МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«_____» _____ 2023р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНИЙ ПРОЄКТ, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ **«БАКАЛАВР»**
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ»

Тема: «Веб-кабінет студента Національного Авіаційного Університету»

Виконавець: студент групи УС-412Б Струк Марія Віталіївна

Керівник: к.т.н., доцент Єгоров Сергій Вікторович

Нормоконтролер: ст. викл. Олександр ШЕВЧЕНКО

Київ – 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 «Інформаційні технології», 122 «Комп'ютерні науки», «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«____»_____2023р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студентки

Струк Марії Віталіївни
(прізвище, ім'я, по батькові)

1. Тема роботи: «Веб-кабінет студента Національного Авіаційного Університету» затверджена наказом ректора від «01» травня 2023 р. за № 623/ст.

2. Термін виконання роботи 15.05.2023 – 25.06.2023

3. Вихідні дані до роботи: інтегроване середовище розробки *Visual Studio Code*, мова *JavaScript*, фреймворк *Vue.js*, система управління базами даних *Firebase*.

4. Зміст пояснювальної записки: вступ, задачі створення веб-кабінету студента, проектування веб-кабінету студента, розробка веб-кабінету студента, висновок.

5. Перелік обов'язкового графічного матеріалу: клієнт-сервер архітектура; структура односторінокового та мультисторінкового додатку; структура файлів проекту; головна сторінка списку навчальних предметів.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Отримання завдання на дипломну роботу, створення плану дипломної роботи та побудова плану-графіку виконання робіт.	15.05.2023 – 17.05.2023	
2.	Розроблення та затвердження календарного плану виконання дипломної роботи.	18.05.2023 – 20.05.2023	
3.	Проведення консультацій з науковим керівником.	21.05.2023 – 23.05.2023	
4.	Огляд та аналіз наукової літератури по темі дипломної роботи та написання Розділу 1.	24.05.2023 –02.06.2023	
5.	Написання Розділу 2 дипломної роботи.	03.06.2023 – 08.06.2023	
6.	Написання Розділу 3 дипломної роботи. Завершення створення пояснювальної записки дипломної роботи.	09.06.2023 – 12.06.2023	
7.	Оформлення та друк пояснювальної записки.	13.06.2023 – 14.06.2023	
8.	Створення презентації, доповіді та підготовка до захисту дипломної роботи.	15.06.2023 – 16.06.2023	
9.	Підготовка матеріалів дипломної роботи для передачі секретарю ДЕК (папка, конверт, диск із файлом диплому, рецензія, відгук).	17.06.2023 – 19.06.2023	

7. Дата видачі завдання: «15» травня 2023 р.

Керівник дипломного проекту _____ Сергій ЄГОРОВ

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Марія СТРУК

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Веб-кабінет студента Національного Авіаційного Університету» складається зі вступу, трьох розділів, висновку, списку бібліографічних посилань і містить 70 сторінок, та 10 рисунків. Список бібліографічних посилань складається з 16 найменувань.

Об'єктом дослідження є: процес аналізу та розробки функцій веб-сервісу «Веб-кабінет студента Національного Авіаційного Університету».

Предметом дослідження є: веб-кабінет студента Національного Авіаційного Університету.

Метою дипломної роботи є: проектування та розробка веб-кабінету студента Національного Авіаційного Університету з метою спростити та поліпшити процеси, що напряму пов’язані зі студентами, їх навантаженням та успішністю, в умовах дистанційного навчання.

Для досягнення мети проектування та розробки «Веб-кабінету студента Національного Авіаційного Університету» та вирішення поставлених завдань було застосовано декілька ***методів дослідження:***

- метод проведення комплексного огляду наявних джерел, інформації;
- метод проведення опитування та інтерв’ю;
- метод порівняння програмних продуктів і платформ;
- метод тестування зручності використання;
- метод створення прототипів.

Результатом дослідження є розроблений «Веб-кабінет студента Національного Авіаційного Університету», який сприяє поліпшенню навчального процесу у вищих навчальних закладах за допомогою веб-технологій, призначений спростити процеси, що напряму пов’язані зі студентами, їх навантаженням та успішністю.

ВСЕСВІТНЯ ПАВУТИНА, ОНЛАЙН-СЕРВІС, ВЕБ-КАБІНЕТ СТУДЕНТА, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, *SPA, HTML, CSS, JAVASCRIPT, FIREBASE.*

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

WEB – *World Wide Web* (всесвітня павутина, всесвітня мережа, інтернет)

HTTP – *Hypertext Transport Protocol* (протокол передачі гіпертексту)

HTML – *Hypertext Markup Language* (мова розмітки гіпертексту)

URI – *Uniform Resource Identifier* (уніфікований ідентифікатор ресурсів)

URL – *Uniform Resource Locator* (адреса ресурсу)

DNS – *Domain Name System* (система доменних імен)

XHTML – *Extensible Hypertext Markup Language* (розширювана мова розмітки гіпертексту)

XML – *Extensible Markup Language* (розширювана мова розмітки)

CSS – *Cascading Style Sheets* (каскадні таблиці стилів)

SPA – *Single Page Application*

API – *Application Programming Interface* (програмний інтерфейс програми)

UI – *User Interface* (призначений для користувача інтерфейс)

DOM – *Document Object Model* (об'єктна модель документа)

ВСТУП

В результаті вдосконалення інформаційних технологій сучасна людина перестала уявляти своє життя без Інтернету. Кожного дня ми користуємося інформацію з всесвітньої мережі, використовуємо електронну пошту, здійснюємо запити на пошук необхідної інформації, дізнаємось про новини та спілкуємося в соціальних мережах з друзями.

У зв’язку з цим виникла можливість поліпшення навчального процесу у вищих навчальних закладах за допомогою веб-технологій. Дізнатися та переглянути більш детальну інформацію про кожну з дисциплін, розклад занять та календар всіх подій в університеті можна в електронному кабінеті студента, який призначений спростити процеси, що напряму пов’язані зі студентами, їх навантаженням та успішністю. Існуючі інструменти веб-розробника, та технології програмування для веб докладно описані в розділах. Всі вони мають свої недоліки та переваги, тому для усунення цього доводиться користуватися декількома технологіями одночасно.

Слід зазначити, що я брала участь у конференції «Інтегровані інтелектуальні робототехнічні комплекси (ІІРТК-2023). Шістнадцята міжнародна науково-практична конференція 23-24 травня 2023 р.», у якій мною була написана стаття на тему «Поліпшення навчального процесу за допомогою веб-сервісів». Участь в конференції зіграла важливу роль у виконанні моєї дипломної роботи та поглибленні моїх знань у вибраній галузі.

Всесвітня павутина (*WEB*) представляє собою систему взаємопов'язаних гіпертекстових документів і ресурсів, доступних через Інтернет. *WEB* використовує стандарти і протоколи, такі як *HTTP*, *HTML* і *URL* для передачі та представлення інформації.

Всесвітня мережа складається зі статичних та динамічних веб-сторінок, які можуть містити текст, зображення, відео, звук та інші мультимедійні елементи. Користувачі можуть взаємодіяти з веб-сторінками, переходячи за посиланнями, заповнюючи форми, відправляючи дані на сервер та виконуючи інші дії.

Веб-додатки є одним з ключових аспектів вебу. Вони використовуються для надання користувачам більш складних функцій та можливостей, які вимагають взаємодії з сервером та обробки даних. Веб-додатки можуть використовувати різні технології, такі як *JavaScript*, серверні мови програмування, бази даних та інші, для розробки функціональності та логіки додатків.

Одним з ключових переваг вебу є його доступність. Користувачі можуть отримати доступ до веб-сторінок та додатків з різних пристроїв, таких як комп'ютери, смартфони, планшети та інші, за допомогою різних веб-браузерів.

Онлайн-сервіс (веб-сервіс) – це сукупність програмно-технічних засобів, інтегрованих з метою ефективного опрацювання веб-ресурсів, які знаходяться у веб-просторі. Онлайн-сервіси можуть бути розроблені для виконання різних завдань, таких як комунікація, обмін інформацією, покупки товарів і послуг, здійснення фінансових операцій, розваги та багато іншого. Веб-сервіси стали надзвичайно популярними в сучасному світі, оскільки вони забезпечують зручність, доступність та глобальну підключеність.

Кожен сервіс має свої переваги та недоліки, тому що їх створює людина, а не комп’ютер. Існуючі веб-кабінети, для надання інформації, яка потрібна під час навчального процесу студенту, дають змогу переглядати картку дисциплін, розклад занять, детальний опис кожного предмету в поточному семестрі та анонси додаткових заходів для студентів. Ось декілька прикладів найпопулярніших веб-кабінетів студента: *Blackboard, Moodle, Canvas* та *Classroom,* які надають змогу проводити навчання в дистанційному форматі.

Невід'ємною частиною створення онлайн-сервісу являється чіткий опис задачі та проектування відповідно до вимог, без цього неможливо створити якісний продукт. Проаналізувавши різні веб-кабінети студента, був спроектований онлайн-сервіс, який має мету максимально спрощувати і автоматизувати процеси, які напряму пов’язані зі студентами.

Сучасний *WEB* дедалі частіше використовує технології *Single Page Application* (*SPA*) для фронтенд розробки за допомогою якого був спроектований онлайн-сервіс.

SPA – це веб-додаток, розміщений на одній веб-сторінці, яка для забезпечення роботи завантажує весь необхідний код разом із завантаженням самої сторінки.

Основне місце в *SPA*-архітектурі при проектуванні веб-кабінету студента займає відображення (*view*) – те, що бачить і з чим взаємодіє користувач. Результатом роботи відображення є звичайний *HTML*, що відображається браузером. *SPA*-додаток використовує *DOM* лише для запису змін, але не для читання, тобто не для зберігання даних. Для зберігання даних використовується ще один компонент *SPA*-архітектури – модель (*model*), що було використано при проектуванні веб-кабінету студента.

Оскільки, більшість програмних продуктів під *WEB* використовують на концептуальному рівні клієнт-серверну архітектуру, то це призвело до розділення логіки роботи веб-орієнтованих систем на дві частини – фронтенд (*англ. front-end*) і бекенд (*англ. back-end*). Фронтенд і бекенд є двома основними складовими частинами веб-розробки. Вони відповідають за різні аспекти створення та функціонування веб-додатків.

На даному етапі розвитку WEB, односторінкові додатки швидко витісняють мультисторінкові додатки і є великим внеском у розробку масштабних, швидких, динамічних веб-систем, що було враховано при проектуванні додатку веб-кабінету студента.

Динаміка розвитку сайтів вимагає готовності швидко реагувати на зміни і впроваджувати їх з максимальною оперативністю. Це також стосується навчальних порталів, зокрема онлайн-кабінетів для студентів.

РОЗДІЛ 1
ЗАДАЧІ СТВОРЕННЯ ВЕБ-КАБІНЕТУ СТУДЕНТА

1.1. Поняття Всесвітньої мережі та онлайн-сервісу

Всесвітня мережа (*WEB*) – це єдиний інформаційний простір, який складається із сотень мільйонів взаємозв'язаних гіпертекстових електронних документів, що зберігаються на веб-серверах. Вона вважається найпопулярнішою і найцікавішою службою мережі Інтернет, яка дозволяє отримувати доступ до інформації незалежно від місця її розташування.

Бернерс-Лі став піонером раннього розвитку Інтернету. До жовтня 1990 року Бернерс-Лі написав три фундаментальні технології, які стали основою Інтернету, включаючи найперший редактор/браузер веб-сторінок (*WorldWideWeb.app*):

- *HTML* (HyperText Markup Language) – стандартизована мова розмітки документів, якою створюють структуру сторінки (заголовки, абзаци, списки);
- *URI* або *URL* (Uniform Resource Locator) – уніфікований ідентифікатор ресурсу або локатор, унікальна адреса, що використовується для ідентифікації кожного ресурсу в Інтернеті;
- *HTTP* (HyperText Transfer Protocol) – протокол прикладного рівня передачі гіпертексту, який дозволяє отримувати пов’язані ресурси з усього Інтернету.

Для перегляду інформації, отриманої від веб-сервера використовується спеціальне програмне забезпечення на стороні клієнта (користувача) під назвою веб-браузер. Основна функція веб-браузера − відображення гіпертексту.

Всесвітня мережа міцно пов'язана з такими поняттями як, гіпертекст і гіперпосилання. Більша частина інформації у всесвітній павутині являє собою саме гіпертекст.

Гіпертекст – це спосіб організації тексту, графіки й інших даних, у якому елементи даних пов’язані між собою. Пов’язані можуть бути як елементи одного документа, так і різних документів. Загальновідомим прикладом гіпертексту є веб-сторінки – окремі документи Всесвітньої павутини. Веб-сторінка – це текстовий файл, що містить опис зображення мультимедійного документа на мові гіпертекстової розмітки (*HTML*). Сторінка може містити не тільки форматований текст, а й графічні, звукові та відео об'єкти.

Для створення і відображення гіпертексту у Всесвітній павутині традиційно використовується мова *HTML*. Написання коду гіпертексту називається верстkою. Після написання коду *HTML*, він зберігається у спеціальному файлі; такий *HTML*-файл є найпоширенішим ресурсом Всесвітньої мережі. Після того, як цей файл стає доступним веб-серверу, його починають називати «веб-сторінкою». Набір веб-сторінок утворює веб-сайт.

Веб-сторінки пов’язані між собою за допомогою гіперпосилань, за допомогою яких можна встановити зв'язок між окремими елементами сторінки, між сторінками різних сайтів тощо. З будь-яким фрагментом тексту або, наприклад, із малюнком, можна пов'язати інший веб-документ, тобто встановити гіперпосилання. У цьому разі під час клацання лівою клавішею миші на тексті або рисунку, що є гіперпосиланням, відправляється запит на доставку нового документа. Цей документ, у свою чергу, також може мати гіперпосилання на інші документи. Таким чином сукупність величезного числа гіпертекстових електронних документів, які зберігаються в веб-серверах, утворює своєрідний гіперпростір документів, між якими можливе переміщення.

Для покращення візуального сприйняття Всесвітньої павутини розробники застосовують спеціальну мову стилю сторінок *CSS*, яка дозволяє задавати єдині стилі оформлення для багатьох веб-сторінок одразу. *CSS* (Cascading Style Sheets) – мова для опису та стилізації зовнішнього вигляду документа. Завдяки *CSS*-коду браузер

розуміє, як саме відображати елементи. Наприклад, *CSS* задає кольори та параметри шрифтів, визначає, як розташовуватимуться різні блоки сайту.

Протокол *HTTP* – це протокол, який веб-браузери використовують для передачі файлів із веб-сервера. *HTTP* також відомий як протокол передачі гіпертексту. Це набір правил обміну даними між комп'ютерами, підключеними до всесвітньої павутини. Сторінки зберігаються на серверах, й після передаються на ваш комп'ютер, коли ви щось запитуєте в інтернеті. У результаті мережа цих з'єднань і створює Інтернет. Без *HTTP* не існувало б Всесвітньої павутини.

Онлайн-сервіс (веб-сервіс) – це сукупність програмно-технічних засобів, інтегрованих з метою ефективного опрацювання веб-ресурсів, які знаходяться у веб-просторі. Онлайн-сервіси можуть бути розроблені для виконання різних завдань, таких як комунікація, обмін інформацією, покупки товарів і послуг, здійснення фінансових операцій, розваги та багато іншого.

Онлайн-сервіси можуть включати в себе такі платформи, як веб-сайти, мобільні додатки, соціальні мережі, електронну пошту, хмарні сховища, потокове відео, онлайн-ігри та багато іншого. Вони дозволяють користувачам отримувати доступ до функціональності та ресурсів, що надаються через Інтернет, без необхідності встановлення спеціального програмного забезпечення на своєму комп'ютері або пристрої.

Веб-сервіси стали надзвичайно популярними в сучасному світі, оскільки вони забезпечують зручність, доступність та глобальну підключеність. Вони дозволяють користувачам взаємодіяти з інформацією та послугами в будь-який зручний для них час і місце, що робить їх незамінними в різних сферах життя, включаючи бізнес, освіту, комунікацію, розваги та інші.

За допомогою таких сайтів в Інтернеті можна робити що завгодно: проводити грошові операції, спілкуватися, шукати, зберігати, редагувати, пересилати та розповсюджувати інформацію, організовувати дистанційне навчання та багато іншого.

Існує безліч видів онлайн-сервісів, оскільки їх розмаїтість широка і постійно зростає. Ось кілька загальних видів онлайн-сервісів:

Соціальні мережі: платформи, які дозволяють користувачам створювати профілі, спілкуватися, ділитися відомостями, фотографіями та відео. Приклади: *Facebook, Instagram, Twitter*.

Електронна пошта: сервіси, що надають можливість відправляти, отримувати та керувати електронними листами. Приклади: *Gmail, Outlook, Yahoo Mail*.

Хмарні сховища: сервіси для зберігання та синхронізації даних в Інтернеті, що дозволяють користувачам отримувати доступ до своїх файлів з різних пристроїв. Приклади: *Google Drive, Dropbox, OneDrive*.

Онлайн-торгівля: платформи для покупки та продажу товарів та послуг через Інтернет. Приклади: *Amazon, eBay, Alibaba*.

Онлайн-банкінг: сервіси, що надають можливість клієнтам банків здійснювати фінансові операції через Інтернет, такі як перекази коштів, оплата рахунків та керування банківськими рахунками. Приклади: *PayPal, Revolut,* Online-банкінг від різних фінансових установ.

Онлайн-освіта: платформи, що надають можливість навчатися в Інтернеті, включаючи онлайн-курси, вебінари, відеолекції та інші освітні ресурси. Приклади: *Coursera, Udemy, Khan Academy*.

Потокове відео: сервіси, що надають доступ до стрімінгового відеоконтенту, такого як фільми, серіали, телепередачі. Приклади: *Netflix, YouTube, Amazon Prime Video*.

Онлайн-геймінг: платформи для гри в комп'ютерні ігри через Інтернет, включаючи мультиплеєрні ігри та онлайн-ігрові сервіси. Приклади: *Steam, PlayStation Network, Xbox Live*.

Це лише деякі приклади онлайн-сервісів, але реальна кількість видів сервісів в Інтернеті є набагато більшою і постійно розширюється залежно від потреб та технологічних змін.

1.2. Задачі онлайн-сервісу для веб-кабінету студента

Дізнатися та переглянути більш детальну інформацію про кожну з дисциплін, розклад занять та календар всіх подій в університеті можна в електронному кабінеті студента, який призначений спростити процеси, що напряму пов’язані зі студентами, їх навантаженням та успішністю.

Веб-кабінети студента часто надаються навчальними закладами для студентів з метою забезпечення доступу до різноманітної інформації та функціональності. Основні задачі, які можуть бути вирішені у веб-кабінеті студента, включають:

1. Реєстрація та управління курсами: студенти можуть зареєструватися на певні курси або додати/видалити курси в своєму розкладі. Веб-кабінет дозволяє переглядати і управляти списком курсів, датами та часами занять.

2. Оцінки та академічний прогрес: студенти можуть переглядати свої оцінки за курси, заліки, іспити та інші академічні досягнення. Веб-кабінет може надавати детальну інформацію про результати, середній бал, рейтинг та коментарі викладачів.

3. Розклад занять: студенти можуть переглядати свій індивідуальний розклад занять, включаючи час, місце та викладача. Це допомагає студентам планувати свій час та бути в курсі графіка навчальних занять.

4. Завдання та домашні завдання: веб-кабінет може містити розділ для перегляду та здачі домашніх завдань або завдань, призначених викладачем. Студенти можуть завантажувати роботи, переглядати вимоги до завдань та отримувати повідомлення про строк здачі.

5. Комунікація з викладачами та спілкування: веб-кабінет може включати інструменти для комунікації з викладачами та іншими студентами. Це можуть бути електронна пошта, форуми, чати або системи обміну повідомленнями, які дозволяють студентам задавати питання.

6. Отримання оперативної інформації щодо освітнього процесу.

7. Керування особистими даними: студенти можуть оновлювати свої контактні дані, адресу, номери телефонів тощо, щоб забезпечити актуальність інформації про себе.

8. Доступ до навчальних матеріалів: студентам надається можливість завантажувати або отримувати доступ до навчальних матеріалів, які стосуються їхніх курсів або предметів.

9. Фінансові питання: деякі веб-кабінети студента можуть надавати можливість переглядати фінансову інформацію, таку як рахунки за навчання, оплати, стипендії та інші фінансові деталі.

Конкретний функціонал та задачі існуючих веб-кабінетів студента можуть варіюватися в залежності від університету або навчального закладу.

1.3. Огляд існуючих веб-кабінетів студента

Існує багато різних веб-кабінетів студента, які надаються різними університетами та навчальними закладами. Розглянемо декілька прикладів найпопулярніших веб-кабінетів студента, для того, щоб звернути увагу на їхні плюси і мінуси, та уникнути недоліків у своєму проекті.

Blackboard – це онлайн-сервіс, який дає можливість співробітникам компанії за допомогою навчання, персонального розвитку, тренінгів отримати необхідні знання для виконання роботи. Сервіс поєднує у собі такі функції, як інформаційну систему учнів, систему управління навчанням, планування ресурсів підприємства, управління взаємовідносинами з учасниками. Сучасна хмарна *CRM* допомагає забезпечувати співробітникам та викладачам доступ до аналітики та розширеної інформації про шлях студентів, що дозволяє приймати рішення на основі даних та своєчасно спілкуватися.

Веб-сервіс *Moodle* – навчальна платформа, призначена для об'єднання педагогів, адміністраторів і студентівв одну надійну, безпечну та інтегровану систему для створення персоналізованого навчального середовища. Безкоштовна, відкрита (*Open Source*) система управління навчанням.

Онлайн-платформа *Canvas* для вищої освіти, яка дозволяє викладачам, студентам та навчальним закладам реалізувати дистанційний процес навчання; це

безкоштовна платформа із зручними інструментами для створення окремих занять та курсів.

Google Classroom – це онлайн-клас, який допомагає вчителям керувати навчанням та створювати інтерактивні заняття, допомагаючи студентам розширювати навчання за допомогою інструментів, доступних в Інтернеті. Послуга також дозволяє створювати різні класи, розподіляти завдання та надсилати та отримувати нотатки та зворотній зв'язок.

Кожен сервіс має свої плюси та мінуси, тому що їх створює людина, а не комп'ютер, розглянемо головні з них у вище перелічених онлайн-сервісах.

Серед плюсів *Blackboard* можна виділити: зручна система керування навчанням з багатьма функціями, включаючи розклад занять, завдання, спілкування з викладачами та однокурсниками, доступ до навчальних матеріалів. Серед мінусів можна виділити: інтерфейс може бути дещо складним для нових користувачів, адаптація до системи може зайняти деякий час.

Наступний веб-сервіс *Moodle* має такі плюси: відкрите програмне забезпечення з великою спільнотою користувачів, можливість розробки спеціалізованих модулів та розширень, гнучкість у керуванні навчальними матеріалами та завданнями. Серед мінусів: інтерфейс може виглядати менш сучасним порівняно з іншими платформами, може бути потрібне додаткове налаштування для підтримки певних функцій.

Онлайн-платформа *Canvas* серед переваг має: зручний та інтуїтивно зрозумілий інтерфейс, широкий функціонал для організації навчального процесу, включаючи завдання, форуми, співпрацю, звіти тощо. Недоліками є: деякі користувачі вказують на обмежену можливість настроювання платформи, особливо для більших університетів.

Клас *Google Classroom* відзначився такими перевагами: інтеграція з іншими Google-продуктами, простота використання та навігації, можливість легко створювати завдання, обмінюватися документами та спілкуватися з викладачами та однокурсниками. З мінусів: може бути менш функціональним порівняно з іншими веб-кабінетами, обмежена можливість настроювання та розширення функціоналу.

Кожен веб-кабінет має свої переваги та недоліки, і вибір залежить від конкретних потреб та вимог студента та навчального закладу. Деякі студенти можуть віддавати перевагу простоті використання та інтуїтивному інтерфейсу, тоді як іншим важлива гнучкість та розширюваність платформи.

1.4. Висновок до розділу 1

Всесвітня мережа (*WEB*) – це єдиний інформаційний простір, який складається із сотень мільйонів взаємозв'язаних гіпертекстових електронних документів, що зберігаються на веб-серверах.

В цілому можна зробити висновок, що Всесвітня павутина побудована з використанням: мови *HTML*, уніфікованих локаторів ресурсів *URL* та протоколу *HTTP*.

Онлайн-сервіс (веб-сервіс) – це сукупність програмно-технічних засобів, інтегрованих з метою ефективного опрацювання веб-ресурсів, які знаходяться у веб-просторі.

Онлайн-сервіси можуть включати в себе такі платформи, як веб-сайти, мобільні додатки, соціальні мережі, електронну пошту, хмарні сховища, потокове відео, онлайн-ігри та багато іншого. Вони дозволяють користувачам отримувати доступ до функціональності та ресурсів, що надаються через Інтернет, без необхідності встановлення спеціального програмного забезпечення на своєму комп'ютері або пристрої.

Веб-кабінети студента часто надаються навчальними закладами для студентів з метою забезпечення доступу до різноманітної інформації та функціональності.

Розглянуто декілька існуючих сервісів: онлайн-сервіс *Blackboard*, веб-сервіс *Moodle*, онлайн-платформа *Canvas* та *Google Classroom*. Розглянуто їх плюси та мінуси.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ВЕБ-КАБІНЕТУ СТУДЕНТА

2.1. Створення веб-кабінету студента в клієнт-серверній архітектурі

Головною метою є − проектування онлайн-сервісу, який включає в себе необхідні функції для організації та автоматизації навчального процесу з мінімальними витратами часу і використанням людського ресурсу. Також важливим є зручність використання даного сервісу організаторами і користувачами на всіх етапах, які призначені спростити процеси, що напряму пов’язані зі студентами, їх навантаженням та успішністю.

Клієнт-серверна архітектура (*англ. client-server architecture*) є одним з основних підходів до розподіленої обробки і обміну даними між комп'ютерами у мережі. В основі цієї архітектури лежать два компоненти: клієнт і сервер.

Клієнт – комп'ютер на стороні користувача, який відправляє запит до сервера для надання інформації або виконання певних дій.

Сервер – більш потужний комп'ютер або обладнання, призначене для вирішення певних завдань з виконання програмних кодів, виконання сервісних функцій за запитом клієнтів, надання користувачам доступу до певних ресурсів, зберігання інформації і баз даних.

Клієнт і сервер спілкуються між собою через мережу, таку як Інтернет або локальну мережу. Модель такої системи полягає в тому, що клієнт надсилає запит на сервер, де він обробляється, і готовий результат відправляється клієнтові.

Сервер може обслуговувати кілька клієнтів одночасно. Якщо одночасно приходить більше одного запиту, то вони встановлюються в чергу і виконуються сервером послідовно. Іноді запити можуть мати пріоритети. Запити з більш високими пріоритетами повинні виконуватися раніше. Весь цей процес відбувається за допомогою певного протоколу комунікації, наприклад *HTTP*. (рис. 2.1).

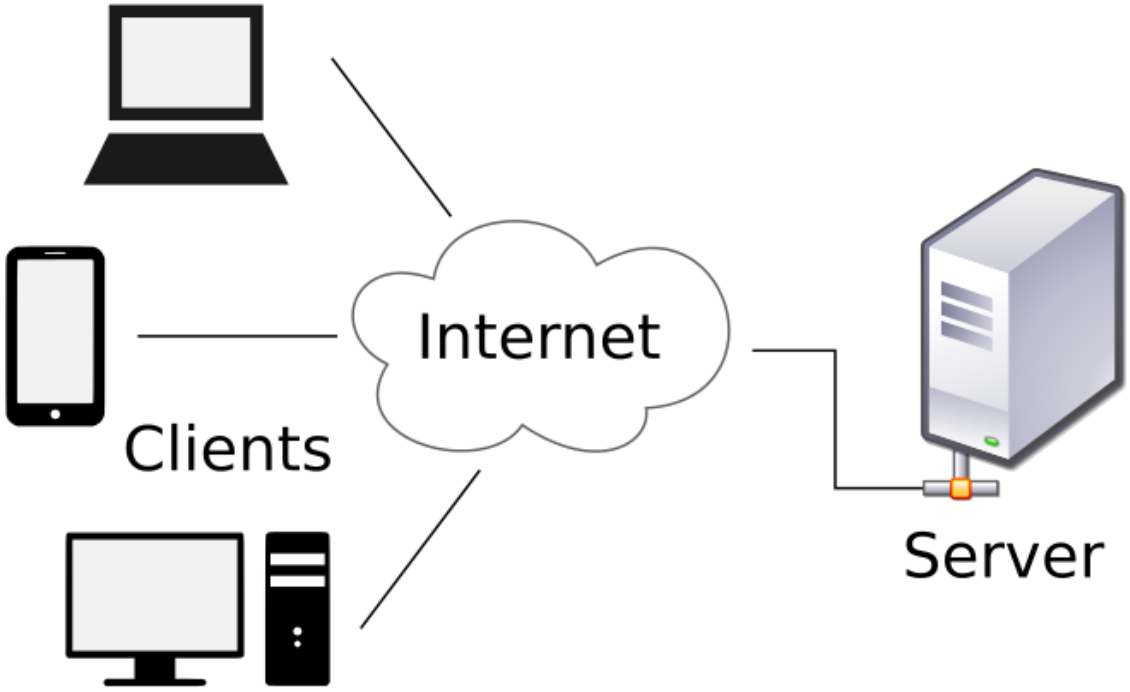

Рис. 2.1. Клієнт-сервер архітектура

Клієнт-серверну архітектуру можна означити, як концепцію інформаційної мережі в якій основна частина її ресурсів зосереджена в серверах, обслуговуючих своїх клієнтів.

Функції, які реалізуються на стороні клієнта:

- надання користувальницького інтерфейсу;
- формулювання запиту до сервера і його відправка;
- отримання результатів запиту і відправка додаткових команд (запитів на додавання, оновлення або видалення даних).

Функції, які реалізуються на сервері:

- зберігання, доступ, захист і резервне копіювання даних;

– обробка клієнтського запиту;

– відправлення результату (відповіді) клієнту.

При проектуванні веб-кабінету студента було використано наступну архітектуру. Оскільки, більшість програмних продуктів під *WEB* використовують на концептуальному рівні клієнт-серверну архітектуру, то це призвело до розділення логіки роботи веб-орієнтованих систем на дві частини – фронтенд (*англ. front-end*) і бекенд (*англ. back-end*). Фронтенд і бекенд є двома основними складовими частинами веб-розробки. Вони відповідають за різні аспекти створення та функціонування веб-додатків.

Фронтенд охоплює все, що відбувається на стороні клієнта (тобто на боці користувача). Це відповідальність за створення і реалізацію інтерфейсу користувача. Розробники фронтенду працюють з мовами програмування, такими як *HTML, CSS* та *JavaScript*, для створення динамічного та інтерактивного веб-інтерфейсу. Вони займаються розміщенням та стилізацією елементів на сторінці, роботою з анімацією, валідацією форм, взаємодією з користувачем та іншими аспектами, пов'язаними з зовнішнім виглядом та функціональністю веб-додатку на стороні клієнта.

Бекенд відповідає за серверну частину веб-додатка. Це відповідальність за обробку запитів, взаємодію з базою даних, бізнес-логіку та інші серверні операції. Розробники бекенду використовують мови програмування, такі як *Python, Java, Ruby, PHP* та інші, для розробки серверної логіки. Вони створюють *API* (*Application Programming Interface*), які надають можливість комунікації між фронтендом та бекендом, забезпечують зберігання та обробку даних, автентифікацію, авторизацію, роботу зі сторонніми сервісами та інші функції, що забезпечують функціональність веб-додатка.

Важливо зазначити, що фронтенд і бекенд є взаємопов'язаними частинами веб-додатка і взаємодіють між собою через мережу. Фронтенд передає запити користувача на бекенд, а бекенд обробляє ці запити та надсилає відповіді назад на фронтенд.

Сучасний *WEB* дедалі частіше використовує технології *Single Page Application* (*SPA*) для фронтенд розробки за допомогою якого був спроектований онлайн-сервіс.

SPA – це веб-додаток, розміщений на одній веб-сторінці, яка для забезпечення роботи завантажує весь необхідний код разом із завантаженням самої сторінки.

Це такий тип програм, який відкривається в браузері і при цьому не вимагає постійного перезавантаження сторінки за допомогою *AJAX*-запитів – всі візуальні елементи конструюються прямо в браузері за допомогою технології *JavaScript* маніпуляцій з *DOM*-структурою документа. Це можливо, оскільки такі програми не потрібно постійно надсилати запити на сервер, а контент на сторінці підвантажується динамічно. Наприклад, раніше у соціальній мережі потрібно було перезавантажити сторінку, щоб перевірити, чи немає нових повідомлень, тепер вони з'являються автоматично. Таким чином, користувач отримує досвід дуже схожий на досвід взаємодії із звичайним десктопним або нативним додатком.

Односторінкові програми найчастіше використовують у сервісах, де користувач проводить на одній сторінці багато часу або робить з нею якісь дії, наприклад:

- переглядає пошту та зазначає листи як спам;
- пише пости та коментує чужі;
- дивиться серіали;
- обирає квартиру;
- розглядає малюнки і збирає в тематичні дошки.

Прикладами технології *SPA* є реалізований сервіс *Gmail* від *Google, Facebook, Netflix, AirBnB* та *Pinterest*. Зверніть увагу, як працює перемикання між розділами в інтерфейсі *Facebook*: якщо відкрити головну, а потім перейти до «Групи», то зміст сторінки зміниться, але шапка залишиться на своєму місці – перезавантаження сторінки не відбувається.

Основне місце в *SPA*-архітектурі при проектуванні веб-кабінету студента займає відображення (*view*) – те, що бачить і з чим взаємодіє користувач. Результатом роботи відображення є звичайний *HTML*, що відображається браузером. *SPA*-додаток використовує *DOM* лише для запису змін, але не для читання, тобто не для зберігання даних. Для зберігання даних використовується ще один компонент *SPA*-архітектури – модель (*model*), що було використано при проектуванні веб-кабінету студента.

Односторінкові додатки (*SPA*) стають все більш популярнішими за мультисторінкові (класичні) додатки (*MPA*). Мультисторінковий додаток – це традиційний веб-додаток, в якому з сервера запитується нова сторінка для відображення щоразу, як відбувається обмін даними туди й назад. Обсяг контенту, який вони несуть, величезний, тому вони, як правило, багаторівневі, зі значною кількістю посилань і складними інтерфейсами.

Тому є актуальними дослідження структури, поведінки, швидкодії односторінкових веб-додатків для того, щоб підвищувати їх якість та надійність. На рисунку 2.2 зображено структуру односторінкового та мультисорінкового додатка.

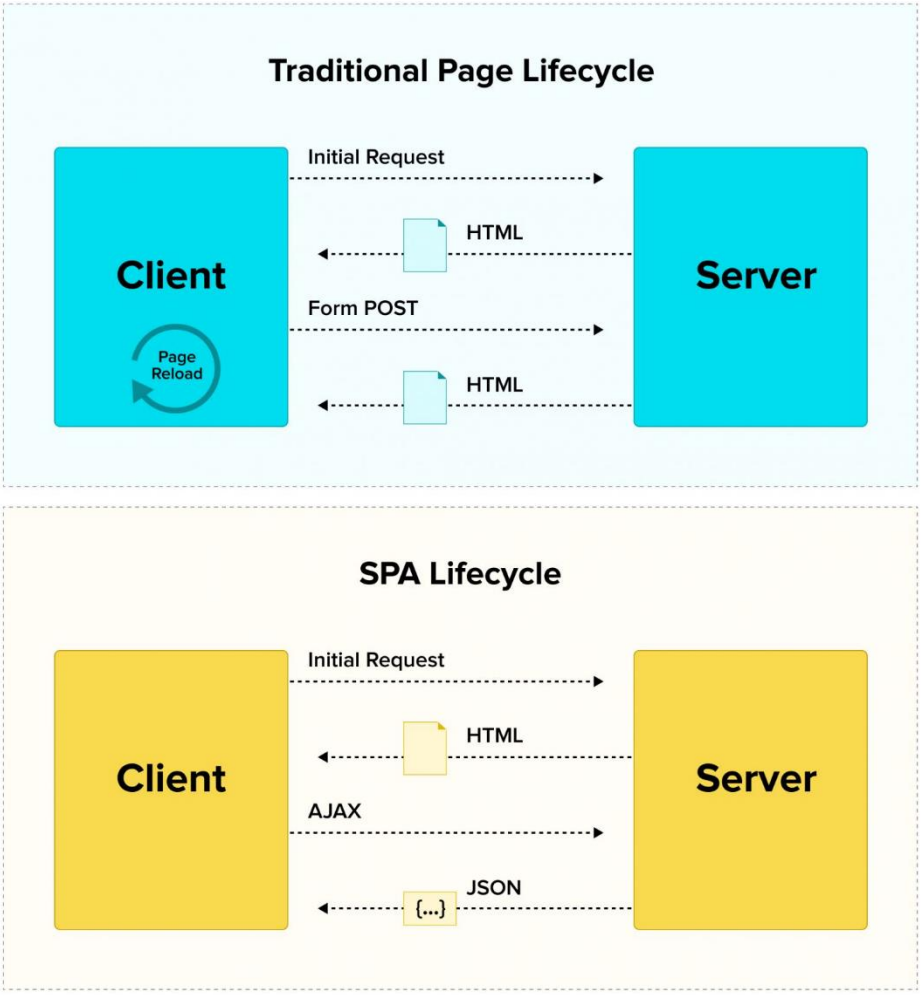

Рис. 2.2. Структура односторінокового та мультисторінкового додатку

Порівнявши ці структури, можна зробити висновок, що справді односторінкові додатки є високопродуктивними завдяки тому, що не навантажують серверну частину

частими запитами як це роблять класичні додатки. Односторінкові додатки працюють швидко, завантажуючи усі дані при зверненні до сторінки. При виконанні певних дій користувачем *SPA* завантажує дані по мірі необхідності без повного перезавантаження. Це є однією із основних переваг при виборі структури веб-додатку.

Серед інших переваг *SPA*, можна виділити такі:

- *SPA* швидкі. Перехід між модулями у додатку відбувається швидше: потрібні ресурси вже завантажені, потрібно просто підставити дані, які користувач запитав. Часто при цьому сервер повертає не вагомий *HTML*, а легкий *JSON* або *XML*.
- Ще використання *JSON* спрощує розробку програми для різних платформ. Якщо для веб-версії розробити звичайний сайт, який приймає від сервера *HTML*, то для мобільного додатку доведеться писати доопрацювання, оскільки *HTML* не підійде. *JSON* робить відповідь сервера універсальною.
- *SPA* гнучкі. Якщо користувач весь час працює з однією сторінкою, простіше робити цікаві переходи та анімацію елементів. Можна працювати зі станом кнопок, вкладок та перемикачів. Таким чином, інтерфейс *SPA* може бути схожий швидше на повноцінний додаток, а не на простий сайт.
- *SPA* працюють скрізь. Все, що потрібно для *SPA* – підтримка JavaScript. Такі сайти добре працюють і на робочому столі, і в Інтернеті, можуть частково замінити повноцінні мобільні програми.

При проектуванні веб-кабінету студента був використаний модульний підхід *SPA*. Додаток розбивається на незалежні модулі, які виконують конкретні функціональні завдання. Кожен модуль має свою відповідальність і виконує обмежений набір функцій. Цей підхід допомагає полегшити розробку, підтримку та масштабування додатку. Клієнтська та серверна частини в *SPA* є повністю незалежними, що дає змогу ізолювати їх один від одного. Зміни у серверній частині, після заміни, ніяким чином не вплине на клієнтську частину.

На даному етапі розвитку *WEB*, односторінкові додатки швидко витісняють мультисорінкові додатки і є великим внеском у розробку масштабних, швидких,

динамічних веб-систем, що було враховано при проектуванні додатку веб-кабінету студента.

2.2. Засоби створення веб-кабінету студента

При проектуванні веб-кабінету студента було використано *JavaScript* (*JS*) – це динамічна мова програмування, яку використовують для написання фронтенд та бекенд-частин сайтів, а також мобільних додатків. Мова відповідає за інтерактивність та функціонал інтерфейсу, який неможливо виконати лише за допомогою *HTML* та *CSS*.

JavaScript називають мовою сценаріїв або скриптів. *Скрипти* – це набір інструкцій, які виконуються під час завантаження сторінки. Браузер самостійно інтерпретує код *JavaScript*, для цього навіть не потрібна компіляція (переведення мови програмування в машинний код).

За допомогою *JavaScript* можуть бути розроблені як веб-сайти, так і серверні модулі. Основні сфери застосування:

- *Веб-сайти та веб-додатки*. Найпопулярніша сфера застосування мови *JavaScript* – це написання коду для сайту. Майже на кожному сучасному веб-сайті вживають код, написаний на *JS*.
- *Розширення для браузера*. Невеликі прості скрипти, які додають додатковий функціонал – блокують рекламу, дозволяють зберігати аудіо, надсилають повідомлення про нові листи або змінюють колірну схему сайту.
- *Мобільні додатки*. Їх можна писати спеціальними мовами, наприклад *Kotlin*. Але якщо потрібно щось просте, наприклад, інтерфейс для роботи з хмарним сховищем, його можна написати на *JavaScript* і зібрати в додаток за допомогою спеціальних інструментів.
- *Серверна частина сайтів та програм*. Мова програмування *JavaScript* можна використовувати для написання будь-яких сервісів: чатів, комп'ютерних програм і навіть нейромереж. Для цього до нього необхідно підключити двигун *NodeJS*.

JavaScript має кілька переваг, які зробили її однією з найпопулярніших мов програмування:

- *JavaScript* є основною мовою для розробки веб-додатків. *JS* може бути вбудована безпосередньо в *HTML*-код сторінки і виконувати різні функції, що забезпечує інтерактивність та динамічність веб-сторінок.
- Вона легко інтегрується з версткою (*HTML*) та дає змогу налаштувати комунікацію з сервером.
- Має простий і зрозумілий синтаксис, що полегшує її вивчення для початківців у програмуванні. Вона також має багато ресурсів, документацію та активну спільноту, яка допомагає новачкам у розумінні мови.
- Швидкість виконання: Завдяки технології *Just-in-Time* (JIT) компіляції та оптимізаціям в різних веб-переглядачах, *JavaScript* став значно швидшим у виконанні. Це дозволяє розробникам створювати швидкі та ефективні веб-додатки, які можуть обробляти великий обсяг даних.
- *JavaScript* має велику кількість бібліотек та фреймворків, які розширюють її можливості і полегшують розробку. Наприклад, бібліотека *jQuery* допомагає спростити маніпулювання *DOM*, а фреймворки, такі як *React*, *Angular* та *Vue.js*, надають потужні інструменти для створення складних веб-додатків.
- *JavaScript* може бути виконаний на різних платформах, включаючи веб, мобільні пристрої та настільні комп'ютери. Це означає, що один код *JavaScript* може працювати на різних пристроях, що полегшує розробку кросплатформних додатків.
- Мова підтримує асинхронну модель програмування, що дозволяє виконувати багатозадачні операції без блокування інших частин програми. Це особливо корисно для мережевих операцій, таких як отримання даних з сервера, де затримки можуть бути значними.
- *JavaScript* дозволяє створювати багатофункціональні та інтерактивні веб-сторінки, які реагують на дії користувачів. Вона надає можливість

маніпулювати *DOM* для зміни вмісту та вигляду сторінки без необхідності перезавантаження.

Незважаючи на багато переваг, *JavaScript* також має свої недоліки:

- Оскільки *JavaScript* виконується на стороні клієнта, він піддається ризику зловживання та вразливостям безпеки. Це може включати скрипти, які використовуються для впровадження шкідливих програм або зловживання даними користувачів.
- Залежність від браузера: Різні веб-переглядачі можуть мати відмінності в підтримці та виконанні *JS*. Це може призвести до проблем з переносимістю та несправностями коду на різних платформах.
- *JavaScript* у веб-переглядачі обмежена у своїй можливості взаємодіяти з операційною системою (ОС) та файловою системою. Це може створювати обмеження для деяких типів додатків, які потребують ширшого доступу до системних ресурсів.
- *JavaScript* має кілька версій та стандартів, таких як *ECMAScript 5, ECMAScript 6* і так далі. Це може призвести до проблем з сумісністю між різними версіями та інтерпретаторами, особливо при розробці більших проектів.

Слід зазначити, що спільнота користувачів *Javascript* активно покращує мову, усуваючи багато недоліків. Вузьких місць стає дедалі менше. Браузери постійно вдосконалюють роботу із *JS*.

При проектуванні веб-кабінету студента було розглянуто основні бібліотеки та фреймворки *JavaScript*, на яких побудована переважна більшість онлайн-сервісів у *WEB*.

Фреймворк дозволяє розробнику не писати програми та сайти з нуля: у ньому вже є всі необхідні бібліотеки та частини коду. Тобто фреймворк бере на себе роботу з базами даних, з файловою системою та іншими компонентами великої програми, а розробнику залишається просто продумати архітектуру програми, дописати необхідні компоненти та зв'язати їх між собою.

Так як фреймворки пропонують готові рішення для розробки їх постійно плутають з бібліотекою. Відмінностей багато, але найочевидніше – фреймворки визначають архітектуру, а бібліотеки – ні.

Є й інші відмінності. Наприклад, коли ми користуємось бібліотекою, то просто викликаємо бібліотечні функції. Але коли ми пишемо на фреймворку, він викликає наш код. Саме тому ми повинні писати функції того виду, який вимагає фреймворку, щоб він міг їх запустити. І нарешті, будь-яка бібліотека виконує незрівнянно менше завдань, ніж фреймворк.

React (або *React.js*) – це *open-source JavaScript* бібліотека для розробки користувацьких інтерфейсів (*UI*), розроблена командою *Facebook*. *React* набула великої популярності серед розробників завдяки своїй ефективності, швидкості та зручному способу створення високоякісних веб-додатків.

Основною концепцією *React* є створення компонентів – незалежних, повторно використовуваних блоків коду, які представляють частини інтерфейсу користувача. Компоненти можуть бути простими, такими як кнопка або текстове поле, або складними, такими як форма або список даних. Кожен компонент має свою внутрішню логіку, стан і методи, що дозволяють йому реагувати на зміни та взаємодіяти з користувачем.

Одним із ключових принципів *React* є використання віртуального *DOM*. *React* зберігає внутрішню представлення структури *UI* у пам'яті, яке називається віртуальним *DOM*. При зміні стану компонента *React* автоматично оновлює тільки необхідні частини віртуального *DOM* і виконує ефективне оновлення реального *DOM*, забезпечуючи оптимальну продуктивність та ефективне використання ресурсів.

React можна використовувати для розробки веб-додатків, мобільних додатків за допомогою *React Native* і навіть додатків для роботи зі штучним інтелектом за допомогою *React VR*. Його популярність серед розробників продовжує зростати завдяки його простоті використання, швидкості та розширюваності.

При цьому *React* підійде майже будь-якому проекту, але оскільки бібліотека спочатку створювалася для *Facebook*, то в додатках соцмереж вона проявить себе у всій красі: на *React* працюють *Pinterest* та *Instagram*.

Vue.js – це прогресивний фреймворк для створення інтерфейсів користувача (*UI*). *Vue.js* швидко став популярним серед розробників завдяки своїй простоті у вивченні, гнучкості та ефективності. Основними принципами фреймворку є декларативний підхід до розробки інтерфейсів, компонентна архітектура та реактивність. Vue.js дозволяє створювати компоненти, що перевикористовуються, які є незалежними блоками коду, що відповідають за певні частини інтерфейсу користувача. Компоненти можуть містити *HTML*-розмітку, *CSS*-стилі та *JavaScript*-логіку.

Однією з переваг *Vue.js* є його невеликий розмір та простота інтеграції з існуючими проектами. Він може використовуватися як основний фреймворк для розробки цілісних веб-додатків, так і як бібліотека для додавання інтерактивності та динамічних елементів на існуючі сторінки.

Vue.js орієнтований на розробку односторінкових програм (*SPA*), але також може використовуватися для створення компонентів усередині традиційних багатосторінкових програм. Він має активну спільноту розробників, які активно підтримують та розвивають цей фреймворк.

Будь-який фреймворк універсальний, але *Vue.js* особливо добрий для створення онлайн-магазинів: його використовують *Nintendo, Louis Vuitton* та *BMW USA*.

Angular – це платформа та фреймворк для розробки веб-додатків, створений та підтримуваний компанією *Google*. Він пропонує інструменти і функціональність для створення потужних односторінкових додатків (*SPA*) і додатків з багатим інтерфейсом користувача.

На ранньому етапі *Vue.js* надихався найкращими практиками *Angular*, але поріг входу суттєво відрізняється: щоб писати на Vue.js, досвідченому розробнику потрібен лише день вдумливого читання документації, чого не можна сказати про *Angular*.

Важливою особливістю *Angular* є те, що він створювався для роботи саме з великими програмами. При цьому *Vue* та *React* підходять для проектів будь-якої

складності, а ще до них можна без проблем підключити *TypeScript*, який є обов'язковим у роботі з *Angular*.

Через те, що *Angular* вимагає *TypeScript*, фреймворк залучає команди з бекендом *C*-подібними мовами (*C#, C++, Java*).

Головна перевага *Angular* є головним недоліком: цей фреймворк пропонує тільки один спосіб будувати архітектуру програми. Це підходить тим, хто хоче відразу сісти та писати код, не думаючи про архітектуру. Однак доведеться довго копатися в технічних деталях, перш ніж ви зможете продуктивно працювати.

Angular не беруть для легких статичних сайтів або одноразових додатків. В інших випадках його дуже вигідно використовувати: проект може бути більшим і складним, але його легко тестувати, масштабувати та рефакторити. З цих причин *Angular* використовують у *Gmail, Microsoft Office* та *Paypal*.

Онлайн-сервіс веб-кабінет студента спроектовано на технологіях клієнтської частини, але веб-сервіс повинен зберігати дані, тому було прийняте рішення використати систему керування базою даних *Firebase*.

У 2014 році *Google* придбала хмарну базу даних на основі *NoSQL* з назвою *Firebase*. *Firebase* – це платформа для розробки мобільних та веб-додатків з величезним функціоналом. Firebase надає набір інструментів та сервісів, які допомагають розробникам швидко створювати функціональні та ефективні додатки. Починалася вона як стартап, а сьогодні її використовують при розробці кращих кроссплатформних додатків. Головна перевага платформи в тому, що вона дозволяє розробнику не відволікатися на створення бекенду. І це спрощує і прискорює створення мобільних додатків, дає можливість повністю зосередитися саме на призначеному для користувача інтерфейсі.

Firebase – це одне з *BaaS*-рішень (*Backend as a Service*), яке дає розробнику масу можливостей. Це і сервер, і база даних, і хостинг, і аутентифікація в одній платформі. Так, *Firebase Realtime Database* надає розробникам *API*, який синхронізує дані додатки між клієнтами і зберігає їх в хмарному сховищі.

Firebase Realtime Database – це хмарна база даних типу *NoSQL*, яка використовує веб-сокети, що дозволяє клієнту отримувати інформацію в реальному

часі – без необхідності відправляти запити *GET* на сервер. Веб-сокет (*WebSocket*) – це протокол зв'язку між клієнтом та сервером, який дозволяє встановити постійне двостороннє з'єднання між ними. Він використовується для обміну даними в реальному часі та надає більш ефективний та масштабований спосіб спілкування порівняно з традиційними HTTP-запитами. Цей сервіс дозволяє розробникам зберігати та синхронізувати дані між різними клієнтськими додатками в режимі реального часу.

Firebase Realtime Database працює на основі моделі ключ-значення, де дані зберігаються у вигляді *JSON*-об'єктів. Зміни в даних негайно передаються всім підключеним клієнтам, що дозволяє створювати реактивні додатки, які оновлюються автоматично при зміні даних.

База даних *Firebase Realtime* є потужним інструментом для створення реального часу додатків, таких як чати, спільне редагування документів, співпраця в режимі реального часу та інші сценарії, де необхідна миттєва синхронізація даних між різними пристроями. Сервіс керування базою даних інтегрується з *Firebase Authentication* для забезпечення простої та інтуїтивної автентифікації для розробників.

При створенні веб-кабінету студента багато уваги приділяється питанням безпеки. Створювати систему аутентифікації кожен раз з нуля досить затратно, причому витрати ці найчастіше невиправдані. Впоратися з більшістю викликів дозволяє система аутентифікації *Firebase Authentication* – це сервіс, який надається *Firebase* для аутентифікації користувачів у мобільних та веб-додатках. Він дозволяє розробникам забезпечувати безпечну та просту аутентифікацію користувачів, щоб вони могли входити в систему, реєструватися та керувати своїми обліковими записами.

Користувачі можуть створювати облікові записи, використовуючи свою електронну пошту та пароль. *Firebase* автоматично зберігає та захищає облікові дані користувачів. Аутентифікацію можна виконувати через різні соціальні мережі, такі як *Google, Facebook, Twitter, GitHub* та інші. Це дозволяє користувачам використовувати свої наявні облікові записи для входу в додаток.

Firebase Authentication також забезпечує різні функції безпеки, такі як підтвердження електронної пошти, підтвердження номера телефону та обмеження доступу до певних ресурсів на основі ролей та прав доступу. Він також інтегрується з іншими сервісами *Firebase*, щоб забезпечити безпеку та зручність використання.

Переваги *Firebase Authentication* були враховані при створенні веб-кабінету студента та полягають в наступному: тісна інтеграція з іншими функціями *Firebase,* використання галузевих стандартів, таких як *OAuth 2.0* і *OpenID Connect*, які спрощують інтеграцію з серверним кодом, два підходи використання, безпека автентифікації, безпечний доступ до сервісів *Google*.

Використовуються два варіанти для розробки: *FirebaseUI* – повністю універсальне рішення для виконання автентифікації або набір інструментів та бібліотек *Firebase Authentication SDK*, який дозволяє вручну інтегрувати один або кілька методів входу в додаток.

При проектуванні веб-кабінету були вибрані дані сервіси *Firebase* через їхню пристосованість до онлайн-сервісів. Це забезпечить легкість роботи з ними. А також, дані сервіси мають хорошу захищеність, що відміняє необхідність в реалізації методів захисту в даному веб-додатку.

2.3. Проектування функцій модулів онлайн-сервісу

Однією з основних частин створення онлайн-сервісу є чіткий опис задачі та проектування згідно вимог, без цього просто неможливо створити хороший та якісний продукт. Проаналізувавши різні веб-кабінети студента, був спроектований онлайн-сервіс, який має мету максимально спрощувати і автоматизувати процеси, які напряму пов’язані зі студентами. Кожен з модулів онлайн-сервісу містить в собі набір функцій та цілей для вирішення відповідних завдань. Схематично всі завдання веб-сервісу можна розділити на сім модулів: реєстрація, автентифікація, головна сторінка списку навчальних предметів, сторінка позакласних заходів в університеті, додавання предметів, сторінка детального опису програми предмету та профіль користувача.

Модуль реєстрації – процес створення нового облікового запису користувача на веб-сервісі, який дозволяє користувачеві мати особистий профіль і отримати доступ до певних функцій та ресурсів. Реєстрація є першим кроком, коли користувач стає членом системи або отримує право на доступ до певних ресурсів або послуг. Цей модуль повинен відображати форму з такими полями реєстрації:

- текстове поле *електронна пошта*;
- текстове поле *ім'я*;
- текстове поле *пароль*;
- текстове поле *повторне підтвердження паролю*;
- кнопка типу «*checkbox*» для погодження з умовами та положеннями сервісу, кнопку типу «*submit*» для відправки даних користувача на сервер та подальшого запису в системі управління бази даних Firebase;
- текстове посилання на модуль авторизації.

Після того, як користувач зареєструвався і отримав обліковий запис, він може використовувати свої облікові дані для автентифікації і отримання доступу до системи, ресурсів або послуг.

Модуль автентифікації – виконує функцію входу користувача у систему. Це процес підтвердження ідентичності користувача. Автентифікація відбувається під час процесу входу, коли користувач надає свої ідентифікаційні дані для перевірки. Модуль відображає такі поля:

- текстове поле *електронна пошта*;
- текстове поле *пароль*;
- кнопка типу «*submit*» для формування запиту та відправки даних користувача на сервер для подальшої перевірки введених даних з даними у базі даних, що дозволить вхід у сервіс або сповістить про невдалу спробу входу;
- текстове посилання на модуль реєстрації.

Автентифікація є важливою складовою безпеки і захисту інформації, оскільки вона перевіряє правомірність доступу до системи або послуги. Це дозволяє

забезпечити, що лише правомірні користувачі можуть отримувати доступ до конфіденційної інформації і здійснювати певні операції.

Модуль головної сторінки списку навчальних предметів – перша сторінка, яку відвідувач бачить при завантаженні веб-сайту. Вона містить основну інформацію, функціональні елементи та навігацію, які надають користувачеві загальне уявлення про сайт та допомагають орієнтуватися в його структурі. Модуль виконує функцію відображення списку дисциплін студента в поточному році та складається з таких компонентів:

- верхній колонтитул, «шапка» сторінки (*header*), яка відображає логотип-посилання на головну сторінку та текстові-посилання на даний модуль з різними параметри фільтрації відображення списку предметів, на модуль позакласних заходів в університеті, на модуль профілю користувача та оброблювач події виходу з облікового запису користувача;
- компонент-фільтр списку предметів за семестром: всі, перший семестр та другий семестр;
- компонент текстове поле пошуку дисципліни за назвою або прізвищем викладача;
- компонент перемикач відображення предметів у вигляді списку або календаря;
- нижній колонтитул, «підвал» сторінки (*footer*) з повідомленням про авторське право.

Модуль сторінки позакласних заходів в університеті – виконує функцію відображення інформації про цікаві позакласні події, які відбуваються в університеті та містить у собі такі компоненти:

- компонент-фільтр списку заходів часовими параметрами: всі, майбутні та минулі;
- компонент текстове поле пошуку заходу за назвою;
- компонент перемикач відображення подій у вигляді списку або календаря;
- фото-банер з назвою події;

– три текстові вкладки, які відображають місцезнаходження, дату та час проведення заходу.

Модуль сторінки детального опису програми предмету – виконує функцію відображення повної, розгорнутої інформації про певну дисципліну та містить у собі такі компоненти:

– фото-банер з назвою дисципліни;
– секція загального опису дисципліни;
– секція програми навчальної дисципліни, по темам та часовому графіку проведення модульних та семестрової контрольних робіт;
– інформативний компонент з двома текстовими вкладками, які відображають прізвище викладача та корпус, аудиторію проведення лекційних та практичних занять.

Модуль додавання предметів – виконує функцію створення предмету та містить такі поля:

– текстове поле назви предмету;
– поле календаря для встановлення часових параметрів дисципліни;
– текстове поле опису предмету;
– поле файлового типу для завантаження фото-банера дисципліни;
– розширена форма програми курсу з випадаючим списком типів занять (лекція чи практика), текстовими полями: корпус, поверх та аудиторія, часовими полями для встановлення початку та кінця пари, заголовком й описом предмету;
– кнопки «*скасувати*» та «*створити*» для відміни створення предмету та повернення до модуля головної сторінки або створення запиту на сервер та подальшого запису предмету в базі даних.

Модуль профіль користувача – зберігає та керує інформацією про користувача системи. Використовується для зберігання основних даних користувача, таких як ім'я, електронна пошта, пароль, налаштування та надає змогу редагування даних профіля. Модуль повинен відображати такі поля:

– текстове поле *ім'я студента*;
– текстове поле *електронна пошта студента*;
– поле календаря для встановлення дати народження студента;
– текстове поле *курс та поточна група студента*;
– кнопка «*редагувати*» для формування запиту на сервер та редагування даних у базі даних певного користувача.

Правильно спроектований та описаний онлайн-сервіс є дуже важливим кроком перед тим, як конструювати та створювати код веб-сервісу. В наступному розділі буде розглянуто розробку, тобто яким чином створено веб-кабінет студента НАУ.

2.4. Висновок до розділу 2

Головною метою є − проектування онлайн-сервісу, який включає в себе необхідні функції для організації та автоматизації навчального процесу з мінімальними витратами часу і використанням людського ресурсу. Також важливим є зручність використання даного сервісу організаторами і користувачами на всіх етапах, які призначені спростити процеси, що напряму пов'язані зі студентами, їх навантаженням та успішністю.

Клієнт-серверна архітектура (*англ. client-server architecture*) є одним з основних підходів до розподіленої обробки і обміну даними між комп'ютерами у мережі. Модель такої системи полягає в тому, що клієнт надсилає запит на сервер, де він обробляється, і готовий результат відправляється клієнтові.

Сучасний *WEB* дедалі частіше використовує технології *Single Page Application* (*SPA*) для фронтенд розробки за допомогою якого був спроектований онлайн-сервіс. Був використаний модульний підхід SPA, при якому додаток розбивається на незалежні модулі, які виконують конкретні функціональні завдання. Кожен модуль має свою відповідальність і виконує обмежений набір функцій. Цей підхід допомагає полегшити розробку, підтримку та масштабування додатку.

При проектуванні веб-кабінету студента було використано *JavaScript* (*JS*) – це динамічна мова програмування, яку використовують для написання фронтенд та

бекенд-частин сайтів, а також мобільних додатків. Мова відповідає за інтерактивність та функціонал інтерфейсу, який неможливо виконати лише за допомогою *HTML* та *CSS*.

Firebase Realtime Database – це хмарна база даних типу *NoSQL*, яка використовує веб-сокети, що дозволяє клієнту отримувати інформацію в реальному часі – без необхідності відправляти запити *GET* на сервер. Веб-сокет (*WebSocket*) – це протокол зв'язку між клієнтом та сервером, який дозволяє встановити постійне двостороннє з'єднання між ними. Він використовується для обміну даними в реальному часі та надає більш ефективний та масштабований спосіб спілкування порівняно з традиційними HTTP-запитами. Цей сервіс дозволяє розробникам зберігати та синхронізувати дані між різними клієнтськими додатками в режимі реального часу.

При створенні веб-кабінету студента багато уваги приділяється питанням безпеки. Створювати систему аутентифікації кожен раз з нуля досить затратно, причому витрати ці найчастіше невиправдані. Впоратися з більшістю викликів дозволяє система аутентифікації *Firebase Authentication* – це сервіс, який надається *Firebase* для аутентифікації користувачів у мобільних та веб-додатках. Він дозволяє розробникам забезпечувати безпечну та просту аутентифікацію користувачів, щоб вони могли входити в систему, реєструватися та керувати своїми обліковими записами.

При проектуванні веб-кабінету було використано наступну архітектуру. Оскільки, більшість програмних продуктів під *WEB* використовують на концептуальному рівні клієнт-серверну архітектуру, то це призвело до розділення логіки роботи веб-орієнтованих систем на дві частини – фронтенд (*англ. front-end*) і бекенд (*англ. back-end*). Фронтенд і бекенд є двома основними складовими частинами веб-розробки. Вони відповідають за різні аспекти створення та функціонування веб-додатків.

Проаналізувавши різні веб-кабінети студента, був спроектований онлайн-сервіс, який має мету максимально спрощувати і автоматизувати процеси, які напряму пов’язані зі студентами. Схематично всі завдання веб-сервісу можна

розділити на сім модулів: реєстрація, автентифікація, головна сторінка списку навчальних предметів, сторінка позакласних заходів в університеті, додавання предметів, сторінка детального опису програми предмету та профіль користувача.

Правильно спроектований та описаний онлайн-сервіс є дуже важливим кроком перед тим, як конструювати та створювати код веб-сервісу. В наступному розділі буде розглянуто розробку, тобто яким чином створено веб-кабінет студента НАУ.

РОЗДІЛ 3
РОЗРОБКА ВЕБ-КАБІНЕТУ СТУДЕНТА

3.1. Створення проекту та структура файлів

Розробка веб-сервісу складається з таких етапів: вибір та налаштування середовища розробки, створення початкового *Vue.js* середовища та завантаження основних програм для розробки.

Перший етап створення будь-якого проекту завжди починається з вибору інтегрованого середовища розробки, де відбувається створення, редагування та налагодження програмного коду. Для проектування веб-кабінету було обрано *Microsoft Visual Studio Code*, тому що цей редактор є одним з найпопулярніших і розширюваних середовищ розробки на ринку та має такі переваги:

- підтримує операційні системи *Windows*, *macOS* та *Linux*, що дозволяє розробникам використовувати його на різних платформах;
- інтерфейс є інтуїтивно зрозумілим та простим у використанні. Він надає широкі можливості налаштування, але залишається легким та швидким;
- має потужну систему розширень, яка дозволяє розробникам встановлювати різні плагіни для розширення функціональності редактора;
- має багатий набір функцій редактора, таких як підсвічування синтаксису, автодоповнення, розгортання коду, рефакторинг, швидкі переходи між файлами та багато іншого. Він також підтримує режими роботи з одним або кількома файлами одночасно;
- та головна перевага *VS Code* є те, що редактор безкоштовний.

Після встановлення та налаштування редактору коду потрібно встановити розширення, які необхідні, щоб спростити велику кількість механічних дій, налагодження та збільшать стисліcть, зрозумілість, легкість у використанні та підтримку коду.

Vetur – це головне розширення для *Vue.js* розробника в редакторі *VS Code*. Забезпечує підсвічування синтаксису для, що допомагає розробникам легше сприймати структуру коду, надає автоматичне доповнення коду, що спрощує введення довгих або складних конструкцій, допомагає виявляти та виправляти помилки в коді шляхом перевірки синтаксису та підказок про можливі проблеми.

ESLint – це інструмент для перевірки та аналізу якості коду в проектах *JavaScript*. Він допомагає розробникам виявляти та виправляти потенційні проблеми з кодом, такі як синтаксичні помилки, стилістичні неузгодженості та загрози безпеки.

Vue VSCode Snippets – надає набір скорочень (*snippets*) для швидкого написання коду в проектах, які повсюди використовуються у *Vue.js* програмах.

Bracket Pair Colorizer – надає кольорове виділення та візуальну ідентифікацію парних дужок у коді, що значно спрощує пошук об’єкта при великій вкладеності синтаксичних конструкцій.

Наступний крок у розробці веб-кабінету є створення початкового середовища *Vue.js* проекту за допомогою *Vue CLI*.

Vue CLI – це потужний інструмент для швидкої і зручної розробки проектів *Vue.js*. Він спрощує налаштування, розробку та розгортання проектів і дозволяє розробникам фокусуватися на розробці функціональності своїх додатків. Ось декілька основних інструментів, які надаються *Vue CLI*: створення нових проектів з нуля або за допомогою попередньо налаштованих шаблонів через *@vue/cli*, швидке прототипування через *@vue/cli* та *@vue/cli-service-global*, можливість розробки з живим переглядом, що дозволяє вам бачити зміни в реальному часі без необхідності перезавантаження сторінки, підтримує гаряче перезавантаження компонентів, дозволяє додавати, оновлювати та видаляти залежності вашого проекту зручними командами в командному рядку та підтримує велику кількість корисних розширень.

Для початкової конфігурації проекту за допомогою *Vue CLI* потрібно завантажити *Node.js* для того, щоб мати можливість користуватися стандартним менеджером пакетів *NPM* (*node package manager*), якій в свою чергу надає змогу встановити *Vue CLI* для початку розробки проекту.

Після завантаження вище згаданих інструментів потрібно створити термінал у редакторі *VS Code* або через командний рядок *cmd.exe* виконати команду «*npm install -g @vue/cli*» для встановлення *Vue CLI*. Одразу після завантаження буде отримано доступ до команд та функцій системи, команда «*vue*» відображатиме список усіх доступних команд.

Після встановлення всіх необхідних утиліт, приступаємо до створення проекту, за допомогою команди «*vue create conferences-service*» у терміналі, яка складається з: ключового слова *Vue CLI* - «vue», команди створення нового проекту – «*create*» та «*conferences-service*» – назви проекту.

Після виконання команди у командному рядку буде можливість вибору налаштувань конфігурації проекту: за замовчуванням (*default*) або ручні (*manual*) налаштування. Веб-кабінет включає у себе кілька сторінок та обробку великої кількості даних, для таких вимог потрібно обрати ручні налаштування та зазначити у початковій конфігурації такі інструменти як:

1) *Babel* – це компілятор, інструмент для трансляції (компіляції) сучасного *JavaScript* коду в підтримуваний браузерами інтерпретований код *JavaScript*.

2) *Vue Router* – це офіційний маршрутизатор для розробки односторінкових додатків *SPA* (*Single-Page Applications*) з використанням фреймворку *Vue.js*. Він дозволяє вам організувати навігацію між сторінками вашого додатку і забезпечує динамічне змінення контенту без повної перезавантаження сторінки.

3) *Vuex* – це становий керівник (*state management pattern*) та бібліотека для управління станом додатків, побудованих з використанням фреймворку *Vue.js*. Він дозволяє ефективно управляти та спільно використовувати стан (дані) між різними компонентами вашого додатку.

4) *Linter* – перевіряє код на наявність синтаксичних помилок, невикористаних змінних, неправильного форматування і навіть потенційних помилок безпеки, а *Formatter* – забезпечує єдиний стиль форматування для всього коду в проекті.

Розглянемо структуру проекту зображену на рисунку 3.1 , яку створив *Vue CLI*.

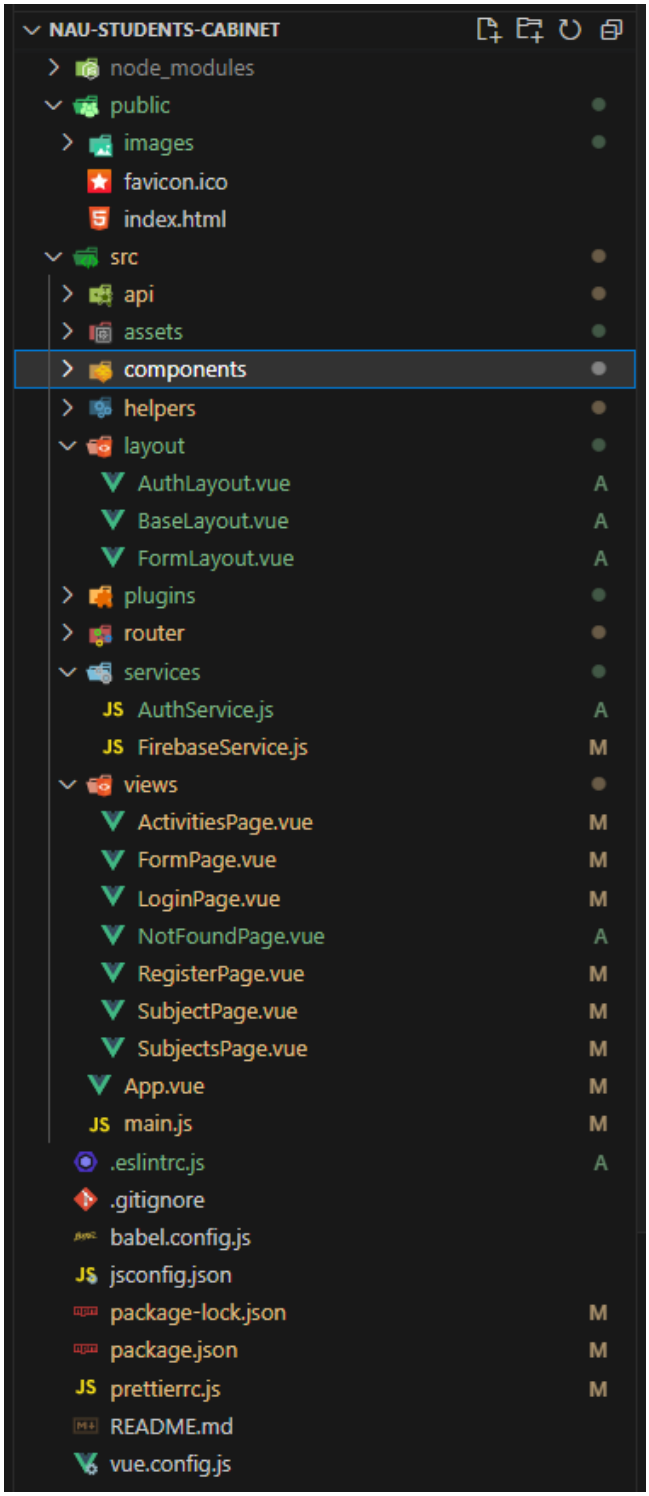

Рис. 3.1. Структура файлів проекту

Вибравши необхідні інструменти, потрібно визначити розташування конфігураційних налаштувань. Для проекту веб-кабінету студента було обрано розташування безпосередньо в одному файлі «*package.json*», тому що утримувати всі конфігураційні налаштування в одному файлі спрощує керування проектом. Ви можете легко знайти всі налаштування, що використовуються в проекті, і внести необхідні зміни, не шукаючи окремі файли конфігурації.

Папка з назвою «*node_modules*» є стандартною папкою у проектах, що використовують платформу *Node.js*. Вона містить усі залежності (модулі) проекту, встановлені за допомогою пакетного менеджера *npm*. Папка «*public*» містить статичні файли, які будуть доступні безпосередньо клієнтській стороні (браузеру) без необхідності обробки сервером: основний *HTML*-файл «*index.html»* та іконка вкладки онлайн-сервісу у веб-браузері «*favicon.ico*», які не будуть застосовані в процесі комплектування фінальної версії проекту. Файл «*.gitignore*» використовується в системі контролю версій *Git*. Він містить список файлів, папок, розширень або шаблонів, які *Git* повинен ігнорувати під час відстеження змін у репозиторії. Файл «*babel.config.js»* визначає, які плагіни та пресети *Babel* слід використовувати, а також інші параметри конфігурації. Файл «*package.json*» містить метадані про проект, список залежностей, скрипти складання та іншу інформацію, необхідну для управління проектом. Файл «*package-lock.json*» містить детальну інформацію про кожен встановлений пакет, включаючи його версію, хеш, залежності та іншу метаінформацію. Файл «*README.md*» містить документацію, опис та загальну інформацію про проект. Файл «*jsconfig.json*» – це файл конфігурації, який використовується в проектах, що базуються на *JavaScript*, для налаштування середовища розробки та визначення налаштувань компіляції. Конфігурація проекту має гнучкі налаштування, але при необхідності ми можемо змінити їх, написавши різні команди у «*vue.config.js*».

«*src*» є головною папкою проекту та вважається робочою областю розробника, де здійснюється основна робота з проекту. Ця папка має таку структуру:

– папка «*assets*», використовується для зберігання різних статичних ресурсів, таких як зображення, логотипи, шрифти, відео, аудіо, *CSS* стилі та інші

файли, які не є вихідним кодом, але використовуються у програмі та які будуть включені в процес комплектування фінальної версії проекту;

- папка «*components*» використовується для зберігання компонентів програми. Вона містить файли, які є незалежними і самодостатніми частинами функціональності програми;
- папка «*router*» містить файли, пов'язані з маршрутизацією або навігацією всередині програми;
- папка «*store*» містить конфігураційний файл для налаштування централізованого сховища (*Vuex*) проекту;
- папка «*views*» містить файли, які визначають компоненти, що відповідають за відображення даних та взаємодію з користувачем. Файл «*App.vue*» є кореневим компонентом програми, а «*main.js*» – це файл *JavaScript*, який створює поточні об'єкти *Vue.js* та визначає точку входу та основну конфігурацію програми.

3.2. Верстка веб-кабінету

Отож, докладно розібравшись із завантаженням головних інструментів й утиліт та переглянувши структуру файлів проекту, можна приступати до верстки сторінок, шаблонів та компонентів.

При розробці онлайн-сервісу основна увага приділяється його інтерфейсу, оскільки він повинен бути інтуїтивно зрозумілим користувачам, а також швидко завантажуватися з будь-яких пристроїв.

Верстка веб-кабінету студента складається з таких частин: верстки головних сторінок, верстки шаблонів та верстки допоміжних компонентів.

Верстка сайту відіграє ключову роль у створенні візуального та інтерфейсу користувача веб-сторінки. Вона відповідає за організацію та структурування контенту, стилізацію елементів та забезпечення правильного відображення на різних пристроях. Вона впливає на враження користувачів, зручність використання сайту та загальну ефективність веб-додатку.

Верстка сайту включає програмування структури та вмісту веб-сторінки за допомогою *HTML*, що містить заголовки, абзаци, списки, таблиці, форми та інші елементи контенту та додавання *CSS (Cascading Style Sheets)* – мови для опису та стилізації зовнішнього вигляду та оформлення елементів веб-сторінки.

Існує кілька видів верстки, кожен з яких має свої особливості та застосовується у різних ситуаціях: статична, таблична та блочна. Попри ці три найбільш поширені види вересни, сучасна розширена типологія відкриває ще декілька додаткових типів верстки сайту: резинова, семантична, адаптивна та гнучка.

Статична верстка – це підхід до створення веб-сторінок, при якому всі елементи та вміст сторінки містять фіксований розмір, який залишається незмінним незалежно від розмірів гаджета на якому відображається веб-сайт. Статична верстка може бути корисною для простих і невеликих веб-сторінок, таких як лендінг-пейджі, інформаційні брошури або статичні портфоліо. Однак для більш складних сайтів з динамічним вмістом та функціональністю зазвичай застосовуються інші типи верстки.

Таблична верстка – це вид верстки, при якому елементи сторінки організовані в таблицю з рядами та стовпцями, використовуючи *HTML*-елементи <*table*>, <*tr*> та <*td*>. У цьому підході кожен елемент та вміст розміщується в окремій частині таблиці, що дозволяє легко контролювати розташування та вирівнювання елементів на сторінці. Сьогодні цей підхід вважається застарілим, таблична верстка рідко використовується в сучасній веб-розробці, особливо для створення макетів та оформлення сайтів.

Блочна верстка – це найбільш поширений вид верстки серед розробників. Тут сітка створюється на основі *HTML*-тегів <*div*>, вкладених один в один. Щоб до них дістатися, верстальники використовують атрибути *id* й *class*. Переваги блокової верстки включають гнучкість адаптивності, легкість стилізації та хорошу семантику, що робить її кращим вибором для більшості веб-сайтів та програм.

Резинова верстка – тип верстки, при якому веб-сторінка розтягується на всю ширину браузера, яких би розмірів вона не набувала та на якому з гаджетів не відкривалася б. Ширина елементів веб-сторінки задається у відсотках від ширини

вікна браузера, і, таким чином, розтягується до країв вікна. Мінусом такої верстки є лише те, що сторінки по-різному виглядають на кожному гаджеті, хоча за шириною розтягнуті однаково.

Семантична верстка – є продовженням блочної верстки. Стала можливою завдяки *HTML5*. Нові теги роблять сторінку структурованою та логічно вмотивованою. Ідеально підходять для лендингів, які продають товари та послуги, або для сайтів-візитівок.

Адаптивна верстка – це підхід до створення веб-сторінок, який дозволяє сторінці автоматично адаптуватися до різних розмірів екранів та пристроїв, забезпечуючи оптимальне відображення та зручність використання.

Гнучка верстка – розпочинається з блочної верстки, а вже потім потрібні елементи перетворюються на гнучкі контейнери, за допомогою *CSS3* веб-модуля *Flexbox*. У стилях елемента вказується *display: flex*. Елементу можна задати напрямок головної вісі та вирівняти його. Такі чарівні маніпуляції зробили сторінки надгнучкими та комфортними для перегляду користувачами на всіх видах гаджетів.

Розглянувши вище зазначені види та типи верстки, для веб-кабінету було обрано використовувати семантичну верстку, адже такі сторінки дуже люблять пошукові браузери, адаптивну верстку для коректного відображення на планшетах та мобільних пристроях та гнучку верстку для зручного розташування елементів на сторінках.

Спочатку розробляються шаблони для компонентів, що повторюються, такі як: заголовок сайту «шапка» (*header*), «підвал» (*footer*), навігаційне меню (*navigation*) та інші, часто використовувані, елементи. Вони є важливими елементами веб-сторінки, які часто присутні на всіх сторінках сайту. Ці компоненти забезпечують навігацію сайтом, а також містять інформацію, яка повторюється на всіх сторінках.

Головний шаблон верстки (*BaseLayout*) має такий вигляд:

```
<div class="wrapper bg-grey">
  <Header />
  <main class="main">
    <slot />
```

```
    </main>
    <Footer />
  </div>
```

Шаблон створений для обгортки модулів та складається з: обгортки <*div class="wrapper"*>, шапки <*Header* />, підвалу <*Footer* />, а також основного змісту <*main*></*main*>, який містить спеціальний *Vue.js*-елемент <*slot*/> для зазначення місця розташування контенту при використанні шаблону. Для обгортки (*wrapper*) блока <*div*> задано такі *CSS*-стилі:

```
.wrapper {
    min-height: 100vh;
    display: flex;
    flex-direction: column;
}
.bg-grey {
    background-color: var(--grey-light);
}
```

В стилях задано мінімальну висоту контенту, створено контейнер за допомогою *Flexbox*, визначено напрямок, у якому розташовуються елементи у контейнері та задано колір фону.

Компонент заголовку сторінки (*header*) має такий *HTML*-код:

```
<header class="header">
    <div>
      <h1>
        <router-link :to="{ name: 'subjects' }"
          class="router-link-exact-active router-link-active">
          <img class="logo" src="../assets/nau-logo.png" alt="Subjects"/>
          </router-link>
      </h1>
    </div>
    <nav>
```

```
<router-link v-if="showReturnToSubjects" :to="{ name: 'subjects' }">
  &larr; Повернутися до списку
</router-link>
<router-link v-if="user" :to="{ name: 'subjects' }">Предмети</router-link>
<router-link v-if="user" :to="{ name: 'activities' }">
Позакласні заходи</router-link>
<router-link v-if="!user" :to="{ name: 'login' }">Вхід</router-link>
<router-link v-if="!user" :to="{ name: 'register' }">
 Реєстрація</router-link>
<router-link :to="{ name: 'create' }">Додати предмет</router-link>
<a v-if="user" @click="signOut">{{ user.displayName }} (Вихід)</a>
</nav>
</header>
```

«Шапка» сайту складається з: семантичного *HTML*-тегу <*header*>, логотип-посилання на головну сторінку веб-кабінету спроектований за допомогою компонента <*router-link*> та *HTML*-тегу, який використовується для вставлення зображень на веб-сторінці – <*img*/>; семантичний тег навігаційного меню <*nav*>, яке складається з посилань маршрутизатора на різні модулі веб-кабінету.

«Підвал» сторінки (*footer*) є розділом, розташованим внизу сторінки, та містить такий код:

```
<footer class="footer">
  <div class="container">
    © 2023. Maria Struk
  <div><a href="https://learn.javascript.ru/courses/vue">Project code</a></div>
  </div>
</footer>
```

Верстка нижньої частини сторінки містить: дату створення веб-кабінету, ім'я автора та посилання на базу коду веб-сервісу, яке створене за допомогою тега *HTML*-посилання <*a*>.

Компонент шаблону реєстрації та автентифікації (*AuthLayout*) має такий *HTML*-код:

```
<div class="page page_onboarding">
  <div class="container">
    <h1 class="page__title text-center">{{ title }}</h1>
    <slot></slot>
  </div>
</div>
```

Вище згаданий шаблон складається з: контейнеру * *, динамічного заголовку за допомогою *HTML*-тегу *<h1>*, який відображає заголовок залежно від модуля та спеціального *Vue.js*-елементу *<slot/>* для позначення розташування полів для входу та реєстрації користувача.

Для контейнеру * * з класом «*container*», створено такі *CSS*-стилі:

```
.container {
  max-width: 1008px;
  width: 100%;
  margin: 0 auto;
}
```

CSS-властивості контейнера задають максимальну ширину контенту (*max-width*), максимальне значення до якого елемент може розширюватися по горизонталі та за допомогою зовнішніх відступів (*margin*) центрує їх по середині сторінки.

Верстка шаблону-компоненту (*FormLayout*):

```
<div class="page page_subject-forms">
  <div class="container">
    <h2 class="page__title">{{ title }}</h2>
    <slot></slot>
  </div>
</div>
```

Компонент (*FormLayout*) був спеціально спроектований для обгортання модулів додавання і редагування навчальних дисциплін та створення загальних *CSS*-

стилів *HTML*-полів. Він складається з такої структури: контейнер <*div class="container"*>, динамічний заголовок та *Vue.js*-елемент <*slot/*> для зазначення розташування форми полів додавання та редагування пердметів. Логіка даної структури така сама, як в минулого шаблону, різниця тільки в стильових класах, які задаються внутрішні відступи контенту (*padding*).

Наступним етапом є верстка головних сторінок веб-кабінету, створено такі компоненти: *SubjectsPage*, *SubjectPage*, LoginPage, *RegisterPage* та *FormPage*.

Головна сторінка *SubjectsPage* спроектована, щоб містити у собі верстку демонстрації, пошуку, сортування та фільтрації навчальних предметів. Вона містить у своїй структурі такі елементи та теги: контейнер, фільтр-компоненту, текстового поле для пошуку предмету, елементу перемикача для зміни відображення предметів у вигляді календаря. Контейнер задає внутрішню ширину контенту, центрування та розташування. Фільтр-компонент – задає такий внутрішній елемент, який демонструє текстові радіо-кнопки за допомогою *HTML*-тегу <*input type="radio"/*>, щоб *JavaScript* зміг фільтрувати предмети. Пошук за назвою створений за допомогою текстового *HTML*-елементу <*input/*> та стилізований всередині зі зображенням «лупи», для кращого розуміння. Перемикач містить у собі дві гарно стилізовані кнопки для зміни звичайного списку предметів у вигляд календаря та навпаки. Сторінка результату верстки *SubjectsPage* зображено на рис. 3.2.

Для відображення афіши подій в університеті, спроектована сторінка *ActivitiesPage*. Вона містить у собі верстку демонстрації, пошуку, сортування та фільтрації позакласних заходів в університеті. За своєю структурою, повністю схожа на головну сторінку *SubjectsPage*. Вигляд сторінки результату верстки *ActivitiesPage* зображено на рис. 3.3.

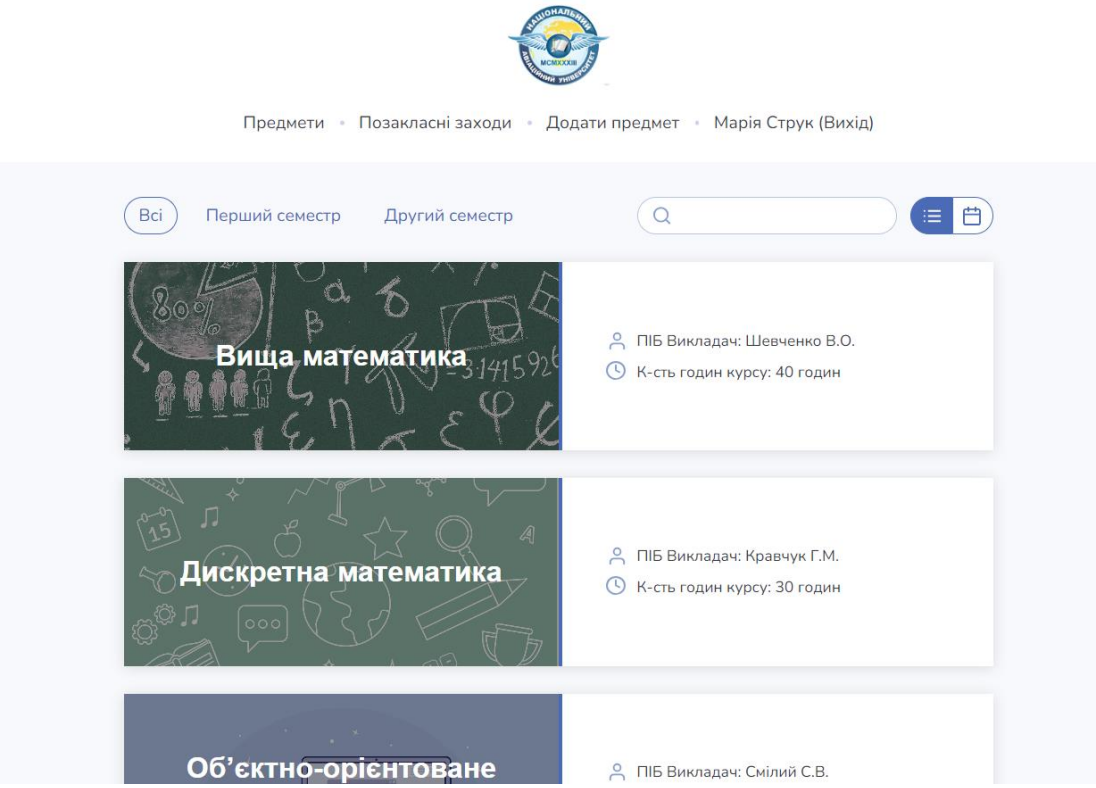

Рис. 3.2. Головна сторінка списку навчальних предметів

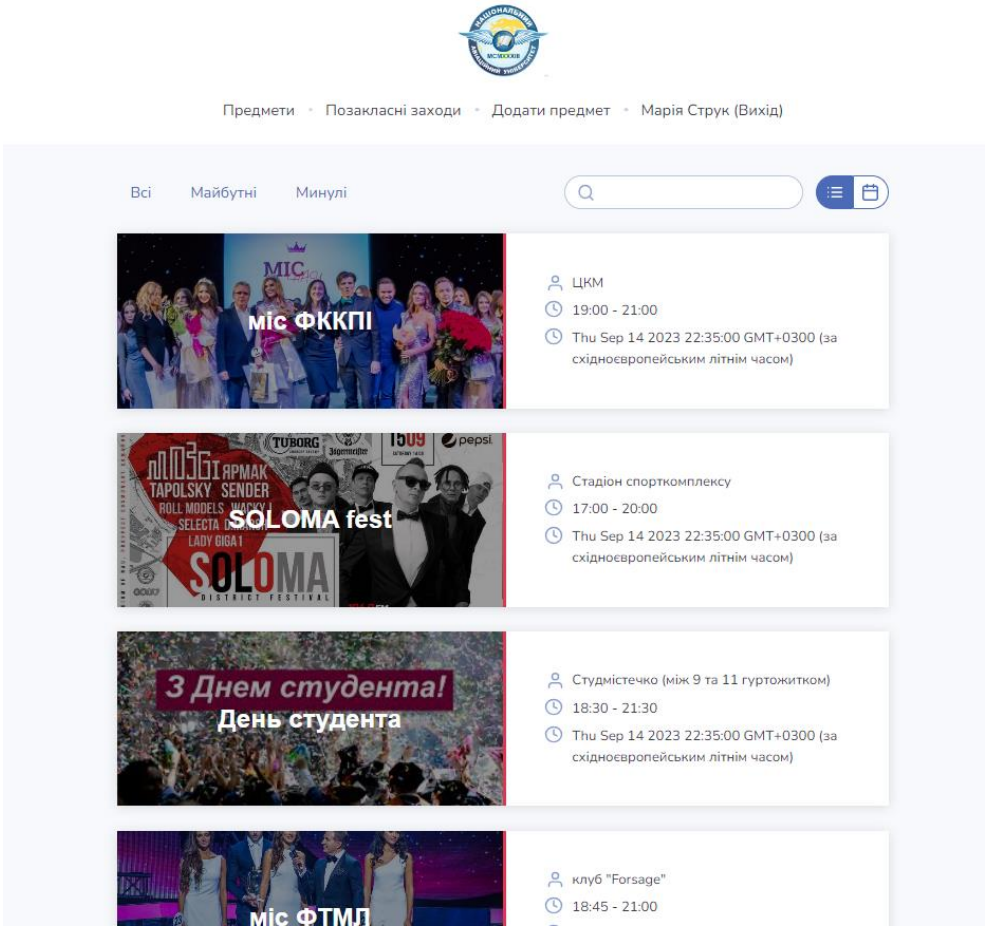

Рис. 3.3. Сторінка позакласних заходів в університеті

Сторінка розкладу предметів *SubjectsCalendar* спроектована, щоб відображати розклад предметів у вигляді календаря. Вона складається з верстки клітинок календаря та активних клітинок з відображенням кількості пар та їх часом. Календар був створений за допомогою CSS *Grid* – це спосіб створення сітки, яка відображає календарні дані з використанням *CSS* властивостей. *CSS Grid* дозволяє легко розмістити елементи на сторінці у формі таблиці з рядками і колонками. Було створено контейнер для календаря, * * та використано *CSS* властивість *display: grid;*, щоб вказати, що контейнер повинен використовувати сітку. Сітка складається з горизонтальних рядків, які визначаються властивістю *grid-template-rows* контейнера сітки. Рядки вказують висоту клітинок сітки. Сітка складається з вертикальних колонок, які визначаються властивістю *grid-template-columns* контейнера сітки. Колонки вказують ширину клітинок сітки. Властивості *grid-row* і *grid-column* встановлюють положення дочірніх елементів у сітці, вказуючи номери рядків і колонок, які вони займають. Властивість *gap* встановлює проміжок між рядками та колонками сітки, створюючи відступи між елементами. Сторінка результату *SubjectsCalendar* зображено на рис. 3.4.

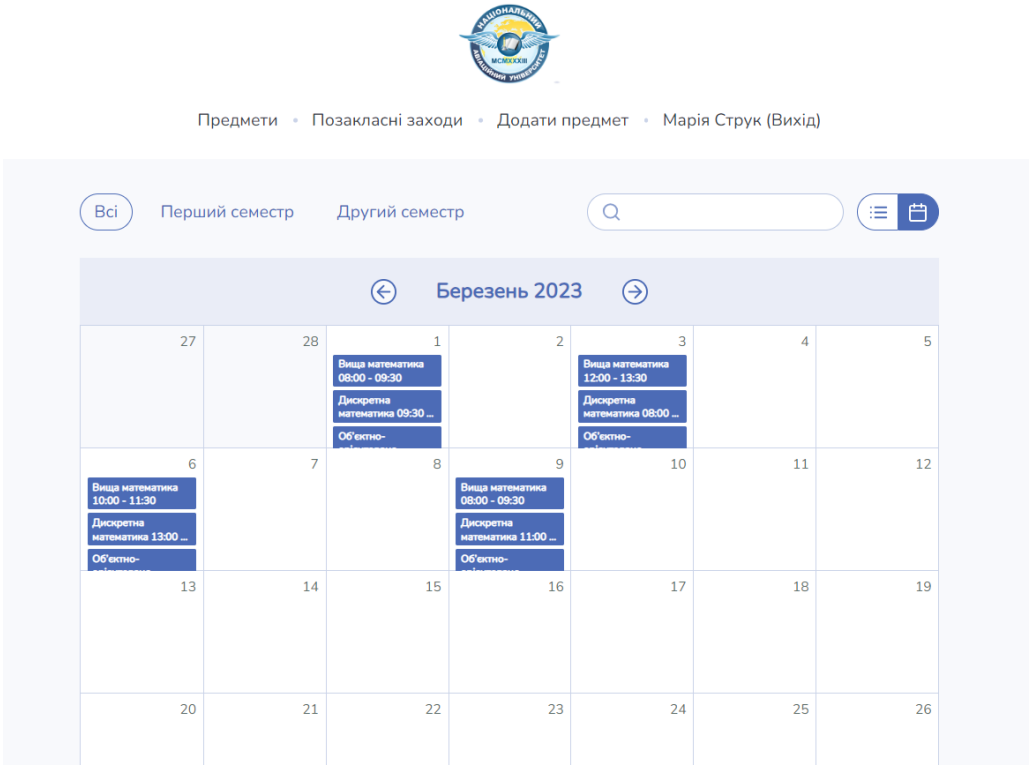

Рис. 3.4. Розклад предметів у вигляді календаря

SubjectPage – це сторінка створена для детального відображення повної, розгорнутої інформації про конкретну дисципліну та складається з таких частин: компонент фото-банер зображення предмету (*subject-cover*), компонент детального опису дисципліни (*subject-description*), програма (*subject-agenda*) та загальна інформація про предмет (*subject-info*), а саме прізвище викладача, корпус та аудиторію проведення занять.

Компонент фото-банер розміщує у собі зображення, яке розтягується повністю на весь блок <*div*> за допомогою *CSS*-стилів та заголовку <*h1*> із назвою предмету. За допомогою кнопок-перемикачів, можемо побачити компонент опису дисципліни та програму. Компонент детального опису містить тільки один параграф <*p*> у якому відображається детальний опис предмету. Програма – це блок з стилізованих елементів розгорнутої програми дисципліни з таким *HTML*-кодом: іконка-зображення відповідає кожному типу елемента програми <*img/*>, кількість відведених годин на тему відображено у тегу <*span*>, підзаголовок назви відповідної теми предмета в тегу <*h5*>. Загальна інформація по предмету відтворена за допомогою маркованого списку <*ul*>, якій містить елементи списку <*li*>. В елементах списку зазначено прізвище викладача, корпус, аудиторію проведення занять, а також в кожному елементі тематична іконка-зображення <*img/*>, яка відповідає кожному типу елемента. Сукупність елементів оброблювачів подій спроектований такими компонентами: *PrimaryButton, SecondaryButton, DangerButton,* які містять в собі *HTML*-тег <*button*> з відмінними *CSS*-стилями для наголошення сенсу оброблювача. Результат спроектованої сторінки *SubjectPage* зображено на рис. 3.5.

RegisterPage – це сторінка створення нового облікового запису користувача. Даний компонент використовує обгортку *AuthLayout* та використовує таку верстку: *HTML*-форму, поле для вводу *email*, поле *ім'я*, поле *пароль*, поле *повторного введення паролю, checkbox* елемент для погодження з умовами сервісу та *submit*-кнопку входу та посилання маршрутизатора на сторінку автентифікації (*LoginPage*).

Рис. 3.5. Сторінка опису програми предмету

Елемент <*form*> є одним з основних елементів *HTML*, які використовуються для створення інтерактивних форм на веб-сторінках. Він дозволяє користувачам вводити та надсилати дані на сервер для подальшої обробки. Тег <*input*>

призначений для створення інтерактивних полів введення на веб-сторінках. Цей елемент дозволяє користувачам вводити різні типи даних. Поле електронної адреси користувача створено за допомогою <*input*> з типом *email*: <*input type=”email”/*>. Поле імені користувача має тип за замовчуванням <*input type=”text”/*>, поле паролю з типом *password*: <*input type=”password”/*>, повтор паролю, також, має тип *password*, *checkbox* елемент – <*input type=”checkbox”/*>. Елемент <*button*> використовується для створення кнопок. Він призначений для виконання дій, коли користувач натискає кнопку. Кнопка зареєструватися реалізована у *HTML*: <*button type=”submit”*>, що дозволить обробити певну функцію при імплементації *JavaScript*. Результат сторінки зображено на рис. 3.6.

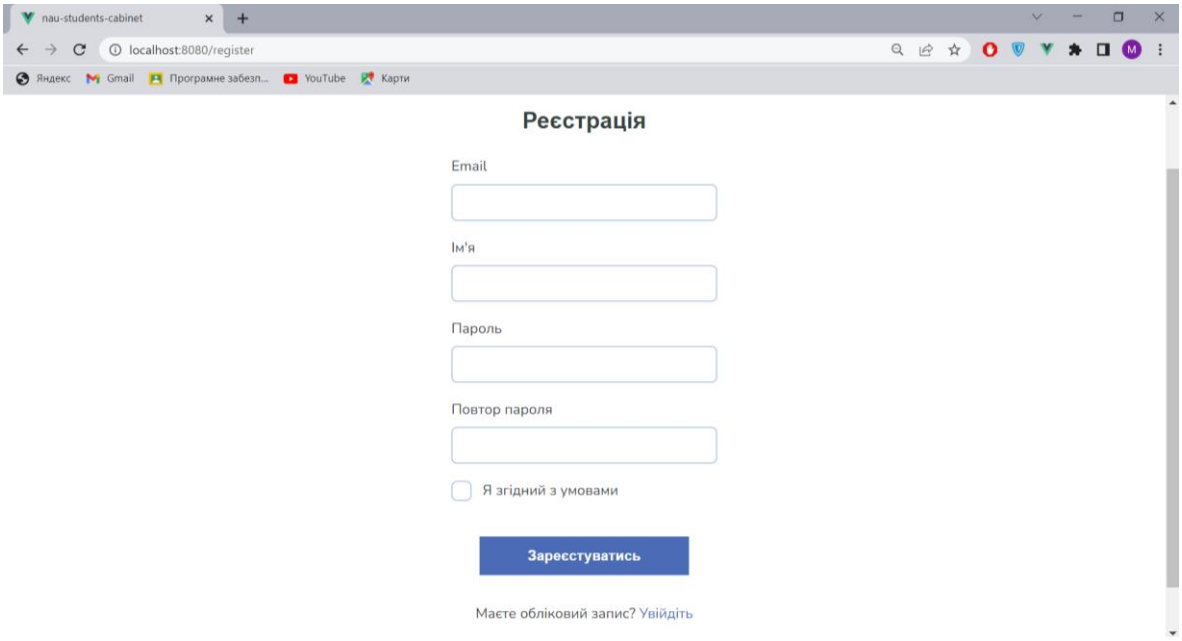

Рис. 3.6. Сторінка реєстрації

LoginPage – це сторінка, входу користувача у систему. Це процес підтвердження ідентичності користувача. Цей компонент має схожу будову до сторінки *RegisterPage* та містить таку верстку: *HTML*-форму, поле *email*, поле *пароль*, та *submit*-кнопку входу та маршрутизатора на сторінку реєстрації (*RegisterPage*). Ці перелічені елементи форми побудовані за допомогою тегу <*input/*>, з відмінними типами. Поле електронної адреси з типом *email*, а поле паролю з типом *password*.

Кнопка входу спроектована завдяки змозі задавати тип кнопки у *HTML <button type=”submit”>*. Результат *LoginPage* зображено на рис. 3.7.

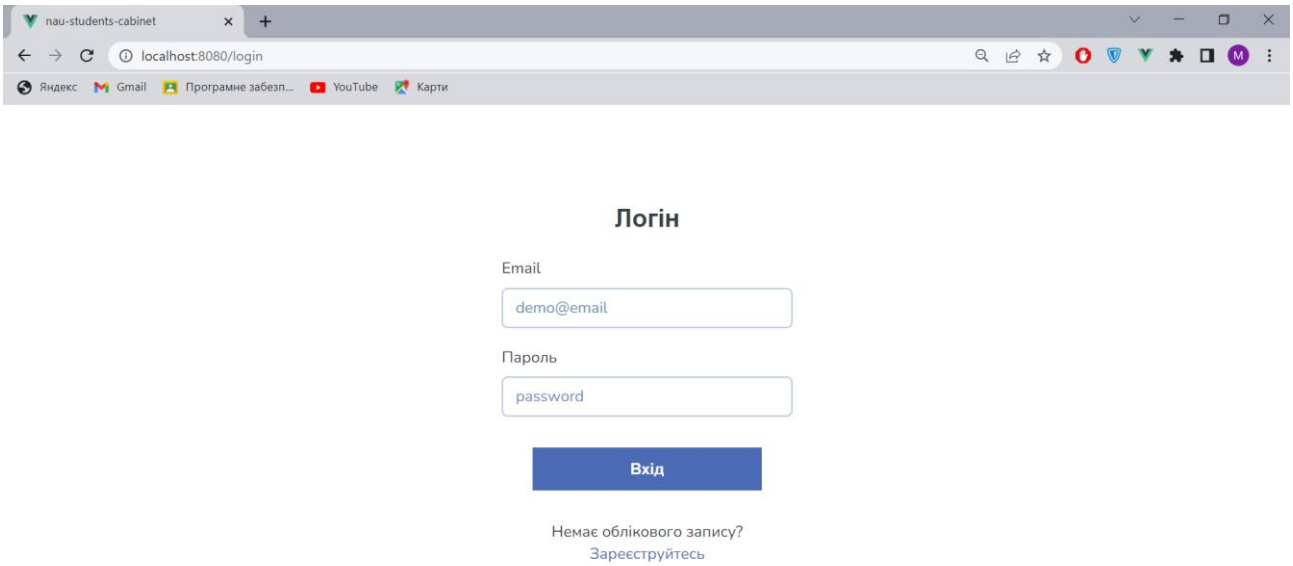

Рис. 3.7. Сторінка автентифікації

FormPage – ця сторінка була спроектована для створення та редагування предметів. *HTML*-код має таку структуру: обгортку *FormLayout* – форму, яка містить у собі пусті поля для внесення даних про предмет або заповнені для редагування певної дисципліни, а також спеціальний компонент *SubjectAgendaItemForm* для покрокового створення програми предмета. Форма реалізована елементом *HTML <form>*, який обгортає поля тегом *<fieldset>* для групування пов’язаних елементів в формі. Компонент містить текстове поле *<input/ type=”text”>* для внесення назви дисципліни, поле типу *date <input/ type=”date”>* – дата відповідного семестру, в якому буде прочитаний предмет, область для вводу декількох рядків тексту *<textarea>* та поле файлового типу для додавання зображення дисципліни *<input/ type=”file”>*. У компоненті *SubjectAgendaItemForm* основна структура це: випадаючий список *DropdownButton,* створений за допомого *HTML*-тегу *<select>*, для вибору типу занять (лекція чи практика), два часових поля *<input/ type=”time”>* для встановлення часу проведення занять, а також ще декілька другорядних полів, які мають тип за замовчуванням *<input type=”text”/>*. Кінцевою роботою створення

даної сторінки є реалізація секції кнопок <*button*> для відміни змін (кнопка «Скасувати») та збереження і створення предмету (кнопка «Створити»). Результат проектування *FormPage* зображено на рисунку 3.8.

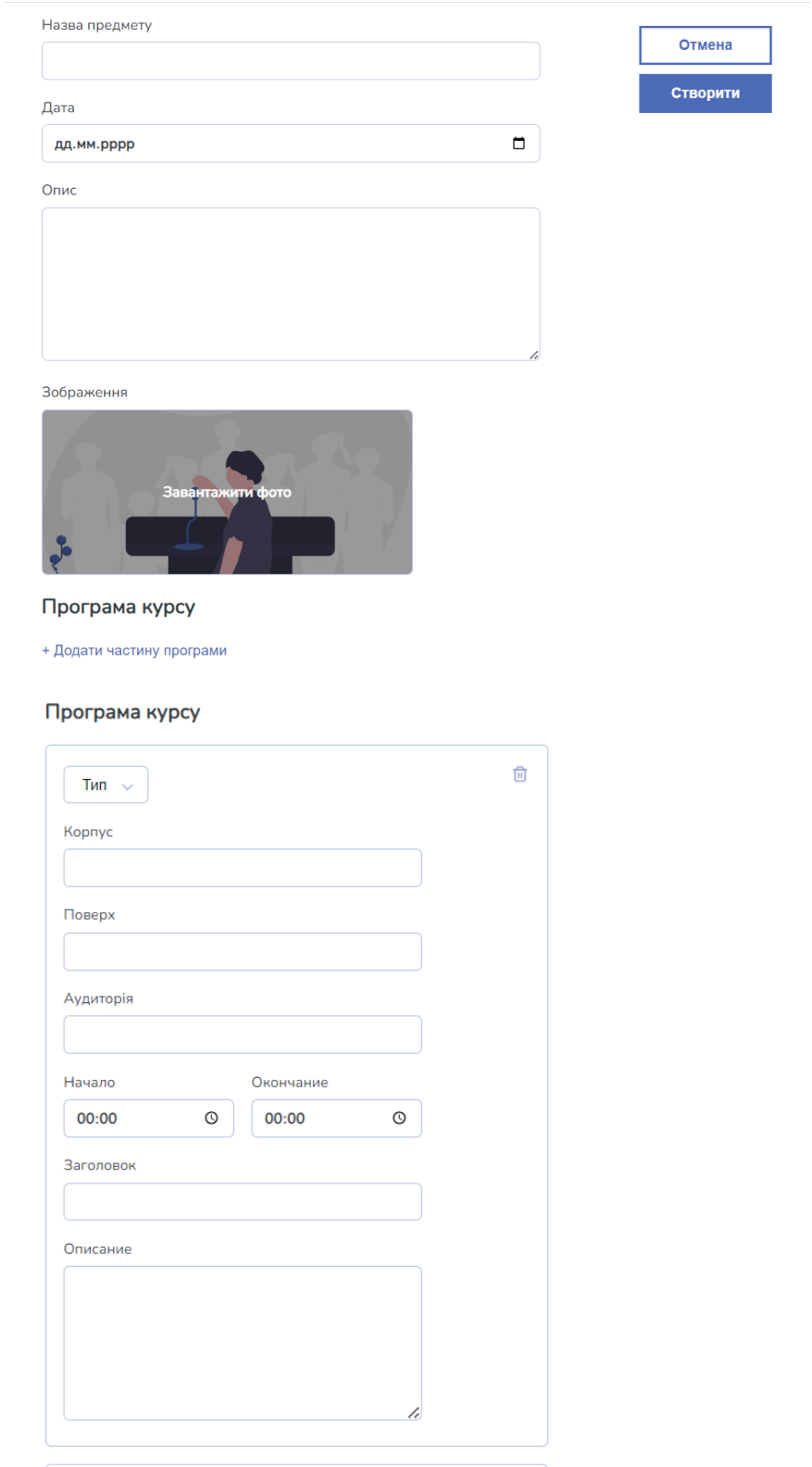

Рис. 3.8. Сторінка створення предмету

3.3. Модулі та їх функції веб-кабінету

Після того як було розроблено верстку веб-кабінету, відбувається процес перетворення концепції, головних функцій та специфікації в реальний робочий код, для встановлення зв'язку з базою даних *Firebase Realtime Database*, обробки подій, налаштування реєстрації та автентифікації *Firebase Authentication* та реалізація необхідних функцій.

Розглянувши у другому розділі клієнт-серверну архітектуру, прийшов час розповісти та переглянути детально, який архітектурний підхід був використаний, за допомогою якого інтерфейсу був створений зв’язок з сервером та які саме інструменти та компоненти використовувалися на стороні клієнті для реалізації цих завдань.

API (*Application Programming Interface*) – це набір правил і протоколів, які визначають, як різні програми або компоненти програмного забезпечення можуть взаємодіяти між собою. API дозволяє програмам комунікувати і обмінюватися даними без необхідності розкриття внутрішньої реалізації.

REST (*Representational State Transfer*) є архітектурним стилем, який використовується для проектування мережевих додатків. Він встановлює набір принципів та обмежень для створення легких, масштабованих та добре зрозумілих веб-сервісів. *REST* використовує протокол *HTTP* для здійснення комунікації між клієнтом і сервером. *HTTP* надає набір методів, статусів та заголовків, які використовуються в *REST* для взаємодії з ресурсами.

Ресурс – це інформаційний або функціональний елемент, до якого можна отримати доступ через інтернет. Ресурс може бути будь-яким ідентифікованим об'єктом (наприклад зображення), який можна маніпулювати або отримати даними.

Ресурси можуть бути представлені у різних форматах, таких як *JSON, XML, Atom, CSV* або *HTML*, залежно від вимог програми або клієнта. Клієнти можуть вказувати бажаний формат даних через заголовки запиту. *JSON* дозволяє передавати різні типи даних, включаючи рядки, числа, логічні значення, об'єкти та масиви. Він став популярним у веб-розробці через свою простоту та легкість використання. *JSON*

може бути легко оброблено та розібрано як на стороні сервера, так і на стороні клієнта.

API-інтерфейси *REST* надають доступ до ресурсів шляхом використання *URI* (*Uniform Resource Identifier*). Клієнти можуть здійснювати *HTTP* запити до відповідних *URI*, щоб отримати доступ до ресурсів і виконувати різні операції з ними, такі як отримання даних, створення нових ресурсів, оновлення існуючих ресурсів або видалення ресурсів.

REST API (або *RESTful API*) надає стандартизовані методи (*GET*, POST, *PUT*, *DELETE*) для отримання, створення, оновлення і видалення ресурсів. Кінцева точка (*endpoint*) – це специфічна URI, до якої можна зробити *HTTP* запит для отримання доступу до певного ресурсу або виконання певної операції над ресурсом. Приклад кінцевої точки для отримання наявних дисциплін з серверу: *https://subjects-service-default-rtdb.europe-west1.firebasedatabase.app/ subjects.json*.

Fetch API – це інтерфейс браузера, який надає можливість здійснення мережевих запитів у *JavaScript* побудований на основі *Rest API*. Інтерфейс забезпечує узагальнене визначення таких об'єктів: *Request* та *Response*.

При проектуванні функцій для запитів з клієнту, було вирішено використовувати *Axios* – це бібліотека *JavaScript*, яка надає простий і зручний спосіб виконувати *HTTP* запити з використанням обіцянок (*promises*) у браузері або на серверній стороні (з використанням *Node.js*).

На стороні клієнта використовується *Vuex* – це бібліотека керування станом для *Vue.js*, яка дозволяє зберігати та управляти даними, що використовуються в додатку. Вона забезпечує централізований контейнер стану, до якого можуть отримувати доступ всі компоненти додатку.

Функція *signUp* – це асинхронна функція (*action*) в *Vuex*, яка містить такий код:

```
import {getAuth, createUserWithEmailAndPassword } from 'firebase/auth';
  async register(context, email, password) {
  createUserWithEmailAndPassword(getAuth(), email, password)
   .then(() => {
    SuccessMessage('Користувач успішно зареєстрований!')
```

```
        this.$router.push({ name: 'login'})
      })
      .catch((error) => ErrorMessage(error))
    }
```

Перед ти необхідно імпортувати такі функції як: *getAuth, createUserWithEmailAndPassword* з модуля *firebase/auth*, щоб користувач був зареєстрований у базі даних. Функція *signUp* має такі три параметри: *context, email, password*. Параметр *context* – це об'єкт, який використовується для отримання доступу до контексту модуля *Vuex*, *email* та *password* – це поля електронної адреси та паролю. Функція спрацьовує, коли натискається оброблювач подій під назвою «Зареєструватись» на сторінці реєстрації (*RegisterPage*). Реалізація цієї функції створена за допомогою двох методів, які ми отримуємо після встановлення *npm*-пакету «*firebase*» та його налаштування, а саме такі методи як: *getAuth* та *createUserWithEmailAndPassword*. Метод *createUserWithEmailAndPassword* приймає в себе три параметри: *callback*-функцію *getAuth*, *email* та *пароль*, що дозволяє додати користувача до бази даних, якщо було пройдено валідацію.

Функція *signIn* – це асинхронна функція (*action*), яка використана для обробки процесу входу користувача та складається з такого коду:

```
import {getAuth, signInWithEmailAndPassword } from 'firebase/auth';
    async register(context, email, password) {
    signInWithEmailAndPassword(getAuth(), email, password)
      .then(() => {
        SuccessMessage('Користувач успішно авторизувався!')
        this.$router.push({ name: 'meetups'})
      })
    }
```

Основним завданням функції *signIn* є перевірка введених користувачем даних, таких як електронна адреса та пароль, та здійснення відповідних дій, таких як встановлення статусу входу в систему або збереження аутентифікаційних токенів. Функція спрацьовує, коли користувач системи натискає «Увійти» на сторінці

автентифікації (*LoginPage*) та виконується функція *signIn* з дуже схожими методами як у *signUp*, різниця в тому, що замість метода *createUserWithEmailAndPassword*, імпортується метод *signInWithEmailAndPassword* та передані такі самі аргументи як при реєстрації, після чого відправиться запит на сервер, який перевірить чи є введені користувачем дані у базі даних.

Функція *getSubjects* – це асинхронна функція (*action*) має такий вигляд:

```
async getSubjects(context) {
state.isLoadingSubjects = true;
   this.$axios("https://subjects-service-default-rtdb.europe-
west1.firebasedatabase.app/ subjects.json")
  .then(subjects => {
     context.commit('SET_SUBJECTS', subjects);
     state.isLoadingSubjects = false;
   })
  .catch(error => {
     ErrorMessage(error);
   })
}
```

Функція містить лише один параметр *context*, який дозволяє нам отримати доступ до контексту модуля *Vuex*. Функція спілкується з сервером для отримання списку предметів з бази даних, щоб *HTTP*-клієнт *Axios* за допомогою метода *GET* отримав усі наявні предмети, якщо запит буде успішний, то дисципліни збережуться у *Vuex*-сховище за допомогою *Vuex*-мутації *SET_SUBJECTS* та вимкне прапорець *isLoadingSubjects*, якій відповідає за відображення стану завантаження предметів. Якщо запит виявиться невдалим, тоді з'явиться компонент *ErrorMessage*, який сповістить, що спроба була невдала. Компонент *SubjectsPage* за допомогою спеціального метода *Vuex – dispatch,* буде викликати функцію *getSubjects* в хуку життєвого циклу компонента *created*, щоб отримати предмети ще до повного створення браузерного *DOM*-дерева та відтворити їх, відразу коли дерево буде готове та сторінка відображена.

Функція *getSubject* – це асинхронна функція (*action*) має таку будову:

```
async getSubject(context, subjectId) {
state.isLoadingSubject = true;
   this.$axios(“https://subjects-service-default-rtdb.europe-
west1.firebasedatabase.app/subjects/subjectId(ідентифікатор_предмету).json”)
   .then(subject => {
      context.commit('SET_SUBJECT', subject);
      state.isLoadingSubject = false;
   })
   .catch(error => {
      ErrorMessage(error);
   })
}
```

Функція містить в собі два параметри: *context* та *id* (ідентифікатор предмету). Роль параметру *context* описано у функції *getSubjects*, а *id* допомагає отримати одну відповідний навчальний предмет. *Axios* відправляє *GET*-запит на *“https://subjects-service-default-rtdb.europe-west1.firebasedatabase.app/subjects/subjectId(ідентифікатор_предмету).json”*, якщо запит успішний, то отриманий об’єкт предмету (*subject)* збережеться у *Vuex*-сховище та вимкне прапорець *isLoadingSubject*, який відповідає за відображення стану завантаження предмету. Якщо запит буде невдалий, то *ErrorMessage* покаже помилку. Ця функція запускається під час натискання на конкретний предмет у списку на сторінці *SubjectsPage*, тоді модуль *SubjectPage* отримає конкретну дисципліну зі сховища *Vuex* та відобразить його.

Наступна функція *createSubject,* має такий код:

```
async createSubject(context, newSubject) {
   this.$axios.post(“https://subjects-service-default-rtdb.europe-
west1.firebasedatabase.app/subjects.json”, newSubject)
   .then(successMsg => {
      SuccessMessage(successMsg.text)
```

```
        state.isLoadingSubject = false;
      })
      .catch(error => {
        ErrorMessage(error);
      })
    }
```

Вище наведена функція створює предмет, та має такі параметри: *context* та *newSubject.* З параметром *context* знайомі з попередніх функцій, а *newSubject* – це об'єкт нового предмету, який буде відправлений на сервер та записаний у базу даних. На сторінці *FormPage*, користувач додає предмет та натискає на кнопку «Створити» для виклику функції *createSubject* з *Vuex*-сховища, яка приймає аргументом об'єкт нового предмету та надсилає *POST*-запит на адресу "*https://subjects-service-default-rtdb.europe-west1.firebasedatabase.app/subjects.json"* за допомогою *Axios*. При успішному запиті, ми отримає відповідь від сервера, що новий предмет був успішно доданий у базу даних, за допомогою компонента *SuccessMessage*, а у разі помилки побачимо *ErrorMessage*. Після успішної операції, при переході на головну сторінку (*SubjectsPage*) з'явиться доданий предмет у списку, тому що функція *getSubjects* відпрацює знову, так як вхідні дані зміняться.

Функція *deleteSubject* – це асинхронна функція (*action*), складається з такого коду:

```
async deleteSubject(context, deletedSubject) {
    this.$axios.delete("https://subjects-service-default-rtdb.europe-
west1.firebasedatabase.app/subjects/id(ідентифікатор_предмету).json",
deletedMeetup)
      .then(successMsg => {
        SuccessMessage(successMsg.text)
        state.isLoadingSubject = false;
      })
      .catch(error => {
        ErrorMessage(error);
```

```
    })
}
```

Дана функція видаляє предмет та доступний лише викладачу, якій має право видалити дисципліну за допомогою кнопки під назвою «Видалити», що дозволяє *Axios* виконати *DELETE*-запит за адресою "*https://subjects-service-default-rtdb.europe-west1.firebasedatabase.app/subjects/id(ідентифікатор_ предмету).json*" та видалити конкретний предмет з бази даних, що спричинить повторне виконання функції *getSubjects* та оновлення головної сторінки зі списком дисциплін.

3.4. Висновок до розділу 3

Розробка веб-кабінету трьох етапів: вибір та налаштування середовища розробки, створення початкового *Vue.js* середовища та завантаження основних програм для розробки.

На першому етапі було обрано та налаштовано інтегроване середовище розробки, де відбувається створення, редагування та налагодження програмного коду. Для проектування веб-кабінету було обрано *Microsoft Visual Studio Code*.

Наступним кроком у розробці веб-кабінету було створення початкового середовища *Vue.js* проекту за допомогою *Vue CLI* та налаштовано під потреби розробки. Він спрощує налаштування, розробку та розгортання проектів і дозволяє розробникам фокусуватися на розробці функціональності своїх додатків. Для початкової конфігурації проекту за допомогою *Vue CLI* було завантажено *Node.js* для того, щоб мати можливість користуватися стандартним менеджером пакетів *NPM,* якій в свою чергу надає змогу встановити *Vue CLI* для початку розробки проекту.

Отож, докладно розібравшись із завантаженням головних інструментів й утиліт, було виконано верстку сторінок, шаблонів та компонентів.

Верстка веб-кабінету студента складається з таких частин: верстки головних сторінок, верстки шаблонів та верстки допоміжних компонентів.

Верстка сайту відіграє ключову роль у створенні візуального та інтерфейсу користувача веб-сторінки. Вона відповідає за організацію та структурування

контенту, стилізацію елементів та забезпечення правильного відображення на різних пристроях.

Було розроблено головні сторінки програми: *RegisterPage* сторінка для реєстрації нового користувача; *LoginPage* − сторінка, яка відображає поля для входу в профіль користувача; *SubjectsPage* для відображення, сортування, пошуку та фільтрації предметів; *SubjectPage* для детального відображення опису та інформації про конкретний предмет; *FormPage* − сторінка для створення та додавання предметів.

Після етапу розробки верстки веб-кабінету студента, відбулася імплементація головних функцій для встановлення взаємозв'язку з базою даних *Firebase Realtime Database*, обробки подій, налаштування реєстрації й автентифікації *Firebase Authentication* та реалізація допоміжних функцій. Проект було спроектовано з чітким дотриманням діючих стандартів та положень.

ВИСНОВКИ

Кожного дня ми користуємося інформацію з всесвітньої мережі, у зв’язку з цим виникла можливість поліпшення навчального процесу у вищих навчальних закладах за допомогою веб-технологій. Дізнатися та переглянути більш детальну інформацію про кожну з дисциплін, розклад занять та календар всіх подій в університеті можна в електронному кабінеті студента, який призначений спростити процеси, що напряму пов’язані зі студентами, їх навантаженням та успішністю. Існуючі інструменти веб-розробника, та технології програмування для веб докладно описані в розділах. Всі вони мають свої недоліки та переваги, тому для усунення цього доводиться користуватися декількома технологіями одночасно.

Веб-сервер – це програмне забезпечення або серверний комп'ютер, який надає послуги передачі веб-сторінок та інших ресурсів через Інтернет. Він відповідає на запити веб-клієнтів і передає їм веб-сторінки або інші ресурси, що були запитані.

Основна функція веб-сервера полягає в обробці *HTTP*-запитів, які надходять від клієнтів. Коли веб-браузер користувача відправляє запит на отримання певної веб-сторінки, веб-сервер обробляє цей запит і відправляє відповідь, яка містить *HTML*-код сторінки. Веб-сервер може також обробляти інші типи ресурсів, такі як зображення, статичні файли, відео, аудіо тощо.

Всесвітня павутина (*World Wide Web*) побудована з використанням таких ключових технологій:

- *HTTP*: *HTTP* є протоколом передачі даних, який використовується для комунікації між веб-клієнтами і веб-серверами. Він визначає правила для запитів та відповідей, що передаються між ними.
- *HTML*: *HTML* використовується для створення структури та розмітки веб-сторінок. Він визначає елементи та їхню взаємодію, включаючи текст, зображення, посилання та інші елементи.

– *URL*: *URL* - це адреса, за допомогою якої можна ідентифікувати ресурси в Інтернеті. Вона включає протокол, доменне ім'я, шлях до ресурсу та, за потреби, параметри запиту.

Дізнатися та переглянути більш детальну інформацію про кожну з дисциплін, розклад занять та календар всіх подій в університеті можна в електронному кабінеті студента, який призначений спростити процеси, що напряму пов’язані зі студентами, їх навантаженням та успішністю.

Веб-кабінети студента часто надаються навчальними закладами для студентів з метою забезпечення доступу до різноманітної інформації та функціональності. Основні задачі, які можуть бути вирішені у веб-кабінеті студента, включають:

1. Розклад занять: студенти можуть переглядати свій індивідуальний розклад занять, включаючи час, місце та викладача. Це допомагає студентам планувати свій час та бути в курсі графіка навчальних занять.

2. Завдання та домашні завдання: веб-кабінет може містити розділ для перегляду та здачі домашніх завдань або завдань, призначених викладачем. Студенти можуть завантажувати роботи, переглядати вимоги до завдань та отримувати повідомлення про строк здачі.

3. Комунікація з викладачами та спілкування: веб-кабінет може включати інструменти для комунікації з викладачами та іншими студентами. Це можуть бути електронна пошта, форуми, чати або системи обміну повідомленнями, які дозволяють студентам задавати питання.

4. Отримання оперативної інформації щодо освітнього процесу.

5. Керування особистими даними: студенти можуть оновлювати свої контактні дані, адресу, номери телефонів тощо, щоб забезпечити актуальність інформації про себе.

6. Доступ до навчальних матеріалів: студентам надається можливість завантажувати або отримувати доступ до навчальних матеріалів, які стосуються їхніх курсів або предметів.

Конкретний функціонал та задачі існуючих веб-кабінетів студента можуть варіюватися в залежності від університету або навчального закладу.

Головною метою є − проектування онлайн-сервісу, який включає в себе необхідні функції для організації та автоматизації навчального процесу з мінімальними витратами часу і використанням людського ресурсу. Також важливим є зручність використання даного сервісу організаторами і користувачами на всіх етапах, які призначені спростити процеси, що напряму пов’язані зі студентами, їх навантаженням та успішністю.

Клієнт-серверна архітектура (*англ. client-server architecture*) є одним з основних підходів до розподіленої обробки і обміну даними між комп'ютерами у мережі. В основі цієї архітектури лежать два компоненти: клієнт і сервер.

Клієнт і сервер спілкуються між собою через мережу, таку як Інтернет або локальну мережу. Модель такої системи полягає в тому, що клієнт надсилає запит на сервер, де він обробляється, і готовий результат відправляється клієнтові. Сервер може обслуговувати кілька клієнтів одночасно. Якщо одночасно приходить більше одного запиту, то вони встановлюються в чергу і виконуються сервером послідовно. Іноді запити можуть мати пріоритети. Запити з більш високими пріоритетами повинні виконуватися раніше. Весь цей процес відбувається за допомогою певного протоколу комунікації, наприклад *HTTP*.

При проектуванні веб-кабінету студента був використаний модульний підхід *SPA*. Додаток розбивається на незалежні модулі, які виконують конкретні функціональні завдання.

Також, було використано *JavaScript* (*JS*) – це динамічна мова програмування, яку використовують для написання фронтенд та бекенд-частин сайтів, а також мобільних додатків. Мова відповідає за інтерактивність та функціонал інтерфейсу, який неможливо виконати лише за допомогою *HTML* та *CSS*.

Переважно бібліотеки та фреймворки *JavaScript* забезпечують прозору структуру програми та реалізуються на одному з двох найпопулярніших шаблонів дизайну розробки веб-додатків *MVC* (*Model-ViewController*) та *Model-View-ViewModel* (*MVVM*).

Firebase Realtime Database – це хмарна база даних типу *NoSQL*, яка використовує веб-сокети, що дозволяє клієнту отримувати інформацію в реальному

часі – без необхідності відправляти запити *GET* на сервер. Веб-сокет (*WebSocket*) – це протокол зв'язку між клієнтом та сервером, який дозволяє встановити постійне двостороннє з'єднання між ними. Він використовується для обміну даними в реальному часі та надає більш ефективний та масштабований спосіб спілкування порівняно з традиційними *HTTP*-запитами. Цей сервіс дозволяє розробникам зберігати та синхронізувати дані між різними клієнтськими додатками в режимі реального часу.

При створенні веб-кабінету студента багато уваги приділяється питанням безпеки. Створювати систему аутентифікації кожен раз з нуля досить затратно, причому витрати ці найчастіше невиправдані. Впоратися з більшістю викликів дозволяє система аутентифікації *Firebase Authentication* – це сервіс, який надається *Firebase* для аутентифікації користувачів у мобільних та веб-додатках. Він дозволяє розробникам забезпечувати безпечну та просту аутентифікацію користувачів, щоб вони могли входити в систему, реєструватися та керувати своїми обліковими записами.

Однією з основних частин створення онлайн-сервісу є чіткий опис задачі та проектування згідно вимог, без цього просто неможливо створити хороший та якісний продукт. Проаналізувавши різні веб-кабінети студента, був спроектований онлайн-сервіс, який має мету максимально спрощувати і автоматизувати процеси, які напряму пов’язані зі студентами. Кожен з модулів онлайн-сервісу містить в собі набір функцій та цілей для вирішення відповідних завдань. Схематично всі завдання веб-сервісу можна розділити на сім модулів: реєстрація, автентифікація, головна сторінка списку навчальних предметів, сторінка позакласних заходів в університеті, додавання предметів, сторінка детального опису програми предмету та профіль користувача.

Було розроблено головні сторінки програми: *RegisterPage* сторінка для реєстрації нового користувача; *LoginPage* − сторінка, яка відображає поля для входу в профіль користувача; *SubjectsPage* для відображення, сортування, пошуку та фільтрації предметів; *SubjectPage* для детального відображення опису та інформації про конкретний предмет; *FormPage* − сторінка для створення та додавання предметів. Проект було спроектовано з чітким дотриманням діючих стандартів та положень.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАННИХ ДЖЕРЕЛ

1. Мельник Р. А. Програмування веб-застосувань (фронт-енд та бек-енд) / Р. А. Мельник ; – М. : «Видавництво Львівської політехніки», 2018. – 248 с.

2. Робсон Е. Head First HTML and CSS / Робсон, Е., Фрімен, Е. ; – O'Reilly Media: Sebastopol, CA, 2014. – 720 с. – Електрон. аналог друк. вид.: режим доступу: http://artsites.ucsc.edu/sdaniel/170a_2014/Head_First_HTML_CSS_XHTML.pdf (дата звернення 05.05.2023) – Назва з екрана.

3. Шклар Л. Архітектура web додатків: принципи, протоколи та практики / Шклар, Л., Розен, Р. ; – John Wiley & Sons Ltd. – England, 2013. – 374 с. – Електрон. аналог друк. вид.: режим доступу: http://bedford-computing.co.uk/learning/wp-content/uploads/2016/07/Web-Application-Architecture-Principles-Protocols-and-Practices.pdf (дата звернення: 16.05.2023) – Назва з екрана.

4. Дакетт Дж. Web дизайн із набором HTML, CSS, JavaScript і jQuery / Дж. Дакетт ; John Wiley & Sons, Inc., – Indianapolis, 2017. – 514 с. – Електрон. аналог друк. вид.: режим доступу: https://d-pdf.com/book/2539/read (дата звернення: 01.05.2023) – Назва з екрана.

5. Маркотт І. Адаптивний web дизайн / І. Маркотт ; – Нью Йорк : A Book Apart, 2014. – 150 с.

6. С. Макконнелл Совершенный код. Мастер-класс. Пер. с англ. / С. Макконнелл ; – СПб. : «БХВ», 2022. – 896 с.

7. Vue.js Documentation [Електронний ресурс]: режим доступу: https://vuejs.org/v2/guide/ (дата звернення 23.04.2023). – Назва з екрана.

8. Firebase Documentation [Електронний ресурс]: режим доступу: https://firebase.google.com/docs (дата звернення 20.04.2023). – Назва з екрана.

9. Vuex Documentation [Електронний ресурс]: режим доступу: https://vuex.vuejs.org/ (дата звернення 22.04.2023). – Назва з екрана.

10. Learn.javascript [Електронний ресурс]: режим доступу: https://learn.javascript.ru/ (дата звернення 26.04.2023). – Назва з екрана.

11. w3schools [Електронний ресурс]: режим доступу: https://www.w3schools.com/ (дата звернення 13.05.2023). – Назва з екрана.

12. Developer Mozilla documentation [Електронний ресурс]: режим доступу: https://developer.mozilla.org/ (дата звернення 21.05.2023). – Назва з екрана.

13. J. Chaffer Learning jQuery / J. Chaffer, K. Swedberg ; Packt Publishing Ltd. – Livery Place 35 Livery Street Birmingham B3 2PB, UK : 2013. – 444 с. – Електрон. аналог друк. вид.: режим доступу: https://it.dru.ac.th/o-bookcs/pdfs/28.pdf (дата звернення: 07.05.2023) – Назва з екрана.

14. Інтегровані інтелектуальні робототехнічні комплекси (ІІРТК-2023). Шістнадцята міжнародна науково-практична конференція 23-24 травня 2023 р., Київ, Україна. – К. : НАУ, 2023. – 402 с. (збірка тез).

15. Босько В. В. Web- програмування. Частина 1 (frontend) : навч. посіб. / В. В. Босько, Л. В. Константинова, К. М. Марченко, О. С. Улічев ; – Кропивницький : ЦНТУ, 2022. – 208 с.

16. Грицюк Ю. І. Аналіз вимог до програмного забезпечення / Ю. І. Грицюк ; – Львів : Вид-во НУ «Львівська політехніка», 2018. – 456 с.