МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ФІЗИКИ ТА МАТЕМАТИКИ
КАФЕДРА ІНФОРМАТИКИ, ПРОГРАМНОЇ ІНЖЕНЕРІЇ ТА ЕКОНОМІЧНОЇ КІБЕРНЕТИКИ

ВИКОРИСТАННЯ РЕАКТИВНОГО ПРОГРАМУВАННЯ ПРИ СТВОРЕННІ WEB UI

Кваліфікаційна робота (проєкт)

на здобуття ступеня вищої освіти "бакалавр"

Виконав: студент 4 курсу
Спеціальності 121 Інженерія програмного забезпечення
Освітньо-професійної програми «Інженерія програмного забезпечення» першого (бакалаврського) рівня освіти
Лукінов Геннадій Геннадійович
Керівники доктор педагогічних наук, доцент
Круглик Владислав Сергійович,
доктор педагогічних наук, професор
Співаковський Олександр Володимирович
Рецензент кандидат педагогічних наук, доцент Кузьмич Людмила Василівна

Херсон – 2020

ЗМІСТ

ВСТУП

Актуальність дослідження. На сьогодні майже всі програмні продукти з якими взаємодіє користувач мають інтерфейс. Більшість підприємств визнають, що відмінний інтерфейс користувача є життєво важливим для формування лояльності клієнтів.

Інтерфейси оточують нас всюди: телефони, автомобілі, вулиці та літаки, квиткові автомати і сайти – вони у всьому, на що людина може вплинути своїми діями

Перед розробниками веб-інтерфейсів у будь-якому проєкті поставлено завдання створення саме дружнього по відношенню до користувача інтерфейсу. Однак це не завжди таке просте завдання, як може здатися на перший погляд, і часом вимагає не малого досвіду проєктування. Головні вимоги тут - зручність, практичність і інтуїтивне та швидке керування.

Ще однією не менш важливою частиною розробки Web-UI є архітектура та інструменти для розробки, які визначають як ми будемо працювати з даними, надавати дані користувачеві, змінювати їх і тд. Зараз є велика різноманітність бібліотек для програмування інтерфейсів, але ми розглянемо розробку інтерфейсів саме на прикладі реактивного програмування, що являється дуже популярним підходом розробки у наш час.

Об’єкт дослідження: технології створення інтерфейсу користувача веб-додатків.

Предмет дослідження: технології створення інтерфейсу користувача з використанням парадигми реактивного програмування.

Мета дослідження: розробка інтерфейсу користувача на прикладі навчального веб-додатку “StudHub”.

У зв'язку з поставленою метою були визначені **завдання**:

1. Визначити основні принципи розробки Web-UI.
2. Проаналізувати реактивний підхід до розробки користувацьких інтерфейсів.
3. Розглянути бібліотеки для розробки UI, навести їх порівняльну характеристику.
4. Виконати аналіз вимог до компонентів React.
5. Спроєктувати серверну та клієнтську частини сайту.
6. Розробити та протестувати веб-додаток за допомогою бібліотеки React.

Структура роботи: Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1
ПОНЯТТЯ РЕАКТИВНОГО ПРОГРАМУВАННЯ

1.1. Програмування інтерфейсу користувача

Користувацький інтерфейс (UI) - це канал між взаємодією людини та комп'ютера - графічний макет програмного забезпечення / програми. Він включає елементи, які дозволяють користувачам взаємодіяти з машиною, програмним забезпеченням, додатком або обладнанням природним та інтуїтивним шляхом. Це простір, який показує, як користувачі можуть взаємодіяти з веб-сайтом, загальним дизайном та способом подання інформації. Інтерфейс користувача складається з кнопок, тексту, зображень, повзунків та всіх елементів, з якими користувачі взаємодіють. Відмінний інтерфейс користувача створює ефективний та дружній досвід для користувачів. [1]

Розробка великого та складного графічного інтерфейсу користувача і управління складними бізнес-даними та процесами є досить складним завданням. Без належного проєктування системи код UI та бізнес-логіка можуть стати заплутаними, що приведе до проблеми організації коду та неочікуваних помилок.

Універсальних критеріїв надійної архітектури веб-додатків [2]:

- Ефективність;
- Гнучкість;
- Розширюваність;
- Дотримання принципу відкрито-закритого типу;
- Масштабованість процесу розробки;
- Легке тестування;
- Багаторазовість;
- Добре структурований і зрозумілий код.

Чітка організація коду та принципи розробки є важливими протягом усього життєвого циклу системи.

Враховуючи ці вимоги, загальний підхід у розробці програмного забезпечення слід розпочинати з визначення архітектури системи. [2]

Шаблони зазвичай застосовуються в рамках системного підходу до розробки системи, де архітектура системи спочатку визначається з урахуванням системних вимог. За допомогою проєктування системи, за принципом розділення проблем.

У наш час є велика кількість бібліотек для розробки інтерфейсу користувача, які побудовані на декларативній парадигмі програмування. Величезна кількість операцій для взаємодії даних з інтерфейсом, їх відображенням та оновленням, абстраговано, що робить розроблювану систему дуже гнучкою та легко підтримуваною. Що задовольняє критерії архітектури користувацьких інтерфейсів та веб-додатків в цілому. [3]

Декларативне програмування.

Декларативне програмування - це парадигма програмування побудови структури та елементів комп'ютерних програм, що виражає логіку програми, не описуючи її потік управління. [4]

Декларативне програмування робить код [5]:

- Легко читаємим, оскільки приховуються деталі низького рівня.
- Простіше описуваним, оскільки код сильно абстрагується, описується рішення замість процедури.

1.2. Реактивне програмування

Реактивне програмування - це декларативна парадигма програмування, що стосується потоків даних та поширення змін. За допомогою цієї парадигми можна з легкістю виражати статичні

(наприклад, масиви) або динамічні (наприклад, випромінювачі подій) потоки даних, а також повідомляти, що існує висновок про залежність у відповідній моделі виконання, що полегшує автоматичне поширення змінених даних. [6]

Реактивна система реагує своєчасно на зміни, якщо це можливо. Чутливі системи фокусуються на забезпеченні швидкого та послідовного реагування, встановлення надійних меж, щоб вони забезпечували стабільну якість обслуговування. Ця послідовна поведінка в свою чергу спрощує обробку помилок, формує довіру користувачів та заохочує їх до подальшої взаємодії.

Реактивна система залишається чутливою до несподіваних подій. Це стосується не лише критично важливих систем - будь-яка система, яка не є стійкою, не буде реагувати після відмови. Стійкість досягається шляхом стримування, ізоляції та делегування. Поломки містяться всередині кожного компонента, ізолюючи компоненти один від одного і тим самим забезпечуючи стійкість системи, частини системи можуть вийти з ладу та відновитись без шкоди для системи в цілому. Відновлення кожного компонента делеговано іншому (зовнішньому) компоненту, а висока доступність забезпечується шляхом реплікації, де це необхідно. Клієнт компонента не обтяжений поводженням з його помилками.

Реактивна система - еластична. Система залишається чутливою при різному навантаженні. Реактивні системи можуть реагувати на зміни швидкості введення, збільшуючи або зменшуючи ресурси, виділені для обслуговування цих входів. Можна розшаровувати або копіювати компоненти та розподіляти вхідні дані між ними. Реактивні системи підтримують прогнози, а також алгоритми реактивного масштабування, забезпечуючи відповідні заходи в реальному часі. Вони досягають еластичності економічно на товарних і програмних платформах.

Керовані повідомлення: Реактивні системи покладаються на асинхронну передачу повідомлень для встановлення межі між компонентами, що забезпечує вільне з'єднання, ізоляцію та прозорість розташування. Цей кордон також забезпечує засоби для делегування відмов як повідомлень. Використання явної передачі повідомлень дозволяє керувати навантаженням, еластичністю та контролем потоку, формуючи та контролюючи черги повідомлень у системі та застосовуючи зворотний тиск, коли це необхідно. Розташування прозорого обміну повідомленнями як засобу комунікації дозволяє керувати непрацездатністю однакових конструкцій та семантики в кластері або в межах одного хоста. Неблокуючий зв’язок дозволяє одержувачам лише споживати ресурси під час активних дій, що призводить до менших витрат системи.

Потік даних – центральний параметр парадигми реактивного програмування, це послідовність подій, що відсортованих у часі. Потік має три основних типи подій: значення, помилку, сигнал “завершення”. [7]

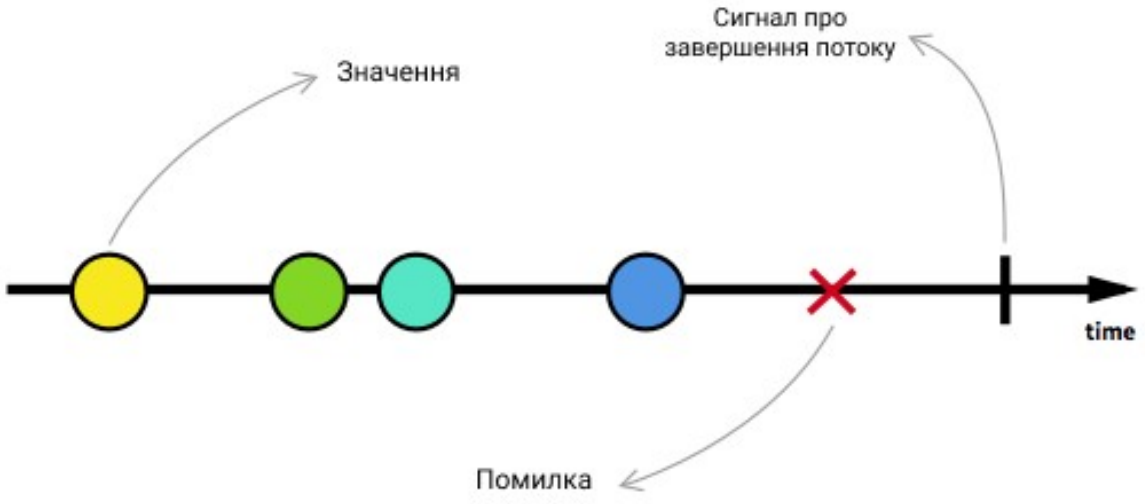

Рис. 1.1 Ілюстрація роботи потоку

Ці події події фіксуються лише асинхронно, визначаючи функцію, яка буде виконуватися, для кожного типу подій існує своя функція. Але

обов’язковим являється лише функція для обробки значень. "Прослуховування" потоку називається підпискою. Функції, які ми визначаємо, - це спостерігачі. Потік - це предмет який спостерігається. Це поведінковий шаблон проектування “Спостерігач”. Це надає можливість об’єктам постійно реагувати на зміни.

Найбільші в світі системи покладаються на архітектури, що базуються на цих властивостях і обслуговують потреби мільярдів людей щодня.

1.3. Архітектура Flux

Flux - це архітектура для створення шарів даних у додатках JavaScript. Він був розроблений у компанії Facebook разом із бібліотекою програмування інтерфейсів React. Flux робить акцент на створенні явних та зрозумілих шляхів оновлення даних програми, що робить простеження змін під час розробки простішими та спрощує відстеження та виправлення помилок. [8]

Щоб найкраще описати Flux, порівняємо його з однією з провідних архітектурних форм на базі клієнта: MVC. У програмі MVC на стороні клієнта взаємодія з користувачем запускає код у контролері. Контролер знає, як координувати зміни в одній або декількох моделях, викликаючи методи на моделях. Коли моделі змінюються, вони сповіщають про один або більше переглядів, які, в свою чергу, читають нові дані з моделей і відповідно оновлюються, щоб користувач міг бачити ці нові дані.

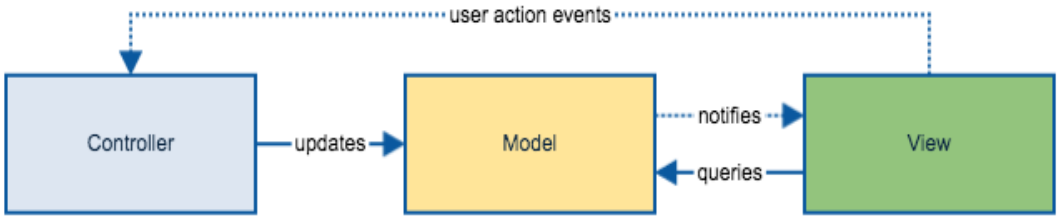

Рис. 1.2 Принцип роботи MVC

У міру зростання програми MVC і додавання контролерів, моделей залежності стають складнішими.

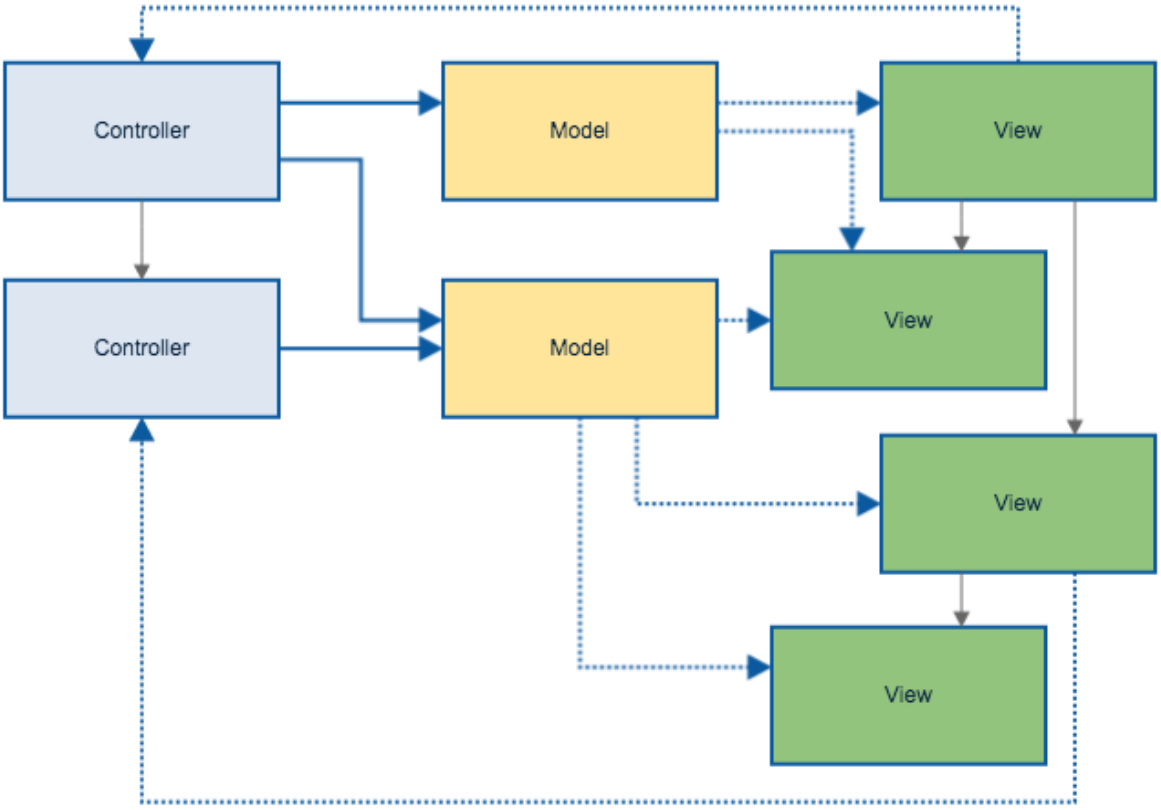

Рис. 1.3 Приклад біль складного MVC додатку

З додаванням лише трьох представлень, одного контролера та однієї моделі, графік залежності вже важче простежити. Коли користувач взаємодіє з інтерфейсом користувача, виконуються кілька шляхів розгалуження коду, а проблеми налагодження в стані програми стають складністю з'ясувати, який модуль (або модулі) в одному (або декількох) цих потенційних кодових шляхах містять помилку. У гірших випадках взаємодія з користувачем викликає оновлення, що в свою чергу викликає додаткові оновлення, що призводить до схильних до помилок та важких для налагодження каскадних ефектів.

Flux уникає цієї конструкції на користь однобічного потоку даних. Усі взаємодії користувачів у представленні викликають творця дії, що спричиняє випадок події дії від одиночного dispatcher.

Dispatcher - це єдиний об'єкт, який транслює дії / події у всі зареєстровані сховища. Сховища потрібно зареєструвати, коли програма запускається.

Коли буде здійснено дію, вона буде передана її всім зареєстрованим сховищам.

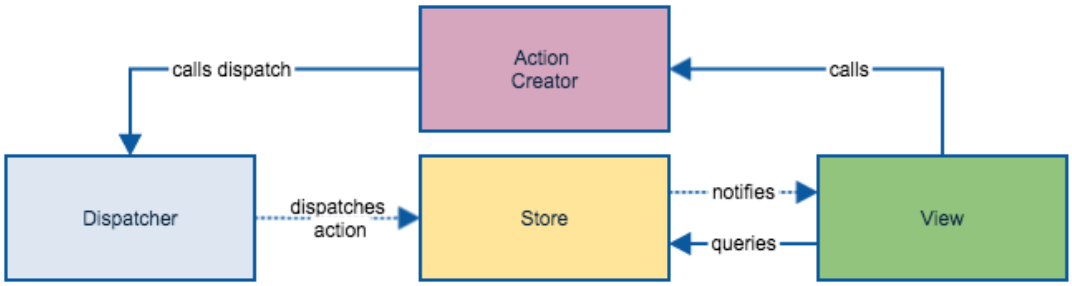

Рис. 1.4 Принцип роботи Flux

Потік суттєво не змінюється для додаткових Stores чи представлень. Диспетчер просто надсилає кожну дію до всіх Stores програми. Зауважте, що він не містить знань про те, як насправді оновити Store - самі Stores містять цю логіку.

Коли Store оновлюється, він видає подію зміни. У багатьох програмах React спеціальні представлення (відомі іноді як "перегляди контролерів") відповідають за спостереження за цією подією зміни, зчитуванням нових даних Store і передачею цих даних через властивості до дочірніх компонентів. У програмах React для події зміни Store часто викликається повторне перемальовування компонентів верхнього рівня. Це повністю дозволяє уникнути складних помилок та проблем із роботою, які можуть виникнути внаслідок спроби стежити за конкретними змінами властивостей на моделях та лише змінювати частини поглядів. [30]

Основні властивості.

Архітектура Flux має кілька ключових властивостей, які роблять його унікальним і гарантують, що всі властивості зосереджені на тому, щоб зробити потік даних явним і зрозумілим і збільшити локальність помилок (так що не доведеться шукати багато кодові шляхи, щоб знайти

неправильну логіку). Є багато потоків та потокоподібних реалізацій; ці властивості для найбільш важливі для архітектури Flux.

Синхронність.

Дія відправки та їх обробники всередині Stores є синхронними. Усі асинхронні операції повинні запускати дію, що повідомляє системі про результат операції. Хоча творці дій можуть робити запит до асинхронних API, обробники дій у Stores в ідеалі цього не робитимуть. Це правило робить потік інформації в додатку надзвичайно явним; налагодження помилок у стані програми просто передбачає з'ясування того, яка дія спричинила некоректний стан.

Інверсія управління.

Оскільки сховища оновлюються всередині себе у відповідь на дії (а не оновлюються ззовні контролером чи подібним модулем), жоден інший фрагмент системи не повинен знати, як змінити стан програми. Вся логіка оновлення стану міститься в самому сховищі. Оскільки сховища оновлюються лише у відповідь на дії, і лише синхронно, тестування сховищ стає просто справою приведення їх у початковий стан, надсилання їх дії та тестування, щоб перевірити, чи знаходяться вони у правильному кінцевому стані.

Семантичні дії.

Оскільки сховищам потрібно оновлюватись у відповідь на дії, дії, як правило, мають семантичний опис. Наприклад, щоб позначити потік як прочитаний, ви можете відправити дію з типом MARK_THREAD_READ. Дія (і компонент, що генерує дію) не знає, як виконати оновлення, але описує, що вона хоче відбутися.

Через цю властивість вам рідко доводиться змінювати типи дій, лише те, як магазини реагують на них. Поки у вашій програмі є поняття "нитка" і у вас є кнопка або інша взаємодія, яка повинна позначати потік

як прочитаний, тип дії MARK_THREAD_READ є семантично дійсним. [9]

Висновки до розділу 1

У першому розділі було розглянуто теоретичну основу, яка використовується у кваліфікаційній роботі. Дано визначення інтерфейсу користувача, розглянуто поняття реактивного програмування та його особливості і переваги використання в розробці інтерфейсу користувача. Розглянуто архітектурний підхід Flux, розроблений для проєктування інтерфейсів користувача, які поєднуються з реактивним програмуванням.

Можна зробити висновок, що допомогою **парадигми реактивного програмування** можна з легкістю виражати потоки даних, а також автоматично повідомляти та поширювати змінених даних. Реактивна система своєчасно реагує на зміни, якщо це можливо. Ця послідовна поведінка в свою чергу спрощує обробку помилок, формує довіру користувачів та заохочує їх до подальшої взаємодії.

Flux – архітектура створення шарів даних у веб-додатках. Робить простеження змін під час розробки простішими та спрощує відстеження та виправлення помилок.

Основні властивості архітектури:

- Синхронність
- Інверсія управління
- Семантичні дії

РОЗДІЛ 2
ОГЛЯД ІСНУЮЧИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБ-ДОДАТІВ

2.1. Огляд Node.js

Node.js – це асинхронне багатоплатформове оточення JavaScript з відкритим вихідним кодом, спроєктоване для побудови масштабних мережевих додатків.

Платформа Node.js розроблена на основі двигуна V8, що використовується в браузері Chrome. В основному використовується для розробки веб-сервісів, але має набагато більше коло застосування.

Розглянемо основні особливості платформи:

1. Швидкість – JavaScript-код може бути в декілька разів швидший, ніж код компілюємих мов програмування. Це досягається завдяки неблокованій архітектурі платформи.
2. Мова програмування – Node.js інтерпретує код написаний на мові JS. Це дає змогу розробникам клієнтської частини веб-додатків почати розробляти серверну частину без потреби вивчати нову мову програмування. Як в браузері так і на сервері використовуються одинакові концепції мови.
3. Асинхронність – в традиційних мовах програмування всі інструкції є блокованими за замовчуванням. JS навпаки значно спрощує написання асинхронного коду з використанням функцій зворотного виклику. Асинхронні механізми дозволяють єдиному серверу одночасно обробляти тисячі запитів автоматично, без необхідності програмісту налаштовувати потоки та організовувати виконання паралельного коду. [10]

4. Бібліотеки – завдяки простоті використання вбудованого пакетного менеджеру NPM (Node.js Package Manager) кількість корисних бібліотек для розробки з кожним днем зростає. На даний момент каталог npm нараховує більше 1 мільйона модулів. [11]

2.2. Огляд та порівняння бібліотек для розробки інтерфейсів

JavaScript-бібліотеки для розробки інтерфейсів останнім часом стрімко розвиваються. Це обумовлено великим попитом веб-додатків, продукти великих компаній і невеликих стартапів так чи інакше розвиваються у web. Тому це сприяє розробці бібліотек та фреймворків, їх підтримці від передових компаній.

Найпопулярнішими реактивними бібліотеками в даний момент являються: React, Vue та Angular.

1. React.

Декларативна та гнучка бібліотека для розробки складних користувацьких інтерфейсів. Дозволяє створювати інтерфейс за допомогою ізольованих частин коду, які називають “компонентами”. Бібліотека розроблюється та підтримується компанією Facebook з 2013 року. [12]

Основні особливості React:

- Легкий у вивченні та розробці.
- Використовує HTML-подібну мову розмітки для шаблонізації;
- Дуже швидкий, використовує технологію віртуального DOM та багато інших оптимізацій;
- Оновлення до нових версій проходить безболісно та швидко, автоматизується більшість процесів;
- Дуже добре описана документація, багато прикладів та курсів;

- Має велику кількість готових компонентів та бібліотек для будь-яких завдань;
- Знання React дозволяють розробляти додатки для Android та IOS на React Native;
- Відкритий вихідний код. [13]

Компанії, які використовують React: Taskade, Reddit, Cloudflare, Tesla, GitHub, Facebook, Instagram та ін. [14]

2. Vue.

Реактивна бібліотека з відкритим вихідним кодом для розробки користувацьких інтерфейсів. Перша версія була випущена Еваном Ю у 2013 році. На даний момент підтримується Еваном Ю та ком'юніті [15]

Основні особливості Vue:

- Бібліотека легка у вивченні;
- Детальна документація, велика кількість відео-курсів та прикладів;
- Невеликий розмір бібліотеки;
- Відкритий вихідний код;
- Простота та схожість з React з точки зору архітектури дає можливість легко перейти на Vue.

Компанії, які використовують Vue: Facebook, Netflix, Adobe, Xiaomi, GitLab, Behance та ін. [16;28]

3. Angular.

Платформа для розробки веб-додатків на мові JavaScript. Основною відмінністю від бібліотек описаних вище є шаблон проєктування MVC, ціль фреймворку – спростити розробку веб-додатків на основі саме цього шаблону. Розроблюється компанією Google з 2009 року. [17]

Основні особливості Angular:

- Велика кількість вбудованих рішень (маршрутизація, запити, тестування), не потребує встановлення додаткових модулів для більшості рішень;
- Модульна структура, що дозволяє розділити додаток на багаторазові компоненти;
- Має вбудований набір UI компонентів, побудованих на основі Material дизайну; [18]

У таблиці 2.1. наведено порівняння самих популярних бібліотек для розробки інтерфейсу. [27]

Таблиця 2.1.

Порівняння бібліотек для розробки інтерфейсу

Бібліотека	**Angular**	**Vue**	**React**
Розробник	Google	Evan You	Facebook
Розмір бібліотеки	246.8 KB	41.28 KB	91.5 KB
Рік релізу	2010	2014	2013
Кількість зірок у репозиторії GitHub	162 тис.	59 тис.	147 тис.
Час завантаження	1.78	1.00	1.10
Швидкість взаємодії с DOM	1.30	1.31	1.32
Остання версія	1.7.9 Nov 19, 2019	2.6.11 Dec 13, 2019	16.13.1 19 Mar, 2020
Використовуються	Forbes Santander Bank BMW	FontAwesom e 9GAG Alibaba	Facebook Instagram Netflix

2.3. Огляд Redux

Redux – бібліотека для фронт-енд розробки з відкритим вихідним кодом для керування станом всередині додатку. Побудований на архітектурі Flux та розроблений Дэном Абрамовим та Эндрю Кларком у 2014 році, та активно підтримується. [29]

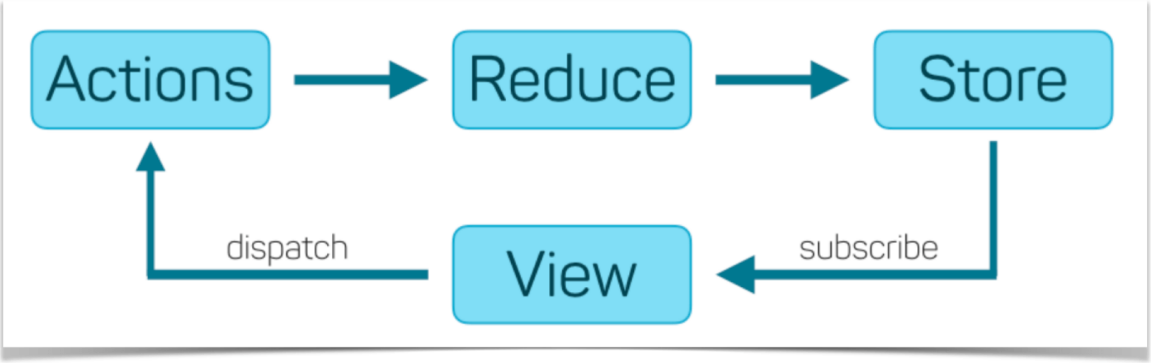

Рис. 2.1 Архітектура Redux

Store – об'єкт, який зберігає дерево стану, має необхідні методи для зміни стану.

Action - це об'єкт який приймає вид дії та стан, який необхідно змінити.

Reduce - це чиста функція, яка приймає попередній стан і Action і повертає новий стан. Саме завдяки цьому вдається налаштувати гнучку взаємодію клієнту та сервера і організувати зрозумілий потік даних в веб-додатку. [19]

Основними особливостями є:

- невеликий розмір бібліотеки (2kB разом із залежностями)
- простота у використанні та велика кількість документації, навчальних матеріалів, курсів
- постійна підтримка з боку розробників та спільноти
- можливість використання разом з більшістю бібліотек для фронт-енд розробки (React, Angular)

Оскільки вимоги до веб-додатків постійно стають більш складнішими, доводиться керувати більшою кількістю даних всередині

додатку. Це може бути як відповіді від сервера, локальні, кешовані дані. Стан інтерфейсу користувача також ускладнюється. Правильно керувати станом, що постійно змінюється дуже складно та з масштабуванням проєкту це може викликати непередбачені помилки. Тож Redux – це бібліотека для керування станом, яка намагається зробити зміну стану передбачуваною, встановлює певні обмеження, щодо того, як і коли можуть відбуватись оновлення.

Ці обмеження відображені в трьох принципах Redux:

1. Єдине джерело істини.

Стан всього додатку зберігається в дереві об’єктів в єдиному сховищі. Це полегшує створення універсальних додатків, оскільки стан, який запитується з серверу до клієнту не потребує зусиль. Єдине дерево стану також полегшує дебаг та тестування програми.

2. Стан доступний лише для зчитування.

Єдиний спосіб змінити стан – викликати дію, об’єкт, що описує те, що сталось. Це гарантує, що зміна стану буде строго контролюватись і не буде викликати додаткових помилок. Оскільки всі зміни централізовані і відбуваються в строгому порядку. Дії (Actions) – прості об’єкти, їх можна серіалізувати, зберігати і в подальшому відтворювати для налагодження тестування.

3. Зміни вносяться за допомогою чистих функцій.

Щоб описати як дерево стану змінюється – використовуються чисті редуктори (Reducer). Це чиста функція, яка приймає попередній стан і дію у якості параметрів та повертає новий стан з необхідними змінами. [20]

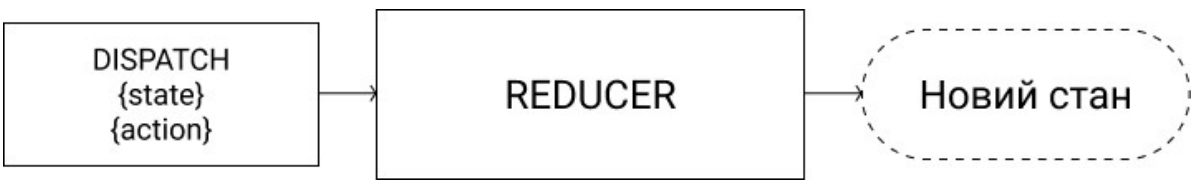

Рис. 2.2. Демонстрація роботи Reducer

Використання з React.

З самого початку необхідно підкреслити, що Redux не має відношення до React. Redux можливо використовувати з великою кількістю бібліотек: React, Angular, Ember, jQuery.

При цьому Redux особливо добре працює з React, оскільки він дозволяє описати інтерфейс як функцію від стану веб-додатку, а Redux займається оновленням стану.[21]

Висновки до розділу 2

У другому розділі були розглянуті бібліотеки та фреймворки для розробки інтерфейсу користувача, наведена їх порівняльна характеристика.

Vue – бібліотека, розроблена Еваном Ю. з відкритим вихідним кодом для розробки користувацьких інтерфейсів. Легка в освоєнні, добре документована.

Angular – платформа для розробки веб-додатків, розробляється компанією Google. Основною відмінністю від заявлених бібліотек є підтримка розробки MVW, MVVM, MVC додатків.

React – бібліотека, розроблена компанією Facebook для розробки складних користувацьких інтерфейсів. Дозволяє створювати інтерфейс за допомогою багаторазових компонентів.

Redux – бібліотека для керування станом, яка робить зміну стану додатку передбачуваною, встановлює певні обмеження, щодо того, як і коли можуть відбуватись оновлення.

РОЗДІЛ 3
ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБ-ДОДАТКУ

3.1. Постановка задач

1. Проєктування клієнт-серверної частини веб-додатку
2. Розробка серверної частини (авторизації, API)
3. Розробка інтерфейсу користувача
4. Налагодження взаємодії клієнт-сервер
5. Тестування

Ціллю кваліфікаційної роботи являється розробка веб-додатка для додаткового навчання за допомогою реактивного програмування.

StudHub - веб-додаток, Основною ідеєю якого є просте і зрозуміле поширення курсів. На відміну від комплексних рішень таких як Moodle, ISpring, Blackboard даний веб-додаток більш вузько направлене рішення, що дозволяє швидко розгорнути свій сервіс для створення курсів. Перевагою являється використання сучасних технологій для розробки, що сприятиме легкому масштабуванню та підтримці сервісу.

Загальна структура веб-додатка.

Розроблюваний веб-додаток складається з трьох основних компонентів, представлених на рис. 3.1.

Клієнтська частина веб-додатку - це інтерфейс для взаємодії користувача з сайтом, клієнтська частина також формує запити для взаємодії з сервером. В проєкті кваліфікаційної роботи було використано бібліотеку React для програмування інтерфейсу.

Серверна частина - це процес на сервері для обробки запитів користувача та взаємодії з даними. В проєкті кваліфікаційної роботи було використано бібліотеку Express для програмування серверної частини.

База даних (БД, або система управління базами даних, СУБД) - програмне забезпечення на сервері, що займається зберіганням даних і їх видачею в потрібний момент. База даних розташовується на сервері. Серверна частина веб-додатки звертається до бази даних, витягуючи дані, які необхідні для формування сторінки, запитаної користувачем. В проєкті кваліфікаційної роботи було використано MongoDB.

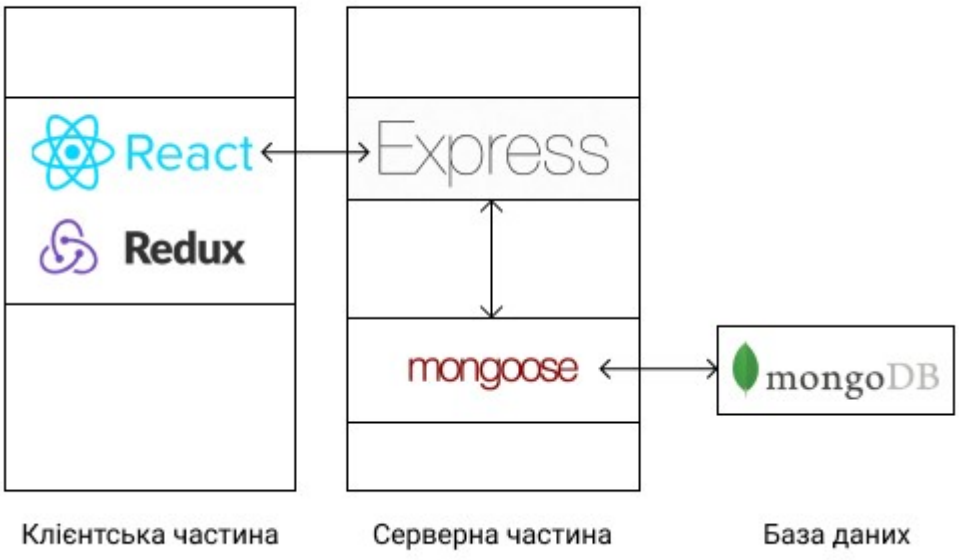

Рис. 3.1 Архітектура веб-додатка

3.2. Проєктування веб-додатку

Проєктування полягає в розробці набору моделей (діаграм) на мові UML. Для опису функціональних вимог щодо розроблюваного ПЗ створюються діаграми варіантів використання (usecases).

Дана діаграма відображає таблиці бази даних і відносин між ними.

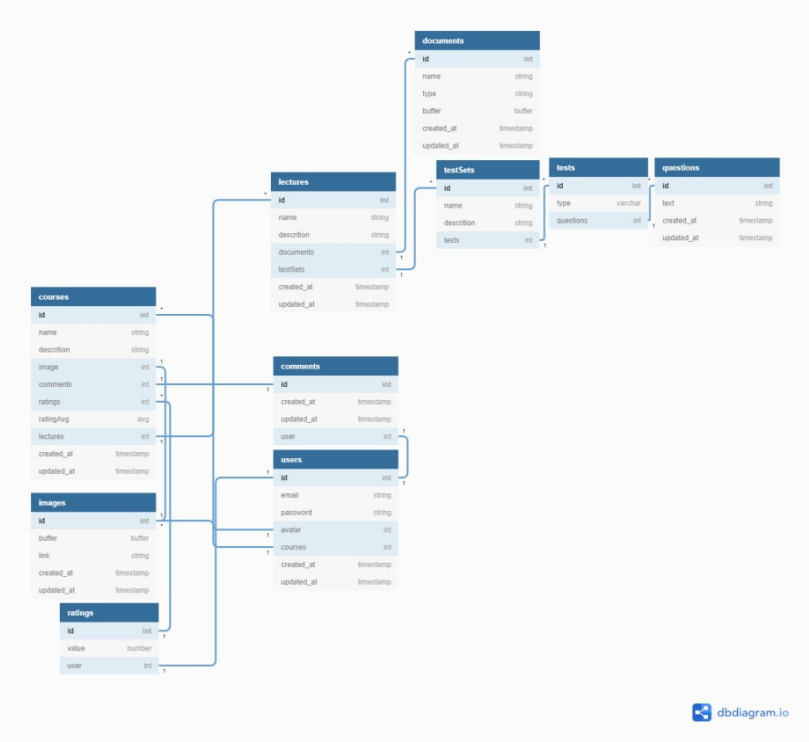

Рис 3.2 Діаграма бази даних

У системі є 2 ролі: користувач та адміністратор. Вони виконують конкретні функції і взаємодіють з веб-додатком.

Була досліджена предметна область, і на основі вимог побудовані діаграми варіантів використання. Виявлено функції і властивості розроблюваної системи.

Таблиця 3.1.

Документування об'єктів діаграми варіантів використання

Об’єкт	**Документація**
Адміністратор	Відповідає за розміщення курсів. Редагування лекцій, додавання тестів.
Користувач	Взаємодіє з створеними курсами.
Варіанті використання	
Автентифікація	Перевірка облікового запису

	користувача (логіну та паролю) при вході в систему. Якщо перевірка пройшла успішно - здійснюється вхід.
Створення / редагування курсів Створення / редагування лекцій Створення / редагування тестів Створення / редагування файлів	Додавання відповідних записів до БД через адмін-панель.
Залишити коментар	Можливість створювати/видаляти свої коментарі
Керування коментарями	Можливість керувати коментарями користувачів
Оцінка курсу	Можливість залишити оцінку до курсу (0-5)
Записатися на курс	Можливість записатись на курс, отримувати повідомлення про нові лекції. Обраний курс з’явиться в профілі користувача.

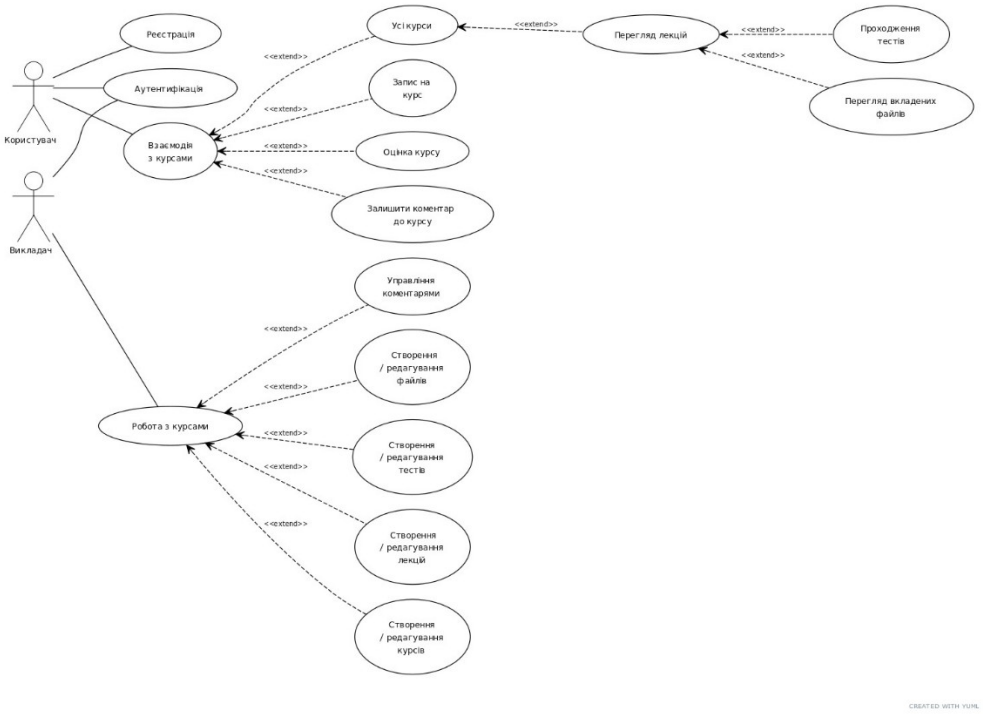

Рис 3.3 Діаграма варіантів використання

3.3. Розробка серверної частини

У якості бази даних для веб-додатка було використано MongoDB. Це NoSQL база даних, де кожен запис є документом, що складається з пар ключових значень, схожих на об'єкти JSON. MongoDB є гнучким і дозволяє його користувачам створювати схеми, бази даних, таблиці тощо. Документи, які можна ідентифікувати за допомогою первинного ключа, складають основну одиницю MongoDB. [22]

Була віддана перевага MongoDB адже він:

- Швидкий - це база даних, орієнтована на документи, легко індексувати документи. Тому швидша реакція.
- Масштабованість - великі дані можна обробляти, розділивши їх на кілька машин.
- Використання JavaScript - MongoDB використовує JavaScript, що є найбільшою перевагою для нашого проєкту.
- Схеми - будь-який тип даних у окремому документі.

- Дані, що зберігаються у вигляді JSON -
- Об'єкти, учасники об'єкта, масиви, значення та рядки
- Синтаксис JSON дуже простий у використанні.
- JSON має широкий спектр сумісності браузера. [23]

Для взаємодії з базою даних була використана бібліотека Mongoose. Всі колекції MongoDB у Mongoose описуються за допомогою схем. Кожна схема відображає і визначає форму документів у цій колекції.

Приклад схеми документа Курсу:

```
const {Schema, model, Types} = require('mongoose')

const schema = new Schema({
    name: {type: String, required: true, unique: true},
    description: {type: String, required: true},
    image: [{type: Types.ObjectId, ref: ‘Image}],
    comments: [{type: Types.ObjectId, ref: 'Comment'}],
    ratings: [{type: Types.ObjectId, ref: 'Rating'}],
    ratingAvg: {type: Number},
    lectures: [{type: Types.ObjectId, ref: 'Lecture'}]
},{ timestamps: true })

module.exports = model('Course', schema)
```

Рис 3.4 Приклад опису схеми документу MongoDB

Схеми можна використовувати повторно, і вони можуть містити також кілька дочірніх схем. У наведеному вище прикладі значення властивості name, description, ratingAvg - це простий тип значення. В той час як comments, ratings, lectures - відображають зв'язок один-до-багатьох. Далі на основі схеми створюється модель, їй призначається ім’я та експортується для подальшого використання.

Для обробки запитів веб-додатка було використано веб-фреймворк express, який забезпечує надійний набір функцій для веб і мобільних

додатків. Дозволяє гнучко обробляти http запити, та створювати мінімалістичні та швидкі API. Зпроєктована діаграма послідовності, яка відображає взаємодію користувача, інтерфейсу та API. [24]

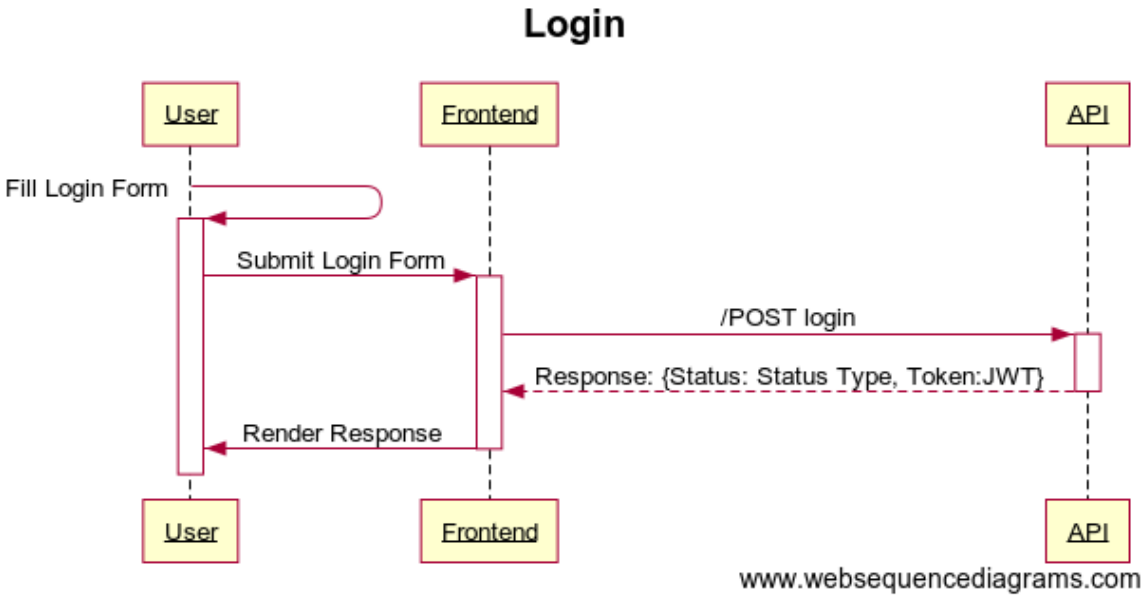

Рис 3.5 Діаграма послідовності для сценарію авторизації

```
app.use('/api/courses', require('./routes/course.routes'))

// GET /api/courses/
router.get('/', async (req, res) => {
    try {
        let { limit } = req.body
        const courses = await Course.find().limit(limit[0], limit[1])
        res.status(200).json({"courses": courses})
    } catch (e) {
        console.log(e)
        res.status(500).json({'message': "Error!"})
    }
})
```

Рис 3.6 Приклад функції, яка обробляє GET запит користувача для отримання списку курсів в заданому проміжку.

Деякі параметри обробника запиту:

- Router (об'єкт маршрутизатора) - це окремий екземпляр програмного забезпечення та маршрутів. Об’єкт здатний

виконувати лише функції маршрутизації. Кожна програма Express має вбудований маршрутизатор.

- req - об'єкт запиту.
- res - об'єкт відповіді.
- res.status(...),json(...) - функції об’єкта для відправлення коду відповіді та даних в форматі JSON.
- Course - модель колекції в базі даних, описаний раніше. Дозволяє взаємодіяти з документами в базі даних.

3.4. Розробка інтерфейсу користувача на прикладі веб-додатка “StudHub”

Для програмування інтерфейсу веб-додатку було використано бібліотеку React. Для керування станом всередині веб-додатка - бібліотека Redux, яка побудована на архітектурі Flux.

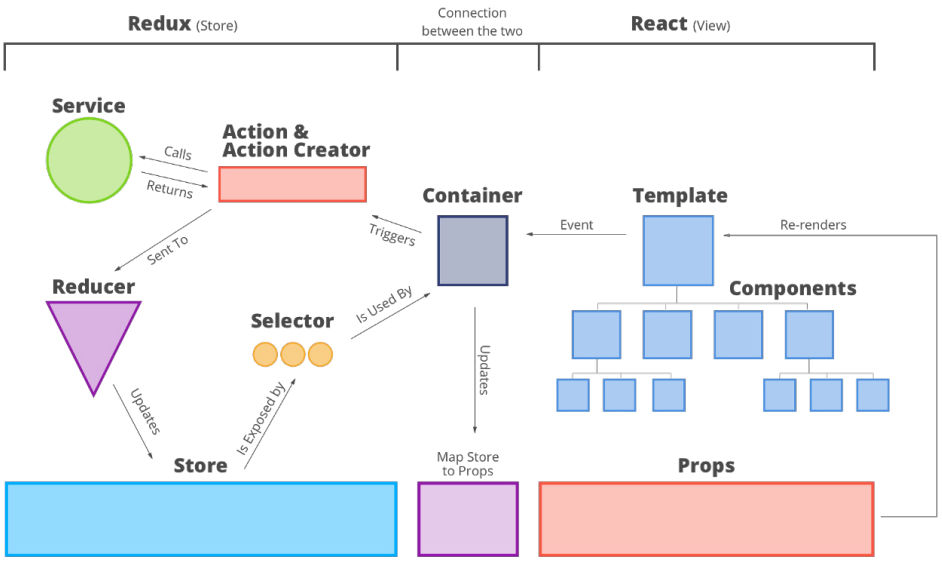

Рис 3.7 Загальна архітектура для React + Redux

Компоненти React - це невеликі багаторазові фрагменти коду, які повертають елемент React для відображення на сторінку. Найпростіша

версія компонента React - це звичайна функція JavaScript, яка повертає елемент React.

Container - об’єкт, який дозволяє взаємодіяти Redux Store з компонентами React і реагувати на зміни в Store.

Service - сервіс, який відповідає за взаємодію клієнт-сервер. Формуються запити до API.

Selector - визначає які данні використовуються компонентом і у разі зміни Store - викликають перерендер компонента.

Action - це об’єкт який приймає вид дії та стан, який необхідно змінити. [25]

Використовуючи інформацію попередніх розділів була сформована структура веб-додатка, показана на рис. 3.3.

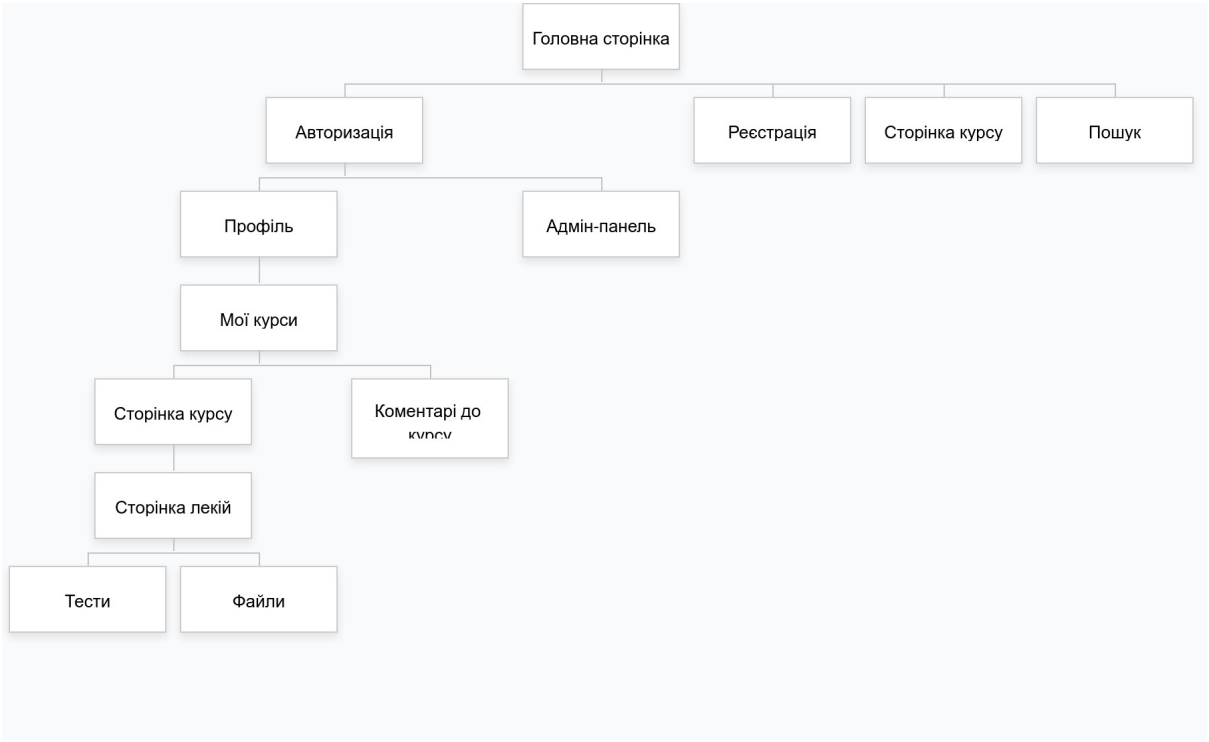

Рис. 3.8 Структура веб-додатка

Розробка інтерфейсу користувача.

Були спроєктовані та розроблені макети інтерфейсу користувача.

Компоненти в React дозволяють розділити користувальницький інтерфейс на незалежні багаторазові частини та використовувати кожен фрагмент окремо. Кожен компонент можна використовувати

багаторазово, що дозволяє розробляти дуже гнучкі додатки, які легко розширювати, додавати нові модулі та функціонал.

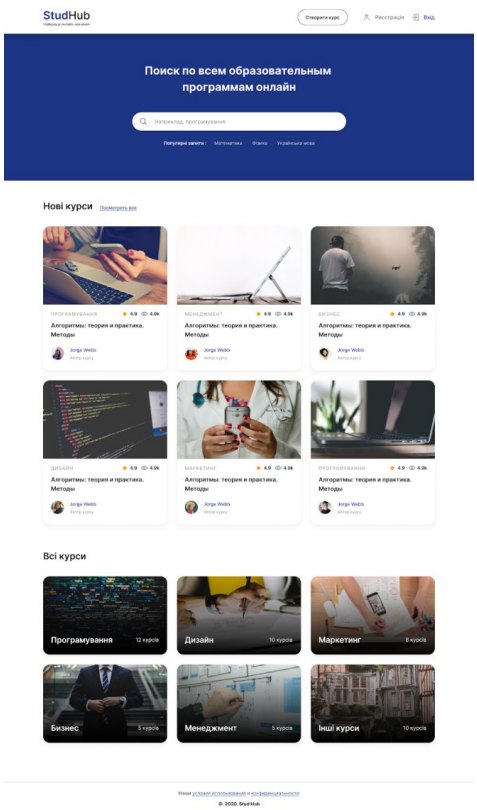

Рис. 3.9 Макет головної сторінки веб-додатку

При розробці клієнтської частини сайта було використано функціональні та класові компоненти.

Приклад компоненту головної сторінки веб-додатка наведено у додатку А.

Даний компонент приймає властивості (properties) з батьківського компонента і відображає їх за шаблоном JSX, який описано всередині оператора return.

Було розроблено компоненти користувацького інтерфейсу, кожен з яких можна багаторазово використовувати при розробці. Такий підхід при розробці інтерфейсів дає можливість з легкістю змінювати наявні компоненти і використовувати їх повторно, зміна компоненту призведе до його оновлення всюди де він імпортований, це перетворює розробку в безперервний процес поліпшення продукту.

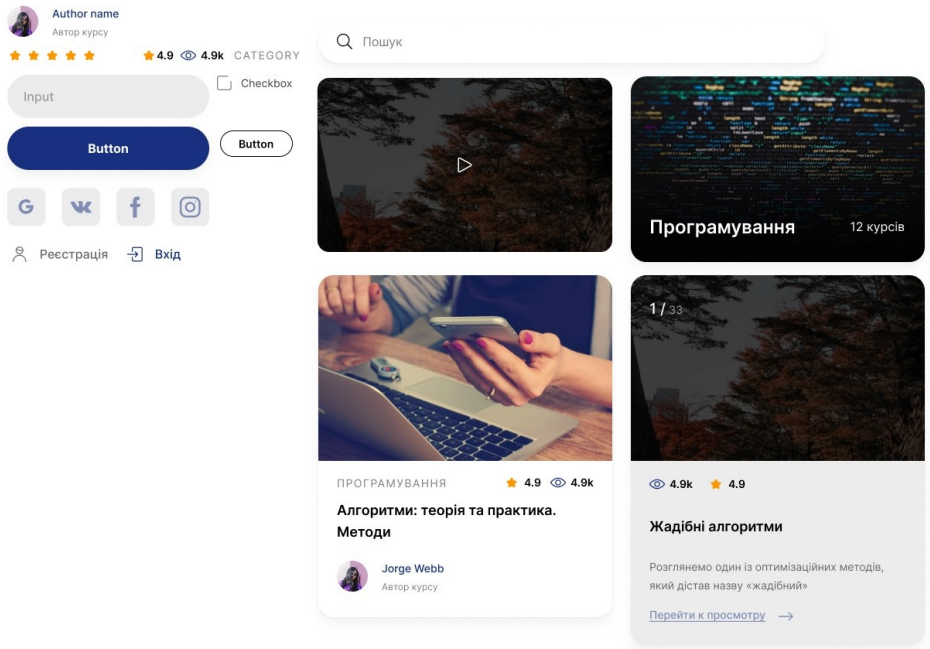

Рис. 3.10 Приклад розроблених React-компонентів

Коротко розглянемо розроблені компоненти:

1. NavBar - компонент, який відповідає за відтворення навігаційної панелі веб-додатка.
2. Search - компонент, який відповідає за відтворення рядку пошуку по курсам веб-додатка. Може працювати у зв'язку з компонентом Filter.
3. Filter - компонент, який відповідає за фільтрування курсів за необхідними критеріями. Це можуть бути: категорія, рейтинг, дата створення або оновлення, автор, популярність.
4. CourseList - компонент, який відображає список курсів.
5. CourseLabel - компонент, який відображає блок курсу на головній сторінці, або сторінці пошуку. Включає в себе: назву курсу, категорію, ім’я автора, рейтинг, дату створення або оновлення.
6. Rating - компонент, який відображає рейтинг певного курсу. Для авторизованих користувачів, які записані на даний курс

є можливість оцінити його від 1 до 5. Дані зберігаються лише один раз.

7. CommentForm - компонент для авторизованих користувачів. Дозволяє залишити коментар до курсу, якщо користувач записаний на курс - користувач також може обрати рейтинг курсу, який буде відображено з коментарем.
8. CommentList - компонент, який відображає всі передані до нього коментарі.
9. CommentItem - компонент, який відображає блок коментаря. Включає в себе: ім'я автора, текст коментаря, дату створення та рейтинг.

Приклад базової сторінки веб-додатка:

```
import CourseList from '../components/courses/CourseList'
import Search from '../components/Search'
export class Main extends Component {
    render() {
        return (
            <div>
                <Search/>
                <h2>Нові курси</h2>
                <CourseList courses={this.props.courses} />
                <Footer/>
            </div>
        )
    }
}
```

Рис. 3.11 Приклад базової сторінки веб-додатка

В даному прикладі було створено класовий компонент, який наслідує клас Component із бібліотеки React. Це дозволяє використовувати методи цього класу, в часності метод render для його відтворення та повертає його сутність до корневого компонента. Цей компонент складається з більш атомарних компонентів, приклади яких

було наведено вище. Це дає змогу краще розуміти як працює веб-додаток і в подальшому простіше супроводжувати розробку.

3.5. Налаштування зв'язку клієнт-сервер

Після розробки компонентів необхідно налаштувати взаємодію внутрішнього стану веб-додатка та сервера.

Для керування станом було використано бібліотеку Redux, яку було розглянуто у другому розділі кваліфікаційної роботи. Для функціонування Redux потрібні три основні складові частини: Actions, Reducers та Stores.

Для кожного етапу асинхронного запиту до серверу, було об'явлено 3 типи дій та еспортовано, для подальшого імпорту в інші файли проєкту:

export const FETCH_COURSES_REQUEST = 'FETCH_COURSES_REQUEST'

export const FETCH_COURSES_SUCCESS = 'FETCH_COURSES_SUCCESS'

export const FETCH_COURSES_FAILURE = 'FETCH_COURSES_FAILURE'

```
const fetchCoursesRequest = courses => {
    return {
        type: FETCH_COURSES_REQUEST
    }
}
```

Рис. 3.12 Приклад Actions для асинхронного запиту курсів

Action Creator - це функції, які створюють дії. В нашому випадку вони використовуються, для виклику побічних дій після відповіді на запит до серверу.

```
export const fetchCourses = () => {
    return (dispatch) => {
        dispatch(fetchCoursesRequest())
        axios.get('/courses').then((response) => {
            const courses = response.data
            dispatch(fetchCoursesSuccess(courses))
        })
        .catch((error) => {
            const { message } = error
            dispatch(fetchCoursesFailure(message))
        })
    }
}
```

Рис. 3.13 Приклад Action Creator для для асинхронного запиту списку курсів

Деякі параметри функції fetchCourses:

- axios - гнучкий http клієнт для генерації запитів;
- fetchCoursesRequest - функція, яка повертає дію з типом FETCH_COURSES_REQUEST, що сигналізує про те, що було створено запит до сервера;
- fetchCoursesSuccess - функція, яка повертає дію з типом FETCH_COURSES_SUCCESS, що сигналізує про успішний запит, та передає дані курсів, що передаються до редуктора і в подальшому зберігаються в Store;
- fetchCoursesFailure - функція, яка повертає дію з типом FETCH_COURSES_FAILURE, що сигналізує про помилку та передає дані про помилку до редуктора;

Було розроблено Reducer для роботи з запитами до сервера. Як приклад Reducer для роботи з даними курсів, Actions яких було описано вище.

```
const reducer = (state = initialState, action) => {
    switch (action.type) {
        case FETCH_COURSES_SUCCESS: {
            return {
                ...state,
                courses: action.payload,
                loading: false
            }
        }
        default: {
            return state
        }
    }
}
```

Рис. 3.14 Reducer для роботи з даними курсів

В прикладі описано реакцію лише на Action, який викликається після успішного запиту курсів. Функція компонує весь стан із змінами і передає їх в Store.

- state - деструктурований об’єкт загального стану;
- action.payload - дані, які було передано до Reducer;
- loading - змінна, що характеризує чи відбувається запит

Використання Redux стану всередині компонентів досягається завдяки функції connect (міститься в бібліотеці react-redux) , що підключає компонент React до store Redux.

Це надає підключеному компоненту чистини необхідних даних і функції, які він може використовувати для відправки дій в редуктор для оновлення даних. [26]

Це не змінює клас компонентів, переданий йому; натомість він повертає новий, підключений клас компонентів, який обгортає компонент.

const mapStateToProps = ({coursesReducer}) => {

const {courses,error,loading} = coursesReducer;

```
    return ({courses,error,loading})
}
const mapDispatchToProps = (dispatch) => {
    return {fetchCourses: () => dispatch(fetchCourses())}
}
export default connect(mapStateToProps, mapDispatchToProps)(Main)
```

Приклад підключення компоненту головної сторінки веб-додатку до Redux store.

Якщо вказана функція mapStateToProps, підключений компонент підписується на оновлення store Redux. Це означає, що щоразу, коли store буде оновлений, mapStateToProps буде викликано і UI, який залежить від цього стану буде оновлено.

Якщо вказана функція mapDispatchToProps, очікується, що ваші функція поверне об'єкт. Кожне поле об'єкта має бути функцією, виклик якої, відправить дію до редуктора.

Таким чином досягається реактивність інтерфейсу, його залежність від стану веб-додатку.

Висновки до розділу 3

У третьому розділі були поставлені задачі для розробки веб-додатку “StudHub”. Здійснене проєктування сайту. Описані UML діаграми взаємодії користувача з сайтом, в залежності від його ролі, взаємодія об’єктів, обмін даних всередині додатку та зв’язок із сервером. Описані схеми бази даних для всіх сутностей.

Описана розробка серверної частини сайту:

- розробка структури бази даних MongoDB на основі спроєктованих діаграм

- розробка сервісу реєстрації та авторизації на NodeJs
- розробка REST-API

Розробка інтерфейсу користувача:

- розробка компонентів інтерфейсу користувача
- створення Store, Actions, Reducers для зберігання та взаємодії з даними

ВИСНОВКИ

Метою кваліфікаційної роботи є розробка інтерфейсу користувача веб-додатку “StudHub”.

У ході виконання роботи були виконані наступні завдання:

- Визначено основні принципи розробки Web-UI.
- Проаналізовано реактивний підхід до розробки інтерфейсу користувача.
- Розглянуто бібліотеки для розробки Web-UI, наведено їх порівняльну характеристику.
- Виконано аналіз вимог до компонентів React.
- Спроєктовано серверну та клієнтську частини сайту.
- Розроблено веб-додаток за допомогою бібліотеки React.

Для реалізації поставленої задачі було досліджено сучасний підхід до розробки веб-додатків, був проведений детальний аналіз технологій для розробки інтерфейсу користувача. Було використано сучасні технології для розробки, що забезпечує стабільність системи, яку легко масштабувати та підтримувати.

Для розробки серверної частини було обрано: платформу Node.js, бібліотеку Express для створення маршрутів та обробки запитів, MongoDB як систему управління базами даних. Разом ці технології забезпечують стабільну роботу системи, легкий доступ та керування даними.

В результаті аналізу для розробки інтерфейсу було обрано бібліотеку React. Адже React: ефективно взаємодіє з UI, стабільно підтримується розробниками, має детальну документацією, легко тестувати та ппідтримувати.

Для керування станом всередині веб-додатка було обрано бібліотеку Redux, яка особливо добре працює з React, оскільки він дозволяє

описати інтерфейс як функцію від стану веб-додатку, а Redux займається його оновленням. Це робить систему гнучкою, її легко підтримувати.

Результатом роботи є готовий веб-додаток, який дозволяє легко поширювати користувацькі курси.

Перевагами є:

- сучасні технології використані при розробці;
- реактивність, яка лежить в основі системи;
- асинхронний зв’язок із сервером;
- легко масштабувати та підтримувати;

У сукупності це робить веб-додаток ефективним, зручним у використанні, має високий рівень користувацького досвіду.

Розроблений веб-додаток задовольняє всі вимоги, сформовані на етапі постановки завдання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Difference Between UX And UI [Електронний ресурс] – Режим доступу до ресурсу: https://www.mageplaza.com/blog/what-is-user-interface.html#what-is-the-user-interface-ui.

2. Web Application Architecture. [Електронний ресурс] – Режим доступу до ресурсу: https://litslink.com/blog/web-application-architecture.

3. Front-end Developer Handbook. Cody Lindley. [Електронний ресурс] – Режим доступу до ресурсу: https://frontendmasters.com/books/front-end-handbook/2019/#1.

4. Declarative programming. [Електронний ресурс] – Режим доступу до ресурсу: https://www.ionos.com/digitalguide/websites/web-development/declarative-programming/.

5. Declarative Programming & React. [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/itsjzt/declarative-programming--react-3bh2.

6. Реактивное программирование на реальных примерах: подробное введение. [Електронний ресурс] – Режим доступу до ресурсу: https://tproger.ru/translations/reactive-programming/.

7. The introduction to Reactive Programming you've been missing. [Електронний ресурс] – Режим доступу до ресурсу: https://gist.github.com/staltz/868e7e9bc2a7b8c1f754.

8. Facebook: MVC Does Not Scale, Use Flux Instead. [Електронний ресурс] – Режим доступу до ресурсу: https://www.infoq.com/news/2014/05/facebook-mvc-flux/.

9. Flux Overview. [Електронний ресурс] – Режим доступу до ресурсу: http://fluxxor.com/what-is-flux.html.

10. Node.js – Опис [Електронний ресурс]—Режим доступу: https://www.tutorialspoint.com/nodejs/nodejs_tutorial.pdf.

11. Руководство по Node.js, часть 1: общие сведения и начало работы. [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/ruvds/blog/422893/.

12. React Overview. [Електронний ресурс] – Режим доступу до ресурсу: https://reactjs.org/docs/getting-started.html.

13. Advantages of Using React.js. [Електронний ресурс] – Режим доступу до ресурсу: https://da-14.com/blog/its-high-time-reactjs-ten-reasons-give-it-try.

14. Companies/Brands Using ReactJS Development [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/front-end-weekly/top-25-companies-brands-using-reactjs-development-8be87b32cec2.

15. Vue.js - JS Framework [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Vue.js.

16. Companies That Have Trusted Vue.js – Examples of Applications [Електронний ресурс] – Режим доступу до ресурсу: https://www.netguru.com/blog/13-top-companies-that-have-trusted-vue.js-examples-of-applications.

17. AngularJS MVC Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://intellipaat.com/blog/tutorial/angularjs-tutorial/mvc-angularjs/.

18. Benefits of Angular You Should Know if You Want to Build a Digital Product. [Електронний ресурс] – Режим доступу до ресурсу:

https://www.netguru.com/blog/7-benefits-of-angular-you-should-know-if-you-want-to-build-a-digital-product.

19. A quick guide to Redux for beginners [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/a-quick-guide-to-redux-for-beginners-971d18c0509b/.

20. JavaScript by Example [Електронний ресурс] – Режим доступу до ресурсу: https://subscription.packtpub.com/book/web_development/9781788293969/7/07lvl1sec38/what-is-redux.

21. Redux Usage with React. [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/basics/usage-with-react.

22. Онлайн-посібник з MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/nosql/mongodb/.

23. What Is MongoDB? [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/what-is-mongodb.

24. Guiding Principles of REST [Електронний ресурс] – Режим доступу до ресурсу: https://restfulapi.net/.

25. Developing modern offline apps with ReactJS, Redux [Електронний ресурс] – Режим доступу до ресурсу: https://blog.codecentric.de/en/2017/12/developing-modern-offline-apps-reactjs-redux-electron-part-3-reactjs-redux-basics/.

26. React-Redux documentation. [Електронний ресурс] – Режим доступу до ресурсу: https://react-redux.js.org/api/connect.

27. “React vs Angular vs Vue.js — What to choose in 2020?” [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@TechMagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d.

28. What is Vue.js. [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/v2/guide/#What-is-Vue-js.

29. Redux An Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.smashingmagazine.com/2016/06/an-introduction-to-redux/.

30. Flux. In-Depth Overview. [Електронний ресурс] – Режим доступу до ресурсу: https://facebook.github.io/flux/docs/in-depth-overview.

ДОДАТКИ

Додаток А

Компонент головної сторінки:

```
import React, { Component } from 'react'
import { connect } from 'react-redux'
import CourseList from '../components/courses/CourseList'
import Search from '../components/Search'
import { fetchCourses } from '../reducers'
import { Link } from 'react-router-dom'
export class Main extends Component {
  componentDidMount() {
    if(!this.props.courses.length) {
      this.props.fetchCourses()
    }
  }
  render() {
    return (
      <div>
        <Search/>
        <div className="container">
        <div className="heading__container aic">
          <h2>Нові курси</h2>
          <div className="heading__col">
            <Link className="anc">Переглянути всі</Link>
          </div>
        </div>
        {this.props.loading ? (
          <div>Loading...</div>
        ) : (
          this.props.error ? (
            <div>{this.props.error}</div>
          ) : (
            <CourseList courses={this.props.courses} />
```

```
            )
          )
          }

          <div className="heading__container">
            <h2>Всі курси</h2>
            <div className="heading__col">
              {this.state.category.map(({id, name}) => <li key={id}
className="heading__category__item">{name}</li>)}
            </div>
          </div>

          {this.props.loading ? (
            <div>Loading...</div>
          ) : (
            this.props.error ? (
              <div>{this.props.error}</div>
            ) : (
              <CourseList courses={this.props.courses} />
            )
          )
          }

        </div></div>
      )
    }
}
const mapStateToProps = ({coursesReducer}) => {
  const {courses,error,loading} = coursesReducer;
  return ({courses,error,loading})
}
const mapDispatchToProps = (dispatch) => {
  return {fetchCourses: () => dispatch(fetchCourses())}
}
export default connect(mapStateToProps, mapDispatchToProps)(Main)
```

Додаток Б

КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ ХЕРСОНСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ

Я, Лукінов Геннадій Геннадійович,
учасник(ця) освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

– дотримуватися:

- вимог законодавства України та внутрішніх нормативних документів університету, зокрема Статуту Університету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:

- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційному репозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;

– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;

– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;

– своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивного іміджу;

– не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;

– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;

– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні й політичні переконання;

– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;

– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науково-дослідницькі завдання;

– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;

– не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;

– не підроблювати документи;

– не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів і співробітників;

– не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійснення впливу на зміну отриманої академічної оцінки;

– не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;

– не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власності інших студентів та/або працівників;

– не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних з діяльністю університету;

– не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;

– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальності й до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

23.04.20		Геннадій Лукінов
(дата)	(підпис)	(ім'я, прізвище)