Державний торговельно-економічний університет

Кафедра інженерії програмногозабезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:
«Онлайн-платформа дистанційного навчання»

Студента 2м курсу, 2 групи, спеціальності 121 «Інженерія програмного забезпечення» освітньої програми «Інженерія програмного забезпечення» _____ Рудича Максима Олеговича
підпис студента

Науковий керівник кандидат економічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ Палагута Катерина Олексіївна
підпис керівника

Гарант освітньої програми кандидат педагогічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ Котенко Наталія Олексіївна
підпис гаранта

КИЇВ – 2023

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Освітня програма 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«13» грудня 2022 р.

Завдання
на випускну кваліфікаційну роботу студентові
Рудича Максима Олеговича
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи «Онлайн-платформа дистанційного навчання»

Затверджена наказом ректора від «06» грудня 2022 р. № 3285

2. Строк здачі студентом закінченої роботи 27 листопада 2023

3. Цільова установка та вихідні дані до роботи

Мета роботи є розроблення системи контролю успішності студентів на основі веб-додатку для онлайн-платформи дистанційного навчання.

Об'єкт дослідження є інформаційні системи та їх використання в різного роду структурах, зокрема в системах дистанційного навчання.

Предмет дослідження є система контролю успішності студентів в контексті онлайн-платформи дистанційного навчання.

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поняття веб-додатку

1.2. Основні принципи веб-розробки

1.3. Поняття контролю навчального процесу

1.4. Огляд існуючих рішень

1.5. Висновки до розділу 1

РОЗДІЛ 2 АНАЛІЗ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Огляд мови програмування

2.2. Огляд середовища розробки

2.3. Огляд додаткового інструментарію

2.4. Висновки до розділу 2

РОЗДІЛ 3 ПРОЕКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1. Аналіз варіантів використання

3.2. Проектування внутрішньої будови

3.3. Розробка графічного інтерфейсу системи

3.4. Тестування

3.5. Висновок до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ТЕХНІЧНЕ ЗАВДАННЯ

ТЕСТУВАННЯ ДОДАТКА

ДОДАТКИ

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	*07.11.2022*	*07.11.2022*
2.	*Розробка та затвердження завдання на роботу магістра (стац/заоч)*	*13.12.2022*	*13.12.2022*
3.	*Вступ та перелік літературних джерел*	*24.02.2023*	*24.02.2023*
4.	*Розробка технічного завдання*	*15.03.2023*	*15.03.2023*
5.	*Розділ 1. Теоретичні основи соціальної підтримки*	*10.04.2023*	*10.04.2023*
6.	*Розділ 2 Аналіз інструментальних засобів реалізації*	*24.05.2023*	*24.05.2023*
7.	*Розділ 3. Проектування і розробка програмного продукту*	*06.09.2023*	*06.09.2023*
8.	*Розробка програми та методики тестування*	*18.10.2023*	*18.10.2023*
9.	*Написання наукової статті*	*17.05.2023*	*17.05.2023*
10.	*Керівництво користувача*	*25.10.2023*	*25.10.2023*
11.	*Висновки та пропозиції*	*01.11.2023*	*01.11.2023*
12.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*	*06.11.2023*	*06.11.2023*
13.	*Підготовка автореферату та презентації доповіді*	*06.11.2023*	*06.11.2023*
14.	*Попередній захист випускної кваліфікаційної роботи*	*20.11.2023 – 24.11.2023*	*20.11.2023 – 24.11.2023*
15.	*Здача зброшурованої випускної кваліфікаційної роботи*	*27.11.2023*	*27.11.2023*
16.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	*29.11.2023*	*29.11.2023*
17.	*Підготовка до публічного захисту випускної кваліфікаційної роботи*	*05.12.2023-06.12.2023*	*05.12.2023-06.12.2023*

7. Дата видачі завдання «13» грудня 2022 р.

8. Науковий керівник випускної кваліфікаційної роботи Палагута К.О.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Котенко Н.О.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Рудич М. О.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускної кваліфікаційної роботи

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента Рудич М. О.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми Котенко Н.О.

(прізвище, ініціали, підпис)

Завідувач кафедри Криворучко О. В.

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Кваліфікаційна робота магістра присвячена розробці програмного продукту для підтримки процесу дистанційного навчання з використанням мови програмування Java та веб-фреймворку Spring. У цій роботі розглядаються основні теоретичні аспекти дистанційного навчання та розробки веб-додатків, їх основні властивості і призначення. Окрім цього в ході роботи було обрано інструментальні засоби розробки, проведено глибоке проектування системи, починаючи від функціональних вимог, закінчуючи проектуванням внутрішньої будови, проведено розробку фронтенд і бекенд частин веб-додатку.

Результатом роботи є веб-додаток, який забезпечує функціонал платформи дистанційного навчання з функціоналом контролю успішності.

Ключові слова: веб-додаток, дистанційне навчання, електронне навчання, web, spring, java.

Випускна кваліфікаційна робота на тему «Онлайн-платформа дистанційного навчання» містить 68 сторінок, 13 рисунків і 1 таблиці. Перелік використаних джерел налічує 32 найменування.

ABSTRACT

The master's thesis is dedicated to the development of a software product to support the process of distance learning using the Java programming language and the Spring web framework. This work explores the fundamental theoretical aspects of distance learning and web application development, including their key properties and purposes. Additionally, the research involves the selection of development tools, in-depth system design from functional requirements to internal structure, and the development of both frontend and backend components of the web application.

The outcome of this work is a web application that provides the functionality of a distance learning platform with performance monitoring features.

Keywords: web application, distance learning, e-learning, web, Spring, Java.

The graduation thesis on the topic "Online Distance Learning Platform" consists of 68 pages, 13 figures, and 1 table. The bibliography includes references to 32 sources.

ЗМІСТ

ВСТУП

В сучасному світі кожного дня кількість інформації, яку необхідно зберігати росте в геометричній прогресії , набуваючи величезних розмірів. В певний момент старі варіанти зберігання інформації, такі як паперові носії, або навіть усна форма, віджили своє і більше не могли задовольняти потреби сучасного соціуму.

Саме в цей момент і з’явилося систем автоматизації, які до сьогоднішнього дня допомагають людям у структуруванні та зберіганні інформації різного роду, взаємодії між ролями користувачів, тощо.

На сьогоднішній день кожне підприємство та установа мають свої внутрішні клієнти, які допомагають зберігати і структурувати інформацію про роботи даної установи, автоматизувати деякі процеси, тощо.

Виходячи з актуальності явища систем автоматизації, об’єктом дослідження є інформаційні системи і їх використання в різного роду структурах.

Предмет дослідження — система контролю успішності студентів.

Мета роботи — розроблення системи контролю успішності студентів.

Для досягнення поставленої мети слід виконати наступні завдання:

Для виконання поставленої мети слід виконати наступні завдання:

- Провести огляд поняття веб-додатку
- Провести аналіз методів та засобів розробки веб-сайтів
- Провести огляд поняття контролю навчання
- Провести огляд існуючих рішень
- Обрати мову програмування

- Обрати додаткові інструменти
- Обрати середовище розробки
- Провести аналіз варіантів використання
- Провести проектування внутрішньої будови
- Розробити графічний інтерфейс користувача
- Провести тестування систем

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поняття веб-додатку

Веб-додаток — це тип програмного забезпечення, яке функціонує на веб-сервері, на відміну від традиційних додатків, які запускаються на операційній системі пристрою локально. Користувачі доступаються до веб-додатків через веб-браузер із підключенням до Інтернету. Вони розробляються за моделлю клієнт-сервер, де "клієнт" (користувач) взаємодіє з додатком через веб-браузер, а обробка даних відбувається на сервері.

Типові приклади веб-додатків включають веб-пошту, онлайн-магазини, онлайн-банкінг та онлайн-аукціони. Вони забезпечують функціональні можливості, схожі на програми для настільних комп'ютерів або мобільні додатки.

Важливим аспектом сучасних веб-додатків є HTML5, який дозволяє створювати програми, які можна завантажувати як веб-сторінки, але водночас зберігати дані локально та працювати в автономному режимі.

Односторінкові веб-додатки (Single Page Applications, SPA) відрізняються тим, що вони не потребують перезавантаження цілої сторінки для оновлення чи зміни контенту. Це досягається завдяки тому, що різні компоненти веб-сторінки можуть бути замінені або оновлені динамічно без необхідності перезавантажувати всю сторінку. Такий підхід сприяє підвищенню швидкості та зручності користування веб-додатками.

Односторінкові фреймворки в сучасному веб-розробленні є ефективним інструментом для створення динамічних веб-додатків, особливо для мобільних платформ. Вони дозволяють оптимізувати пропускну здатність та зменшити необхідність завантаження зовнішніх файлів, оскільки більша частина вмісту завантажується один раз при першому запиті до веб-сторінки.

Традиційно, у моделях клієнт-сервер, навантаження було розподілено між серверною частиною та клієнтськими додатками, встановленими на комп'ютерах користувачів. Така модель вимагала окремого встановлення та оновлення клієнтських програм на кожному пристрої, що було часомістким і неефективним, особливо при зміні серверного коду.

На відміну від цього, веб-додатки використовують веб-документи, створені у стандартних форматах, як-от HTML та JavaScript. Ці документи підтримуються багатьма веб-браузерами, що дає можливість використовувати веб-додатки на різних пристроях та операційних системах без необхідності встановлення додаткового програмного забезпечення. Клієнтська частина веб-додатків завантажується прямо у веб-браузері під час відвідування відповідної сторінки, і може оновлюватися з кожним новим відвідуванням. Це значно спрощує процес розробки та підтримки веб-додатків, а також робить їх більш доступними для широкої аудиторії користувачів.

На зорі Інтернету кожна веб-сторінка була статичним документом, який доставлявся клієнту, але інтерактивність досягалася за допомогою веб-форм, через які користувачі могли відправляти дані. Проте будь-які значні зміни на сторінці вимагали повернення до сервера для оновлення всієї сторінки.

У 1995 році Netscape представила JavaScript, мову скриптів на стороні клієнта, що дозволила додавати динамічні елементи до інтерфейсу користувача, які виконувалися локально, без необхідності відправлення даних на сервер. Це дало змогу виконувати такі завдання, як перевірка введених даних або зміна частин веб-сторінки без її перезавантаження.

У 1996 році Macromedia випустила Flash, інструмент для вбудовування векторної анімації у веб-сторінки, що додатково розширило можливості для створення інтерактивного вмісту на стороні клієнта.

У 1999 році концепція веб-додатку була розвинута в специфікації Java Servlets версії 2.2. До цього моменту як JavaScript, так і XML вже існували, але технологія Ajax ще не була розроблена, і лише нещодавно був представлений об'єкт XMLHttpRequest у Internet Explorer 5 як об'єкт ActiveX.

У 2005 році термін Ajax був введений в обіг, і такі додатки, як Gmail, почали робити свої клієнтські сторони більш інтерактивними. Веб-сценарії тепер могли зв'язуватися з сервером для зберігання або отримання даних без необхідності перезавантаження всієї сторінки.

У 2007 році Стів Джобс оголосив, що веб-додатки на основі HTML5 та AJAX будуть основною платформою для розробки додатків на iPhone, виключаючи потребу в SDK і інтегруючи додатки безпосередньо через браузер Safari. Це згодом було змінено з запуском App Store, що забезпечило контрольоване середовище для розробки додатків.

У 2014 році HTML5 було завершено як стандарт, який забезпечує графічні та мультимедійні можливості без необхідності додаткових клієнтських плагінів. HTML5 також збагатив семантичний зміст веб-документів і став ключовою частиною веб-розробки, особливо з використанням WebGL для розширеної 3D-графіки.

У 2016 році на Google I/O були представлені прогресивні веб-додатки (PWA), які відкрили нову еру у веб-розробці. PWA поєднують переваги традиційних веб-сторінок із функціональністю нативних додатків. Цей підхід отримав підтримку як від Google, так і від Microsoft.

На стороні інтерфейсу користувача веб-додатки використовують JavaScript, CSS, а також інші технології (раніше Java, Flash, Silverlight) для створення інтерактивних елементів, таких як малювання на екрані, відтворення звуку і управління клавіатурою та мишкою. Зусилля були спрямовані на створення більш знайомого інтерфейсу, схожого на операційну

систему. Розробники також використовують клієнт-серверні скрипти для додання функціональності, особливо для створення інтерактивного досвіду без необхідності перезавантаження сторінки. Сучасні технології також дозволяють координувати роботу клієнтських скриптів з серверними технологіями, такими як ASP.NET, J2EE, Perl/Plack і PHP.

Ajax, веб-розробка, яка використовує комбінацію різних технологій, справді створює більш інтерактивний веб-досвід. Веб-додатки часто розробляються з використанням багаторівневої архітектури, де кожен рівень має свою специфічну роль.

Трирівнева модель є однією з найпоширеніших структур для веб-додатків. Вона включає:

1. Рівень презентації: це веб-браузер, який відображає користувацький інтерфейс.
2. Рівень логіки програми: тут використовуються різні технології динамічного веб-контенту, такі як ASP, CGI, JSP/Java, Node.js, PHP, Python або Ruby on Rails.
3. Рівень зберігання: це база даних, де зберігається інформація.

Веб-браузер надсилає запити до середнього рівня, який обробляє ці запити, виконує операції з базою даних і генерує відповідний користувацький інтерфейс.

Для складніших веб-додатків може бути корисним використання багаторівневої (n-рівневої) архітектури, де бізнес-логіка розбивається на більш дрібні компоненти. Це дозволяє, наприклад, відокремити рівень доступу до даних від інших рівнів, що спрощує доступ до даних і забезпечує гнучкість у виборі баз даних і технологій. Наприклад, замість прямого виконання SQL-запитів до бази даних можна використовувати функції вищого рівня, такі як "list_clients()". Це робить систему більш адаптивною та легшою у підтримці.

Деякі люди розглядають веб-додаток як дворівневу архітектуру. Це може бути «розумний» клієнт, який виконує всю роботу та запитує «німий»

сервер, або «німий» клієнт, який покладається на «розумний» сервер.[4] Клієнт буде обробляти рівень презентації, сервер матиме базу даних (рівень сховища), а бізнес-логіка (рівень програми) буде на одному з них або на обох.[4] Хоча це збільшує масштабованість програм і відокремлює дисплей і базу даних, це все одно не дозволяє справжньої спеціалізації шарів, тому більшість програм переросте цю модель.[4]

1.2. Основні принципи веб-розробки

Веб-сайт — це колекція взаємопов'язаних веб-сторінок, які містять текст, зображення, відео, аудіо тощо, і які ідентифікуються спільним доменним ім'ям і розміщені на щонайменше одному веб-сервері. Видатні приклади включають веб-сайти, такі як Wikipedia, Google та Amazon.

Загалом усі веб-сайти, доступні для загальної публіки, формують Всесвітню павутину. Є також приватні веб-сайти, доступ до яких обмежений специфічною групою користувачів, наприклад корпоративні внутрішні веб-сайти для співробітників.

Веб-сайти часто присвячені конкретній темі чи меті, як-от новини, освіта, комерція, розваги чи соціальні мережі. Вони включають гіперпосилання для навігації між сторінками, і зазвичай починаються з головної сторінки.

Користувачі можуть переглядати веб-сайти на різноманітних пристроях, включаючи настільні комп'ютери, ноутбуки, планшети та смартфони, використовуючи веб-браузер.

Всесвітня павутина була створена у 1990 році Тімом Бернерсом-Лі, фізиком із ЦЕРН. У 1993 році ЦЕРН оголосив, що Всесвітня павутина буде відкритою і безкоштовною для використання всіма. До появи HTTP для доступу до файлів на сервері використовувалися інші протоколи, такі як FTP та Gopher, які пропонували просту структуру каталогів для навігації та завантаження файлів. Веб-документи спочатку подавалися переважно у вигляді простих текстових файлів або у форматах текстових процесорів.

Веб-сайти мають широкий спектр застосувань і можуть слугувати різним цілям, включаючи персональні веб-сайти, корпоративні веб-сайти бізнесу, урядові веб-сайти, веб-сайти організацій тощо. Вони можуть належати приватним особам, комерційним організаціям чи іншим установам, і зазвичай фокусуються на певній темі чи меті.

Між веб-сайтами часто існують гіперпосилання, що забезпечують перехід від одного сайту до іншого, це може робити межі між різними сайтами нечіткими з точки зору користувача.

Деякі веб-сайти вимагають від користувачів реєстрації або підписки для доступу до певного вмісту. Це можуть бути бізнес-сайти, новинні портали, академічні журнали, ігрові платформи, файлообмінні сервіси, форуми, електронні поштові служби, соціальні мережі, сайти з даними про фондовий ринок у реальному часі та інші сервіси.

Термін "веб-сайт" зазнав еволюції у написанні. Хоча спочатку він писався як "Веб-сайт" (з великою буквою "В", оскільки "Веб" вважався власною назвою для Всесвітньої павутини), сучасне написання — "веб-сайт" — стало більш поширеним і визнаним. Ця зміна відображена в провідних посібниках зі стилю, включаючи Чиказький посібник стилю та AP Stylebook.

Статичний веб-сайт складається з веб-сторінок, які зберігаються на сервері у форматі, готовому до відправлення веб-браузеру клієнта. Основою для створення таких веб-сторінок є HTML (HyperText Markup Language), який використовується для структуризації змісту. Каскадні таблиці стилів (CSS) використовуються для надання стилістичного оформлення цьому HTML-контенту. Зображення включаються для покращення візуального сприйняття та як частину основного контенту, а аудіо та відео можуть бути включені як нерухомі (статичні) елементи, якщо вони відтворюються автоматично або не вимагають взаємодії з користувачем.

Статичні веб-сайти, як правило, відображають однакову інформацію для всіх відвідувачів, схоже на роздачу друкованих брошур. Вони надають стандартну інформацію протягом тривалого періоду часу. Хоча власник веб-

сайту може вносити періодичні оновлення, але редагування тексту, фотографій та іншого вмісту зазвичай виконується вручну. Це може вимагати від власника базових навичок веб-дизайну та відповідного програмного забезпечення.

Типові приклади статичних веб-сайтів включають класичні веб-сайти, веб-сайти з обмеженою кількістю сторінок (наприклад, п'ятисторінкові веб-сайти) або "веб-сайти-брошури". Ці сайти зазвичай надають фіксовану, незмінну інформацію користувачеві, яка може включати деталі про компанію, її продукти та послуги, і яка представлена через текст, фотографії, анімації та аудіо/відео матеріали.

Статичні веб-сайти можуть включати серверні компоненти, такі як серверне включення (SSI), для спрощення процесу редагування, наприклад, щоб мати спільні елементи, такі як меню навігації, на кількох сторінках. Однак, оскільки взаємодія з користувачем на таких сайтах не змінюється, вони все ще вважаються статичними.

На противагу статичним, динамічні веб-сайти автоматично змінюються та адаптуються. Вони генерують веб-сторінки "на льоту" за допомогою серверного програмування, яке виробляє HTML, в той час як CSS контролює зовнішній вигляд. Для створення динамічних сайтів використовуються різноманітні програмні системи та фреймворки, такі як CGI, Java-сервлети, JSP, ASP, ColdFusion, Perl, PHP, Python, Ruby та інші.

Динамічні сайти можуть адаптувати вміст до поведінки користувача або змін в оточенні, наприклад, відображати актуальні новини, змінювати вміст залежно від користувацьких запитів або попередньої історії переглядів. Вони можуть бути інтерактивними, зберігати та читати дані з cookies браузера чи створювати персоналізований вміст згідно з попередніми діями користувача. Динамічний HTML, що використовує JavaScript, дозволяє веб-браузеру інтерактивно змінювати вміст сторінки.

Іноді динамічний веб-сайт може імітувати поведінку динамічного сайту за допомогою періодичного автоматичного оновлення великої кількості

статичних сторінок, щоб зберегти продуктивність і уникнути необхідності запускати динамічний механізм для кожного користувача або сеансу.

У початковий період існування інтернету, веб-сайти обмежувались лише текстом, але з часом на них почали з'являтися зображення. Згодом, за допомогою різних плагінів для браузерів, веб-сайти набули можливості відтворювати аудіо та відео та стали інтерактивними. Це дозволило створювати на веб-сайтах складні програми, аналогічні до настільних, наприклад текстові редактори. Деякі з відомих плагінів - це Microsoft Silverlight, Adobe Flash, Adobe Shockwave, а також Java-аплети. HTML5 вже включає можливості аудіо та відео без потреби у додаткових плагінах. JavaScript, який є частиною більшості сучасних веб-браузерів, дозволяє розробникам сайтів надсилати інструкції браузеру для інтерактивної зміни контенту сторінки та комунікації з веб-сервером за потреби. Структура, в якій веб-браузери представляють вміст, називається об'єктною моделлю документа (DOM), а метод - динамічним HTML.

WebGL, який є частиною сучасного JavaScript API, використовується для створення інтерактивної 3D-графіки без плагінів. Він дозволяє користувачам насолоджуватися інтерактивним контентом, таким як 3D-анімації та відео, надаючи більш інтуїтивно зрозумілий досвід. [5]

Однією з важливих тенденцій у веб-дизайні 2010 року був "адаптивний дизайн", який забезпечує оптимальний досвід перегляду, адаптуючи веб-сайти до різних пристроїв та мобільних платформ. Такий підхід дозволяє веб-сайтам змінювати свій інтерфейс для забезпечення найкращого користувацького досвіду. [6]

Веб-сайти можна розділити на дві великі категорії - статичні та інтерактивні. Інтерактивні сайти є частиною спільноти веб-сайтів Web 2.0 і дозволяють взаємодіяти між власником сайту та відвідувачами або користувачами сайту. Статичні сайти обслуговують або збирають інформацію, але не дозволяють взаємодіяти з аудиторією або користувачами

безпосередньо. Деякі веб-сайти є інформаційними або виготовляються ентузіастами або для особистого користування чи розваги. Багато веб-сайтів мають на меті заробляти гроші, використовуючи одну або кілька бізнес-моделей, зокрема:

- Розміщення цікавого контенту та продаж контекстної реклами або шляхом прямих продажів, або через рекламну мережу.
- Електронна комерція: товари чи послуги купуються безпосередньо через веб-сайт
- Реклама товарів або послуг, доступних у цегельному та будівельному бізнесі
- Freemium: базовий вміст доступний безкоштовно, але преміум-вміст вимагає оплати (наприклад, веб-сайт WordPress, це платформа з відкритим кодом для створення блогу чи веб-сайту).

Деякі веб-сайти можуть включати елементи різних категорій. Наприклад, корпоративний сайт може не тільки рекламувати продукцію компанії, але й містити інформаційні матеріали, такі як технічні документи. Крім основних категорій, існує безліч специфічних підкатегорій. Наприклад, сайт для дорослих може бути різновидом сайту електронної комерції чи бізнес-сайту, оскільки пропонує платні підписки, або мати елементи соціальної мережі. Фан-сайти часто присвячені певним знаменитостям. Веб-сайти мають архітектурні обмеження, такі як обчислювальна потужність, призначена для сайту. Для обробки великих обсягів трафіку великі веб-сайти, такі як Facebook, Yahoo!, Microsoft і Google, використовують численні сервери та обладнання для балансування навантажень, наприклад, комутатори служб вмісту від Cisco для розподілу трафіку між різними комп'ютерами. Станом на початок 2011 року Facebook використовував 9 дата-центрів з близько 63000 серверами.

За даними Netcraft, компанії, яка займається моніторингом Інтернету з 1995 року, у лютому 2009 року кількість веб-сайтів із доменними іменами та

вмістом на них становила 215 675 903, у порівнянні з 19 732 сайтами в серпні 1995 року. [8] Після досягнення позначки в 1 мільярд веб-сайтів у вересні 2014 року, що було підтверджено оглядом веб-серверів Netcraft у жовтні 2014 року і про що повідомив засновник Всесвітньої павутини Тім Бернерс-Лі у Twitter, загальна кількість веб-сайтів зменшилася, опустившись нижче 1 мільярда через коливання кількості неактивних сайтів. Тим не менш, станом на березень 2016 року, кількість веб-сайтів знову перевищила 1 мільярд і продовжувала зростати. [9]

Веб-розробка охоплює діяльність, пов'язану зі створенням веб-сайтів для Інтернету або внутрішніх корпоративних мереж. [1] Сфера веб-розробки охоплює широкий спектр проектів: від створення простих сторінок, що складаються з чистого тексту, до складних веб-додатків, електронної комерції та соціальних мереж. До типових завдань у веб-розробці належать веб-інженерія, веб-дизайн, створення контенту, робота з клієнтською стороною, скриптинг на стороні клієнта та сервера, налаштування безпеки сервера та мережі, а також розвиток сфери електронної комерції.

У професійному співтоваристві під терміном "веб-розробка" зазвичай мають на увазі ключові аспекти створення веб-сайтів, які не стосуються дизайну, а саме написання коду та розмітки. [2] Веб-розробники часто використовують системи управління контентом (CMS), щоб полегшити та зробити більш доступним процес редагування вмісту для тих, хто має обмежені технічні навички.

У великих організаціях команди веб-розробників можуть нараховувати сотні спеціалістів, що слідують стандартним методологіям, як-от Agile, у процесі створення веб-сайтів. В невеликих організаціях роль веб-розробника може бути покладена на одного спеціаліста або виконуватися як додаткова функція іншими працівниками, наприклад, графічним дизайнером або IT-техніком. Часто веб-розробка є результатом спільної роботи кількох відділів, а не окремої групи. Спеціалізації у веб-розробці включають розробників інтерфейсів, серверних розробників та full-stack розробників. Веб-розробники

відповідають за функціональність і візуальний вигляд сайтів, що відображаються у браузерах користувачів, тоді як серверні розробники працюють із серверною частиною.

З моменту комерціалізації Інтернету, веб-розробка стала стрімко розвиватися як галузь. Її розвиток значною мірою обумовлений бажанням компаній використовувати свої веб-сайти для реклами та продажу товарів і послуг споживачам. [3]

Веб-розробка значною мірою покладається на інструменти з відкритим кодом, такі як BerkeleyDB, GlassFish, стек LAMP (Linux, Apache, MySQL, PHP), Perl / Plack. Ці інструменти знизили витрати на навчання для веб-розробників. Іншим чинником, що стимулює зростання цієї галузі, є поява легких у використанні програм для веб-розробки, як-от Adobe Dreamweaver, BlueGriffon та Microsoft Visual Studio. Хоча для ефективного використання цих програм потрібні знання HTML або мов програмування, основні навички можна швидко освоїти.

Постійне розширення набору інструментів та технологій сприяло створенню більш динамічних та інтерактивних веб-сайтів. Тепер веб-розробники також можуть пропонувати традиційні настільні програми у вигляді веб-служб. Це сприяло децентралізації поширення інформації та медіа. Наприклад, хмарні сервіси, такі як Adobe Creative Cloud, Dropbox та Google Drive, дозволяють користувачам працювати з програмами з будь-якої точки світу, не будучи прив'язаними до одного робочого місця.

Електронна комерція є однією з ключових областей, що перетворилася завдяки веб-розробці. Сайти аукціонів, як-от eBay, змінили способи пошуку та купівлі товарів споживачами. Онлайн-магазини, такі як Amazon.com та Buy.com, трансформували досвід покупок. Також веб-розробка внесла значний вклад у сферу блогінгу. Веб-додатки, як-от WordPress та Movable Type, сприяли створенню індивідуальних блогів. Широке використання систем управління контентом з відкритим кодом та корпоративних систем

управління контентом збільшило вплив веб-розробки на онлайн-інтеракцію та спілкування.

Розвиток веб-розробки значно вплинув на особисті мережі та маркетинг. Веб-сайти вже давно перестали бути лише інструментами для бізнесу чи роботи; вони також стали важливими засобами для спілкування та соціальної взаємодії. Платформи як Facebook та Twitter дозволяють людям спілкуватися між собою, а організаціям - встановлювати більш особисті та інтерактивні зв'язки з аудиторією.

У сфері веб-розробки велике значення приділяється питанням безпеки, як-от перевірка введених даних через форми, фільтрація вихідних даних та шифрування. Існує ризик таких дій, як SQL-ін'єкції, які можуть бути здійснені користувачами з базовими знаннями веб-розробки. Веб-сайти можуть бути вразливі до несанкціонованого доступу, що загрожує безпеці таких даних, як електронні адреси, паролі та інформація кредитних карт.

Безпека сервера залежить від серверного середовища, де використовуються такі мови сценаріїв як ASP, JSP, PHP, Python, Perl або Ruby, і не завжди залежить від розробника. Проте, рекомендується проводити ретельне тестування веб-додатків перед їх запуском для попередження подібних проблем. Якщо на веб-сайті є контактна форма, вона повинна містити поле captcha, щоб запобігти автоматичному заповненню форм та розсиланню спаму.

Зміцнення порту сервера є одним із способів захисту веб-сервера від вторгнень. Різні технології використовуються для захисту інформації в Інтернеті під час її передачі. Наприклад, сертифікати TLS (раніше відомі як сертифікати SSL) видаються для запобігання онлайн-шахрайству. Багато розробників застосовують різні методи шифрування для захисту конфіденційної інформації під час передачі та зберігання. Глибоке розуміння питань безпеки є важливою частиною компетенції веб-розробника.

Оскільки нові веб-програми виявляють нові діри в безпеці навіть після тестування та запуску, оновлення виправлень безпеки часто трапляються для

широко використовуваних програм. Часто робота веб-розробників - постійно оновлювати програми, коли випускаються виправлення безпеки та виявляються нові проблеми безпеки.

1.3. Поняття контролю навчального процесу

Контроль у навчанні – це не просто механізм перевірки знань, але й важливий інструмент у процесі освіти, який дозволяє вчителю й учню ефективно взаємодіяти для досягнення навчальних цілей. Цей процес можна розглядати в кількох ключових аспектах:

1. Зворотний зв'язок: Контроль надає учителю зворотний зв'язок про рівень засвоєння матеріалу учнями. Він дозволяє виявити прогалини в знаннях та зрозуміти, які теми потребують додаткового пояснення або повторення.
2. Адаптація навчального процесу: На основі отриманої інформації вчитель може адаптувати навчальний план, змінюючи зміст, методи та форми навчання, щоб вони відповідали потребам учнів.
3. Розвиток навичок самоаналізу: Контроль також важливий для учнів, оскільки він стимулює їх до самоаналізу, самооцінки та розвитку навичок критичного мислення.
4. Виховна роль: Через систему контролю вчитель може не тільки оцінювати знання, але й виховувати в учнів відповідальне ставлення до навчання, дисципліну, цілеспрямованість.
5. Гнучкість методів: Контроль може бути реалізований через різноманітні форми, такі як письмові та усні перевірки, проекти, реферати, творчі завдання тощо, що допомагає залучити учнів і підвищити їх мотивацію.
6. Оптимізація навчального процесу: Ефективний контроль дозволяє оптимізувати весь навчальний процес, враховуючи особливості класу та індивідуальні потреби кожного учня.
7. Залучення учнів: Інтерактивні форми контролю, такі як вибіркові опитування, групова робота, дебати, сприяють активному залученню

учнів у навчальний процес, розвивають комунікативні навички та співпрацю.

8. Технології в контролі: Використання сучасних технологій, таких як електронні тести, мобільні додатки для освіти, онлайн-платформи, може зробити процес контролю більш ефективним і цікавим для учнів.
9. Розвиток особистісних якостей: Крім академічних навичок, контроль у навчанні сприяє розвитку особистісних якостей, таких як витривалість, самодисципліна, здатність до самостійного вирішення проблем.

Контроль у навчанні, таким чином, відіграє ключову роль у побудові ефективного навчального процесу, що адаптується до потреб учнів і сприяє їх всебічному розвитку.

1.4. Огляд існуючих рішень

Дистанційне навчання стало невід'ємною частиною освітньої сфери, особливо в умовах глобалізації та технологічного прогресу. Огляд існуючих платформ дистанційного навчання дозволяє оцінити їхні можливості та недоліки. Далі представлений аналіз найбільш популярних платформ.

1. Moodle

Переваги:

- Відкрите джерело, широкі можливості налаштування.
- Велика спільнота користувачів та розробників.
- Підтримка різноманітних плагінів.

Недоліки:

- Складність у використанні, потребує технічних знань для налаштування.
- Інтерфейс може здаватися застарілим.
- Потребує великих ресурсів сервера для великих груп студентів.

2. Blackboard

Переваги:

- Широкий функціонал для управління курсами.
- Інтеграція з різними електронними ресурсами та системами.

Недоліки:

- Висока вартість для інституцій.
- Може бути складним для навігації для нових користувачів.
- Відсутність гнучкості в плані налаштування.

3. Canvas

Переваги:

- Інтуїтивно зрозумілий інтерфейс.
- Хороша інтеграція з сторонніми інструментами та сервісами.
- Ефективні інструменти співпраці.

Недоліки:

- Обмеження у вільному використанні; платформа орієнтована на інституційних користувачів.
- Деякі функції можуть бути недоступні без платної підписки.
- Вимоги до широкосмугового Інтернету для деяких функцій.

4. Google Classroom

Переваги:

- Легкість у використанні та налаштуванні.
- Інтеграція з Google Apps (Docs, Sheets, Slides).
- Безкоштовне використання для шкіл та некомерційних установ.

Недоліки:

- Обмежений функціонал порівняно з іншими платформами.
- Відсутність деяких розширених інструментів оцінювання та керування.
- Залежність від інших продуктів Google.

5. Coursera, edX, Udemy (MOOC платформи)

Переваги:

- Великий вибір курсів від відомих університетів та експертів.
- Гнучкість у навчанні, можливість самостійного вивчення.

Недоліки:

- Обмежена взаємодія з викладачами.

- Сертифікати часто є платними.
- Відсутність індивідуалізації навчального процесу.

Загальні недоліки дистанційного навчання:

1. Відсутність особистісного контакту: Дистанційне навчання часто веде до відсутності прямого спілкування між студентами та викладачами, що може негативно впливати на процес навчання та мотивацію.
2. Технічні проблеми: Потреба в постійному доступі до Інтернету, високі вимоги до технічного забезпечення.
3. Самодисципліна та організація: Навчання на відстані вимагає високого рівня самодисципліни та організації з боку студентів.
4. Обмеження практичних навичок: Неможливість проведення деяких видів практичних занять та лабораторних робіт в онлайн-форматі.
5. Стандартизація матеріалу: Часто матеріал та ресурси дистанційного навчання є стандартизованими і не враховують індивідуальні потреби кожного студента.
6. Оцінювання та контроль: Оцінювання знань на дистанційних курсах може бути менш ефективним через відсутність безпосереднього контролю.

1.5. Висновки до розділу 1

1. Поняття веб-додатку визначено як комплексне програмне забезпечення, яке працює в інтернет-браузері та надає користувачам інтерактивний інтерфейс для виконання навчальних завдань та доступу до навчальних ресурсів.
2. Основні принципи веб-розробки, зокрема, використання клієнт-серверної архітектури, респонсивного дизайну та асинхронних веб-технологій, дозволяють створювати доступні та гнучкі веб-системи для дистанційного навчання.
3. Поняття контролю навчального процесу проаналізовано з точки зору вимог дистанційного навчання. Важливість інструментів оцінювання,

зворотного зв'язку та адаптації навчальних матеріалів підкреслено як ключові для підтримки успішного навчання.

4. Огляд існуючих рішень дозволив виявити спільні недоліки, такі як відсутність персоналізації, проблеми з масштабованістю та складність інтеграції з іншими системами, що мають бути враховані при розробці нової веб-системи.

На основі проведеного аналізу можна зробити висновок, що розробка веб-системи для підтримки дистанційного навчання має базуватися на сучасних веб-технологіях, що підтримують гнучкість, масштабованість та інтерактивність. Така система має включати ефективні механізми контролю та зворотного зв'язку для оптимізації навчального процесу та забезпечення високої якості освіти.

РОЗДІЛ 2
АНАЛІЗ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Огляд мови програмування

Java є високорівневою, клас-орієнтованою, об'єктно-орієнтованою мовою програмування, створеною з метою мінімізації залежностей в реалізації. Ця мова універсального призначення розроблена таким чином, щоб програмісти могли писати програми лише один раз і запускати їх на будь-якій платформі, що підтримує Java, без потреби у повторній компіляції.[17] Java-програми зазвичай компілюються в байт-код, який може виконуватись на будь-якій Java-віртуальній машині (JVM), незалежно від архітектури обладнання. Синтаксис Java нагадує C і C++, але відрізняється меншими можливостями на низькому рівні в порівнянні з ними. Виконавче середовище Java пропонує динамічні функції, такі як динамічне відображення та зміна коду під час виконання, які зазвичай не доступні у традиційних компільованих мовах. Станом на 2019 рік, Java залишається однією з найбільш вживаних мов програмування, особливо для розробки клієнт-серверних веб-додатків, згідно з даними GitHub, [19][20] з ком'юніті у 9 мільйонів розробників.[21]

Java була спочатку розроблена Джеймсом Гослінгом у Sun Microsystems і випущена у травні 1995 року як ключовий компонент Java-платформи Sun Microsystems. Перші компілятори, віртуальні машини та класові бібліотеки Java були випущені Sun під власними ліцензіями. У травні 2007 року, відповідно до специфікацій Java Community Process, Sun переліцензувала більшість своїх Java-технологій під GPL-2.0 ліцензію. Oracle, яка наразі є власником Java, пропонує свою версію Java-віртуальної машини - HotSpot.

Офіційною референтною реалізацією Java-віртуальної машини є OpenJDK JVM, яка є безкоштовним і відкритим програмним забезпеченням. Вона широко використовується більшістю розробників і є стандартною JVM для майже всіх дистрибутивів Linux. Станом на березень 2022 року остання версія Java - це Java 18, а версії Java 17, 11 та 8 вважаються поточними версіями з довгостроковою підтримкою (LTS). Oracle припинила випускати безкоштовні оновлення для комерційного використання застарілої версії Java 8 LTS у січні 2019 року, проте продовжує підтримувати Java 8 з оновленнями для особистого використання на невизначений термін. Тим часом, інші виробники пропонують безкоштовні збірки OpenJDK 8 і 11, які продовжують отримувати оновлення безпеки та інші оновлення.

Oracle та інші компанії настійно рекомендують видаляти застарілі та непідтримувані версії Java через безпекові ризики, пов'язані зі старими версіями.[22] Oracle радить своїм користувачам переходити на підтримувані версії, такі як одну з версій LTS (8, 11, 17).

Щодо Java JVM і байт-коду, однією з основних цілей Java є забезпечення переносимості, тобто можливості виконання програм на різноманітних апаратних та програмних платформах без змін. Це досягається компіляцією Java-коду в проміжний байт-код, що може виконуватися на будь-якій Java-віртуальній машині (JVM), незалежно від конкретної апаратної платформи. Інструкції байт-коду Java схожі на машинний код, але призначені для виконання Java-віртуальною машиною (VM), яка адаптована під апаратне забезпечення хоста. Користувачі зазвичай встановлюють Java Runtime Environment (JRE) на свої пристрої для запуску автономних Java-програм або використовують веб-браузер для запуску Java-аплетів.

Стандартні бібліотеки Java забезпечують уніфікований спосіб взаємодії з функціями, специфічними для певного хоста, такими як графіка, обробка потоків даних та мережеві можливості.

Хоча використання універсального байт-коду спрощує процес перенесення програми на різні платформи, інтерпретація байт-коду в

машинний код традиційно призводила до сповільнення виконання програм порівняно з нативними виконуваними файлами. Для вирішення цієї проблеми були розроблені компілятори Just-in-time (JIT), які компілюють байт-код у машинний код прямо під час виконання програми. Ця технологія була введена на ранніх етапах розвитку Java. Компілятор HotSpot Java, наприклад, поєднує в собі два компілятори, а GraalVM (включений в Java 11, але вилучений у Java 16) дозволяє багаторівневу компіляцію.[47] Сама Java є незалежною від платформи, а її адаптація до конкретної платформи здійснюється за допомогою Java віртуальної машини (JVM), яка перетворює байт-код Java на машинний код платформи.[48]

Що стосується продуктивності, програми, написані на Java, традиційно мали репутацію більш повільних та вимогливих до пам'яті в порівнянні з програмами, написаними на C++.[49][50] Проте, швидкість виконання програм на Java значно покращилася після впровадження JIT компіляції в 1997-1998 роках з Java 1.1,[51] доповнення мовних особливостей для кращого аналізу коду (наприклад, внутрішні класи, клас StringBuilder, опціональні асерти тощо) та оптимізації в Java віртуальній машині, особливо з HotSpot, яка стала стандартною JVM Sun в 2000 році. З Java 1.5 продуктивність була покращена завдяки введенню пакету java.util.concurrent, який включає безблокові реалізації ConcurrentMaps та інших багатоядерних колекцій, та додатково удосконалена в Java 1.6.

Java впроваджує автоматичне управління пам'яттю через механізм збору сміття. Програмісти в Java відповідають за створення об'єктів, а Java Runtime Environment (JRE) - за звільнення пам'яті, коли ці об'єкти більше не потрібні. Це стає можливим, коли на об'єкт вже не посилаються, тобто він стає недосяжним, що дозволяє збірнику сміття звільнити пам'ять, яку він займає. Тим не менш, ситуації, подібні до витоку пам'яті, можуть виникати, коли програма зберігає посилання на об'єкти, які більше не використовуються, особливо якщо ці об'єкти зберігаються в колекціях, які продовжують

використовуватися. Спроби викликати методи на нульовому посиланні викликають виняток нульового посилання.[53][54]

Основна ідея автоматичного управління пам'яттю в Java полягає в тому, що вона звільняє програмістів від необхідності явного управління пам'яттю, яке є загальним для деяких інших мов програмування, де пам'ять виділяється як на стеку, так і на купі. У випадках, коли програмісти відповідають за явне управління пам'яттю, неуважне використання може призвести до витоків пам'яті або помилок, таких як спроби доступу до вже звільненої пам'яті, що робить програму нестабільною або призводить до її аварійного завершення. Хоча розумні вказівники можуть допомогти вирішити ці проблеми, вони вводять власні накладні витрати та складності. Важливо зазначити, що збір сміття в Java не запобігає "логічним" витокам пам'яті, коли об'єкти, які все ще посилаються, але не використовуються, займають пам'ять.

Збір сміття в Java може відбуватися в будь-який час і, ідеально, відбувається, коли програма не активна. Він гарантовано виконується, коли в купі не вистачає вільної пам'яті для виділення нового об'єкта, що може спричинити зупинку програми. Явне управління пам'яттю не підтримується в Java.

Java не дозволяє арифметику вказівників, яка є характерною для мов програмування C/C++. У Java неможливо виконувати арифметичні операції з адресами об'єктів, такі як додавання або віднімання зміщень. Ця особливість дозволяє збірнику сміття переміщати об'єкти у пам'яті без порушення посилань на них, що сприяє безпеці та типовій безпеці в Java.

Також як і в C++ та інших об'єктно-орієнтованих мовах, у Java змінні примітивних типів зберігаються безпосередньо у полях об'єктів або на стеку (для методів), а не в купі, як це зазвичай відбувається з непримітивними типами даних. Це було свідомим рішенням розробників Java для підвищення продуктивності.

Java пропонує декілька видів збірників сміття. З появою Java 9, HotSpot по замовчуванню використовує Garbage First Garbage Collector (G1GC).[55]

Існують також інші збірники сміття, які можна налаштувати для управління купою. Для більшості Java-програм G1GC є достатньо ефективним. У Java 8 був використаний Parallel Garbage Collector.

Незважаючи на автоматичне управління пам'яттю, програмісти все ще несуть відповідальність за належне управління іншими ресурсами, такими як мережеві з'єднання, підключення до баз даних, дескриптори файлів тощо, особливо в контексті винятків.

Jakarta Servlet (раніше відомий як Java Servlet) є програмним компонентом Java, що розширює можливості сервера. Сервлети часто використовуються для реалізації веб-контейнерів, що господарюють веб-додатками на веб-серверах, тим самим функціонуючи як серверний компонент веб-API. Це аналог Java технологій динамічного веб-контенту, таких як PHP і ASP.NET.

Jakarta Servlet - це компонент платформи Jakarta EE, який обробляє запити у середовищі Java. Він слідує стандарту API сервлетів Jakarta, який визначає як Java класи мають відповідати на запити. Хоча сервлети теоретично можуть взаємодіяти через будь-який протокол клієнт-сервер, найчастіше вони використовуються для роботи з HTTP. Таким чином, термін "сервлет" часто вживається як синонім "HTTP-сервлета".[2] Сервлети дозволяють розробникам додавати динамічний вміст до веб-сервера, використовуючи Java. Згенерований вміст зазвичай включає HTML, XML або частіше JSON. Сервлети також можуть підтримувати збереження стану між різними транзакціями сервера за допомогою HTTP-куків або зіставлення URL-адрес.

API Jakarta Servlet було частково замінено двома іншими стандартами Java для веб-сервісів:

- Jakarta RESTful Web Services (JAX-RS 2.0), який є корисним для AJAX, JSON, та REST-сервісів.
- Jakarta XML Web Services (JAX-WS), використовуваний для SOAP-веб-сервісів.

Для розгортання та запуску сервлета необхідний веб-контейнер. Веб-контейнер, також відомий як контейнер сервлетів, є частиною веб-сервера, що взаємодіє з сервлетами. Веб-контейнери керують життєвим циклом сервлетів, зіставляють URL-адреси з конкретними сервлетами та перевіряють права доступу для запитувачів URL-адрес.

API сервлетів, що міститься в ієрархії пакетів Java javax.servlet, встановлює очікувані взаємодії між веб-контейнером та сервлетом.[2] Сервлет є Java-об'єктом, що обробляє запити та генерує відповіді на них. Пакет javax.servlet.http визначає HTTP-специфічні підкласи загальних елементів сервлетів, включаючи механізми управління сесіями, які відстежують взаємодії між веб-сервером та клієнтом. Сервлети можна пакувати у WAR-файли для розгортання як веб-додатків.

Сервлети Jakarta можуть бути автоматично створені з Jakarta Server Pages (JSP) за допомогою компілятора Jakarta Server Pages. Основна відмінність між сервлетами та JSP полягає у способі вбудовування: у сервлетах HTML зазвичай вбудований у код Java, тоді як JSP дозволяє вбудовувати Java-код у HTML. Попри те, що пряме використання сервлетів для генерації HTML вже не таке розповсюджене, вони все ще активно використовуються у веб-фреймворку MVC вищого рівня Jakarta EE (JSF) для низькорівневої обробки HTTP-запитів і відповідей через FacesServlet. Також їх використання залишається актуальним у "Моделі 2" - різновиді архітектури модель-вид-контролер, де сервлети поєднуються з JSP.

На даний момент актуальною є версія Servlet 5.0.[3]

Jakarta Server Pages (раніше JavaServer Pages) - це технологія, що дозволяє розробникам створювати динамічно генеровані веб-сторінки, використовуючи HTML, XML, SOAP чи інші типи документів. JSP, запущений у 1999 році компанією Sun Microsystems[1], функціонально схожий на PHP та ASP, але використовує Java як мову програмування. Для запуску та розгортання JSP необхідний веб-сервер, сумісний з контейнером сервлетів, наприклад Apache Tomcat або Jetty.

Архітектурно, JSP можна вважати високорівневою абстракцією сервлетів Java. JSP перетворюються в сервлети під час виконання, тобто кожен JSP є сервлетом; скомпільовані JSP-сервлети кешуються і повторно використовуються, поки не зміниться оригінальний JSP-файл.[2]

Jakarta Server Pages можуть використовуватися як окремо, так і як частина архітектури модель-вид-контролер, де вони функціонують як компонент виду. Зазвичай, JavaBeans використовуються як модель, а Java сервлети (або фреймворки на кшталт Apache Struts) - як контролер у цій архітектурі. Цей підхід є характерним для архітектури Model 2.[3]

Jakarta Server Pages (JSP) дозволяє розробникам інтегрувати код Java та спеціально визначені дії зі статичним веб-розмітковим контентом, як-от HTML. Створені таким чином сторінки компілюються та виконуються на сервері для подальшої доставки документа. Скомпільовані JSP-сторінки, разом з усіма залежними Java-бібліотеками, містять Java-байт-код, а не машинний код. Це означає, що, як і інші програми Java (.jar), JSP-код має бути виконаний на Java віртуальній машині (JVM), яка забезпечує взаємодію з операційною системою хоста сервера та створює абстрактне середовище, незалежне від платформи.

JSP зазвичай використовуються для генерації HTML- та XML-документів, але з допомогою OutputStream можуть доставляти й інші типи даних.[4] У процесі роботи з JSP веб-контейнер створює ряд неявних об'єктів, таких як запит, відповідь, сесія, а також контекст сторінки та інші. Ці об'єкти створюються JSP Engine під час етапу перекладу JSP.

Компілятор JavaServer Pages є програмою, яка перетворює JSP у виконувані Java сервлети. Цей компілятор зазвичай інтегрований в сервер додатків і автоматично активується при першому запиті до JSP. Окрім того, JSP можуть бути попередньо скомпільовані для підвищення продуктивності або перевірки на помилки як частина процесу збірки.[9]

Деякі JSP-контейнери дозволяють налаштовувати частоту перевірки часових міток файлів JSP для визначення їх змін. Зазвичай, під час розробки

програмного забезпечення цей інтервал встановлюється коротким (наприклад, в секундах), а для розгорнутих веб-додатків - довшим (можливо, в хвилинах або навіть відключеним), щоб оптимізувати продуктивність.[10]

2.2. Огляд середовища розробки

IntelliJ IDEA — це інтегроване середовище розробки (IDE), створене на мові Java для розробки комп'ютерного програмного забезпечення. Розроблене компанією JetBrains (раніше відомою як IntelliJ), IntelliJ IDEA доступне як у вигляді Apache 2 ліцензованої версії спільноти, так і у вигляді приватної комерційної версії. Обидві версії можуть бути використані для комерційної розробки.[3][4]

Історія IntelliJ IDEA почалася у січні 2001 року, коли була випущена перша версія. Вона стала одним з перших IDE для Java, яке надавало інтегровані можливості для розширеної навігації та рефакторингу коду.[5][6] У 2010 році IntelliJ IDEA отримала найвищу оцінку в тестуванні InfoWorld серед чотирьох провідних Java IDE: Eclipse, IntelliJ IDEA, NetBeans і JDeveloper.[7]

У грудні 2014 року Google анонсувала Android Studio версії 1.0, IDE з відкритим кодом для розробки Android-додатків, засновану на спільнотівській версії IntelliJ IDEA з відкритим кодом.[8] До інших середовищ розробки, побудованих на базі фреймворку IntelliJ, належать AppCode, CLion, DataGrip, GoLand, PhpStorm, PyCharm, Rider, RubyMine, WebStorm та MPS.[9]

Особливості IntelliJ IDEA включають допомогу в кодуванні, такі як:

- Автозавершення коду, засноване на аналізі контексту.
- Навігацію по коду, яка дозволяє легко переходити до класів або оголошень у коді.
- Рефакторинг коду для покращення його структури та читабельності.
- Налагодження для ідентифікації та виправлення помилок.
- Лінтинг і інструменти для виявлення та виправлення невідповідностей коду.[12][13]

IntelliJ IDEA оснащене широким спектром вбудованих інструментів та функцій інтеграції, які спрощують процес розробки. Серед цих інструментів[12]:

Інтеграція з Інструментами Збірки та Пакування: IntelliJ IDEA підтримує інтеграцію з популярними інструментами збірки та пакування, такими як Grunt, Bower, Gradle і SBT, дозволяючи розробникам легко управляти процесами збірки і розгортання їхніх додатків.

Підтримка Систем Контролю Версій: IntelliJ IDEA інтегрується з різними системами контролю версій, включаючи Git, Mercurial, Perforce і SVN. Це дозволяє розробникам легко вести версійний контроль своїх проектів безпосередньо з середовища розробки.

Інтеграція з Базами Даних: У версії Ultimate, IntelliJ IDEA надає можливість безпосереднього доступу до баз даних, таких як Microsoft SQL Server, Oracle, PostgreSQL, SQLite і MySQL. Ця інтеграція включає вбудовану версію DataGrip, що дозволяє виконувати різні операції з базами даних безпосередньо з IDE.

Екосистема Плагінів:

IntelliJ IDEA підтримує потужну екосистему плагінів, яка дозволяє розширювати функціональність IDE. Користувачі можуть завантажувати і встановлювати плагіни з веб-сайту сховища плагінів IntelliJ або через вбудовану функцію пошуку і встановлення плагінів у самому IDE. Залежно від видання IDE (Community або Ultimate), доступно більше 3000 плагінів для кожного з них станом на 2019 рік.[14] Ця різноманітність плагінів дозволяє користувачам налаштовувати своє середовище розробки згідно зі своїми конкретними потребами та перевагами.

2.3. Огляд додаткового інструментарію

HTML (HyperText Markup Language) — це основна мова розмітки, що використовується для створення веб-документів, які відображаються у веб-браузерах. Ця мова доповнюється іншими технологіями, такими як каскадні

таблиці стилів (CSS) та мови сценаріїв, наприклад, JavaScript, для покращення візуальної презентації і додавання функціональності до веб-сторінок.

Веб-браузери читають HTML-документи, отримані з веб-сервера чи локального сховища, і перетворюють їх у мультимедійні веб-сторінки. HTML описує структуру веб-сторінки семантично і спочатку включав інструкції для візуального оформлення документа.

Основні елементи HTML слугують як будівельні блоки для сторінок. За допомогою HTML можна вставляти зображення, інтерактивні форми та інші об'єкти у веб-сторінки. HTML дозволяє створювати структуровані документи, позначаючи семантику тексту, таку як заголовки, абзаци, списки, посилання, цитати тощо. Елементи HTML відокремлюються за допомогою тегів, які записуються в кутових дужках. Деякі теги, наприклад <img /> та <input />, безпосередньо додають вміст на сторінку, тоді як інші, як <p>, форматують та надають контекст тексту, і можуть включати інші теги в якості піделементів. Веб-браузери не відображають самі теги, а використовують їх для того, щоб визначити, як представити вміст сторінки.

HTML також може вбудовувати сценарії, наприклад на JavaScript, що впливають на поведінку та зміст веб-сторінок. Використання CSS дозволяє визначити стиль та макет вмісту. З 1997 року World Wide Web Consortium (W3C), який є колишнім розпорядником HTML і нинішнім розпорядником стандартів CSS, рекомендує використовувати CSS замість явно презентаційних елементів HTML.[2] Сучасна версія HTML, відома як HTML5, включає підтримку відтворення аудіо та відео, часто з використанням елемента <canvas> у поєднанні з JavaScript.

HTML-розмітка складається з декількох ключових компонентів, включаючи теги та їх атрибути, символьні типи даних, символьні посилання та посилання на сутності. Теги в HTML часто зустрічаються парами, наприклад, <h1> і </h1>, де перший тег є початковим (відкриваючим), а другий - кінцевим

(закриваючим). Проте деякі теги, такі як <img>, представляють порожні елементи і не мають закриваючого тега.

Одним з важливих компонентів HTML є оголошення типу документа (DOCTYPE), яке ініціює рендеринг веб-сторінки у стандартному режимі. Ось приклад класичної програми "Привіт, світ!":

```
<!DOCTYPE html>
<html>
 <head>
  <title>Це назва</title>
 </head>
 <body>
  <div>
   <p>Привіт, світ!</p>
  </div>
 </body>
</html>
```

У цьому прикладі, текст між тегами <html> і </html> описує весь вміст веб-сторінки. Текст між <body> і </body> визначає вміст сторінки, який відображається користувачу. Текст у тегах <title>Це заголовок</title> вказує назву сторінки, що зазвичай відображається на вкладках браузера або в його заголовку. Тег використовується для створення розділу на сторінці, який може бути легко стилізований. Між <head> і </head> можна визначити метадані сторінки за допомогою тега <meta>.

Оголошення <!DOCTYPE html> вказує на те, що документ використовує HTML5. Якщо ця декларація відсутня, браузери можуть переключатися в "режим химерності" для рендерингу сторінки, що може призвести до неочікуваного відображення контенту.[70]

Документи HTML структуровані через вкладені HTML-елементи, які представлені за допомогою тегів, обрамлених кутовими дужками, наприклад <p>.[71] В більшості випадків, елементи HTML вказуються через пару тегів:

початковий тег, наприклад , та закриваючий тег, наприклад . Між цими тегами розміщується текстовий вміст елемента.

Теги також можуть вміщувати додаткову розмітку, включаючи інші теги та текст, що вказує на вкладені елементи, які є дочірніми елементами до батьківського елемента. Початкові теги можуть включати атрибути, які надають додаткову інформацію, таку як ідентифікатори для стилізації або, в деяких випадках, як у <img>, посилання на зовнішній ресурс, наприклад <img src="example.com/example.jpg">.

Деякі елементи, як-от розрив рядка
 або
, є порожніми та не дозволяють вміщати будь-який вміст або інші теги. Для них потрібен лише один тег, і вони не вимагають закриваючого тегу.

Важливо зазначити, що деякі теги, як-от закриваючий тег для елемента абзацу , є необов'язковими. HTML-браузери або інші агенти можуть інтерпретувати закінчення елемента, виходячи з контексту та структурних правил HTML. Ці правила можуть бути складними і не завжди інтуїтивно зрозумілими для більшості розробників HTML.

Загальна форма HTML-елемента: <tag attribute1="value1" attribute2="value2">content</tag>. Деякі HTML-елементи визначаються як порожні та мають форму <tag attribute1="value1" attribute2="value2">. Порожні елементи, такі як
 або <img>, не містять вмісту. Назва елемента використовується у тегах, причому закриваючому тегу передує символ косої риски /, а в порожніх елементах закриваючий тег не потрібний і не дозволений. Якщо атрибути не вказані, використовуються значення за замовчуванням.

HTML-документи використовують структурну, презентаційну та гіпертекстову розмітку для організації та відображення вмісту:

1. **Структурна Розмітка:** Вона визначає семантичне призначення тексту. Наприклад, <h2>Golf</h2> встановлює "Гольф" як заголовок другого рівня. Цей тип розмітки не позначає конкретний візуальний стиль, але

більшість браузерів застосовують стилі за замовчуванням. Щоб надати додатковий стиль, використовуються каскадні таблиці стилів (CSS).[72]

2. **Презентаційна Розмітка:** Вона вказує на зовнішній вигляд тексту, не враховуючи його семантичного значення. Наприклад, жирний текст вказує на те, що текст має відображатися жирним шрифтом. Проте, презентаційна розмітка не надає інструкцій для пристроїв, які не можуть це візуально відобразити, наприклад, для аудіо-пристроїв. Для більш семантичної розмітки використовуються теги, як-от та , які мають візуально схожі відображення з та <i>, але надають більше семантичного змісту. Багато елементів презентаційної розмітки стали застарілими в HTML 4.0 на користь CSS.
3. **Гіпертекстова Розмітка:** Вона перетворює частини документу на гіперпосилання на інші документи. Наприклад, елемент Google перетворює слово "Google" у гіперпосилання. Для відображення зображення як гіперпосилання, елемент <img> вставляється в елемент <a>. Приклад: <img src="image.gif" alt="Описовий текст" width="50" height="50" border="0">. Елемент <img> є порожнім елементом з атрибутами, але без вмісту або закриваючого тегу.

Атрибути

Більшість атрибутів елемента – це пари ім’я-значення, розділені знаком = і записані в початковий тег елемента після імені елемента. Значення можна брати в одинарні або подвійні лапки, хоча значення, що складаються з певних символів, можна залишити без лапок у HTML (але не в XHTML).[73][74] Залишати значення атрибутів без лапок вважається небезпечним.[75] На відміну від атрибутів пари ім'я-значення, є деякі атрибути, які впливають на елемент просто своєю присутністю в початковому тегу елемента[6], як-от атрибут ismap для елемента img.[76]

Існує кілька загальних атрибутів, які можуть з’являтися в багатьох елементах:

- Атрибут id забезпечує унікальний ідентифікатор елемента для всього документа. Це використовується для ідентифікації елемента, щоб таблиці стилів могли змінювати його презентаційні властивості, а сценарії могли змінювати, анімувати або видаляти його вміст або презентацію. Доданий до URL-адреси сторінки, він надає глобальний унікальний ідентифікатор для елемента, як правило, підрозділу сторінки. Наприклад, ідентифікатор «Атрибути» в https://en.wikipedia.org/wiki/HTML#Attributes.
- Атрибут class забезпечує спосіб класифікації подібних елементів. Це можна використовувати для семантичних або презентаційних цілей. Наприклад, документ HTML може семантично використовувати позначення <class="notation">, щоб вказати, що всі елементи з цим значенням класу підпорядковані основному тексту документа. У презентації такі елементи можуть бути зібрані разом і представлені у вигляді виносок на сторінці замість того, щоб з'являтися там, де вони зустрічаються у джерелі HTML. Атрибути класу використовуються семантично в мікроформатах. Можна вказати кілька значень класів; наприклад, <class="notation important"> поміщає елемент як до нотації, так і до важливих класів.
- Автор може використовувати атрибут style для призначення властивостей презентації певному елементу. Вважається кращою практикою використовувати ідентифікатор елемента або атрибути класу для вибору елемента з таблиці стилів, хоча іноді це може бути занадто громіздким для простого, конкретного або спеціального стилю.
- Атрибут title використовується для додавання підтекстового пояснення до елемента. У більшості браузерів цей атрибут відображається як підказка.
- Атрибут lang визначає природну мову вмісту елемента, яка може відрізнятися від мови решти документа. Наприклад, в англомовному документі:

Ну що ж, c'est la vie, як кажуть у Франції.

Семантичний HTML є способом написання HTML-коду, який зосереджується на значенні інформації, а не на її візуальному представленні. Від самого початку HTML включав семантичну розмітку[83], але також містив презентаційні елементи, такі як теги <font>, <i> і . Існують також семантично нейтральні теги, такі як і . З кінця 1990-х років, з появою каскадних таблиць стилів (CSS), веб-авторів заохочували уникати використання презентаційної HTML-розмітки, щоб розділити презентацію від вмісту.[84]

Тім Бернерс-Лі та інші під час дискусій 2001 року про семантичну мережу прогнозували, що інтелектуальні програмні агенти зможуть автоматично сканувати Інтернет, знаходячи, фільтруючи та зіставляючи раніше не пов'язані факти, корисні для користувачів.[85] Хоча такі агенти досі не стали буденним явищем, деякі аспекти Web 2.0, мішапи та порталі порівняння цін використовують подібні концепції. Веб-сканери та пошукові системи, які автоматично читають та індексують веб-сторінки, залежать від семантичної чіткості веб-сторінок для ефективної роботи.

Семантична структура в HTML повинна бути широко та рівномірно використана, щоб дозволити пошуковим системам та іншим веб-агентам правильно інтерпретувати значення опублікованого тексту.[86] Презентаційна розмітка не рекомендується в сучасних стандартах HTML та XHTML, оскільки вона може погіршувати доступність, збільшувати витрати на обслуговування сайту та збільшувати розміри документів.[87]

Добре написаний семантичний HTML також покращує доступність веб-документів. Наприклад, екранный зчитувач або аудіобраузер може ефективно визначати структуру документа, не змушуючи користувача з порушеннями зору витрачати час на зчитування неважливої або невідповідної інформації.

Каскадні таблиці стилів (CSS) є мовою для опису візуального представлення документів, написаних мовою розмітки, такою як HTML. CSS

є однією з ключових технологій Всесвітньої мережі, поряд з HTML та JavaScript.[2]

CSS (Каскадні Таблиці Стилів) було розроблено для розділення візуальної презентації веб-сторінок від їхнього вмісту, включаючи макет, кольори, шрифти та інші аспекти дизайну.[3] Це розділення не тільки покращує доступність вмісту, але й забезпечує більшу гнучкість та контроль у визначенні характеристик презентації. Окрім того, CSS дозволяє використовувати спільне форматування на кількох веб-сторінках, шляхом вказівки CSS у окремому .css файлі. Це сприяє ефективній організації вмісту, зменшує повторюваність і дозволяє кешувати .css файли для швидшого завантаження веб-сторінок.[3]

CSS також дозволяє представляти один і той же вміст на різних пристроях (екран, друк, голос, Брайлівський рядок) у відповідних стилях. Крім того, CSS підтримує альтернативне форматування для мобільних пристроїв.[4]

Концепція каскадності в CSS означає, що визначаються певні правила пріоритету для застосування стилів, якщо декілька правил відповідають одному елементу. Специфікації CSS підтримуються Консорціумом World Wide Web (W3C), і тип MIME для CSS визначено як text/css згідно з RFC 2318. W3C надає безкоштовний інструмент для перевірки валідності CSS.[5]

Окрім HTML, CSS також підтримується іншими мовами розмітки, включаючи XHTML, XML, SVG і XUL.

Блок Декларацій у CSS:

- Блок декларацій складається зі списку декларацій у дужках, де кожна декларація включає властивість, двокрапку (:) і значення.
- Якщо в блоці є декілька декларацій, вони розділяються крапкою з комою (;). Можна використовувати додаткову крапку з комою після останньої декларації.[9]

Властивості та Значення:

- Властивості в CSS визначені у стандарті, і кожна властивість має свій набір допустимих значень.
- Значеннями можуть бути ключові слова (наприклад, "center"), числові значення (наприклад, "200px"), відсотки (наприклад, "80%") чи кольори у різних форматах (наприклад, "#FF0000", "rgb(255, 0, 0)", "rgba(255, 0, 0, 0.8)" або "hsl(0, 100%, 50%)").[12]

CSS було розроблено для відокремлення презентації веб-сторінок (такої як макет, кольори та шрифти) від їхнього вмісту, що покращує доступність, забезпечує більшу гнучкість і контроль над стилем презентації, дозволяє багатьом веб-сторінкам спільно використовувати форматування (за допомогою зовнішніх CSS файлів), і оптимізує швидкість завантаження сторінок.[3]

Одиниці Довжини в CSS:

В CSS використовуються різні одиниці довжини для лінійних мір. Вони можуть бути абсолютними (наприклад, см, мм, дюйми) або відносними (наприклад, em, ex, px). Відносні одиниці залежать від інших факторів, як-от розміру шрифту батьківського елемента. CSS 1 включав вісім основних одиниць довжини, а CSS 3 пропонує додаткові одиниці, такі як rem, vh, vmax, vmin, і vw.[13][14]

Використання CSS:

Перед появою CSS, всі атрибути презентації HTML-документів були вбудовані безпосередньо в HTML-розмітку. CSS дозволило авторам перемістити більшу частину цієї інформації в окремі стилі CSS, спрощуючи HTML-код і роблячи його більш керованим. CSS відокремлює презентацію від структури, дозволяючи визначити типографічні аспекти (такі як колір, шрифт, вирівнювання) незалежно для екранного та друкованого відображення. Відмова W3C від використання презентаційної HTML-розмітки також включає в себе пропагування використання CSS для стилювання.[15]

Наприклад, заголовок у HTML без CSS був би представлений так:

<h1>Розділ 1.</h1>

За допомогою CSS той самий заголовок можна визначити так:

```
<h1>Розділ 1.</h1>
h1 {
   color: red;
}
```

Такий підхід робить код більш читабельним, легшим для обслуговування та доступним, особливо для аудіо браузерів і читачів екрану.

Використовуючи CSS, той самий елемент можна закодувати, використовуючи властивості стилю замість атрибутів HTML презентації:

```
<h1 style="color: red;">Розділ 1.</h1>
```

Переваги цього можуть бути не відразу зрозумілі, але потужність CSS стає більш очевидною, коли властивості стилю розміщені у внутрішньому елементі стилю або, ще краще, у зовнішньому файлі CSS. Наприклад, припустимо, що документ містить елемент стилю:

```
<стиль>
  h1 {
     колір: червоний;
  }
</style>
```

Усі елементи h1 в документі автоматично стануть червоними, не вимагаючи жодного явного коду. Якщо пізніше автор захотів зробити елементи h1 синіми, це можна зробити, змінивши елемент стилю на:

```
<стиль>
  h1 {
     колір: синій;
  }
</style>
```

замість того, щоб копітко переглядати документ і змінювати колір для кожного окремого елемента h1.

MySQL є популярною системою управління реляційними базами даних (RDBMS), яка є відкритим програмним забезпеченням. Її назва поєднує "My", ім'я дочки одного зі співзасновників, Майкла Віденіуса, та абревіатуру "SQL", що означає Structured Query Language.[5][6][7]

Реляційні бази даних організують дані в таблицях, де типи даних можуть бути взаємопов'язані, що сприяє структурованому зберіганню інформації. SQL використовується для створення, оновлення та отримання даних з реляційної бази даних, а також для управління доступом користувачів до цих даних. СУБД, як MySQL, співпрацюють з операційною системою для забезпечення функціонування реляційної бази даних, управління доступом користувачів, мережевого доступу, перевірки цілісності бази даних та створення резервних копій.

MySQL є відкритим програмним забезпеченням, ліцензованим за умовами Загальної публічної ліцензії GNU, а також доступним на підставі комерційних ліцензій. Спочатку MySQL розроблявся шведською компанією MySQL AB, яку пізніше придбала Sun Microsystems, а потім Oracle Corporation.[8] Після придбання Sun Oracle у 2010 році, Віденіус створив MariaDB, відкритий проект на основі MySQL.[9]

MySQL часто використовується як частина веб-додатків у поєднанні з іншими технологіями, такими як Linux, Apache, Perl/PHP/Python (стек LAMP), і використовується багатьма відомими веб-додатками, такими як Drupal, Joomla, phpBB і WordPress, а також великими веб-сайтами, включаючи Facebook, Flickr, MediaWiki, Twitter і YouTube.[10][11][12][13][14][15]

MySQL має клієнтські додатки, які дозволяють користувачам взаємодіяти безпосередньо з базою даних за допомогою SQL, але частіше використовується у поєднанні з іншими програмами для створення програм, що вимагають реляційної бази даних.

MySQL, розроблений на мовах програмування C і C++, є відомою системою управління реляційними базами даних (RDBMS). Його SQL синтаксичний аналізатор розроблено з використанням yacc, але він включає також власний лексичний аналізатор.[16] MySQL підтримує широкий спектр операційних систем, включаючи AIX, FreeBSD, Linux, macOS, Microsoft Windows, OpenBSD, Oracle Solaris і багато інших. Це робить його гнучким рішенням для різноманітних системних платформ.[17]

MySQL використовує подвійну схему ліцензування для свого серверного програмного забезпечення та клієнтських бібліотек. Вони доступні як під GPL версії 2, так і під комерційною ліцензією, що надає гнучкість у використанні і розповсюдженні.[18]

Підтримка MySQL доступна через офіційний посібник, а також безкоштовно через канали IRC та форуми. Oracle, власник MySQL, також пропонує платну підтримку через MySQL Enterprise, яка включає різні рівні сервісу та цін. Крім того, існує ряд сторонніх організацій, які надають підтримку та послуги для MySQL.[19]

MySQL отримав позитивні відгуки за свою надійність та продуктивність. Він визнаний за свою ефективність у середньому використанні, високоякісні інтерфейси розробника та документацію. Він також отримав високу оцінку як швидкий, стабільний та справді багатокористувацький, багатопоточний SQL сервер баз даних.[20][21]

Розгортання MySQL:

- MySQL можна встановлювати з вихідного коду, але частіше він встановлюється з бінарних пакетів для спрощення процесу.
- У більшості дистрибутивів Linux MySQL може бути легко встановлено через систему керування пакетами.
- Після встановлення часто потрібна додаткова конфігурація для налаштування безпеки та оптимізації.

MySQL, що спочатку був розроблений як альтернатива більш потужним комерційним системам управління базами даних, з часом розвивався для

задоволення потреб більш масштабних застосувань. Його популярність особливо виражена у малих та середніх використаннях, часто як частина веб-додатків на основі LAMP або як окремий сервер бази даних. MySQL приваблює своєю простотою та зручністю, підтримуючись екосистемою інструментів з відкритим кодом, таких як phpMyAdmin.

Масштабування MySQL:

Масштабування Верх:

У середньому діапазоні MySQL може бути масштабований шляхом розгортання на більш потужному обладнанні, такому як сервери з кількома процесорами та великою кількістю пам'яті. Це підходить для підвищення продуктивності в рамках одного серверу.

Масштабування Горизонтально:

Для більших масштабів використання MySQL вимагає багатосерверних розгортань для покращення продуктивності та надійності. Типова високопродуктивна конфігурація може включати головний сервер, що займається записом даних, та кілька підпорядкованих серверів, які обробляють запити на читання.[96] Головний сервер реплікує дані на підпорядковані сервери через події binlog, забезпечуючи високий рівень доступності та надійності.

Кешування та Шардування:

Додаткове покращення продуктивності може бути досягнуто шляхом кешування запитів до бази даних у пам'яті (наприклад, через memcached) та розбиття бази даних на сегменти, які можуть бути розподілені між декількома серверними кластерами.[97]

Цей гнучкий підхід до масштабування робить MySQL підходящим для широкого спектру застосувань, від невеликих проектів до великих, розподілених систем.

2.4. Висновки до розділу 2

1. Java обрана в якості основної мови програмування для реалізації веб-системи через її високу ефективність, надійність та широкі можливості в застосуванні. Мова підтримує розробку складних алгоритмів та ефективну обробку великих обсягів даних, що є ключовим для функціонування веб-додатка.
2. В якості веб-фреймворку для розробки вибрано Spring, оскільки він надає гнучкість та ефективність у створенні великих проектів. Його функціонал дозволяє легко інтегрувати різноманітні веб-технології та використовувати сучасні підходи до веб-розробки. Використання інтегрованого середовища розробки, такого як IntelliJ IDEA, забезпечує зручність та продуктивність при створенні веб-додатків на Java з використанням Spring.
3. HTML та CSS визначені як основні технології для створення користувацького інтерфейсу. Вони дозволяють створювати структуровані та візуально привабливі веб-сторінки, що є необхідним для залучення та збереження уваги користувачів дистанційної веб-системи.
4. SQLite використовується як система управління базами даних через її легкість, невеликі вимоги до ресурсів та хорошу інтеграцію з Python. Ця СУБД є оптимальним вибором для веб-системи, орієнтованої на обробку та зберігання невеликих до середніх обсягів даних.

На основі проведеного аналізу можна зробити висновок, що обрані інструментальні засоби є доречними для розробки веб-системи для підтримки дистанційного навчання. Вони забезпечують необхідні можливості для створення надійної, гнучкої та зручної у використанні системи, яка може бути легко адаптована до змінних вимог користувачів та освітніх установ.

РОЗДІЛ 3
ПРОЕКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1. Аналіз варіантів використання

Діаграми випадків використання є зручним інструментом для представлення взаємодії користувачів з системою. Вони відображають, як користувачі (або інші системи) взаємодіють із системою для досягнення конкретної мети. На таких діаграмах випадки використання зображуються у вигляді кругів або еліпсів, які представляють функціональні можливості або процеси, доступні користувачам.

Ключові характеристики діаграм випадків використання:

- **Високий рівень огляду:** Діаграми випадків використання надають високорівневий огляд системи, показуючи важливі взаємодії між користувачами (або "акторами") та системою. Вони не вдаються в деталі про те, як функції реалізовані.
- **Комунікаційний інструмент:** Ці діаграми є відмінним інструментом комунікації між розробниками, менеджерами проектів, користувачами та іншими зацікавленими сторонами. Вони допомагають усім зацікавленим сторонам розуміти основні функції та можливості системи.
- **Спрощене представлення:** На відміну від більш складних діаграм, таких як діаграми класів, діаграми випадків використання надають більш спрощене і легкозрозуміле представлення системи.
- **Фокус на користувацькому досвіді:** Діаграми випадків використання зосереджені на тому, як користувач взаємодіє з системою, ідентифікуючи різні сценарії взаємодії.

- **Динамічний аспект системи:** Ці діаграми відображають динамічний аспект системи, показуючи, як система відповідає на різні дії користувача.

Діаграми випадків використання в UML (Unified Modeling Language) є важливим інструментом для візуалізації вимог системи, особливо з точки зору користувачів. Ці діаграми допомагають розробникам і зацікавленим сторонам розуміти, як користувачі взаємодіють з системою і які функції системи є для них важливими. Вони забезпечують загальне уявлення про систему, відображаючи її основні функції, але без детального технічного аналізу.

Основні елементи діаграми випадків використання:

1. **Рамки системи (System Border):**
   - Це прямокутник, який представляє систему або підсистему.
   - Верхня частина прямокутника містить назву системи, а всередині знаходяться еліпси, які представляють випадки використання.
   - Рамки системи можуть бути іноді опущені, але вони допомагають візуально відокремити випадки використання від зовнішнього середовища.
2. **Актор (Actor):**
   - Стилізований образ людини, який представляє користувачів або зовнішні сутності, які взаємодіють із системою.
   - Актори можуть бути людьми, зовнішніми системами, або іншими сутностями, які взаємодіють з системою.
   - Актори не можуть бути з'єднані між собою, крім відносин щодо спадкування або реалізації.
3. **Випадок використання (Use Case):**
   - Представлені еліпсами з назвою або описом виконуваних дій.
   - Вони описують функціональні можливості системи з точки зору кінцевих результатів для акторів.
   - Кожен випадок використання відповідає сценарію або послідовності дій, які ілюструють взаємодію акторів з системою.

Застосування діаграми випадків використання:

- Діаграми випадків використання корисні для надання загального огляду функціональних вимог системи.
- Вони є відмінним інструментом для спілкування між розробниками, аналітиками та зацікавленими сторонами.
- Хоча ці діаграми не надають повного технічного аналізу, вони важливі для розуміння загальних вимог та інтерфейсу користувача.
- Для детальнішого технічного опису системи використовуються інші види діаграм UML та документації.

На рисунку 3.1 зображено діаграму варіантів використання, яка описує можливі дії користувача в системі.

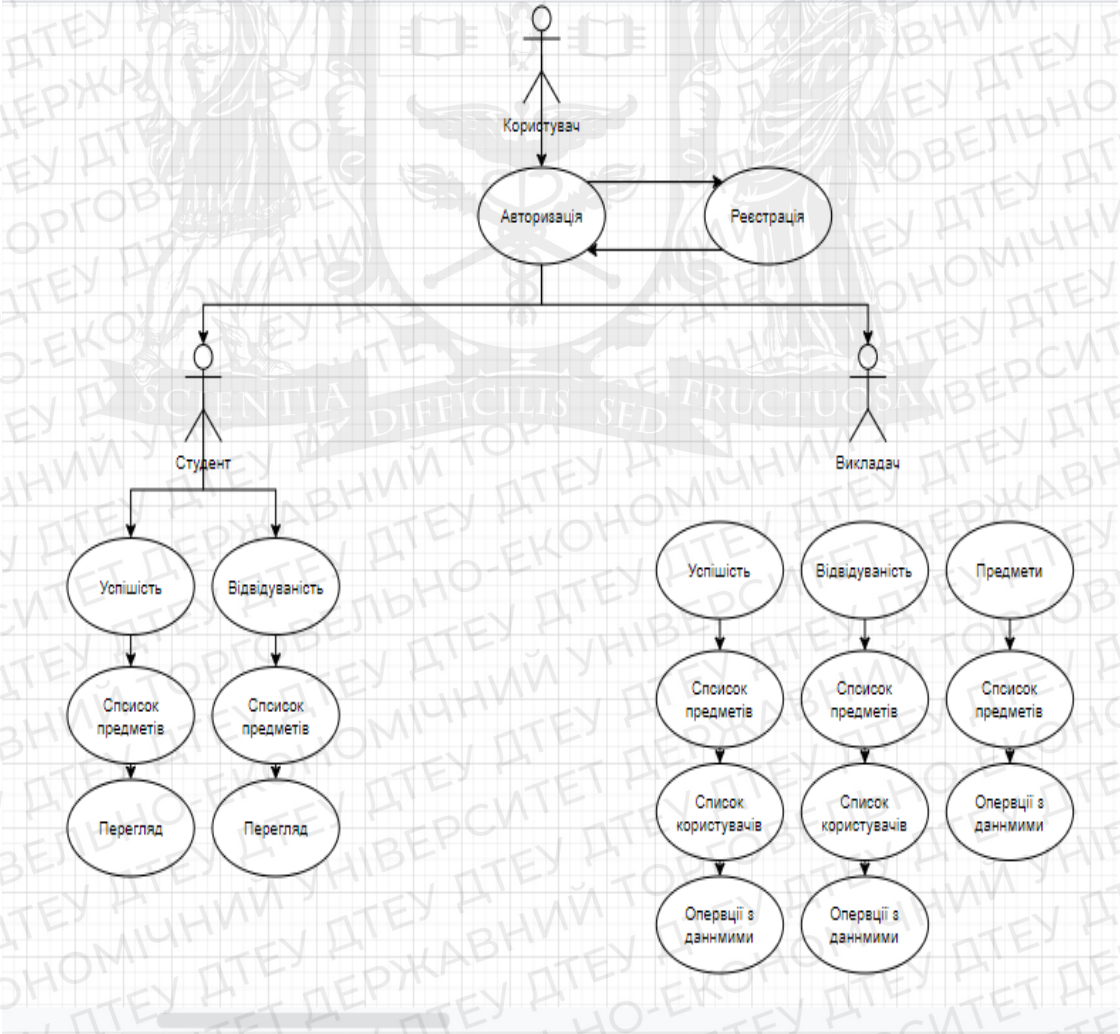

Рис. 3.1. Діаграма варіантів використання

3.2. Проектування внутрішньої будови

Діаграми класів в Уніфікованій мові моделювання (UML) є ключовим компонентом об'єктно-орієнтованого аналізу та проектування програмного забезпечення. Вони відіграють центральну роль у візуалізації, специфікації та документуванні структурних характеристик систем.

Основні характеристики діаграми класів:

1. **Представлення класів:**
   - Класи зображуються прямокутниками, поділеними на три розділи: назва класу, атрибути та операції (методи).
   - Назва класу пишеться з великої літери і розміщується в верхньому відділенні.
   - Атрибути, які представляють властивості об'єктів класу, розміщуються в середньому відділенні.
   - Операції або методи, які виконують дії, пов'язані з класом, знаходяться в нижньому відділенні.
2. **Відносини між класами:**
   - **Залежність:** Представлена штриховою лінією з відкритою стрілкою, яка вказує від клієнта до постачальника. Це односторонній зв'язок, де зміни в одному класі можуть вплинути на інший.
   - **Спадкування (наслідування):** Представлене суцільною лінією з порожньою стрілкою, спрямованою від підкласу до надкласу.
   - **Асоціація:** Вказує на відносини, де об'єкти одного класу пов'язані з об'єктами іншого класу.
   - **Агрегація та композиція:** Особливі форми асоціації, які показують відносини "частина-ціле".
3. **Моделювання даних:**
   - Діаграми класів можуть використовуватися не тільки для моделювання програм, але й для структур даних.
4. **Концептуальне та детальне моделювання:**

- На ранніх етапах проектування діаграми класів використовуються для концептуального моделювання, допомагаючи визначити основні компоненти системи та їх взаємодії.
- На більш пізніх етапах вони служать для детального моделювання, що допомагає у реалізації класів у програмному коді.

Діаграми класів є фундаментальним інструментом для об'єктно-орієнтованого аналізу та дизайну, оскільки вони забезпечують структуроване відображення програмних компонентів і їх взаємозв'язків, що сприяє кращому розумінню та ефективному проектуванню систем.

Діаграми стану або машини станів в UML ефективно доповнюють діаграми класів, надаючи детальніше розуміння поведінки системи. Вони відображають можливі стани об'єкта в рамках класу та переходи між цими станами.

Асоціації в UML:

1. **Асоціація:**
   - Асоціації вказують на статичні відносини між класами і можуть включати різні типи: двонаправлені, односпрямовані, агрегаційні та рефлексивні.
   - **Двонаправлена асоціація:** Вказує, що обидва класи знають один про одного.
   - **Односпрямована асоціація:** Один клас знає про інший, але не навпаки.
   - **Агрегація:** Спеціальний тип асоціації, що відображає відносини "частина-ціле". В UML відображається за допомогою порожнистого ромба.
2. **Агрегація vs Композиція:**
   - Агрегація описує відносини, де частини можуть існувати незалежно від цілого. Наприклад, професор може існувати незалежно від відділу.

- Композиція є сильнішою формою агрегації, яка вказує на сильну залежність між частиною та цілим. Наприклад, кімната не може існувати без будинку.

3. **Рефлексивна асоціація:**
   - Відносини, де клас асоціюється сам із собою. Наприклад, працівник може бути менеджером інших працівників.

Важливість асоціацій:

- Асоціації допомагають визначити, як класи взаємодіють та залежать один від одного. Це ключовий елемент в моделюванні об'єктно-орієнтованих систем.
- Агрегації та композиції особливо важливі для визначення життєвого циклу об'єктів та їх відносин.
- Ці відносини визначають, як дані та поведінка розподіляються між різними класами, та сприяють модульності та зменшенню залежності між різними частинами системи.

Використання діаграм станів та асоціацій в UML допомагає розробникам створювати більш гнучкі, зрозумілі та ефективно структуровані системи, що сприяє кращому проектуванню та підтримці програмного забезпечення.

Згідно з вашим описом, ви розглядаєте два основних типи відносин у UML: агрегацію та узагальнення.

Агрегація:

- У випадку з бібліотекою та студентами, агрегація відображає відносини "частина-ціле", де студент може існувати без бібліотеки.
- Графічно в UML це представляється за допомогою порожнистого ромба на кінці лінії, що з'єднує "ціле" (наприклад, бібліотеку) з "частиною" (студент).

Узагальнення:

- Узагальнення в UML відображає ієрархічні відносини між класами, де один клас (підклас) є спеціалізованою версією іншого класу (суперкласу).
- Наприклад, "дуб є типом дерева" означає, що клас "дуб" успадковує властивості та методи від загальнішого класу "дерево".
- Графічно узагальнення представляється у вигляді лінії з порожнистим трикутником на кінці, що вказує на суперклас.

Важливо пам'ятати:

- Відносини агрегації та узагальнення служать для різних цілей в моделюванні системи. Агрегація використовується для відображення структурних відносин, тоді як узагальнення використовується для відображення ієрархічних наслідувань.
- Узагальнення відображає концепцію "із-який", що дозволяє підкласам успадковувати та потенційно перевизначати поведінку та атрибути суперкласу.
- Обидва типи відносин є ключовими для створення чіткої та ефективної архітектури програмного забезпечення.

Уміле використання цих відносин дозволяє проектувати системи, які є більш гнучкими, легкими для розуміння та обслуговування.

Ваше описання включає кілька ключових елементів UML (Unified Modeling Language) - стандарту, який використовується для візуалізації, специфікації, конструювання та документації об'єктів програмних систем. Ось основні пункти з вашого опису:

Реалізація

- **Відображення**: У UML реалізація зображується як штрихова лінія з порожнистим трикутником на кінці, що вказує на інтерфейс.
- **Значення**: Цей взаємозв'язок показує, що клас чи компонент реалізує операції, вказані в інтерфейсі.

Залежність

- **Відображення**: Залежність відображається як штрихова лінія, часто зі стрілкою.
- **Значення**: Цей тип взаємозв'язку існує, коли один клас використовує функції чи методи іншого класу.

Асоціація

- **Відображення**: Асоціація зображується як пряма лінія, що з'єднує два класи.
- **Значення**: Вказує на відносини між двома класами. Може включати деталі, такі як напрямок (одно- чи двонаправлена), ролі, множинність (скільки об'єктів може бути пов'язано).

Класи Сутності

- **Відображення**: Зазвичай зображуються як звичайні класи з позначенням стереотипу "сутність".
- **Значення**: Представляють інформацію, яку обробляє система, і можуть включати поведінку, пов'язану з цією інформацією.

Кожен з цих елементів відіграє важливу роль в структуруванні та зображенні системи в UML. Вони допомагають проектувальникам і розробникам розуміти, як компоненти системи взаємодіють між собою, а також допомагають ідентифікувати структурні та поведінкові шаблони в програмній системі.

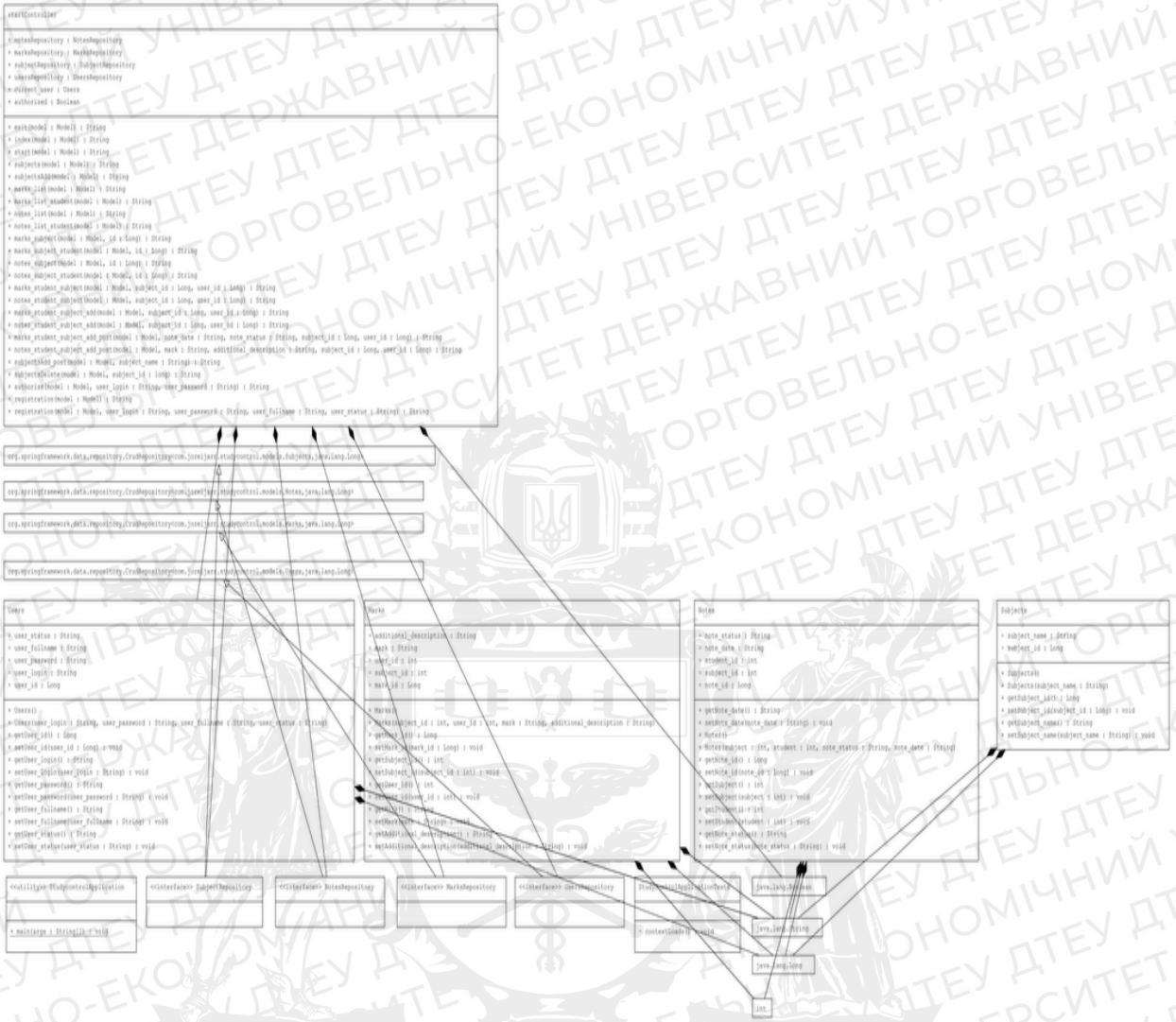

Рис. 3.2. Діаграма класів

3.3. Розробка графічного інтерфейсу системи

Графічний інтерфейс системи складається з таких основних сторінок:

- сторінка реєстрації (рис. 3.3);
- сторінка авторизації (рис. 3.4);
- головна сторінка викладача (рис. 3.5);
- сторінка предметів (рис. 3.6);
- сторінка успішності для викладача (рис. 3.7);
- сторінка відвідуваності для викладача (рис. 3.8);
- сторінка редагування даних для викладача (рис. 3.9);
- головна сторінка для студента (рис. 3.10);
- сторінка успішності для студента (рис. 3.11);
- сторінка відвідуваності для студента (рис. 3.12).

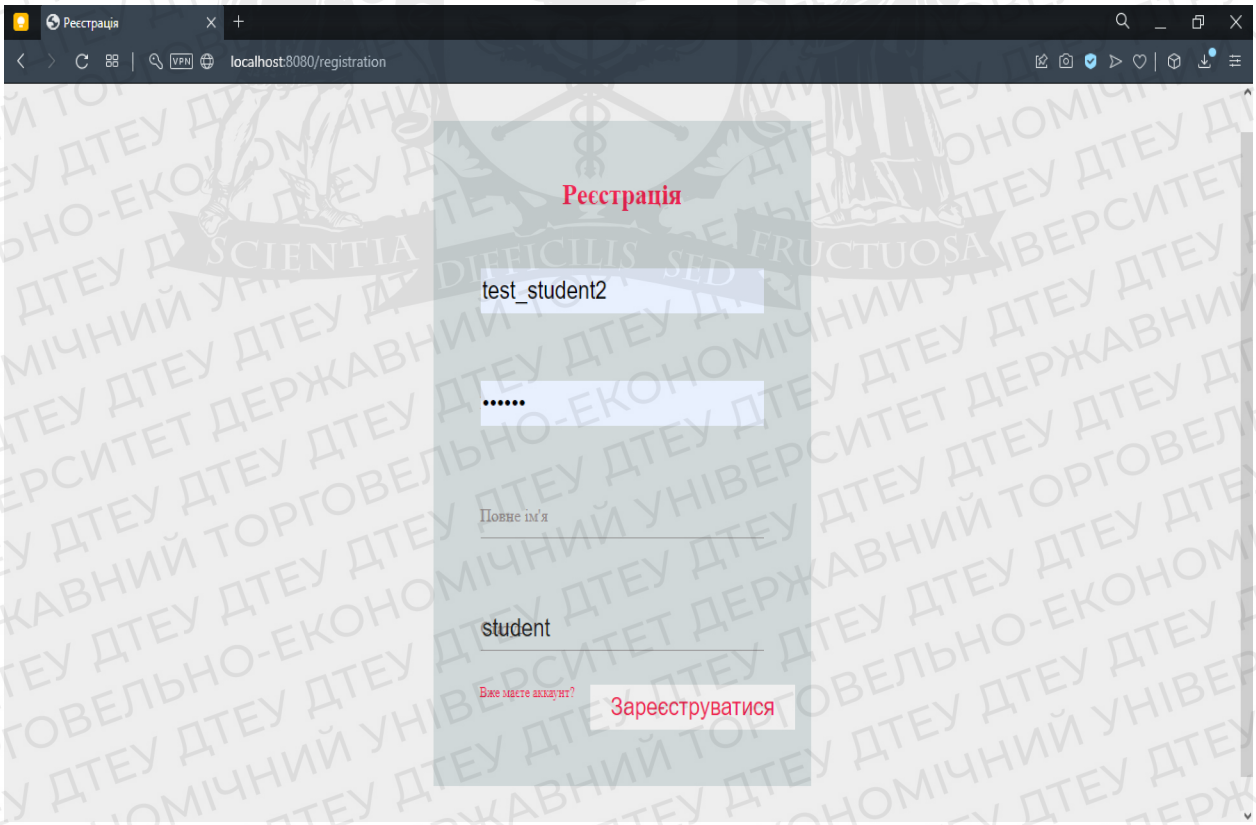

Рис. 3.3. Сторінка реєстрації

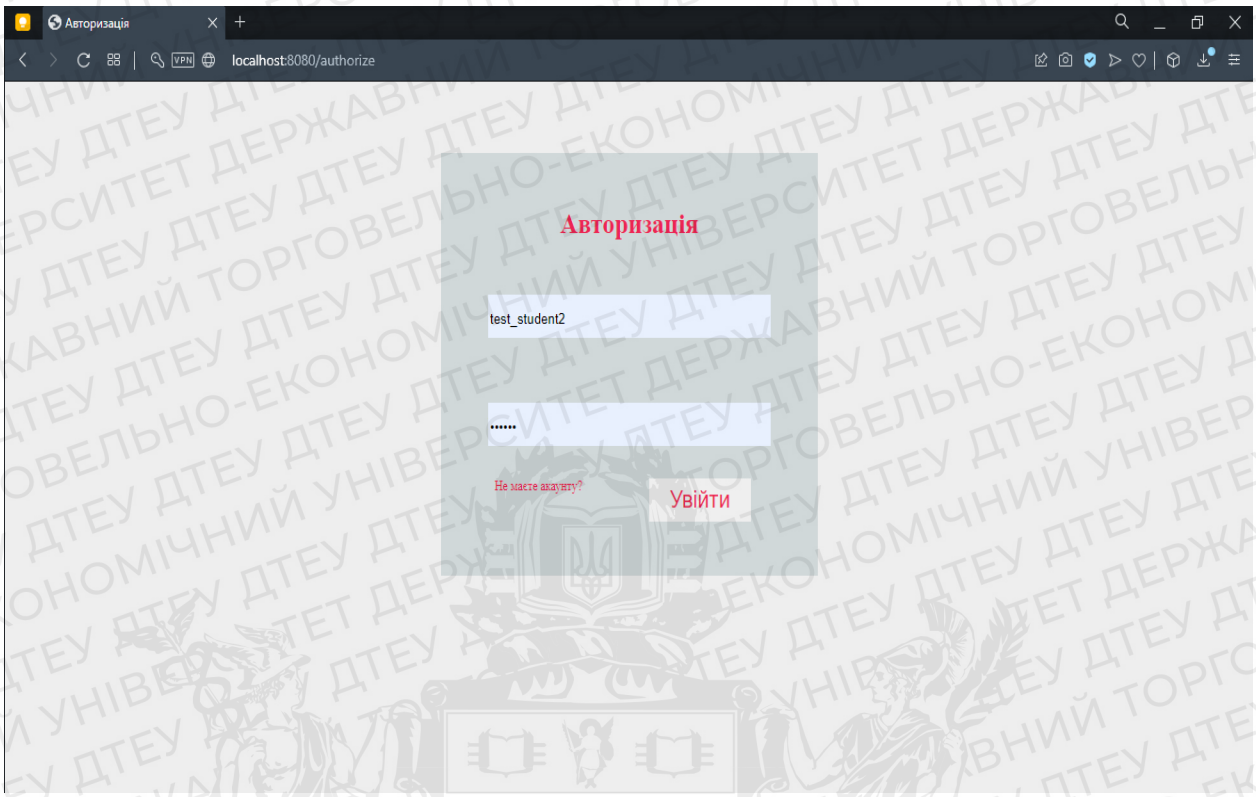

Рис. 3.4. Сторінка авторизації

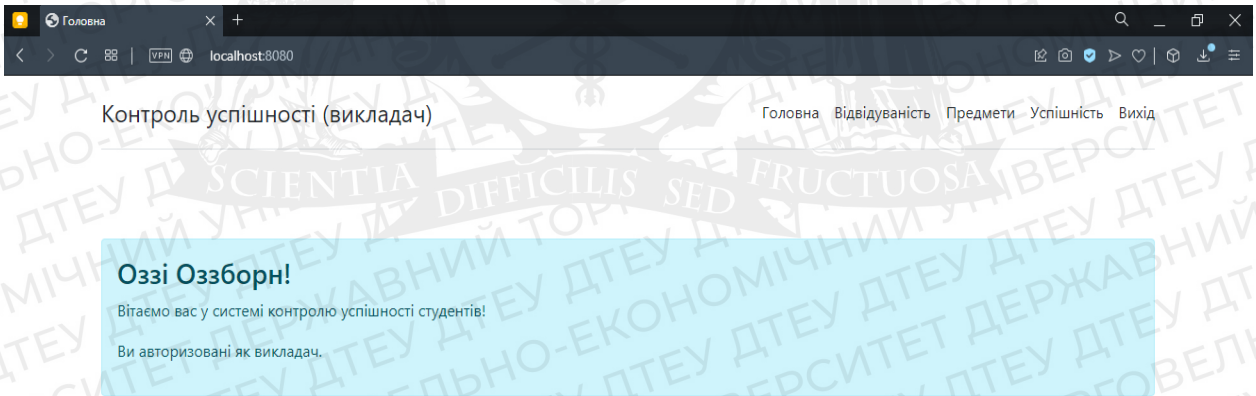

Рис. 3.5. Головна сторінка викладача

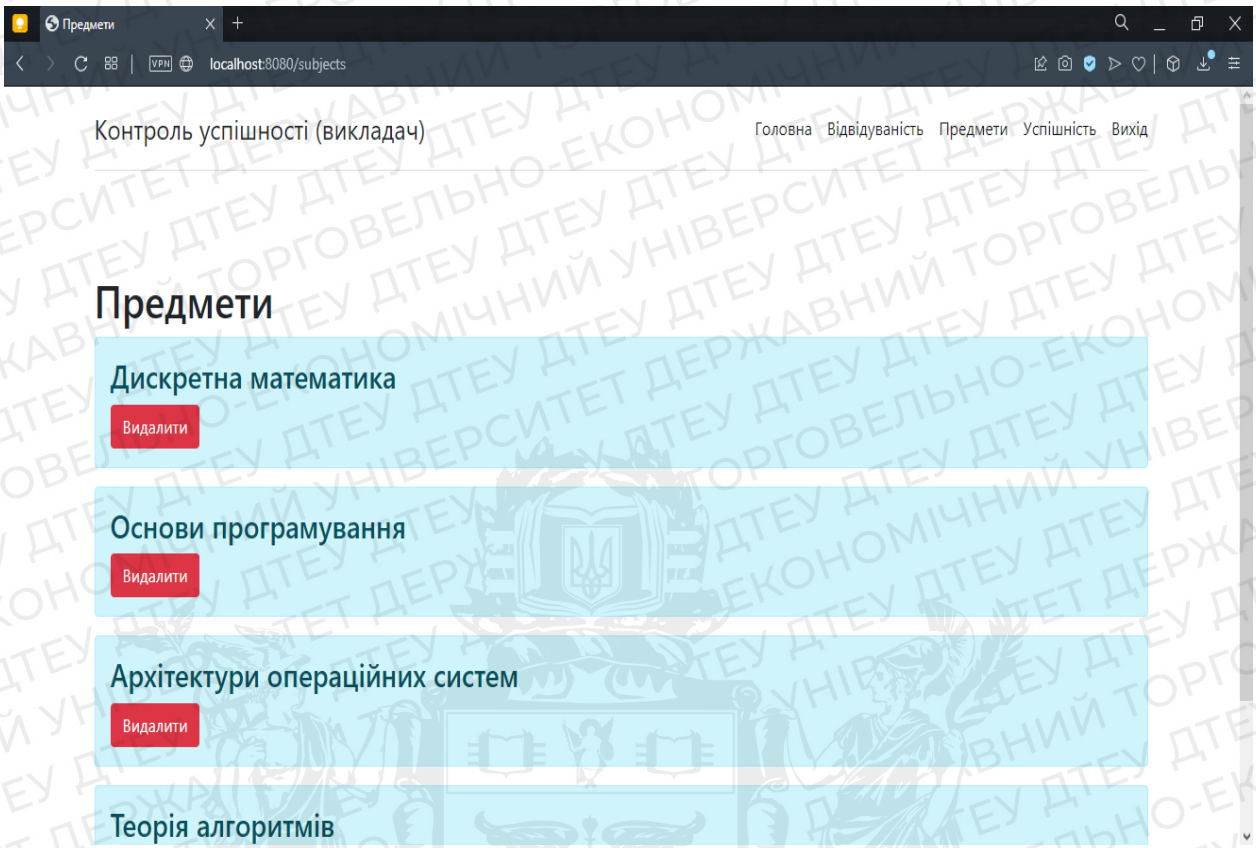

Рис. 3.6. Сторінка предметів

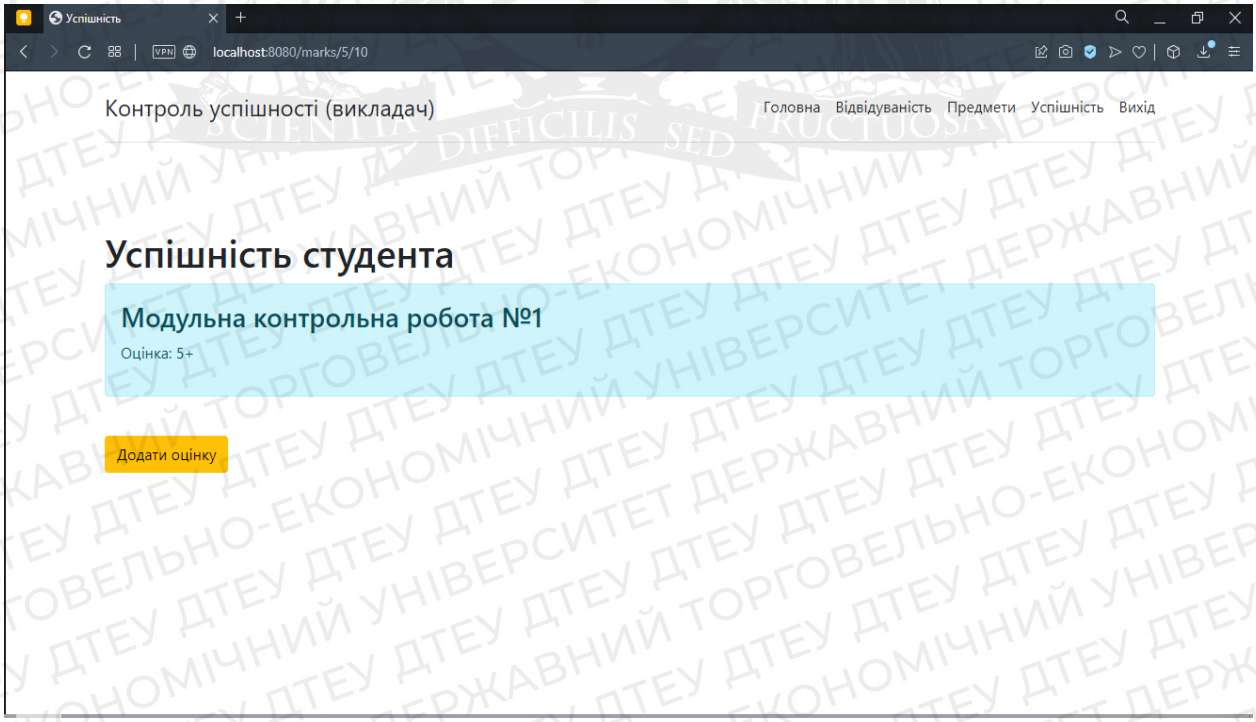

Рис. 3.7. Сторінка успішності студента для викладача

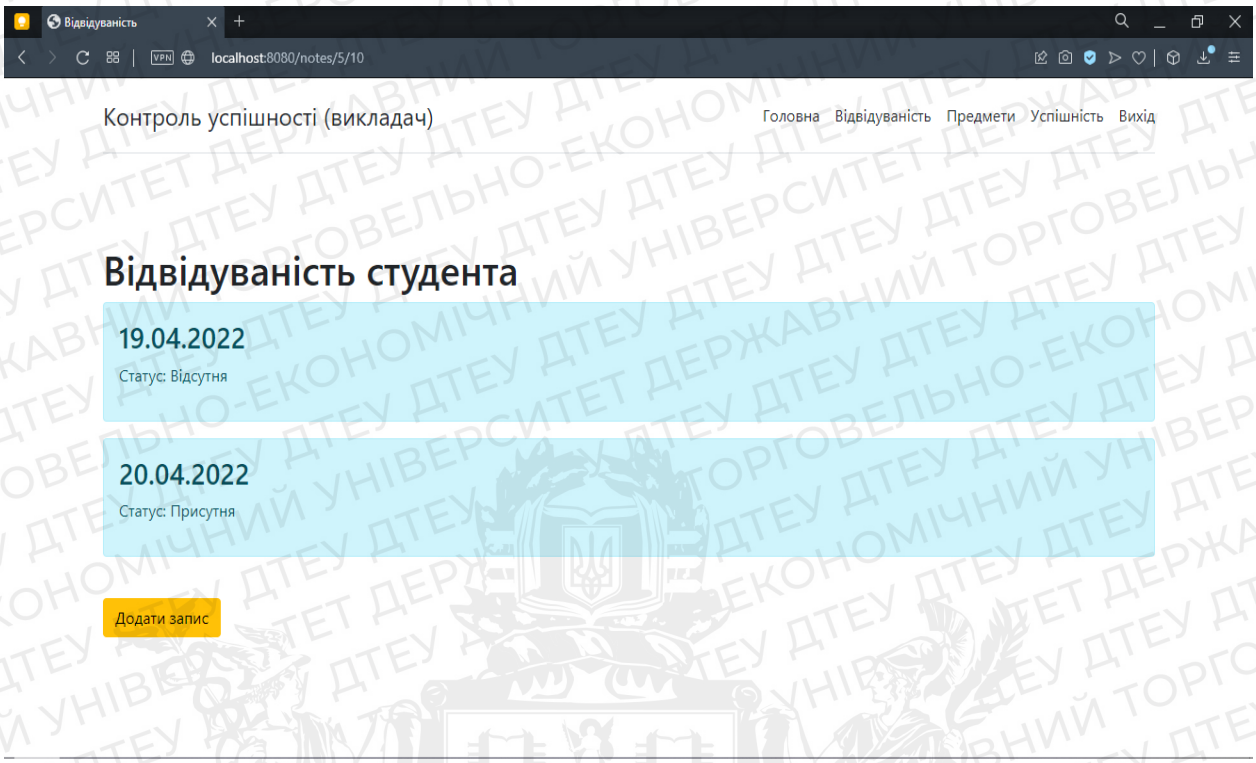

Рис. 3.8. Сторінка відвідуваності для викладача

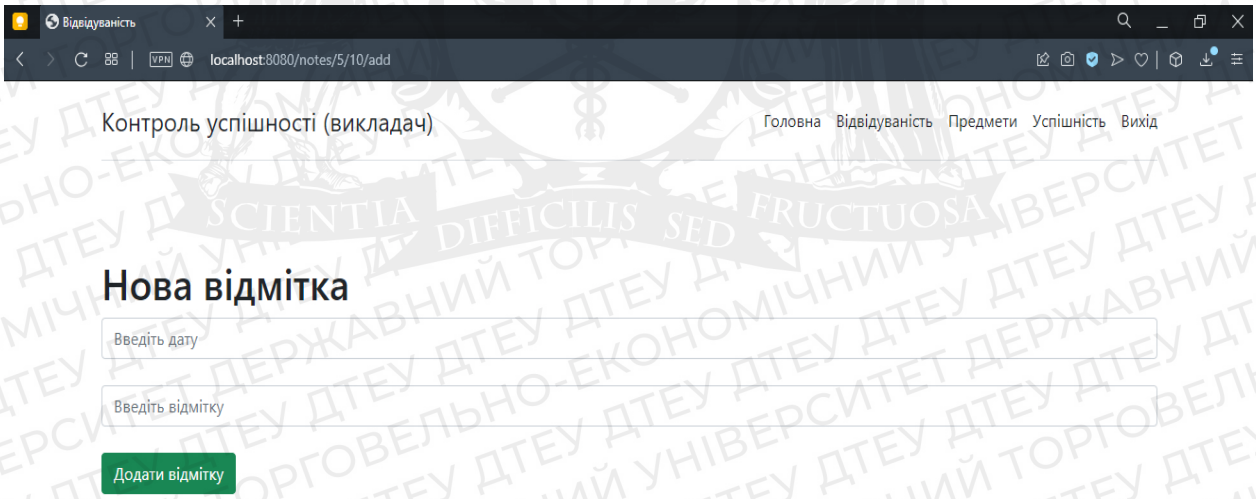

Рис. 3.9. Сторінка редагування даних для викладача

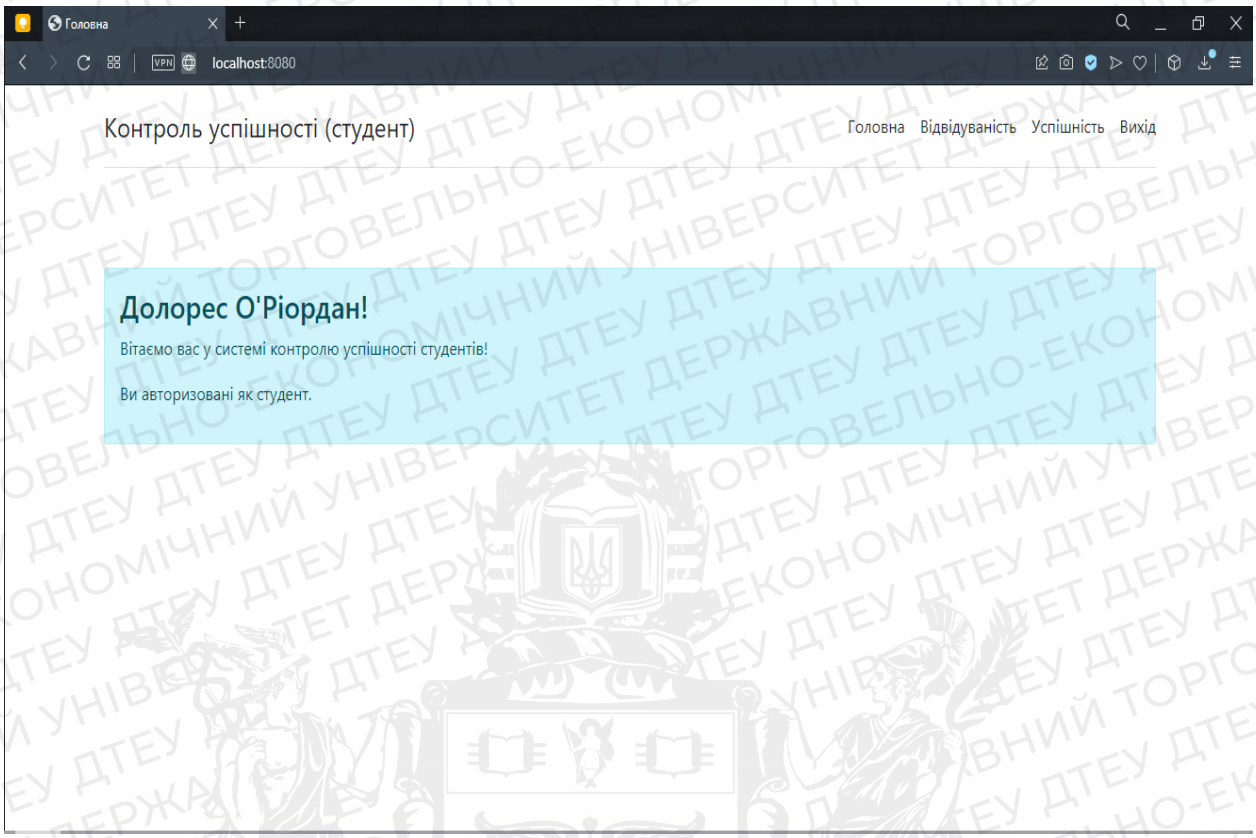

Рис. 3.10. Головна сторінка студента

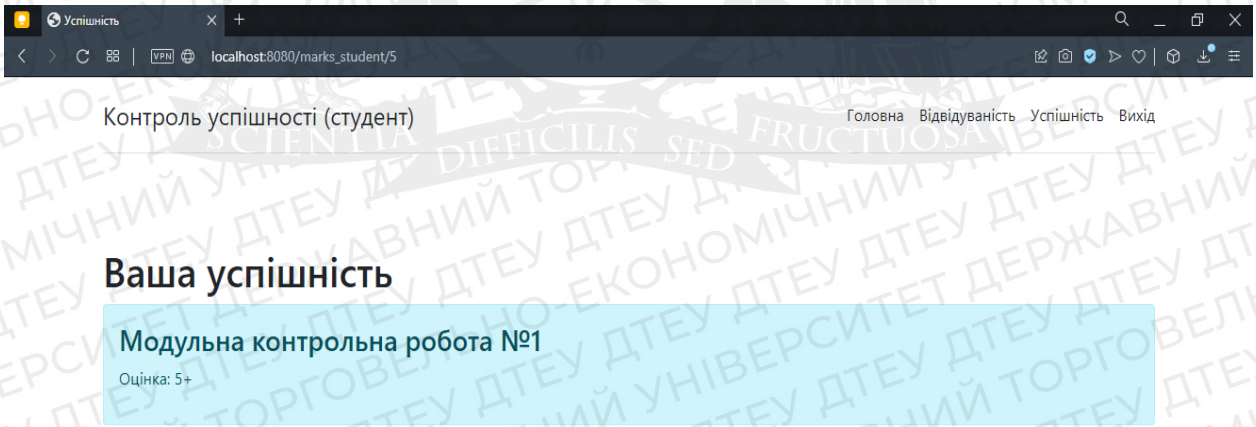

Рис. 3.11. Сторінка успішності студента

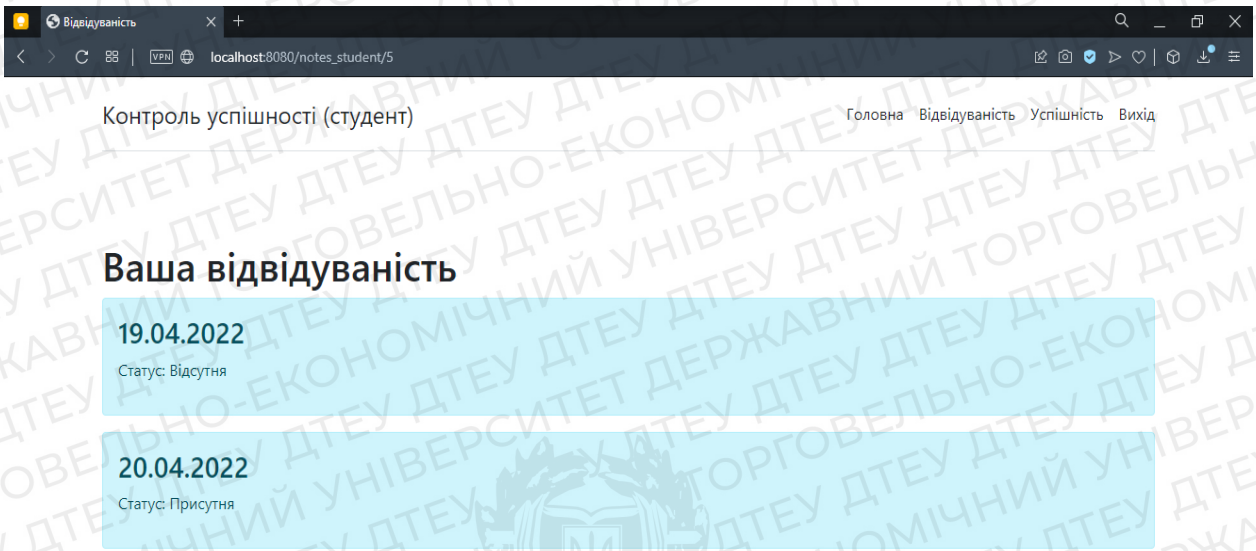

Рис. 3.12. Сторінка відвідуваності студента

3.4. Тестування

Тестування програмного забезпечення є критично важливою частиною процесу розробки. Воно допомагає забезпечити, що програмне забезпечення функціонує правильно, відповідає вимогам і безпечне для кінцевих користувачів. Ось основні аспекти процесу тестування програмного забезпечення:

Основні Елементи Тестування ПЗ:

1. **Аналіз Вимог до Продукту**:
   - Перевірка повноти та коректності вимог.
   - Аналіз з точки зору різних аспектів: бізнесу, індустрії, продуктивності, безпеки тощо.
2. **Перегляд Архітектури та Дизайну**:
   - Співпраця з розробниками для підвищення якості коду.
   - Використання різних методів тестування, включаючи тестування граничних умов.

3. **Виконання Програми**:
    - Перевірка поведінки програми або додатка під час виконання.
4. **Перевірка Інфраструктури Розгортання**:
    - Тестування скриптів розгортання та процесів автоматизації.
5. **Моніторинг та Спостереження**:
    - Використання методів моніторингу для збору даних про продуктивність та стабільність системи.

Важливість Тестування:

- Тестування надає об'єктивну, незалежну інформацію про якість ПЗ.
- Дозволяє оцінити ризики, пов'язані з впровадженням програмного забезпечення.
- Помагає у виявленні помилок, які можуть призвести до несправностей або збоїв.

Помилки та Збої:

- Помилка (або "баг") в коді може призвести до несправностей або збоїв у програмі.
- Не всі помилки призводять до збоїв, але вони можуть стати причиною збоїв при зміні умов використання.
- Одна помилка може мати різноманітні симптоми збоїв.

Тестування - це не просто виявлення помилок; це процес забезпечення, що програмне забезпечення буде функціонувати ефективно і безпечно в усіх передбачуваних сценаріях використання. Тому воно включає в себе широкий спектр діяльності, від перевірки коду до моніторингу системи в реальному часі.

Тестування програмного забезпечення включає різні методики і підходи, кожен з яких має свої переваги та області застосування. Ось детальний опис основних видів тестування:

Статичне, Динамічне та Пасивне Тестування:

1. **Статичне Тестування**:
    - Включає перевірку вимог, дизайну та коду без його виконання.

- Приклади включають рецензування коду, коректуру та використання інструментів для статичного аналізу.

2. **Динамічне Тестування**:
   - Включає виконання програмного коду з метою перевірки його поведінки.
   - Використовуються методи налагодження та тестові сценарії для виявлення дефектів.
3. **Пасивне Тестування**:
   - Зосереджено на аналізі системних журналів та трасувань без втручання в роботу системи.
   - Використовується для пошуку аномалій та встановлення моделей поведінки.

Дослідницький Підхід:

- **Дослідницьке Тестування**:
  - Цей підхід передбачає одночасне навчання, розробку тесту та його виконання.
  - Тестувальник має велику свободу в оцінці та виборі тестових випадків, фокусуючись на оптимізації якості тестування.

"Коробковий" Підхід:

1. **Тестування "Білого Ящика"**:
   - Вимагає знання внутрішньої структури та дизайну програмного забезпечення.
   - Фокусується на внутрішній логіці та структурі коду.
2. **Тестування "Чорного Ящика"**:
   - Тестування зосереджено на функціональності системи без знання її внутрішньої структури.
   - Тести засновані на специфікаціях та вимогах до функціональності.
3. **Тестування "Сірого Ящика"**:
   - Гібридний підхід, який поєднує елементи білого та чорного ящиків.

- Використовується для тестування з урахуванням деяких аспектів внутрішньої структури.

Прогалини у Вимогах:

- Прогалини у вимогах, особливо нефункціональні, такі як тестованість, масштабованість, безпека, можуть призводити до серйозних дефектів у програмному забезпеченні.

Кожен з цих підходів та методик має свої переваги та недоліки, і вибір конкретного підходу залежить від контексту розробки, специфіки програмного продукту та цілей тестування.

Таблиця 4.2

Тестування додатку

№	Тест-кейс	Очікуваний результат	Отриманий результат
1	Невірні данні при авторизації	При введенні невірних даних система повідомляє користувача про те, що такого користувача не існує, або дані введені невірно	При введенні невірних даних система повідомляє користувача про те, що такого користувача не існує, або дані введені невірно
2	Пусті поля при авторизації	При спробі авторизації з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.	При спробі авторизації з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.
3	Неспівпадаючі паролі при реєстрації	При введенні неспівпадаючих паролів система повідомляє користувача про те,	При введенні неспівпадаючих паролів система повідомляє користувача про те,

		що паролі не співпадають.	що паролі не співпадають.
4	Пусті поля при реєстрації	При спробі реєстрації з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.	При спробі реєстрації з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.
5	Створення відмітки з пустими полями	При спробі створення відмітки з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.	При спробі створення відмітки з пустими полями система повідомляє користувача про те, що ці поля необхідно заповнити.
6	Спроба перейти в панель адміністратора без наявності прав адміністратора	При спробі користувача без прав адміністратора перейти в панель адміністрування система повідомляє користувача, шо він не є адміністратором і не може перейти на дану сторінку.	При спробі користувача без прав адміністратора перейти в панель адміністрування система повідомляє користувача, шо він не є адміністратором і не може перейти на дану сторінку.

Результати тестування показують, що система повністю функціональна і готова до використання в реальних умовах.

3.5. Висновки до розділу 3

1. Аналіз варіантів використання підтвердив, що розроблена веб-система може бути використана в широкому спектрі освітніх сценаріїв, від індивідуального навчання до колаборативної роботи в групах. Виявлені ключові сценарії використання допомагають точно відповідати на потреби користувачів та гарантують високу адаптивність системи до різних навчальних контекстів.
2. Проектування внутрішньої будови дозволило створити оптимізовану структуру бази даних, яка відповідає всім необхідним вимогам до зберігання та обробки даних. Запропонована схема бази даних є масштабованою, забезпечує високу швидкість запитів та може бути легко модифікована для розширення функціоналу системи.
3. Розробка графічного інтерфейсу системи виконана з урахуванням сучасних тенденцій UX/UI дизайну, забезпечуючи інтуїтивно зрозуміле та привабливе середовище для кінцевих користувачів. Інтерфейс є гнучким та доступним на різних пристроях, що забезпечує користувачам комфортне та ефективне навчання.
4. Тестування програмного продукту показало високу надійність та стабільність системи. Процес тестування включав різноманітні методи, від юніт-тестування до інтеграційного та приймального тестування. Це дозволило виявити та усунути потенційні проблеми до запуску системи в експлуатацію.

Загальний висновок цього розділу підкреслює, що проектування та розробка програмного продукту були виконані з високим рівнем деталізації та уваги до потреб користувачів. Результати аналізу, проектування, розробки інтерфейсу та тестування вказують на те, що веб-система готова до впровадження та може ефективно підтримувати процес дистанційного навчання.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Розробка системи контролю успішності студентів в якості веб-додатку вимагає дотримання чіткої послідовності кроків та аналізу різних аспектів проектування та розробки. Ось докладний план та рекомендації для кожного етапу:

1. Огляд поняття веб-додатку

- Дослідження основних характеристик веб-додатків.
- Аналіз переваг веб-додатків у контексті системи контролю успішності.

2. Аналіз методів та засобів розробки веб-сайтів

- Оцінка сучасних підходів до розробки веб-додатків.
- Розгляд фреймворків та технологій для фронтенду та бекенду.

3. Огляд поняття контролю навчання

- Дослідження методів контролю успішності в освітньому процесі.
- Аналіз потреб користувачів системи.

4. Огляд існуючих рішень

- Аналіз конкурентів та аналогічних систем.
- Визначення недоліків та переваг існуючих рішень.

5. Вибір мови програмування

- Вибір між популярними мовами, як-от JavaScript (для Node.js), Python, Ruby тощо.
- Оцінка потреб проекту та можливостей мов.

6. Вибір додаткових інструментів

- Визначення бази даних (MySQL, PostgreSQL, MongoDB).
- Вибір інструментів для тестування, версійного контролю тощо.

7. Вибір середовища розробки

- Розгляд IDE, таких як Visual Studio Code, IntelliJ IDEA, PyCharm.
- Вибір засобів для спільної роботи та розгортання коду.

8. Аналіз варіантів використання

- Розробка діаграм варіантів використання UML.
- Визначення основних функціональних та нефункціональних вимог.

9. Проектування внутрішньої будови

- Створення діаграми класів, архітектури системи.
- Вибір архітектурного підходу (наприклад, MVC).

10. Розробка графічного інтерфейсу користувача

- Дизайн користувацького інтерфейсу з урахуванням UX/UI принципів.
- Ітеративна розробка та відгуки від користувачів.

11. Тестування системи

- Впровадження юніт-тестування, інтеграційного та приймального тестування.
- Використання автоматизованих інструментів для тестування.

Пропозиції щодо розвитку

- Розширення функціональності: додавання модулів для відеоконференцій, поліпшення системи безпеки.
- Інтеграція з іншими освітніми платформами або інструментами.
- Впровадження аналітики для збору зворотного зв'язку та покращення системи.

Завдяки послідовному та системному підходу до розробки веб-додатку можна створити ефективну та корисну систему контролю успішності студентів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Танг, Ліам (2020-06-15). JavaScript creator Eich: My take on 20 years of the world's top programming language. ZDNet. URL: https://www.zdnet.com/.
2. Девідсон, Джеймс Дункан; Ковард, Денні (1999-12-17). Java Servlet Specification (Specification) Version: 2.2 Final Release. Сан Майкросистемс. Сс. 43–46. Дата звернення: 27.07.2008.
3. Хоффман, Джей (2019-03-04). What Does AJAX Even Stand For?. Дата звернення: 18.10.2021.
4. Петерсен, Джеремі (2008-09-04). Benefits of using the n-tiered approach for web applications.
5. Top Tips for Secure App Development. Dell.com. Архів оригіналу за 22.05.2012. Дата звернення: 22.06.2012.
6. Множина (wiki). Web application framework. Docforge. Архів оригіналу за 20.06.2020. Дата звернення: 06.03.2010.
7. Множина (wiki). Framework. Docforge. Архів оригіналу за 07.10.2018. Дата звернення: 06.03.2010.
8. What is Web Development? - Definition from Techopedia. Techopedia.com. Дата звернення: 07.12.2018.
9. Кемпбелл, Дженніфер (2017). Web Design: Introductory. Cengage Learning. С. 27.
10. Бюро статистики праці, Міністерство праці США. Information Security Analysts, Web Developers, and Computer Network Architects. Occupational Outlook Handbook, 2012-13 Edition. Дата звернення: 17.01.2013.

11. Tim Berners-Lee. www.w3.org. Дата звернення: 17.11.2021.
12. The website of the world's first-ever web server. Дата звернення: 30.08.2008.
13. Кайльо, Роберт. A Little History of the World Wide Web. Дата звернення: 16.02.2007.
14. Internet, Web, and Other Post-Watergate Concerns. Університет Чикаго. Дата звернення: 18.09.2010.
15. AP Stylebook [@APStylebook] (16.04.2010). Responding to reader input, we are changing Web site to website. This appears on Stylebook Online today and in the 2010 book next month. [Твіт]. Дата звернення: 18.03.2019 – через Twitter.
16. OpenGL ES for the Web. khronos.org. 19.07.2011. Дата звернення: 01.04.2019.
17. ЛеПейдж, Піт. Responsive Web Design Basics | Web. Google Developers. Дата звернення: 13.03.2017.
18. Перрін, Ендрю; Андерсон, Моніка (10.04.2019). Social media usage in the U.S. in 2019 | Pew Research Center. PewResearch.Org. Pew Research. Дата звернення: 20.07.2019. Графіка *Дослідження було цитовано в Forbes. {{cite web}}: Зовнішнє посилання в |quote= (допомога).
19. Web Server Survey. Netcraft. Дата звернення: 13.03.2017.
20. A total number of Websites | Internet live stats. internetlivestats.com. Дата звернення: 14.04.2015.
21. Web Server Survey. Netcraft News. Дата звернення: 17.05.2021.
22. Деон (26.05.2020). How Many Websites Are There Around the World? [2021]. Siteefy. Дата звернення: 17.05.2021.
23. Internet 2009 in numbers. Pingdom. Дата звернення: 17.05.2021.
24. Number of internet users worldwide. Statista. Дата звернення: 17.05.2021.
25. Internet audiences worldwide 2020. Statista. Дата звернення: 17.05.2021.
26. Facebook MAU worldwide 2020. Statista. Дата звернення: 17.05.2021.
27. JAVASOFT SHIPS JAVA 1.0. Архів оригіналу за 10.03.2007. Дата звернення: 13.05.2018.

28. Object-oriented Programming with Java: Essentials and Applications. Tata McGraw-Hill Education. С. 34.
29. JSG – Java Study Group. open-std.org. Архів оригіналу за 25.08.2006. Дата звернення: 02.08.2006.
30. Why Java™ Was – Not – Standardized Twice [PDF]. Архів (PDF) оригіналу за 13.01.2014. Дата звернення: 03.06.2018.
31. What is ECMA—and why Microsoft cares. ZDNet. Архів оригіналу за 06.05.2014. Дата звернення: 06.05.2014.
32. Java Community Process website. Jcp.org. 24.05.2010. Архів оригіналу за 08.08.2006. Дата звернення: 09.06.2010.

ДОДАТКИ

Додаток А

Додаток А «Лістинг програмного коду»

```
package com.joreijarr.studycontrol.controllers;

import com.joreijarr.studycontrol.models.Marks;
import com.joreijarr.studycontrol.models.Notes;
import com.joreijarr.studycontrol.models.Subjects;
import com.joreijarr.studycontrol.models.Users;
import com.joreijarr.studycontrol.repo.MarksRepository;
import com.joreijarr.studycontrol.repo.NotesRepository;
import com.joreijarr.studycontrol.repo.SubjectRepository;
import com.joreijarr.studycontrol.repo.UsersRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.ArrayList;
import java.util.List;

@Controller
public class startController {


    public Boolean authorized = false;
    public Users current_user;
    @Autowired
    public UsersRepository usersRepository;

    @Autowired
```

```
    public SubjectRepository subjectRepository;

    @Autowired
    public MarksRepository marksRepository;

    @Autowired
    public NotesRepository notesRepository;


    @GetMapping("/exit")
    public String exit(Model model)
    {
        current_user = null;
        authorized = false;
        return "redirect:/";
    }


    @GetMapping("/")
    public String index(Model model)
    {
        model.addAttribute("title", "Головна");
        if(authorized)
        {
            model.addAttribute("fullname",
current_user.getUser_fullname());
            if(current_user.getUser_status().equals("user"))
            {
                return "index_user";
            }
            else
            {
                return "index_student";
            }
        }
        else
        {
            return "redirect:/authorize";
        }
```

```
    }

    @GetMapping("/authorize")
    public String start(Model model) {

        model.addAttribute("title", "Авторизація");
        if(!authorized)
        {
            return "authorize";
        }
        else
        {
            return "redirect:/";
        }
    }

    @GetMapping("/subjects")
    public String subjects(Model model) {

        model.addAttribute("title", "Предмети");
        if(authorized                                    &&
current_user.getUser_status().equals("user"))
        {
            Iterable<Subjects>          subjects_it          =
subjectRepository.findAll();
            List<Subjects> subjects = new ArrayList<>();
            subjects_it.forEach(subjects::add);
            List<Subjects> fnl = new ArrayList<>();
            for(int i = 0; i < subjects.size(); i++)
            {

if(Integer.parseInt(subjects.get(i).getTeacher_id())                ==
Math.toIntExact(current_user.getUser_id()))
                {
                    fnl.add(subjects.get(i));
                }
            }
```

```
            model.addAttribute("subjects", fnl);
            return "subjects_user";
            }
            else
            {
            return "redirect:/";
            }
        }

        @GetMapping("/subjects/add")
        public String subjectsAdd(Model model) {

            model.addAttribute("title", "Новий предмет");
            if(authorized                                    &&
current_user.getUser_status().equals("user"))
            {
            return "subjects_add";
            }
            else
            {
            return "redirect:/";
            }
        }


        @GetMapping("/marks")
        public String marks_list(Model model) {

            model.addAttribute("title", "Успішність");
            if(authorized                                    &&
current_user.getUser_status().equals("user"))
            {
                Iterable<Subjects>          subjects_it          =
subjectRepository.findAll();
                List<Subjects> subjects = new ArrayList<>();
                subjects_it.forEach(subjects::add);
                List<Subjects> fnl = new ArrayList<>();
                for(int i = 0; i < subjects.size(); i++)
                {
```

```
if(Integer.parseInt(subjects.get(i).getTeacher_id())                    ==
Math.toIntExact(current_user.getUser_id()))
                    {
                        fnl.add(subjects.get(i));
                    }
                }
                model.addAttribute("subjects", fnl);
                return "marks_list_user";
            }
            else
            {
                return "redirect:/";
            }
        }

        @GetMapping("/marks_student")
        public String marks_list_student(Model model) {

            model.addAttribute("title", "Успішність");
            if(authorized                                                   &&
current_user.getUser_status().equals("student"))
            {
                Iterable<Subjects>          subjects_it                 =
subjectRepository.findAll();
                List<Subjects> subjects = new ArrayList<>();
                subjects_it.forEach(subjects::add);
                List<Subjects> fnl = new ArrayList<>();
                for(int i = 0; i < subjects.size(); i++)
                {

if(subjects.get(i).getSubject_group().equals(current_user.getUser_grou
p()))
                    {
                        fnl.add(subjects.get(i));
                    }
                }
                model.addAttribute("subjects", fnl);
                return "marks_list_student";
```

```
            }
            else
            {
                return "redirect:/";
            }
        }


        @GetMapping("/notes")
        public String notes_list(Model model) {

            model.addAttribute("title", "Відвідуваність");
            if(authorized                                                    &&
current_user.getUser_status().equals("user"))
            {
                Iterable<Subjects>          subjects_it          =
subjectRepository.findAll();
                List<Subjects> subjects = new ArrayList<>();
                subjects_it.forEach(subjects::add);
                List<Subjects> fnl = new ArrayList<>();
                for(int i = 0; i < subjects.size(); i++)
                {

if(Integer.parseInt(subjects.get(i).getTeacher_id())                    ==
Math.toIntExact(current_user.getUser_id()))
                    {
                        fnl.add(subjects.get(i));
                    }
                }
                model.addAttribute("subjects", fnl);
                return "notes_list_user";
            }
            else
            {
                return "redirect:/";
            }
        }
```

```
        @GetMapping("/notes_student")
        public String notes_list_student(Model model) {

            model.addAttribute("title", "Відвідуваність");
            if(authorized                                                    &&
current_user.getUser_status().equals("student"))
            {
                Iterable<Subjects>          subjects_it          =
subjectRepository.findAll();
                List<Subjects> subjects = new ArrayList<>();
                subjects_it.forEach(subjects::add);
                List<Subjects> fnl = new ArrayList<>();
                for(int i = 0; i < subjects.size(); i++)
                {

if(subjects.get(i).getSubject_group().equals(current_user.getUser_grou
p()))
                    {
                        fnl.add(subjects.get(i));
                    }
                }
                model.addAttribute("subjects", fnl);
                return "notes_list_student";
            }
            else
            {
                return "redirect:/";
            }
        }

        @GetMapping("/marks/{id}")
        public String marks_subject(Model model, @PathVariable(name =
"id") Long id) {

            model.addAttribute("title", "Успішність");
            if(authorized                                                    &&
current_user.getUser_status().equals("user"))
            {
```

```
                Iterable<Subjects>          subjects_it          =
subjectRepository.findAll();
                List<Subjects> subjects = new ArrayList<>();
                subjects_it.forEach(subjects::add);
                String pair_group = "";
                for(int i = 0; i < subjects.size(); i++)
                {
                    if(subjects.get(i).getSubject_id() == id)
                    {
                        pair_group                                    =
subjects.get(i).getSubject_group();
                        break;
                    }
                }
                Iterable<Users> users_it = usersRepository.findAll();
                List<Users> users = new ArrayList<>();
                users_it.forEach(users::add);
                List<Users> final_users = new ArrayList<>();
                for(int i = 0; i < users.size(); i++)
                {

if(users.get(i).getUser_status().equals("student") &&
                    users.get(i).getUser_group().equals(pair_group))
                    {
                        final_users.add(users.get(i));
                    }
                }
                model.addAttribute("users", final_users);
                model.addAttribute("subject_id", id);
                return "marks_subject_user";
            }
            else
            {
                return "redirect:/";
            }
        }


        @GetMapping("/marks_student/{id}")
```

```
        public String marks_subject_student(Model model,
@PathVariable(name = "id") Long id) {

            model.addAttribute("title", "Успішність");
            if(authorized &&
current_user.getUser_status().equals("student"))
            {
                Iterable<Marks> marks_it = marksRepository.findAll();
                List<Marks> marks = new ArrayList<>();
                marks_it.forEach(marks::add);
                List<Marks> final_marks = new ArrayList<>();
                for(int i = 0; i < marks.size(); i++)
                {
                    if(marks.get(i).getSubject_id() == id &&
                            marks.get(i).getUser_id() ==
current_user.getUser_id())
                    {
                        final_marks.add(marks.get(i));
                    }
                }
                model.addAttribute("marks", final_marks);
                return "marks_subject_student";
            }
            else
            {
                return "redirect:/";
            }
        }


        @GetMapping("/notes/{id}")
        public String notes_subject(Model model, @PathVariable(name =
"id") Long id) {

            model.addAttribute("title", "Відвідуваність");
            if(authorized &&
current_user.getUser_status().equals("user"))
            {
```

```
                Iterable<Subjects>          subjects_it          =
subjectRepository.findAll();
                List<Subjects> subjects = new ArrayList<>();
                subjects_it.forEach(subjects::add);
                String pair_group = "";
                for(int i = 0; i < subjects.size(); i++)
                {
                    if(subjects.get(i).getSubject_id() == id)
                    {
                        pair_group                                    =
subjects.get(i).getSubject_group();
                        break;
                    }
                }
                Iterable<Users> users_it = usersRepository.findAll();
                List<Users> users = new ArrayList<>();
                users_it.forEach(users::add);
                List<Users> final_users = new ArrayList<>();
                for(int i = 0; i < users.size(); i++)
                {

if(users.get(i).getUser_status().equals("student") &&

users.get(i).getUser_group().equals(pair_group))
                    {
                        final_users.add(users.get(i));
                    }
                }
                model.addAttribute("users", final_users);
                model.addAttribute("subject_id", id);
                return "notes_subject_user";
            }
            else
            {
                return "redirect:/";
            }
        }

        @GetMapping("/notes_student/{id}")
```

```
        public String notes_subject_student(Model model,
@PathVariable(name = "id") Long id) {

            model.addAttribute("title", "Відвідуваність");
            if(authorized &&
current_user.getUser_status().equals("student"))
            {
                Iterable<Notes> notes_it = notesRepository.findAll();
                List<Notes> notes = new ArrayList<>();
                notes_it.forEach(notes::add);
                List<Notes> final_notes = new ArrayList<>();
                for(int i = 0; i < notes.size(); i++)
                {
                    if(notes.get(i).getSubject() == id &&
                            notes.get(i).getStudent() ==
current_user.getUser_id())
                    {
                        final_notes.add(notes.get(i));
                    }
                }
                model.addAttribute("notes", final_notes);
                return "notes_subject_student";
            }
            else
            {
                return "redirect:/";
            }
        }


        @GetMapping("/marks/{subject_id}/{user_id}")
        public String marks_student_subject(Model model,
@PathVariable(name = "subject_id") Long subject_id,
                                            @PathVariable(name = "user_id")
Long user_id) {

            model.addAttribute("title", "Успішність");
            if(authorized &&
current_user.getUser_status().equals("user"))
```

```
        {
            Iterable<Marks> marks_it = marksRepository.findAll();
            List<Marks> marks = new ArrayList<>();
            marks_it.forEach(marks::add);
            List<Marks> final_marks = new ArrayList<>();
            for(int i = 0; i < marks.size(); i++)
            {
                if(marks.get(i).getSubject_id() == subject_id &&
                marks.get(i).getUser_id() == user_id)
                {
                    final_marks.add(marks.get(i));
                }
            }
            model.addAttribute("marks", final_marks);
            model.addAttribute("subject_id", subject_id);
            model.addAttribute("user_id", user_id);
            return "marks_subject_student_user";
        }
        else
        {
            return "redirect:/";
        }
    }


    @GetMapping("/notes/{subject_id}/{user_id}")
    public String notes_student_subject(Model model,
@PathVariable(name = "subject_id") Long subject_id,
                                        @PathVariable(name =
"user_id") Long user_id) {

        model.addAttribute("title", "Відвідуваність");
        if(authorized &&
current_user.getUser_status().equals("user"))
        {
            Iterable<Notes> notes_it = notesRepository.findAll();
            List<Notes> notes = new ArrayList<>();
            notes_it.forEach(notes::add);
            List<Notes> final_notes = new ArrayList<>();
```

```
            for(int i = 0; i < notes.size(); i++)
            {
                if(notes.get(i).getSubject() == subject_id &&
                        notes.get(i).getStudent() == user_id)
                {
                    final_notes.add(notes.get(i));
                }
            }
            model.addAttribute("notes", final_notes);
            model.addAttribute("subject_id", subject_id);
            model.addAttribute("user_id", user_id);
            return "notes_subject_student_user";
        }
        else
        {
            return "redirect:/";
        }
    }


    @GetMapping("/marks/{subject_id}/{user_id}/add")
    public    String    marks_student_subject_add(Model    model,
@PathVariable(name = "subject_id") Long subject_id,
                                    @PathVariable(name   =  "user_id")
Long user_id) {

        model.addAttribute("title", "Успішність");
        if(authorized                                                    &&
current_user.getUser_status().equals("user"))
        {
            model.addAttribute("subject_id", subject_id);
            model.addAttribute("user_id", user_id);
            return "marks_subject_user_add";
        }
        else
        {
            return "redirect:/";
        }
    }
```

```
        @GetMapping("/notes/{subject_id}/{user_id}/add")
        public    String    notes_student_subject_add(Model    model,
@PathVariable(name = "subject_id") Long subject_id,
                                                    @PathVariable(name   =
"user_id") Long user_id) {

            model.addAttribute("title", "Відвідуваність");
            if(authorized                                                &&
current_user.getUser_status().equals("user"))
            {
                model.addAttribute("subject_id", subject_id);
                model.addAttribute("user_id", user_id);
                return "notes_subject_user_add";
            }
            else
            {
                return "redirect:/";
            }
        }


        @PostMapping("/notes/{subject_id}/{user_id}/add")
        public String marks_student_subject_add_post(Model model,
                                                    @RequestParam
String note_status,

@PathVariable(name = "subject_id") Long subject_id,

@PathVariable(name = "user_id") Long user_id) {
            Notes  notes  =  new  Notes(Math.toIntExact(subject_id),
Math.toIntExact(user_id), note_status);
            notesRepository.save(notes);
            return "redirect:/notes/" + subject_id + "/" + user_id;
        }
```

```
        @PostMapping("/marks/{subject_id}/{user_id}/add")
        public String notes_student_subject_add_post(Model model,
@RequestParam String mark,
                                                     @RequestParam
String additional_description,

@PathVariable(name = "subject_id") Long subject_id,

@PathVariable(name = "user_id") Long user_id) {
            Marks marks = new Marks(Math.toIntExact(subject_id),
Math.toIntExact(user_id), mark, additional_description);
            marksRepository.save(marks);
            return "redirect:/marks/" + subject_id + "/" + user_id;
        }



        @PostMapping("/subjects/add")
        public String subjectsAdd_post(Model model, @RequestParam
String subject_name,
                                       @RequestParam String group,
                                       @RequestParam String date,
                                       @RequestParam String pair,
                                       @RequestParam String comment) {
            Subjects subject = new Subjects(subject_name, group, date,
pair,
                    String.valueOf(current_user.getUser_id()),
                    comment);
            subjectRepository.save(subject);
            return "redirect:/subjects";
        }

        @PostMapping("/subjects/{id}/delete")
        public String subjectsDelete(Model model, @PathVariable (value
= "id") long subject_id) {
            Subjects                    subject                =
subjectRepository.findById(subject_id).orElseThrow();
            subjectRepository.delete(subject);
            return "redirect:/subjects";
```

```
    }


    @PostMapping("/authorize")
    public String authorize(Model model, @RequestParam String
user_login, @RequestParam String user_password)
    {
        Iterable<Users> users_it = usersRepository.findAll();
        List<Users> users = new ArrayList<>();
        users_it.forEach(users::add);
        for (int i = 0; i < users.size(); i++)
        {
            if(users.get(i).getUser_login().equals(user_login) &&
users.get(i).getUser_password().equals(user_password))
            {
                authorized = true;
                current_user = users.get(i);
                break;
            }
        }
        return "redirect:/";
    }


    @GetMapping("/registration")
    public String registration(Model model)
    {
        model.addAttribute("title", "Реєстрація");
        if(!authorized)
        {
            return "registration";
        }
        else
        {
            return "redirect:/";
        }
    }
```

```
    @PostMapping("/registration")
    public String registration(Model model, @RequestParam String
user_login, @RequestParam String user_password,
                               @RequestParam              String
user_fullname, @RequestParam String user_status,
                               @RequestParam String group)
    {
        Users new_user = new Users(user_login, user_password,
                user_fullname, user_status, group);
        usersRepository.save(new_user);
        return "redirect:/authorize";
    }

}
package com.joreijarr.studycontrol.repo;

import com.joreijarr.studycontrol.models.Notes;
import org.springframework.data.repository.CrudRepository;

public interface NotesRepository extends CrudRepository<Notes,
Long> {
}
package com.joreijarr.studycontrol.repo;

import com.joreijarr.studycontrol.models.Marks;
import org.springframework.data.repository.CrudRepository;

public interface MarksRepository extends CrudRepository<Marks,
Long> {
}
package com.joreijarr.studycontrol.repo;

import com.joreijarr.studycontrol.models.Users;
import org.springframework.data.repository.CrudRepository;

public interface UsersRepository extends CrudRepository<Users,
Long> {
}
package com.joreijarr.studycontrol.repo;
```

```
import com.joreijarr.studycontrol.models.Subjects;
import com.joreijarr.studycontrol.models.Users;
import org.springframework.data.repository.CrudRepository;

public interface SubjectRepository extends
CrudRepository<Subjects, Long> {
}
package com.joreijarr.studycontrol.models;

import javax.persistence.*;

@Entity
public class Notes {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long note_id;

    private int subject_id;

    private int student_id;

    private String note_status;

    public Notes() {
    }

    public Notes(int subject, int student, String note_status) {
        this.subject_id = subject;
        this.student_id = student;
        this.note_status = note_status;
    }



    public Long getNote_id() {
        return note_id;
    }
```

```
    public void setNote_id(Long note_id) {
        this.note_id = note_id;
    }

    public int getSubject() {
        return subject_id;
    }

    public void setSubject(int subject) {
        this.subject_id = subject;
    }

    public int getStudent() {
        return student_id;
    }

    public void setStudent(int student) {
        this.student_id = student;
    }

    public String getNote_status() {
        return note_status;
    }

    public void setNote_status(String note_status) {
        this.note_status = note_status;
    }
}
package com.joreijarr.studycontrol.models;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Marks {
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long mark_id;
    private int subject_id;
    private int user_id;
    private String mark;
    private String additional_description;

    public Marks() {
    }

    public Marks(int subject_id, int user_id, String mark, String
additional_description) {
        this.subject_id = subject_id;
        this.user_id = user_id;
        this.mark = mark;
        this.additional_description = additional_description;
    }

    public Long getMark_id() {
        return mark_id;
    }

    public void setMark_id(Long mark_id) {
        this.mark_id = mark_id;
    }

    public int getSubject_id() {
        return subject_id;
    }

    public void setSubject_id(int subject_id) {
        this.subject_id = subject_id;
    }

    public int getUser_id() {
        return user_id;
    }

    public void setUser_id(int user_id) {
```

```
        this.user_id = user_id;
    }

    public String getMark() {
        return mark;
    }

    public void setMark(String mark) {
        this.mark = mark;
    }

    public String getAdditional_description() {
        return additional_description;
    }

    public void setAdditional_description(String
additional_description) {
        this.additional_description = additional_description;
    }
}
package com.joreijarr.studycontrol.models;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Users {

    public Users() {
    }

    public Users(String user_login, String user_password, String
user_fullname, String user_status, String user_group) {
        this.user_login = user_login;
        this.user_password = user_password;
        this.user_fullname = user_fullname;
        this.user_status = user_status;
```

```
        this.user_group = user_group;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long user_id;

    private String user_login;
    private String user_password;
    private String user_fullname;
    private String user_status;

    public String getUser_group() {
        return user_group;
    }

    public void setUser_group(String user_group) {
        this.user_group = user_group;
    }

    private String user_group;

    public Long getUser_id() {
        return user_id;
    }

    public void setUser_id(Long user_id) {
        this.user_id = user_id;
    }

    public String getUser_login() {
        return user_login;
    }

    public void setUser_login(String user_login) {
        this.user_login = user_login;
    }

    public String getUser_password() {
```

```
        return user_password;
    }

    public void setUser_password(String user_password) {
        this.user_password = user_password;
    }

    public String getUser_fullname() {
        return user_fullname;
    }

    public void setUser_fullname(String user_fullname) {
        this.user_fullname = user_fullname;
    }

    public String getUser_status() {
        return user_status;
    }

    public void setUser_status(String user_status) {
        this.user_status = user_status;
    }
}
```

```
package com.joreijarr.studycontrol.models;


import javax.persistence.*;

@Entity
public class Subjects {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long subject_id;
    private String subject_name, subject_group, subject_date,
subject_pair, teacher_id, teacher_comment;

    public Subjects() {
    }
```

```
    public Subjects(String subject_name, String subject_group,
String subject_date, String subject_pair, String teacher_id, String
teacher_comment) {
        this.subject_name = subject_name;
        this.subject_group = subject_group;
        this.subject_date = subject_date;
        this.subject_pair = subject_pair;
        this.teacher_id = teacher_id;
        this.teacher_comment = teacher_comment;
    }

    public Long getSubject_id() {
        return subject_id;
    }

    public void setSubject_id(Long subject_id) {
        this.subject_id = subject_id;
    }

    public String getSubject_name() {
        return subject_name;
    }

    public void setSubject_name(String subject_name) {
        this.subject_name = subject_name;
    }

    public String getSubject_group() {
        return subject_group;
    }

    public void setSubject_group(String subject_group) {
        this.subject_group = subject_group;
    }

    public String getSubject_date() {
        return subject_date;
    }
```

```
    public void setSubject_date(String subject_date) {
        this.subject_date = subject_date;
    }

    public String getSubject_pair() {
        return subject_pair;
    }

    public void setSubject_pair(String subject_pair) {
        this.subject_pair = subject_pair;
    }

    public String getTeacher_id() {
        return teacher_id;
    }

    public void setTeacher_id(String teacher_id) {
        this.teacher_id = teacher_id;
    }

    public String getTeacher_comment() {
        return teacher_comment;
    }

    public void setTeacher_comment(String teacher_comment) {
        this.teacher_comment = teacher_comment;
    }
}
```