення індивідуального графіку, і доступ до цієї послуги має тільки студент. Для того щоб оформити даний документ, студенту потрібно вказати інформація про себе, тобто ПІБ, поштова скринька, номер мобільного телефона, та інформація про факультет, спеціальність, групу, після чого потрібно заповнити форму, в якій вказати успішність за минулий семестр, це забезпечить більш комфортний процес розглядання як для викладача так і для співробітника деканату, отже, також студенту потрібно обрати предмети, по яким студент хоче отримати можливість індивідуального навчання, після чого обираються дати, в діапазоні яких буде дійсний оформлений документ.

За більшість функцій, які були вище описані відповідає клієнтська частина. Сервер відповідає тільки за обробку, та збереження даних. Таким чином, коли весь процес зрозумілий, переходимо до побудування структури модулю (рисунок 2.6), та реалізації базових функцій, які знадобляться клієнту.

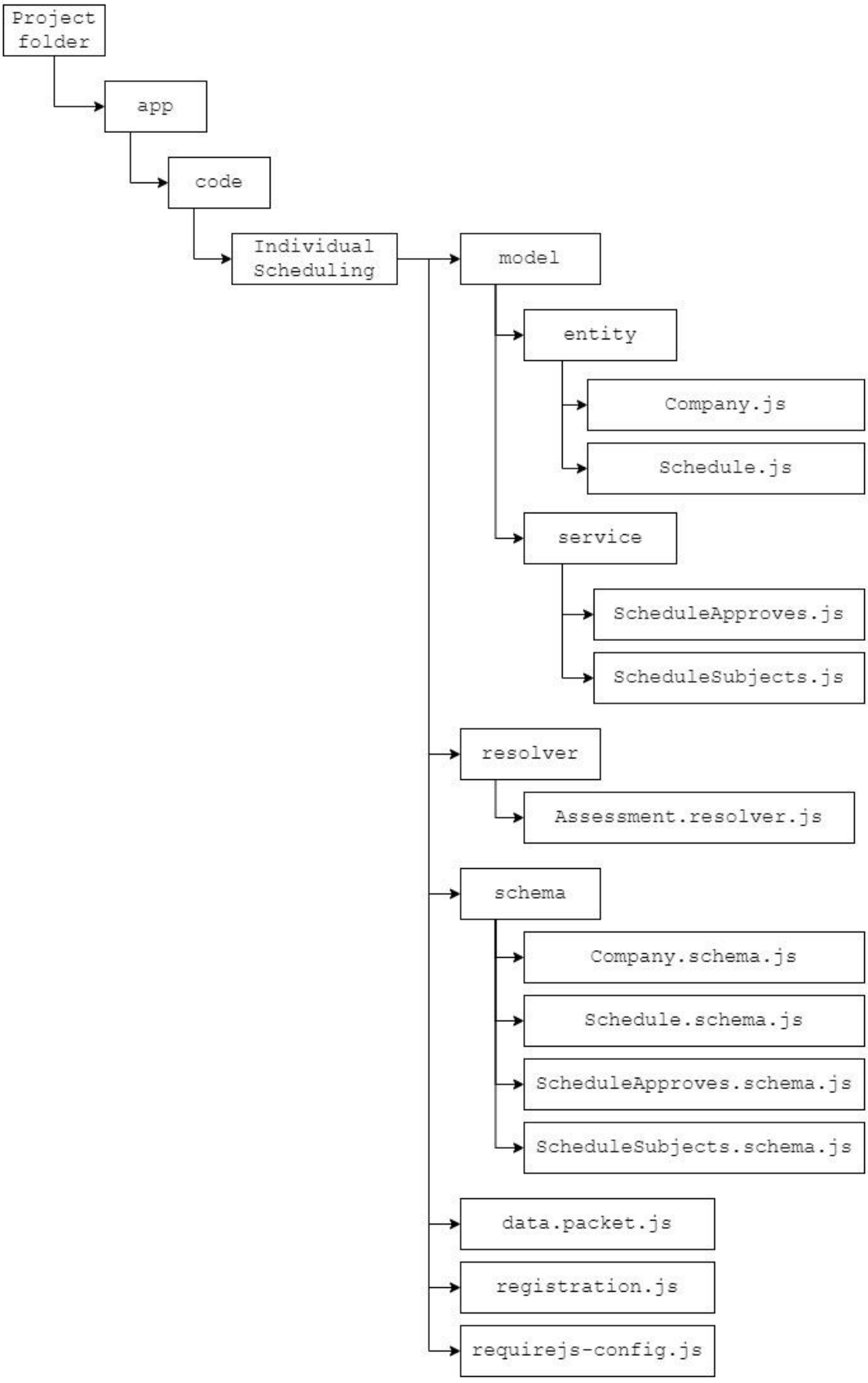


Рисунок 2.6 – Структура модулю, **IndividualScheduling**

Основні методи, які обробляють запит на створення нового запиту, або на отримання даних, наразі виконуються в **schedule.resolver.js.** (рисунок 2.7)

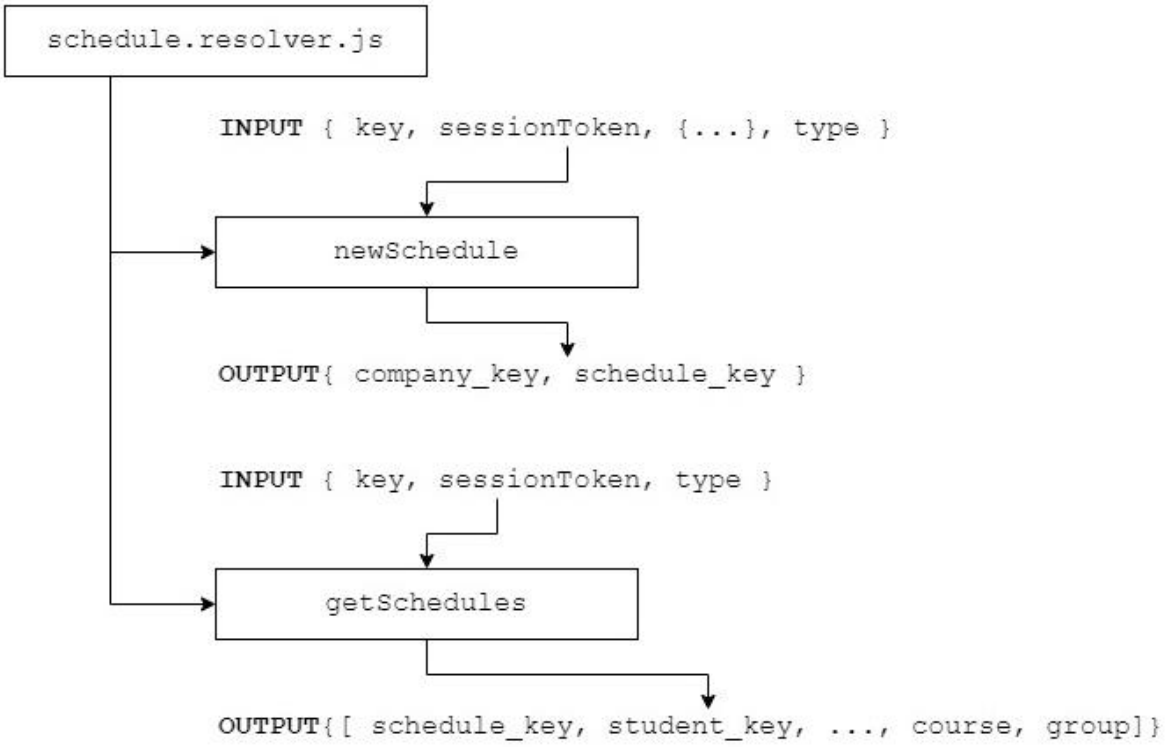


Рисунок 2.7 – Відображення методів **schedule.resolver.js**

Розберемо поетапно реалізацію метода `newSchedule`. Першим кроком виконується перевірка користувача, так як метод відноситься до другого типу запитів (рисунок 1.4), далі виконується перевірка обов'язкових даних:

```
const {
    company_name, company_position, date_from, date_to, subjects
} = data;
if (_.isEmpty(company_name) || _.isEmpty(company_position)) {
    throw new Error('The company name or position value is empty.');
}
if (_.isEmpty(subjects)) {
    throw new Error('The array of subjects are empty.');
}
```

Після вдалої перевірки потрібно отримати об'єкт студента за допомогою унікального ідентифікатора який передається в параметрах до методу (рисунок

2.7). В процесі отримання студента, користуємося можливістю та перевіряємо, чи існує студент.

```
const studentInstance = await Student.get({ key });
if (!studentInstance) {
     throw new Error(`Student is not found!`);
}
const groupInstance = await Group.findByPk(studentInstance.group_id);
if (!groupInstance) {
     throw new Error('A user doesn\'t have a group');
}
```

Для створення індивідуального графіку, нам потрібні дані про факультет, спеціальність, курс, та компанію, якщо вказана компанія не існує, відбувається створення компанії.

```
const courseInstance = await Course.findByPk(
     groupInstance.course_id
);
const specialtyInstance = await Specialty.findByPk(
     groupInstance.specialty_id
);
const facultyInstance = await Faculty.findByPk(
     specialtyInstance.faculty_id
);
const [ companyInstance, isCompanyCreated ] = await
Company.findOrCreate({
    where: {
        [Op.and]: [
            { name: company_name },
            { student_id: studentInstance.id }
        ]
    },
    defaults: {
        name: company_name,
        position: company_position,
        student_id: studentInstance.id
    }
});
```

Таким чином, виконується вся обробка інформації, яка на кінцевому етапі створює новий запис в таблиці **Schedule**.

```
const scheduleInstance = await Schedule.create({
    date: date,
    date_from: dateFrom,
    date_to: dateTo,
    student_id: studentInstance.id,
    company_id: companyInstance.id,
    faculty: facultyInstance.name,
    faculty_abbreviation: facultyInstance.abbreviation,
    specialty: specialtyInstance.name,
    course: `${ courseInstance.course } (${ courseInstance.degree })`,
    group: groupInstance.name
});
```

Звернемо увагу на те, що інформація про факультет, спеціальність, курс та групу, зберігається в двох місцях. Перше місце це таблиці **Faculty**, **Specialty**, **Course**, **Group**, а друге, це таблиця **Schedule**. Це було реалізовано по тій причині, що наразі користувач має право зміни цих даних в будь який момент, та для того щоб запобігти зміни документа після його оформлення, були створені додаткові поля. Після інтеграції з наявними сервісами які присутні в університеті, ці поля будуть статичними, і їх зміна буде не можлива.

Таким чином, весь основний функціонал включаючи схеми та резолвери був створений, переходимо до реалізації, та налаштуванню клієнт частини.

2.2 Розробка client частини

Розробка клієнту, розпочинається з встановлення пакетів. Встановлення пакетів відбувається таким же чином, як і на серверній частині, але, клієнт має зовсім інші пакети, тому для прикладу наведемо декілька з них:

```
"react": "^16.13.1",
"@blueprintjs/core": "^3.26.0",
"@blueprintjs/datetime": "^3.18.1",
"@blueprintjs/icons": "^3.15.1",
"@blueprintjs/select": "^3.13.1"
```

Насправді кількість пакетів налічує більше 30, і це все для того, щоб забезпечити якісне, та комфортне перебування, та користування у веб-додатку.

Схематичне відображення структури клієнт частини зображено на рисунках 1.5 та 1.6.

Надалі кожний під-розділ буде містити в собі розробку певної функціональності, під певною функціональністю мається на увазі окрема сторінка або компонент. Розпочнемо реалізацію інформаційної платформи з автентифікація та реєстрації користувачів.

2.2.1 Автентифікація та реєстрація користувачів

Автентифікація та реєстрація користувача складається з відповідних до них форм, розпочнемо з форми реєстрації. Форми реєстрації буде складатися з компонентів які відповідають до типу користувача, наприклад, для студента буде обов'язково вказати студентський квиток при реєстрації, в відмінності від звичайного користувача, та викладача. В майбутньому викладач буде мати обов'язкове поле «Довідка з роботи», або ж реєстрація викладача буде заборонена, і буде здійснюватися через адміністратора інформаційної платформи, що є цілком вірним, але наразі, форма для реєстрації звичайного користувача та викладача однакові.

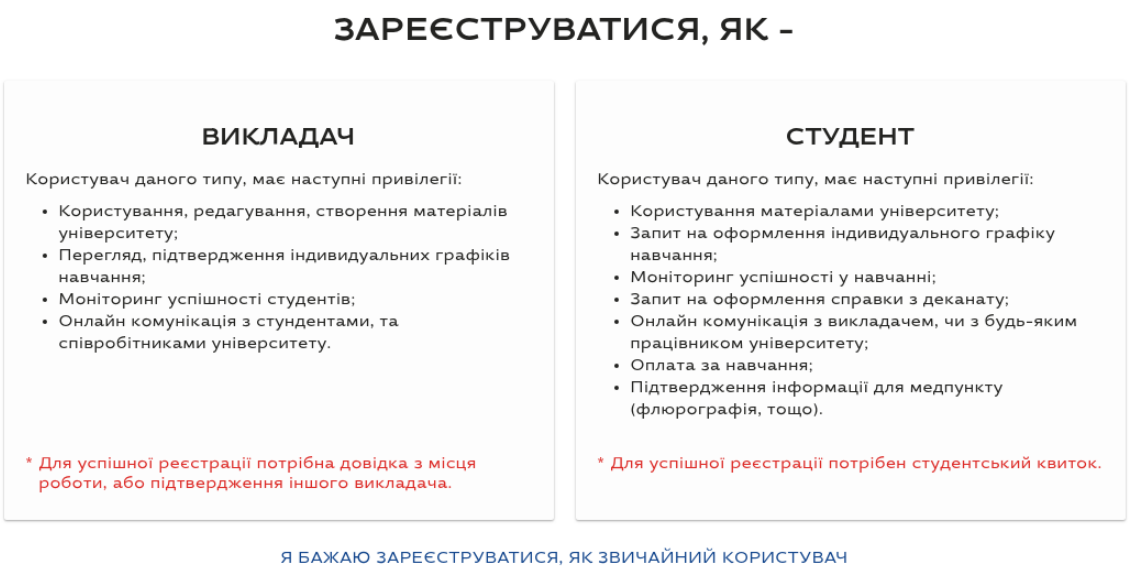


Рисунок 2.8 – Вибір типу користувача

В перших розділах було проаналізовано технології, які будуть використовуватися під час розробки інформаційної платформи, це важливо, оскільки в основі client-side знаходиться React, а це означає що веб-додаток складається з компонентів, та кожен компонент має власну імплементацію.

Отож розглянемо схему реалізації компоненту реєстрації, (рисунок 2.10), який складається перше, з вибору типу користувача (рисунок 2.8), друге з форми реєстрації (рисунок 2.9).

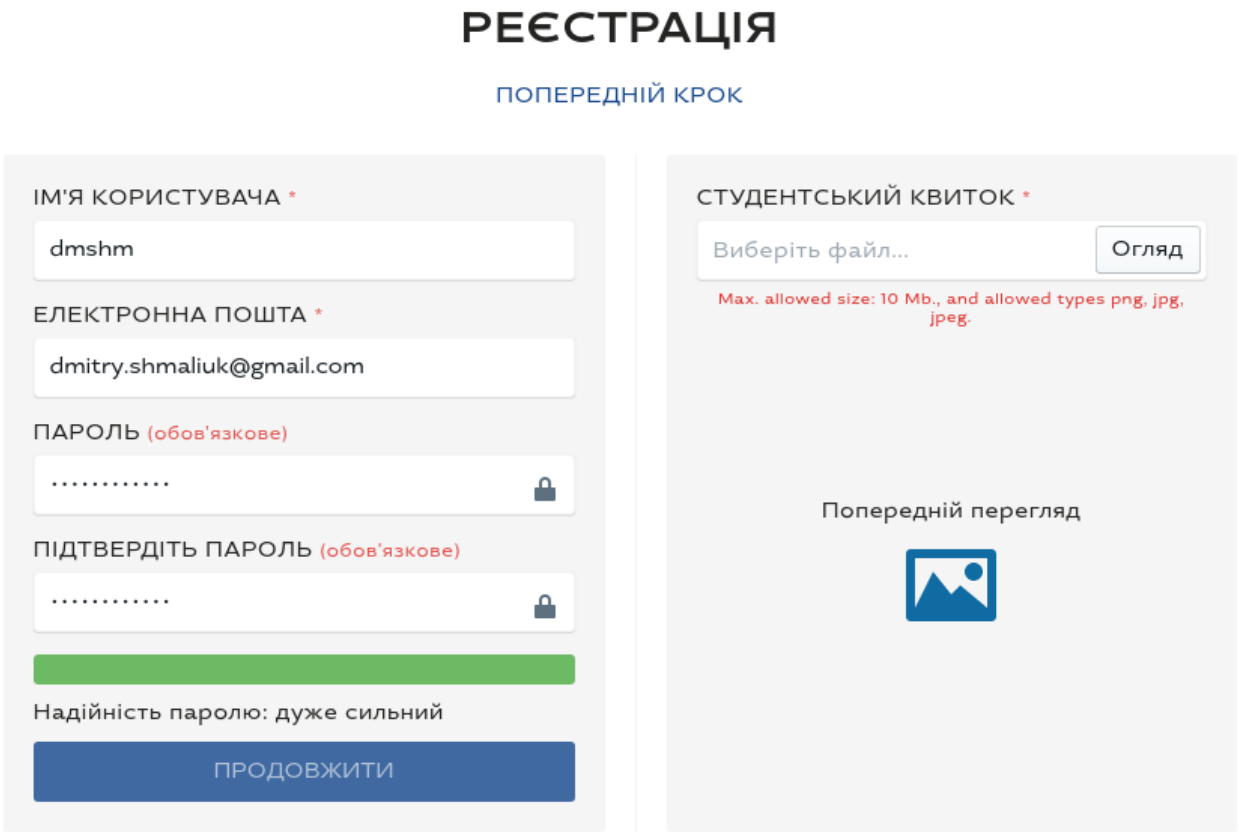


Рисунок 2.9 – Форма реєстрації студента

Компонент реєстрації, складається з методів для відображення HTML, при виборі типу користувача, з метода зв'язку з сервером та з методів які відповідають за кроки, тобто на якому кроці знаходиться користувач. Це надає певну низку можливостей для розробника, так як розробник може вбудовувати власну функціональність без втручання в інші методи. При розробці даного компоненту враховувалися правила, які дозволяють підтримувати проект якісним, та з

можливостю підтримки в майбутньому. Дані правила були проаналізовані в першому розділі.

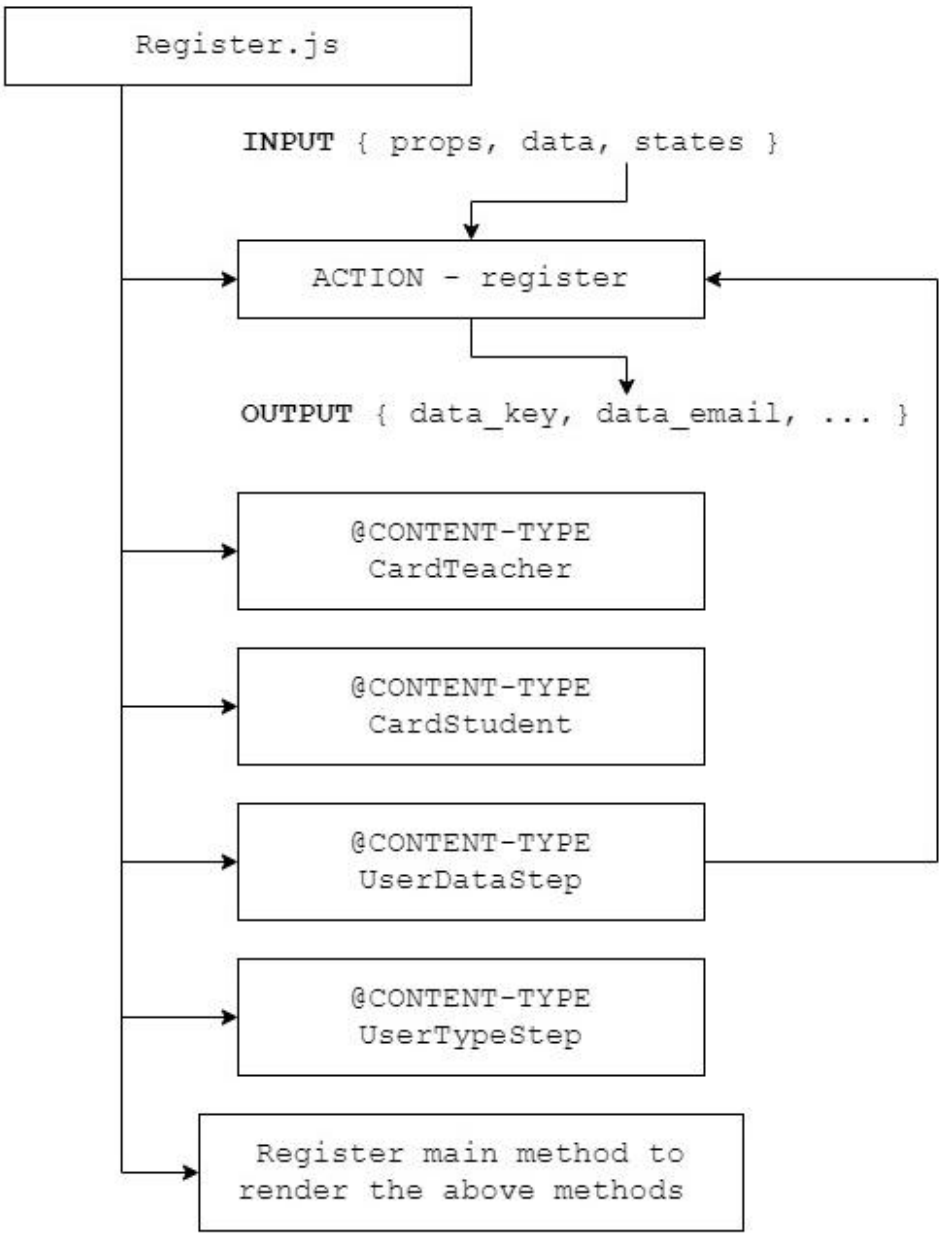


Рисунок 2.10 – Схема зв’язків між методами в компоненті **Register.js**

На рисунку 2.10 зображені методи, та виконає питання, як веб-додаток дозволяє поділити функціональність на методи, та залишитися все таким же гнучким. В першому розділі було проаналізовано зберігання змінних стану в глобальну сховищі, але для таких випадків коли дані на мають великого впливу, та не будуть використовуватися в іншому місці, використовується стандартний підхід до використання змінних стану, наприклад змінна стану **username** користувача

виглядає наступним чином `const [ username, setUsername ] = useState('');`, оголошується змінна для читання значення, та функція для оновлення даних, саме це робить React унікальним інструментом для розробника, оскільки при зміні стану в будь якому місці, всі інші компоненти які використовують те, чи інше значення, будуть знать про цю зміну. Отже, після побудови структури методів, та встановлення зв'язку між ними, побудуємо форму, за допомогою якої буде відбуватися збір даних в **UserDataStep** та передавати дані до **register action** (рисунок 2.10). Побудова форм в інформаційній платформі відбувається завдяки вже створеним компонентам форми, таким як **Input**, **Password**, **PasswordStrength**, **Button**, все це унікальні компоненти які відповідають за певний функціонал, за допомогою їх й побудуємо форму для реєстрації:

```
<Input
  type="text"
  inputFor={InputFor.USERNAME}
  label="Username"
  labelInfo="required"
  placeholder="Username"
  isValid={isFormValidSet}
  onChangeHandler={(event) => setUsername(event.target.value)}
/>;
<Password
  type="password"
  label="Password"
  isValid={isFormValidSet}
  labelInfo="required"
  placeholder="Password"
  id="password"
  onChangeHandler={(event) => setPassword(event.target.value)}
/>;
<PasswordStrength password={ password } />
<Button
  intent={Intent.PRIMARY}
  type={ButtonType.SUBMIT}
  disabled={!isDisable}
  buttonName="Continue"
  onClickHandler={submitHandler}
/>;
```

Після того як користувач зареєструвався інформаційна платформа автоматично відправить лист на вказаному електронну адресу (рисунок 2.11), з повідомленням про успішну реєстрацію та кодом для підтвердження електронної скриньки. Підтвердження електронної скриньки є важливим моментом при розробці інформаційної платформи, оскільки вся комунікація між користувачами на теперішньому етапі розробки, буде відбуватися через email.

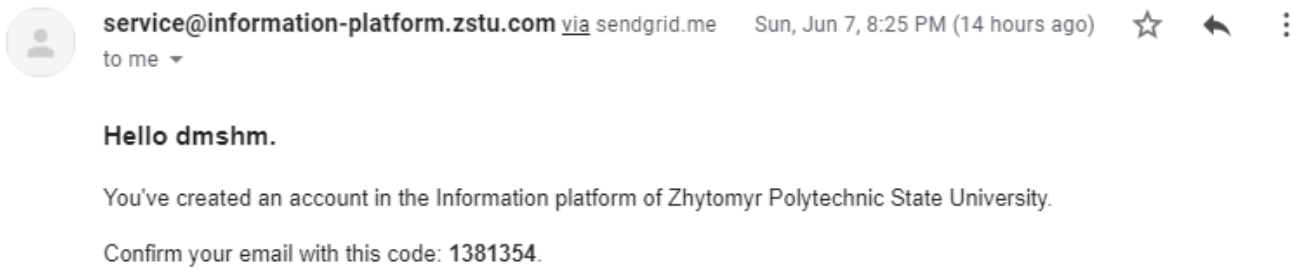


Рисунок 2.11 – Лист про успішну реєстрацію, з кодом на підтвердження email

За допомогою компоненту **ConfirmEmail**, який відповідає за функціонал підтвердження email, відбувається процес. Після того як лист був надісланий на пошту, на стороні клієнта відбувається відкриття модального вікна, яке дає користувачу можливість підтвердити зараз, або потім, (рисунок 2.12).

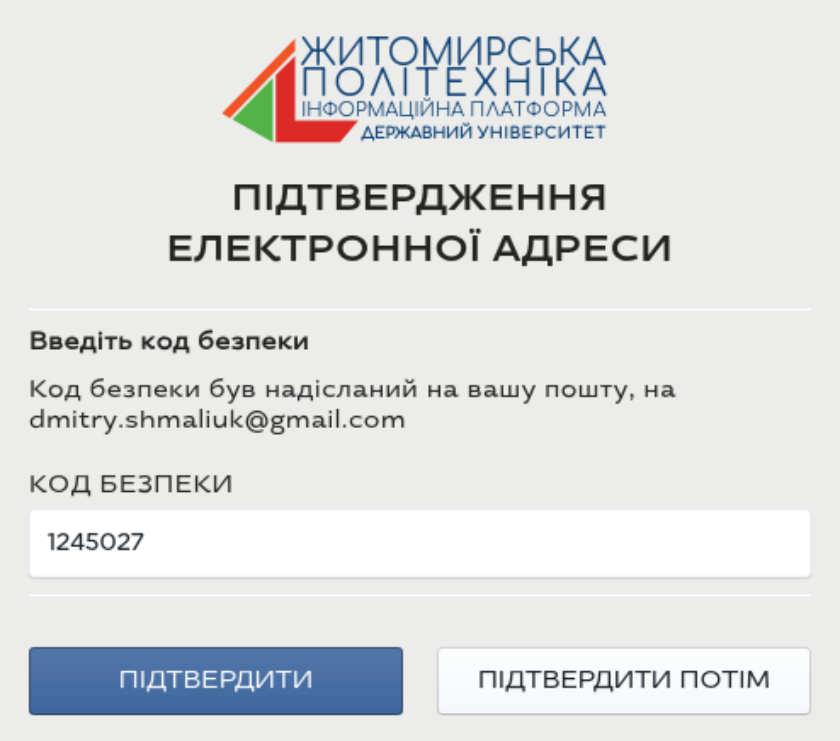


Рисунок 2.12 – Модальне вікно підтвердження електронної адреси

Після натискання кнопки, підтвердити, з’являється повідомлення про успішно виконано операцію, після чого реалізуємо перенаправлення до сторінки автентифікації, або більш розпорошену назву серед користувач, авторизацію, (рисунок 2.13).

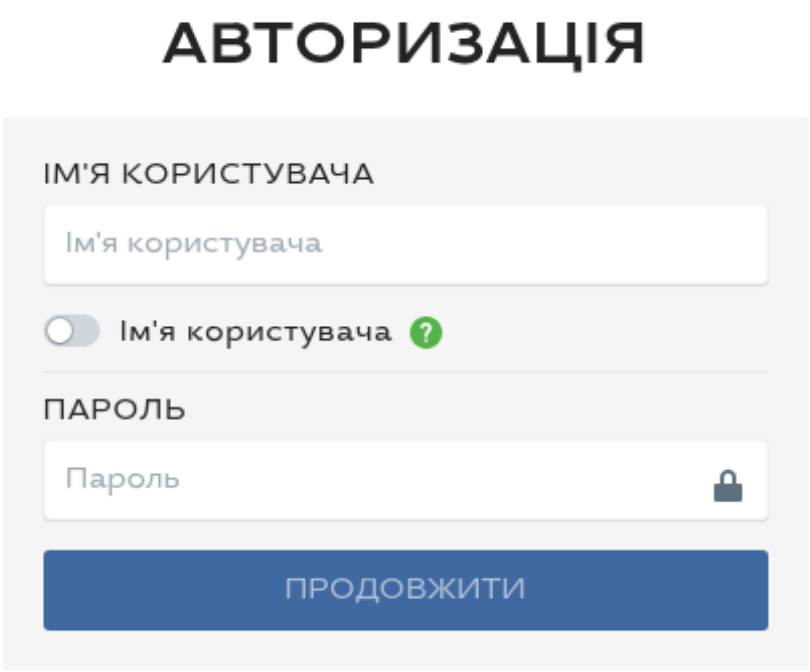


Рисунок 2.13 – Форма авторизації користувача

Перенаправлення до сторінки авторизації, замість моментальної авторизації після реєстрації, було реалізовано для того щоб, по-перше, дотримування правил SOLID, кожен метод повинен відповідати за надану йому функцію, по-друге компоненти React, повинні бути невеликими, гнучкими, та відповідати за надану їм функцію.

Форма авторизації складається з поля для заповнення логіну та паролю, та перемикача, в залежності від того, яким даними користувач хоче скористатися для того, щоб авторизуватися, в користувача є можливість обрати або за допомогою логіна, або за допомогою електронної адреси, це було розроблено в цілях покращення комфортності використання веб-додатку.

Якщо користувач заповнив всі дані вірно, веб-додаток авторизує даного користувач, якщо ні, або якщо користувач з такими даними не існує, веб-додаток відобразить відповідне повідомлення про помилку.

2.2.2 Персональний кабінет користувача

Користувач кожного типу має можливість відкрити власний кабінет, який містить певний функціонал, наприклад по редагування та збереженню певних даних. Розпочнемо з реалізації першої сторінки, яка буде відповідати за відображення основних даних, (рисунок 2.14) та за відображення останніх запитів на оформлення певних документів.

Загальну концепцію збереження та обробки інформації за допомогою змінних станів було оглянуто в під-розділі 2.2.1, тому надалі будуть наводиться основні аспекти, та методи вирішення проблем під час розробки.

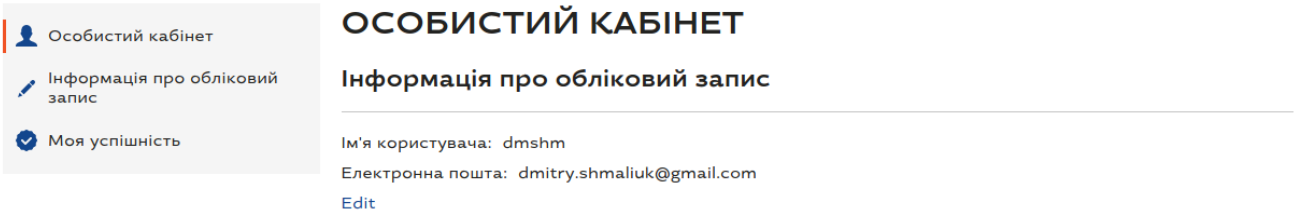


Рисунок 2.14 – Особистий кабінет користувача. Голова сторінка

Під час реалізації головної сторінки для особистого кабінету, було використано дані з глобального сховища, як було продемонстровано на рисунку 1.6 кожна змінна, має власний **action** та **reducer**, але ці типи методів використовуються для того, щоб контролювати данні, тобто оновлювати, задавати чи видаляти, цей аналіз був проведений в першому розділі. Для того щоб скористатися даними, використовуються певна конструкція, яка буде важлива надалі, в процесі розробки, отже конструкція виглядає наступним чином:

```
const mapStateToProps = (state) => {
  return {
    userInfo: state.user,
    isAuthenticated: state.auth.isAuthenticated,
  };
};
```

Якщо розібрати вище приведений код, то за допомогою `state.user` відбувається звертання до сховища, з ключем **user**, після чого всі дані з сховища копіюються в змінну **userInfo**, яку далі, можемо використати.

На сторінці «Інформація про обліковий запис» надається можливість переглянути, та змінити інформацію про обліковий запис. Важливим моментом є те, що на даній сторінці існує три блока, кожний блок відображається для певного типу користувача, це було реалізовано для того щоб додавати певний функціонал до кожного типу облікового записку. Перший блок відповідає за базову інформацію та відображається у всіх трьох типах облікового запису. (рисунок 2.15)

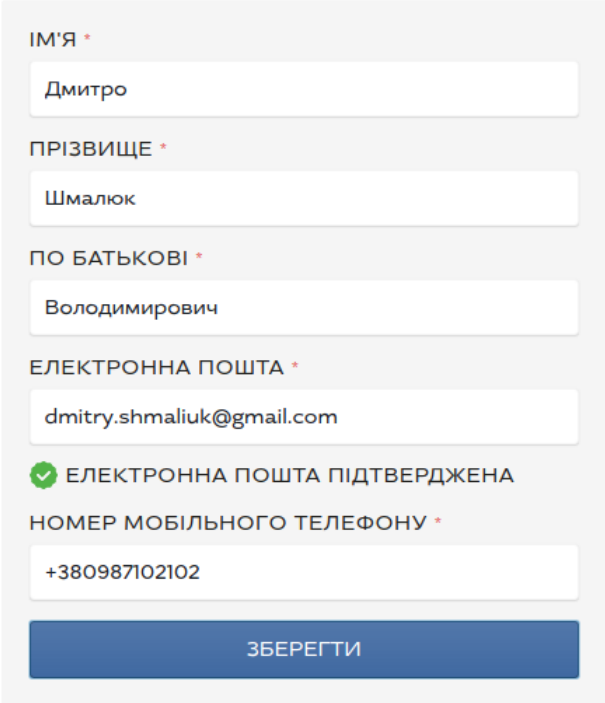


Рисунок 2.15 - Інформація про обліковий запис

На рисунку 2.15 під текстовим полем «Електронна пошта» відображається інформація про підтвердження електронної пошти, саме на цій сторінці, користувач який пропустив підтвердження електронної пошти при реєстрації, може підтвердити електронну пошту. Якщо користувач змінить електронну пошту, та збереже данні, веб-додаток автоматично помітить електронну адресу, як не підтверджену, що призведе до появи повідомлення «Підтвердити електронну пошту», (рисунок 2.16).

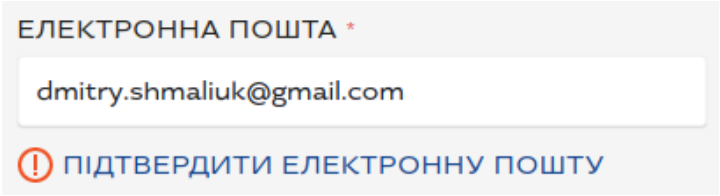


Рисунок 2.16

Після того як користувач натисне на повідомлення «Підтвердити електронну пошту», з'явиться модальне вікно, яке було розглянуто при реалізації функціональності автентифікації та реєстрації, (рисунок 2.12).

Другий блок відповідає за інформацію, яка відноситься до університету. Цей блок з'являється у облікових записів типу викладач та студент, (рисунок 2.17).

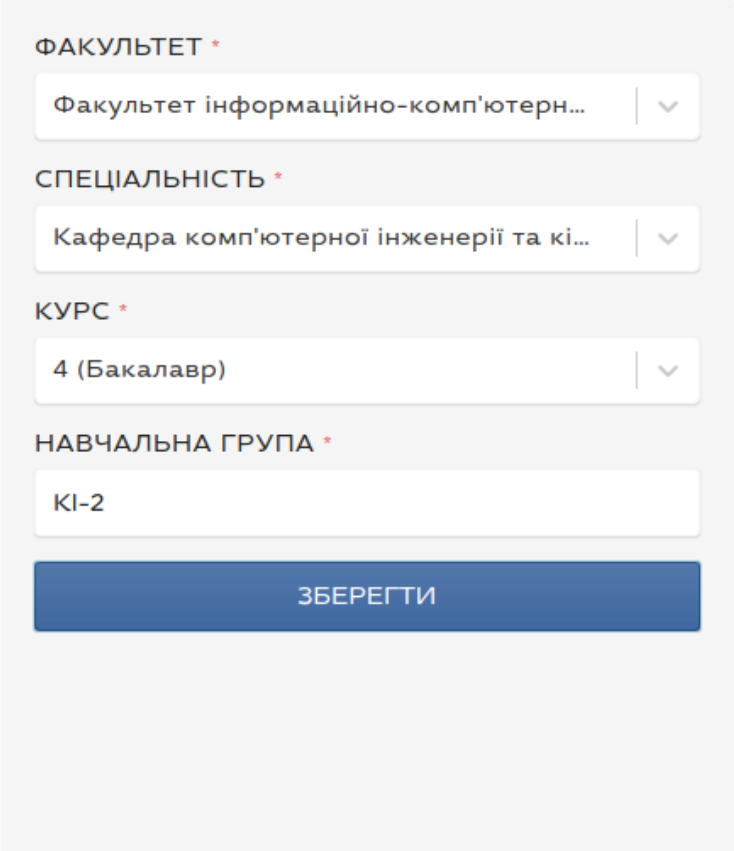


Рисунок 2.17 - Інформація про факультет, спеціальність, курс та навчальну групу

Користувач має можливість вибрати факультет на якому навчається. Після чого автоматично будуть підібрані спеціальності, які відносяться до вибраного факультету. Після того як користувач заповнює поле факультет, спеціальність та курс, розблоковується поле «Навчальна група», і тільки після введення всіх даних,

користувач має можливість зберегти дані. Даний алгоритм дій, був розроблений для типу користувача студент, а для викладача, компонент реалізує зовсім інший алгоритм, по-перше поле «Курс» та «Навчальна група» не відображаються, та по-друге, викладач має можливість зберегти дані, одразу після вибору факультету, до якого відноситься.

Третій блок відповідає за відображення, та можливу зміну студентського квитка. Відповідно до функціональності блоку, даний блок відноситься до типу облікового запису «Студент», (рисунок 2.18).

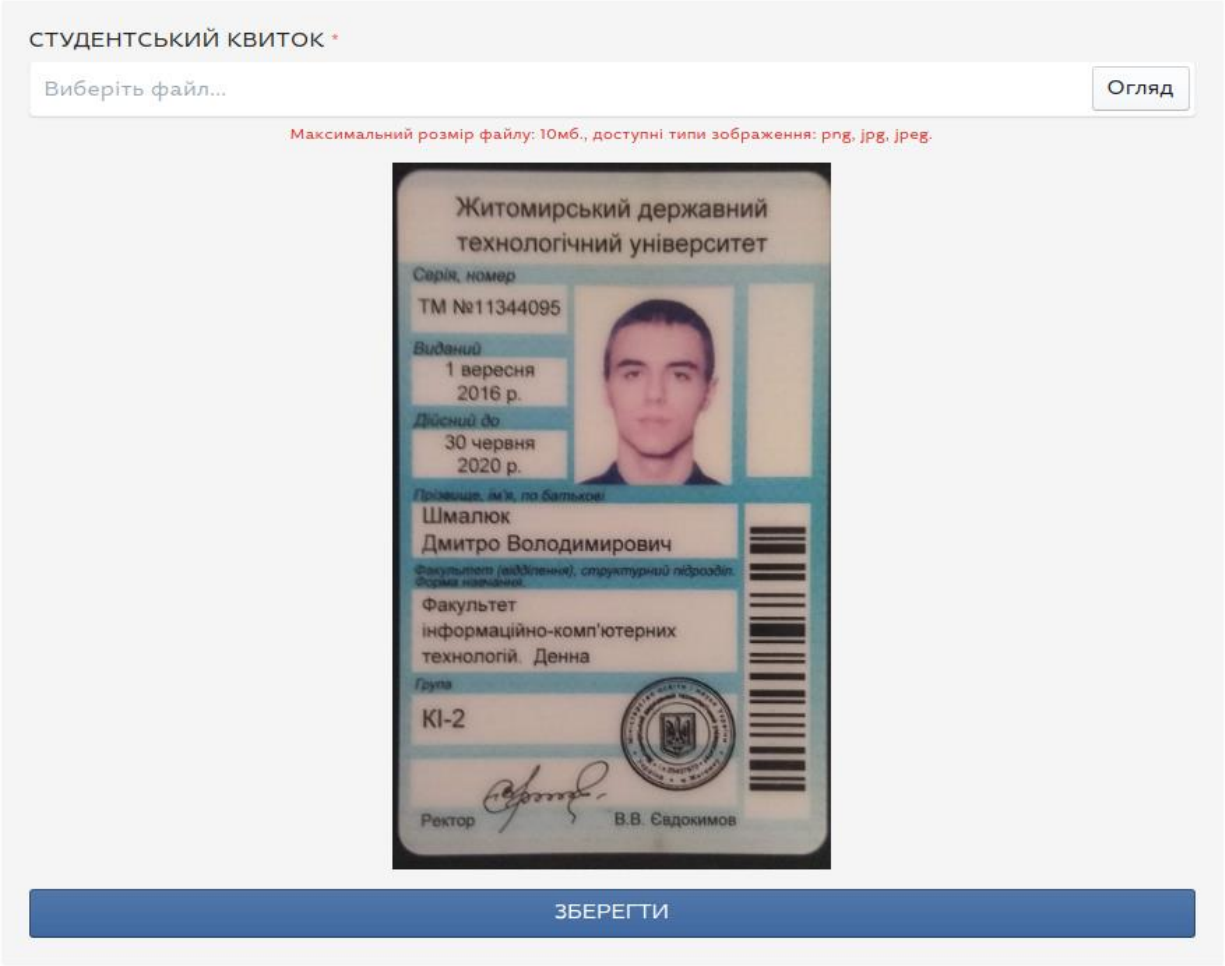


Рисунок 2.18 – Інформація про студентський квиток

Розроблюючи блок, який відноситься тільки до типу облікового запису «Студент», переходимо до сторінки «Моя успішність», (рисунок 2.19). Дана сторінка надає студенту контролювати свою успішність, вказуючи всі потрібні дані для цього, тобто назву предмета, викладача, оцінку, дату складання, тощо.

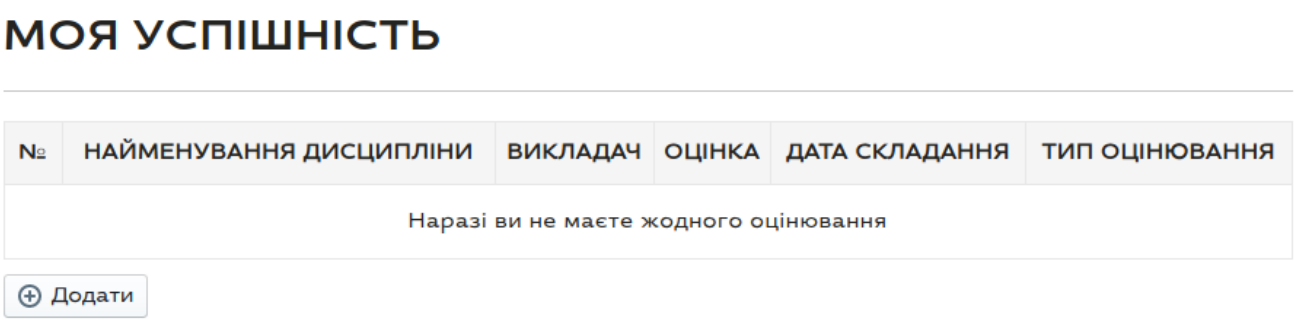


Рисунок 2.19 – Таблиця для відображення оцінювання студента

Для заповнення таблиці успішність використовується спеціально розроблено модальне вікно, (рисунок 2.20), за допомогою аналізів проведених в першому розділі. Даний компонент має назву **Assessment**, та наступну структурну схему.

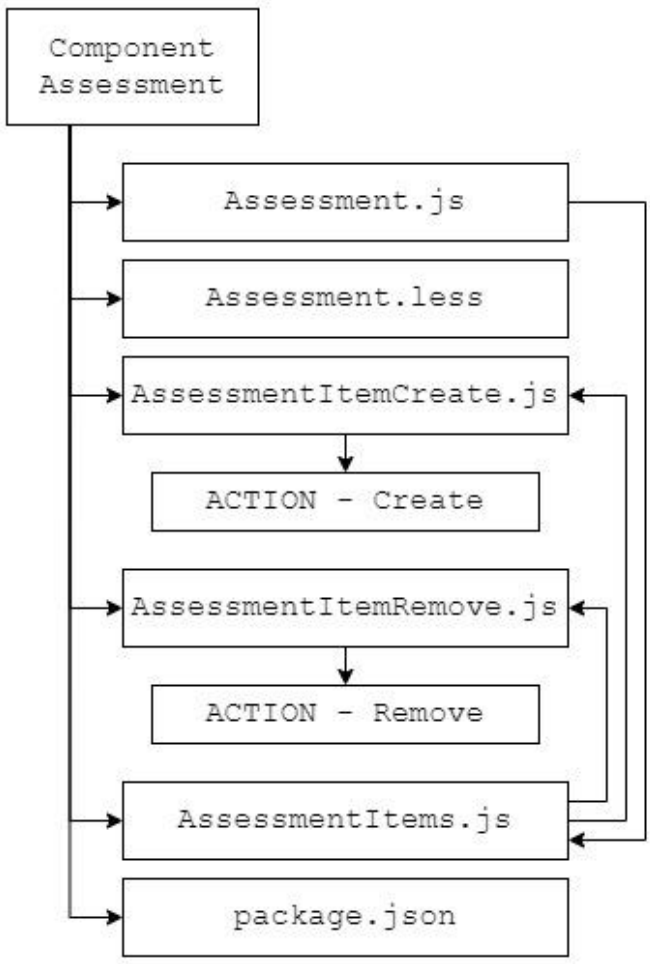


Рисунок 2.19 – Структурна схема компоненту **Assessment**

Для виконання всіх правил, потрібно було розділити обов’язки, таким чином **Assessment.js** виконує роль контейнеру, який відображає елементи за допомогою компоненту **AssessmentItems.js**, в той же час, **AssessmentItems.js** надає можливість

видалити певний елемент, та якщо розробник бажає відображати функцію додавання оцінювання, відображається кнопка «Додати», таким чином дотримуючись всіх правил, було створено компонент, який має три функції, відображення інформації, додавання та видалення. Звернемо увагу на те, що даний компонент є важливим для створення форми, для оформлення індивідуального графіку навчання, оскільки як викладачу, так і будь якому співробітнику університету, потрібно знати про успішність студента.

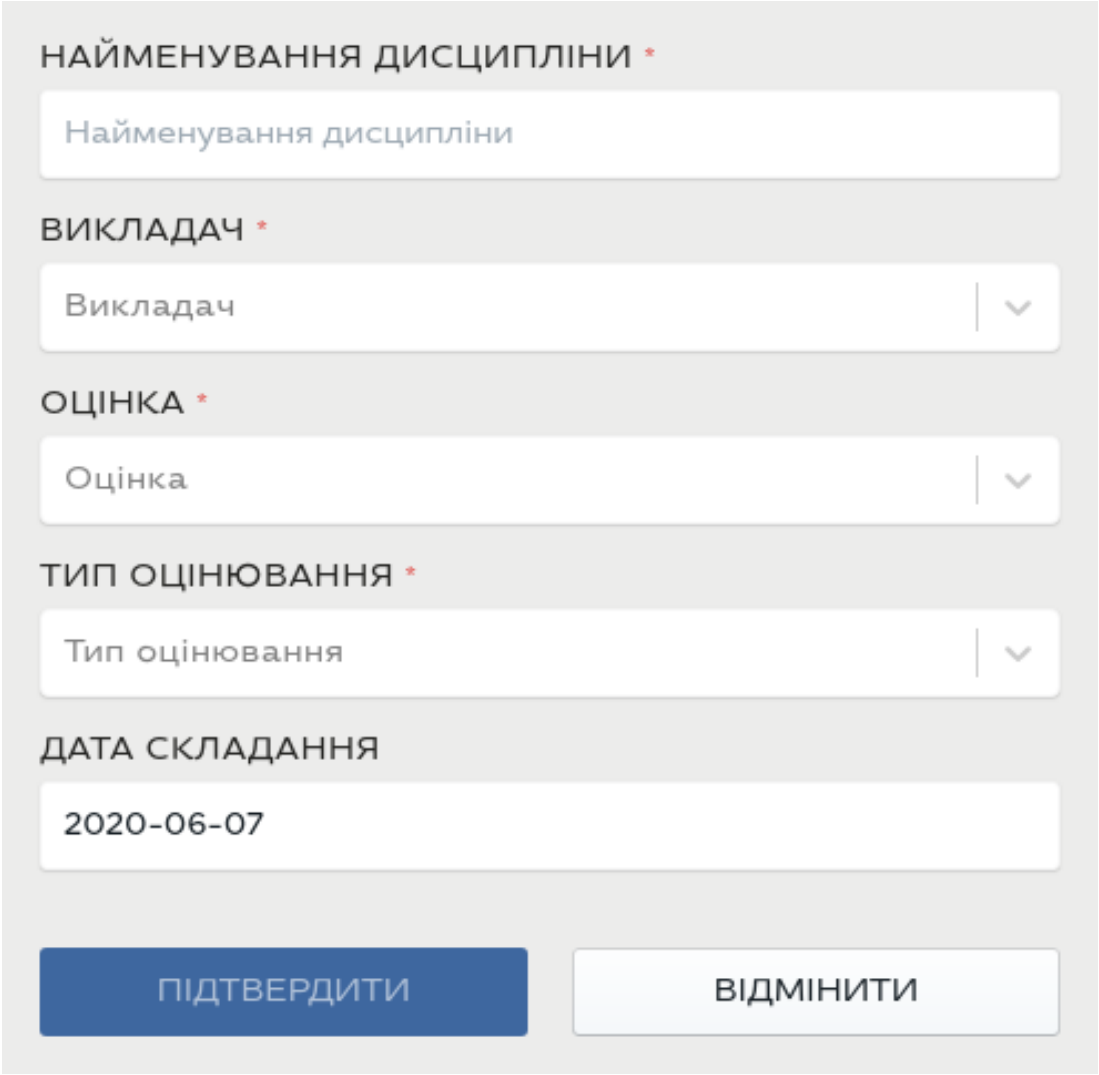


Рисунок 2.20 – Продовження рисунку 2.19

Після ведення всіх даних, при натисканні кнопки підтвердити, дані будуть відправлені на сервер, після чого вся інформація перезавантажиться, на в таблиці «Моя успішність» буде доданий новий запис.

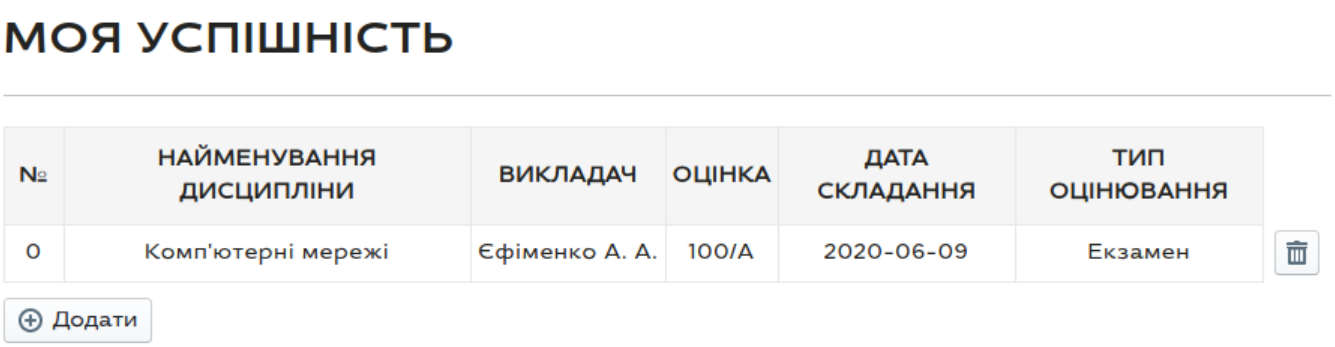


Рисунок 2.21 – Продовження рисунку 2.19

Таким чином, підготувавши всі потрібні дані, та можливість їх заповнити, переходимо до реалізації основного компоненту веб-додатку, сторінка «Послуги». Саме за допомогою даної сторінки, студент має можливість вибрати послугу оформлення індивідуального графіку.

2.2.3 Сторінка для надання послуг користувачу

Сторінка для відображення послуг наразі буде складатися з одного контент блоку, який при натисканні буде перенаправляти на сторінку оформлення індивідуального графіку, (рисунок 2.22).

Кожен тип облікового запису має певний масив послуг, який є доступним цього типу користувача. Наприклад для студента доступна послуга оформлення індивідуального графіку навчання, в той час коли для викладача, та для звичайного користувача, дана послуга не є доступною.

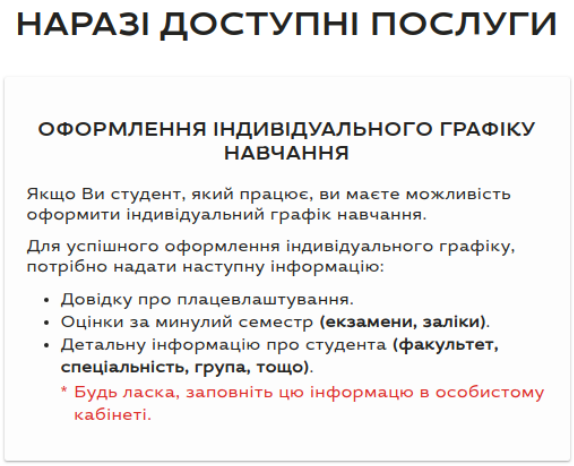


Рисунок 2.22 – Відображення наразі доступної послуги «Оформлення індивідуального графіку навчання»

2.2.3.1 Послуга оформлення індивідуального графіку навчання

Процес оформлення індивідуального графіку, починається з першого кроку **«Introduction»**, на першому кроці відбувається перевірка всієї потрібної інформації, та відображення цієї інформації з проханням до користувача, перевірити її коректність. Сторінка оформлення індивідуального графіку має дуже простий інтерфейс, в який входить: заголовок, елемент відображення прогресу, основний контейнер для роботи з даними, та контейнер з елементами інтерфейсами, які дозволяють переходити між кроками.

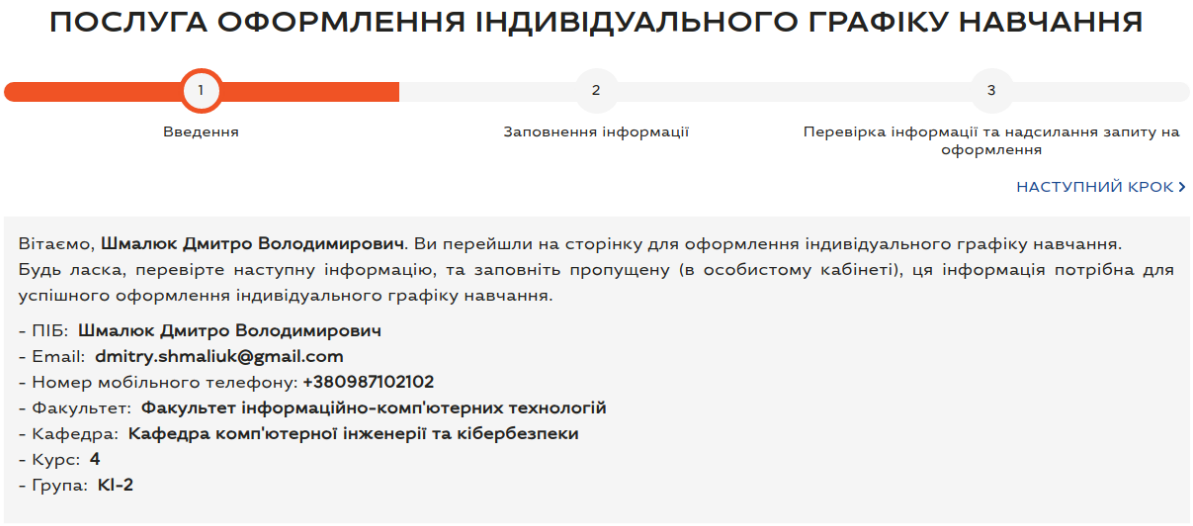


Рисунок 2.23 – Перший крок «Оформлення індивідуального графіку навчання»

Існує декілька моментів, які були реалізовані, наприклад, якщо користувач не має інформації про ПІБ, при привітанні буде відображатися логін користувача, або якщо інформація про факультет, кафедру, тощо, не буде вказана, будуть відображатися пропуски, та елемент інтерфейсу «Наступний крок» буде заблокований.

Після заповнення всіх даних користувач переходить до другого кроку, на якому вказує назву компанії, назву посади, яку займає в компанії, довідку про працевлаштування, інформацію про останні результати сесії та предмети по яким потрібна можливість індивідуального навчання.

При реалізації другого кроку виникла потрібність збереження даних в **localstorage**, це забезпечить можливість відновлення даних, якщо користувач

випадково перезавантажить сторінку. Даний функціонал реалізований за допомогою **React Hooks**.

React Hooks - це нове доповнення в **React 16.8**. Вони дозволяють використовувати стан та інші функції **React** без написання класу. На початку розділку був розглянутий Hook стану, він зберігає інформацію як і звичайна змінна, але якщо сторінка буде перезавантажена, змінна оновиться, і дані які були в ній записані, пропадуть, тому, для вирішення цього питання застосуємо технології які нам доступні, тобто ми об'єднаємо **localstorage** та **useState Hook**.

```
import React, { useEffect, useState } from 'react';

export const useStickyState = (defaultValue, key) => {

    const [ value, setValue ] = useState(() => {
        const stickyValue = window.localStorage.getItem(key);

        return stickyValue !== null
            ? JSON.parse(stickyValue)
            : defaultValue;
    });

    useEffect(() => {
        window.localStorage.setItem(key, JSON.stringify(value));
    }, [ key, value ]);

    return [ value, setValue ];
}
```

Для використання вище описаного **React Hook**, використовуємо наступну конструкцію:

```
const [
    IndividualSchedulingData, setIndividualSchedulingData
] = useStickyState(
  {
    companyName: "",
    companyPosition: "",
    certificateOfEmployment: null,
    ...
  },
  "IndividualSchedulingData"
);
```

Таким чином перед оголошення нових даних, даний **hook**, буде перевіряти наявність інформації в **localstorage**.

Для вказання предметів, для індивідуального навчання на майбутній семестр, користувач має можливість використати модальне вікно, в якому надається можливість вказати назву предмета та обрати відповідного викладача, після чого дані збережуться в змінну стану, та в **localstorage**. Введені дані можливо побачити в таблиці «Дисципліни для індивідуального графіку», (див. рисунок 2.24).

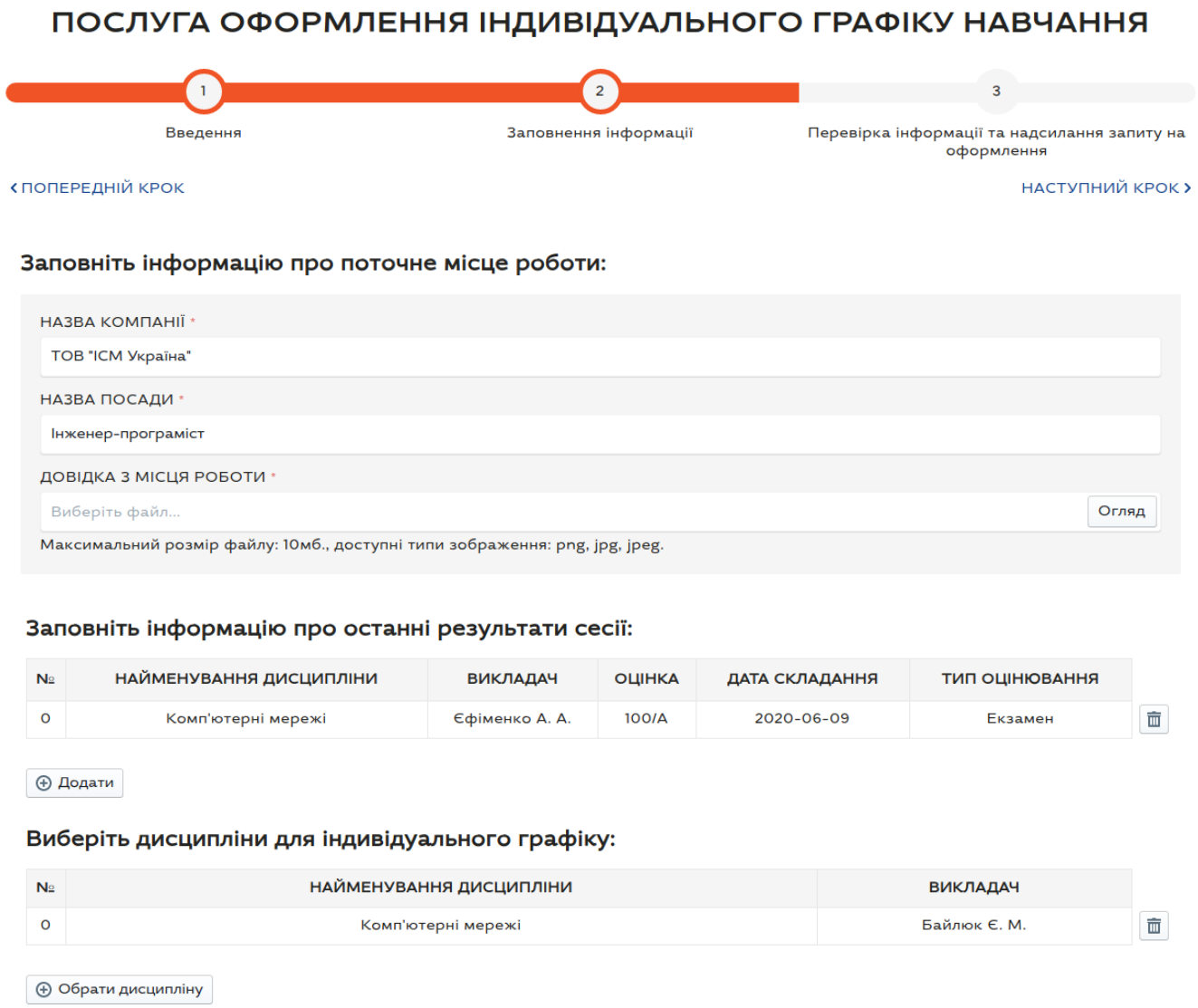


Рисунок 2.24 – Другий крок «Оформлення індивідуального графіку навчання»

В результаті проведеної розробки, маємо два кроки для заповнення важливої інформації. При переході на третій крок реалізовано умови перевірки заповнених даних на другому кроці, тобто, користувач не має можливості подати заяву на оформлення індивідуального графіку навчання, якщо певні дані не заповнені.

Після того як користувач заповнив всі дані коректно, користувачу надається можливість вибору дати дії документу, тобто визначити період часу, в діапазоні якого буде дійсний індивідуальний графік навчання. Також на третьому кроці, надається довідкова інформація для користувача, про те кому будуть надіслані запити. На рисунку 2.25 продемонстровано, що запити будуть надіслані декану факультету інформаційно-комп'ютерних технології, завідувачу кафедри, тощо.

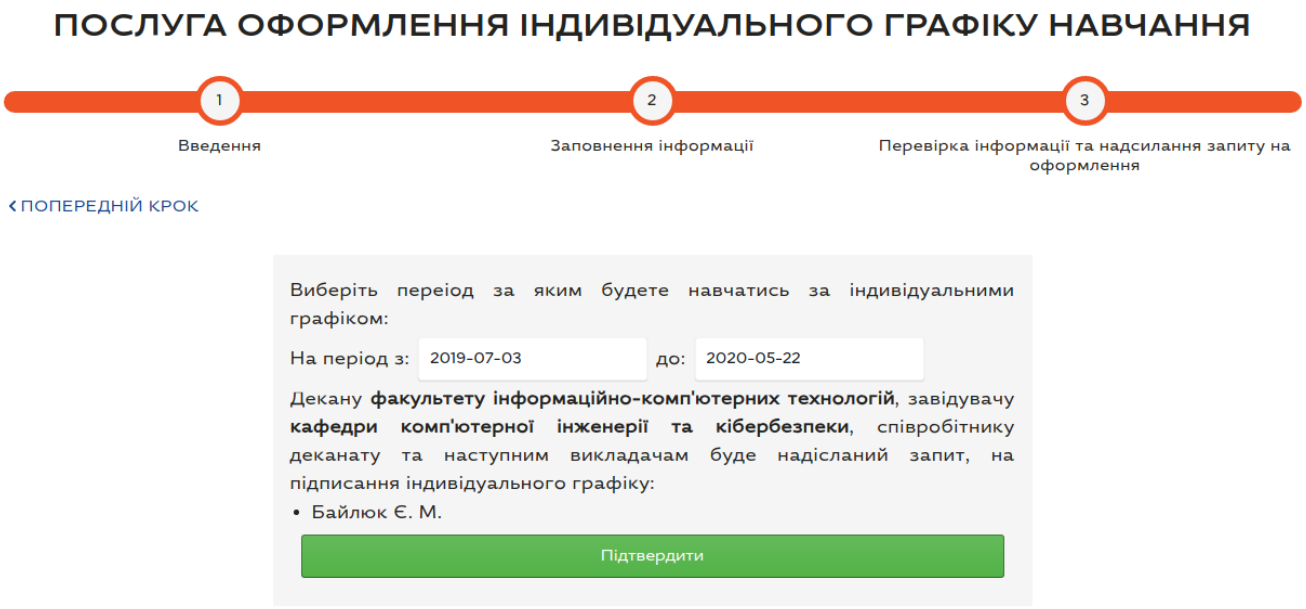


Рисунок 2.25 – Третій крок «Оформлення індивідуального графіку навчання»

Після перевірки всієї інформації користувач натискає кнопку «Підвередити», після чого, запити в першу чергу будуть надсилатися до співробітників деканату, а тільки після того, як співробітник деканату надає дозвіл на надсилання запитів до викладачів, надсилається запит на підтвердження до викладачів, але наразі момент з підтвердженням, який стосується співробітника деканату, пропущений, та закладений на реалізацію в майбутньому.

Надалі, студент має можливість оглядати процес, виконання запиту в особистому кабінеті, (рисунок 2.26). Це надає можливість контролювати процедури оформлення індивідуального графіку, та уразі необхідності зв'язуватися з деканатом або викладачем, для вирішення організаційних моментів. Слід зауважити на тому, що даний процес здійснюється автоматизовано, що особливо актуально для організації дистанційного навчання.

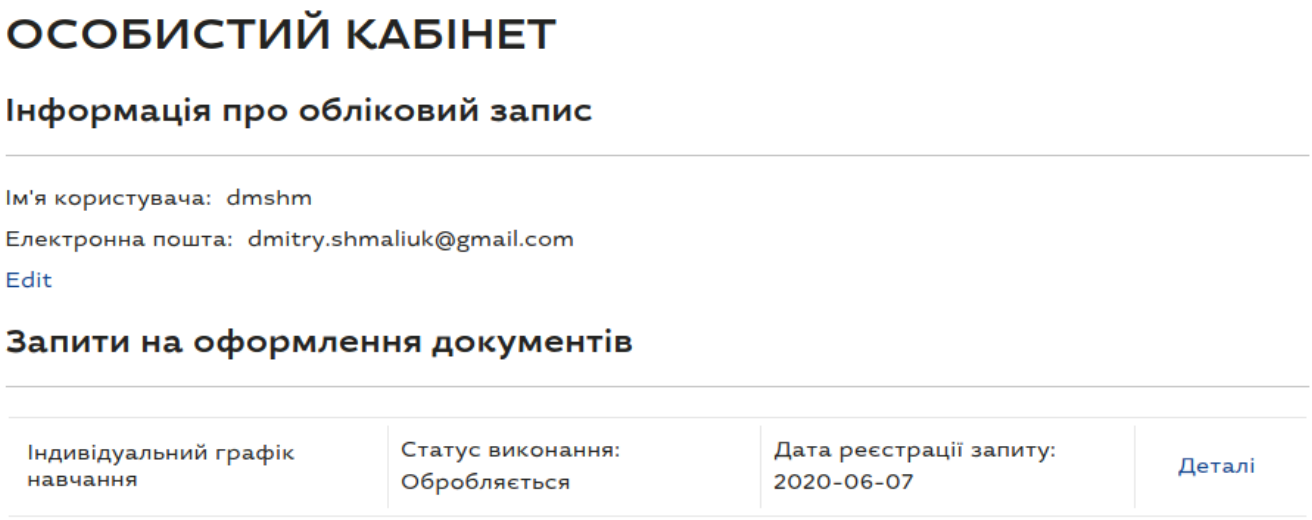


Рисунок 2.26 – Можливість переглядання запитів на оформлення документів, в особистому кабінеті

Таким чином відбувається інформованість користувачів щодо формування запитів на оформлення документів.

2.2.4 Перегляд запитів на оформлення документів

Для комфортного перегляду запиту на оформлення певного документу зі сторони студента, та для керування запитами з сторони викладач, було розроблено спеціальний компонент який відповідає за дану функціональність.

Одразу після надсилання запиту на оформлення, користувач в особистому кабінету має можливість перегляду даних, які були вказані під час заповнення форма, на теперішній момент, зміна всіх даних не можлива, а тільки загальних, такий як ім’я, поштова скринька чи номер мобільного телефону. Теоретично ці дані можуть бути змінені.

Отож, при переходу до власного кабінету, користувач отримує можливість перегляду детальної інформації, яка відноситься до запиту користувача, (рисунок 2.27).

Модальне вікно поділяється на дві секцій, в першій секції відображається інформація про користувача, та обов’язкові дані які відносяться до певного типу документа. В другій секції відображається список викладачів, та статус виконання, до яких був надісланий запит на підтвердження.



Рисунок 2.27 – Перегляд детальної інформації запиту на оформлення документа

В правому верхньому кутку, відображається додаткові функції, такі як формування **PDF** документа, та кнопка для відкриття довідки з місця роботи в новому вікні браузера, це зроблено для того, щоб не засмічувати інтерфейс модального вікна, та полегшити його.

2.2.5 Глобальні повідомлення

При виконанні будь якої дії в будь якому веб-додатку, може виникнути проблема, або навпаки, потрібно нотифікувати користувача про успішно виконану операцію, за даний функціональність відповідає окремий модуль який має назву **GlobalMessage**. Всього для нотифікація користувача розробника має три типи повідомлень.

1. Повідомлення про успіх. Даний тип повідомлень нотифікую користувача про успішно виконану операцію, (рисунок 2.28).

2. Для того щоб нотифікувати користувача про щось важливе, використовується другий тип повідомлення, (рисунок 2.29).

3. Третій тип повідомлень, повідомляє користувача про помилку, яка виникла в ході виконання певного запиту, (рисунок 2.30).

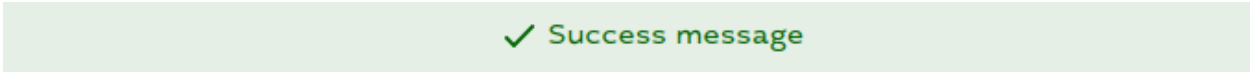


Рисунок 2.28 – Вигляд повідомлення про успіх



Рисунок 2.29 – Вигляд повідомлення, для нотифікація користувача



Рисунок 2.30 – Повідомлення про помилку

Для того щоб скористатися глобальним повідомленням, по-перше, в компоненті потрібно оголосити обробник змінних стану, та за допомогою певної конструкції передати повідомлення, та його тип. На прикладі підтвердження електронної адреси, демонструємо конструкцію виклику глобального повідомлення:

```
props.globalMessageShow(
     'Email has been confirmed!', GlobalMessageType.SUCCESS
);
globalMessageShow: (message, messageType) => dispatch(
     actions.globalMessageShow(message, messageType)
);
props.globalMessageShow(
    error, GlobalMessageType.WARNING
);
const mapDispatchToProps = dispatch => {
    return {
        globalMessageShow: (message, messageType) => dispatch(
            actions.globalMessageShow(message, messageType
        ))
    };
};
```

2.2.6 Домашня сторінка

Домашня сторінка була створена для систематизації всіх наявних сервісів Державного університету «Житомирська політехніка» на одній сторінці, та для того щоб демонструвати останні новини, тощо, (рисунок 2.31). При розробці домашньої сторінки дотримувалася кольорова гамма, для того щоб весь веб-додаток виглядав однаково.

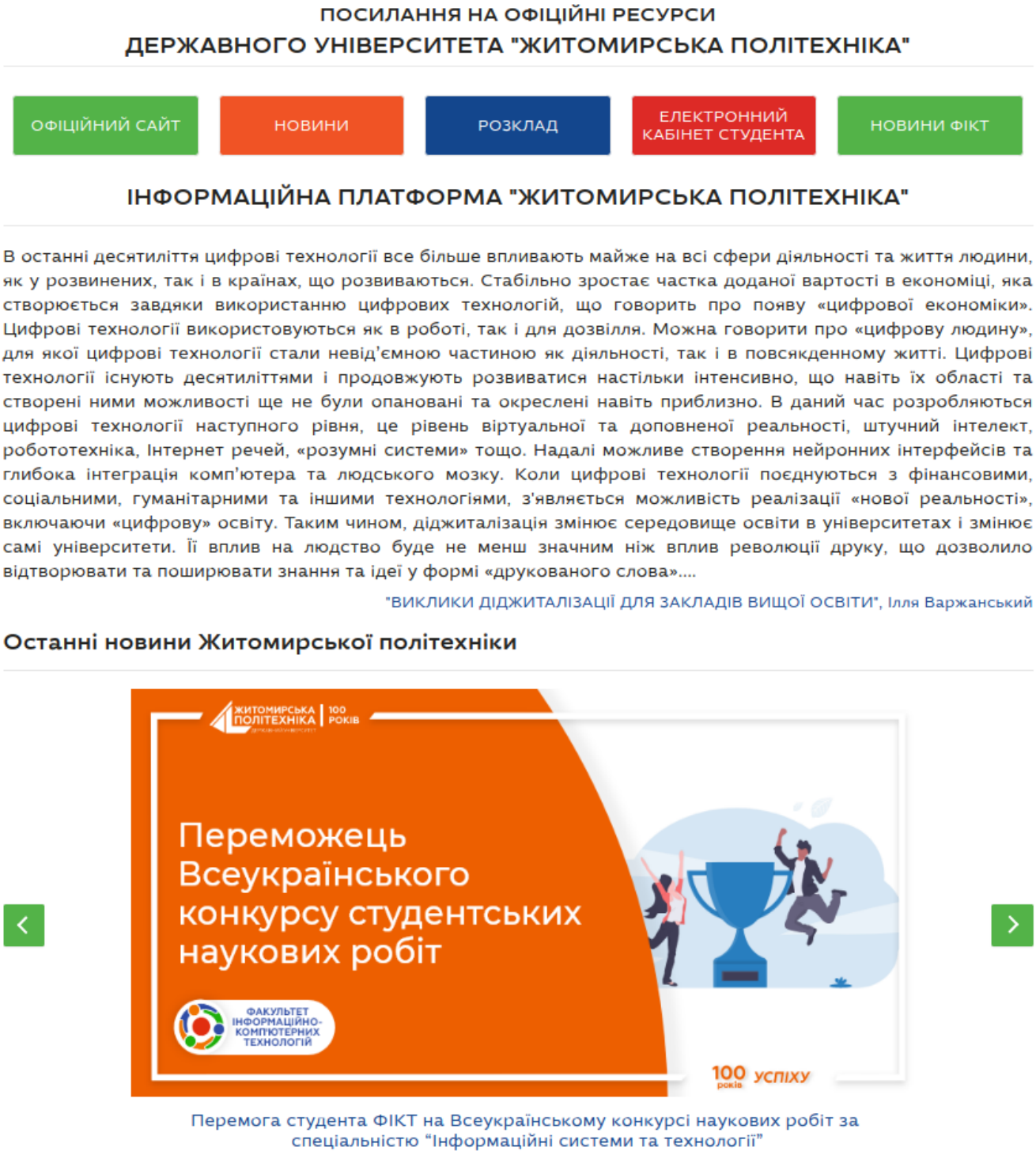


Рисунок 2.31 – Домашня сторінка

Таким чином інформаційна платформа здатна до розширення та за бажанням може включати ще більше інформації.

Висновки до другого розділу

1. В ході виконання другого розділу було розроблено **client-side** та **server-side** додатки, які разом складають інформаційну платформу Державного університету «Житомирська політехніка».

2. Запропоновано модуль для оформлення індивідуального графіку студента, що включає модуль збереження даних про працевлаштування та збереження даних всіх запитів щодо оформлення індивідуального графіку. Сформовано сервісні таблиці, що відповідають за збереження проміжних даних, які використовується для забезпечення коректної роботи оформлення, та використання індивідуального графіку.

3. Запропоновано форми реєстрації користувачів інформаційної платформи з підтримкою методів автентифікації.

4. Запропоновано структуру персонального кабінету користувача та проведено її практичну реалізацію із визначеним та описаним функціоналом.

5. Проведено реалізацію надання послуг, яка наразі включає один контент блок для оформлення індивідуального графіку. Реалізовано послугу оформлення індивідуального графіку з підтримкою контролю стану оформлення.

РОЗДІЛ 3. ТЕСТУВАННЯ ВЕБ-ДОДАТКУ

В даному розділі буде продемонстрована робота з веб-додатком, під час якої буде виконано тестування його функціональних можливостей. Процесом тестування називають розслідування, яке проводиться з метою надання зацікавленим особам інформації про якість досліджуваного програмного продукту чи послуги. Тестування програмного забезпечення також може забезпечити об'єктивний, незалежний погляд на програмне забезпечення, що дозволить оцінити та зрозуміти ризики впровадження програмного забезпечення. Методи тестування включають процес виконання програми та з метою пошуку програмних помилок (помилок чи інших дефектів) та перевірки відповідності програмного продукту для використання.

3.1 Автентифікація та реєстрація користувача

Почнемо з реєстрації користувача. Створимо користувача типу «Студент», рисунок 3.1.

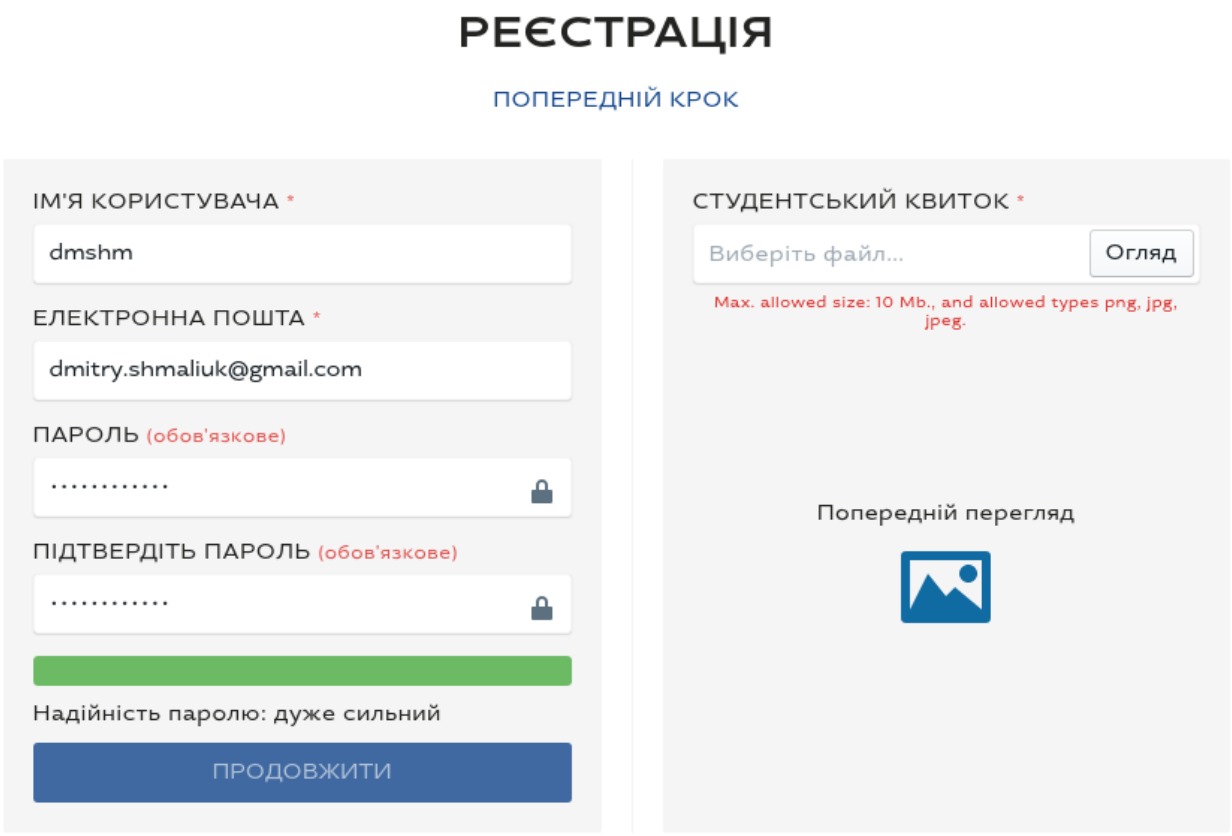


Рисунок 3.1 – Реєстрація користувача

Після чого виконаємо підтвердження електронної пошти, рисунок 3.2, за допомогою коду **1381354** який був надісланий на електронну пошту, вказану при реєстрації.

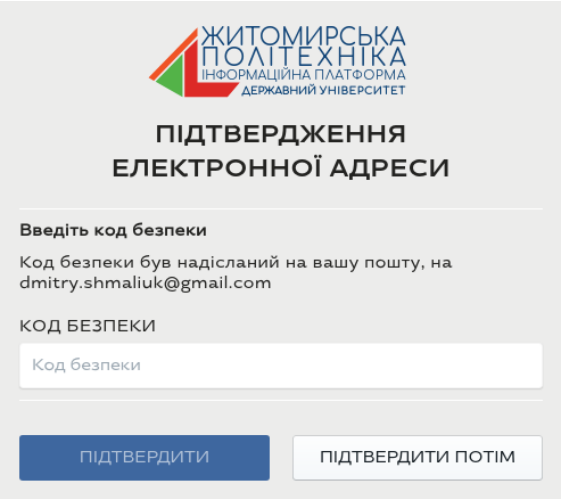


Рисунок 3.2 – Модальне вікно підтвердження електронної адреси

Після натискання кнопки підтвердити отримаємо повідомлення про успішне підтвердження електронної пошти, рисунок 3.3.



Рисунок 3.3 – Повідомлення про успішно підтверджену електронну адресу

Зареєструвавши користувача, ті підтвердивши електронну адресу, переходимо до авторизації користувача, рисунок 3.4.

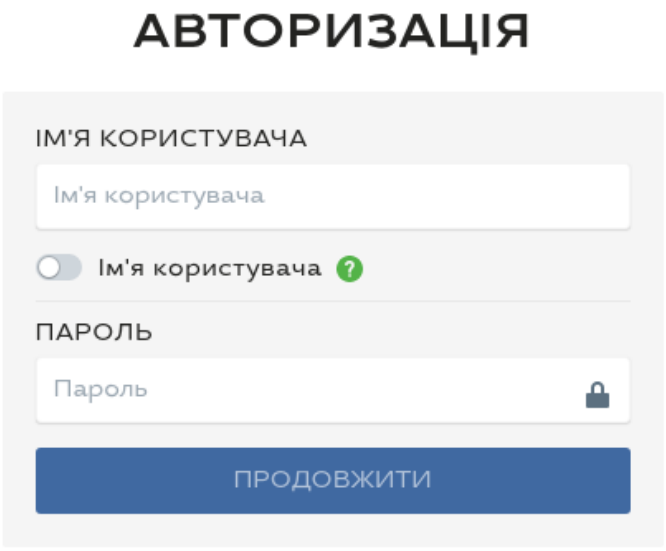


Рисунок 3.4 – Форма авторизації користувача

Пройшовши етап авторизації, отримуємо повідомлення про успішну авторизацію та програма «завантажує» особистий кабінет користувача.

3.2 Особистий кабінет

За допомогою навігації веб-додатку, рисунок 3.5, переходимо до особистого кабінету, щоб заповнити детальну інформацію про користувача, рисунок 3.6.

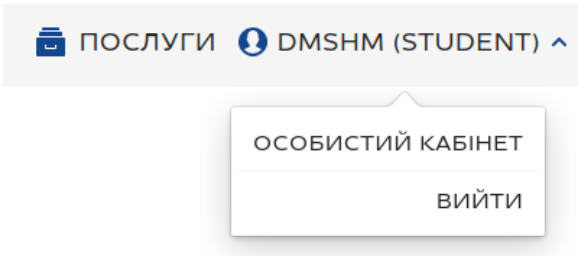


Рисунок 3.5 – Використання навігації веб-додатку для переходу до особистого кабінету

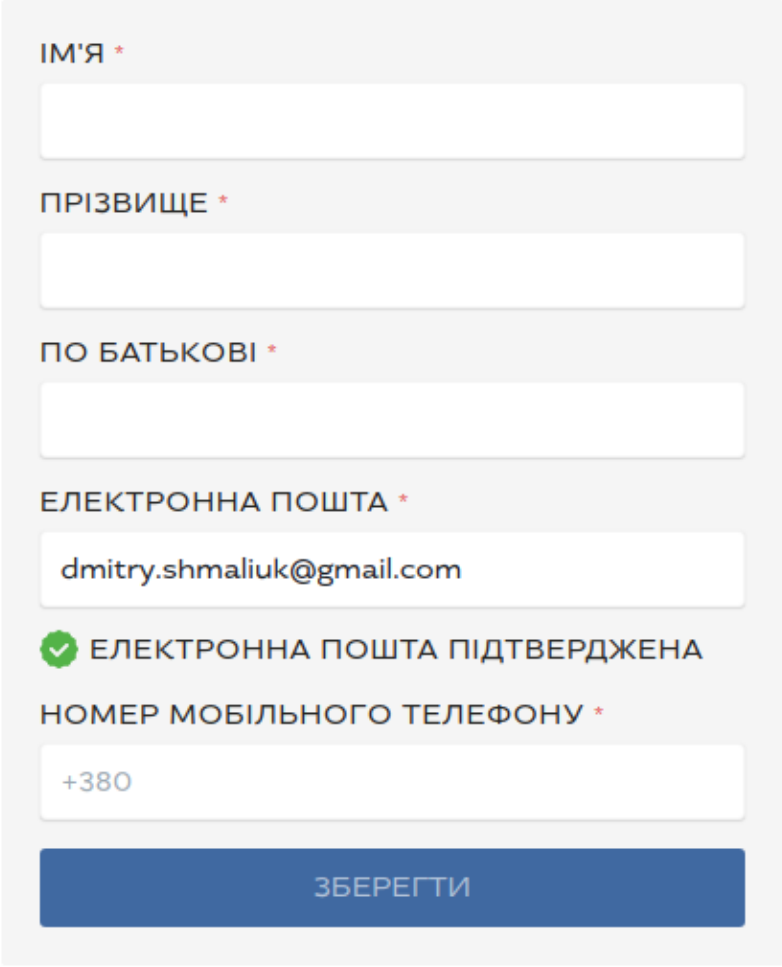


Рисунок 3.6 – Форма для заповнення даних

Таким чином, заповнюємо всю потрібно інформацію, натискаємо кнопку зберегти, та переходимо до заповнення інформації, яка відноситься до університету, рисунок 3.7.

При заповненні інформації яка відноситься до університету, вибираємо факультет (див. рисунок 3.7), спеціальність (див. рисунок 3.8), курс (див. рисунок 3.9), та заповнюємо поле група. Звертаємо увагу на те, що після вибору факультету, спеціальності завантажуються відповідно до вибраного факультету.

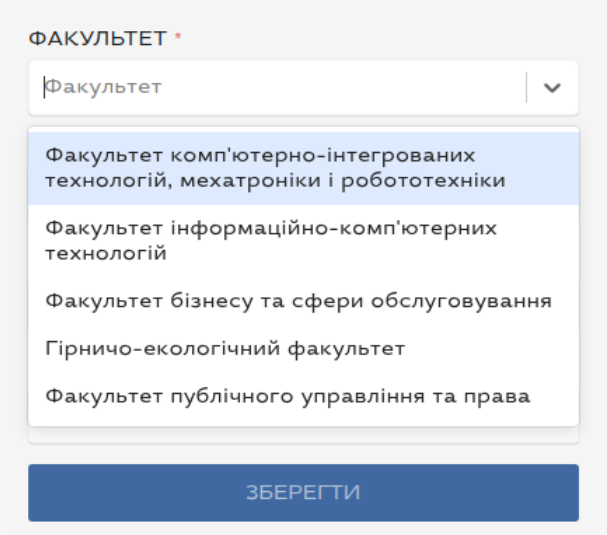


Рисунок 3.7 – Форма вибору факультету

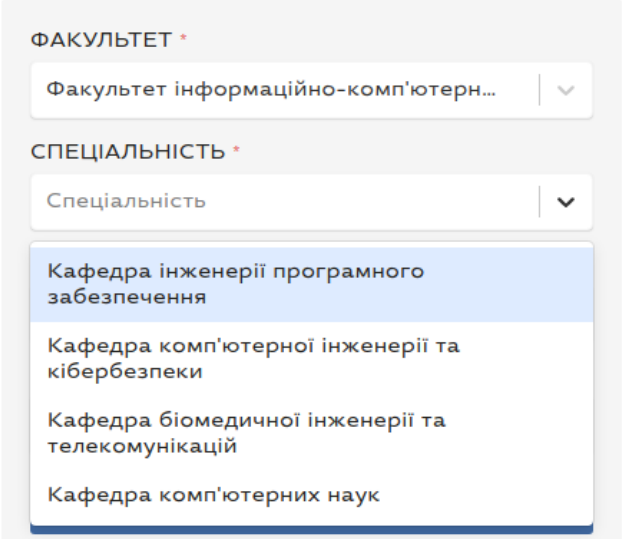


Рисунок 3.8 – Форма вибору спеціальності

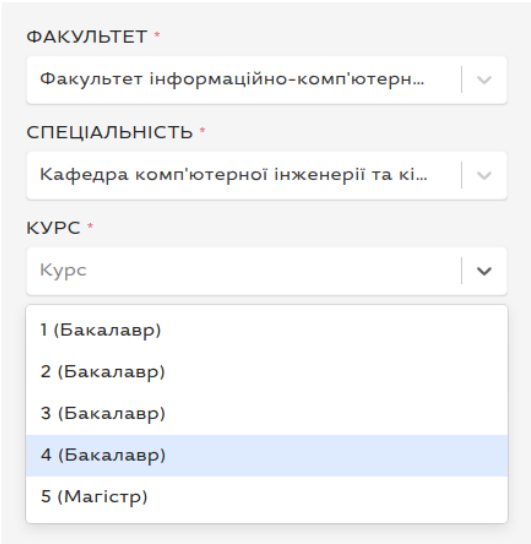


Рисунок 3.9– Форма вибору курсу

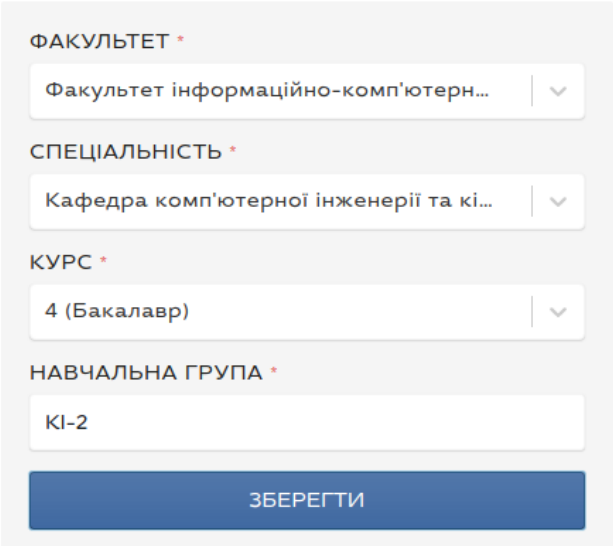


Рисунок 3.10 – Форма вибору групи

Після збереження даних, отримуємо наступне повідомлення, (рисунок 3.11).



Рисунок 3.11– Повідомлення про успішне збереження даних користувача

Після заповнення всіх обов'язкових даних, переходимо до заповнення інформації про успішність. Отже, натискаємо на елемент інтерфейсу «Моя успішність», та натискаємо на кнопку «Додати», після чого відкривається модальне вікно, що представлено на рисунку 12, за допомогою якого вводимо всі потрібні

дані, та зберігаємо оцінювання. Звертаємо увагу на те, що користувач отримує можливість пошуку потрібного викладача, за першими символами.

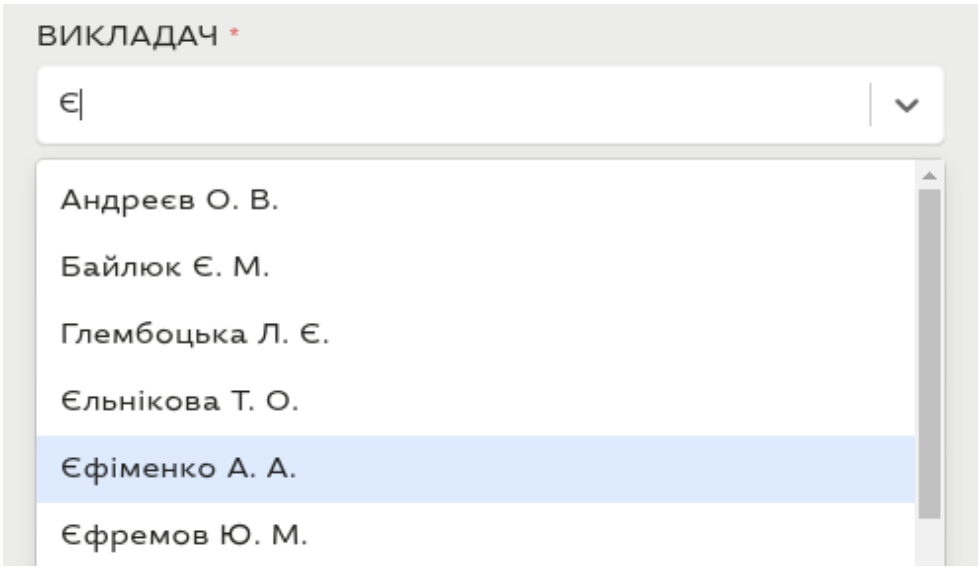


Рисунок 3.12 – Процес вибору викладача

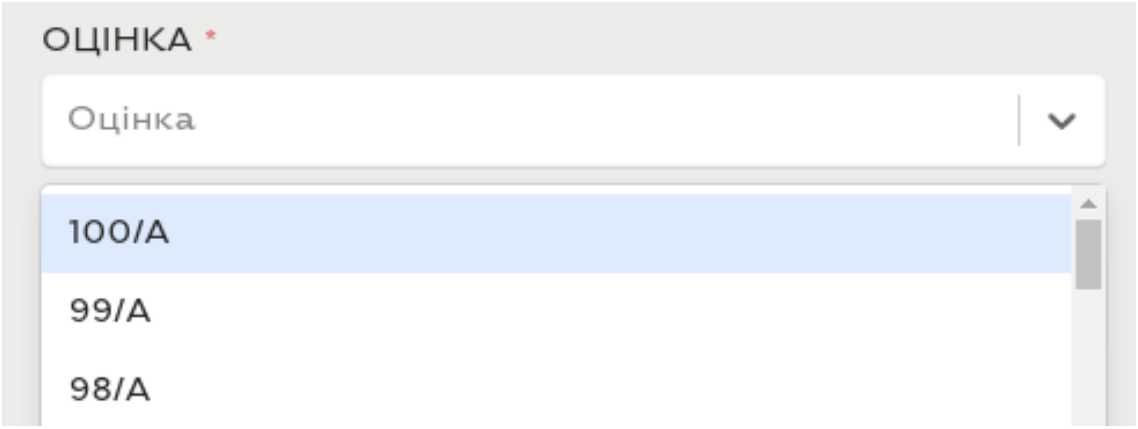


Рисунок 3.13 – Вибір оцінки

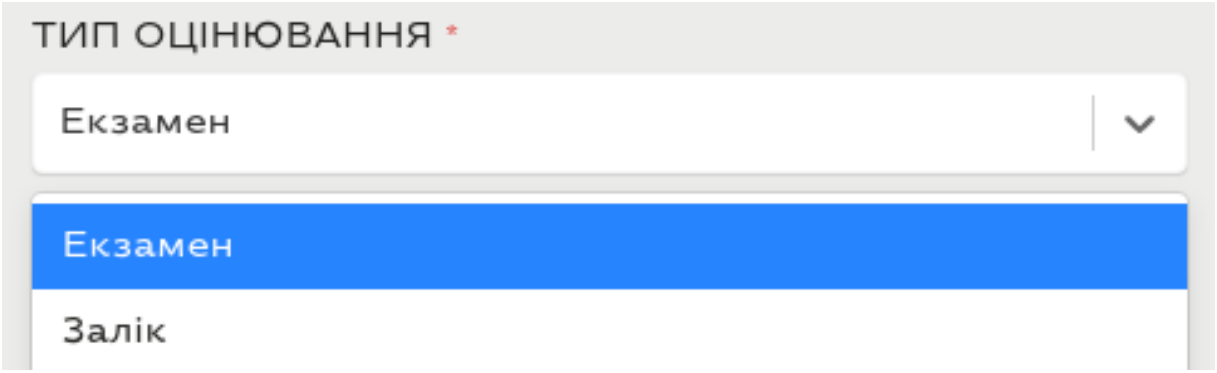


Рисунок 3.14 – Вибір типу оцінювання

Після збереження даних, отримуємо таблицю з одним оцінюванням, рисунок 2.21.

3.3 Процес оформлення індивідуального графіку

Приклад заповнення даних, був наведений в розділі другому, під час розробки кроку, що відповідав за оформлення індивідуального графіку (див. рисунок 2.23-25), тому переходимо до найголовнішого, генерація документу, та дозвіл викладача. Почнемо з того, що кожен студент, або викладач, при натисканні кнопки «Відрити PDF документ» (рисунок 3.15), отримує згенерований документ (рисунок 3.16).



Рисунок 3.15 – Кнопка «Відкрити PDF документ»

Ректору ДУ "Житомирська політехніка"
проф. Євдокимову В.В.
студента факультету ІКТ
денної форми навчання
спеціальності комп'ютерної інженерії та кібербезпеки
бюджетної форми фінансування
Шмалюк(а) Дмитро(а) Володимирович(а)
+380987102102

ЗАЯВА

Прошу перевести мене на навчання за індивідуальним графіком на період з 2019-07-03 до 2020-05-22, в зв'язку з тим, що я працюю в компанії ТОВ "ІСМ Україна" на посаді Інженер-програміст.

З умовами переводу на навчання за індивідуальним графіком ознайомлений та згодний.

До заяви додано план індивідуального графіку навчання та довідка з місця роботи.

2020-06-07 _____

Рисунок 3.16 – Згенерований PDF документ

Авторизуємося під обліковим записом викладача, та виконаємо декілька тестів, перший тест, відноситься до перевірки наразі доступних послуг (рисунок 3.17).

ВИБАЧТЕ! НАРАЗІ ДЛЯ ВАШОГО ТИПУ ОБЛІКОВОГО ЗАПИСУ, НЕМАЄ ЖОДНОЇ ПОСЛУГИ, АЛЕ ЦЕ ТИМЧАСОВО!

Рисунок 3.17 – Наразі доступні послуги, для користувача типу викладач

Другий тест відноситься до перевірки елементів підтвердження або спростування запиту, на оформлення індивідуального графіку, (рисунок 3.18).

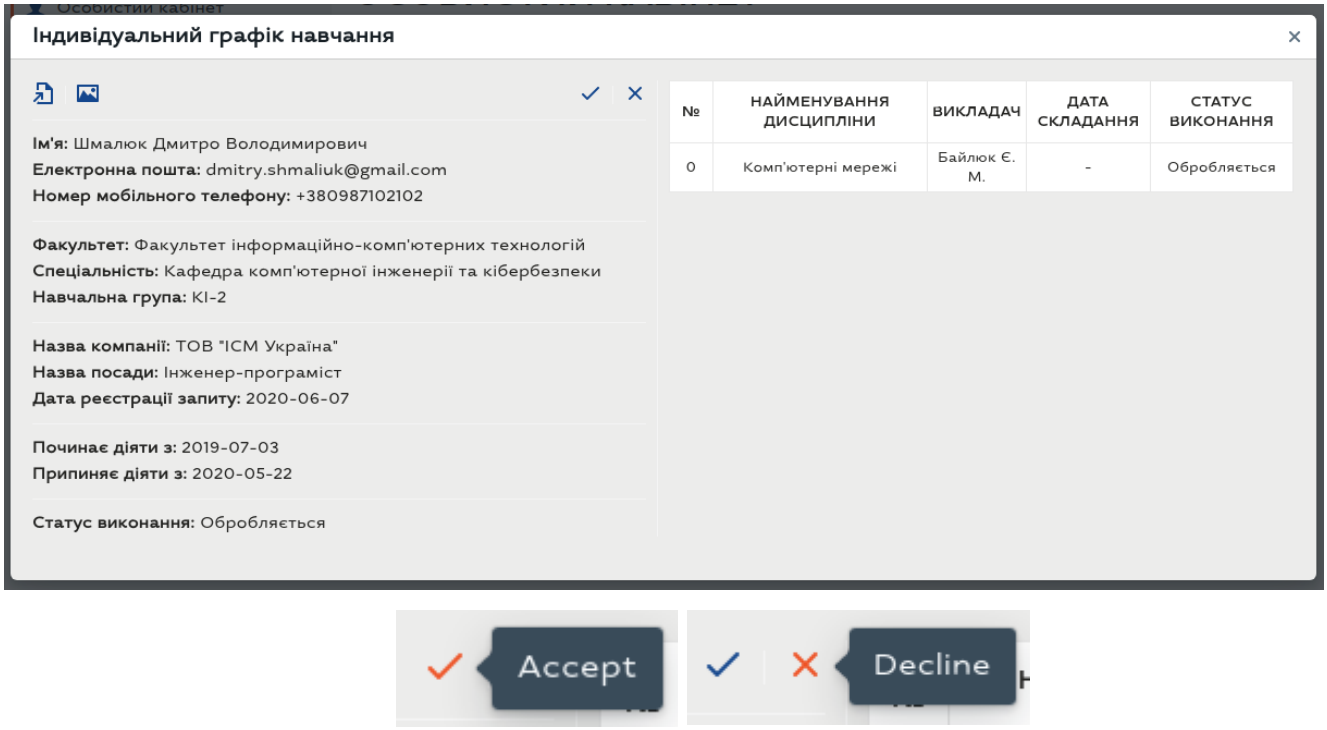


Рисунок 3.18 – Елементи для підтвердження або спростування запиту

Таким чином, перевірка працездатності веб-додатку закінчена, переходимо до формування висновків.

Висновки до третього розділу

1. Проведено тестування основних модулів розробленої захищеної інформаційної Web-орієнтованої системи для оформлення індивідуального графіку навчання студентів Житомирської політехніки в результаті якого підтверджено функціональних розроблених модулів системи та продемонстровано особливості використання та приклад заповнення форм.

ВИСНОВКИ

1. Проведено аналіз технологій для розробки інформаційної платформи дозволив сформувати вимоги до побудови веб-додатку з метою запуску процесу діджиталізації для забезпечення більш якісного процесу надання адміністративних послуг закладом вищої освіти.

2. Проведено аналіз методів розробки, методів захисту веб-додатку, в результаті якого визначено будувати веб-додаток у вигляді двох частин: клієнтської та серверної, що забезпечить якісну підтримку, та безпроблемне масштабування веб-додатку в майбутньому.

3. Було спроектовано та розроблено **client-side** та **server-side** додатки, які разом складають інформаційну платформу Державного університету «Житомирська політехніка».

4. Запропоновано модуль для оформлення індивідуального графіку студента, що включає модуль збереження даних про працевлаштування та збереження даних всіх запитів щодо оформлення індивідуального графіку. Сформовано сервісні таблиці, що відповідають за збереження проміжних даних, які використовується для забезпечення коректної роботи оформлення, та використання індивідуального графіку.

5. Запропоновано форми реєстрації користувачів інформаційної платформи з підтримкою методів автентифікації. Запропоновано структуру персонального кабінету користувача та проведено її практичну реалізацію із визначеним та описаним функціоналом.

6. Проведено реалізацію надання послуг, яка наразі включає один контент – блок для оформлення індивідуального графіку. Реалізовано послугу оформлення індивідуального графіку з підтримкою контролю стану оформлення.

7. Проведено тестування основних модулів розробленої захищеної інформаційної Web-орієнтованої системи для оформлення індивідуального графіку навчання студентів Житомирської політехніки в результаті якого підтверджено функціональних розроблених модулів системи та продемонстровано особливості використання та приклад заповнення форм.

ПЕРЕЛІК ПОСИЛАНЬ

Електронні ресурси

1. Sequelize. Режим доступу до ресурсу: https://sequelize.org/master/
2. Blueprint. Режим доступу до ресурсу: https://blueprintjs.com/docs
3. Node.js. Режим доступу до ресурсу: https://nodejs.org/docs/latest-v12.x/api/
4. React. Режим доступу до ресурсу: https://reactjs.org/
5. Require.js. Режим доступу до ресурсу: https://requirejs.org/
6. Understanding module.exports and exports in Node.js. Режим доступу до ресурсу: https://www.sitepoint.com/understanding-module-exports-exports-node-js/
7. Multi Page Application with React. Режим доступу до ресурсу: https://itnext.io/building-multi-page-application-with-react-f5a338489694
8. 5 React Architecture Best Practices. Режим доступу до ресурсу: https://www.sitepoint.com/react-architecture-best-practices/
9. React Native Login Remember Me Checkbox / Switch Solution Code. Режим доступу до ресурсу: https://medium.com/@webcore1/react-native-login-remember-me-checkbox-solution-a16efdfdf056
10. React validation. Режим доступу до ресурсу: https://www.npmjs.com/package/react-validation
11. Node JS | Password Hashing with Crypto module. Режим доступу до ресурсу: https://www.geeksforgeeks.org/node-js-password-hashing-crypto-module/
12. Redux. Режим доступу до ресурсу: https://react-redux.js.org/introduction/basic-tutorial
13. Redux Actions. Режим доступу до ресурсу: https://redux.js.org/basics/actions
14. GraphQL. Режим доступу до ресурсу: http://spec.graphql.org/draft/
15. Async/Await. Режим доступу до ресурсу: https://learn.javascript.ru/async-await

16. What are cookies? What are the differences between them?. Режим доступу до ресурсу: https://www.cisco.com/c/en/us/support/docs/security/web-security-appliance/117925-technote-csc-00.html

17. React Cookie. Режим доступу до ресурсу: https://www.npmjs.com/package/react-cookie

18. Validate JS. Режим доступу до ресурсу: https://validatejs.org/

19. Validator. Режим доступу до ресурсу: https://www.npmjs.com/package/validator

20. React Forms. Режим доступу до ресурсу: https://reactjs.org/docs/forms.html

21. Blueprint Icons. Режим доступу до ресурсу: https://blueprintjs.com/docs/#icons

22. Composition vs Inheritance. Режим доступу до ресурсу: https://reactjs.org/docs/composition-vs-inheritance.html

23. Mixins for Rem Font Sizing. Режим доступу до ресурсу: https://css-tricks.com/snippets/css/less-mixin-for-rem-font-sizing/

24. Understanding and Using rem Units in CSS. Режим доступу до ресурсу: https://www.sitepoint.com/understanding-and-using-rem-units-in-css/

25. SendGrid. Режим доступу до ресурсу: https://app.sendgrid.com/

26. SQL. Режим доступу до ресурсу: https://www.dofactory.com/sql

27. Stack Overflow “The “amdefine” cannot load a custom module”. Режим доступу до ресурсу: https://stackoverflow.com/questions/61703332/the-amdefine-can-not-load-a-custom-module/61704852#61704852

28. Universal cookie. Режим доступу до ресурсу: https://www.npmjs.com/package/universal-cookie

29. How to use GraphQL enum and its best practices. Режим доступу до ресурсу: https://atheros.ai/blog/how-to-use-graphql-enum-type-and-its-best-practices

30. Zxcvbn. Режим доступу до ресурсу: https://www.npmjs.com/package/zxcvbn

31. Object-relational mapping. Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Object-relational_mapping

32. What Is PWA? All You Need to Know About Progressive Web Apps. Режим доступу до ресурсу: https://www.simicart.com/blog/what-is-progressive-web-app/

33. File verification. Режим доступу до ресурсу: https://en.wikipedia.org/wiki/File_verification

34. RegExLix. Режим доступу до ресурсу: http://regexlib.com/

35. Stack Overflow “React : difference between <Route exact path=“/” /> and <Route path=“/” />”. Режим доступу до ресурсу: https://stackoverflow.com/questions/49162311/react-difference-between-route-exact-path-and-route-path

36. Moment.js. Режим доступу до ресурсу: https://momentjs.com/

37. i18n. Режим доступу до ресурсу: https://www.i18next.com/overview/configuration-options

38. 15 React Best Practices You Need to Follow in 2020. Режим доступу до ресурсу: https://www.codeinwp.com/blog/react-best-practices/

39. React carousel. Режим доступу до ресурсу: https://openbase.io/js/@brainhubeu/react-carousel

40. React PDF. Режим доступу до ресурсу: https://react-pdf.org/rendering-process

41. React Select. Режим доступу до ресурсу: https://react-select.com/home

42. LESS. Режим доступу до ресурсу: http://lesscss.org/usage/

43. Grunt. Режим доступу до ресурсу: https://gruntjs.com/

44. ESLint. Режим доступу до ресурсу: https://eslint.org/

45. Prettier. Режим доступу до ресурсу: https://prettier.io/

46. MySQL. Режим доступу до ресурсу: https://www.mysql.com/

ДОДАТКИ

Додаток А

Фрагмент лістингу серверної частини

User.js

```
if (typeof define !== 'function') {
    var define = require('amdefine')(module, require('requirejs'));
}

define([
    'underscore',
    'sequelize',
    'app/database',
    'model/password',
    'model/role',
    'model/data'
], (
    _,
    Sequelize,
    sequelize,
    Password,
    RoleObject,
    Data
) => {
    const Operation = Sequelize.Op;

    const { Role, RoleType } = RoleObject;

    /**
     * @model User
     * @description Initialize model and table in database
     */

    class User extends Sequelize.Model {

        /**
         * Method create a new User entity.
         *
         * @param {{username:string, password:string, email:string, role:string}} userInput
         * @return Promise<<User>>
         */
        static async new (userInput) {
            const {
                username,
                password,
                email,
                role,
                data_id
            } = userInput;

            const isExist = await this.isExist({ username, email });

            if (isExist) {
                throw new Error(`The user is already exist!`);
            }

            const encryptedPassword = Password.encrypt(password);

            const entityRole = await Role.get({ name: role });
```

```
    return this.create({
        email: email,
        username: username,
        hash: encryptedPassword.hash,
        salt: encryptedPassword.salt,
        is_email_confirmed: false,
        role_id: entityRole.id,
        data_id: data_id || null
    });
}

// TODO: Need refactoring

/**
 * Method returns the user object {key|username|email}.
 *
 * @param {{key:string?, username:string?, email:string?}} userInput
 * @return Promise<object|null>
 */
static async get (userInput) {
    let {
        key,
        username,
        email
    } = userInput;

    key = key || null;
    email = email || null;
    username = username || null;

    let entity;

    if (this.name === 'User') {

        entity = await this.findOne({
            where: { [Operation.or]: [ { key: key }, { username: username }, { email: email } ] }
        });

        if (_.isNull(entity)) {
            return null;
        }

        return entity;

    } else {

        entity = await this.findOne({
            include: [ {
                model: User,
                where: { [Operation.or]: [ { key: key }, { username: username }, { email: email } ] }
            } ]
        });

        if (_.isNull(entity)) {
            return null;
        }

        const user = await User.findByPk(entity.user_id);

        // TODO: Return object without sequelize methods
```

```
            return Object.assign({}, user.dataValues, entity.dataValues);
        }
    }

    /**
     * @param {object} user
     * @param {object} userInfo
     * @return {Promise<any>}
     */
    static async resave (user, userInfo) {

        if (!user || !userInfo) {
            throw new Error('The parameters are incorrect!');
        }

        user.update(userInfo);

        return await user.save();
    }

    /**
     * Method for check existing of user
     *
     * @param {{email:string?, username:string?, key:string?}} data
     * @return {Promise<boolean>}
     */
    static async isExist (data) {
        const entity = await this.get(data);

        return !!entity;
    }
}

User.init({
    id: {
        type: Sequelize.INTEGER,
        autoIncrement: true,
        allowNull: false,
        primaryKey: true
    },
    key: {
        type: Sequelize.STRING,
        allowNull: false
    },
    email: {
        type: Sequelize.STRING,
        allowNull: true
    },
    username: {
        type: Sequelize.STRING,
        allowNull: false
    },
    hash: {
        type: Sequelize.STRING,
        allowNull: false
    },
    salt: {
        type: Sequelize.STRING,
        allowNull: false
    }
}, { sequelize, modelName: 'User', tableName: 'Users', timestamps: false, underscored: true });
```

```
    User.belongsTo(Role, { foreignKey: 'role_id' });

    User.belongsTo(Data, { foreignKey: 'data_id' });

    return User;
});
```

Auth.js

```
if (typeof define !== 'function') {
    var define = require('amdefine')(module, require('requirejs'));
}

define([
    'underscore',
    'model/user',
    'model/role',
    'model/session',
    'model/password'
], (
    _,
    User,
    RoleObject,
    Session,
    Password
) => {

    const { Role, RoleType } = RoleObject;

    /**
     * @model Authentication
     * @description Initialize model
     */

    class Auth extends User {

        /**
         * @param {{login:string!, password:string!}} authenticationInput
         * @return {Promise<{key: *, token: string | {allowNull: boolean, type} | null | string}>}
         */
        static async authenticate (authenticationInput) {
            const {
                login,
                password
            } = authenticationInput;

            const isExist = await User.isExist({ username: login, email: login });

            if (!isExist) {
                throw new Error(`User is not found!`);
            }

            const user = await User.get({ username: login, email: login });

            if (
                !Password.validate(password, user.salt, user.hash)
            ) {
                throw new Error('The password is wrong!');
            }

            const oldSessions = await Session.findAll({
                where: { user_id: user.id }
```

```
        });

        if (!_.isEmpty(oldSessions)) {
            await Session.destroy({ where: { user_id: user.id } });
        }

        const session = await Session.create({
            sessionToken: Session.generateToken(), user_id: user.id
        });

        return {
            key: user.key,
            sessionToken: session.sessionToken
        };
    }

    /**
     * @param {string} sessionToken
     * @param {{key:string?, entity:object?}} data
     * @return {Promise<boolean>}
     */
    static async isAuthenticated (sessionToken, { key, entity }) {
        entity = entity || await User.get({ key });

        if (_.isNull(entity)) {
            throw new Error('User does not exist!');
        }

        return await Session.validateTokenByUserKey(sessionToken, { key, entity });
    }

    /**
     * @param {string} role
     * @param {string} sessionToken
     * @param {{key:string?, entity:object?}} data
     * @return {Promise<boolean>}
     */
    static async isAuthorized (role, sessionToken, { key, entity }) {
        entity = entity || await User.get({ key });

        if (_.isNull(entity)) {
            throw new Error('User does not exist!');
        }

        const isAuthenticated = await this.isAuthenticated(sessionToken, { entity: entity });

        if (isAuthenticated) {
            const entityRole = await Role.getById(entity.role_id);

            if (_.isNull(entityRole)) {
                throw new Error('Role does not exist!');
            }

            return role === entityRole.name;
        }

        return false;
    }
}

return Auth;
});
```

Email.js

```
if (typeof define !== 'function') {
    var define = require('amdefine')(module, require('requirejs'));
}

define([
    'underscore',
    'uniqid',
    '@sendgrid/mail',
    'app/sendgrid',
    'sequelize',
    'app/database',
    'model/user'
], (
    _,
    uniqid,
    SendGrid,
    SendGridSettings,
    Sequelize,
    sequelize,
    User
) => {

    SendGrid.setApiKey(
        SendGridSettings.key
    );

    /**
     * @model Email
     * @description Initialize model
     */

    class Email extends Sequelize.Model {

        /**
         * Checking confirmation of email.
         *
         * @param {string} key
         * @return {boolean}
         */
        static async isConfirmed (key) {

            if (!key) {
                throw new Error('The params are empty!');
            }

            const userInstance = await User.get({ key });

            const emailInstance = await this.findOne({
                where: { user_id: userInstance.id }
            });

            if (!emailInstance) {
                return false;
            }

            return emailInstance.is_confirmed;
        }

        /**
         * @param {string} to
```

```
     * @param {string} subject
     * @param {string} text
     * @param {string} html
     * @return {Promise<[, {}]>}
     */
    static async send (to, subject, text, html) {

        if (!to || !subject) {
            throw new Error('Params are incorrect');
        }

        const from = SendGridSettings.email;

        const emailBody = {
            to: to,
            from: from,
            subject: subject,
            text: text,
            html: html
        };

        try {
            return await SendGrid.send(emailBody);
        } catch (e) {
            throw new Error(e);
        }
    }

    /**
     * @param {object} user
     * @return {Promise<{}>}
     */
    static async sendConfirmationEmail (user) {
        const { id, username, email } = user;

        const max = 999999, min = 100000;

        if (!id || !username || !email) {
            throw new Error('Params are incorrect');
        }

        const randomValue = Math.floor(Math.random() * (max - min + 1)) + max;

        const confirmation = {
            code: randomValue,
            to: email,
            subject: 'Confirmation email',
            text: 'Confirmation email'
        };

        // TODO: HTML
        const confirmationTemplate = `
            <h3>Hello ${ username }.</h3>
            <p>You've created an account in the Information platform of Zhytomyr Polytechnic State University.</p>
            <p>Confirm your email with this code: <b>${ confirmation.code }</b>.</p>
        `;

        await this.destroy({
            where: { user_id: id }
        });

        await this.create({
```

```
            code: confirmation.code,
            user_id: id
        });

        return await this.send(
            confirmation.to,
            confirmation.subject,
            confirmation.text,
            confirmationTemplate
        );
    }

    /**
     * @param {object} user
     * @return {Promise<{}>}
     */
    static async sendNotificationEmail (user) {
        const { id, username, email } = user;

        if (!id || !username || !email) {
            throw new Error('Params are incorrect');
        }

        const confirmation = {
            to: email,
            subject: 'Notification email',
            text: 'Notification email'
        };

        const confirmationTemplate = `
            <h3>Hello ${ username }.</h3>
            <p>You were assigned to review the Individual Scheduling.</p>
            <p>Confirm your email with this code: <b>${ confirmation.code }</b>.</p>
        `;

        return await this.send(
            confirmation.to,
            confirmation.subject,
            confirmation.text,
            confirmationTemplate
        );
    }
}

Email.init({
    id: {
        type: Sequelize.INTEGER,
        autoIncrement: true,
        allowNull: false,
        primaryKey: true
    },
    key: {
        type: Sequelize.STRING,
        allowNull: false
    },
    code: {
        type: Sequelize.STRING,
        allowNull: false
    },
    is_confirmed: {
        type: Sequelize.BOOLEAN,
        allowNull: false,
```

```
            defaultValue: false
        }
    }, { sequelize, modelName: 'Email', tableName: 'Email_Confirmations', timestamps: false, underscored: true });

    Email.belongsTo(User, { foreignKey: 'user_id' });

    return Email;
});
```

Express.js

```
if (typeof define !== 'function') {
    var define = require('amdefine')(module, require('requirejs'));
}

define([
    'model/auth'
], (
    Auth,
) => {

    class Express {

        /**
         * @EXPRESS
         * @QUERY
         *
         * @param {object} request
         * @param {object} response
         * @param {function} next
         * @return {Promise<*|boolean>}
         */
        static async isAuthenticated (request, response, next) {
            const { key, sessionToken } = request.query;

            if (!key || !sessionToken) {
                response.status(400).send('The parameters are incorrect!');
            }

            try {
                const result = await Auth.isAuthenticated(sessionToken, { key: key });

                !result ? response.status(403).send('You don\'t have all rights to this request.') : next();
            } catch (e) {
                response.status(403).send(e.message);
            }
        }
    }

    return Express;
});
```

File.js

```
if (typeof define !== 'function') {
    var define = require('amdefine')(module, require('requirejs'));
}

define([
    'fs',
    'path',
    'crypto',
```

```
    'underscore',
    'model/email',
    'model/user',
    'model/student',
    'app/resolver',
    'app/settings',
    'model/company'
], (
    fs,
    path,
    crypto,
    _,
    Email,
    User,
    Student,
    Resolver,
    settings,
    Company
) => {

    const FileType = {
        STUDENT_CARD: 'student_card',
        CERTIFICATE_OF_EMPLOYMENT: 'certificate_of_employment'
    }

    class File {

        /**
         * This resolver using in app.js
         *
         * @param {object} request
         * @param {object} response
         * @return {Promise<*>}
         */
        static async upload (request, response) {

            // TODO: Security

            const allowedMimeType = [
                'image/png', 'image/jpg', 'image/jpeg'
            ];

            const { key, type } = request.body;

            const { company_key } = _.has(request.body, 'data') && JSON.parse(request.body.data);

            if (!request.files || Object.keys(request.files).length === 0) {
                return response.status(400).send('No files were uploaded.');
            }

            const file = request.files.image;

            if (!allowedMimeType.includes(file.mimetype)) {
                return response.status(403).send(`Not allowed mimetype ${ file.mimetype }.`);
            }

            const mimeType = file.mimetype.replace('image/', '');

            const isExist = await User.isExist({ key: key || '' });

            if (!isExist) {
                return response.status(404).send(`User with ${ key } not found.`);
```

```
    }

    let prefix = 'img';

    switch (type) {
      case FileType.STUDENT_CARD: {
        prefix = 'student_card';
        break;
      }
      case FileType.CERTIFICATE_OF_EMPLOYMENT: {
        prefix = 'certificate_of_employment';
        break;
      }
    }

    const name = `${ key }-${ prefix }`;

    const fileName = `${ name }.${ mimeType }`;

    const basePath = path.join(settings.PROJECT_DIR, '/media/uploads/');

    const uploadPath = path.join(basePath, fileName);

    File.clear(name, basePath, allowedMimeType);

    file.mv(uploadPath, async (err) => {
      if (err) {
        return response.status(500).send(err);
      }

      let uploadResult;

      switch (type) {
        case FileType.STUDENT_CARD: {
          uploadResult = await Student.studentCardUpload(key, { name: fileName, basePath });
          break;
        }
        case FileType.CERTIFICATE_OF_EMPLOYMENT: {
          uploadResult = await Company.certificateUpload(company_key, { name: fileName, basePath });
          break;
        }
      }

      if (uploadResult) {
        response.status(200).send(`File successfully uploaded.`);
      } else {
        fs.unlink(uploadPath);

        response.status(500).send('File wasn\'t upload.');
      }
    });
  }

  /**
   * @param {string} fileName - File name without mimetype
   * @param basePath
   * @param allowedMimeTypes
   */
  static clear (fileName, basePath, allowedMimeTypes) {

    allowedMimeTypes.forEach(mimeType => {
      mimeType = mimeType.replace('image/', '');
```

```
            const uploadedFile = path.join(basePath, `${ fileName }.${ mimeType }`);

            const isFileExist = fs.existsSync(uploadedFile);

            if (isFileExist) {
                fs.unlink(uploadedFile, (error => error && new Error(error.message)));
            }
        });
    }
}

return File;
});
```

Password.js

```
if (typeof define !== 'function') {
    var define = require('amdefine')(module, require('requirejs'));
}

define([
    'underscore',
    'crypto',
    'app/encryption'
], (
    _,
    crypto,
    encryptionSettings
)=> {

    /**
     * @model Password
     * @description Initialize model and table in database
     */

    class Password {

        /**
         * @param {string} password
         *
         * Method to set salt and hash the password for a user setPassword method first creates a salt
         * unique for every user then it hashes the salt with user password and creates a hash this
         * hash is stored in the database as user password.
         *
         * @return {{salt:string, hash:string}}
         */
        static encrypt (password) {
            const {
                iterations,
                keylen,
                digest,
                encoding,
                randomBytes
            } = encryptionSettings;

            let salt = crypto.randomBytes(
                randomBytes.size
            ).toString(encoding);

            let hash = crypto.pbkdf2Sync(
                password, salt, iterations, keylen, digest
```

```
            ).toString(encoding);

            return {
                salt: salt,
                hash: hash
            };
        }

        /**
         * @param {string} password
         * @param {string} passwordSalt
         * @param {string} passwordHash
         *
         * Method to check the entered password is correct or not valid password method checks whether the user
         * password is correct or not. It takes the user password from the request and salt from user database
         * entry. It then hashes user password and salt then checks if this generated hash is equal to user's
         * hash in the database or not. If the user's hash is equal to generated hash then the password
         * is correct otherwise not.
         *
         * @return {boolean}
         */
        static validate (password, passwordSalt, passwordHash) {
            const {
                iterations,
                keylen,
                digest,
                encoding
            } = encryptionSettings;

            if (!password || !passwordSalt || !passwordHash) {
                return false;
            }

            const hash = crypto.pbkdf2Sync(
                password, passwordSalt, iterations, keylen, digest
            );

            return passwordHash === hash.toString(encoding);
        }
    }

    return Password;
});
```

Role.js

```
if (typeof define !== 'function') {
    var define = require('amdefine')(module, require('requirejs'));
}

define([
    'underscore',
    'sequelize',
    'app/database'
], (
    _,
    Sequelize,
    sequelize
) => {

    /**
     * @type {{ADMIN: string, USER: string, MODER: string}}
```

```
 */

const RoleType = {
    USER: "User",
    MODER: "Moder",
    ADMIN: "Admin"
};

/**
 * @model Role
 * @description Initialize model and table in database
 */

class Role extends Sequelize.Model {

    /**
     * @param {string} name
     */
    static async new (name) {
        if (!name) {
            throw new Error('Type of role is undefined!');
        }

        const isExist = await this.isExist({ name });

        if (!isExist) {
            return this.create({ name: name });
        }

        return this.get({ name });
    }

    /**
     * @param {{name:string!}} data
     */
    static async get ({ name }) {
        if (!name) {
            throw new Error('Type of role is undefined!');
        }

        let role = await this.findOne({
            where: { name: name }
        });

        if (!role) {
            role = await this.new(name);
        }

        return role;
    }

    /**
     * @param {number|string} id
     */
    static async getById (id) {
        return this.findByPk(id);
    }

    /**
     * @param {{name:string!}} data
     */
    static async isExist ({ name }) {
```

```
            if (!name) {
                throw new Error('Incorrect property!');
            }

            let role = await this.findOne({
                where: { name: name }
            });

            return !_.isNull(role);
        }
    }

    Role.init({
        id: {
            type: Sequelize.INTEGER,
            autoIncrement: true,
            allowNull: false,
            primaryKey: true
        },
        key: {
            type: Sequelize.STRING,
            allowNull: false
        },
        name: {
            type: Sequelize.STRING,
            allowNull: false
        }
    }, { sequelize, modelName: 'Role', tableName: 'Roles', timestamps: false, underscored: true });

    return { Role, RoleType };
});
```

Session.js

```
if (typeof define !== 'function') {
    var define = require('amdefine')(module, require('requirejs'));
}

define([
    'underscore',
    'crypto',
    'sequelize',
    'app/database',
    'app/encryption',
    'model/user'
], (
    _,
    crypto,
    Sequelize,
    sequelize,
    encryptionSettings,
    User
) => {

    /**
     * @model Session
     * @description Initialize model and table in database
     */

    class Session extends Sequelize.Model {

        /**
         * Generates cryptographically strong pseudo-random data. The size
```

```
     * argument is a number indicating the number of bytes to generate.
     *
     * @return {string}
     */
    static generateToken () {
        const {
            encoding,
            randomBytes
        } = encryptionSettings;

        return crypto.randomBytes(
            randomBytes.size
        ).toString(encoding);
    }

    /**
     * Validate session token by user key.
     *
     * @param {string} sessionToken
     * @param {{key:string?, entity:object?}} data
     * @return {Promise<boolean>}
     */
    static async validateTokenByUserKey (sessionToken, { key, entity }) {
        try {
            if (!sessionToken || !key) {
                new Error('Key or token can\'t be undefined');
            }

            const user = entity || await User.get({ key });

            return await this.validateTokenByUserId(sessionToken, user.id);
        } catch (e) {
            throw new Error(e);
        }
    }

    /**
     * Validate session token by user id.
     *
     * @param {string} sessionToken
     * @param {string} id
     * @return {Promise<boolean>}
     */
    static async validateTokenByUserId (sessionToken, id) {
        try {
            if (!sessionToken || !id) {
                new Error('Key or token cannot be undefined');
            }

            const session = await this.findOne({
                where: { user_id: id }
            });

            return sessionToken === session.sessionToken;
        } catch (e) {
            throw new Error(e);
        }
    }
}

Session.init({
    id: {
```

```
            type: Sequelize.INTEGER,
            autoIncrement: true,
            allowNull: false,
            primaryKey: true
        },
        sessionToken: {
            type: Sequelize.STRING,
            allowNull: false
        }
    }, { sequelize, modelName: 'Session', tableName: 'Sessions', timestamps: false, underscored: true });

    Session.belongsTo(User, { foreignKey: 'user_id' });

    return Session;
});
```

Data.js

```
if (typeof define !== 'function') {
    var define = require('amdefine')(module, require('requirejs'));
}

define([
    'underscore',
    'sequelize',
    'app/database'
], (
    _,
    Sequelize,
    sequelize
) => {

    /**
     * @model Data
     * @description Initialize model and table in database
     */

    class Data extends Sequelize.Model {

        /**
         * @param {object} user
         * @param {object} userInfo
         * @return {Promise<any>}
         */
        static async new (user, userInfo) {

            if (!user || !userInfo) {
                throw new Error('The parameters are incorrect!');
            }

            const data = await this.create(userInfo);

            if (!data) {
                throw new Error('Data is undefined!');
            }

            user.update({ data_id: data.id });

            await user.save();

            return data;
        }
```

```
    /**
     * @param {object} user
     * @return {Promise<string|null>}
     */
    static async getFullName (user) {

        if (!user) {
            throw new Error('The parameters are incorrect!');
        }

        const { data_id } = user;

        const data = await this.getById(data_id);

        if (!data) {
            return null;
        }

        const { first_name, second_name, last_name } = data;

        return `${ second_name } ${ first_name } ${ last_name }`;
    }

    /**
     * @param {object} user
     * @param {object} userInfo
     * @return {Promise<any>}
     */
    static async resave (user, userInfo) {

        if (!user || !userInfo) {
            throw new Error('The parameters are incorrect!');
        }

        const { data_id } = user;

        const data = await this.getById(data_id);

        if (!data) {
            return await this.new(user, userInfo);
        }

        data.update(userInfo);

        return await data.save();
    }

    /**
     * @param {number|string} id
     */
    static async getById (id) {
        return this.findByPk(id);
    }
}

Data.init({
    id: {
        type: Sequelize.INTEGER,
        autoIncrement: true,
        allowNull: false,
        primaryKey: true
```

```
},
key: {
   type: Sequelize.STRING,
   allowNull: false
},
gender: {
   type: Sequelize.BOOLEAN,
   allowNull: true,
   validate: {
      notEmpty: true
   }
},
first_name: {
   type: Sequelize.STRING,
   allowNull: true,
   validate: {
      notEmpty: true
   }
},
second_name: {
   type: Sequelize.STRING,
   allowNull: true,
   validate: {
      notEmpty: true
   }
},
last_name: {
   type: Sequelize.STRING,
   allowNull: true,
   validate: {
      notEmpty: true
   }
},
date_of_birth: {
   type: Sequelize.STRING,
   allowNull: true,
   validate: {
      notEmpty: true
   }
},
country: {
   type: Sequelize.STRING,
   allowNull: true,
   validate: {
      notEmpty: true
   }
},
city: {
   type: Sequelize.STRING,
   allowNull: true,
   validate: {
      notEmpty: true
   }
},
street: {
   type: Sequelize.STRING,
   allowNull: true,
   validate: {
      notEmpty: true
   }
},
phone_number: {
```

```
            type: Sequelize.STRING,
            allowNull: true,
            validate: {
               notEmpty: true
            }
         }
      }, { sequelize, modelName: 'Data', tableName: 'Data', timestamps: false, underscored: true });

      return Data;
   });
```

Додаток Б

Фрагмент лістингу клієнт частини

ConfirmationEmail.js

```
import React, { useState } from 'react';
import { connect } from 'react-redux';
import { Classes, Intent, Dialog } from '@blueprintjs/core';
import Logo from '../../web/media/logo-ua.svg';
import useTranslate from '../../hooks/useTranslate';
import { Input, InputFor } from '../Input';
import { Button, ButtonType } from '../Button';
import { GlobalMessageType } from '../GlobalMessage';
import * as actions from '../../services/store/actions';
import { Email } from '../../actions/Email';

/**
 * Send confirmation email request
 *
 * @param {object} props
 * @param {string} key
 * @param {string} code
 * @param {{setRequestState:function, setIsOpen:function}} states
 * @return {boolean}
 */
const confirmEmail = (props, key, code, states) => {

    Email.confirm(key, code).then(response => {
        const [ { data, error }, { setRequestState, setIsOpen } ] = [ response, states ];

        if (data) {
            setIsOpen(false);

            props.globalMessageShow('Email has been confirmed!', GlobalMessageType.SUCCESS);

            return setRequestState(true);
        }

        props.globalMessageShow(error, GlobalMessageType.WARNING);
    });
```

```
};

/**
 * @param {object} props
 * @param props {function} setIsOpen
 * @param props {boolean} isOpen
 * @param props {string} email
 * @return {*}
 */
const ConfirmEmail = (props) => {

    const { userKey, userEmail, isOpen, setIsOpen, setRequestState } = props;

    const [ code, setCode ] = useState('');

    const [ isValid, setIsValid ] = useState(false);

    const [ titleText, enterSecurityCodeText, contentText ] = [
        useTranslate('Confirm your email'), useTranslate('Enter security code'),
        useTranslate('A security code has been sent to your email address at')
    ];

    const submitHandler = (e) => {
        confirmEmail(props, userKey, code, { setRequestState, setIsOpen });

        return e.preventDefault();
    };

    return (
        <div className="dialog confirmation-email">
            <Dialog
                isOpen={ isOpen }
                autoFocus={ true }
                enforceFocus={ true }
                usePortal={ true }
                isCloseButtonShown={ false }
                className="dialog-confirmation-email"
            >
                <div className={ Classes.DIALOG_BODY }>
                    <div className="head">
                        <img className="logo" src={ Logo } alt="Logo" width="auto" height="auto" />
```

```
                <h2 className="title">{ titleText }</h2>
              </div>
              <div className="content">
                <b>{ enterSecurityCodeText }</b>
                <p>{ contentText.replace('%1', userEmail) }</p>
              </div>
              <Input type="text" inputFor={ InputFor.TEXT } label="Security code"
                  placeholder="Security code" isValid={ setIsValid }
                  onChangeHandler={ event => setCode(event.target.value) } />
            </div>
            <div className={ Classes.DIALOG_FOOTER }>
              <Button
                intent={ Intent.PRIMARY }
                type={ ButtonType.SUBMIT }
                disabled={ !isValid }
                buttonName={ useTranslate('Confirm') }
                onClickHandler={ submitHandler }
              />
              <Button
                intent={ Intent.NONE }
                type={ ButtonType.BUTTON }
                buttonName={ useTranslate('Confirm later') }
                onClickHandler={ () => {
                  setIsOpen(!isOpen);
                  setRequestState(true);
                } }
              />
            </div>
          </Dialog>
        </div>
    );
};

/**
 * @inheritDoc
 */
const mapDispatchToProps = dispatch => {
    return {
        globalMessageShow: (message, messageType) => dispatch(actions.globalMessageShow(message, messageType))
    };
};
```

```
export default connect(null, mapDispatchToProps)(ConfirmEmail);
```

Data.js

```
import React, { useEffect } from 'react';
import * as actions from '../../services/store/actions';
import { connect } from 'react-redux';
import { GlobalMessageType } from '../GlobalMessage';
import { University } from '../../actions/University/University';
import _ from 'underscore';
import { isError } from '../../services/utils';

const Data = (props) => {

    const [ { key, sessionToken }, { type } ] = [ props.userAuth, props.userInfo ];

    const {
        isNeedToUpdateFaculties,
        isNeedToUpdateSpecialties,
        isNeedToUpdateCourses,
        isNeedToUpdateTeachers,
        isNeedToUpdateAssessments,
        isNeedToUpdateAssessmentTypes
    } = props.storageData;

    const {
        faculty,
        facultyKey,
        specialty,
        specialtyKey,
        course,
        courseKey
    } = props.storageData;

    useEffect(() => {

        if (props.isAuthenticated) {

            const baseUserInput = { key, sessionToken, type };

            if (isNeedToUpdateFaculties) {
```

```
    University.getFaculties(baseUserInput).then(response => {
      const { data, error } = response;

      if (isError(response)) {
        return props.globalMessageShow(error, GlobalMessageType.WARNING);
      }

      props.setFaculties(data);
    });
  }

  if (isNeedToUpdateSpecialties) {
    const { facultyKey } = props.storageData;

    if (_.isNull(facultyKey) || _.isEmpty(facultyKey)) {
      return;
    }

    University.getSpecialties({ ...baseUserInput, facultyKey }).then(response => {
      const { data, error } = response;

      if (isError(response)) {
        return props.globalMessageShow(error, GlobalMessageType.WARNING);
      }

      props.setSpecialties(data);
    });
  }

  if (isNeedToUpdateCourses) {
    University.getCourses(baseUserInput).then(response => {
      const { data, error } = response;

      if (isError(response)) {
        return props.globalMessageShow(error, GlobalMessageType.WARNING);
      }

      props.setCourses(data);
    });
  }
```

```
        if (isNeedToUpdateTeachers) {
            University.getTeachers(baseUserInput).then(response => {
                const { data, error } = response;

                if (isError(response)) {
                    return props.globalMessageShow(error, GlobalMessageType.WARNING);
                }

                props.setTeachers(data);
            });
        }

        if (isNeedToUpdateAssessments) {
            University.getAssessments(baseUserInput).then(response => {
                const { data, error } = response;

                if (isError(response)) {
                    return props.globalMessageShow(error, GlobalMessageType.WARNING);
                }

                props.setAssessments(data);
            });
        }

        if (isNeedToUpdateAssessmentTypes) {
            University.getAssessmentTypes(baseUserInput).then(response => {
                const { data, error } = response;

                if (isError(response)) {
                    return props.globalMessageShow(error, GlobalMessageType.WARNING);
                }

                props.setAssessmentTypes(data);
            });
        }
    }

}, [
    isNeedToUpdateFaculties,
    isNeedToUpdateSpecialties,
    isNeedToUpdateCourses,
```

```
    isNeedToUpdateTeachers,
    isNeedToUpdateAssessments,
    isNeedToUpdateAssessmentTypes
]);

useEffect(() => {

    if (props.isAuthenticated) {

        const baseUserInput = { key, sessionToken, type };

        if (!_.isEmpty(facultyKey) && _.isEmpty(faculty)) {
            University.getFaculty({ ...baseUserInput, facultyKey }).then(response => {
                const { data, error } = response;

                if (isError(response)) {
                    return props.globalMessageShow(error, GlobalMessageType.WARNING);
                }

                props.setFaculty(data);
            });
        }

        if (!_.isEmpty(specialtyKey) && _.isEmpty(specialty)) {

            University.getSpecialty({ ...baseUserInput, specialtyKey }).then(response => {
                const { data, error } = response;

                if (isError(response)) {
                    return props.globalMessageShow(error, GlobalMessageType.WARNING);
                }

                props.setSpecialty(data);
            });
        }

        if (!_.isEmpty(courseKey) && _.isEmpty(course)) {
            University.getCourse({ ...baseUserInput, courseKey }).then(response => {
                const { data, error } = response;

                if (isError(response)) {
```

```
                    return props.globalMessageShow(error, GlobalMessageType.WARNING);
                }

                props.setCourse(data);
            });
        }
    }
  }, [ facultyKey, specialtyKey, courseKey ])

  return null;
};

/**
 * @inheritDoc
 */
const mapStateToProps = state => {
  return {
    userAuth: state.auth,
    userInfo: state.user,
    storageData: state.data,
    isAuthenticated: state.auth.isAuthenticated
  };
};

/**
 * @inheritDoc
 */
const mapDispatchToProps = dispatch => {
  return {
    setFaculty: (faculty) => dispatch(actions.setFaculty(faculty)),
    setSpecialty: (specialty) => dispatch(actions.setSpecialty(specialty)),
    setCourse: (course) => dispatch(actions.setCourse(course)),
    setFaculties: (faculties) => dispatch(actions.setFaculties(faculties)),
    setSpecialties: (specialties) => dispatch(actions.setSpecialties(specialties)),
    setCourses: (courses) => dispatch(actions.setCourses(courses)),
    setTeachers: (teachers) => dispatch(actions.setTeachers(teachers)),
    setAssessments: (assessments) => dispatch(actions.setAssessments(assessments)),
    setAssessmentTypes: (assessmentTypes) => dispatch(actions.setAssessmentTypes(assessmentTypes)),
    globalMessageShow: (message, messageType) => dispatch(actions.globalMessageShow(message, messageType))
  };
};
```

```
export default connect(mapStateToProps, mapDispatchToProps)(Data);
```

GlobalMessage.js

```
import React, { useEffect } from 'react';
import useTranslate from '../../hooks/useTranslate';
import { connect } from 'react-redux';
import { Icon as CoreIcon } from '@blueprintjs/core';
import { IconNames } from '@blueprintjs/icons';
import * as actions from '../../services/store/actions';
import _ from 'underscore';

export const GlobalMessageType = {
   SUCCESS: 'SUCCESS',
   WARNING: 'WARNING',
   ERROR: 'ERROR',
   INFO: 'INFO'
};

const Icon = (props) => {

   const IconType = {
      [GlobalMessageType.SUCCESS]: IconNames.TICK,
      [GlobalMessageType.INFO]: IconNames.INFO_SIGN,
      [GlobalMessageType.WARNING]: IconNames.WARNING_SIGN,
      [GlobalMessageType.ERROR]: IconNames.ERROR
   };

   return (
      <span className="icon">
         <CoreIcon icon={ IconType[props.globalMessage.messageType] } iconSize={ CoreIcon.SIZE_LARGE } />
      </span>
   );
};

const Message = (props) => {
   return (
      <span className="message">
         { useTranslate(props.globalMessage.message) }
      </span>
```

```
    );
};

const GlobalMessage = (props) => {
    const { globalMessage } = props;

    useEffect(() => {
        setTimeout(() => {
            !_.isNull(props.globalMessage.message) && props.globalMessageHide();
        }, 5000);
    });

    if (_.isNull(props.globalMessage.message)) {
        return null;
    }

    return (
        <div className="global-message" data-message-type={ globalMessage.messageType }>
            <Icon { ...props } />
            <Message { ...props } />
        </div>
    );
};

/**
 * @inheritDoc
 */
const mapStateToProps = state => {
    return {
        globalMessage: state.action
    };
};

/**
 * @inheritDoc
 */
const mapDispatchToProps = dispatch => {
    return {
        globalMessageHide: () => dispatch(actions.globalMessageHide()),
    };
};
```

```
export default connect(mapStateToProps, mapDispatchToProps)(GlobalMessage);
```

PasswordStength.js

```
import React from 'react';
import * as PasswordStrengthApi from 'zxcvbn';
import { ProgressBar, Intent } from '@blueprintjs/core';
import useTranslate from '../../hooks/useTranslate';

/**
 * @param {object} props
 * @return {*}
 */
export const PasswordStrength = (props) => {
    const { password } = props;

    const [ Weak, Fair, Good, Strong, Strength ] = [
        useTranslate('Weak'), useTranslate('Fair'),
        useTranslate('Good'), useTranslate('Strong'),
        useTranslate('Password strength')
    ];

    const passwordLabel = [ Weak, Weak, Fair, Good, Strong ];

    const passwordStrength = PasswordStrengthApi(password)['score'];

    return (
        <div className="password-strength-meter">
            <div className="progress-bar" data-value={ passwordStrength }>
                <ProgressBar
                    intent={ Intent.NONE } stripes={ false }
                    value={ passwordStrength / 4 }
                />
            </div>
            <label className="password-strength-meter-label">
                { password && (
                    <span>
                        <span className="label">{ Strength }</span>
                        <span className="value">{ passwordLabel[passwordStrength] }</span>
                    </span>
```

```
            ) }
          </label>
        </div>
    );
};
```