ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «РОЗРОБКА ВЕБ-ПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ОПИТУВАНЬ СТУДЕНТІВ З ВИКОРИСТАННЯМ БІБЛІОТЕКИ React. JS»

Виконав: студент 4 курсу, групи ПД–43
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Шевчук Ю.О.
(прізвище та ініціали)
Керівник Залива В.В.
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Напрям підготовки - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
О.В. Негоденко
“ ” 2023 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ
ШЕВЧУКУ ЮРІЮ ОЛЕКСАНДРОВИЧУ
(прізвище, ім‘я, по батькові)

1. Тема роботи:«РОЗРОБКА ВЕБ-ПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ОПИТУВАНЬ СТУДЕНТІВ З ВИКОРИСТАННЯМ БІБЛІОТЕКИ React.JS»
Керівник роботи Залива В.В.
(прізвище, ім‘я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від “ 24 ” лютого 2023 року №26

2. Строк подання студентом роботи “ 1 ” червня 2023 року

3. Вхідні дані до роботи:

3.1 Науково-технічна література, пов’язана з розробкою Веб-платформи
3.2 Технічна документація баз даних та комунікації між системами.

4. Зміст розрахунково-пояснювальної записки(перелік питань які потрібно розробити).
4.1 Аналіз та огляд літератури
4.1.1 Огляд існуючих систем для опитувань студентів
4.1.2 Аналіз особливостей бібліотеки React.js
4.1.3 Огляд архітектури Flux та її переваги
4.2 Проектування архітектури системи
4.2.1 Вибір архітектурного підходу
4.2.2 Діаграма компонентів системи

4.2.3 Діаграма послідовності взаємодії компонентів
4.3 Розробка фронтенду з використанням бібліотеки React.js
4.3.1 Структура проекту
4.3.2 Розробка компонентів для опитувань
4.3.3 Управління станом додатка з використанням Flux
4.4 Тестування та валідація розробленої платформи

5. Перелік демонстраційного матеріалу
5.1 Мета, об’єкт та предмет дослідження
5.2 Аналіз Аналогів
5.3 Вимоги до програмного забезпечення
5.4 Засоби реалізаціїї
5.5 Схема бази даних
5.6 Екранні форми
5.7 Висновки

6. Дата видачі завдання “ 25 ” лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	25.02.23-27.02.23	Виконано
2	Аналіз та дослідження існуючих аналогів	28.02.23-10.03.23	Виконано
3	Дослідження програмних засобів	13.03.23-24.03.23	Виконано
4	Моделювання об’єкту проектування	27.03.23-28.04.23	Виконано
5	Розробка функціоналу роботи	01.05.23-05.05.23	Виконано
6	Вступ,висновки,реферат	08.05.23-12.05.23	Виконано
7	Розробка обов’язкових демонстраційних матеріалів	15.05.23-24.05.23	Виконано
8	Попередній захист роботи	25.05.23	Виконано
9	Здача роботи	01.06.23	Виконано

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

Реферат

Текстова частина бакалаврської роботи 60с.,35 рис.,6 табл.,10 джерел.

HTML,CSS,JAVASCRIPT,REACT.JS,REDUX,FLUX,POSTGERSQL

Мета дослідження – спростити організацію опитувань.

Об’єкт дослідження – процес опитування студентів.

Предмет дослідження – веб-платформа для опитування студентів.

Методи дослідження - методи проектування та розробки програмного забезпечення, методи опрацювання та аналізу отримання результатів, методи тестування програмного забезпечення.

Фінальний проект дипломної роботи має на меті створення ефективного та зручного інструменту для проведення опитувань студентів у навчальному середовищі.

У проекті використовується бібліотека React.js, яка є однією з найпопулярніших інструментів для розробки користувацького інтерфейсу (UI) у веб-додатках. React.js дозволяє створювати компоненти, які можна повторно використовувати та ефективно маніпулювати станом додатка.

Для кращого управління станом додатка та забезпечення одностороннього потоку даних використовується архітектура Flux. Ця архітектура включає центральний зберіг даних (Store), диспетчер (Dispatcher), який керує потоком даних, та дії (Actions), які ініціюють зміни в збереженому стані.

Основні характеристики виконаної роботи:

1. Розробка користувацького інтерфейсу з використанням бібліотеки React.js, що дозволяє створювати ефективні та зручні компоненти для опитувань студентів.

2. Використання архітектури Flux для керування станом додатка та забезпечення одностороннього потоку даних.

3. Розробка функціональності для створення, збереження та відображення опитувань студентів.

4. Інтеграція з серверною частиною, яка забезпечує збереження та обробку даних опитувань.

5. Реалізація функціоналу для аутентифікації студентів та доступу до опитувань.

6. Тестування функціональності платформи та виправлення помилок.

7. Впровадження платформи в реальному середовищі та забезпечення її стабільної роботи.

Завдяки використанню бібліотеки React.js та архітектури Flux, розроблена веб-платформа надає зручні та ефективні інструменти для проведення опитувань студентів. Вона дозволяє студентам зручно взаємодіяти з платформою, а викладачам отримувати цінний зворотний зв'язок та оцінювати ефективність навчання

Зміст

ВСТУП

Архітектурний підхід Flux використовується у багатьох веб-застосунках для передбачуваного, одностороннього керування станом програми, що виділяє його поміж інших принципів та підходів. Також така бібліотека як Redux, яка була створена на основі цього архітектурного способу організації веб- застосунків, наразі є часто використовувана за даними npm trends[1]. У цій роботі висвітлено основні принципи використовування цього підходу на прикладі вищезазначеної бібліотеки для React застосунків, зокрема для розробки клієнтської частини сервісу опитування студентів.

Мета дослідження полягає у ознайомленні із архітектурою Flux, принципами та специфікою цього підходу, а також у застосуванні цього підходу для розробки веб-застосунку. Основними завданнями даного дослідження є:

1. висвітлення основних концепцій архітектури Flux;
2. опис проблем та рішення, які надає підхід Flux;
3. порівняння цього підходу із MVC архітектурою;
4. аналіз бібліотек Redux та React Redux;
5. створення веб-застосунку з використанням бібліотек React.js та Redux.

Об’єктом дослідження є архітектура Flux, запропонована компанією Facebook для розробки клієнтських веб-застосунків.

У цій роботі дослідження проводилися шляхом аналізу документації Flux та бібліотеки Redux, перегляд статей за темою дослідження та порівняння підходів MVC та Flux. Також було протестовано цей підхід через розробку застосунку на основі висвітлених даних у цій роботі. Створення схем відбувалося з допомогою онлайн-сервісу для побудови діаграм та схем Lucidchart.

До джерел дослідження відносяться такі електронні ресурси як офіційна

документація опрацьованих у цій роботі бібліотек та електронні статті з теми дослідження, зазначені у списку використаних джерел.

Робота складається з 5 розділів.

Перший розділ призначено висвітленню та аналізу архітектури Flux та опису проблем, які вирішує даний підхід. Також у розділі з'ясовуються відмінності між цим підходом та підходом MVC.

В другому розділі наведено основні можливості бібліотеки Redux, яка є прикладом застосування досліджуваного підходу для організації веб-застосунків. Показано відмінності специфікації цієї бібліотеки від запропонованих підходом Flux. Розглянуто бібліотеку React Redux для зв'язування React.js та Redux.

У третьому розділі наводиться реалізація клієнтської частини сервісу опитування студентів з використанням Flux підходу. Висвітлюються потоки даних застосунку та описуються ключові аспекти у використанні бібліотеки Redux на практиці.

У результаті виконання дослідження створено програмний продукт, який призначений для створення платформи опитування студентів про якість викладання курсів у їх вищому навчальному закладі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття «Web-додаток» та «Web-сайт»

Способи розробки web-додатків можуть бути поділені на три великі категорії: підходи, що ґрунтуються на програмуванні чи скриптах: зовнішні програми чи скрипти; розширення веб-сервера; підходи, засновані на використанні шаблонів веб-сторінок, що включають вставки коду скриптів та спеціальних серверних тегів; об'єктні середовища (каркаси, фреймверки, frameworks).

Хоча між цими категоріями є перетину (а також різні думки про те, до якої категорії належить конкретна технологія розробки), більшість широко відомих підходів пов'язані з однією конкретною категорією.

Розглянемо програмні підходи. В даному підході web-додатком (динамічним ресурсом, пов'язаним з URL адресою) є зовнішня програма, складена деякою універсальною мовою програмування високого рівня (наприклад, як Java або C++) або скрипт, складений за допомогою скриптової мови (scripting language), Виконання якого проводиться також за допомогою зовнішньої програми - інтерпретатора скриптів (script engine).

Основною проблемою з програмним підходом до розробки web-додатків є їхня орієнтація на написання коду. Розмітка HTML та інші конструкції форматування вбудовуються у логіку роботи програми за допомогою операторів виводу. Це обмежує можливості web-дизайнерів робити свій внесок в оформлення сторінки, що створюється додатком.

Web-дизайнер може розробляти макет сторінки, а програміст повинен потім перетворювати його на код і зв'язати зі скриптом або програмою. Для зміни практично будь-якого елемента сторінки, що формується, потрібне втручання

програміста, чи стосується це зміни логіки роботи програми, або зміни оформлення і розташування елементів сторінки.

Зовнішні програми. Найпростіший спосіб динамічно формувати web-сторінки у відповідь на HTTP запит полягає в тому, щоб передати роботу з вирішення необхідної задачі та формування HTML сторінки зовнішньої програми, яка повинна отримувати вхідні параметри, що передані в HTTP запиті, і сформувати вихідну сторінку на мові HTML.

Першою широко використовується, незалежною від типу web-сервера, програмною технологією створення та виконання web-додатків була технологія Common Gateway Interface (CGI, загальний шлюзовий інтерфейс). Вона визначала набір правил, яким повинна виконуватися програма, щоб вона могла виконуватися на різних HTTP серверах та операційних системах. Відповідно до CGI технології при вступі до web-сервер HTTP запиту, який включає посилання не на статичну сторінку, а на CGI програму (наприклад: prog.exe), створюється новий процес, в якому запускається необхідна прикладна програма.

Технологія CGI задає спосіб передачі програмі параметрів, що входять до складу HTTP запиту. Передача вхідних даних може виконуватися або за допомогою фіксованого набору змінних середовища (environment va-riables), які можуть створюватися однією програмою та використовуватись іншими програмами), або через вхідні дані функції, з якої починається робота програми (функція main()), а результати роботи програми (HTML сторінка) повертаються за допомогою стандартного потоку виводу STDOUT. Наведемо приклад простої CGI програми, написаної мовою З, яка формує HTML сторінку з переліком переданих їй параметрів.

Технологія CGI дозволяє використовувати будь-яку мову програмування, яка може працювати зі стандартними пристроями вводу/виводу. Крім цього, CGI програми можна писати з використанням скриптових мов, які називаються CGI скриптами.

Прикладами скриптових мов CGI є, наприклад, Perl, Python або Tcl. При використанні скрипта web-сервер викликає на виконання зовнішню програму - інтерпретатор скриптів (script engine), якій передаються дані HTTP запиту та ім'я файлу, в якому міститься скрипт, що запитується користувачем.

А потім ця програма виконує вказаний скрипт і повертає серверу сформовану сторінку HTML. Недоліки технології CGI. Технологія CGI є досить простим способом динамічно формувати інформацію у web-мережі, але вона має суттєві недоліки, які роблять її не практичною у більшості випадків: — основною проблемою є продуктивність: для кожного HTTP запиту до CGI програми web-сервер запускає новий процес, який закінчує роботу лише після завершення програми.

Робота зі створення та завершення процесів є досить трудомісткою, що може дуже швидко знизити продуктивність системи, крім того, різні активні процеси починають конкурувати за системні ресурси, такі як оперативна пам'ять.

1. для складання та налагодження CGI програм розробник повинен мати досить великий досвід програмування однією з мов, якими можна програмувати CGI програми.
2. у програмах CGI програмний код і код розмітки повністю перемішані.

Дизайнер повинен знати програмування, щоб змінювати структуру веб-сторінок. Спробою поєднати переносимість програм CGI з ефективністю є технологія FastCGI. Ця технологія ґрунтується на простій ідеї: замість необхідності щоразу запускати новий процес для обробки CGI скрипта, FastCGI дозволяє не закривати процеси, пов'язані з CGI скриптами, після закінчення обробки, а використовувати їх для обробки нових запитів до програм CGI. А це означає, що не потрібно постійно запускати і видаляти нові процеси, оскільки один і той самий процес може використовуватися багаторазово для обробки запитів.

Такі процеси можуть ініціалізуватися лише один раз за її створення. Модулі сервера, які виконують функціональність FastCGI, взаємодіють з сервером HTTP за допомогою своїх власних API.

Ці API намагаються приховати деталі реалізації та конфігурування від FastCGI додатків, але розробники однаково повинні знати особливості реалізації технології FastCGI, оскільки модулі різних типів серверів не сумісні між собою.

Розширення веб-серверів. Недоліки технологій CGI можна подолати шляхом розширення можливостей web-серверів за допомогою спеціальних компонентів.

Використовуючи такі розширення, програми, що формують HTTP відповіді, можуть виконуватися більш ефективно, без необхідності їх завершення після обробки кожного запиту та за рахунок використання спільних ресурсів кількома програмами.

Такі технології зазвичай надають можливість зберігати в основній пам'яті дані сеансів роботи користувачів, які взаємодіють із додатком протягом великої кількості запитів HTTP. Інтерфейс Java Servlet API. Інший широко використовуваної технології розширення архітектури web-сервера є прикладний інтерфейс Java Servlet API, який пов'язує web-server з віртуальною машиною Java Virtual Machine (JVM).

Віртуальна машина JVM підтримує виконання спеціальної Java програми (контейнер сервлетів), яка відповідає за керування даними сеансу роботи та виконання Java-сервлетів. Сервлети – це спеціальні класи на мові Java (програми), які мають доступ до інформації з HTTP запитів. Вони формують HTTP відповіді, які повертаються браузерам. Контейнер сервлетів (середовище виконання) відповідає за отримання від web-сервера HTTP запитів на виконання сервлетів; створення сеансу роботи користувача, якщо це потрібно; виклик сервлета, пов'язаного з HTTP запитом; передачу сервлету параметрів, які містяться в запиті HTTP, представлених у вигляді Java об'єктів.

На відміну від ISAPI розширень, технологія Servlet API переноситься між різними webсерверами, операційними системами і комп'ютерними платформами.

Сервлети виконуються однаково в будь-якому середовищі, яке надає сумісний із ними контейнер сервлетів. Технологія Servlet API використовується великою кількістю розробників та підтримується багатьма відомими web серверами. Підходи на основі шаблонів.

Підходи, засновані на шаблонах (template approaches, шаблонні підходи) використовують як об'єктів, що адресуються (має URL-адресу) не програми або скрипти, а «шаблони».

По суті шаблони є HTML файлами з додатковими "тегами", які задають методи включення контенту, що динамічно формується. Таким чином, файл шаблону містить HTML код, який описує загальну структуру сторінки, і додаткові серверні теги, розміщені таким чином, щоб зміст сторінці, що формується з їх допомогою, мало необхідний вигляд.

В даний час до найбільш поширених технологій розробки web-додатків на основі шаблонів відносяться такі: Server-Side Includes (SSI), Cold Fusion, PHP, Active Server Pages (ASP) та Java Server Pages (JSP).

Технологія Cold Fusion. Іншою досить популярною технологією, яка базується на шаблонах, є технологія Cold Fusion, розроблена компанією Adobe. Перевага даного підходу полягає в тому, що даний шаблон може створюватися та підтримуватись дизайнером сторінки, який має базові знання мови HTML та web-графіки, але не має досвіду програмування.

Спеціальні теги, які є "розширенням" HTML. Технологія PHP Hypertext Preprocessor. Технологія PHP Hypertext Preprocessor або просто PHP дозволяє розробникам вбудовувати програмний код у шаблони, за допомогою мови, подібної до мови скриптів Perl. Технологія Active Server Pages.

1.2 Основні відмінності web- додатку від web-сайту

Компанія Microsoft розробила технологію ASP (Active Server Pages), яка об'єднала можливості створення шаблонів, що включають скрипти, з доступом до набору OLE та COM об'єктів, що є з операційною системою Windows, у тому числі й до ODBC джерел даних. Ця технологія, об'єднана з безкоштовним web-сервером Internet Information Server (IIS), швидко стала популярною серед програмістів, що використовують Visual Basic, які оцінили можливість використання в шаблонах мови VBScript.

Як і PHP шаблони, ASP сторінки можуть включати блоки скриптів (які використовують посилання на COM об'єкти), упереміш з HTML форматуванням. На відміну від таких об'єктно-орієнтованих мов, як Java або C++, мова, що використовується в ASP сторінках, була плоскою, лінійною та строго процедурною.

На відміну від технології PHP, ASP не пов'язаний з однією конкретною мовою скрипту. В ASP як стандартна мова використовується мова Visual Basic Scripting Edition (VBScript), але може використовуватися і мова JavaScript. В ASP шаблони (також, як і в PHP шаблони) можуть включатися блоки, виділені за допомогою тегів , які містять код скрипта, який виконується інтерпретатором ASP шаблонів, при формуванні відповіді. HTML розмітка, що знаходиться поза такими блоками, розглядається як вихідний HTML код і просто переписується у сторінку, що формується.

Крім цього, на початок шаблону можуть додаватися директиви сторінки, такі, як наприклад , яка інформує систему обробки про використовувану скриптову мову. Підходи з урахуванням об'єктних середовищ. Звичайні скриптові технології на стороні сервера використовують різні об'єкти, але не дозволяють розробляти та використовувати власні класи та створювати на їх основі об'єкти. У зв'язку з цим розвиток web-технологій було з створенням спеціальних об'єктно-орієнтованих технологій розробки web-додатків.

Використання даних технологій дозволяє зробити розробку web-додатків більш подібною до розробки звичайних об'єктно-орієнтованого програмного забезпечення. Об'єктні середовища (фреймверки, frameworks) є наступним рівнем вдосконалення розробки web-додатків.

Замість об'єднання розмітки та логіки в єдиний модуль, об'єктні середовища (frameworks) підтримують принцип відокремлення змісту від уявлення. Модулі відповідальні за створення контенту відокремлюються від модулів, які показують цей зміст у конкретному форматі. В даний час є два підходи до створення об'єктно-орієнтованих web-додатків:

1. підходи, що ґрунтуються на наборі спеціальних web-сторінок (web-форм), пов'язаних з описами класів, об'єкти яких будуть створюватися, та використовуватися при їхньому виклику (наприклад: технологія ASP.Net Web Forms; технологія JavaServer Faces);
2. підходи, що ґрунтуються на використанні наборів класів, що відповідають шаблону Model-View-Controller (MVC) (наприклад: технології на основі мови Java – Tapestry, Struts, Spring та технологія компанії Microsoft – ASP.Net MVC).

Підхід з урахуванням архітектурного шаблону MVC. Відповідно до архітектурного шаблону MVC всі класи, складові додаток (у тому числі і web-додаток) поділяються на три основні групи (компоненти): Модель (Model), Подання (View) та Контролер (Controller).

Кожен із цих компонентів відповідає за свої завдання:

1. Модель (Model) – це набір класів, які реалізують всю бізнес-логіку web-додатків. Ці класи відповідають за обробку даних, розміщення їх у БД, читання з БД, а також за взаємодію між самими об'єктами, що становлять такі дані.
2. Подання (View) – набір класів, які відповідають інтерфейс взаємодії з користувачами (User Interface, UI).

З їхньою допомогою формуються HTML сторінки, що показують користувачам дані. Подання використовує дані з Моделі та надає користувачам можливість виконувати їх редагування. - Контролер (Controller) - це сполучна ланка між першими двома компонентами.

1.3 Переваги та недоліки Web-додатку

Класи даного компонента отримують дані про запит до сервера (наприклад, значення, отримані з відправленої форми), і передає їх у Модель для обробки та збереження. Після обробки отриманих даних Контролер вибирає, як показати їх клієнту з допомогою використання деякого класу з Подання.

В результаті такого поділу web-додатка на компоненти, розробник отримує повний контроль над формованим HTML документом; спрощується структура програми; полегшується завдання виконання тестування програми; досягається повне відділення логіки роботи програми від представлення даних.

Прикладами технологій розробки на основі MVC є:

1. технологія Struts (заснована мовою Java);
2. технологія ASP.Net MVC, що входить до складу набору технологій ASP.Net платформи .Net Framework;
3. технологія Ruby on Rails (Ruby – мова програмування, а Rails – фреймворк, що використовує цю мову) особливо популярна останнім часом.

Загальні рекомендації щодо розробки web-додатків. Основною метою архітектора програмного забезпечення при проектуванні web-додатків є максимальне спрощення їх структури шляхом поділу завдань на функціональні області, забезпечуючи при цьому безпеку та високу продуктивність.

Для ефективної роботи web-додатків у звичайних для них сценаріях необхідно:

1. Виконати логічний поділ функціональності програми, використовуючи багатошарову структуру для логічного поділу програми на шар уявлення, бізнес-шар і шар доступу до даних. Це допомагає створити зручний в обслуговуванні код і дозволить відстежувати та оптимізувати продуктивність кожного шару окремо.

2. Чіткий логічний поділ також забезпечує ширші можливості масштабування програми.

3. Використовувати абстракцію для реалізації слабкого зв'язування між шарами.

Цей підхід можна реалізувати за допомогою інтерфейсних типів або абстрактних базових класів можна визначити абстракцію, що спільно використовується, яка повинна бути реалізована інтерфейсними компонентами. - Визначити, як буде реалізована взаємодія компонентів один з одним.

Для цього необхідно розуміти сценарії розгортання, які має підтримувати програму. — Використовувати кешування для зменшення кількості мережних дзвінків та звернень до бази даних. - Використовувати протоколювання та інструментування.

Необхідно виконувати аудит та протоколювання дій у шарах та рівнях програми. Журнали реєстрації подій можуть бути використані для виявлення підозрілих дій, що часто забезпечує раннє виявлення атак на систему. - Продумати аспекти автентифікації користувачів на межах довіри. При проектуванні програми необхідно передбачити аутентифікацію користувачів при перетині меж довіри, наприклад, при доступі до віддаленого бізнес-шару з шару подання. — не надсилати важливі конфіденційні дані через мережу у вигляді відкритого тексту.

Якщо потрібно надсилати конфіденційні дані, такі як пароль або куки автентифікації, використовувати шифрування та підписи даних або шифрування за допомогою протоколу Secure Sockets Layer (SSL). — проектувати web-додаток для виконання з менш привілейованим обліковим записом. Процес повинен мати

обмежений доступ до файлової системи та інших ресурсів системи. Це дозволить максимально скоротити можливі негативні наслідки у разі, якщо зловмисник спробує взяти процес під свій контроль.

Існує низка загальних питань, на які слід звернути увагу під час проектування. Ці питання можна згрупувати за певними областями проектування: - обробка запитів додатку; автентифікація; авторизація; кешування; управління винятками; протоколювання та інструментування; навігація; компонування сторінки; формування візуального відображення сторінки; керування сеансами; перевірка введених даних (валідація).

1.4 Архітектура та принципи роботи типового web-додатку

eCommerce - додатки електронної комерції, до них відносяться інтернет-магазини, маркетплейси, B2B-портали і онлайн-аукціони. Для таких сайтів важлива інтеграція з платіжними системами і службами доставки. Іншіпріоритети включають забезпечення безпеки і створення максимально комфортного користувацького досвіду.

Системи автоматизації бизнесу – це CRM, ERP-системи, програми для бухгалтерського обліку і платформи електронного документообігу також можна встановлювати в хмарі. Таким чином всі дані будуть консолідовані на безпечному сервері. А співробітники і партнери з необхідним рівнем доступу зможуть користуватися можливостями системи без прив'язки до конкретного комп'ютера.

SaaS-платформи - SaaS-додатки поширюються не в формі, як традиційний софт, а завантажуються безпосередньо в браузер користувача з вашого сервера. Така модель вигідна як постачальнику ПО, так і користувачам, але щоб побудувати SaaS-продукт, вам знадобиться кваліфікований розробник з досвідом реалізації складних проектів.

MVP - оптимальний спосіб запустити стартап - розробка MVP - мінімального

життєздатного продукту. Такий веб-додаток буде мати всі необхідні функції для тестування вашої ідеї і залучення перших користувачів. Миможемо виступити вашим технологічним партнером і побудувати для вас робочу версію продукту, з якої буде набагато простіше залучити інвесторів.

Агрегатори даних – сайти-агрегатори - автоматизовані платформи, які беруть дані від новинних агентств, тематичних сайтів і інших постачальників контенту. З точки зору користувачів, перевага агрегатора в тому, що всі матеріали зібрані на єдиному майданчику в зручному форматі. У той же час, у власника агрегатора немає необхідності утримувати власну редакцію авторів.

B2B-портали – це інтернет-майданчик, в межах якого реалізуються угоди між компаніями (наприклад, виробником і гуртковиком, між гуртковиком і роздрібним продавцем). До порталу можна підключити партнерів з різних рівнів ланцюга розподілу: дистриб'юторів, дилерів, незалежних гуртовиків, рітейлерів. На порталі здійснюються оптові продажі, оформлення і попередня обробка замовлень, обмін документами і довідковою інформацією.

1.5 Види інтернет-додатків

Web-додаток являється клієнт-серверним, в я кому клієнтом виступає браузер, а сервером web-сервер. Логіка web-додатку розподілена між сервером та клієнтом. Збереження даних переважно здійснюється на сервері. Обмін інформації проходить у мережі Інтеренет.

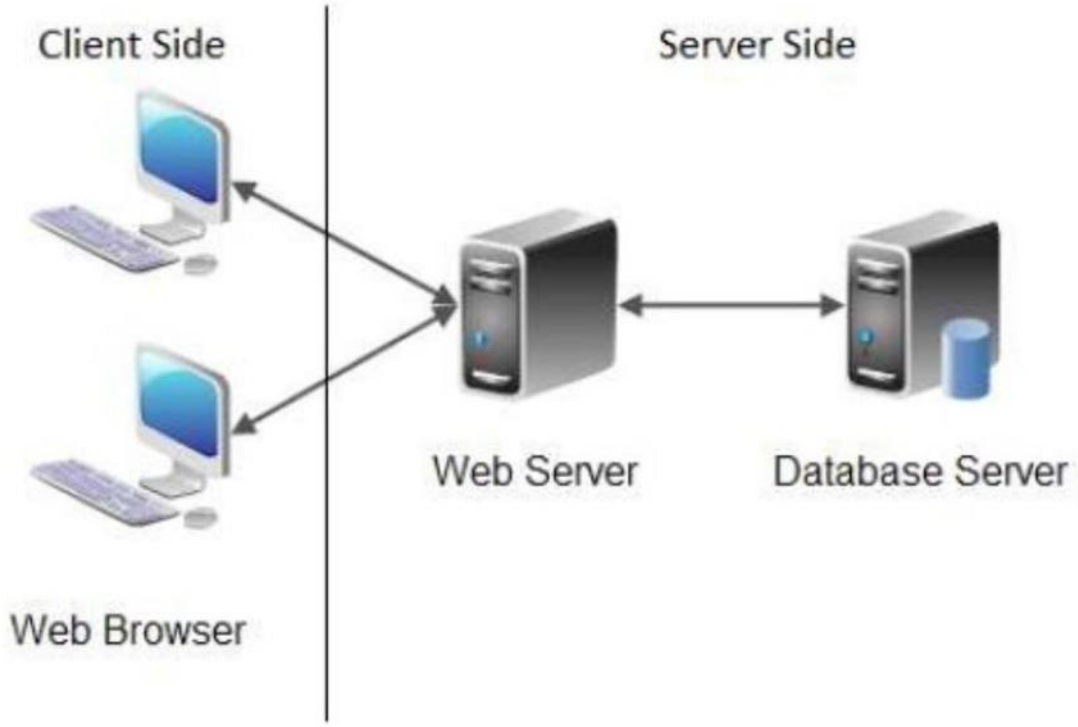


Рисунок 1.1 - Діаграма типового web-додатку

Є дві програми, які одночасно працюють у web-додатках:

1. код на сервері і відповідає на HTTP-запити;
2. код у браузері, який реагує на введення користувача

Код сервера не видимий для клієнта, він може відповідати лише на HTTP-запити з певною URL-адресою, а не на будь-який вхід користувача. Цей код зазвичай пишеться на таких мовах програмування та фреймворках: Java JavaScript (Node.js), Python (Django), PHP, Ruby On Rails, Java, C# та інші.

Код клієнта аналізується браузером користувача. На відміну від сервера, він може реагувати на введення користувача, доступний користувачеві для повного перегляду та редагування. Файли сервера неможливо прочитати безпосередньо, сервер повинен обробляти HTTP-запити. Мови програмування друкують HTML, CSS, JavaScript.

Технологічний стек (Рисунок 1.2) представляє собою:

1. операційну систему (файлову систему). Частіше всього це Linux;
2. web-сервер (Apache, NGINX, IIS та ін.);
3. база данних (MySql, MongoDB, MS Sql, Oracle, PostgreSql, Redis та ін.)

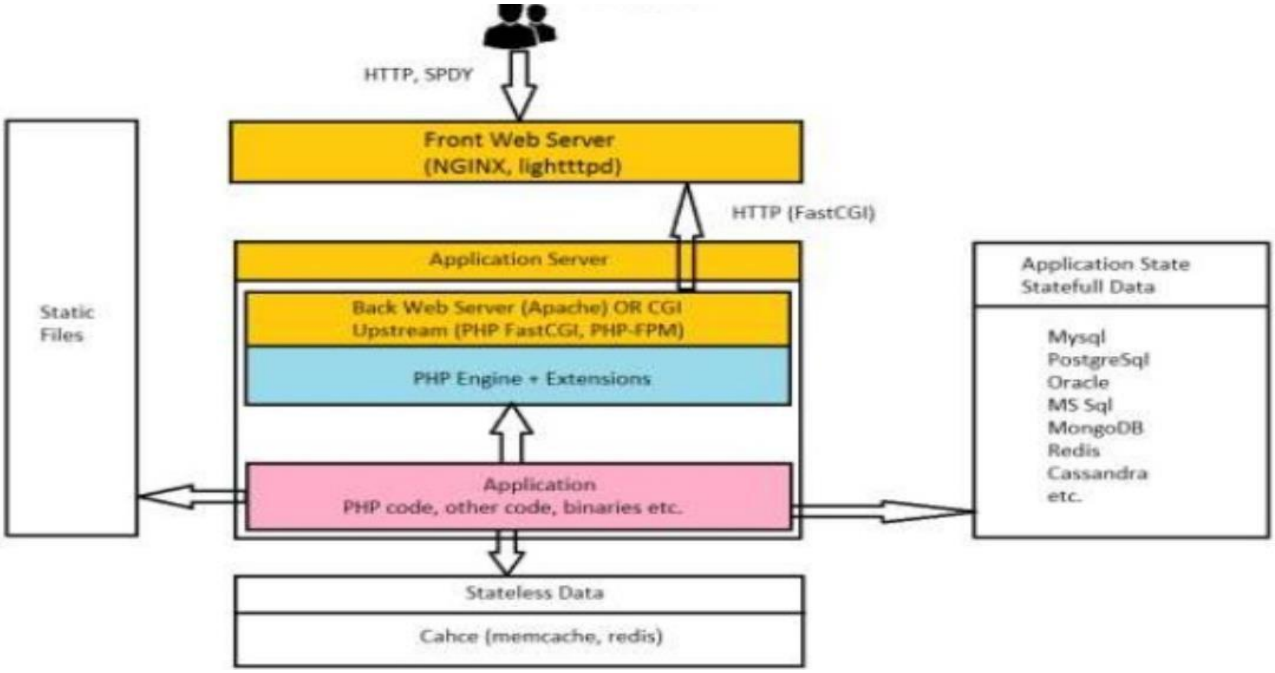


Рисунок 1.2 - Технологічний стек

Шлях запиту:

1. DNS.
2. HTTP, SPDY.
3. Front Server (NGINX).
4. Back Server or Application Server (Apache, PHP-FPM).
5. PHP code, opcode.
6. Response.

DNS (Domain Name System) - ієрархічна розподілена система перетворенняімені хоста (комп'ютера або іншого мережевого пристрою) в IP-адресу. Всі комп'ютери, підключені до Інтернету, включаючи смартфони, настільні комп'ютери і сервери, що надають контент для величезних торгових веб-сайтів, знаходять один одного і обмінюються інформацією за допомогою цифр. Ці цифри називаються IP-адресами.

Щоб відкрити веб-сайт в браузері, не потрібно запам'ятовувати довгі набори цифр. Досить ввести доменне ім'я, наприклад example.com, і браузер відкриє потрібну сторінку.

Шлях користувацького запиту починається з DNS:

1. Користувач звертається до DNS для отримки IP адреси.
2. DNS віддає IP адресу користувачеві.
3. Користувач звертається до web-додатка по IP через TCP/IP.
4. Відкривається мережеве з’єднання (HTTP, SPDY).

Стек протоколів TCP/IP – це набір мережевих протоколів, що забезпечують передачу даних, які використовуються в мережах, включаючи мережу Інтернет. Назва походить від двох найважніших протоколів: TCP (Transmission Control Protocol), IP (Internet Protocol).

IP – протокол, який лежить у основі Інтернета, його назва так і розшифровується: Internet Protocol. Наразі використовується дві версії протоколу

1. IPv6 – порівняно нова (опублікована в 1998); IP-адреса має розрядність 128 біт і записується у виді восьми 16-ти бітних полів, з використанням шістнадцяткової системи числення і з можливістю скороченням двох чи більше послідовних нульових полів до «::». Приклад: 2001:db8:1234:5::1:1.
2. IPv4 – опублікована в 1981р; IP- адреса має розрядність 32 біта і записується у виді чотирьох десятичних чисел в діапазоні 0…255 через точку; приклад: 192.0.3.43.

Кожен вузол може бути підключений тільки безпосередньо до вузлів мережі (наприклад, підключений до однієї і тієї ж частини Інтернету), що визначає мережеву адресу - частину IP-адреси, в залежності від маски мережі. Підключення до інших мереж здійснюється через допоміжні вузли - маршрутизатори.

TCP – це протокол, який базується на IP для доставки пакетів, але добавляє дві важливі речі: встановлення з’єднання – це дозволяє йому, на відміну від IP, гарантувати доставку пакетів; порти – дозволяють обмінюватись пакетами між

застосунками, а не просто вузлами. Протокол TCP призначений для обмінуданими – це надійний протокол, тому що:

1. Забезпечує надійну доставку даних, так як передбачає встановлення логічного з’єднання.

2. Нумерує пакети і підтверджує їх прийом квитанцією, а у випадку загублення пакету організує повторну передачу.

3. Ділить переданий потік байтів на частини – сегменти, і переда їх нижньому рівню, на прийомній стороні знову збирає їх у потік байтів. З’єднання двух вузлів починається з handshake (рукопотискання):

4. Вузол А посилає вузлу В спеціальний пакет SYN

5. В відповідає пакетом SYN-ACK – згодою на установлення з’єднання.

6. А посилає пакет ACK – підтвердження, що згоду отримано.

Після цього TCP з’єднання рахується установленим, і додатки, працюючі на цих вузлах, можуть відправляти один одному пакети з даними. З’єднання означає, що вузли пям’ятають один одного, нумерують усі пакети, які ідуть в обидві сторони, посилають підтвердження про отримання кожного пакету і зновупосилають загублені по дорозі пакети. Для вузла А це з’єднання називається вихіднім, а для вузла В – вхідним.

HTTP (Hyper Text Transfer Protocol) – протокол прикладного рівня передачі даних (початково – у виді гіпертекстових документів в форматі «HTML», наразі використовується для передачі довільних даних).

SPDY (читається як «speedy», «спіді») – протокол прикладного рівня для передачі веб-контенту. Протокол розроблено корпорацією Google. По задуму розробників, даний протокол позиціонується як заміна деяких частин протоколу HTTP – таких, як управління з’єднаннями і форматами передачі даних.

Клієнт спілкується з сервером через HTTP-повідомлення. Обмін повідомленнями йде по звичайній схемі «запит-відповідь».

Кожне HTTP-повідомлення складається із чотирьох частин, які передаються у

вказанному порядку:

1. Стартовий рядок (англ. Starting line) – оприділяє типповідомлення;
2. Заголовки (англ. Headers) – характеризує тіло повідомлення, параметри передачі та інші дані.
3. Тіло повідомлення (англ. Message Body) – безпосередньо дані повідомлення. Обов'язково має відділятись від заголовків пустим рядком.

Відмінність

полягає в версії 0.9 протоколу, в якій повідомлення запиту містить лише першийрядок, а повідомлення-відповідь є лише частиною повідомлення.

Стартові рядки розрізняються для запиту і для відповіді. Рядок запиту виглядає так:

1. GET URI – для версії 0.9;
2. Метод URI HTTP/Версія – для інших версій.

Тут:

Метод – назва запиту, одне слово заглавними буквами. У версії HTTP 0.9 використовується тільки метод GET;

URI оприділяє шлях до запитаного документу;

Версія – пара розділених точкою цифр (Наприклад: 1.0);

Стартова строка відповіді має наступний формат: HTTP/Версія, код стану, пояснення, де:

1. Версія – пара розділених точкою цифр, як в запиті;
2. Код стану – три цифри. По коду стану оприділяється подальший вміст повідомлення і поведінка клієнта;
3. Пояснення – текстове коротке пояснення до коду відповіді для користувача. Ніяк не впливає на повідомлення і являється необов'язковим.

Методи HTTP - будь-яка послідовність символів, крім елементів керування та обмежень, які показують основну функцію ресурсу. Кожен сервер повинен

підтримувати принаймні методи GET і HEAD. Якщо сервер не знає методу, зазначеного клієнтом, має бути повернута умова 501 (не виконується). Якщо сервер знає метод, але його неможливо застосувати до певного джерела, буде повернуто повідомлення з кодом 405 (неавторизований метод). Крім методів GETі HEAD, є методи:

1. POST: Цей метод використовується для надсилання даних на сервер для обробки. Зазвичай використовується для створення нового ресурсу або виконання операцій, які можуть мати побічні ефекти на сервері.
2. PUT: Цей метод використовується для оновлення існуючого ресурсу на сервері. Клієнт надсилає нові дані, які повинні замінити попередні значення ресурсу.
3. DELETE: Цей метод використовується для видалення ресурсу на сервері. Клієнт надсилає запит на видалення, і сервер виконує відповідну дію.
4. PATCH: Цей метод використовується для часткового оновлення ресурсу на сервері. Клієнт надсилає тільки змінені дані, і сервер застосовує їх до ресурсу.

2 ПОРІВНЯЛЬНИЙ АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ, ІНСТРУМЕНТІВ ТА ТЕХНОЛОГІЙ

2.1 Аналітичний огляд інструментів та технологій для розробки веб-додатку

2.1.1 Hypertext Markup Language

HTML або Hypertext Markup Language – це код, який використовується для створення веб-сторінок і веб-додатків. HTML був створений у всесвітній мережі наприкінці 1980-х і на початку 1990-х років. Ця мова дозволяє веб-розробникам вказувати веб-браузерам, як відображати такі функції, як зображення, текст, форми та інтерактивні функції.

Веб-майстри зазвичай використовують HTML разом із двома іншими мовами програмування, CSS і JavaScript, для створення веб-сторінок і програм, доступ до яких можна отримати через веб-браузери, такі як Chrome, Firefox, Safari та Edge. Стандарти HTML і CSS історично підтримували World Wide Web Consortium (W3C).

У 1980 році Тім Бернер-Лі почав працювати над прототипом HTML для Європейської організації ядерних досліджень. Протягом десятиліття Burners Lee розробив всюдисущу мову, а також інтернет-браузер і серверне програмне забезпечення.

Протягом наступних двох десятиліть Консорціум World Wide Web Consortium (W3C) продовжить розробляти міжнародні стандарти коду. Друге

видання HTML було випущено в 1995 році, а протягом наступних кількох років відбулося ще кілька оновлень.

Хоча HTML заснований на правилах SGML (стандартна загальна мова розмітки, яку можна використовувати для опису деяких металевих мов для маркування документів), XHTML заснований на правилах XML, суворому підрозділі правил SGML. Документи XHTML повинні бути точними XML- документами, тому,

на відміну від HTML, їх можна запускати за допомогою стандартних інструментів обробки документів XMTML, що вимагає відносно складнішого, складного та повільнішого аналізу. XHTML часто розглядається як інтерфейс HTML і XML, оскільки ця вимога є покращенням HTML за допомогою XML. XHTML 1.0 Рекомендований консорціумом W3C 26 січня 2000 р. XHTML

1.1 став рекомендацією W3C 31 травня 2001 року.

HTML перекладається браузером і виглядає як зручний для людини документ.

HTML – це програма SGML (стандартна загальна мова аутентифікації) і відповідає міжнародним стандартам ISO 8879.

HTML: документ – це текстовий файл, позначений спеціальними (природними, текстовими) командами. Текстовий формат для вибору веб- документів вибирається виходячи з основних вимог до веб-документації: простота, можливість безпосереднього перекладу в будь-яку операційну систему, мінімальний розмір файлу, простота редагування та інтерпретації.

Мова розмітки гіпертексту HTML дозволяє вказувати різні типи елементів, які забезпечують функції документації.

Елементи HTML визначаються за допомогою тегів (від англійського label - тег). Усі теги HTML знайдені в тексті документа перекладаються браузером при відображенні документа.

HTML – це мова розмітки, фраза, яка використовується для опису того, як програміст використовує код для позначення тексту. Мови розмітки, такі як

HTML і XML, читаються і відрізняються від машинних мов, які є шістнадцятковим або двійковим кодом.

Створюючи веб-сторінку з використанням HTML, автор використовує ряд ключових елементів або функцій, які може прочитати будь-який Інтернет- браузер, який має доступ до Інтернету. Більшість із цих функцій поєднуються з

«Початковим тегом» і «Останнім тегом».

Наприклад, автор може використовувати букву «Р» у дужках (Рисунок 2.1),

щоб позначити початок абзацу, а другу «P» у кінці другого набору дужок. Перший фрагмент коду відкриває абзац, а другий фрагмент коду закриває абзац.

```
<p>Текст</p>
```

Рисунок 2.1 - Приклад використання

HTML 5 є найбільш відповідним стандартом для цієї мови жестів сьогодні. За цим критерієм базується найбільша кількість веб-сайтів в Інтернеті. Ми отримуємо підтримку мультимедіа (аудіо та відео) в останній версії HTML 5. Основною особливістю веб-сторінок сьогодні є HTML 5. Забезпечує послідовний перегляд веб-сторінок. А всі інші стилі та функції додаються за допомогою CSS, PHP, JavaScript. Тому вам потрібно вивчити HTML, перш ніж ви зможете вивчатиці мови.

Можна сказати, що HTML є основним будівельним блоком веб-розробки. Усі сучасні сервіси веб-дизайну, такі як WordPress, Wiki, Webli, використовуютьHTML 5 для створення веб-сайту.

Елемент TML відрізняється від іншого тексту в документі «тегами» за назвою навколишнього елемента «<» і «>». Назва елемента в тегу не чутлива до регістру. Тобто це можуть бути великі або малі літери чи комбінації. Наприклад, тег title можна записати як "Title", "Title" або будь-яким іншим способом.

Розробник веб-сайту може використовувати функції та елементи HTML, щоб визначити кожен аспект сторінки та те, як вона виглядає для користувача. Для створення веб-сайту програміст може використовувати CSS і JavaScript, а також додаткові мови, такі як PHP, jQuery та XML.

Програмісти-початківці вивчатимуть HTML як оригінальну мову програмування та продовжуватимуть вивчати розширені посібники з JavaScript та CSS.

Вивчення основ HTML може бути корисним для майбутніх програмістів, а

також для тих, хто хоче зрозуміти, що потрібно для створення веб-сайтів, які можна читати на комп’ютері чи смартфоні щодня.

2.1.2 Cascading style sheets

CSS – це мова каскадних таблиць стилів (англ. cascading style sheets) , яка використовується для опису стилів багаторазового використання для подання документів, написаних мовою розмітки. Його концепція була створена компанією Håkon Wium Lie у 1994 році. У грудні 1996 року WSC розробила специфікацію для CSS і сьогодні дозволяє веб-розробникам змінювати макет і зовнішній вигляд своїх веб-сторінок. Наприклад, CSS може використовуватися для зміни шрифту, який використовується в певному HTML-елементі, а також його розміру та кольору. Один CSS-файл може бути пов'язаний з декількома сторінками, що дозволяє розробнику одночасно змінювати зовнішній вигляд усіх сторінок. Наведене нижче поле містить основний приклад використання коду CSS для визначення шрифтів, кольору гіперпосилань та кольору посилання, коли курсор миші наводить курсор миші. У цьому конкретному прикладі ми змінюємо лише теги HTML <a> та <body>, а не створюємо нові селектори класу чи id. [ ]

```
body {
 font: normal 100% "trebuchet ms", Arial, Helvetica, sans-serif;
}
a {
 color: #000000;
}
A:visited {
 color: #005177;
}
a:hover {
 color: #005177;
}
```

Рисунок 2.2 - Поле з CSS кодом

Переваги CSS:

1. CSS економить час - ви можете написати CSS один раз, а потім повторно використовувати той же аркуш на декількох HTML-сторінках. Ви можете визначити стиль для кожного елемента HTML і застосувати його до якомога більше веб-сторінок.

2. Сторінки завантажуються швидше - якщо ви використовуєте CSS, вам не потрібно щоразу писати атрибути тегів HTML. Просто напишіть одне правило CSS тегу та застосуйте його до всіх випадків цього тегу. Так менше коду означає швидше завантаження.

3. Простота обслуговування - Щоб зробити глобальну зміну, просто змініть стиль, і всі елементи на всіх веб-сторінках будуть оновлені автоматично.

4. Покращені стилі до HTML - CSS має набагато ширший набір атрибутів, ніж HTML, тому ви можете надати набагато кращий погляд на свою сторінку HTML порівняно з атрибутами HTML.

5. Сумісність декількох пристроїв - аркуші стилів дозволяють оптимізувати вміст для більш ніж одного типу пристроїв використовуючи один і той же документ HTML, різні версії веб-сайту можуть бути представлені для портативних пристроїв, таких як КПК тамобільних телефонів або для друку.

Розглянемо версії CSS. Каскадні таблиці стилів рівня 1 (CSS1) вийшли з W3C як рекомендація в грудні 1996 року. Ця версія описує мову CSS, а також просту модель візуального форматування для всіх тегів HTML.

CSS2 став рекомендацією W3C у травні 1998 року та базується на CSS1. Ця версія додає підтримку для специфічних таблиць стилів, наприклад, принтери та аудіопристрої, завантажувані шрифти, розміщення елементів та таблиці.

2.1.3 JavaScript

JavaScript є найбільш часто використовуваною мовою сценаріїв для створення скриптів веб-браузера, вбудованих у веб-сторінки. JavaScript Sun Microsystems, Inc -

це зареєстрована торгова марка.

JavaScript створено для «оновлення веб-сторінок». Програми на цій мові називаються скриптами. Вони можуть бути записані безпосередньо на веб- сторінки HTML і автоматично запускатися, коли сторінка завантажується. Скрипти надаються і будуть реалізовані як простий текст. Для запуску вони не вимагають спеціальної підготовки чи компіляції. У цьому відношенні JavaScript сильно відрізняється від будь-якої іншої мови під назвою Java.

Коли JavaScript був створений, він спочатку називався «Живий скрипт». Але в той час Java була дуже популярна, тому було вирішено, що це допоможе зберегти нову мову як Java «молодший брат».

Але в міру покращення JavaScript став повністю незалежною мовою з власним ECMAScript і тепер не має нічого спільного з Java.

JavaScript зазвичай використовується як сценарна мова клієнта. Це означає, що код JavaScript написаний на сторінці HTML. Коли користувач шукає HTML-сторінку з JavaScript, сценарій надсилається браузеру, і браузер повинен вирішити щось зробити.

Оскільки сценарій знаходиться на сторінці HTML, ваші сценарії можуть переглядати та копіювати будь-хто, хто переглядає вашу сторінку. Однак я вважаю, що ця прозорість є великою перевагою, оскільки ви можете бачити, вивчати та використовувати скрипт Java, який знайдете на WWW.

JavaScript можна використовувати в контекстах за межами веб-браузера. JavaScript на стороні сервера Netscape, створений як мова CGI, може виконувати те саме, що Perl або ASP. Немає жодних причин, чому JavaScript не слід використовувати для написання реальних і складних програм. Однак цей сайт стосується лише використання JavaScript у веб-браузерах.

Сьогодні JavaScript працює не тільки в браузері, а й на сервері або будь- якому іншому пристрої за допомогою спеціальної програми під назвою JavaScript Mechanical. Браузер має вбудований «движок», який іноді називають віртуальною

машиною JavaScript. Різні двигуни мають різні «кодові назви». приклад:

1. V8 - в Chrome і Opera.
2. SpiderMonkey - у Firefox.

Сучасний JavaScript є «безпечною» мовою програмування. Низькорівневий доступ до пам’яті або ЦП не надається, оскільки він розроблений для браузерів, яким це не потрібно.

Можливості JavaScript значною мірою залежать від середовища, в якому він використовується. Наприклад, Node.js JavaScript підтримує функції, які дозволяють читати/записувати випадкові файли, надсилати мережеві запити тощо.

У браузері JavaScript може робити все, що стосується шахрайства на веб- сайті, взаємодії з користувачем та веб-сервера. Наприклад, JavaScript може наступне у веб-браузері:

1. Додати новий HTML на сторінку, змінити існуючий вміст, змінити стилі;
2. Реагувати на дії користувача(натискання миші, рух вказівника, натискання клавіш);
3. Надсилати запити по мережі на віддалених серверів, закачувати та завантажувати файли (так звані технології AJAX та COMET);
4. Отримати та встановити файли cookie, задавати питання відвідувачеві, показувати повідомлення;
5. Запам’ятовувати дані на стороні клієнта («локальнезберігання»).

2.2 Необхідність застосування фрейморку у веб-розробці

Фреймворки – це програмні продукти, які полегшують створення та обслуговування технічно складних проектів. Фреймворк зазвичай містить лише базові програмні модулі, а всі компоненти на основі проекту реалізовані

розробником. Це забезпечує не тільки високі темпи зростання, але й високу продуктивність та надійність рішень.

Web Framework – це платформа для створення веб-сторінок та веб- додатків, які полегшують розробку та інтеграцію різних компонентів великого програмного проекту. Завдяки широкому спектру можливостей і високій продуктивності в бізнес-логіці ця платформа особливо підходить для створення складних сайтів, бізнес-додатків і веб-сервісів.

З комерційної точки зору розробка на фреймворку є більш економічно ефективною та якісною, ніж написання чистою мовою програмування без використання будь-яких платформ. Розробка без використання платформи можебути ефективним вирішенням лише двох проблем - проект дуже простий і не потребує подальшої розробки або дуже зайнятий і вимагає оптимізації дуже низького рівня (наприклад, десятки тисяч викликів веб-сервісу в секунду) . У всіх інших випадках розробка програмної платформи відбувається швидше і краще.

Якщо порівнювати фреймворки з іншими классовими платформами - SaaS,CMS або CMF - фреймворки ефективніше використовувати в складній бізнес- логіці та проектах з високими вимогами до швидкості, надійності та безпеки. Але без значних вимог у простих і звичайних проектах темпи зростання і вартість на фреймворку будуть вище, ніж у SaaS або CMS.

Однією з головних переваг використання фреймів є те, що фреймворк визначає інтегровану структуру для додатків, побудованих на основі. Тому додатки на фреймворках дуже легко виправити і відфільтрувати, тому що стандартна структура класу зрозуміла всім розробникам на цій платформі і не потрібно багато часу, щоб знайти місце для реалізації іншого функціоналу. Більшість фреймворків в розробці веб-додатків використовують методологію MVC (model-view-control) – тобто багато фреймворків мають подібний підхід до організації елементів програмування, що дозволяє легко зрозуміти архітектуру програми навіть для незнайомого розробника фреймворка. [2]

Розробка архітектури програмного забезпечення також дуже проста при створенні на основі фреймворка – дана методологія часто включає в себе найкращі методи розробки програмного забезпечення і легко дотримуючись цихправил усуває багато проблем і помилок у проектуванні. ПО своїй суті являє

собою сукупність конкретних і абстрактних класів, пов'язаних між собою і впорядкованих за каркасним методом. Конкретні класи зазвичай реалізуютьвзаємні відносини між класами, а абстрактні класи являють собою точки розширення, в яких закладений у фреймворк базовий функціонал може бути використаний «як є» або адаптований під завдання конкретного додатка. Для забезпечення можливостей у більшості фреймів використовуються методи об’єктно-орієнтованого програмування наприклад, компоненти програми можуть бути успадковані від базових компонентів фреймворка або окремі модуліможуть бути з’єднані як сміття.

Веб-екосистеми також багаті на багатофункціональні готові програми. Розробникам не потрібно «винаходити велосипеди» під час співпраці, оскільки вони можуть використовувати вже створений додаток спільноти І це не тільки заощаджує час і гроші, але й робить рішення більш стабільним — компонент, який використовують тисячі інших розробників, часто краще реалізується та краще тестується в різноманітних ситуаціях як у випадку одного розробника так і навіть невеликою групою.

2.3 Аналіз веб–додатків зі сторони фінансових рішень

З комерційної точки зору розробка на фреймворку є більш економічно ефективною та якісною, ніж написання чистою мовою програмування без використання будь-яких платформ. Розробка без використання платформи можебути ефективним вирішенням лише двох проблем - проект дуже простий і не потребує подальшої розробки або дуже зайнятий і вимагає оптимізації дуже низького рівня (наприклад, десятки тисяч викликів веб-сервісу в секунду) . У всіх інших випадках

розробка програмної платформи відбувається швидше і краще.

Якщо порівнювати фреймворки з іншими классовими платформами - SaaS,CMS або CMF - фреймворки ефективніше використовувати в складній бізнес- логіці та проектах з високими вимогами до швидкості, надійності та безпеки. Але без значних вимог у простих і звичайних проектах темпи зростання і вартість на фреймворку будуть вище, ніж у SaaS або CMS.

Однією з головних переваг використання фреймів є те, що фреймворк визначає інтегровану структуру для додатків, побудованих на основі. Тому додатки на фреймворках дуже легко виправити і відфільтрувати, тому що стандартна структура класу зрозуміла всім розробникам на цій платформі і не потрібно багато часу, щоб знайти місце для реалізації іншого функціоналу. Більшість фреймворків в розробці веб-додатків використовують методологію MVC (model-view-control) – тобто багато фреймворків мають подібний підхід до організації елементів програмування, що дозволяє легко зрозуміти архітектуру програми навіть для незнайомого розробника фреймворка. [2]

Розробка архітектури програмного забезпечення також дуже проста при створенні на основі фреймворка – дана методологія часто включає в себе найкращі методи розробки програмного забезпечення і легко дотримуючись цих правил усуває багато проблем і помилок у проектуванні. ПО своїй суті являє

собою сукупність конкретних і абстрактних класів, пов'язаних між собою і впорядкованих за каркасним методом.

Конкретні класи зазвичай реалізують взаємні відносини між класами, а абстрактні класи являють собою точки розширення, в яких закладений у фреймворк базовий функціонал може бути використаний «як є» або адаптований під завдання конкретного додатка. Для забезпечення можливостей у більшості фреймів використовуються методи об’єктно-орієнтованого програмування — наприклад, компоненти програми можуть бути успадковані від базових компонентів фреймворка або окремі модулі можуть бути з’єднані як сміття.

3 ПРОЕКТУВАННЯ РІШЕНЬ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-ДОДАТКУ

3.1 Концептуальна модель додатку

Існує безліч визначень архітектури, зокрема архітектури програмного забезпечення. Хоча визначення та відрізняються один від одного, всі вони в основному вказують на те, що архітектура пов'язана зі структурою та поведінкою тільки зі значущими рішеннями, може мати якийсь стиль, на неї впливають оточення та зацікавлені особи, вона втілює рішення на основі логічного обґрунтування [17].

Архітектура інформаційної системи – концепція, визначальна модель, структуру, функції і взаємозв'язок компонентів інформаційної системи [18]. Говорячи про архітектуру веб-додатків, виділяють класичну архітектуру клієнт-сервер та багаторівневі архітектури клієнт-сервер.

Клієнт-сервер – це архітектура програми, в якій його функціональні частини взаємодіють за схемою запит-відповідь. Двома взаємодіючими частинами такої архітектури є клієнт (виконує активну функцію, тобто ініціює запити) і сервер (відповідає на запити). Класична архітектура клієнт-сервер має на увазі розподіл трьох основних частин веб-додатку (інтерфейс взаємодії з користувачем, серверну логіку та сховище даних) за двома фізичними модулями, в чому і полягає основний недолік даної архітектури [1].

Багаторівнева архітектура клієнт-сервер – це різновид описаної вище дворівневої архітектури, в якій обробка даних розподілена між одним або декількома серверами, що дозволяє ефективно розділити функції зберігання, обробки та подання даних. Ця архітектура реалізує ефективніший підхід до використання можливостей як серверів, і клієнтів [20]. Одним із різновидів багаторівневої архітектури є трирівнева архітектура.

Трирівнева архітектура – це архітектура програми, поділена натри основні

компоненти: клієнт, сервер додатків та сервер баз даних. Архітектура представлена візуально.

У рамках розробки Journal System було прийнято рішення розробити платформу відповідно до принципів трирівневої архітектури з наступних причин [21]:

1. спрощення масштабованості;
2. спрощення реалізації повноважень користувачів рахунок перенесення прикладної логіки на серверну сторону;
3. більш високий рівень безпеки, порівняно з дворівневою архітектурою.

Іншою особливістю визначення архітектури системи, що розробляється, є використовуваний шаблон проектування. Одним із найпоширеніших шаблонів у веб-розробці є MVC [22], який було вирішено використовувати при розробці Journal System.

MVC – це шаблон проектування веб-додатків, що включає кілька дрібніших шаблонів. MVC поділяє дані, прикладну логіку та інтерфейс користувача на три компоненти: модель, уявлення, контролер [23].

Модель – це центральний компонент шаблону, структура даних, незалежна від інтерфейсу користувача. Модель реагує на запити з контролера та повертає або змінює свій стан відповідно.

Подання – це будь-яка візуалізація інформації, включаючи таблиці, діаграми та графіки. З уявленням взаємодіє користувач.

Контролер – це інтерпретатор дій користувача, який приймає запити користувача та перетворює їх у команди для моделі чи подання. Як правило, фільтрація даних та авторизація користувача також є функціоналом контролера.

Загальна схема MVC представлена рисунку 3.1.

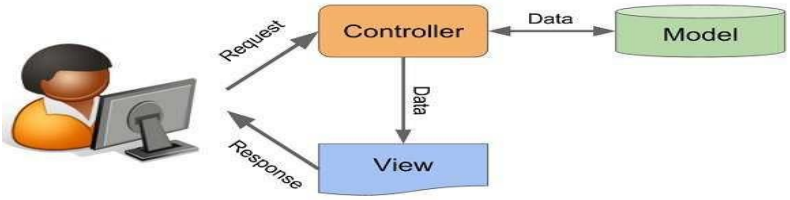


Рисунок 3.1 – Схема шаблону проектування MVC

3.2 Логічна модель веб-додатку

Проектування бази даних є важливим і комплексним завданням, як і проектування інших компонентів системи. Визначення структури проектованої бази є одним з першочергових завдань. Структура бази даних – це принцип чи порядок організації записів у базі даних та зв'язків між ними [4].

У веб-платформі, що розробляється, використовується реляційна модель представлення даних, що означає, що дані мають певні зв'язки між собою і організуються у вигляді таблиць, що складаються зі стовпців і рядків. Таблиці зберігають інформацію про об'єкти, які у базі даних.

Загалом у базі даних налічується 30 таблиць. У таблиці 1 містяться найменування та описи таблиць системи.

Таблиця 3.1 – Опис таблиць бази даних

№	Найменування	Опис
1	User	Зареєстрований користувач системи.
2	Role	Роль користувача системи.
3	user_role	Таблиця, що усуває ставлення «багато хто багатьом» між користувачем та його ролями.
4	Credentials	Персональні дані всіх зареєстрованих користувачів системи.

Продовження таблиці 3.1 – Опис таблиць бази даних

№	Найменування	Опис
5	Country	Список країн українською та англійською мовами.
6	Submission	Метадані статті, поданої у заявці на публікацію.
7	submission_file	Файли прикріплені до заявки на публікацію.
8	file_type	Типи файлів, які можна прикріпити до заявки на публікацію.
9	submission_history	Історія заявки на публікацію.
10	submission_status	Статус заявки на публікацію
11	author_submission	Автори статті, поданої у заявці на публікації.
19	Message	Повідомлення між автором (користувачем системи) та відповідальним секретарем.
20	message_file	Файли, прикріплені до повідомлення.
21	Issue	Випуск журналу.
22	issue_status	Статус випуску журналу
23	issue_submission	Статті випуску журналу.
24	issue_decision	Рішення члена редакційної колегії про придатності номера до публікації.
25	Volume	Том журналу.
26	academic_degree	Список наукових ступенів.
27	academic_title	Список вчених звань.

Продовження таблиці 3.1 – Опис таблиць бази даних

№	Найменування	Опис
27	academic_title	Список вчених звань.
28	review_template	Шаблони для рецензії.
29	File	Таблиця із файлами.
30	Language	Мова статті, а також ключових слів.
31	Subscription	Таблиця, що містить інформацію про передплату користувача на певні поштові розсилки.
32	user_subscription	Таблиця, що усуває ставлення «багато хто багатьом» між користувачем та його підписками.

Відносини таблиць та його поля зручно уявити графічно як схеми. Схема була логічно розділена на 4 частини та представлена на рисунках 3.2–3.5.

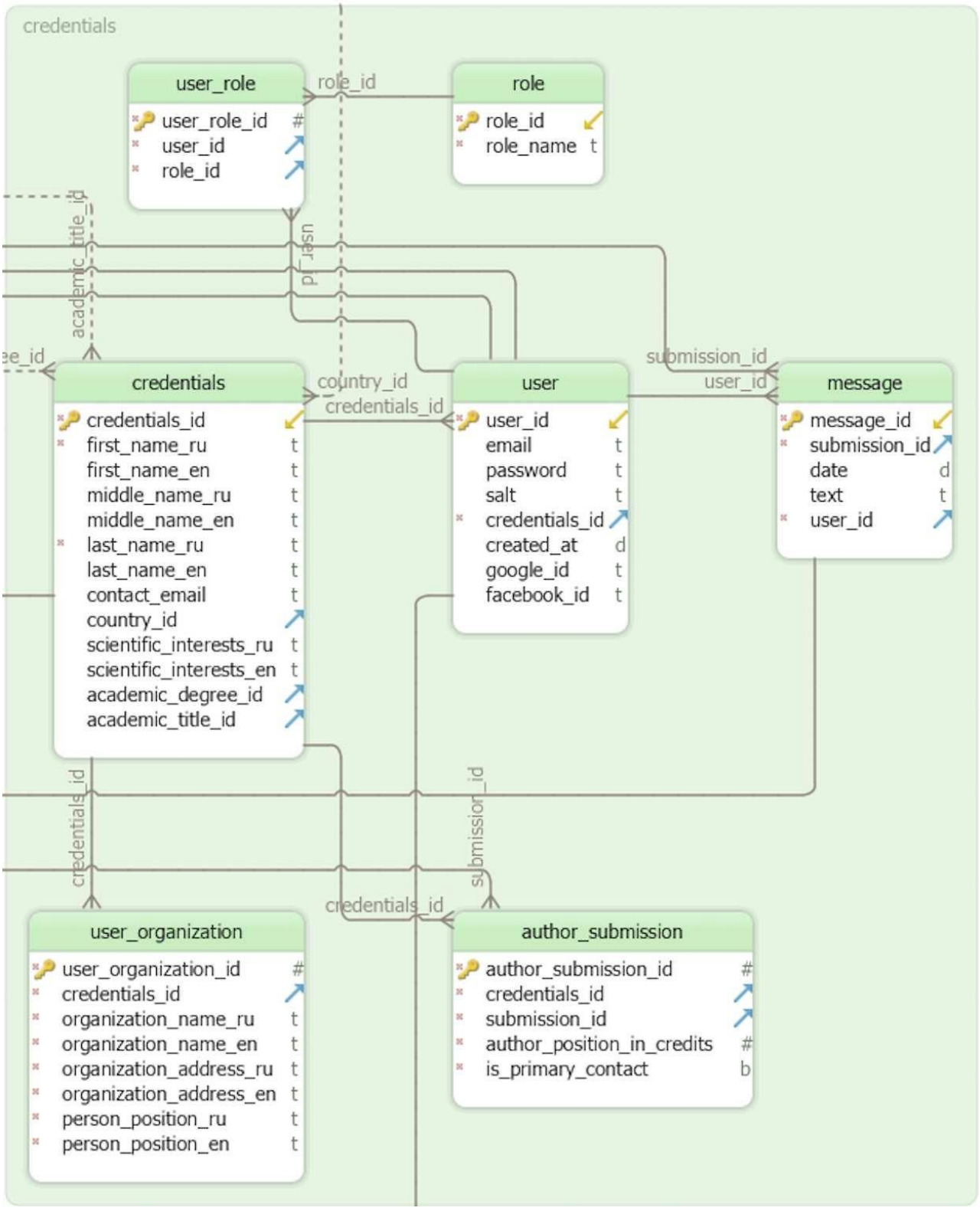


Рисунок 3.2 – Частина 1 схеми бази даних.

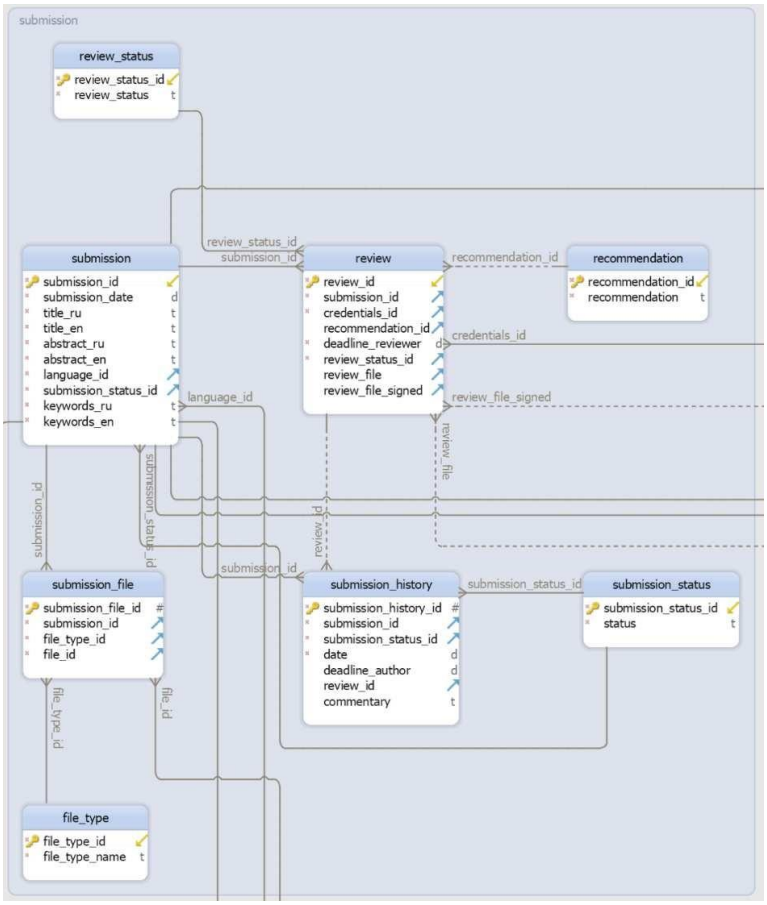


Рисунок 3.3 – Частина 2 схеми бази даних

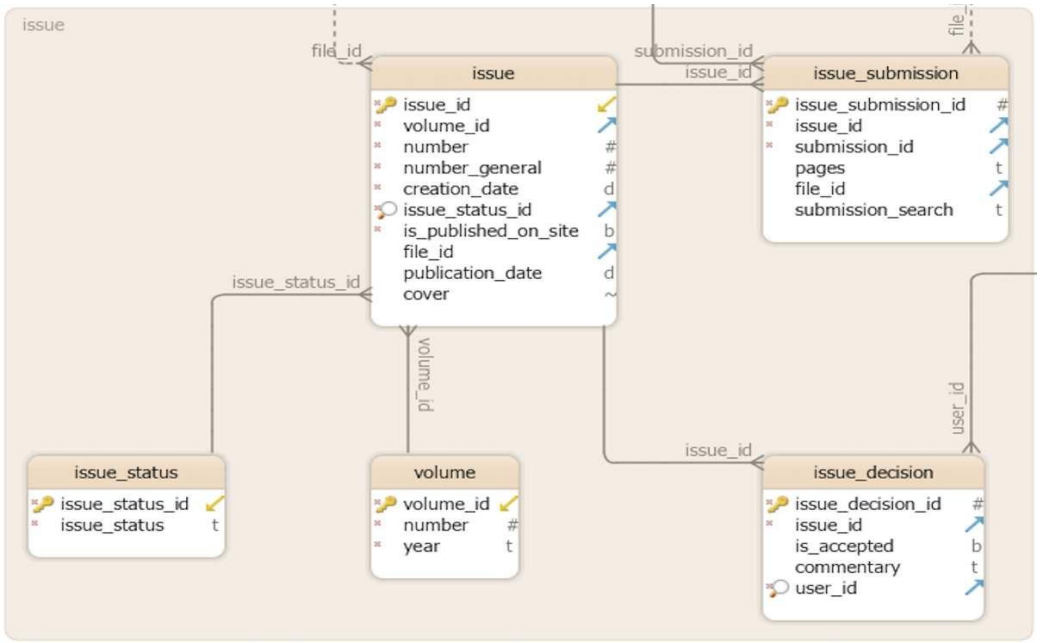


Рисунок 3.4 – Частина 3 схеми бази даних

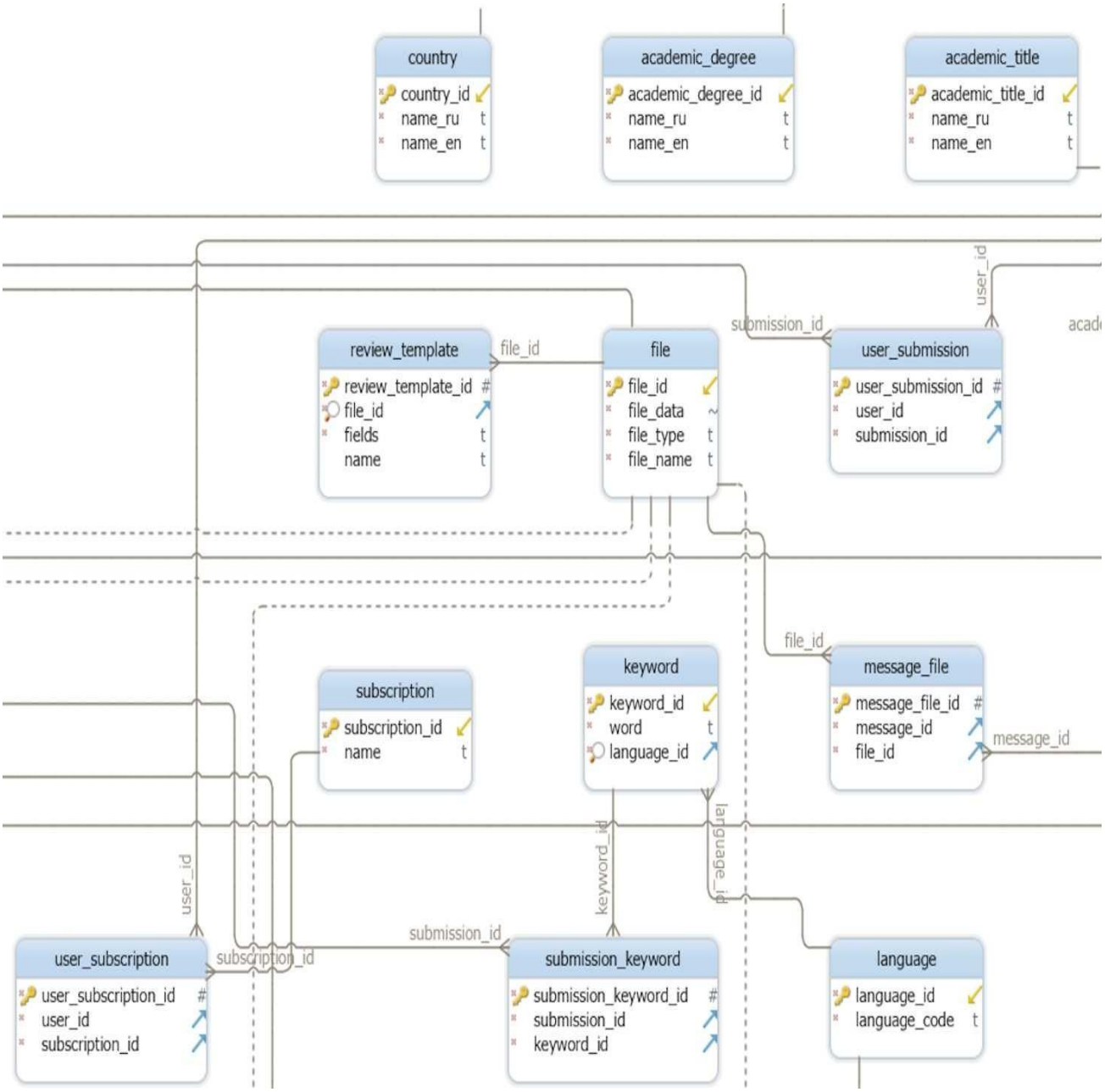


Рисунок 3.5 – Частина 4 схеми бази даних

В даний час успіх веб-сайту багато в чому залежить від його інтерфейсу. Грамотно спроектований інтерфейс легкий у освоєнні, його основні функції винесені на передній план, а зайві заховані. Розташування елементів управління впливає на зручність роботи з сайтом, а його зовнішній вигляд на перше враження та задоволення від використання [5].

З розвитком Інтернету та веб-технологій у користувачів складалися певні

уявлення роботи з веб-сайтами. Подібними уявленнями можуть бути: розташування навігаційної панелі, кнопок реєстрації та входу в систему у шапці сайту, контактної інформації, правил та іншого у футері сторінки та багато інших. Дані особливості безперечно

необхідно враховувати під час створення такого веб-додатку, як сайт наукового журналу.

Так як проблеми з визначенням цільової аудиторії відсутні (вчені та освічені люди з різними галузями наукових інтересів), то, ґрунтуючись на цьому, можна виділити основні вимоги до інтерфейсу:

1. лаконічність та простота;
2. використання нейтральних кольорів, які не викликають неприємних чи тривожних почуттів;
3. використання ненав'язливої, «м'якої» анімації графічних компонентів;
4. наявність англомовної версії сайту;
5. адаптація інтерфейсів під різні розміри екранів мобільних пристроїв внаслідок зростання їхньої поширеності на ринку [6].

Більше того, до сайтів наукових журналів пред'являються інші різнобічні вимоги щодо змісту веб-сторінок журналу, так і їх візуального представлення [27, 28]. Вимоги стосуються структури інтерфейсу сайту, наявності метаданих у HTML-коді сторінок, відсутності реклами, явних вказівок на правила та політику журналу.

У ході розробки прототипу інтерфейсу веб-платформи, були створені начерки сторінок програми. Макет головної сторінки представлений рисунку 9.

3.3 Фізична модель веб-додатку

Розробка клієнтської та серверної частин велася в редакторі вихідного коду Visual Studio Code (VS Code), оскільки VS Code є безкоштовним, невибагливим до

ресурсів, підтримує безліч мов програмування, автодоповнення, підсвічування синтаксису, налагодження програм, плагіни та інше.

Частиною початку розробки є визначення файлової структури проекту. Структура представлена рисунку 3.6.

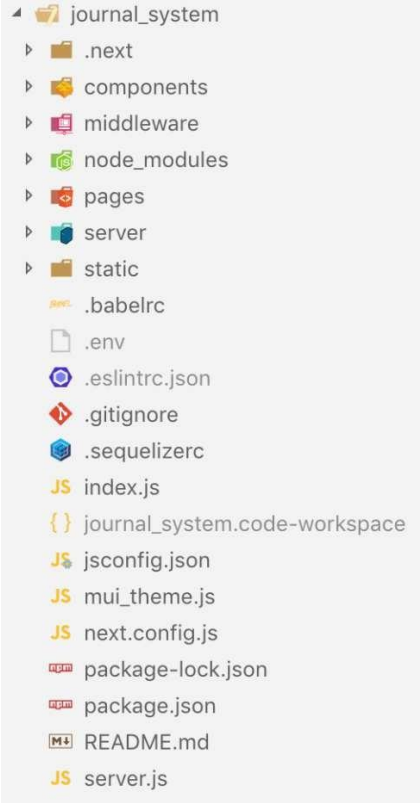


Рисунок 3.6 – Файлова структура проекту

Файли ініціалізації та налаштувань знаходяться у кореневому каталозі. Кореневий каталог включає наступні директорії:

1. next – створюється та керується однойменним фреймворком;
2. node_modules – створюється менеджером пакетів npm, сховище встановлених пакетів;
3. components – React-компоненти;
4. middleware – методи фронтенду, а також проміжний код, необхідний як

клієнтській, так і серверній стороні;

5. pages – сторінки веб-сайту;

6. server – файли, що стосуються тільки серверної сторони, такі як: контролери, моделі та інше;

7. static – статичні файли, призначені для заповнення веб-сторінок.

3.4 Діаграма класів

Першим етапом розробки серверної сторони було визначення об'єктних моделей за допомогою бібліотеки ORM Sequelize. Для цього в директорії server був створений файл налаштувань sequelizerc і описані моделі в папці models. Приклад моделі, що ґрунтується на таблиці ролей користувачів, представлений на рисунку 3.7.

```
module.exports = (sequelize, DataTypes) => {
  const Role = sequelize.define('role', {
    role_id: {
      allowNull: false,
      autoIncrement: true,
      primaryKey: true,
      type: DataTypes.SMALLINT
    },
    role_name: {
      type: DataTypes.STRING,
      allowNull: false
    }
  }, {
    freezeTableName: true,
    timestamps: false
  });
  Role.associate = models => {
    Role.belongsTo(models.UserRole, {
      foreignKey: 'role_id',
      targetKey: 'role_id'
    });
  };
  return Role;
};
```

Рисунок 3.7 – Модель Role

Бібліотека Sequelize надає можливість реалізації міграцій та попереднього

заповнення бази даних для спрощення розгортання програми.

У директорії server також були створені папки migrations та seeders. Файли migrations описують створення таблиць та його полів. Приклад файлу міграції представлений рисунку 3.8.

```
module.exports = {
  up: (queryInterface, Sequelize) => {
    return queryInterface.createTable('role', {
      role_id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.SMALLINT
      },
      role_name: {
        type: Sequelize.STRING,
        allowNull: false,
      }
    });
  },
  down: (queryInterface, Sequelize) => {
    return queryInterface.dropTable('role');
  }
};
```

Рисунок 3.8 – Міграційний файл для таблиці ролей користувача

Файли міграції були написані кожної таблиці бази даних, що дозволяє описати базу даних однією командою в терміналі.

У веб-платформі, що розробляється, існують такі таблиці, зміст яких повинен бути зумовлений (наприклад, назви країн, ролей системи). Для виконання цього завдання було створено директорію seeders, вміст якої описує вміст таблиці бази даних. Приклад функції, що виконує попереднє заповнення таблиці ролей, представлений на рисунку 3.9. Виклик операції автозаповнення простий, як і міграції.

```
module.exports = {
  up: (queryInterface, Sequelize) => {
    return queryInterface.bulkInsert('role', [
      { role_name: 'Администратор' },
      { role_name: 'Автор' },
      { role_name: 'Ответственный секретарь' },
      { role_name: 'Главный редактор' },
      { role_name: 'Рецензент' },
      { role_name: 'Член редакционной коллегии' }
    ], {});
  },

  down: (queryInterface, Sequelize) => {
    return queryInterface.bulkDelete('role', null, {});
  }
};
```

Рисунок 3.9 – Функція, що заповнює таблицю ролей системи

Наступний етап полягав у написанні функцій, що реалізують прикладну логіку веб-платформи, що розробляється. Даним функціям було виділено директорія методів. Файли з директорії методів містять експортовані функції, які використовуються контролерами.

Перш ніж реалізовувати контролери, необхідно було скласти карту API-шляхів веб-платформи. Кожній ролі системи було виділено свій шлях до запитів. При створенні назв шляхів використовувалися лише іменники, наприклад, шлях отримання адміністратором всіх користувачів системи це /api/admin/users, а отримання одного користувача системи це /api/admin/user/:slug, де slug – унікальний ідентифікатор користувача. Подібне найменування спрощує розуміння та використання API [30].

Робота зі створеними шляхами здійснюється за допомогою об'єкта Router фреймворку Express. У директорії controllers був створений файл index.js (рисунок 3.10), що реалізує направлення запитів у відповідні контролери згідно з API-шляхами.

```
const router = express.Router();

router.get(ADMIN_USERS, async (req, res) => {
  try {
    const result = await getUsers();
    res.status(200).json(result);
  } catch (err) {
    res.status(err.status || 500).json({ message: err.message || err.toString() });
  }
});

router.post(ADMIN_NEW_USER, async (req, res) => {
  try {
    await createUser({ data: req.body, isAdmin: true });
    res.status(200).end();
  } catch (err) {
    res.status(err.status || 500).json({ message: err.message || err.toString() });
  }
});

router.put(ADMIN_USER, async (req, res) => {
  try {
    await editUser(req.params.slug, req.body);
    res.status(200).end();
  } catch (err) {
    res.status(err.status || 500).json({ message: err.message || err.toString() });
  }
});

router.delete(ADMIN_USER, async (req, res) => {
  try {
    await deleteUser(req.params.slug);
    res.status(200).end();
  } catch (err) {
    res.status(err.status || 500).json({ message: err.message || err.toString() });
  }
});
```

Рисунок 3.10– Фрагмент коду файлу index.js у директорії controllers

Контролери, своєю чергою, приймають за запитом користувача і виконують функції, які відповідають за бізнес-логіку.

3.5 Діаграма послідовності

Завдяки подібній реалізації бекенда контролери виглядають мінімалістично та зрозуміло, а їх додавання та зміни в карті API-шляхів не потребують великої кількості дій.

Оскільки Journal System розділяє функціонал користувачів згідно з присвоєними їм ролями, необхідно було розробити механізм аутентифікації та авторизації. Для виконання цього завдання було вирішено використовувати бібліотеку Passport.js, яка надає спрощений підхід до реалізації так званих стратегій

аутентифікації.

Для роботи з бібліотекою в директорії server був створений файл конфігурації passport.js, в якому були описані «стратегії» локальної аутентифікації (через електронну адресу та пароль) та через сторонні системи Google та Facebook (для яких потрібно було отримати ключі клієнта та секретні ідентифікатори програми на порталах розробника даних систем).

Щоб ідентифікувати клієнта, веб-платформа поміщає cookie у браузер користувача (що виконано з використанням бібліотеки cookie-session, яка не зберігає cookie на стороні сервера, а лише на клієнті). Для зберігання та вилучення інформації, що зберігається в куки, були розроблені методи serializeUser і deserializeUser. Функція serializeUser, використовуючи заданий секрет, кодує ідентифікатор користувача в cookie і зберігає їх у браузері клієнта. При зверненні клієнта до сервера, сервер отримує куки, Passport.js за допомогою методу deserializeUser витягує унікальний ідентифікатор, що зберігається в куки, перевіряє його валідність, а також звертається до бази даних для встановлення прав користувача.

Авторизація користувача виконана у функції authorize (рисунок 3.11).

```
function authorize( ...roles) {
  return async (req, res, next) => {
    try {
      if (!req.isAuthenticated()) {
        return res.redirect(302, '/?unauthorized=true');
      }

      const user = req.user;

      if (roles.length === 0) {
        return next();
      }

      if (user.roles.some(role => roles.includes(role))) {
        return next();
      }

      res.redirect(302, '/');
    }
    catch (err) {
      res.status(500).send({ message: 'Ошибка на сервере' });
    }
  }
}
```

Рисунок 3.11 – Функція авторизації

Функція авторизації служить як проміжна функція і отримує вході масив ролей, авторизованих до виконання операції. Якщо ролі не вказані, вважається, що авторизовані всі автентифіковані запити. Приклад використання цієї функції було представлено у фрагменті коду рисунка 18.

3.6 Рішення по шифровці даних

Початковим етапом розробки клієнтської частини веб-платформи було створення React-компонентів, з яких будуються сторінки сайту. У директорії components були створені папки main та dashboard. Перша містить компоненти, що стосуються тільки сторінок, які видно всім відвідувачам (наприклад, головна сторінка журналу, сторінка номера, статті та інше), друга – компоненти, які використовуються на внутрішніх сторінках сайту, доступних тільки

авторизованим користувачам. Проміжні компоненти, які використовуються на всіх сторінках, розміщуються в кореневому каталозі папки components.

Приклад компонента представлений на рисунку 3.12. Компонент Layout служить розміткою для основних сторінок веб-платформи.

```
const useStyles = makeStyles(theme => ({
  grow: {
    flexGrow: 1
  },
  body: {
    minHeight: '100vh',
    display: 'flex',
    flexDirection: 'column',
    backgroundColor: theme.palette.secondary.main
  },
  content: {
    flex: '1 0 auto'
  }
}));

function Layout(props) {
  const classes = useStyles();
  const { isAuthenticated, value, children, loginOpen } = props;

  return (
    <div className={classes.body}>
      <NavBar isAuthenticated={isAuthenticated} value={value} loginOpen={loginOpen} />
      <div className={classes.content}>
        <Grid container className={classes.grow} justify="center">
          <Grid container item xs={10}>
            { children }
          </Grid>
        </Grid>
      </div>
      <Footer />
    </div>
  );
}
```

Рисунок 3.12 – Компонент Layout

Слід зазначити, що під час створення компонентів React використовувалися "Хукі" (англ. Hooks), нововведення бібліотеки версії 16.8. Використання "Хуків" вимагає розробки компонентів у вигляді функцій JavaScript, а не класів, що робить код більш лаконічним і ємним [31].

Використання «хуків» дозволило просто і ємно реалізувати механізм отримання інформації про авторизованого користувача на сторінках системи.

Для цього у файлі auth.js (рисунок 3.13) директорії middleware було створено об'єкт типу Context.

```
import { createContext } from 'react';

const Auth = createContext();

export default Auth;
```

Рисунок 3.13 – Файл auth.js

Далі у файл _app.js (файл Next.js для формування всіх сторінок сайту) імпортувався об'єкт Auth, у параметр value методу Auth.Provider передавався отриманий об'єкт користувача і вміст сторінки обертався у наведений вище метод (рисунок 3.14).

```
<Auth.Provider value={user}>
  <Component {...pageProps} />
</Auth.Provider>
```

Рисунок 3.14 – Передача даних про користувача у файлі _app.js

Отримання даних про користувача зі сторінок системи здійснюється простим викликом «хука» useContext (рисунок 3.15).

```
const { isAuthenticated, roles } = useContext(Auth);
```

Рисунок 3.15 – Приклад отримання інформації про користувача

Також на рисунку 3.12 продемонстровано використання функції makeStyles бібліотеки Material UI. Об'єкт theme, що передається як параметр у функцію зворотного дзвінка, містить інформацію про встановлених значеннях відступів, основних та вторинних кольорах та інші косметичні особливості. Цей об'єкт описаний у файлі mui_theme.js (рисунок 3.16), що знаходиться в кореневому каталозі репозиторію Journal System.

```
const theme = createMuiTheme({
  palette: {
    primary: {
      main: '#6f79a8', //#00796b
      light: '#9fa8da',
      dark: '#424d79'
    },
    secondary: {
      main: '#fafafa',
      light: '#ffffff',
      dark: '#c7c7c7'
    },
    error: {
      main: '#e57373',
      dark: '#ef5350'
    },
    status: {
      accepted: green[600],
      rejected: red[600],
      new: blue[800],
      other: indigo[800]
    },
    text: {
      hint: "#c7c7c7"
    },
    icon: {
      main: grey[600]
    }
  },
  typography: {
    body2: {
      fontSize: '1rem',
    },
    body1: {
      fontSize: '0.875rem',
    },
    h6: {
      color: 'rgba(0, 0, 0, 0.87)'
    }
  },
  status: {
    success: green[400],
    error: red[400],
    info: indigo[400],
    warning: orange[400]
  }
});
```

Рисунок 3.16 – Об'єкт theme файлу mui_theme.js, що містить інформацію про косметичні параметри веб-програми

У разі потреби ця особливість робить зміну вигляду веб-сайту досить простим завданням, що вимагає редагування декількох рядків файлу mui_theme.js.

Щоб спростити відстеження помилок при розробці компонентів, було прийнято рішення використовувати бібліотеку PropTypes, яка виконує валідацію параметрів, що передаються. Приклад використання представлений малюнку 3.17.

```
Layout.propTypes = {
  value: PropTypes.number,
  children: PropTypes.oneOfType([
    PropTypes.object, PropTypes.array
  ]).isRequired,
  isAuthenticated: PropTypes.bool.isRequired,
  loginOpen: PropTypes.bool
};
```

Рисунок 3.17 – Використання PropTypes у компоненті Layout

Якщо параметри, що одержуються компонентом, визначені, бібліотека надсилає повідомлення в консоль браузера при розбіжності отриманого типу параметра з описаним.

Для зберігання методів звернення до серверної сторони в директорії middleware була створена папка api. Приклад функції отримання персональних даних представлений малюнку 3.18.

```
export const getCredentials = cookie ⇒
  sendRequest(ACCOUNT_BASE_PATH + ACCOUNT_CREDENTIALS, {
    method: 'GET',
    headers: requestHeaders(cookie),
    credentials: 'include'
  });
```

Рисунок 3.18 – Функція запиту персональних даних з боку клієнта

Динамічне заповнення метаданих сторінок у тезі <head> було реалізовано за допомогою методів бібліотеки Next.js. Так, наприклад, сторінка статті для кращої індексації пошуковими сервісами, зокрема системою Google Scholar, повинна

містити метатеги з інформацією про авторів та статтю [32]. Наповнення тега <head> сторінки статті представлено малюнку 3.19.

```
<Head>
  <title></title>
  <meta name="citation_title" content={`${article.submission.title_ru} ${general.journalName.value}`} />
  { article.authors.map(author => (
      <meta name="citation_author" content={getFullName(author, 'ru')} />
    ))
  }
  <meta name="citation_publication_date" content={parseDate(article.issue.publication_date)} />
  <meta name="citation_journal_title" content={general.journalName.value} />
  <meta name="citation_volume" content={article.issue.volume.number} />
  <meta name="citation_issue" content={article.issue.number} />
  <meta name="citation_firstpage" content={article.pages.split('-')[0]} />
  <meta name="citation_lastpage" content={article.pages.split('-')[1]} />
  <meta name="description" content={article.submission['abstract_ru']} />
</Head>
```

Рисунок 3.19 – Заповнення тега <head> на сторінці статті

Для зручності керування контентом сторінок веб-застосунку, в директорії static була створена папка page_content, що містить json-файли, що зберігають зміст сторінок системи.

3.7 Загальносистемні рішення

В результаті розробки серверної частини системи було реалізовано бізнес-логіку, складено API-шляхи та розроблено відповідні методи.

Розроблені методи API описані у таблицях 3.2–3.6.

Таблиця 3.2 – Базовий API

№	Метод	Найменування	Опис
1	POST	registration	Реєстрація користувача в системі за допомогою, як і зв'язування логін-пароль, так і сторонніх сервісів Google та Facebook.

Продовження таблиці 3.2 – Базовий API

№	Метод	Найменування	Опис
2	POST	login	Вхід в систему за допомогою зв'язування логін- пароль, Google чи Facebook.
3	POST	logout	Вихід з системи.
4	GET	getIssues	Повертає всі номери, що вийшли.
5	GET	getIssue	Повертає номер за ID.
6	GET	getCurrentIssue	Повертає поточні номери.
7	GET	getArticle	Повертає статтю з ID.
8	POST	search	Повертає результати пошуку за статтями журналу.
9	GET	downloadFile	Завантаження файлу.

Таблиця 3.3 – API користувача

№	Метод	Найменування	Опис
1	GET	getCredentials	Повертає особисті дані користувача.
2	PUT	editCredentials	Редагує особисті дані користувача.
3	GET	getSettings	Повертає налаштування користувача, включаючи передплати.
4	PUT	changeEmail	Змінює електронну адресу користувача.
5	PUT	changePassword	Змінює пароль користувача.

Таблиця 3.4 – API адміністратора системи

№	Метод	Найменування	Опис
1	GET	getUsers	Повертає інформацію про користувачів системи.
2	POST	createUser	Створює користувача.
3	PUT	editUser	Зміна інформації про користувача, у тому числі його ролі.
4	DELETE	deleteUser	Безповоротно видаляє користувача з системи.

Таблиця 3.5 – API головного редактора

№	Метод	Найменування	Опис
1	GET	getIssues	Повертає номери у стадії формування.
2	GET	getIssue	Повертає номери у стадії формування за ID.
3	POST	decide	Виносить остаточне рішення про публікації номерів.

Таблиця 3.6 – API члена редакційної колегії

№	Метод	Найменування	Опис
1	GET	getIssues	Повертає номери у стадії формування.
2	GET	getIssue	Повертає номер у стадії формування за ID.
3	POST	vote	Виносить рекомендаційне рішення про публікації номерів.

4 ВИПРОБУВАННЯ ТА ДЕМОНСТРАЦІЯ ВЕБ ДОДАТКУ ДЛЯ КОНТРОЛЮ ТА ОРГАНІЗАЦІЇ РОБОЧОГО ПРОЦЕСУ

4.1 Загальні положення випробування веб-додатку

Для розробки практичної частини курсової роботи було обрано створити клієнтський додаток, який буде відображати пройдені курси студентів за певний період навчання у ЗВО. Студенти зможуть проходити опитування по пройденимними курсам. Дані будуть надсилатися на сервер через Web API, який буде надсилати, оброблювати та зберігати ці дані. Окрім студентів, додаток буде підтримувати функціонал для перегляду результатів опитування адміністрацією ЗВО. Також складовою цього застосунку буде сторінка адміністраторів ЗВО, якізможуть вносити та змінювати дані про студентів, викладачів, курси, структурні підрозділи, спеціальності тощо. Для кожної ролі буде призначено відповідні права доступу до елементів застосунку. Наприклад, студент не зможе передивитися результати опитування та не матиме доступу до сторінки адміністратора застосунку, в той час як адміністратори зможуть переглядати дані свого ЗВО на даній сторінці. В перспективах подальшої розробки проекту буде створено функціонал для проведення аналізу освітніх програм, які існують у ЗВО.

4.2 Функціональне тестування

Отже, для розробки застосунку з використанням бібліотеки Redux потрібно зробити початкову конфігурацію.

Рисунок 4.1 - Створення сховища стану програми

На рисунку 4.1 зображено код для створення сховища Redux. Функція *createStore* приймає у параметрах визначений нами регулятор, а також проміжнеПЗ за необхіодності його використання. Таким чином, *rootReducer* – це окремийрегулятор, який був скомпонований з усіх прописаних регуляторів в ході розробки програми, а *composeEnhancers* – це функція, яка за наявності встановленого розширення Redux

DevTools підключає наше сховище для відображення у окремій вкладці інструментів розробника у використовуваному браузері. Якщо розширення не було завантажено, то використовується функція compose, яка компонує усі надані проміжні ПЗ, надані через *applyMiddleware* функцію, для підтримки додаткових функцій. У цьому застосунку використовується *redux-thunk*. Це ПЗ, як вже зазначалося у розділі 2.1, надає диспетчеру підтримку здійснення асинхронних функцій.

```
import institutions from "./institution";
import surveys from "./survey";
import results from "./result"
import admin_data from "./adminData"
import sidebar from "./sidebar"
import {reducer as formReducer} from "redux-form";

const appReducer = combineReducers( reducers: {
  auth: auth,
  institutions: institutions,
  surveys: surveys,
  results: results,
  admin_data: admin_data,
  sidebar: sidebar,
  form: formReducer
});

const rootReducer =  (state, action) => {
  if (action.type === ACTION_TYPES.SIGN_OUT) {
      state = undefined;
  }
  return appReducer(state, action)
};


export default rootReducer;
```

Рисунок 4.2 - Об'єднання регуляторі

На рисунку 4.2 продемонстровано код з файлу index.js папки redux. Суть цього коду – це об’єднання усіх регуляторів в один, який буде передаватися сховищу Redux. Для цього використовується функція*combineReducers()*, яка приймає об’єкт з усіма визначеними регуляторами та повертає загальний регулятор. В кінці файлу ми експортуємо отриманий регулятор для імпорту його у наше сховище.

Остання ланка ініціалізації сховища для використання у компонентах – це передача підготовленої функції *store()* компоненту – обгортці кореневого

компонента App нашого застосунку, а саме Provider, який був детально

розглянутий у розділі 2.2. Ці дії висвітлено на рисунку 4.3.

```
import React from "react";
import ReactDOM from "react-dom";
import {Provider} from "react-redux";
import store from "./redux/store";

import App from "./scenes/App/App";

ReactDOM.render(
    <Provider store={store()}>
        <App/>
    </Provider>,
    document.querySelector("#root")
);
```

Рисунок 4.3-Передача компоненту Provider функції store() для створення сховищаRedux

У результаті наших дій було сконфігуровано глобальне сховище, у якому будуть триматися дані про поточний стан нашого застосунку і до якого будуть звертатися наші компоненти для отримання та оновлення даних за принципами Flux архітектури.

Реалізація клієнтської архітектури сервісу на основі підходу Flux на прикладі компонентів Survey та Surveys, які відповідають за показ пройдених студентами курсів та створення відгуків на дисципліни.

Інтерфейс користувача було розроблено у вигляді функціональних та класових компонент, щоб показати застосування і хуків бібліотеки Redux, і функції *connect()*.

Спочатку переглянемо реалізацію сторінки курсів студента. На рисунку 4.4 наведено інтерфейс цієї сторінки. Отримання даних про курси відбувається шляхом відправки HTTP запиту на сервер. Після успішної обробки відповіді, ці дані переходять у сховище. Так як у компоненті Surveys відбувається зв’язування з даними курсів у відповідному вузлі об’єкта стану, компонент отримує оновлені дані

та відображає їх у вигляді таблиці. Далі студент обирає курс, на який він хоче написати відгук та переходить на сторінку з анкетою.

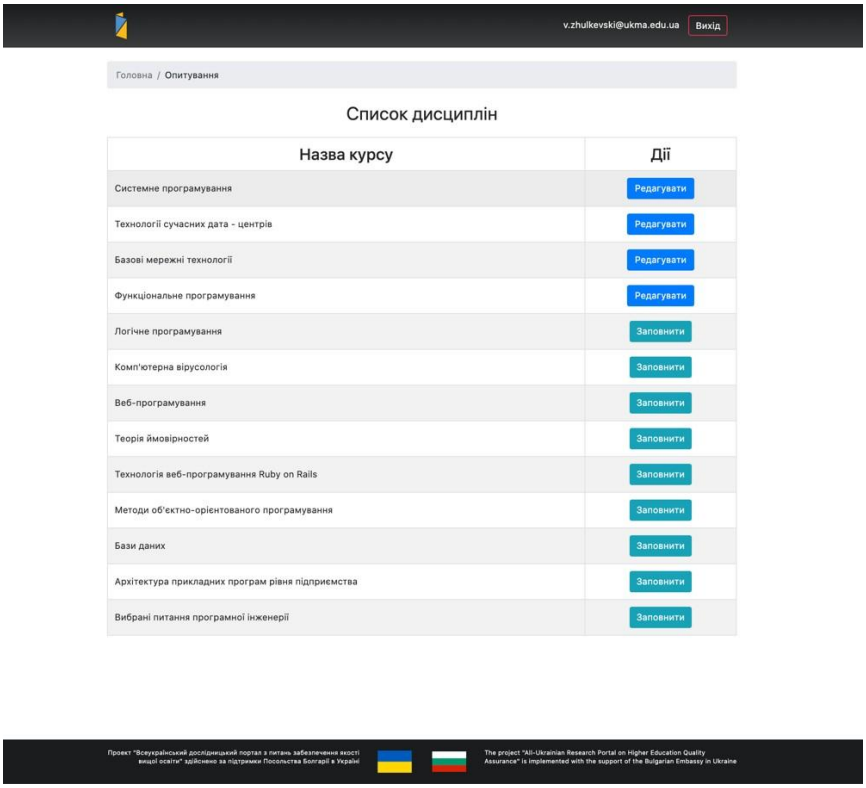


Рисунок 4.4 - Сторінка опитувань студента

Тепер опишемо детальніше, що відбувається у коді самого застосунку.

Увесь процес починається при першому відображенні компоненту Surveys (Додаток Б). Спочатку ми визначаємо, що наш класовий компонент буде отримувати через функції connect(), mapStateToProps і mapDispatchToProps такі дані зі сховища: user (інформація про поточного користувача), surveys(список усіх курсів) та fetchSurveys (генератор дії). Так як, потрапляючи перший раз на сторінку, ми не маємо дані про курси, то відображається спеціальний компонент Loader, який показує, що дані ще не підвантажилися з сервера. В цей час компонент за допомогою генератора дій fetchSurveys() у так званому *lifecycle* методі *ComponentDidMount()*, який виконується тільки один раз після першого відображення компоненту, продукує функцію та відправляє її на обробку

диспетчером. Диспетчер за допомогою Redux-thunk та HTTP клієнту axios відправляє запит на сервер аби отримати інформацію про курси. Як тільки сервер надсилає відповідь, генерується дія з типом “FETCH_USER_SURVEYS”і даними про курси у полі payload та передається на обробку регулятору.

4.3 Нефункціональне тестування

```
const doFetchUserSurveys = (id) => async dispatch => {
    api.get('/student/' + id + "/course/survey").then((response) => {
        dispatch({type: ACTION_TYPES.FETCH_USER_SURVEYS, payload: response.data.data})
    }).catch(() => {
        addNotification( message: "Не вдалося отримати курси студента!");
    });
};
```

Рисунок 4.5 - Генератор дії для передачі отриманих курсів в сховище програми

Отримавши дію, регулятор порівнює тип цієї дії із кожним значенням у конструкції switch. Якщо назва дії збігається із визначеною в case, тоді здійснюється “зміна” стану. Все, що було у попередньому стані, передається у новий із зміною потрібних полів. В даному випадку в нас виконуватиметься case з назвою FETCH_USER_SURVEY. Він збереже попередній стан сховища та оновить поле surveyList даними, переданими у дії в полі payload.

```
function reducer(state = initialState, action) {
    switch (action.type) {
        case ACTION_TYPES.GET_STUDENT_INFO:
            return {...state, student: action.payload};
        case ACTION_TYPES.FETCH_USER_SURVEYS:
            return {...state, surveysList: action.payload};
        case ACTION_TYPES.FETCH_USER_SURVEY:
            return {...state, currentSurvey: action.payload};
        case ACTION_TYPES.POST_USER_SURVEY:
            return {...state, currentSurvey: null, errors: null};
        case ACTION_TYPES.DISPLAY_ERRORS:
            return {...state, errors: action.payload};
        case ACTION_TYPES.REMOVE_ERROR:
            return {...state, errors: state.errors.filter((err)=>err !== action.payload)};
        default:
            return state;
    }
}
```

Рисунок 4.6 - Регулятор survey

Після цього оновлені дані надходять до компоненту Surveys та відображаються у вигляді таблиці. Для перегляду поточного стану сховища нашого застосунку скористаємось розширенням браузера Redux DevTools.

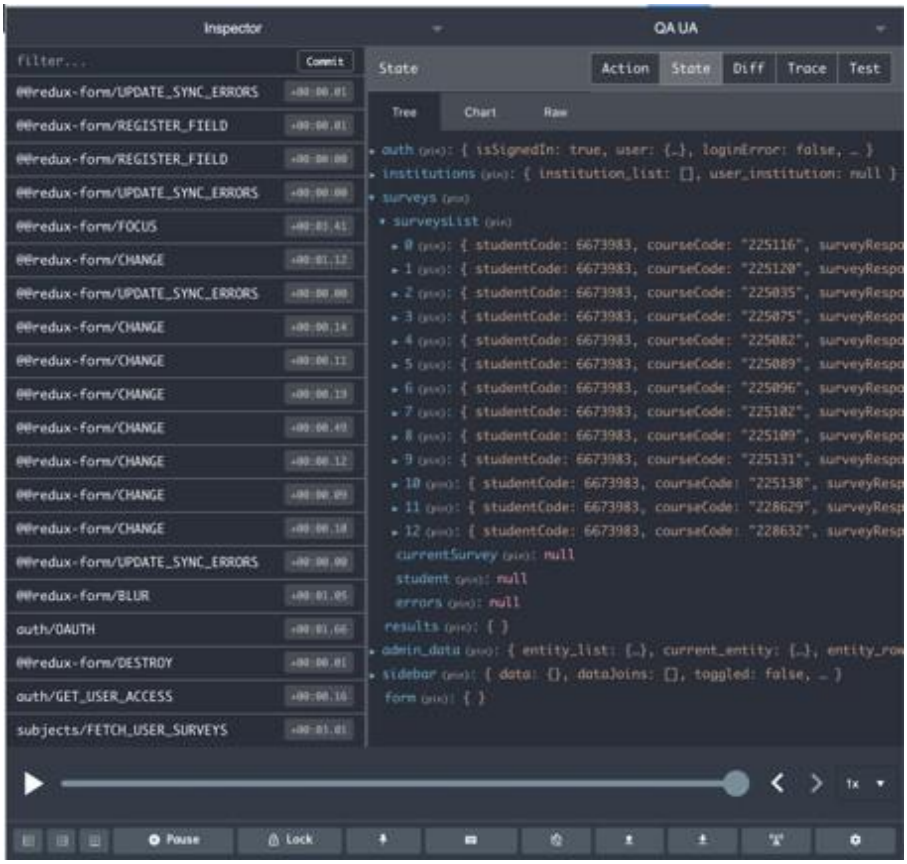


Рисунок 4.7 - Стан сховища у Redux DevTools

У такий же спосіб давайте проаналізуємо сторінку заповнення анкети, за відображення якої відповідає функціональний компонент Survey (Додаток Б). Інтерфейс сторінки продемонстровано на рисунку 4.7. При початковому завантаженні цієї сторінки здійснюється отримання інформації про користувача, форму курса та помилок зі сховища даних. Однак цього разу ми використаємо хуки *useSelector()* та *useDispatch()*, робота яких детально описані у пункті 4.3. На рисунку 4.6 вказано код компонента Survey, який відповідає за отримання вищезазначених

даних.

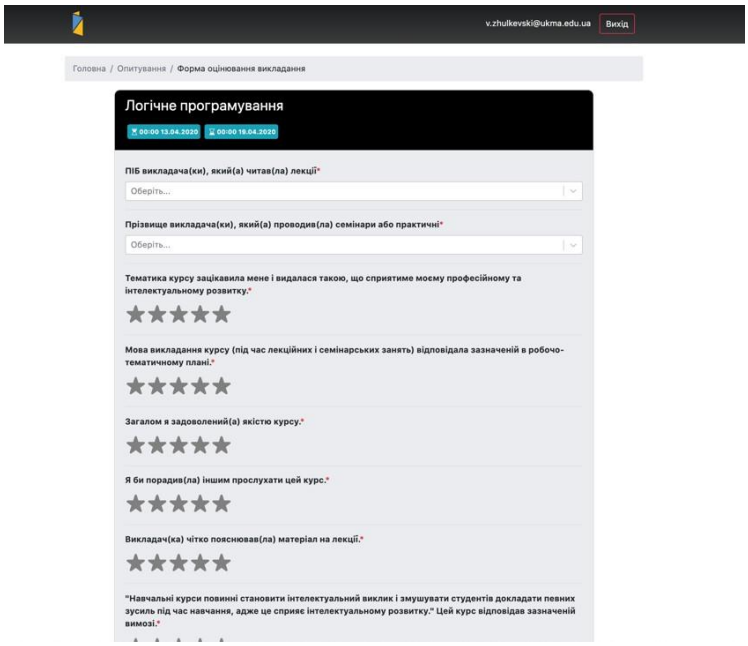


Рисунок 4.4 - Інтерфейс сторінки заповнення анкети

```
const Survey = (props) => {
    const user =useSelector(state=>state.auth.user);
    const survey = useSelector(state=>state.surveys.currentSurvey);
    const errors = useSelector(state=>state.surveys.errors);
    const dispatch = useDispatch();

    const st_id = user.person.students[0].code;
    const course_id = props.match.params.id;
    useEffect(  effect: () => {
        dispatch(actionCreators.doFetchUserSurvey(st_id, course_id));
    },  deps: [st_id,course_id,dispatch]);
```

Рисунок 4.5 - Використання хуків useSelector та useDispatch

Окрім цього, у компоненті використовується хук useEffect, представлений у бібліотеці React, принцип дії якого наступний: useEffect() приймає два параметри – колбек функцію для виклику та масив, у якому задаються залежні змінні. Кожного разу, коли змінюється одне із значень таких змінних, хук буде викликати функцію колбек та оновлювати компонент [9]. Функція-колбек обов’язково викликатиметься при початковому відображенні. Виходячи з цього, при першому завантаженні

сторінки буде викликатися функція, яка буде генерувати дію FETCH_USER_SURVEY. Генерація відбуватиметься у методі *doFetchUserSurvey()*, який зображено на рисунку 4.5.

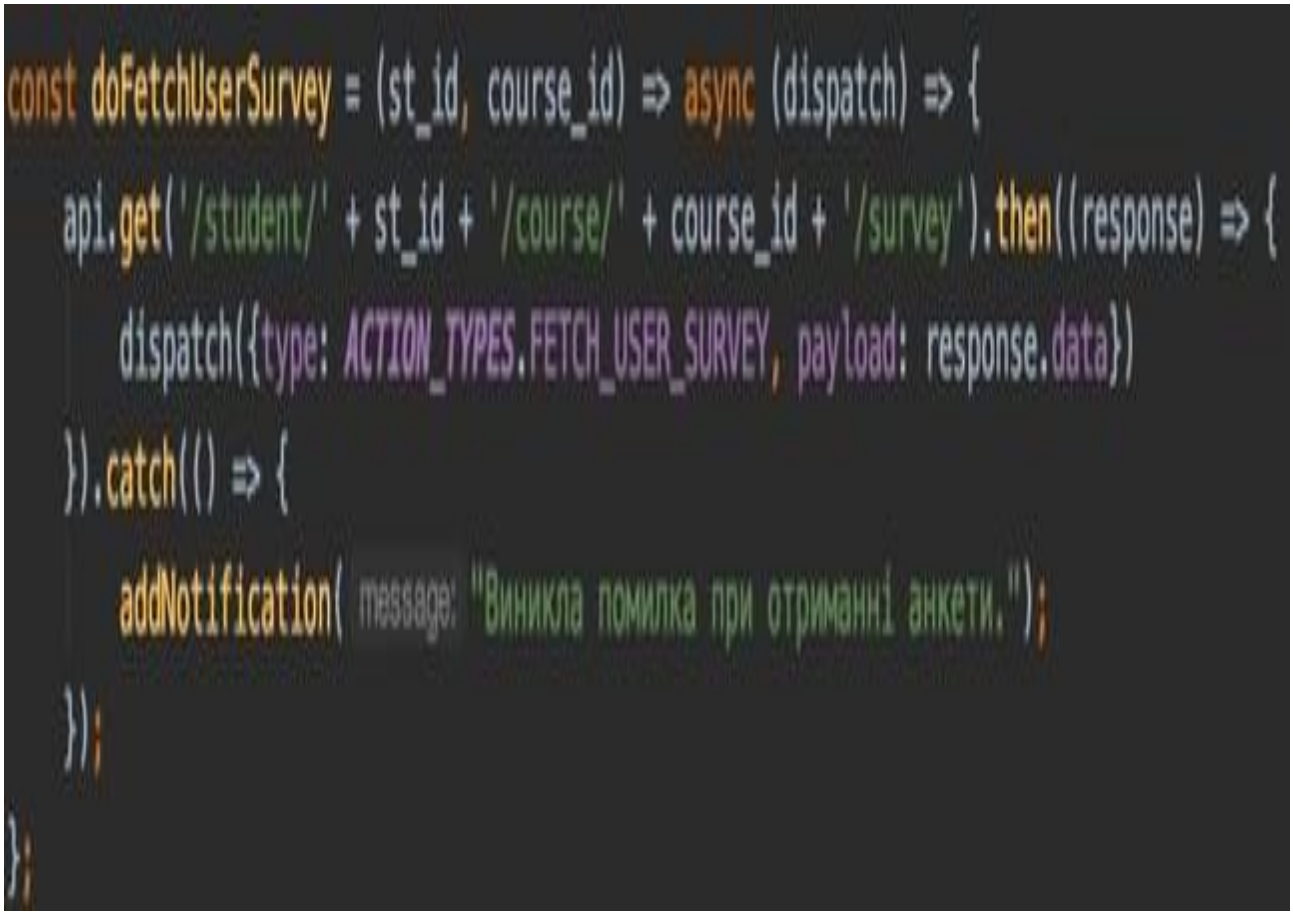


Рисунок 4.6 - Генератор дії для передачі конкретного курсу в сховище програми

Далі відбуваються ті самі процеси, які вже описувалися вище у цьому розділі. Дія надходить до регулятора, він знаходить case для створення нового стану, дані у сховищі оновлюються та отримуються у компоненті після чого компонент відображає отримані дані на сторінці. У Redux DevTools можемо переглянути отримані дані та дії, які були передані диспетчеру та оброблені регулятором.

4.4 Огляд розробленого веб-додатку

Крім цього функціоналу, компонент Survey містить дочірні компоненти Field з бібліотеки Redux Form. Ця бібліотека надає легкий спосіб керуванням

форми та збереження даних форми у стані сховища. У цій бібліотеці реалізується такий самий потік даних як і у Redux. Від розробників, які використовують цю бібліотеку, потрібно лише додати до загального регулятора регулятор із цієї бібліотеки та обгорнути компонент функцією reduxForm(), а в параметрах надати об'єкт з налаштуваннями форми, обов'язково вказавши поле form із унікальним значенням [10].

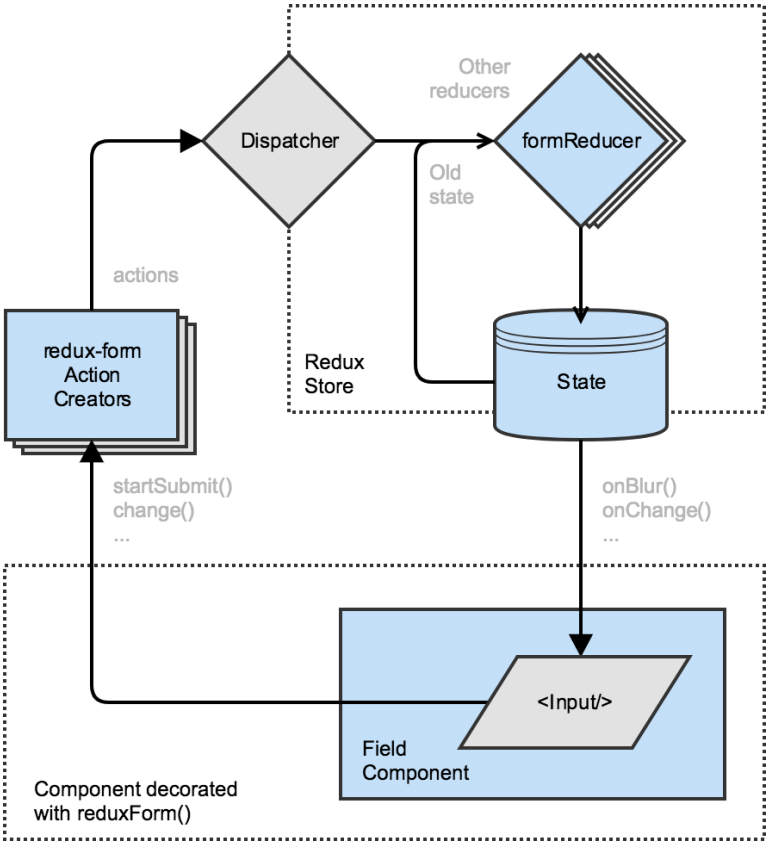


Рисунок 4.7 - Потік даних Redux Form [10]

Компонент Field реєструє поле у об'єкті форми, який є вузлом станусховища, із назвою, зазначеною в атрибуті name цього компонента. Також у пропсах Field

потрібно обов’язково вказати component, який буде відображатися для введення даних користувачем. Це можуть бути теги input, textarea, select або ж власний компонент.

Коли користувач буде взаємодіяти із компонентом Field, обгортка redux- from буде генерувати дії для зміни даних у стані сховища. При натисканні кнопки підтвердження відправки форми, ми викликаємо колбек функцію, яка має бути

обгорнута функцію *handleSubmit* для того, щоб дані форми автоматично передалися у параметри нашого колбеку. Далі просто генерується дія, відправляється диспетчеру, той надсилає дані на сервер, і в залежності від відповіді сервера, змінює стан сховища через регулятори.

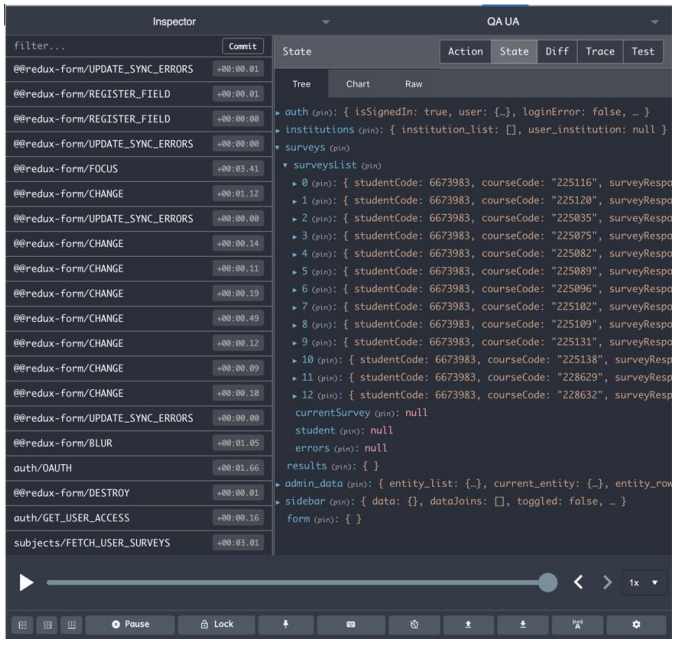


Рисунок 4.8 - Реєстрація полів форми та перегляд стану сховища у Redux DevTools

Підсумовуючи, ми бачимо, що архітектурний підхід Flux дає можливості для передбачуваного використання стану програми. За допомогою Redux DevTools можна переглядати дані у сховищі для кожного стану програми, а регулятори та генератори дій дають можливість для легкого масштабування нашого застосунку при збільшенні функціоналу нашого застосунку.

ВИСНОВКИ

У цій роботі було проведено детальний аналіз архітектури Flux. Цей архітектурний підхід виник як рішення проблеми з механізмом передачі даних по дереву компонент програмного застосунку. У цьому підході визначається чотири складові частини, а саме диспетчер, сховище, представлення та дії. Така структура забезпечує односторонній потік, що має переваги в порівнянні із двостороннім потоком. Зокрема із переваг можна виділити централізованийдоступ до даних, бо усі дані зберігаються у сховищі, зручніші можливості для дебагу застосунку та відносно легку масштабованість за рахунок створення дій та реєстрації колбеків у диспетчері.

Бібліотека Redux, яка взяла за основу цей підхід, зараз складає провідне місце у розробці клієнтських застосунків. Розробники внесли деякі корективи доскладових цього підходу, проте потік даних зберігся такий самий, як і передбачається у Flux. Redux додав регулятори, які змінюють стан сховища, змінив представлення сховища до plain old javascript object. Теж саме можна сказати і про дії, які також перетворилися у pojo та використовуються для створення нового стану застосунку. Разом із бібліотекою React Redux, Redux створює зручні інструменти для реалізації клієнтських застосунків на основі Flux підходу.

І нарешті було розроблено програмний застосунок для опитування студентів про якість вищої освіти для різних ЗВО. У застосунку використовувався Redux для реалізації Flux-подібної архітектури та на прикладі двох сторінок було продемонстровано односторонній потік даних, який забезпечується обраною архітектурою. Ці властивості дозволяють підходу Flux забезпечувати сприятиме пришвидшенню додавання нового функціоналу та легку масштабованість застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. NPM Trends [Електронний ресурс]. Режим доступу: https://www.npmtrends.com/angular-vs-react-vs-vue-vs-redux-vs-react-redux
2. Flux. Application architecture for building user interfaces [Електронний ресурс]. Режим доступу: https://facebook.github.io/flux/
3. Flux in Depth. Overview and components [Електронний ресурс]. Режим доступу: https://blog.mgechev.com/2015/05/15/flux-in-depth-overview- components/
4. What is the Flux Application Architecture? [Електронний ресурс]. https://brigade.engineering/what-is-the-flux-application-architecture- b57ebca85b9e
5. Flux Application Architecture and React [Електронний ресурс]. Режим доступу: https://freecontent.manning.com/react-in-action-flux-application- architecture/
6. Redux. A Predictable State Container for JS Apps [Електронний ресурс]. https://redux.js.org/
7. Alex Banks. Learning React: Functional Web Development with React and Redux: Підручник / Alex Banks, Eve Porcello. – К. : O'Reilly, 2017. – ст.183 – 209.
8. React Redux. Official React bindings for Redux [Електронний ресурс]. Режим доступу: https://react-redux.js.org/
9. Огляд хуків. Документація React [Електронний ресурс]. Режим доступу: https://uk.reactjs.org/docs/hooks-overview.html
10. Redux-form documentation. Field component [Електронний ресурс]. Режим доступу: https://redux-form.com/8.3.0/docs/api/field.md/

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА ВЕБ-ПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ОПИТУВАНЬ СТУДЕНТІВ З ВИКОРИСТАННЯМ БІБЛІОТЕКИ React.JS

Виконав студент 4 курсу
Групи ПД-43
Шевчук Юрій Олександрович
Керівник роботи
Асистент кафедри, аспірант Залива Віталій Вікторович

Київ – 2023

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета дослідження – спростити організацію опитувань

Об'єкт дослідження – процес опитування студентів

Предмет дослідження – веб-платформа для опитування студентів

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Вивчення та аналіз сучасних тенденцій у сфері веб-розробки та опитувань студентів.
2. Огляд існуючих систем для опитувань та аналіз їх функціональності та недоліків.
3. Вивчення бібліотеки React.js та її особливостей у контексті розробки веб-платформи.
4. Проектування архітектури платформи з використанням принципів Flux
5. Розробка фронтенду платформи з використанням бібліотеки React.js та впровадження необхідної функціональності.
6. Розробка бекенду платформи та інтеграція з базою даних для збереження опитувань та результатів.
7. Тестування роботи платформи для забезпечення її стабільності та коректності

АНАЛІЗ АНАЛОГІВ

АНАЛІЗ АНАЛОГІВ

		socrative	SurveyMonkey	wooclap	Розроблений Web-додаток
Створення та проходження опитувань	+	–	+	–	+
Створення та проходження тестів	+	+	–	+	–
Обеження по кількості учасників	–	+	–	+	–
Рейтинг викладачів	–	–	–	–	+

5

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Нефункціональні вимоги	Студентські вимоги	Користувацькі вимоги	Адміністрація
використання нейтральних кольорів,	Зручний процес відповіді студентів на питання та навігація між різними етапами опитування.;	Можливість запрошення студентів до участі в опитуваннях та керування списком учасників.	Зручний доступ до звітів та статистики для аналізу результатів опитувань.
використання ненав'язливої, «м'якої» анімації графічних компонентів;		Можливість налаштування параметрів опитувань, таких як тривалість, дата закінчення, обмеження доступу тощо.	Автоматична збір та збереження відповідей студентів.
			Генерація звітів та статистики на основі

6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Visual Studio Code

JS

Flux

HTML CSS

React JS

Redux

PostgreSQL

Потік даних Redux Form

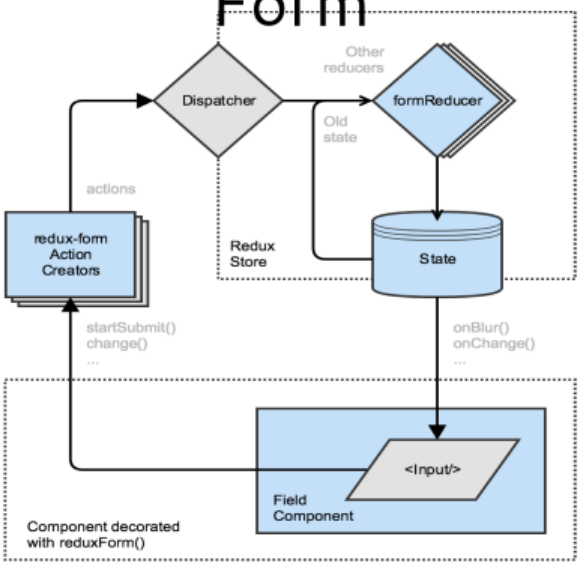


Схема бази даних

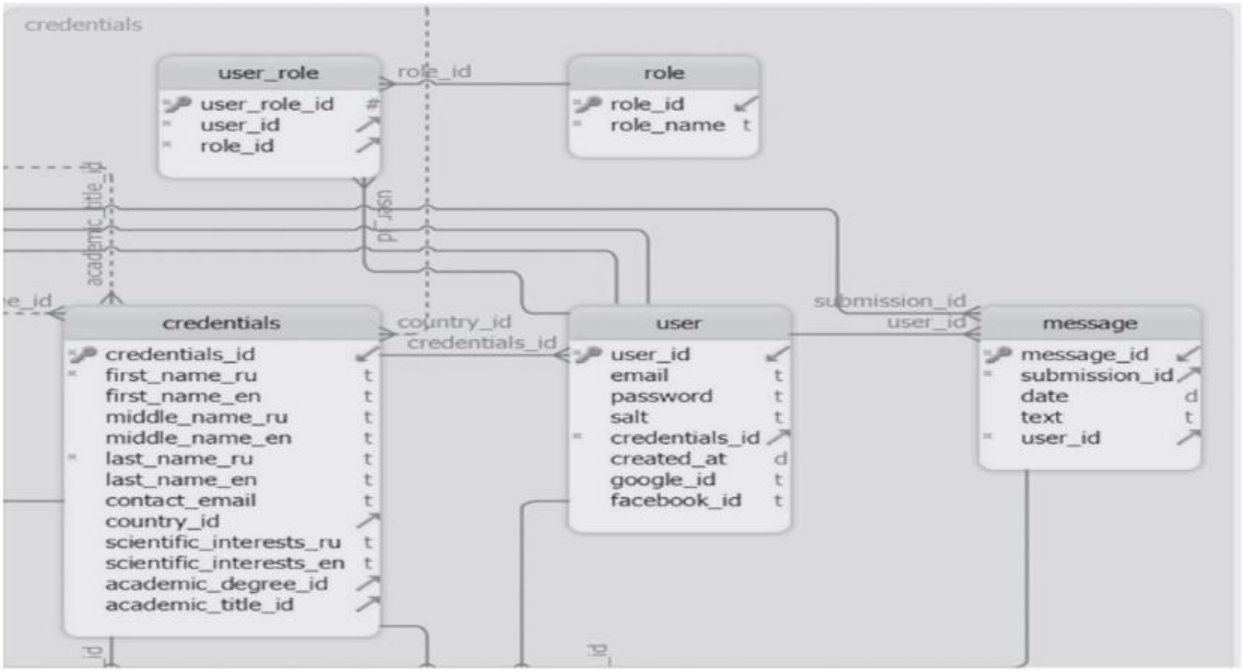


Схема бази даних

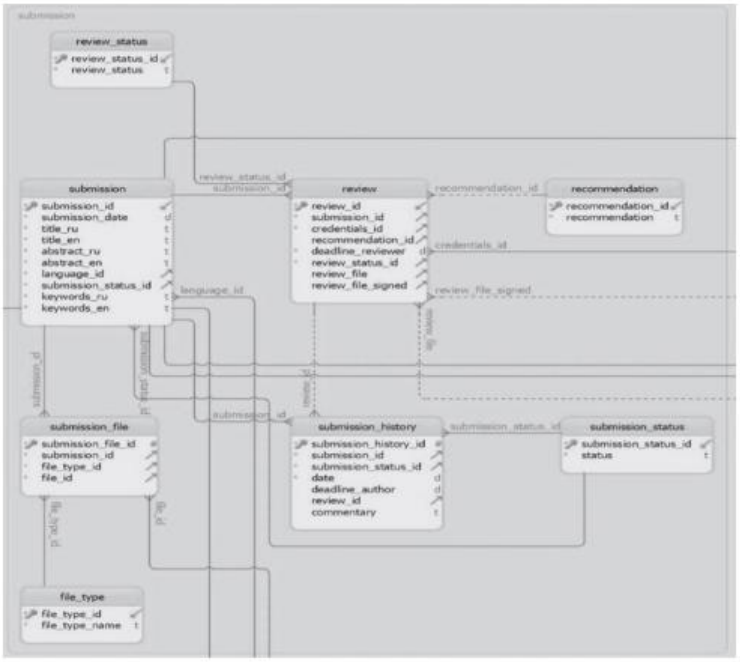


Схема бази даних

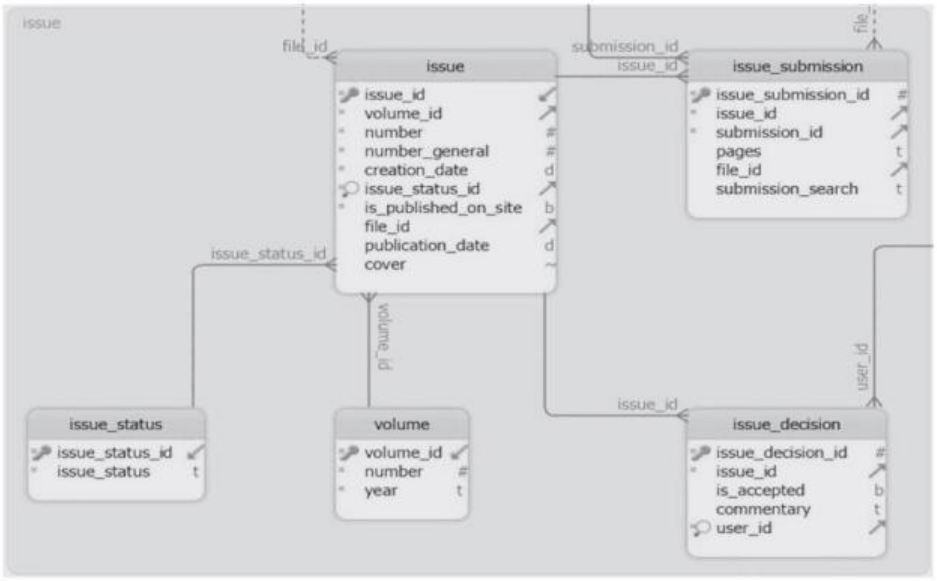
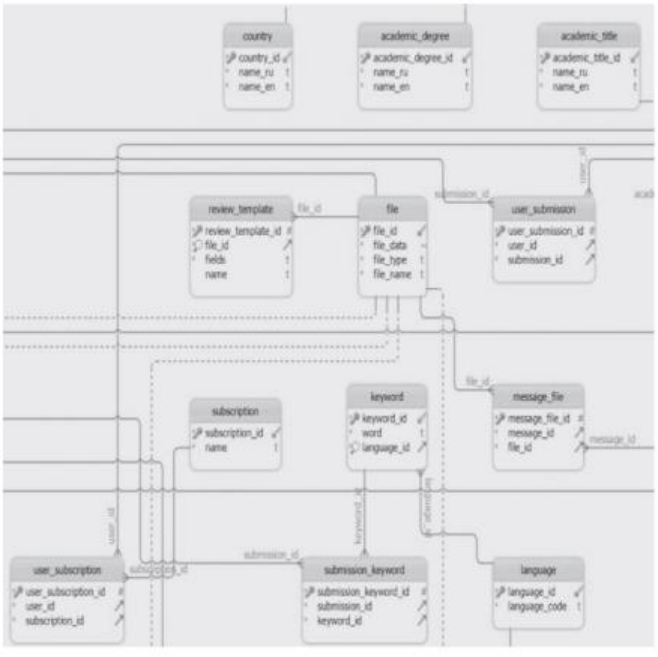
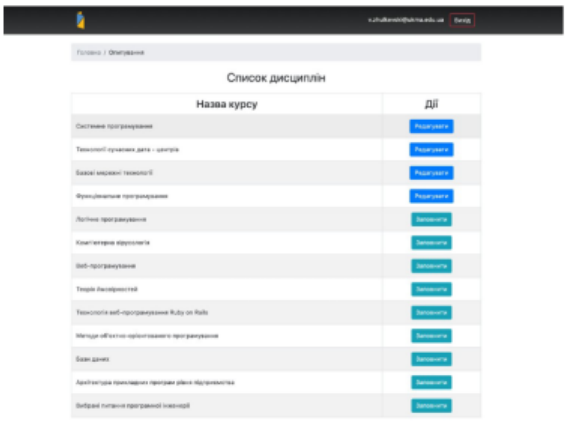


Схема бази даних

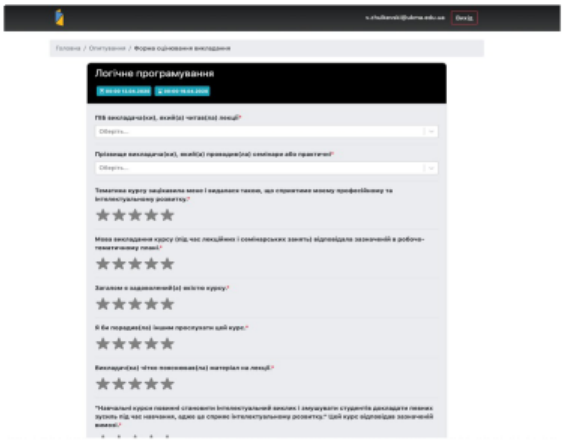


ЕКРАННІ ФОРМИ



Сторінка курсів студента

ЕКРАННІ ФОРМИ



Сторінка заповнення анкети

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Шевчук Ю.О. Використання WEB-орієнтованих технологій у навчальному процесі / Залива В.В., Шевчук Ю.О// Застосування програмного забезпечення в ІКТ: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 20.04.2023, ДУТ, м. Київ — К.: ДУТ, 2017. — С. 60.
2. Шевчук Ю.О Індивідуальний підхід до кожного студента через WEB-орієнтовані технології / Залива В.В., Шевчук Ю.О// Застосування програмного забезпечення в ІКТ: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 20.04.2023, ДУТ, м. Київ — К.: ДУТ, 2017. — С. 60.

Додаток Б

```
import React, { useEffect, Fragment } from
'react';import PropTypes from 'prop-types';
import { connect } from 'react-
redux'; import { Link } from 'react-
router-dom';
import Spinner from '../common/spinner/Spinner';
import { getProfileById } from '../../redux/actions/profileActions';
import ProfileTop from './ProfileTop';
import ProfileAbout from './ProfileAbout';
import ProfileExperience from
'./ProfileExperience';import ProfileEducation from
'./ProfileEducation'; import ProfileGithub from
'./ProfileGithub';
const
```

```
Profile = ({
  auth,
  getProfile
  ById,
  profile: { loading,
  profile },match,
}) => {
  useEffect(() => {
    getProfileById(match.params.id);
  },
  [getProfileByI
  d]);return (
    <Fragment>
      {' '}
      {profile === null || loading ? (
        <Spinner />
      ) : (
        <Fragment>
          {' '}
          <Link to="/profiles" className="btn btn-light">
            {' '}
            Back to Profiles{' '}
          </Link>{' '}
          {auth.isAuthenticated
            && auth.loading ===
            false &&
            auth.user._id === profile.user._id && (
              <Link to="/edit-profile" className="btn btn-dark">
```

```
      {' '}
      Edit Profile{' '}
    </Link>
  )}{' '}
<div className="profile-grid">
  {' '}
  <ProfileTop profile={profile} /> <ProfileAbout profile={profile} />{' '}
  <div className="profile-exp bg-white p-2">
    {' '}
    <h2 className="text-primary"> Experiance </h2>{' '}
    {profile.experience.length > 0 ? (
      <Fragment>
        {' '}
        {profile.experience.map((experience) => (
          <ProfileExperience
            key={experience._i
            d}
          />
        ))}{' '}
      </Fragment>
    ) : (
      <h4> No experience </h4>
    )}{' '}
  </div>{' '}
  <div className="profile-edu bg-white p-2">
    {' '}
    <h2 className="text-primary"> Education </h2>{' '}
```

```
            {profile.education.length > 0 ? (
              <Fragment>
                {' '}
                {profile.education.map((education) => (
                  <ProfileEducation
                    key={education._
                    id}
                    education={educa
                    tion}
                  />
                ))}{' '}
              </Fragment>
            ) : (
              <h4> No Education </h4>
            )}{' '}
          </div>{' '}
          {profile.githubusername && (
            <ProfileGithub username={profile.githubusername} />
          )}{' '}
        </Fragment>
      )}{' '}
    </Fragment>
  );
};
Profile.propTypes = {
  getProfileById:
  PropTypes.func.isRequired,profile:
  PropTypes.object.isRequired,
```

```
  auth: PropTypes.object.isRequired,
};
const mapStateToProps = (state)
  => ({profile: state.profile,
  auth: state.auth,
});
export default
  connect(mapStateToProps, {
  getProfileById,
})(Profile);
```

Додаток В

Програмний код компоненту «відео-чат» додатку:

```
import React from "react";
import { connect } from "react-
redux";import { isEmpty } from
"lodash";
import { Modal, ModalHeader, ModalBody, ModalFooter } from
"reactstrap";const ChatModal = ({
showMod
al,
messages,
handleMes
sage,
closeChat,
sendMessa
ge,
message,
```

```
}) => (
<Modal isOpen={showModal} className="video-chat-modal">
<ModalHeader>Chat Room</ModalHeader>
<ModalBody>
{isEmpty(messages) ? (
<p>No messages yet</p>
) : (
messages.map((item, index) => (
<div key={index} className="video-chat-modal__message">
<b>{item.sender}</b>: <span>{item.data}</span>
</div>
))
)}
<ModalFooter className="div-send-msg">
<input
placeholder="Message...
"value={message}
onChange={(e) => handleMessage(e)}
/>
<button onClick={() => sendMessage()}>Send</button>
<button onClick={() => closeChat()}>Close</button>
</ModalFooter>
</Modal>
);
export default ChatModal;
import React, { useState, useEffect, lazy } from
"react";import { isChrome } from
"../../utils/videoChatConst"; import { connect }
```

```
from "react-redux";
import "./VideoChat.scss";
const UsersList = lazy(() => import("./UsersList"));
const VideoChatContent = lazy(() =>
import("./VideoChatContent"));const VideoChat = ({ user, profile
}) => {
if (!isChrome())
<div className="video-not-support">
<h1>Sorry, yourbrowser doesnt support video chat</h1>
</
di
v
>;
re
tu
rn
(
<div className="video-chat-wrapper">
<UsersList />
<VideoChatContent user={user} profile={profile} />
);
};
const mapStateToProps = (state)
=> ({user: state.auth.user,
profile: state.profile,
});
export default connect(mapStateToProps, {})(VideoChat);
```

```
import React, { useEffect, useState } from "react";
import { connect } from "react-redux";
import { getAllUsersList } from
"../../redux/actions/profileActions";import { profilePhotoUrl }
from "../../utils/urls";
import "./UsersList.scss";
const UsersList = ({ auth: { user }, getAllUsersList, profile: { users } }) => {
useEffect(() => {
getAllUsersList();
}, []);
return (
<div className="users-list-wrapper">
<h3>Users list</h3>
<ul>
{users.map(
(userItem,
index) =>
user._id !== userItem._id && (
<li key={index} className="user-list__item">
<img
className="user-list__photo"
alt="photo"
/>
<p>{userItem.name}</p>
<button className="btn btn-light">invite</button>
</li>
)
```

```
)}
</ul>
</div>
);
};
const mapStateToProps = (state)
=> ({profile: state.profile,
auth: state.auth,
});
export default connect(mapStateToProps, { getAllUsersList })(UsersList);
@import "../../variables";
.users-list-
wrapper {
display: flex;
background:
$darkBlueV1;color:
#fff;
flex-direction:
column;width:
fit-content;
min-width:
300px;
padding:
0.4rem; h3 {
text-align:
center;font-
size: 1rem;
```

```
border-radius:
4px; margin:
0.5rem 0.2rem;
white-space:
nowrap; padding:
5px 0.8rem;
}
.user-list
_____i
tem {
display:
flex; flex-
wrap: wrap;
flex-direction:
row; align-
items: flex-end;
margin-bottom:
1rem;font-
weight: bold;
&:nth-last-
child(1) {
margin-bottom:
0;
}
.btn-light {
margin-top:
.5rem ;width:
```

```
100%;
}
.user-list
_____ph
oto {
overflow:
hidden;
border-
radius: 5%;
object-fit:
cover; width:
50px; height:
50px;
border: 2px solid #eee;
}
margin-left:
0.4rem;
margin-
bottom: 0;
height: min-
content;
}
}
```