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

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО

«_____»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСНИКА ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ

«ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ»

Тема: «Веб-застосунок блогу на базі Spring Boot»

Виконавець: Ілля ЛАВРИНЮК

Керівник: ст. викл. Наталка РИБАСОВА

Нормоконтролер: к.т.н. доц. Олена ТОЛСТІКОВА

КИЇВ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*
Кафедра *комп'ютерних інформаційних технологій*
Спеціальність *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ:
завідувач кафедри КІТ
_____ Аліна САВЧЕНКО
(підпис)
« » 2023 р.

ЗАВДАННЯ
на виконання дипломного проекту
Лавринюка Ілля Миколайовича
(ПІБ випускника)

1. Тема роботи: «Веб-застосунок блогу на базі Spring Boot» затверджена наказом ректора № 623/ст від 01.05.2023р.
2. Термін виконання роботи: з 15 травня 2023 року по 22 червня 2023 року.
3. Вихідні дані до роботи: веб-застосунок блогу на базі Spring Boot
4. Зміст пояснювальної записки: 1. Огляд сучасних технологій та методів створення веб-застосунків. 2. Особливості створення веб-застосунків типу «блог» та аналіз обраних інструментів. 3. Створення веб-застосунку блогу на базі Spring Boot
5. Перелік обов'язкового ілюстративного матеріалу: 1. Огляд фронтенд та бекенд технологій. 2. Огляд інтегрованих середовищ розробки. 3. Огляд методологій розробки ПЗ. 4. Огляд методів тестування веб-застосунку. 5. Огляд видів веб-застосунків за призначенням. 5. Огляд особливостей створення веб-застосунків блогу. 6. Створення веб-застосунку блогу на базі Spring Boot. 7. Вигляд створеного веб-застосунку.

6. Календарний план-графік

№ п/п	Етапи виконання роботи	Термін виконання етапів	Примітки
1	Огляд та аналіз предметної області, написання 1 розділу, представлення керівнику	15.05.2023 – 16.05.2023	
2	Вибір та опис обраних технологій, написання 2 розділу, представлення керівнику	17.05.2023 – 21.05.2023	
3	Створення веб-застосунку блогу, написання 3 розділу, представлення керівнику	22.05.2023 – 28.05.2023	
4	Оформлення, редагування пояснювальної записки та друк	29.05.2023 – 04.06.2023	
5	Створення та оформлення презентації, запис пояснювальної записки на диск	05.06.2023 – 11.06.2023	
6	Проходження нормоконтролю, перепліт пояснювальної записки	12.06.2023 – 16.06.2023	
7	Створення тексту доповіді, підготовка до захисту кваліфікаційної роботи	16.06.2023 – 18.06.2023	

7. Дата видачі завдання: 15.05.2023р.

Керівник кваліфікаційної роботи _____ Наталка РИБАСОВА
(підпис керівника)

Завдання прийняв до виконання _____ Ілля ЛАВРИНЮК
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Веб-застосунок блогу на базі Spring Boot» містить: 65 сторінок, 76 рисунків, 11 інформаційних джерел.

Об’єкт дослідження – веб-застосунок блог.

Предмет дослідження – створення веб-застосунку блогу на базі Spring Boot.

Мета кваліфікаційної роботи – дослідити сучасні методи та технології створення веб-застосунків виду «блог», проаналізувати переваги та недоліки впроваджених в кваліфікаційній роботі інструментів та підходів.

Основна мета поділена на допоміжні завдання: огляд сучасних технологій та методів створення веб-аплікейшинів, особливості створення веб-застосунків типу «блог» та аналіз обраних інструментів, створення веб-застосунку блогу на базі Spring Boot.

Методи дослідження – логічний, синтезу, аналізу, порівняльний, обробка літературних джерел.

ВЕБ-ЗАСТОСУНОК, HTML, CSS, JAVA, HIBERNATE, DAO, SPRING BOOT, URL, 3-TH ARCHITECTURE, CLEAN CODE, OPTIMIZATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

HTML	–	HyperText Markup Language
CSS	–	Cascading Style Sheets
IDE	–	Integrated Development Enviroment
ORM	–	Object Relational Mapping
DAO	–	Data Access Object
URL	–	Uniform Resource Locator
CDN	–	Content Delivery Network
SEO	–	Search Engine Optimization

ВСТУП

Веб-аплікейшини стали невід'ємною частиною 21 сторіччя, надаючи нам зручний спосіб спілкування, обміну інформацією та отримання нових знань. Одним з найпопулярніших видів веб-застосунків є блоги, які дозволяють нам ділитися своїми думками, ідеями та досвідом з іншими людьми по всьому світу, де б не перебували, як читачі, так й автори.

В кваліфікаційній роботі уде розглянуто веб-застосунок блогу, побудований на базі Spring Boot. Spring Boot - це фреймворк розробки веб-додатків для мови програмування Java, який надає швидкий та простий спосіб створення потужних веб-застосунків.

Створення веб-застосунку блогу на базі Spring Boot має багато переваг. Перш за все, він забезпечує потужну платформу для розробки, що дозволяє швидко створювати функціональність, таку як управління користувачами, реєстрація, авторизація, додавання та відображення постів блогу.

Крім того, Spring Boot надає багато інструментів та бібліотек, що спрощують розробку веб-застосунків. Наприклад, Spring Security дозволяє легко реалізувати систему авторизації та аутентифікації користувачів, а Spring Data JPA допомагає взаємодіяти з базою даних швидко та ефективно.

Однією з ключових переваг використання Spring Boot є його вбудована підтримка контейнерізації. Це означає, що ви можете легко розгортати ваші веб-застосуноки на платформах, таких як Docker, і забезпечувати їх простоту та масштабованість.

РОЗДІЛ 1
ОГЛЯД СУЧАСНИХ ТЕХНОЛОГІЙ ТА МЕТОДІВ СТВОРЕННЯ ВЕБ-ЗАСТОСУНКІВ

1.1. Поняття веб-застосунку

Веб-застосунок це програма, що зберігається на віддаленому сервері та запускається різними клієнтами через веб-браузер. Веб-програма є зручною формою програмного забезпечення, оскільки використання браузерів дозволяє програмі бути сумісною з більшістю пристроїв і операційних систем. Крім того, застосунок не займає багато пам’яті та доступний майже з будь-якого комп’ютера чи пристрою, яким може користуватися людина. Велика кількість користувачів можуть використовувати застосунок одночасно. Хоча веб-програми завжди вимагають підключення до мережі, це обмеження стало менш важливим, оскільки Інтернет стає все більш і більш повсюдним.

За останні десятиліття сучасні технології для створення веб-застосунків пройшли значний шлях. Завдяки розширенню Інтернету та зростанню популярності мережі інтернет, розробники отримали доступ до багатьох потужних інструментів і фреймворків, які значно полегшили процес розробки і дозволили створювати веб-застосунки швидше та ефективніше.

Рис 1.1. Типовий веб-застосунок

1.2. Інструменти для створення веб-застосунків

Коли ми говоримо про інструменти для створення веб-застосунків, на розгляд можна взяти різноманітні технології, фреймворки та сервіси. Ось деякі з найпопулярніших інструментів, які використовуються розробниками для створення веб-застосунків:

а) Front-End (клієнтська сторона):

– HTML/CSS: Основні мови для створення структури та стилізації веб-сторінок.

– JavaScript: Мова програмування, яка дозволяє реалізовувати динамічну функціональність на клієнтській стороні.

– Фреймворки: React.js, Angular, Vue.js - фреймворки, що допомагають у створенні інтерактивних та потужних клієнтських інтерфейсів.

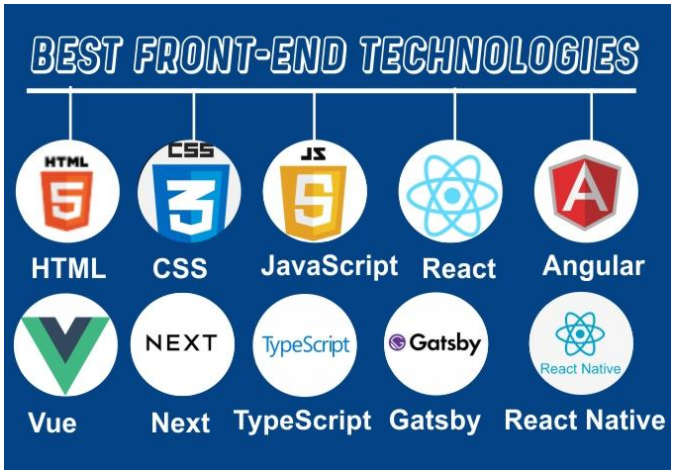


Рис. 1.2. Стек фронтенд технологій

b) Back-End (серверна сторона):

– Java: Мова програмування, часто використовується разом із фреймворком Spring Boot для створення веб-застосунків.

– Python: Мова програмування, популярна завдяки фреймворкам, таким як Django та Flask, які дозволяють швидко створювати веб-застосунки.

– Node.js: Рантайм середовище JavaScript, яке дозволяє виконувати JavaScript на серверній стороні. Розробники використовують фреймворки, такі як Express.js, для створення серверної логіки.

– Ruby: Мова програмування, яка здобула популярність завдяки фреймворку Ruby on Rails для швидкого розроблення веб-застосунків.

– PHP: Серверна мова програмування, яка широко використовується для розробки веб-застосунків. Фреймворки, такі як Laravel та Symfony, допомагають у швидкому розробленні.

1.3. Інтегровані середовища для розробки

Інтегроване середовище розробки (Integrated Development Environment або IDE) - це програмний продукт, який надає розробникам зручне середовище для написання, редагування, тестування та налагодження програмного коду. IDE забезпечують розширені можливості для покращення продуктивності розробника та забезпечують інструменти для швидкого розроблення програмного забезпечення. Ось декілька популярних інтегрованих середовищ для розробки веб-застосунків:

a) Eclipse: Eclipse - це відкрите інтегроване середовище розробки, яке підтримує різні мови програмування, включаючи Java, PHP, JavaScript та інші. Воно надає потужні інструменти для розробки веб-застосунків, такі як вбудовані редактори коду, налагоджувачі, автоматичне завершення коду та вбудовану підтримку для керування версіями.

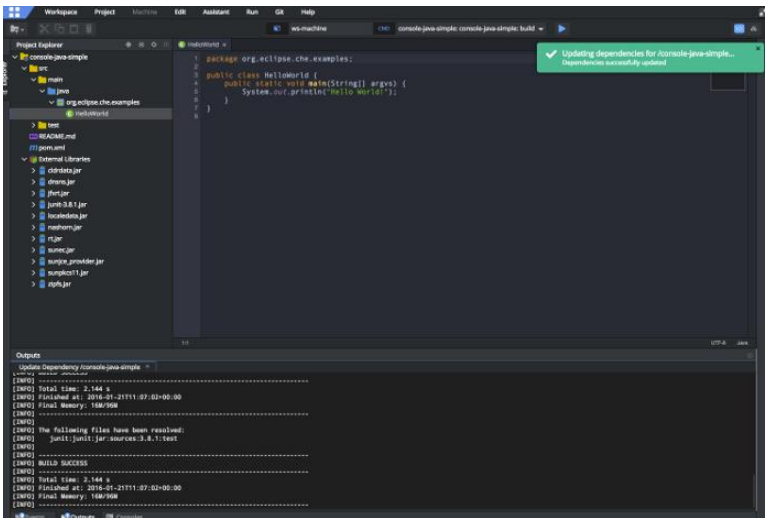


Рис. 1.4. IDE Eclipse

b) IntelliJ IDEA: IntelliJ IDEA є одним з найпопулярніших інтегрованих середовищ для розробки веб-застосунків. Воно підтримує різні мови програмування, включаючи Java, JavaScript, Python, PHP та інші. IntelliJ IDEA надає багато корисних функцій, таких як розумне автодоповнення коду, інструменти для рефакторингу коду, вбудовану систему збірки та налагоджувачі.

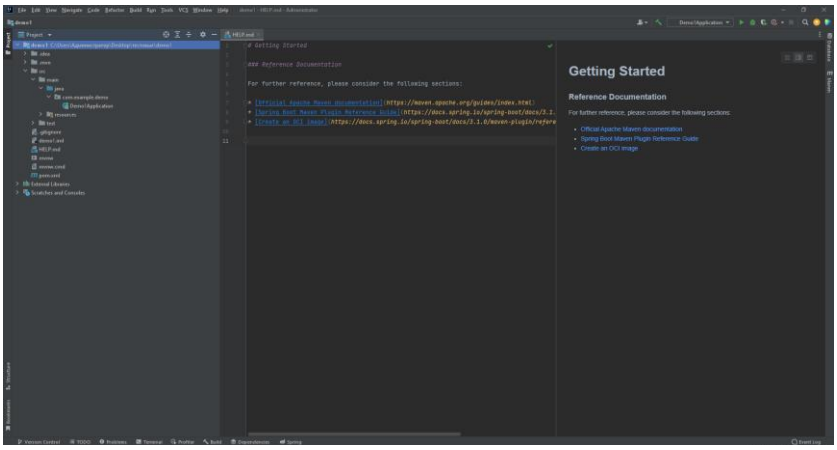


Рис. 1.5. IDE Intellij IDEA

c) Visual Studio Code: Visual Studio Code - це легкий та потужний текстовий редактор, який надає багато функцій, спеціально призначених для розробки веб-застосунків. Він підтримує багато мов програмування та надає розширення для роботи з різними фреймворками та інструментами. Visual

Studio Code здатний надати розробникам зручне та швидке середовище для написання та налагодження коду.

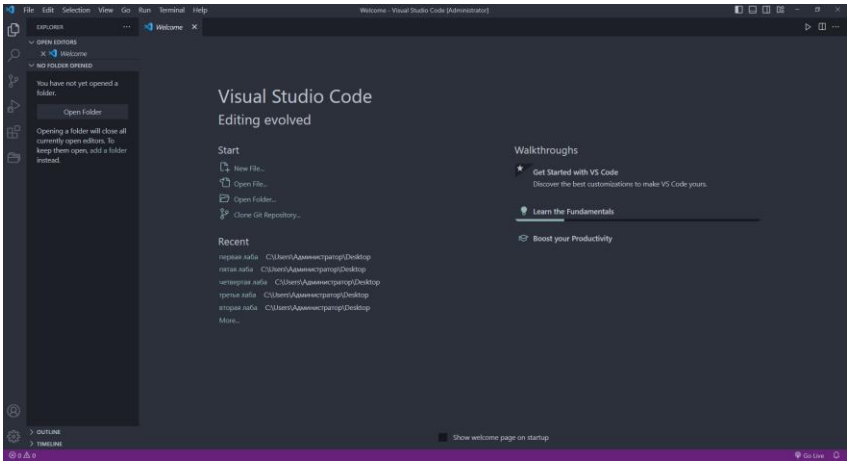


Рис. 1.6. IDE Visual Studio Code

d) NetBeans: NetBeans - це інтегроване середовище розробки, спрямоване на підтримку Java-розробки, включаючи веб-застосунки. Воно має багатий набір функцій, таких як редактор коду, систему керування версіями, інструменти для налагодження та підтримку розробки веб-застосунків з використанням різних фреймворків.

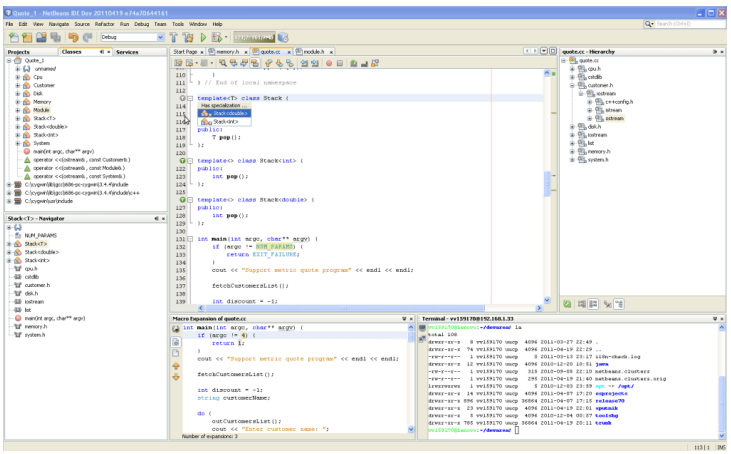


Рис. 1.7. IDE NetBeans

e) Sublime Text: Sublime Text - це швидкий і легкий текстовий редактор, який надає багато функцій для розробки веб-застосунків. Він підтримує різні мови програмування, має розширювану систему за

допомогою плагінів та надає зручний інтерфейс для редагування та оформлення коду.

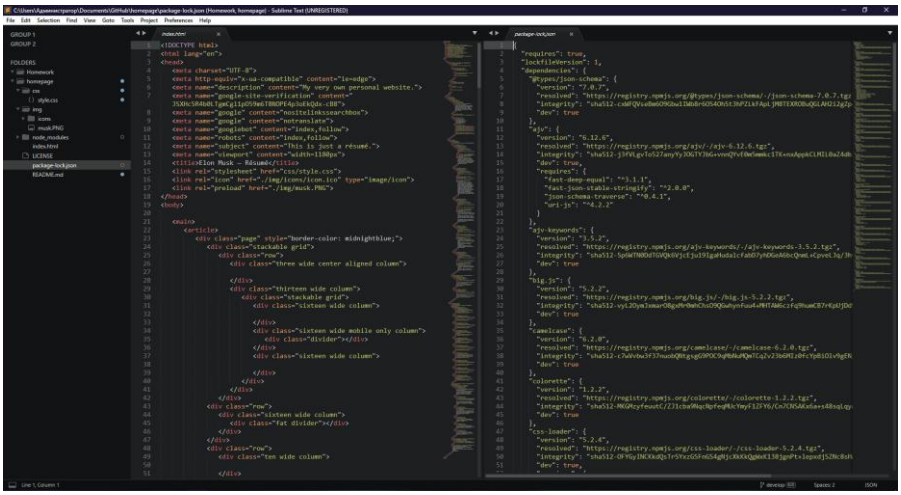


Рис. 1.8. IDE Sublime Text

Ці інтегровані середовища розробки є лише деякими з великої кількості інструментів, які розробники використовують для створення веб-застосунків. Вибір IDE залежить від ваших особистих вподобань, мови програмування та потреб вашого проекту.

1.4. Життєвий цикл веб-застосунку

Життєвий цикл веб-застосунків - це послідовність етапів і процесів, які виникають від початку розробки до підтримки та поновлення веб-застосунку. Цей цикл включає такі етапи:

1) Аналіз та планування: На цьому етапі визначаються вимоги до веб-застосунку, проводиться аналіз ринку, конкурентів та цільової аудиторії. Планується архітектура та функціональні можливості застосунку.

2) Розробка: Після аналізу розробники створюють код веб-застосунку, використовуючи вибрані технології та фреймворки. Розробка включає реалізацію функціональності, дизайну користувацького інтерфейсу, інтеграцію з базою даних та інші аспекти розробки.

3) Тестування: На цьому етапі веб-застосунок піддається тестуванню, щоб перевірити його працездатність, надійність та відповідність

вимогам. Тестування може включати функціональні тести, тести продуктивності, тести безпеки та інші види тестів.

4) Розгортання: Після успішного тестування веб-застосунок готовий до розгортання на живому сервері або хостинговому середовищі. Це включає налаштування серверного середовища, бази даних, встановлення на сервер та підготовку до запуску.

5) Експлуатація та підтримка: Після розгортання веб-застосунок починає свою роботу в реальному середовищі. Розробники та адміністратори забезпечують його стабільність, моніторинг та підтримку. Вони відповідають за виявлення та виправлення помилок, забезпечення безпеки, а також за надання оновлень та функціональних покращень.

6) Оновлення та поновлення: З часом веб-застосунок може вимагати оновлення функціональності, змін у дизайні або покращень безпеки. Розробники здійснюють регулярні оновлення, додають нові функції та покращують роботу застосунку. Цей етап може повторюватися протягом усього життєвого циклу веб-застосунку.

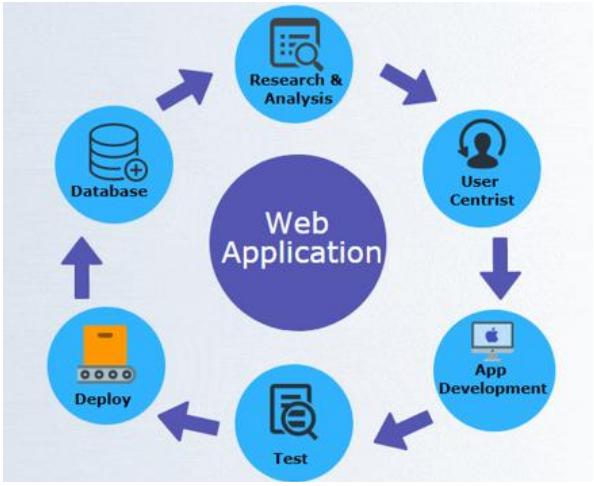


Рис. 1.9. Життєвий цикл веб-застосунків

1.5 Масштабування веб-застосунку

Масштабування веб-додатків - це процес збільшення обсягів ресурсів (серверні, бази даних, мережеві) відповідно до збільшення відвідувачів веб-

сайту або інших потреб користувачів. Для масштабування веб-додатків існують різні способи, включаючи вертикальне масштабування та горизонтальне масштабування.

Вертикальне масштабування - це спосіб збільшення потужності серверів, який полягає в установці більш потужного обладнання, такого як процесор, оперативна пам'ять, жорсткий диск, інші. Цей спосіб підходить для веб-додатків з невеликою кількістю відвідувачів, де не потрібна велика кількість серверів.

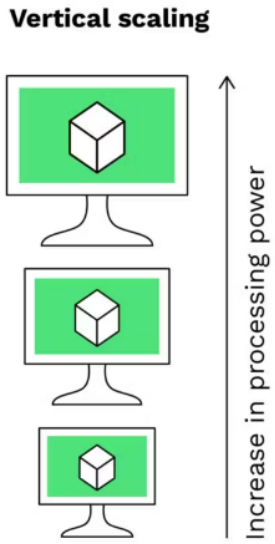


Рис. 1.10. Вертикальне масштабування

Горизонтальне маштабування - це спосіб збільшення потужності серверів шляхом додавання нових серверів в мережу. Цей спосіб забезпечує більшу гнучкість і масштабованість веб-додатків, оскільки дозволяє додавати нові сервери в мережу, коли збільшується кількість відвідувачів. Горизонтальне маштабування може бути складним процесом, оскільки потребує спеціальних інструментів для розподілення навантаження між серверами та синхронізації даних між ними.

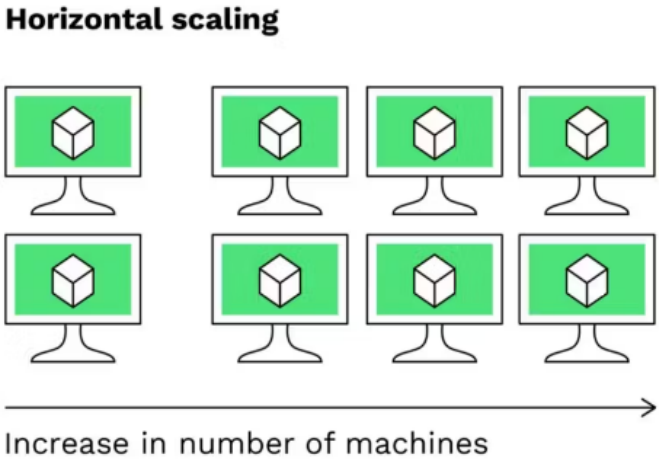


Рис. 1.17. Горизонтальне масштабування

Крім того, існують різні підходи до горизонтального маштабування. Одним з таких підходів є географічне маштабування, де сервери розташовуються в різних географічних регіонах для забезпечення швидкого доступу до веб-додатків користувачам з різних частин світу.

1.6. Оптимізація веб-застосунку

Оптимізація веб-застосунків є важливою частиною розробки, що дозволяє забезпечити швидку та ефективну роботу додатку, зменшити завантаження сервера та знизити витрати на хостинг. Оптимізація може бути спрямована на різні аспекти додатку, такі як швидкість завантаження сторінок, ресурсоємність додатку, підвищення безпеки та інші.

Один з основних аспектів оптимізації веб-додатків - це оптимізація швидкості завантаження сторінок. Швидкість завантаження сторінок залежить від багатьох факторів, таких як кількість запитів до сервера, розмір сторінки, кількість зображень та інші. Для зменшення кількості запитів до сервера можна використовувати кешування, що дозволяє зберігати копії раніше завантажених сторінок на клієнтському комп'ютері. Для зменшення розміру сторінки можна використовувати стиснення файлів, такі як зображення та стилі CSS.

Ще одним аспектом оптимізації є підвищення безпеки веб-додатку. Це може бути досягнуто за допомогою захисту від атак, таких як SQL-ін'єкції та кросс-сайтові скрипти, використання шифрування та аутентифікації.

Ресурсоємність додатку може бути знижена за допомогою ефективного використання кешування та оптимізації запитів до бази даних. Використання CDN (Content Delivery Network) також дозволяє зменшити завантаження сервера, перенаправляючи запити до найближчого сервера, що містить необхідну інформацію.

Основні принципи оптимізації веб-додатків включають наступне:

1) Мінімізація завантаження - це означає, що на сторінку потрібно завантажувати лише ті ресурси, які дійсно необхідні. Для цього можна використовувати компресію зображень та текстових файлів, кешування, змінювати розмір зображень та відео.

2) Оптимізація коду - це означає, що потрібно зробити код більш ефективним і швидким. Для цього можна використовувати мінімізацію, об'єднання та стиснення CSS та JavaScript файлів, використовувати оптимальні алгоритми, а також забезпечити оптимальну роботу баз даних.

3) Використання кешування - це означає, що потрібно зберігати часто використовувані дані в кеші, щоб уникнути зайвих запитів до сервера.

4) Пришвидшення завантаження сторінок - це означає, що потрібно зробити сторінки швидшими. Для цього можна використовувати кешування, попереднє завантаження, змінювати розмір зображень та відео.

5) Використання CDN - це означає, що можна використовувати CDN (міжнародну мережу доставки контенту), щоб додатково зменшити час завантаження сторінок.

1.7. Методи розробки веб-застосунків

Існує кілька методів розробки веб-застосунків, кожен з яких має свої переваги та особливості. Ось декілька найпоширеніших методів розробки веб-застосунків:

1) Водопадна модель (Waterfall Model): Цей метод передбачає лінійний послідовний підхід до розробки, де кожен етап виконується послідовно із фіксованими вимогами та документацією. Він включає такі етапи, як аналіз, проектування, реалізацію, тестування та впровадження. Водопадна модель підходить для проектів зі стабільними вимогами, коли заздалегідь відомо, що потрібно розробити.

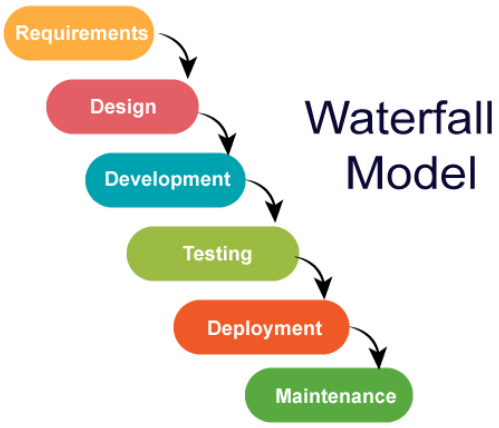


Рис. 1.12. Водопадна модель розробки ПЗ

2) Гнучка методологія (Agile Methodology): Гнучка методологія включає ітеративний та інкрементальний підхід до розробки. Вона сприяє більш гнучкій і співпраці замість жорстких планів. Гнучка методологія надає можливість більш ефективно реагувати на зміни вимог та швидко впроваджувати нові функції. Популярні гнучкі методології включають Scrum та Kanban.

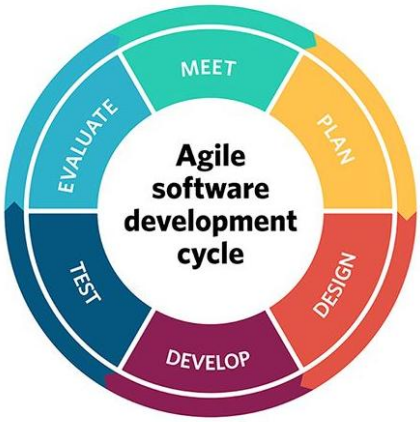


Рис. 1.13. Гнучка методологія розробки ПЗ

3) Розширена розробка (Rapid Application Development, RAD): Цей метод передбачає швидку розробку веб-застосунків за допомогою повторюваних процесів із залученням клієнта та розробників. RAD спрямований на швидку поставку функціональних прототипів, що дозволяє замовнику оцінити та внести зміни ще на ранніх етапах розробки.

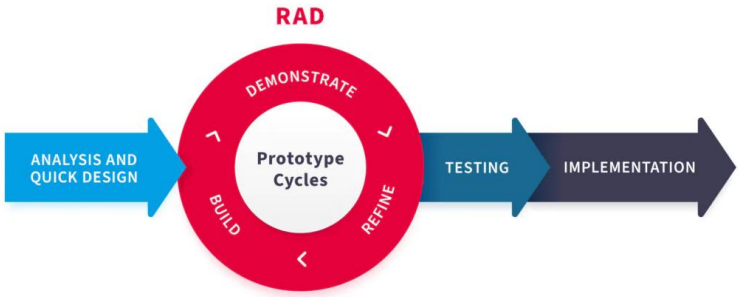


Рис. 1.14. Модель розширеної розробки ПЗ

4) DevOps: DevOps - це підхід, який поєднує розробку (Development) та операції (Operations) для створення зручного для розробки, випробування та впровадження веб-застосунків. DevOps спрямований на автоматизацію процесів, забезпечення злагодженої комунікації між розробниками та адміністраторами, а також на швидку поставку змін та оновлень.

Рис. 1.15. Модель розробки ПЗ DevOps

5) Lean Development: Lean Development базується на концепції Lean Manufacturing, спрямованої на ефективне використання ресурсів та уникнення втрат. У контексті веб-розробки, Lean Development включає етапи від постановки проблеми до впровадження рішення з акцентом на максимальну ефективність та спрощення процесів.

1.8. Тестування веб-застосунку

Тестування веб-додатків є важливою складовою розробки, оскільки дозволяє виявляти та виправляти помилки та недоліки перед тим, як вони стануть причиною проблем для користувачів.

Основні типи тестування веб-додатків:

1) Функціональне тестування – це перевірка чи коректно працюють окремі функції веб-застосунку згідно до вимог. Воно дозволяє виявляти помилки та несправності у функціональності та взаємодії різних компонентів системи.

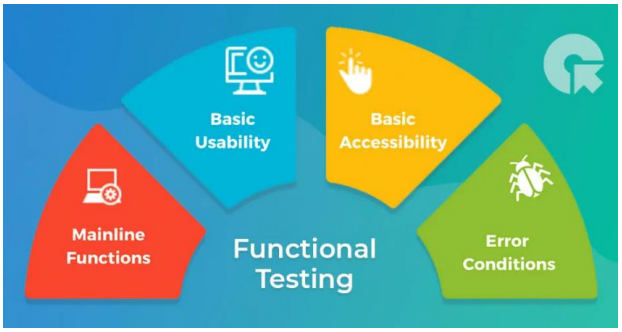


Рис. 1.16. Функціональне тестування

2) Регресійне тестування – це перевірка чи не вплинули зміни або приєднання нових функцій на вже наявну та працюючу функціональність веб-застосунку. Регресійне тестування допомагає виявити помилки та непередбачені наслідки змін, які можуть виникнути в результаті розширення або вдосконалення застосунку.

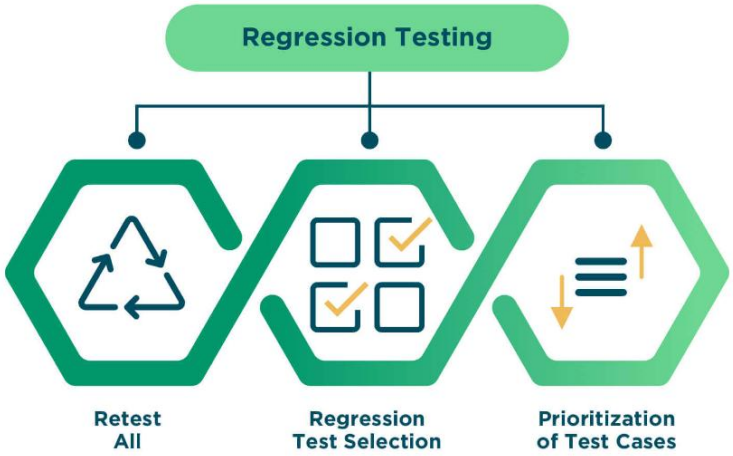


Рис. 1.17. Регресійне тестування

3) Навантажувальне тестування – це перевірка щоб визначити як застосунок працює під навантаженням та чи здатний він витримати очікувану або пікову навантаженість. Основна ціль навантажувального тестування полягає в оцінці продуктивності, шкалюваності та стійкості веб-застосунку за різних умов навантаження.

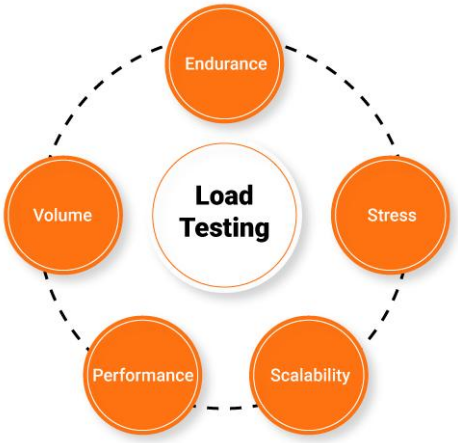


Рис. 1.18. Навантажувальне тестування

4) Безпекове тестування - перевірка на наявність потенційних слабких місць та можливих шляхів злому.

Рис. 1.19. Безпекове тестування

5) Автоматизоване тестування - використання спеціального програмного забезпечення для автоматичного тестування функцій застосунку. Воно дозволяє збільшити швидкість, ефективність та повторюваність перевірки програмного забезпечення на наявні помилки.

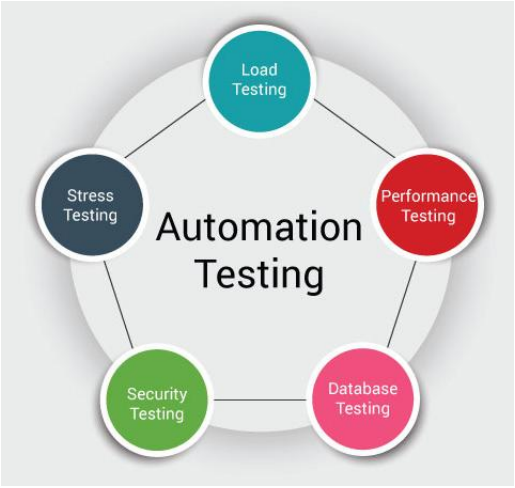


Рис. 1.20. Автоматизоване тестування

6) Мануальне тестування - перевірка функцій застосунку вручну. Воно включає в себе ручне введення даних, взаємодію з інтерфейсом користувача та перевірку результатів.

Рис. 1.21. Мануальне тестування

Основні етапи тестування веб-додатків:

1) Підготовка тест-кейсів - написання плану тестування, визначення критеріїв оцінки результатів.

2) Виконання тест-кейсів - тестування додатку відповідно до написаного плану.

3) Запис результатів - фіксування помилок та відмітка про тестовані функції.

4) Перевірка результатів - аналіз результатів тестування та виявлення помилок.

5) Повторення тестування - виправлення помилок та повторення тестування для перевірки коректності внесених змін.

1.9. Стилі та правила написання хорошого коду

Хороший код - це не просто код, який працює. Це код, який читається і зрозумілий для інших розробників, а також підтримується і розвивається з легкістю. Для написання хорошого коду існують певні стилі та правила.

Рис.1.22. Основні правила чистого коду

Ось кілька стилів та правил, які можуть допомогти написати хороший код:

1) Дотримуйтеся стандарту оформлення коду, який використовується у вашому проекті. Це може бути стандарт, такий як Google Style Guides, або внутрішній стандарт вашої компанії.

2) Дайте правильні назви змінним, функціям і класам. Назви повинні бути зрозумілими і відображати призначення об'єкту.

3) Дотримуйтесь принципу SOLID: принципи єдиної відповідальності, відкритості-закритості, принцип розділення інтерфейсів та принцип інверсії залежностей.

4) Не повторюйте код. Замість цього використовуйте функції та класи, щоб уникнути повторення коду.

5) Використовуйте коментарі, щоб пояснювати складний або неочевидний код. Але не перестарайтеся з коментарями, які надають мало користі або дублюють очевидний код.

6) Використовуйте модульну архітектуру, щоб зберігати код організованим і легко змінювати. Розділіть код на окремі модулі, які виконують окремі функції.

7) Дотримуйтеся правил оформлення коду. Розділіть код на підрозділи з відступами, правильно використовуйте дужки, пропуски та коми, щоб зробити код більш зрозумілим.

8) Пишіть тестовий код, щоб перевірити, що ваш код працює правильно.

Загалом написання чистого коду є важливим аспектом розробки програмного забезпечення, оскільки він полегшує розуміння, підтримку та подальшу модифікацію коду. Дотримуючись описаних правил ви зможете побудувати якісний веб-застосунок.

РОЗДІЛ 2
ОСОБЛИВОСТІ СТВОРЕННЯ ВЕБ-ЗАСТОСУНКІВ ТИПУ «БЛОГ» ТА АНАЛІЗ ОБРАНИХ ІНСТРУМЕНТІВ

Створення веб-застосунків типу блог може бути захоплюючим та цікавим процесом. Веб-блоги є популярними онлайн-платформами, де автори можуть публікувати свої думки, ідеї, статті, фотографії та багато іншого.

У цьому розділі буде освітлено особливості створення веб-застосунків типу блог, проаналізовано набір інструментів та підходів в реалізації проекту.

2.1. Види веб-застосунків за призначенням

Існує безліч видів веб-застосунків залежно від їх призначення і функціональності. Загалом всі вони мають майже однакову архітектуру та можуть бути побудовані за допомогою одного набору інструментів та підходів до створення веб-застосунків. Ось кілька популярних видів веб-застосунків:

1) Соціальні мережі: Це веб-застосунки, які дозволяють користувачам створювати профілі, спілкуватися з іншими користувачами, ділитися контентом, створювати групи та спільноти. Приклади таких веб-застосунків включають Facebook, Twitter, Instagram.

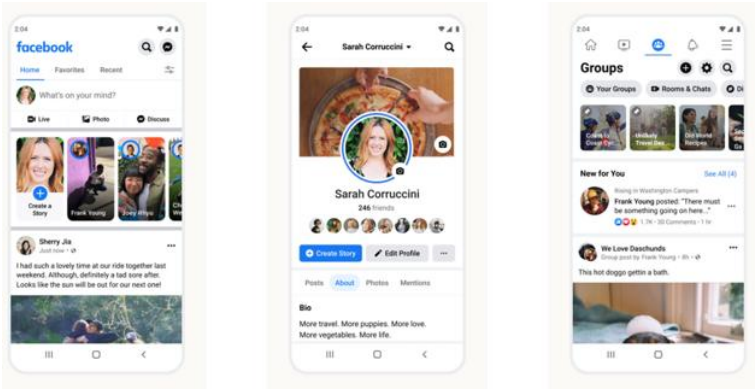


Рис. 2.1. Соціальні мережі

2) Електронна комерція: Це веб-застосунки, що дозволяють продавати товари та послуги в Інтернеті. Вони мають функції, такі як каталог товарів, кошик покупок, онлайн-оплата та доставка. Приклади таких веб-застосунків включають Amazon, eBay, Shopify.

a. 

Рис. 2.2. Ecommerce веб-застосунки

3) Блоги: Блоги є веб-застосунками, де автори можуть публікувати свої думки, статті, новини та іншу інформацію. Вони часто мають коментарі, категорії та можливості підписки. Приклади цих веб-застосунків включають WordPress, Blogger, Medium.

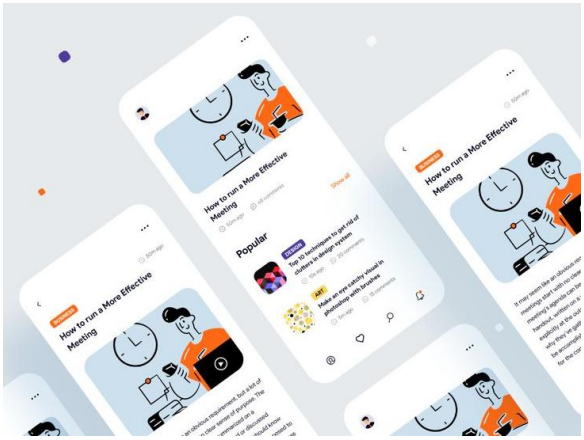


Рис. 2.3. Веб-застосунки типу блог

4) Форуми: Форуми є платформами для обговорення тем, де користувачі можуть створювати облікові записи, розміщувати повідомлення та спілкуватися з іншими учасниками. Приклади таких веб-застосунків включають phpBB, vBulletin, Discourse.

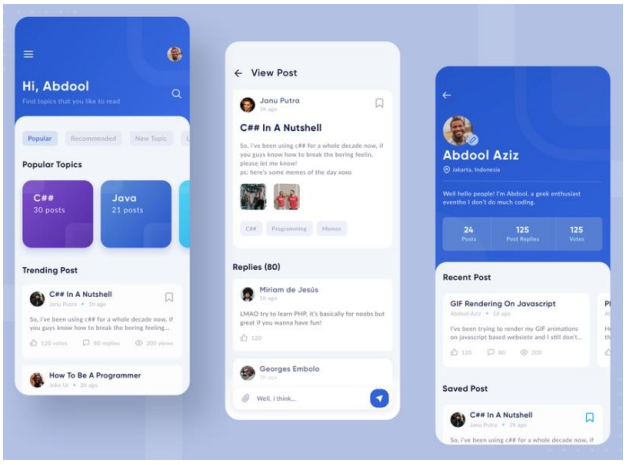


Рис. 2.4. Веб-застосунки типу форум

5) Онлайн-освіта: Це веб-застосунки, які надають можливості для навчання та співпраці в онлайн-середовищі. Вони можуть включати відеолекції, завдання, оцінювання та інші освітні ресурси. Приклади таких веб-застосунків включають Coursera, Udemy, Moodle.

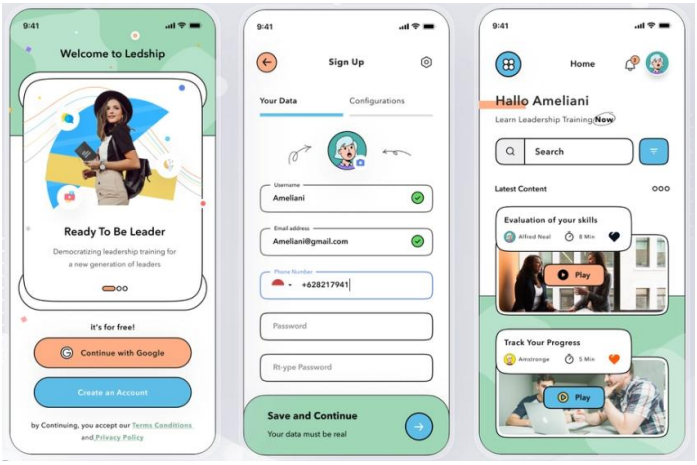


Рис. 2.5. Освітні веб-застосунки

6) Резервування та бронювання: Це веб-застосунки, які дозволяють користувачам бронювати готелі, авіаквитки, ресторани, автомобілі та інші послуги в Інтернеті. Приклади цих веб-застосунків включають Booking.com, Airbnb, OpenTable.

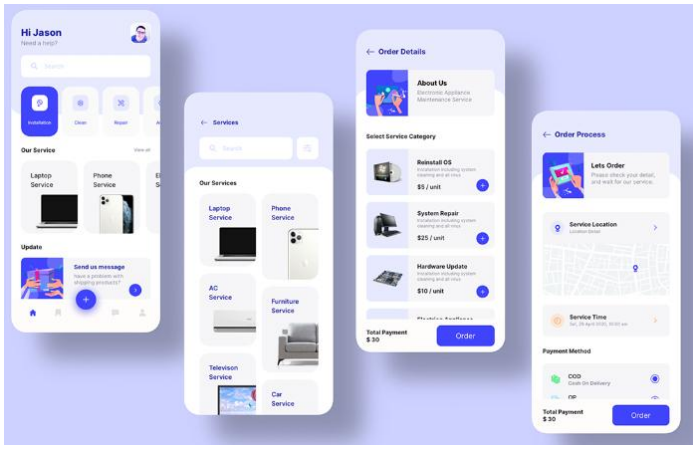


Рис. 2.6. Веб-застосунки бронювання та резервування

Це лише декілька прикладів видів веб-застосунків. Існує безліч інших категорій, таких як новинні портали, соціальні платформи для спільної роботи, медіа-галереї та багато інших, які вибираються залежно від конкретних потреб користувачів та бізнес-цілей.

2.2. Особливості створення веб-застосунків типу блог

Створення блогів має свої особливості, які варто враховувати для забезпечення коректної та ефективної роботи веб-застосунку. Ось детальний огляд особливостей створення блогів:

1) Контент: Блоги зазвичай фокусуються на контенті, тому важливо розробити стратегію контенту. Визначте тематику вашого блогу та аудиторію, щоб визначити, який контент буде найбільш цікавим для вашої аудиторії. Розробіть план публікацій, щоб регулярно додавати новий контент.

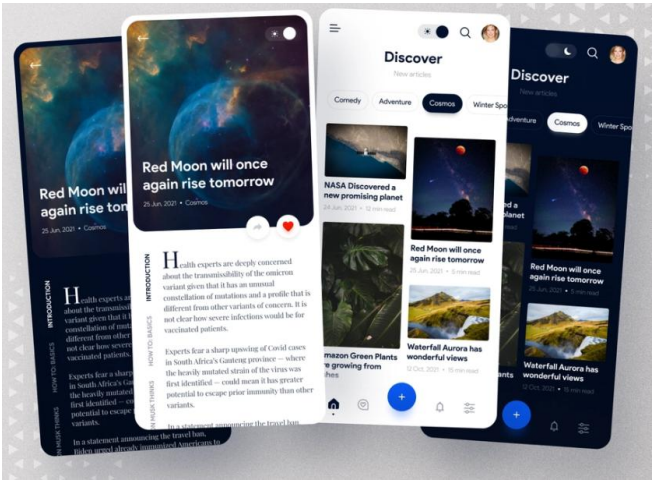


Рис. 2.7. Контент блогу

2) Дизайн і вигляд: Зовнішній вигляд вашого блогу має бути привабливим та легким у використанні. Виберіть дизайн, який відображатиме вашу бренд-ідентичність і відповідає тематиці блогу. Важливо також забезпечити зручну навігацію та читабельність контенту.

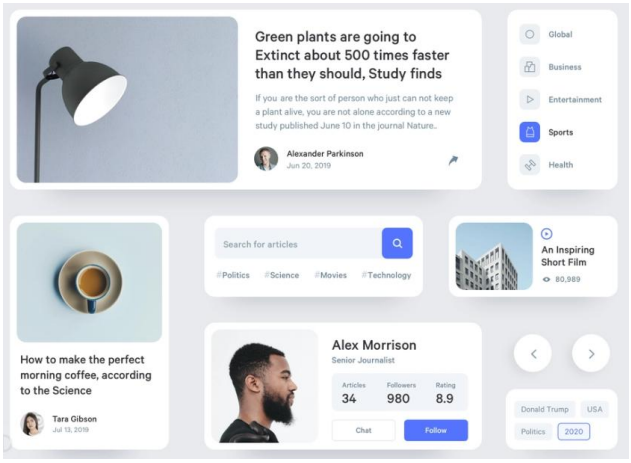


Рис. 2.8. Дизайн та вигляд блогу

3) Розділи та категорії: Розділи допомагають організувати контент на вашому блозі. Визначте основні тематичні розділи та підкатегорії, щоб допомогти читачам знайти потрібну інформацію. Забезпечте наявність чітких міток та тегів, щоб контент був легко знайдений через пошукові системи та внутрішній пошук на вашому блозі.

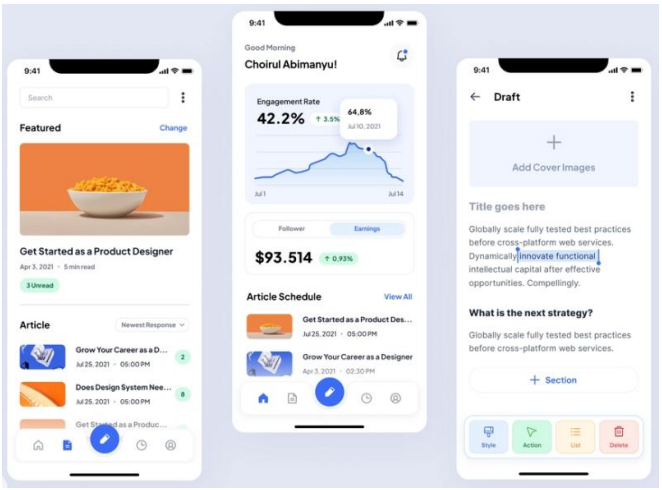


Рис. 2.9. Розділи та категорії блогу

4) Коментарі та взаємодія: Блоги зазвичай мають можливість залишати коментарі. Впевніться, що ваш блог має зручну систему коментування, яка дозволяє вам легко взаємодіяти зі своїми читачами. Ви можете розглянути можливості модерації коментарів, щоб уникнути спаму або неприйнятних відгуків.

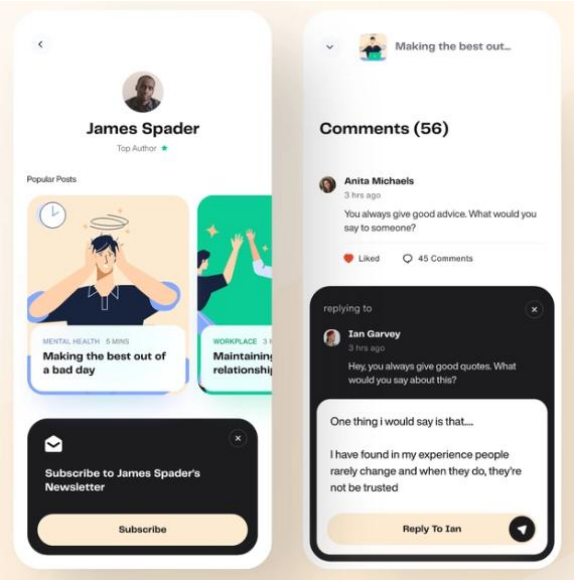


Рис. 2.10. Коментарі та взаємодія у блозі

5) SEO оптимізація: Для забезпечення більшої видимості вашого блогу у пошукових системах важливо виконати оптимізацію для пошукових систем (SEO). Використовуйте відповідні ключові слова в заголовках, описах, URL-адресах та контенті блогу. Розгляньте встановлення плагінів SEO для полегшення процесу оптимізації.

Рис. 2.11. SEO оптимізація блогу

6) Мобільна адаптація: З урахуванням зростаючого використання мобільних пристроїв, важливо забезпечити, щоб ваш блог був мобільно

адаптованим. Впевніться, що ваш дизайн респонсивний, тобто добре відображається на різних розмірах екранів та забезпечує зручну навігацію для мобільних користувачів.

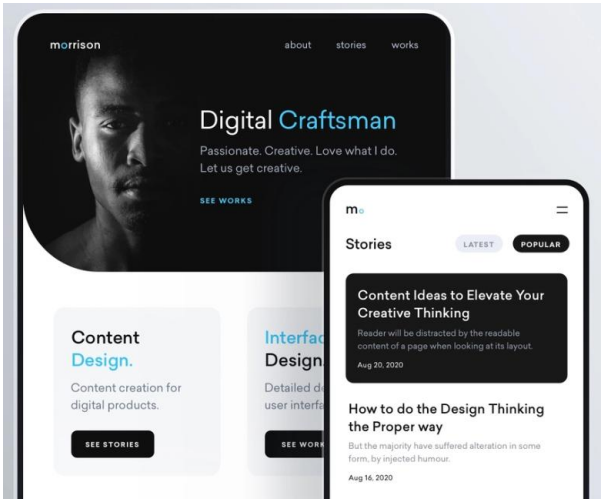
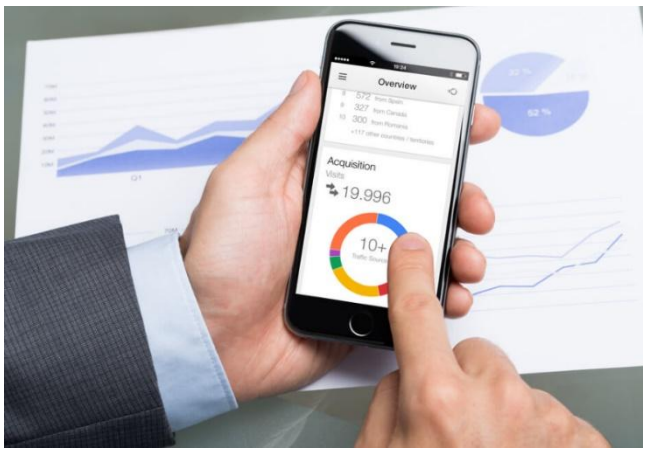


Рис. 2.12. Адаптація під різні види пристроїв

7) Аналітика та відстеження: Встановіть інструменти аналітики, такі як Google Analytics, щоб відстежувати відвідуваність вашого блогу, джерела трафіку, популярність сторінок та інші корисні метри. Це допоможе вам зрозуміти, який контент працює краще та як поліпшити ефективність



вашого блогу.

Рис. 2.13. Інструменти аналітики блогу

Ці особливості допоможуть вам створити привабливий та успішний блог, який приверне увагу читачів та забезпечить зручну взаємодію з ними. Також варто враховувати що це далеко не всі особливості створення блогів, є ще безліч інших підводних каменів у побудові таких веб-застосунків, зіткнутись з якими доведеться в процесі розробки.

2.3. Аналіз обраних інструментів та технологій для розробки блогу

Проведемо невелике дослідження обраних інструментів та технологій. В кваліфікаційній роботі буде використовуватись наступний стек технологій для створення блогу:

–Java

–Spring Boot

–Hibernate

–H2

–Intellij IDEA

2.3.1. Аналіз мови програмування Java

Почнемо аналіз з мови програмування Java. Java - це потужна та популярна мова програмування, яка була розроблена компанією Sun Microsystems (зараз належить Oracle Corporation) у 1995 році. Відома своєю надійністю, портативністю та великою спільнотою розробників. Вона має свої переваги та недоліки при створенні веб-застосунків, включаючи блоги. Огляд основних переваг та недоліків Java у контексті створення блогу:

переваги Java:

1) Платформо незалежність: Однією з головних переваг Java є його платформо незалежність. Програми, написані на Java, можуть працювати на різних операційних системах без потреби перекомпіляції або змін вихідного коду. Це означає, що ваш блог може бути запущений на будь-

якій платформі, що підтримує Java, зменшуючи залежність від конкретної ОС.

2) Велика спільнота та підтримка: Java має велику спільноту розробників та багато ресурсів, таких як документація, форуми та бібліотеки, що допомагають у розробці веб-застосунків. Це означає, що ви можете швидко знайти допомогу, розв'язати проблеми та скористатися готовими рішеннями для свого блогу.

3) Великий вибір фреймворків та бібліотек: Java має широкий вибір різноманітних фреймворків та бібліотек, які полегшують розробку веб-застосунків. Наприклад, фреймворк Spring надає потужні інструменти для розробки веб-застосунків, включаючи підтримку інверсії керування та внедрення залежностей. Це дозволяє прискорити розробку блогу та забезпечити його ефективність.

недоліки Java:

1) Складність: Java може бути дещо складним для вивчення та розуміння, особливо для початківців. Вона вимагає знання об'єктно-орієнтованого програмування та інших концепцій, що можуть затримувати розробку для деяких людей.

2) Громіздкість: Java, порівняно з іншими мовами програмування, може вимагати більшого обсягу коду для досягнення тих самих функціональних можливостей. Це може призвести до більшої тривалості розробки та обслуговування коду.

3) Важкість в розумінні: Для деяких розробників Java може викликати відчуття "важкості", оскільки вона має багато надлишкових конструкцій та строгих правил, які потрібно дотримуватися. Це може призвести до більшої складності у вирішенні простих задач або прискоренні розробки.

Враховуючи переваги та недоліки Java, важливо звернути увагу на особливості вашого проекту та ваш рівень володіння мовою програмування, перш ніж обрати Java для створення блогу.

Рис. 2.14. Логотип Java

2.3.2. Аналіз Java фреймворку Spring Boot

Продовжуємо аналіз обраних технологій, на черзі Spring Boot. Spring Boot - це один з модулів фреймворку Spring для розробки Java-додатків, який дозволяє швидко створювати самостійні, готові до використання веб-додатки. Базується на платформі Spring Framework і надає простий та зручний спосіб побудови високопродуктивних додатків. Має свої переваги та недоліки, які варто розглянути. Ось огляд основних переваг та недоліків Spring Boot у контексті створення блогу:

переваги Spring Boot:

1) Швидкість розробки: Spring Boot надає швидку розробку веб-застосунків завдяки своїй простій конфігурації. Він має вбудований контейнер сервлетів, який дозволяє швидко розгортати та запускати застосунки без необхідності вручну налаштовувати сервлет-контейнер.

2) Інтеграція зі стандартними технологіями: Spring Boot інтегрується зі стандартними технологіями Java та багатьма популярними бібліотеками, що полегшує розробку та забезпечує більшу гнучкість. Дозволяє легко підключити бази даних, системи кешування, шаблонні двигуни, системи безпеки та інші складові, що спрощує розширення та налаштування вашого блогу.

3) Спрощена конфігурація: Spring Boot забезпечує спрощену конфігурацію за замовчуванням, що дозволяє уникнути складних налаштувань та декларувань багатьох компонентів. Багато речей можна автоматично налаштувати і забезпечити додаткові можливості, такі як автоматичне створення бінів, інтеграцію з тестуванням та інше.

4) Велика спільнота та підтримка: Spring Boot існує вже майже два десятиліття, має велику спільноту розробників, що надає доступ до багатої документації, статей, форумів та інших ресурсів. Це дозволяє швидко знайти рішення для проблем, отримати підтримку та вдосконалити свій блог.

недоліки Spring Boot:

1) Ресурсоємність: Spring Boot, як фреймворк, вимагає більшої кількості пам'яті для роботи порівняно з деякими іншими фреймворками. Це може бути проблемою в обмежених обсягах пам'яті на сервері або в хостинговому середовищі.

2) Важкість у вивченні: Якщо ви не маєте попереднього досвіду роботи з Spring, вам може знадобитися певний час на вивчення фреймворку та його основних концепцій. Це може затримувати процес розробки та вимагати більшої кількості зусиль для оволодіння його можливостями.

3) Надмірність: Для невеликого проекту або простого блогу може здатися, що використання Spring Boot є надмірною складністю. Якщо ваш проект не вимагає всіх функцій та можливостей Spring Boot, ви можете вважати його зайвим навантаженням.

Враховуючи перечислені переваги та недоліки, важливо оцінити потреби вашого проекту, ваш рівень володіння фреймворком та обсяг ресурсів, доступних для вас, перед тим як обрати Spring Boot для створення блогу.

Рис. 2.15. Логотип Spring Boot

2.3.3. Аналіз ORM фреймворку Hibernate

Наступна технологія Hibernate. Hibernate - це фреймворк для об'єктно-реляційного відображення (ORM) у Java. Він надає спосіб зручної роботи з базами даних, де об'єкти Java можуть бути прямо відображені на таблиці бази даних, а взаємодія з ними відбувається за допомогою об'єктно-орієнтованих методів. Має свої переваги та недоліки при створенні веб-застосунків, включаючи блоги. Основні переваги та недоліки Hibernate у контексті створення блогу:

переваги Hibernate:

1) ORM: Hibernate надає інструменти об’єктно-реляційного відображення, які дозволяють зберігати та отримувати об'єкти з бази даних безпосередньо з використанням об'єктно-орієнтованого підходу. Це полегшує взаємодію з базою даних та дозволяє розробникам працювати з об'єктами, а не з SQL-запитами.

2) Автоматичне управління сесіями: Hibernate забезпечує автоматичне управління сесіями, що спрощує взаємодію з базою даних та зменшує кількість коду, необхідного для виконання операцій з базою даних. Він відповідає за відкриття, закриття та керування транзакціями, що робить розробку більш простою та безпечною.

3) Кросплатформеність: Hibernate можна використовувати з різними реляційними базами даних, такими як MySQL, PostgreSQL, H2, та інші. Він надає абстракцію бази даних, що дозволяє розробникам працювати з реляційними базами даних незалежно від конкретного постачальника бази даних.

4) Кешування: Hibernate надає можливість кешування запитів та об'єктів, що сприяє покращенню продуктивності застосунку. Ви можете сконфігурувати кеш для певних запитів або об'єктів, що дозволяє зменшити кількість запитів до бази даних та прискорити доступ до даних.

недоліки Hibernate:

1) Складність вивчення: Hibernate є потужним інструментом, але вимагає певного часу та знань для його повного освоєння. Він має свою власну концепцію, аннотації та налаштування, що можуть бути складними для новачків.

2) Продуктивність: Неправильне використання Hibernate може призвести до проблем продуктивності. Неправильна конфігурація або виконання неефективних запитів може призвести до повільної роботи застосунку.

3) Обмежений контроль над SQL-запитами: Hibernate надає велику частину контролю над генерацією SQL-запитів, але в деяких випадках може бути обмеженим, особливо при громіздких запитах або вимогах до оптимізації запитів.

Враховуючи описані переваги та недоліки, важливо врахувати потреби вашого проекту, ваш рівень володіння Hibernate та вимоги до продуктивності перед вибором його для створення блогу.

Рис. 2.16. Логотип Hibernate

2.3.4. Аналіз бази даних H2

Розглянемо обрану базу даних H2. H2 - це реляційна база даних, яка написана на Java. Являє собою embedded вбудовану базу даних, яка може бути використана в веб-застосунках, особливо під час розробки та

тестування. Огляд її основних переваг та недоліків у контексті роботи з веб-застосунком типу блог:

переваги H2:

1) Легкість використання: H2 має простий і зрозумілий синтаксис SQL, що полегшує розробку та налагодження веб-застосунків.

2) Вбудована база даних: H2 може бути використана як вбудована база даних без необхідності встановлення та налаштування окремого сервера баз даних.

3) Швидкодія: H2 є досить швидкодіючою базою даних, що є важливим фактором для оптимізації веб-застосунків.

недоліки MySQL:

1) Масштабованість: H2 краще підходить для невеликих та середніх проектів. Її може не вистачити для великих проектів з великим обсягом даних або високою навантаженістю.

2) Обмеження функцій: У порівнянні з більш розповсюдженими базами даних, такими як MySQL або PostgreSQL, H2 може мати обмеження функцій або не підтримувати деякі розширення SQL.

3) Нестійкість даних: Оскільки H2 зазвичай використовується як вбудована база даних, при перезапуску веб-застосунку може втрачатися збережені дані. Це може бути недоцільним для застосунків, де даних треба зберігати між перезапусками.

Беручи до уваги перечисленні переваги та недоліки, важливо враховувати потреби вашого блогу, його розмір та прогнозовану кількість користувачів, перед вибором H2 для зберігання даних вашого веб-застосунку типу блог.

Рис. 2.17. Логотип H2

2.3.5. Аналіз IDE Intellij IDEA

Останній на черзі інструмент IntelliJ IDEA. Представляє собою один з найпопулярніших інтегрованих середовищ розробки (IDE) для Java-програмування. Розроблений компанією JetBrains, IntelliJ IDEA надає розширений набір інструментів та функцій, що полегшують процес розробки програмного забезпечення. Огляд основних переваг та недоліків IntelliJ IDEA:

переваги IntelliJ IDEA:

1) Великий функціонал: IntelliJ IDEA надає багатий набір функцій, що полегшують розробку Java-програм. Вона підтримує автодоповнення коду, рефакторинг, аналіз коду, налагодження, керування залежностями та багато іншого. Це дозволяє розробникам працювати швидше та ефективніше.

2) Підтримка різних мов програмування: IntelliJ IDEA не обмежується лише Java і має вбудовану підтримку інших мов програмування, таких як Kotlin, Groovy, Scala, JavaScript, HTML, CSS та інші. Це дозволяє розробникам працювати з різними технологіями та стеками технологій в одному IDE.

3) Інтеграція з інструментами розробки: IntelliJ IDEA легко інтегрується з іншими інструментами розробки, такими як системи контролю версій (наприклад, Git), засоби автоматичної збирання (наприклад, Maven або

Gradle), сервери додатків та багато іншого. Це спрощує роботу з іншими інструментами розробки в рамках одного інтерфейсу.

4) Підтримка плагінів: IntelliJ IDEA підтримує велику кількість сторонніх плагінів, що дозволяють розширити його функціонал і адаптувати до потреб конкретного проекту або технології.

недоліки IntelliJ IDEA:

1) Ресурсоємність: IntelliJ IDEA вимагає певних обчислювальних ресурсів, таких як пам'ять та процесор. На менш потужних комп'ютерах вона може працювати повільно або вимагати додаткових настроювань.

2) Обмежений функціонал безкоштовної версії: IntelliJ IDEA має платну версію, яка надає додаткові функції та інструменти. Безкоштовна версія, IntelliJ IDEA Community Edition, має обмежений функціонал.

3) Час на освоєння: Встановлення та налаштування IntelliJ IDEA може бути складним для новачків, особливо якщо вони не знайомі з певними конфігураційними параметрами або інструментами розробки.

Враховуючи перечисленні вище недоліки, IntelliJ IDEA є потужним інструментом для розробки веб-застосунків типу блогу та надає багато переваг, що полегшують процес розробки та підвищують продуктивність розробника.

Рис. 2.18. Логотип Intellij IDEA

РОЗДІЛ 3
СТВОРЕННЯ ВЕБ-ЗАСТОСУНКУ БЛОГУ НА БАЗІ SPRING BOOT

Створення веб-застосунку може бути складним завданням, яке вимагає знань і експертизи веб-розробки, дизайну і баз даних. Однак, зростання технологій, фреймворків і інструментів спрощує цей процес, дозволяючи розробникам швидше та ефективніше втілювати свої ідеї.

Тож давайте розпочнемо подорож у світ веб-розробки та створимо застосунок, який не тільки відповідає вашим потребам, але й надає користувачам незабутній досвід і сприяє вашому успіху в онлайн-світі.

3.1. Створення веб-застосунку

Перше з чого ми маємо почати, це створення так званих сутностей entity. Поняття "entity" відноситься до основних об'єктів або моделей, які представляють різні типи даних, що зберігаються і використовуються в системі. Вони визначають структуру даних, їх атрибути та взаємозв'язки між ними. Створимо так звані моделі entity:

```java
@Entity
@Table(name = "post")
@Getter
@Setter
@NoArgsConstructor
public class Post {
    private static final int MIN_TITLE_LENGTH = 7;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "post_seq_gen")
    @Column(name = "id")
    private Long id;
    @Length(min = MIN_TITLE_LENGTH, message = "Title must be at least " + MIN_TITLE_LENGTH + " characters long")
    @NotEmpty(message = "Please enter the title")
    @Column(name = "title", nullable = false)
    private String title;
    @NotEmpty(message = "Post body can not be empty!")
    @Column(name = "body", columnDefinition = "TEXT", nullable = false)
    private String body;
    @CreationTimestamp
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "creation_date", nullable = false, updatable = false)
    private Date creationDate;
    @OneToMany(mappedBy = "post", cascade = CascadeType.REMOVE, fetch = FetchType.EAGER)
    private Collection<Comment> comments;
    @NotNull
    @ManyToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id", nullable = false)
    private BlogUser user;

    @Override
    public String toString() {
        return "Post{" +
                "id=" + id +
                ", title='" + title + '\'' +
                ", body='" + body + '\'' +
                ", creationDate=" + creationDate +
                '}';
    }
}
```

Рис. 3.1. Створення Post сутності

```java
@Data
@Entity
@Table(name = "users")
@Getter
@Setter
@NoArgsConstructor
@SequenceGenerator(name = "user_seq_gen", sequenceName = "user_seq", initialValue = 10, allocationSize = 1)
public class BlogUser implements UserDetails {
    private static final int MIN_USERNAME_LENGTH = 3;
    private static final int MIN_PASSWORD_LENGTH = 8;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "user_seq_gen")
    @Column(name = "id")
    private Long id;
    @Length(min = MIN_USERNAME_LENGTH, message = "Username must be at least " + MIN_USERNAME_LENGTH + " characters long")
    @NotEmpty(message = "Please enter username")
    @Column(name = "username", nullable = false, unique = true)
    private String username;
    @JsonIgnore
    @Length(min = MIN_PASSWORD_LENGTH, message = "Password must be at least " + MIN_PASSWORD_LENGTH + " characters long")
    @NotEmpty(message = "Please enter the password")
    @Column(name = "password", nullable = false)
    private String password;
    @Column(name = "enabled", nullable = false)
    private Boolean enabled;
    @OneToMany(mappedBy = "user")
    private Collection<Post> posts;
    @ManyToMany(cascade = CascadeType.REMOVE)
    @JoinTable(
            name = "users_authorities",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "authority_id")
    )
    private Collection<Authority> authorities;
```

Рис. 3.2. Створення BlogUser сутності

```java
@Data
@Entity
@Table(name = "comments")
@SequenceGenerator(name = "comment_seq_gen", sequenceName = "comment_seq", initialValue = 10, allocationSize=1)
public class Comment {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "comment_seq_gen")
    @Column(name = "id")
    private Long id;
    @Column(columnDefinition = "TEXT", nullable = false)
    @NotEmpty(message = "Comment body can not be empty!")
    private String body;
    @Temporal(TemporalType.TIMESTAMP)
    @CreationTimestamp
    @Column(name = "creation_date", nullable = false, updatable = false)
    private Date creationDate;
    @NotNull
    @ManyToOne
    @JoinColumn(name = "post_id", referencedColumnName = "id", nullable = false)
    private Post post;
    @NotNull
    @ManyToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id", nullable = false)
    private BlogUser user;

    @Override
    public String toString() {
        return "Comment{" +
                "id=" + id +
                ", body='" + body + '\'' +
                ", creationDate=" + creationDate +
                ", post_id=" + post.getId() +
                ", username=" + user.getUsername() +
                '}';
    }
}
```

Рис. 3.3. Створення Comment сутності

```java
@Data
@Entity
@Table(name = "authorities")
@Getter
@Setter
@NoArgsConstructor
@SequenceGenerator(name = "authority_seq_gen", sequenceName = "authority_seq", initialValue = 10, allocationSize = 1)
public class Authority implements GrantedAuthority {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "authority_seq_gen")
    @Column(name = "id")
    private Long id;
    @Column(name = "authority", unique = true, nullable = false)
    private String authority;
    @ManyToMany(mappedBy = "authorities", cascade = CascadeType.ALL)
    private Collection<BlogUser> users;

    @Override
    public String toString() {
        return "Authority{" +
                "id=" + id +
                ", authority='" + authority + '\'' +
                '}';
    }
}
```

Рис. 3.4. Створення Authority сутності

Після створення сутностей створимо так звану троьх рівневу архітектуру. "3-tier architecture" (також відома як "3-tier model" або "3-layer architecture") відноситься до архітектурного підходу, що розділяє компоненти

застосунку на три рівні, кожен з яких відповідає за певні функціональні аспекти. Ця архітектура дозволяє забезпечити модульність, масштабованість та легку зміну окремих компонентів без впливу на інші.

Основні рівні (також називають шари) в 3-tier architecture включають:

1) Presentation Layer (рівень представлення): Цей рівень відповідає за відображення інформації користувачам через графічний інтерфейс. У нашому веб-застосунку це будуть HTML та CSS, які відповідають за відображення сторінок, їх макет, стилізацію та взаємодію з користувачем.

2) Business Logic Layer (рівень бізнес-логіки): Цей рівень відповідає за обробку логіки бізнес-процесів та виконання основних операцій застосунку. В контексті веб-застосунку це можуть бути програми, які здійснюють обробку та збереження даних, валідацію користувальницького вводу, авторизацію та інші операції, необхідні для функціонування блогу.

3) Data Storage Layer (рівень зберігання даних): Цей рівень відповідає за зберігання даних у базі даних. Він забезпечує доступ до даних, збережених у системі, та виконує операції з додавання, видалення, зміни та отримання даних. У нашому випадку це реляційна база даних, яка зберігає інформацію про користувачів, пости та коментарі.

Почнемо зі створення рівня зберігання даних. Напишемо dao класи для кожної сутності. DAO (Data Access Object) є шаблоном проектування, використовуваним у програмуванні для взаємодії з базами даних. Основна ідея DAO полягає в тому, що кожна сутність бази даних (така як користувач, пост, коментар) має відповідний DAO-клас, який відповідає за всі операції з цією сутністю. DAO надає абстрактний інтерфейс для роботи з базою даних, такий як створення, зчитування, оновлення та видалення записів. Перейдемо до написання коду:

```java
import java.util.Collection;
import com.greenit.model.Post;
import org.springframework.data.jpa.repository.JpaRepository;

public interface PostRepository extends JpaRepository<Post, Long> {
    Collection<Post> findAllByOrderByCreationDateDesc();
}
```

Рис. 3.5. Створення dao для Post сутності

```java
import java.util.Optional;
import com.greenit.model.BlogUser;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BlogUserRepository extends JpaRepository<BlogUser, Long> {
    Optional<BlogUser> findByUsername(String username);
}
```

Рис. 3.6. Створення dao для BlogUser сутності

```java
import com.greenit.model.Comment;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CommentRepository extends JpaRepository<Comment, Long> {
}
```

Рис. 3.7. Створення dao для Comment сутності

```java
import java.util.Optional;
import com.greenit.model.Authority;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AuthorityRepository extends JpaRepository<Authority, Long> {
    Optional<Authority> findByAuthority(String authority);
}
```

Рис. 3.8. Створення dao для Authority сутності

Створимо наступний рівень трьох шарової архітектури, business-logic layer, так званий рівень бізнес-логіки, в якому реалізуємо необхідні service класи. Service - це інтерфейс, який описує контракт для реалізації бізнес-логіки. Він визначає методи та операції, які потрібні для обробки певних функцій або виконання операцій у вашому веб-застосунку. Наприклад може бути service-інтерфейс з методами для створення посту, отримання списку постів, додавання коментаря і т. д. Ці класи допомагають відокремити логіку додатку від презентаційного шару та шару доступу до даних, що сприяє

модульності, легкій масштабованості та підтримуваності нашого веб-застосунку.

Також існують ServiceImpl класи. ServiceImpl - це конкретна реалізація інтерфейсу Service. Вона містить реалізацію методів, вказаних у Service-інтерфейсі. В ServiceImpl класі ви можете виконувати бізнес-логіку, використовувати DAO-класи для доступу до даних та координувати взаємодію з іншими компонентами застосунку. Наприклад, Service-інтерфейс може мати метод save(Post post), а ServiceImpl-клас буде мати реалізацію цього методу, який викликає відповідний DAO-клас для збереження нового посту у базі даних. Перейдемо до написання коду:

```java
import java.util.Collection;
import java.util.Optional;
import com.greenit.model.Post;

public interface PostService {
    Optional<Post> getById(Long id);

    Collection<Post> getAll();

    Post save(Post post);

    void delete(Post post);
}
```

Рис. 3.9. Створення PostService інтерфейсу

```java
import java.util.Optional;
import javax.management.relation.RoleNotFoundException;
import com.greenit.model.BlogUser;
import org.springframework.security.core.userdetails.UserDetailsService;

public interface BlogUserService extends UserDetailsService {
    Optional<BlogUser> findByUsername(String username);

    BlogUser saveNewBlogUser(BlogUser blogUser) throws RoleNotFoundException;
}
```

Рис. 3.10. Створення BlogUserService інтерфейсу

```java
import com.greenit.model.Comment;

public interface CommentService {
    Comment save(Comment comment);

    void delete(Comment comment);
}
```

Рис. 3.11. Створення CommentService інтерфейсу

```java
import java.util.Collection;
import java.util.Optional;
import com.greenit.model.Post;
import com.greenit.repository.PostRepository;
import com.greenit.service.PostService;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@AllArgsConstructor
public class PostServiceImpl implements PostService {
    private final PostRepository postRepository;

    @Override
    public Optional<Post> getById(Long id) { return postRepository.findById(id); }

    @Override
    public Collection<Post> getAll() { return postRepository.findAll(); }

    @Override
    public Post save(Post post) { return postRepository.saveAndFlush(post); }

    @Override
    public void delete(Post post) { postRepository.delete(post); }
}
```

Рис. 3.12. Створення PostServiceImpl інтерфейсу

```java
@Service
@AllArgsConstructor
public class BlogUserServiceImpl implements BlogUserService {
    private static final String DEFAULT_ROLE = "ROLE_USER";
    private final BCryptPasswordEncoder bcryptEncoder;
    private final BlogUserRepository blogUserRepository;
    private final AuthorityRepository authorityRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Optional<BlogUser> blogUser = blogUserRepository.findByUsername(username);
        if (blogUser.isPresent()) {
            return blogUser.get();
        } else {
            throw new UsernameNotFoundException("No user found with username " + username);
        }
    }

    @Override
    public Optional<BlogUser> findByUsername(String username) { return blogUserRepository.findByUsername(username); }

    @Override
    public BlogUser saveNewBlogUser(BlogUser blogUser) throws RoleNotFoundException {
        System.err.println("saveNewBlogUser: " + blogUser);
        blogUser.setPassword(this.bcryptEncoder.encode(blogUser.getPassword()));
        blogUser.setEnabled(true);
        Optional<Authority> optionalAuthority = this.authorityRepository.findByAuthority(DEFAULT_ROLE);
        System.err.println("optionalAuthority: " + optionalAuthority);
        if (optionalAuthority.isPresent()) {
            Authority authority = optionalAuthority.get();
            Collection<Authority> authorities = Collections.singletonList(authority);
            blogUser.setAuthorities(authorities);
            System.err.println("blogUser after Roles: " + blogUser);
            return this.blogUserRepository.saveAndFlush(blogUser);
        } else {
            throw new RoleNotFoundException("Default role not found for blog user with username " + blogUser.getUsername());
        }
    }
}
```

Рис. 3.13. Створення BlogUserServiceImpl інтерфейсу

Після створення рівня бізнес-логіки перейдемо до створення рівня презентації, в якому реалізуємо controller класи. Controller - це клас, який

приймає HTTP-запити від користувачів і взаємодіє з іншими компонентами застосунку для обробки цих запитів. Він містить методи, які відповідають на різні маршрути (URL) та HTTP-методи, наприклад GET, POST, PUT або DELETE. Контролери визначають логіку, яка виконується при отриманні запиту і визначають, які дії треба виконати та яку відповідь повернути користувачеві. Наприклад, у нас є PostController клас з методом getPost, який отримує запит на відображення посту. У цьому методі контролер може викликати відповідний метод Service-класу для отримання посту з бази даних, після чого повертає відповідь, яка містить пост у форматі HTML користувачеві. Перейдемо до написання коду:

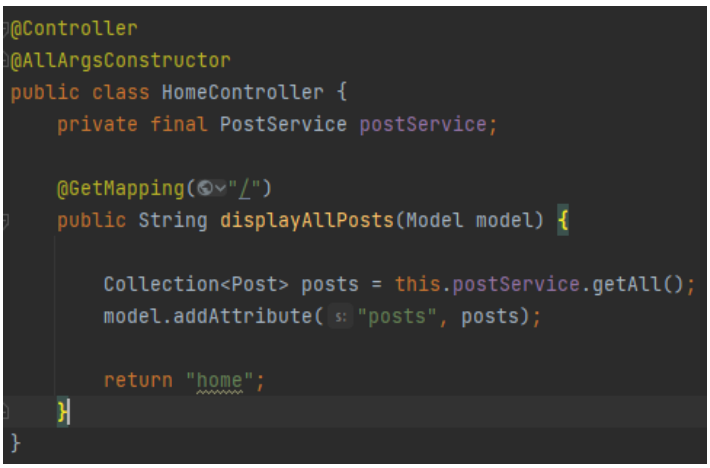


Рис. 3.14. Створення HomeController класу

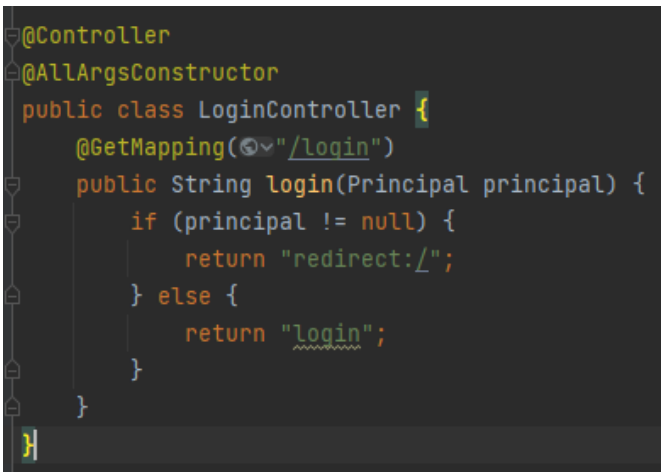


Рис. 3.15. Створення LoginController класу

```java
@Controller
@SessionAttributes("blogUser")
@AllArgsConstructor
public class SignupController {
    private final BlogUserService blogUserService;

    @GetMapping("/signup")
    public String getRegisterForm(Model model) {
        BlogUser blogUser = new BlogUser();
        model.addAttribute( s: "blogUser", blogUser);
        return "registerForm";
    }

    @PostMapping("/register")
    public String registerNewUser(@Valid @ModelAttribute BlogUser blogUser, BindingResult bindingResult,
                                  SessionStatus sessionStatus) throws RoleNotFoundException {
        System.err.println("newUser: " + blogUser);
        if (blogUserService.findByUsername(blogUser.getUsername()).isPresent()) {
            bindingResult.rejectValue( s: "username",  s1: "error.username",  s2: "Username is already registered try other one or go away");
            System.err.println("Username already taken error message");
        }
        if (bindingResult.hasErrors()) {
            System.err.println("New user did not validate");
            return "registerForm";
        }
        this.blogUserService.saveNewBlogUser(blogUser);
        Authentication auth = new UsernamePasswordAuthenticationToken(blogUser, blogUser.getPassword(), blogUser.getAuthorities());
        System.err.println("AuthToken: " + auth);
        SecurityContextHolder.getContext().setAuthentication(auth);
        System.err.println("SecurityContext Principal: " + SecurityContextHolder.getContext().getAuthentication().getPrincipal());
        sessionStatus.setComplete();
        return "redirect:/";
    }
}
```

Рис. 3.16. Створення SignupController класу

```java
@Controller
@SessionAttributes("comment")
@AllArgsConstructor
public class CommentController {
    private final PostService postService;
    private final BlogUserService blogUserService;
    private final CommentService commentService;

    @Secured("ROLE_USER")
    @GetMapping("/comment/{id}")
    public String showComment(@PathVariable Long id, Model model, Principal principal) {
        String authUsername = "anonymousUser";
        if (principal != null) {
            authUsername = principal.getName();
        }
        Optional<BlogUser> optionalBlogUser = this.blogUserService.findByUsername(authUsername);
        Optional<Post> postOptional = this.postService.getById(id);
        if (postOptional.isPresent() && optionalBlogUser.isPresent()) {
            Comment comment = new Comment();
            comment.setPost(postOptional.get());
            comment.setUser(optionalBlogUser.get());
            model.addAttribute( s: "comment", comment);
            return "commentForm";
        } else {
            return "error";
        }
    }

    @Secured("ROLE_USER")
    @PostMapping("/comment")
    public String validateComment(@Valid @ModelAttribute Comment comment, BindingResult bindingResult,
                                  SessionStatus sessionStatus) {
        System.err.println("POST comment: " + comment);
        if (bindingResult.hasErrors()) {
            return "commentForm";
        } else {
            this.commentService.save(comment);
            sessionStatus.setComplete();
            return "redirect:/post/" + comment.getPost().getId();
        }
    }
}
```

Рис. 3.17. Створення CommentController класу

```java
public class PostController {
    private final PostService postService;
    private final BlogUserService blogUserService;

    @GetMapping("/post/{id}")
    public String getPost(@PathVariable Long id, Model model, Principal principal) {
        String authUsername = "anonymousUser";
        if (principal != null) {
            authUsername = principal.getName();
        }
        Optional<Post> optionalPost = this.postService.getById(id);
        if (optionalPost.isPresent()) {
            Post post = optionalPost.get();
            model.addAttribute( s: "post", post);
            if (authUsername.equals(post.getUser().getUsername())) {
                model.addAttribute( s: "isOwner",  o: true);
            }
            return "post";
        } else {
            return "404";
        }
    }

    @Secured("ROLE_USER")
    @GetMapping("/createNewPost")
    public String createNewPost(Model model, Principal principal) {
        String authUsername = "anonymousUser";
        if (principal != null) {
            authUsername = principal.getName();
        }
        Optional<BlogUser> optionalBlogUser = this.blogUserService.findByUsername(authUsername);
        if (optionalBlogUser.isPresent()) {
            Post post = new Post();
            post.setUser(optionalBlogUser.get());
            model.addAttribute( s: "post", post);
            return "postForm";
        } else {
            return "error";
        }
    }
```

Рис. 3.18. Створення PostController класу

```java
@Secured("ROLE_USER")
@PostMapping("/createNewPost")
public String createNewPost(@Valid @ModelAttribute Post post,
                            BindingResult bindingResult, SessionStatus sessionStatus) {
    System.err.println("POST post: " + post);
    if (bindingResult.hasErrors()) {
        System.err.println("Post did not validate");
        return "postForm";
    }
    this.postService.save(post);
    System.err.println("SAVE post: " + post);
    sessionStatus.setComplete();
    return "redirect:/post/" + post.getId();
}

@Secured("ROLE_USER")
@GetMapping("editPost/{id}")
public String editPost(@PathVariable Long id, Model model, Principal principal) {
    String authUsername = "anonymousUser";
    if (principal != null) {
        authUsername = principal.getName();
    }
    Optional<Post> optionalPost = this.postService.getById(id);
    if (optionalPost.isPresent()) {
        Post post = optionalPost.get();
        if (authUsername.equals(post.getUser().getUsername())) {
            model.addAttribute( s: "post", post);
            System.err.println("EDIT post: " + post);
            return "postForm";
        } else {
            System.err.println("Current User has no permissions to edit anything on post by id: " + id);
            return "403";
        }
    } else {
        System.err.println("Could not find a post by id: " + id);
        return "error";
    }
}
```

Рис. 3.19. Створення PostController класу

```java
@Secured("ROLE_USER")
@GetMapping("/deletePost/{id}")
public String deletePost(@PathVariable Long id, Principal principal) {
    String authUsername = "anonymousUser";
    if (principal != null) {
        authUsername = principal.getName();
    }
    Optional<Post> optionalPost = this.postService.getById(id);
    if (optionalPost.isPresent()) {
        Post post = optionalPost.get();
        if (authUsername.equals(post.getUser().getUsername())) {
            this.postService.delete(post);
            System.err.println("DELETED post: " + post);
            return "redirect:/";
        } else {
            System.err.println("Current User has no permissions to edit anything on post by id: " + id);
            return "403";
        }
    } else {
        System.err.println("Could not find a post by id: " + id);
        return "error";
    }
}
```

Рис. 3.20. Створення PostController класу

Після створення моделей та трьох шарової архітектури перейдемо до створення HTML шаблонів для нашого застосунку. HTML шаблони використовуються для відображення сторінок та контенту для користувачів. Вони дозволяють створювати структуру та вигляд веб-сторінок, включаючи розмітку, стилі, зображення та інші елементи. Шаблони будуть використані для створення сторінки блогу, сторінки поста, сторінок реєстрації та авторизації. Перейдемо до написання коду:

```html
<!DOCTYPE html>
<html lang="en"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<head>
    <meta charset="UTF-8">
</head>
<body>

<header th:fragment="header">
    <div class="topnav">
        <a class="active" th:href="@{/}">Home</a>
        <span sec:authorize="hasRole('ROLE_USER')">
        <a class="left" th:href="@{/createNewPost}">New Post</a>
        </span>
        <span sec:authorize="isAuthenticated()">
        <a class="right" th:href="@{/logout}">Logout</a>
        </span>
        <span sec:authorize="isAuthenticated()">
            <a class="right">Welcome <span sec:authentication="name">name?</span></a>
        </span>
        <span sec:authorize="!isAuthenticated()">
        <a class="right" th:href="@{/login}">Log In</a>
        <a class="right" th:href="@{/signup}">Sign Up</a>
        </span>
    </div>
    <div class="header">
        <h2>Cool Blog Name</h2>
    </div>
</header>

</body>
</html>
```

Рис. 3.21. Створення Header шаблону

Шаблон header представляє собою верхню частину веб-сторінки і містить зручне навігаційне меню, за допомогою якого можна швидко переміщуватись по сайту.

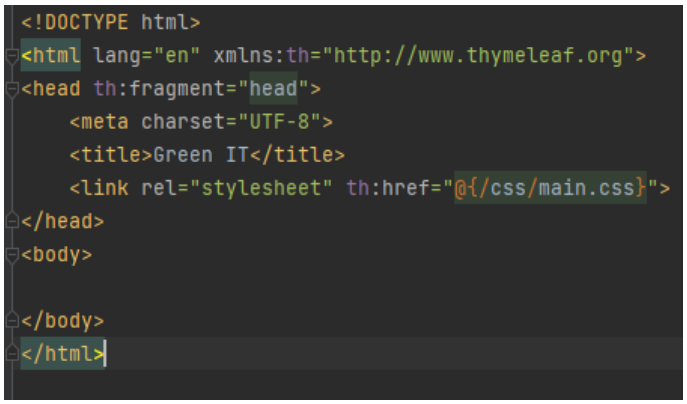


Рис. 3.22. Створення Head шаблону

Head шаблон був створений задля підключення стилів, аби не потрібно було їх підключати окремо в кожному шаблоні. Імпортуємо його в інші шаблони за допомогою команди th:fragment.

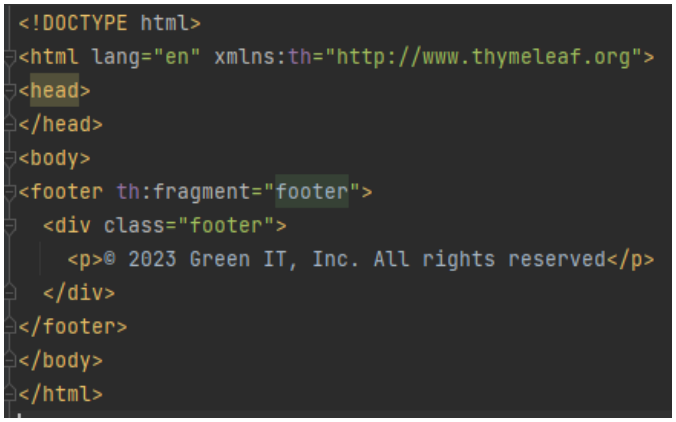


Рис. 3.23. Створення Footer шаблону

Footer шаблон є шаблоном для нижньої частини сторінки, в якому вказана назва сайту, а також в нього окремі від інших частин сторінки стилі. Шаблон створений задля швидкого імпорту в інші шаблони, без необхідності на кожній сторінці прописувати окремий footer.

Рис. 3.24. Створення Posts шаблону

Posts шаблон був створений для швидкого доступу до всіх наявних постів сайту, в ньому прописаний цикл, де відбувається перебір та вивід на екран всіх постів. Імпортуємо його до головної сторінки, де відображаються всі пости.

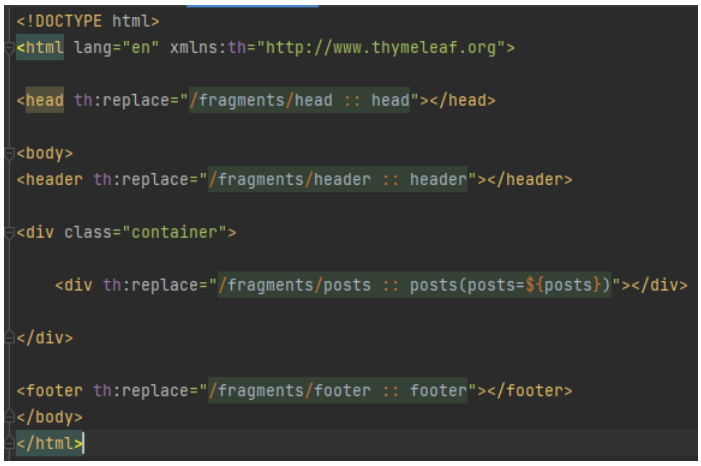


Рис. 3.25. Створення Home шаблону

Шаблон Home репрезентує собою головну сторінку нашого сайту, де відображаються всі пости. Перегляд цієї сторінки доступний всім користувачам.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head th:replace="/fragments/head :: head"></head>

<body>
<header th:replace="/fragments/header :: header"></header>

<div class="container">
    <div th:if="${param.error}">
        <p class="alert-red">Invalid username and/or password</p>
    </div>
    <div th:if="${param.logout}">
        <p class="alert-green"> You have been successfully logged out</p>
    </div>

    <form th:action="@{/login}" method="post">
        <div>
            <input type="text" name="username" id="username" placeholder="UserName" required="true" autofocus="true"/>
        </div>
        <div>
            <input type="password" name="password" id="password" placeholder="Password" required="true"/>
        </div>
        <div class="align-center">
            <label for="remember-me">Remember Me</label>
            <input type="checkbox" id="remember-me" name="remember-me">
        </div>
        <div>
            <input type="submit" class="green-btn blue-btn" value="Login"/>
        </div>

    </form>

</div>

<footer th:replace="/fragments/footer :: footer"></footer>
</body>
</html>
```

Рис. 3.26. Створення Login шаблону

Login шаблон являє собою сторінку з формою аутентифікації до нашого веб-застосунку, також є можливість зберегти дані аби не вводити їх кожен раз коли користувач захоче аутентифікуватись. Логін сторінка доступна не аутентифікованим користувачам.

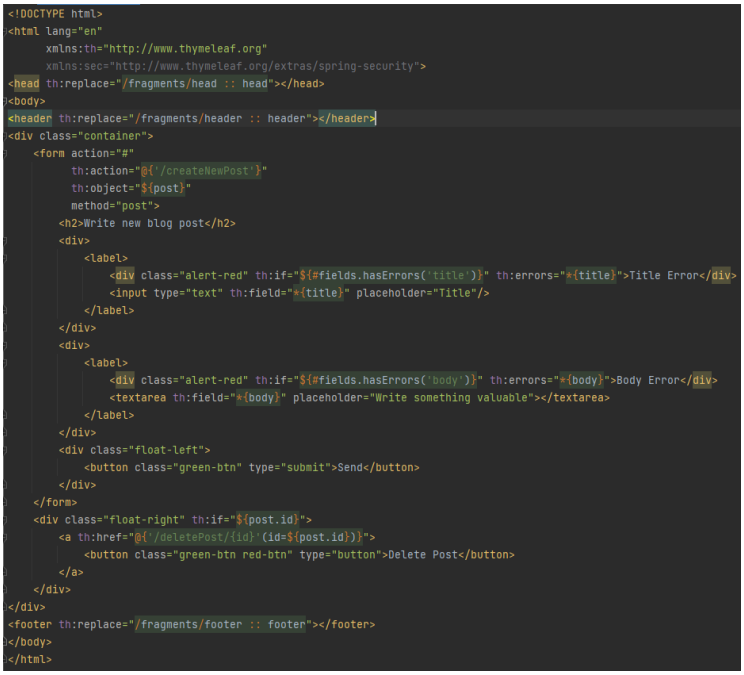
```
<!DOCTYPE html>
<html lang="en"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<head th:replace="/fragments/head :: head"></head>
<body>
<header th:replace="/fragments/header :: header"></header>

<div class="container">
    <div class="post">
        <h2 th:text="${post.title}">Title</h2>
        <h5 th:text="'Published on ' + ${#dates.format(post.creationDate, 'yyyy MMMM dd')} +
        ' by ' + ${post.user.username}">Creation date and by whom</h5>
        <div class="img">Image</div>
        <p th:text="${post.body}">body text</p>
        <div class="align-right" sec:authorize="isAuthenticated()" th:if="${isOwner}">
            <a th:href="@{'/editPost/{id}'(id=${post.id})}">
                <button class="green-btn blue-btn" type="button">Edit</button>
            </a>
        </div>
    </div>
    <div class="comments">
        <h2>Comments</h2>
        <div class="comment" th:each="comment : ${post.comments}">
            <h5 th:text="'Commented on ' + ${#dates.format(post.creationDate, 'yyyy-MM-dd')} +
            ' by ' + ${comment.user.username}">Comment date and by whom</h5>
            <p th:text="${comment.body}">Body</p>
        </div>
    </div>
    <div class="align-center">
        <a th:href="@{'/comment/{id}'(id=${post.id})}">
            <button class="green-btn" type="button">Comment Post</button>
        </a>
    </div>
</div>

<footer th:replace="/fragments/footer :: footer"></footer>
</body>
</html>
```

Рис. 3.27. Створення Post шаблону

Post шаблон представляє собою сторінку поста, на якій можна подивитись повну статтю та коментарі до неї. Якщо користувач аутентифікований він може залишити свій коментар.



```
<!DOCTYPE html>
<html lang="en"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<head th:replace="/fragments/head :: head"></head>
<body>
<header th:replace="/fragments/header :: header"></header>
<div class="container">
    <form action="#"
          th:action="@{'/createNewPost'}"
          th:object="${post}"
          method="post">
        <h2>Write new blog post</h2>
        <div>
            <label>
                <div class="alert-red" th:if="${#fields.hasErrors('title')}" th:errors="*{title}">Title Error</div>
                <input type="text" th:field="*{title}" placeholder="Title"/>
            </label>
        </div>
        <div>
            <label>
                <div class="alert-red" th:if="${#fields.hasErrors('body')}" th:errors="*{body}">Body Error</div>
                <textarea th:field="*{body}" placeholder="Write something valuable"></textarea>
            </label>
        </div>
        <div class="float-left">
            <button class="green-btn" type="submit">Send</button>
        </div>
    </form>
    <div class="float-right" th:if="${post.id}">
        <a th:href="@{'/deletePost/{id}'(id=${post.id})}">
            <button class="green-btn red-btn" type="button">Delete Post</button>
        </a>
    </div>
</div>
<footer th:replace="/fragments/footer :: footer"></footer>
</body>
</html>
```

Рис. 3.28. Створення PostForm шаблону

Шаблон PostForm репрезентує собою сторінку з формою створення поста, створювати пости можуть тільки аутентифіковані користувачі. А також є функціонал редагування поста, редагувати пост може тільки аутентифікований автор поста.

```
<!DOCTYPE html>
<html lang="en"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<head th:replace="/fragments/head :: head"></head>
<body>
<header th:replace="/fragments/header :: header"></header>

<div class="container" th:class="align-center">
    <h2>Register new user</h2>
    <form action="#"
          th:action="@{/register}"
          th:object="${blogUser}"
          method="post">
        <div>
            <label>
                <div class="alert-red" th:if="${#fields.hasErrors('username')}" th:errors="*{username}">Username Error
                </div>
                <input type="text" th:field="*{username}" placeholder="Username"/>
            </label>
        </div>
        <div>
            <label>
                <div class="alert-red" th:if="${#fields.hasErrors('password')}" th:errors="*{password}">Password Error
                </div>
                <input type="password" th:field="*{password}" placeholder="Password"/>
            </label>
        </div>
        <div>
            <button class="green-btn" type="submit">Register</button>
        </div>
    </form>
</div>

<footer th:replace="/fragments/footer :: footer"></footer>
</body>
</html>
```

Рис. 3.29. Створення RegisterForm шаблону

RegisterForm шаблон являє собою сторінку реєстрації, яка доступна не аутентифікованим користувачам.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head th:replace="/fragments/head :: head"></head>
<body>
<header th:replace="/fragments/header :: header"></header>

<div class="container">
    <h2>Write new comment</h2>
    <form action="#"
          th:action="@{'/comment'}"
          th:object="${comment}"
          method="post">
        <label>
            <div class="alert-red" th:if="${#fields.hasErrors('body')}" th:errors="*{body}">Body Error</div>
            <textarea th:field="*{body}" placeholder="Write something valuable"></textarea>
        </label>
        <div>
            <button class="green-btn" type="submit">Send</button>
        </div>
    </form>
</div>

<footer th:replace="/fragments/footer :: footer"></footer>
</body>
</html>
```

Рис. 3.30. Створення CommentForm шаблону

Шаблон CommentForm представляє собою сторінку з формою за допомоги якої можна залишити коментар. Доступна тільки аутентифікованим користувачам.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head th:replace="/fragments/head :: head"></head>

<body>
<header th:replace="/fragments/header :: header"></header>

<div class="container">
    <div class="list-group">
        <div class="alert-red"><h3 th:text="${error}">Status</h3></div>
        <div class="alert-red" th:text="'Status code: '+ ${status}"></div>
    </div>
</div>

<footer th:replace="/fragments/footer :: footer"></footer>
</body>
</html>
```

Рис. 3.31. Створення Error шаблону

Error шаблон репрезентує собою сторінку з виводом помилки, якщо така відбулась в ході взаємодії користувача з сайтом. Користувач перенаправляється на неї автоматично в разі помилки.

Маємо готові до використання сутності, трьох рівневу архітектуру, яка складається з контролерів, сервісів та DAO шарів, а також готові HTML шаблони. Наступним кроком буде підключення CSS стилів. CSS (Cascading Style Sheets) в контексті нашого застосунку використовується для задання зовнішнього вигляду і стилізації елементів HTML. Він дозволяє нам керувати розміщенням, кольорами, шрифтами, фонами, рамками та іншими аспектами візуального відображення елементів на веб-сторінці. CSS стилі підключаємо в шаблоні Head, який імпортується в інші шаблони. Після підключення стилів наш веб-застосунок виглядає наступним чином:

Рис. 3.32. Вигляд головної сторінки веб-застосунку

Рис. 3.33. Вигляд сторінки створення поста

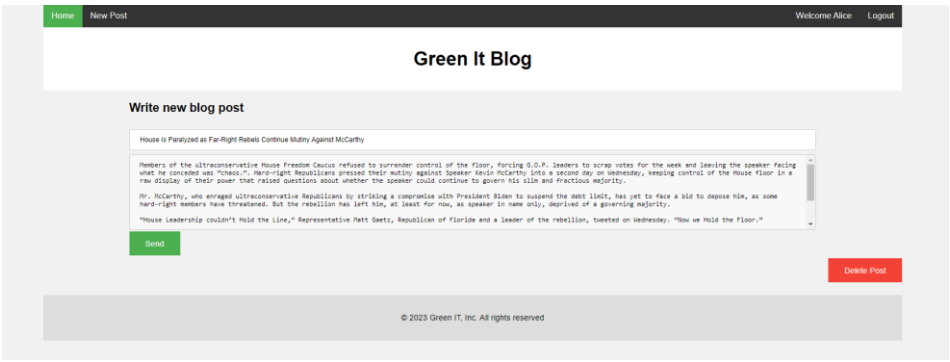


Рис.3.34. Вигляд сторінки редагування поста

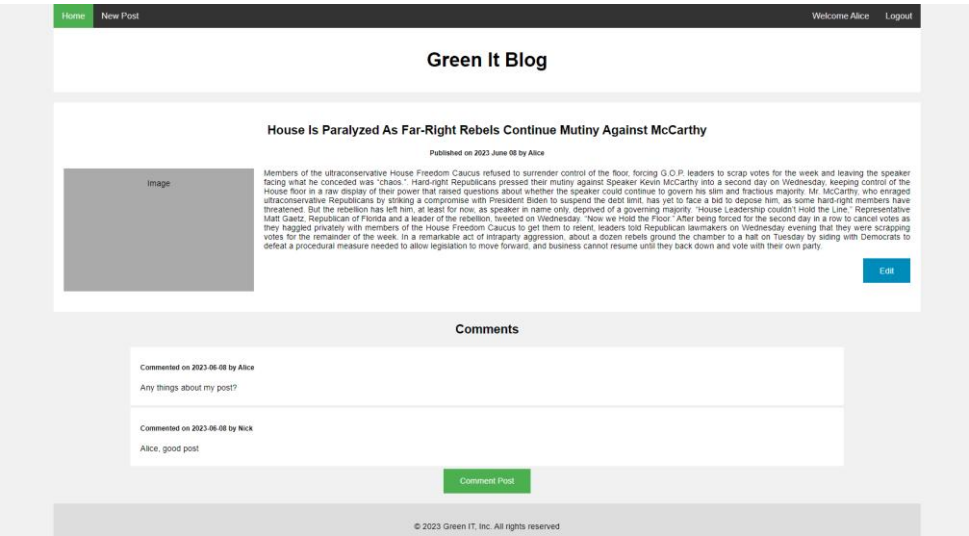


Рис. 3.35. Вигляд сторінки поста

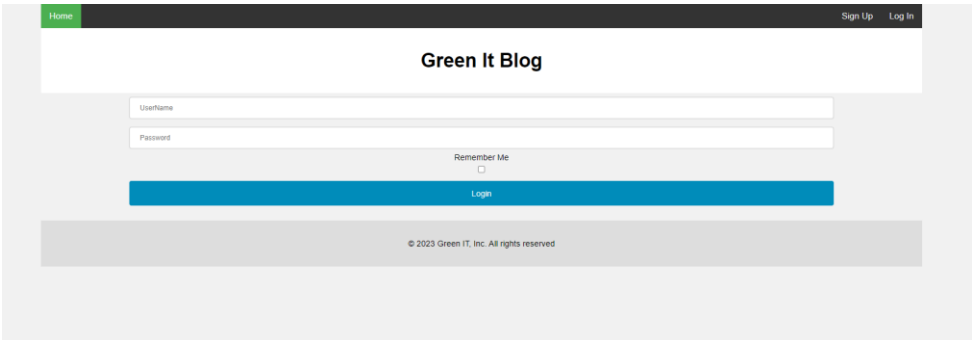


Рис. 3.36. Вигляд сторінки аутентифікації

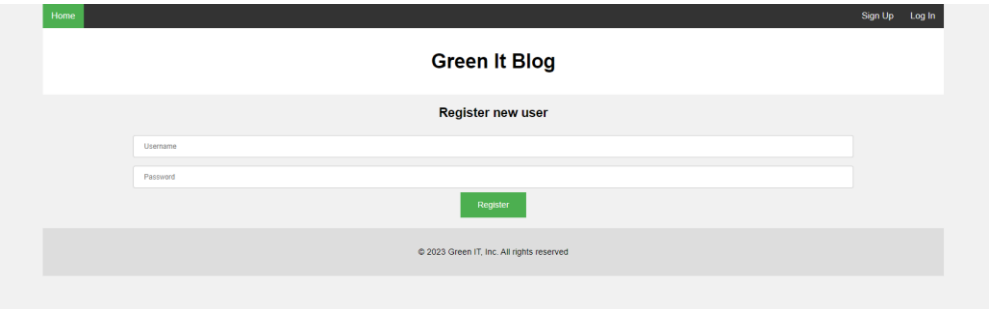


Рис. 3.37. Вигляд сторінки реєстрації

Останнім кроком в створенні нашого веб-застосунку буде конфігурування Spring Security модуля. Spring Security - це потужний безпековий фреймворк для захисту веб-застосунків на основі платформи Spring. Він надає різноманітні функції для аутентифікації, авторизації, керування сеансами та захисту від атак на наш веб-застосунок.

У контексті нашого веб-застосунку, використання Spring Security дозволить забезпечити захищений доступ до різних всіх ресурсів. Використання Spring Security дозволяє легко і ефективно управляти безпекою веб-застосунку, забезпечуючи захист від несанкціонованого доступу і зловживань. Перейдемо до конфігурування:

```
@Configuration
@EnableGlobalMethodSecurity(
        prePostEnabled = true,
        securedEnabled = true,
        jsr250Enabled = true)
public class MethodSecurityConfig extends GlobalMethodSecurityConfiguration {

}
```

Рис. 3.38. Підключення конфігурації Spring Security

```
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    private final DataSource dataSource;
    private static final String USERS_SQL_QUERY = "select username,password,enabled from users where username = ?";
    private static final String AUTHORITIES_SQL_QUERY = "select users.username, authorities.authority\n" +
            "from users\n" + "inner join users_authorities on (users.id = users_authorities.user_id)\n" +
            "inner join authorities on (users_authorities.authority_id = authorities.id)\n" + "where users.username = ?;";

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .csrf().disable()   // if enabled must use post method to logout and additional configuration needed
                .headers().frameOptions().disable() // fix H2 console access
                .and() HttpSecurity
                .authorizeRequests() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
                .antMatchers( ...antPatterns: "/createNewPost/**", "/editPost/**", "/comment/**").hasRole("USER")
                .antMatchers( ...antPatterns: "/deletePost/**").hasRole("USER")
                .antMatchers( ...antPatterns: "/**").permitAll()
                .anyRequest().authenticated()
                .and() HttpSecurity
                .formLogin() FormLoginConfigurer<HttpSecurity>
                .loginPage("/login").loginProcessingUrl("/login") // same as default implicit configuration
                .usernameParameter("username").passwordParameter("password") // same as default implicit configuration
                .defaultSuccessUrl("/").failureUrl("/login?error") // same as default implicit configuration
                .permitAll()
                .and() HttpSecurity
                .rememberMe().rememberMeParameter("remember-me") RememberMeConfigurer<HttpSecurity>
                .and() HttpSecurity
                .logout().logoutUrl("/logout").logoutSuccessUrl("/login?logout") // same as default implicit configuration
                .permitAll() LogoutConfigurer<HttpSecurity>
                .and() HttpSecurity
                .sessionManagement().maximumSessions(1);
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder authenticationManagerBuilder) throws Exception {
        authenticationManagerBuilder
                .jdbcAuthentication()
                .usersByUsernameQuery(USERS_SQL_QUERY)
                .authoritiesByUsernameQuery(AUTHORITIES_SQL_QUERY)
                .dataSource(dataSource)
                .passwordEncoder(bcryptEncoder());
```

Рис. 3.39. Конфігурування доступу до ресурсів веб-застосунку

ВИСНОВКИ

Створення веб-застосунку типу блог вимагає деяких ключових компонентів та архітектурних рішень для досягнення успіху. Використання елементів, таких як Entity, DAO, Service, Controller, HTML шаблони та CSS стилі, допомагає побудувати структуровану та функціональну систему.

Entity визначають моделі даних, які використовуються в блозі, такі як користувачі, повідомлення, коментарі, ролі. DAO класи забезпечують доступ до бази даних і реалізують операції збереження, вибірки та оновлення даних. Service та ServiceImpl класи виконують бізнес-логіку, забезпечуючи взаємодію між DAO та контролерами. Контролери обробляють HTTP-запити та відповідають на них, забезпечуючи маршрутизацію та взаємодію з фронтендом.

HTML шаблони використовуються для створення веб-сторінок блогу, розміщуючи контент, форми та інші елементи, які відображаються для користувачів. CSS стилі відповідають за оформлення сторінок, надаючи їм зовнішній вигляд та дизайн.

Додатково, використання Spring Security дозволяє забезпечити безпеку веб-застосунку, контролюючи аутентифікацію, авторизацію та інші аспекти безпеки. База даних H2 може бути використана як легка та зручна опція для розробки та тестування, але варто розглянути інші бази даних для більш масштабованих проектів.

Висновок полягає в тому, що створення веб-застосунку типу блог вимагає поєднання різних компонентів та технологій, таких як моделі даних, бізнес-логіка, контролери, веб-сторінки та безпека. З правильним використанням цих компонентів можна побудувати функціональний та привабливий блог, який задовольнить потреби користувачів та забезпечить надійну роботу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Dou: Патерни проектування [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/23683/

2. Armedsoft: Тестування веб-застосунків [Електронний ресурс] – Режим доступу до ресурсу: https://armedsoft.com/ua/blog/osoblyvosti-testuvannya-saytu

3. Websecurity: Безпека веб-застосунків [Електронний ресурс] – Режим доступу до ресурсу: http://websecurity.com.ua/?height=gevjiakanej&paged=207

4. Wezom: Види та методи створення веб-застосунків [Електронний ресурс] – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/kak-sozdat-veb-prilozhenie

5. Wikipedia: Інструменти веб розробки [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B8_%D0%B2%D0%B5%D0%B1%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8

6. Bionics: Оптимізація та масштабування веб-застосунків [Електронний ресурс] – Режим доступу до ресурсу: http://bionics.nure.ua/article/view/228458

7. W3Schools: Стаття про веб-застосунки [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/js/js_api_intro.asp

8. Tutorialspoint: Туторіал по Spring Boot фреймворку [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/spring_boot/index.htm

9. Java: Офіційний Java ресурс з описом мови програмування [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/spring_boot/index.htm

10. Baeldung: Туторіал з бази даних H2 [Електронний ресурс] – Режим доступу до ресурсу: https://www.baeldung.com/spring-boot-h2-database

11. Spring IO: Огляд фреймворку Spring Security [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-security