ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «РОЗРОБКА ВЕБ-ДОДАТКУ КОРИСТУВАЧІВ НА ОСНОВІ ТЕХНОЛОГІЙ JAVA SPRING, VUE JS»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Юрій БУРДА*

Виконав:
здобувач вищої освіти
група ІСДМ-64
Юрій БУРДА

Керівник:
науковий ступінь, вчене звання
Оксана ТКАЛЕНКО
к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання
_____
Ім’я, ПРІЗВИЩЕ

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС

_____Каміла СТОРЧАК
«____» _____20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ СТУДЕНТУ
Бурді Юрію Олександровичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: «Розробка веб-додатку користувачів на основі технологій Java Spring, Vue JS»

керівник кваліфікаційної роботи Оксана ТКАЛЕНКО, к.т.н., доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «19» жовтня 2023 року № 145.

2. Строк подання кваліфікаційної роботи: 29 грудня 2023 року.

3. Вихідні дані до кваліфікаційної роботи: Інтегроване середовище розробки;
Інформаційна платформа;
Програмне забезпечення;
Науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Дослідження доцільності побудови РЕСТ додатку на базі Spring та vue.js.
2. Аналіз технологічного стеку та інструментальних засобів для розробки додатку.
3. Результати виконаної роботи та висновки.

5. Перелік ілюстративного матеріалу: *презентація*

1. Актуальність і завдання роботи.
2. RESTful API.
3. Побудова PostgresSql DB.
4. Результати тестування.
5. Висновки по роботі.

6. Дата видачі завдання: 19 жовтня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10 – 23.10.23	
2	Вивчення матеріалів для подальшої взаємодії з ними	24.10 – 02.11.23	
3	Дослідження ефективності побудови RESTful API	03.11 – 09.11.23	
4	Оцінка ефективності використання Spring Boot	10.11 – 17.11.23	
5	Розробка програми з використанням підібраної технологічної схеми	18.11 – 21.11.23	
6	Розробка обов’язкових демонстраційних листів, доповідь	22.11 – 09.12.23	
7	Оформлення роботи: вступ, висновки, реферат	10.12 – 19.12.23	
8	Розробка демонстраційних матеріалів	20.12 – 28.12.23	

Здобувач вищої освіти _____ Юрій БУРДА
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

Керівник роботи
кваліфікаційної роботи _____ Оксана ТКАЛЕНКО
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 106 стор., 31 рис., 33 джерел.

Об'єкт дослідження: Розробка веб-додатку користувачів на основі технологій Java Spring і Vue JS.

Предмет дослідження: Процес розробки веб-додатку, включаючи взаємодію між Java Spring (на боці сервера) та Vue JS (на боці клієнта), а також функціональність та особливості цього додатку.

Мета роботи: Основною метою цієї роботи є дослідження та документування процесу розробки веб-додатку користувачів, використовуючи технології Java Spring і Vue JS, з визначенням кращих практик, архітектурних рішень та проблем, які можуть виникнути під час розробки.

Було досліджено Java Spring та Vue JS як основні інструменти для розробки на боці сервера і клієнта відповідно. Були розглянуті їх можливості, архітектурні рішення та способи взаємодії між ними.

Також було проведено аналіз алгоритмів та підходів до навчання та розгортання веб-додатків на основі Java Spring і Vue JS. Були вивчені кращі практики у валідації даних, маршрутизації, аутентифікації та авторизації.

В результаті цього дослідження було зроблено висновки щодо доцільності використання технологій Java Spring і Vue JS.

КЛЮЧОВІ СЛОВА: JAVA SPRING, VUE JS, FRONTEND ТА BACKEND, REST API, SPA (ОДНОСТОРІНКОВА ПРОГРАМА), МОДУЛЬНІСТЬ, КОМПОНЕНТИ VUE, CRUD, БАЗИ ДАНИХ, SPRING BOOT, SPRING

ABSTRACT

The text part of the qualification work for the master's degree: 106 pages, 31 figures, 33 sources.

Title: Research on the Development of User Web Application using Java Spring and Vue JS Technologies

Object of Research: Development of a user web application based on Java Spring and Vue JS technologies.

Subject of Research: The process of developing a web application, including the interaction between Java Spring (on the server side) and Vue JS (on the client side), as well as the functionality and features of this application.

Research Objective: The main goal of this work is to investigate and document the process of developing a user web application using Java Spring and Vue JS technologies.

Literature review and analysis of open sources to gather information on best practices and recommendations for developing web applications using Java Spring and Vue JS.

Examination of Java Spring and Vue JS as the primary tools for server-side and client-side development, respectively.

Analysis of algorithms and approaches to teaching and deploying web applications based on Java Spring and Vue JS.

Study of best practices in data validation, routing, authentication, and authorization.

Results: The research led to conclusions regarding the feasibility and advantages of using Java Spring and Vue JS technologies in the development of web applications.

KEYWORDS: JAVA SPRING, VUE JS, FRONTEND TA BACKEND, REST API, SPA (ОДНОСТОРІНКОВА ПРОГРАМА), МОДУЛЬНІСТЬ, КОМПОНЕНТИ VUE, CRUD, БАЗИ ДАНИХ, SPRING BOOT, SPRING

ЗМІСТ

ВСТУП

Актуальність теми Розробка веб-додатку користувачів на основі технологій Java Spring і Vue JS з використанням PostgreSQL та стандартів REST надзвичайно важлива в сучасному інформаційному та технологічному світі. Ось деякі аспекти, які підкреслюють її актуальність:

Запит на сучасні веб-додатки: Інтернет використовується для найрізноманітніших завдань, від комунікації до бізнесу. Розробка веб-додатків для користувачів залишається високо актуальною, оскільки існує постійний попит на нові та покращені додатки.

Використання Java Spring та Vue JS: Java Spring та Vue JS є добре встановленими та потужними технологіями для розробки веб-додатків. Вони надають інструменти для реалізації як серверної, так і клієнтської сторінок додатку, а також дозволяють ефективно взаємодіяти з PostgreSQL базою даних.

Споживачі та бізнес-потреби: Виходячи з росту числа користувачів та підприємств, що використовують веб-додатки, розробники повинні реагувати на змінні потреби. Розробка веб-додатку, який відповідає сучасним стандартам та вимогам, є важливим завданням.

Безпека та продуктивність: Забезпечення безпеки та оптимізація продуктивності веб-додатку є актуальними завданнями. Використання Java Spring, Vue JS та PostgreSQL може допомогти у вирішенні цих питань.

Мережа та IoT: Інтернет речей (IoT) набирає обертів, і розробка веб-додатків, які можуть взаємодіяти з IoT-пристроями, є актуальною в галузі зв'язку між веб-додатками та фізичними пристроями.

Ураховуючи ці аспекти та враховуючи ваші конкретні вимоги до веб-додатків на базі Java Spring, Vue JS, PostgreSQL та REST, можна визначити актуальність теми для дослідження та розробки в цій галузі.

Об’єкт дослідження: це процес розробки веб-додатку для користувачів з використанням технологій Java Spring і Vue JS, зі спеціальним акцентом на взаємодію та інтеграцію між цими технологіями.

Предмет дослідження: процес розробки веб-додатку для користувачів на базі технологій Java Spring і Vue JS, включаючи архітектурні рішення, структуру додатку, підходи до роботи з базою даних PostgreSQL та використання стандартів REST для взаємодії між клієнтом і сервером.

Мета і завдання дослідження: Метою дослідження є детальне вивчення та аналіз процесу розробки веб-додатку користувачів, використовуючи технології Java Spring і Vue JS з інтеграцією PostgreSQL та використанням стандартів REST. Основні завдання дослідження включають:

Вивчення технічних особливостей Java Spring та Vue JS, з основним наголосом на їхніх можливостях та способах взаємодії.

Створення веб-додатку для користувачів на базі вищезгаданих технологій, включаючи взаємодію між клієнтом і сервером.

Проведення оцінки функціональності, безпеки та продуктивності веб-додатку та розробка оптимізаційних заходів для покращення якості та швидкості роботи.

Методика дослідження: технології Java Spring та Vue JS

Наукова новизна: Використання комбінації технологій, таких як Java Spring для бекенду, Vue JS для фронтенду, PostgreSQL для бази даних та стандарти REST для взаємодії, є важливим інноваційним кроком. Ця комбінація дозволяє створити сучасний та ефективний веб-додаток з високою продуктивністю та безпекою. Підбір оптимальних заходів для поліпшення продуктивності веб-додатку, що є важливим аспектом у сучасних вимогах до веб-додатків

Практична значущість результатів: Результати дослідження можуть бути використані розробниками для побудови сучасних та продуктивних веб-додатків для користувачів. Вони надають підказки щодо оптимальних архітектурних рішень та способів взаємодії між компонентами додатку. Дослідження включає аналіз безпеки веб-додатків та можливість вдосконалення заходів безпеки.

Апробація результатів магістерської роботи. Бурда Ю.О. «Дослідження технології побудови ефективного додатку з використанням restful api». Тези доповіді на І всеукраїнська науково-технічна конференція «Технологічні

горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу україни і світу». – Київ, 28 листопада 2023 р.

Публікації. Бурда Ю.О. «Дослідження проблеми оптимізації продуктивності в мові програмування Java».Стаття у загальногалузевому науково-виробничому журналі «Зв'язок», м.Київ - №5(165), 2023. – С.15-20

1 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЧНОГО СТЕКУ ДЛЯ РОЗРОБКИ ВЕБ-ДОДАТКУ: JAVA SPRING I VUE JS

1.1 Огляд технологічного стеку та архітектурні особливості RESTful API

RESTful API (Representational State Transfer API) - це архітектурний стиль для створення веб-сервісів, який базується на принципах та обмеженнях, які описані в статті Роя Філдінга "Архітектурний стиль REST" (Representational State Transfer). RESTful API використовує HTTP протокол для здійснення взаємодії між клієнтом та сервером. Основні характеристики та особливості RESTful API включають таке:

Ресурси (Resources): У RESTful API дані та функції представлені як ресурси, кожен з яких ідентифікується унікальним URL. Наприклад, ресурсом може бути "користувач", "замовлення" або "продукт".

HTTP методи: REST використовує стандартні HTTP методи для взаємодії з ресурсами. HTTP методи, що використовуються в RESTful API, відіграють важливу роль у взаємодії між клієнтом і сервером. Основні HTTP методи, які використовуються в REST, включають таке:

GET: HTTP метод GET використовується для отримання інформації з сервера. Клієнт відправляє запит на сервер для отримання ресурсу за певним URL. GET-запити зазвичай є безпечними та ідемпотентними, що означає, що вони не повинні змінювати стан сервера і можуть бути кешовані.

POST: HTTP метод POST використовується для створення нового ресурсу на сервері. Цей метод використовується для відправки даних на сервер для обробки та зберігання. POST-запити не є ідемпотентними, оскільки вони можуть створювати новий ресурс кожен раз, коли вони виконуються.

PUT: HTTP метод PUT використовується для оновлення існуючого ресурсу або для створення його, якщо він не існує. Клієнт відправляє дані на сервер для заміни вмісту ресурсу за вказаним URL. PUT-запити є ідемпотентними, оскільки вони завжди оновлюють ресурс до одного та того ж стану.

PATCH: HTTP метод PATCH використовується для часткового оновлення існуючого ресурсу. Він дозволяє відправити тільки ті дані, які потрібно оновити, без заміни всього ресурсу. Цей метод корисний для зменшення обсягу даних, що передаються.

DELETE: HTTP метод DELETE використовується для видалення ресурсу на сервері. Клієнт надсилає запит на видалення за вказаним URL. DELETE-запити є ідемпотентними, оскільки вони завжди видаляють ресурс до одного і того ж стану.

HEAD: HTTP метод HEAD подібний до GET-запиту, але не повертає тіло відповіді, а лише заголовки. Він корисний для перевірки наявності ресурсу та отримання інформації про ресурс без зайвого обсягу даних.

OPTIONS: HTTP метод OPTIONS використовується для отримання інформації про можливості та параметри ресурсу, включаючи підтриму HTTP методів та заголовків [1].

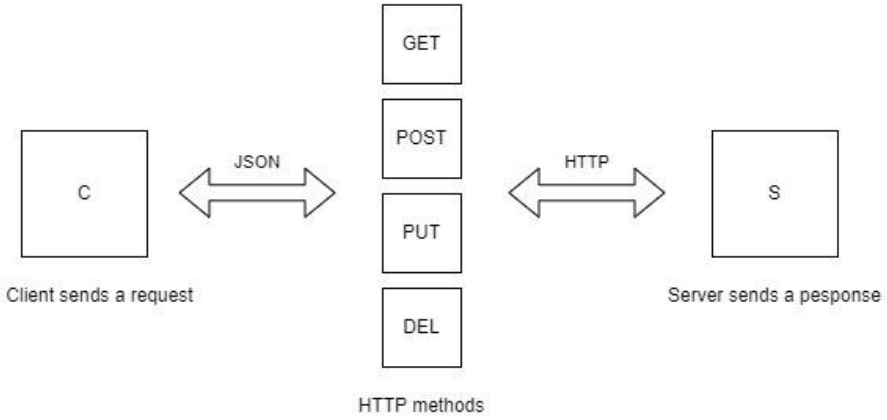


Рисунок 1.1 - Схема стандартного RESTful API

Ідемпотентність - це важливий аспект проектування та реалізації RESTful API, який визначає, як ведуть себе HTTP-запити в контексті повторних викликів. Поняття ідемпотентності означає, що певний HTTP-запит може бути виконаний багато разів, і кожен наступний виклик має той же результат, що і перший, без негативних наслідків або небажаних змін.

Основні характеристики ідемпотентних запитів включають таке:

Незмінність стану: Ідемпотентний запит не змінює стану сервера чи ресурсу

при кожному повторному виклику. Це означає, що незалежно від того, скільки разів ви викликаєте ідемпотентний запит, стан системи після кожного виклику залишається незмінним.

Безпека для використання: Ідемпотентні запити є безпечними для виклику багато разів, оскільки вони не порушують інтегрітету даних чи стану сервера. Це робить їх більш надійними та менше схильними до помилок.

Зменшення суттєвих витрат: Ідемпотентність сприяє економії ресурсів, оскільки повторні виклики можуть бути оброблені сервером без необхідності додаткових операцій або обчислень.

Збірну інформацію представлено на рисунку 1.2 (рис 1.2)

Idempotence

When performing an operation again gives the same result

HTTP method	Idempotence	Safety
GET	YES	YES
Head	YES	YES
PUT	YES	NO
DELETE	YES	NO
POST	NO	NO
PATCH	NO	NO

Рисунок 1.2 - Ідемпотентність та безпечність

HTTP методи дозволяють клієнту та серверу взаємодіяти за стандартизованим способом. Вони використовуються для виконання різних операцій над ресурсами, таких як отримання, створення, оновлення та видалення. З правильним використанням цих методів можна створити добре структуровані та ефективні RESTful API для веб-додатків.

У RESTful API тіло запита та тіло відповіді містять дані, які передаються між клієнтом і сервером. Також у HTTP-запитах і відповідях використовуються заголовки для передачі додаткової інформації. Ось розширена інформація про ці аспекти:

Тіло запита (Request Body):

Тіло запита містить дані, які клієнт надсилає серверу разом із запитом. Ці дані можуть бути в різних форматах, таких як JSON, XML, HTML і т. д., в залежності від типу даних, які передаються.

Наприклад, при створенні нового користувача в системі, клієнт може надсилати POST-запит і включати в тіло запита дані користувача у форматі JSON або XML.

JSON і XML - це два різних формати обміну даними, які часто використовуються в RESTful API для передачі інформації між клієнтом та сервером. Ось деякі відмінності між ними:

JSON (JavaScript Object Notation):

JSON є легким та простим для читання та запису форматом даних. Він базується на синтаксисі мови JavaScript і складається з пар ключ-значення.

JSON підтримує об'єкти (обгортки), масиви, рядки, числа, булеві значення та значення null.

Він зазвичай займає менше місця в порівнянні з XML, оскільки має менше розгортання міток та зайвих символів.

JSON популярний серед розробників веб-додатків, оскільки його легко обробляти та розуміти [2].

Приклад JSON-об'єкта:

```
{
  "name": "John Doe",
  "age": 30,
  "city": "New York"
}
```

XML (eXtensible Markup Language):

XML є розширюваним мовним форматом, який базується на розмітці з використанням тегів та атрибутів. Він використовує пари відкриваючого та закриваючого тегів для структурування даних.

XML підтримує довільні користувацькі теги та атрибути, що робить його більш гнучким для опису даних.

XML має більшу вагу, оскільки включає розмітку, теги та атрибути. Він менш компактний, ніж JSON.

XML частіше використовується в структурованих даних та обміні інформацією між різними системами.

Приклад XML-документа:

```
<person>
  <name>John Doe</name>
  <age>30</age>
  <city>New York</city>
</person>
```

Тіло відповіді (Response Body):

Тіло відповіді містить дані, які сервер надсилає клієнту відповідно до запиту. Як і тіло запита, тіло відповіді може бути в різних форматах і містити інформацію про ресурс, який запитував клієнт.

Наприклад, при GET-запиті для отримання інформації про користувача, сервер може надсилати відповідь з тілом, яке містить дані користувача у форматі JSON або XML.

Заголовки запиту (Request Headers):

Заголовки запиту містять додаткову інформацію про запит, таку як тип даних, які передаються, аутентифікаційні дані, мова запиту та інші метадані. Заголовки запиту допомагають серверу коректно обробляти запит та встановлювати необхідні параметри відповідно до запитуваної дії.

Приклади заголовків запиту включають "Content-Type" (тип вмісту), "Authorization" (аутентифікація), "Accept-Language" (мова запиту) та інші.

Заголовки відповіді (Response Headers):

Заголовки відповіді містять інформацію, що пов'язана з самою відповіддю сервера, таку як статус-код, тип вмісту, додаткові параметри кешування та інші метадані. Заголовки відповіді допомагають клієнту зрозуміти, як обробляти отриману відповідь.

Приклади заголовків відповіді включають "Content-Type" (тип вмісту),

"Cache-Control" (керування кешуванням), "Status" (статус-код відповіді) та інші. Ось, у наведеному прикладі, з'єднання є активним (тобто клієнт бажає, щоб воно залишалося відкритим), і значення дорівнює 100. Зазвичай значення 100 є достатнім для практично будь-якого сценарію. Однак ви можете збільшити його в залежності від кількості файлів, які сервер повинен доставити на веб-сторінку (рис 1.3)

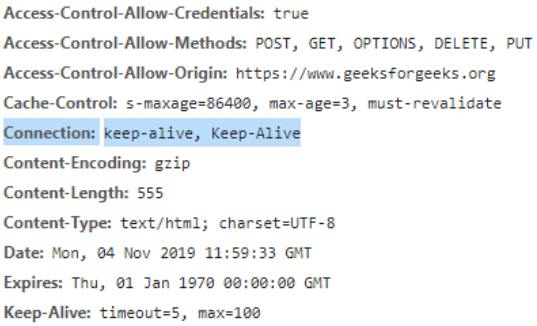


Рисунок 1.3 - Заголовки відповідей [3]

Заголовки та тіла запитів і відповідей використовуються для передачі різних аспектів інформації між клієнтом і сервером і грають важливу роль у взаємодії між ними. Вони допомагають забезпечити коректну та ефективну комунікацію у веб-додатках на основі RESTful API .

Представлення даних: Дані в RESTful API представлені у структурованому форматі, такому як JSON або XML. Це дозволяє клієнтам та серверам легко обмінюватися інформацією.

Безстандартність (Statelessness): REST не зберігає стан клієнта на сервері між запитами. Кожен запит включає всю необхідну інформацію для виконання операції. Це робить систему більш масштабованою та надійною.

Спільність (Uniform Interface): REST використовує спільний інтерфейс для взаємодії, що включає стандартизовані HTTP методи та URL-шаблони для доступу до ресурсів.

Локальність (Locality of Information): Дані про ресурси та їх стан зберігаються на сервері, і клієнт отримує доступ до них через URL.

RESTful API широко використовується для створення веб-сервісів та взаємодії між різними компонентами системи. Вони застосовуються у таких

областях, як:

Розробка веб-додатків: RESTful API дозволяє створювати веб-сервіси, які надають доступ до функціональності додатків через мережу.

Мобільна розробка: RESTful API використовується для забезпечення спільного доступу до даних та функціональності мобільних додатків.

Інтеграція сервісів: RESTful API дозволяє інтегрувати різні сервіси та джерела даних, забезпечуючи їх взаємодію через HTTP.

1.2 Дослідження Java Spring фреймверка та порівння з аналогічними технологіями

Java Spring - це потужний фреймворк для розробки серверних додатків у мові програмування Java. Основні характеристики та переваги включають:

Легкість використання: Java Spring надає ряд інструментів та шаблонів, що спрощують розробку серверних додатків.

Широкий функціонал: Фреймворк пропонує різноманітні модулі, такі як Spring Boot, Spring Security, Spring Data, які дозволяють розробникам легко додавати функціональність до своїх додатків.

Інверсія контролю та внедрення залежностей: Spring використовує принципи інверсії контролю та внедрення залежностей, що сприяє зрозумінню та тестуванню коду.

Підтримка RESTful API: Spring добре підходить для створення RESTful API, що робить його відмінним вибором для розробки серверної частини веб-додатку.

Java Spring - це розширена екосистема для розробки веб-додатків та додатків на основі Java. Вона має велику кількість бібліотек, які спрощують створення різних аспектів додатків. Ось деякі основні та популярні бібліотеки та залежності, які часто використовуються в Java Spring:

*Spring Boot: Spring B*oot - це основна бібліотека для створення самостійних, автономних додатків Spring. Вона автоматизує багато конфігураційних завдань та дозволяє швидко розпочати розробку. [4]

Spring MVC: Spring MVC (Model-View-Controller) - це бібліотека для створення веб-додатків та RESTful служб на основі паттерну MVC.

Spring Data: Spring Data надає спрощений доступ до різних джерел даних (реляційні БД, NoSQL, Redis і т. д.) та допомагає зменшити багато рутинних завдань роботи з даними.

Spring Security: Spring Security - це бібліотека для реалізації аутентифікації, авторизації та управління правами доступу в додатках.

Spring Cloud: Spring Cloud - це набір бібліотек та інструментів для розробки мікросервісних додатків, включаючи інструменти для керування конфігурацією, відкритість, балансування навантаження і багато іншого.

Spring Integration: Spring Integration - це бібліотека для інтеграції різних систем та додатків шляхом визначення шаблонів обміну даними.

Spring Batch: Spring Batch - це бібліотека для обробки великих обсягів даних, таких як пакетна обробка даних, із підтримкою моніторингу та відновлення випадкових збоїв.

Spring WebFlux: Spring WebFlux - це реактивний стек для створення веб-додатків з підтримкою реактивного програмування.

Spring AOP: Spring AOP (Aspect-Oriented Programming) - це бібліотека для реалізації аспектно-орієнтованого програмування для обробки певних аспектів додатка, таких як журналювання, безпека і кешування.

Spring Test: Spring Test - це набір інструментів для тестування додатків, включаючи модульні, інтеграційні та взаємодійні тести.

Spring Boot Starter: Spring Boot Starter - це набір стандартних залежностей для різних типів додатків (наприклад, web, data, security), які можуть бути включені в проект для швидкого старту.

Hibernate: Хоча Hibernate - це окрема бібліотека для роботи з реляційними БД, вона часто використовується в поєднанні з Java Spring для доступу до даних.

Ці бібліотеки і залежності допомагають розробникам створювати високоефективні та розширювані додатки на основі Java Spring, надаючи різноманітні інструменти для роботи з веб-додатками, даними, безпекою,

мікросервісами та багато іншого (рис 1.4).

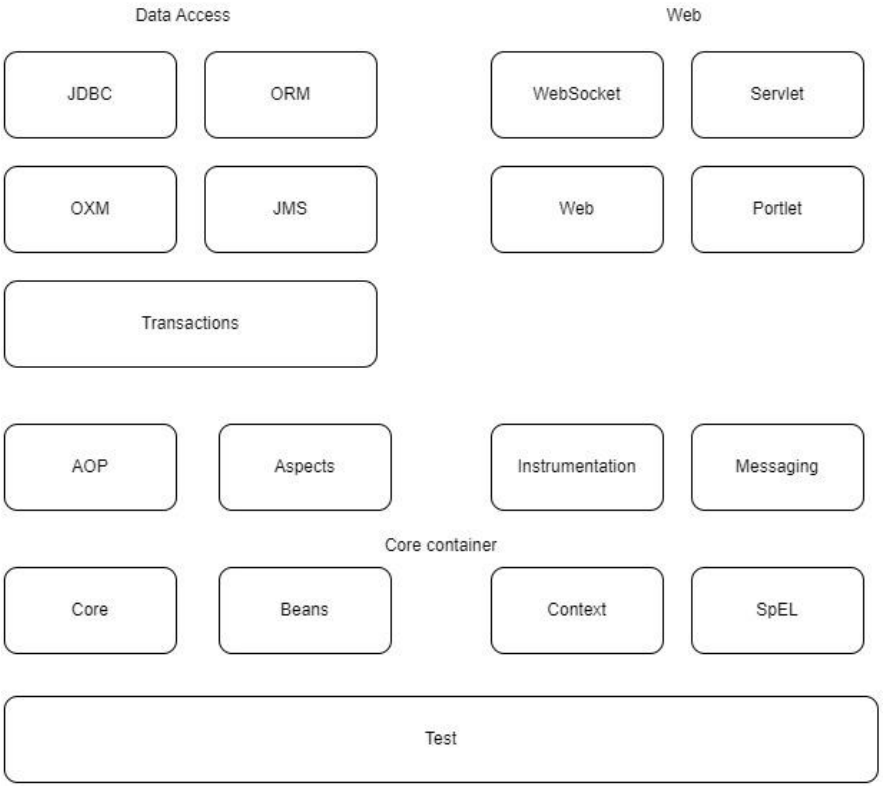


Рисунок 1.4 - Архітектура Spring

Існує кілька альтернатив для Spring у світі розробки додатків на Java, які можна використовувати залежно від потреб та вимог проекту.

Java EE (також відома як Jakarta EE) - це стандартна технологія для розробки підприємницьких додатків на Java. Включає в себе різні специфікації, такі як Servlet, JPA, EJB, інші. Реалізації включають GlassFish, WildFly, і інші.

MicroProfile - це набір специфікацій для розробки мікросервісів на Java. Цей стандарт розширює Java EE та надає підтримку для реактивного програмування та інших сучасних технологій. Реалізації включають Helidon, Quarkus, і інші. [5]

Grails - це фреймворк для розробки веб-додатків на основі Groovy, який працює на платформі JVM. Він використовує Spring під капотом та надає спрощений спосіб розробки додатків.

Play Framework - це фреймворк для розробки веб-додатків та API на Java та Scala. Він популярний серед розробників, які шукають реактивні та асинхронні рішення.

Vert.x - це реактивний та асинхронний фреймворк для розробки мікросервісів та додатків на Java та інших мовах. Він надає можливості для роботи з багатьма паралельними запитами.

Dropwizard - це фреймворк для розробки RESTful веб-служб на Java. Він включає в себе вбудований сервер Jetty та багато інших бібліотек для розробки.

Ninja Framework - це легкий фреймворк для створення веб-додатків на Java. Він використовує Guice для інверсії управління та надає простий спосіб розробки.

Ratpack - це асинхронний фреймворк для розробки високопродуктивних веб-додатків на Java. Він використовує Netty для обробки запитів.

Micronaut Framework - це модерний фреймворк для розробки додатків на Java та інших мовах для виконання на віртуальній машині Java (JVM)

Google Guice - це фреймворк для інверсії управління (IoC, Inversion of Control) та внедрення залежностей (DI, Dependency Injection) для додатків, написаних на Java. Він розроблений Google і надає розробникам інструменти для керування залежностями та конфігурацією компонентів додатку

Нижче буде наведено порівнення технічних можливотсей найпопулярніших конкурентів Spring.

Micronaut Framework та Spring - це два популярних фреймворки для розробки додатків на Java. Обидва фреймворки мають свої особливості та переваги, і вибір між ними може залежати від ваших конкретних потреб та завдань. Ось порівняння між ними:

Реактивність та асинхронність:

Micronaut: Micronaut побудований навколо реактивного програмування і надає вбудовану підтримку асинхронних операцій. Він є ідеальним вибором для створення високопродуктивних та реактивних додатків.

Spring: Spring також підтримує асинхронність, але Spring Boot (як частина Spring-екосистеми) не надає таку вбудовану реактивність. Для реактивних додатків зазвичай використовують Spring WebFlux.

Час запуску:

Micronaut: Micronaut відомий своєю швидкістю запуску додатків, завдяки

оптимізаціям, які дозволяють розробникам швидше розпочати роботу.

Spring: Spring Boot також має досить швидкий час запуску, але в порівнянні з Micronaut, він може бути трохи повільнішим. [6]

Розмір бінарних файлів:

Micronaut: Micronaut має дуже компактні бінарні файли, що дозволяє зменшити обсяг додатків і прискорити розгортання.

Spring: Бінарні файли Spring Boot можуть бути більшими у порівнянні з Micronaut, оскільки Spring надає більше функціональності.

Інверсія управління та внедрення залежностей:

Micronaut: В Micronaut вбудована система інверсії управління та внедрення залежностей, яка використовує анотації та статичний аналіз для створення бінів та внедрення залежностей.

Spring: Spring також використовує систему інверсії управління та внедрення залежностей з великою кількістю анотацій та конфігурацій.

Підтримка мов:

Micronaut: Micronaut підтримує Java, Kotlin і Groovy, що дає вам вибір мови для розробки.

Spring: Spring також підтримує Java, Kotlin, Groovy та інші мови, але Kotlin став особливо популярним у спільноті Spring.

Легкість тестування:

Micronaut: Micronaut надає інструменти для легкого тестування компонентів і заміщення залежностей.

Spring: Spring також підтримує тестування, і в нього є багато інструментів для модульного, інтеграційного та взаємодійного тестування.

Екосистема та підтримка:

Micronaut: Має меншу екосистему та менше спільноти, порівняно з Spring. Проте вона активно розвивається і набирає популярність.

Spring: Spring має велику та добре встановлену екосистему, багато документації та підтримки, що робить його привабливим вибором для великих підприємств.

Обидва фреймворки мають свої переваги і недоліки, і вибір між ними повинен залежати від конкретних потреб та характеристик вашого проекту.

Таблиця 1.1 Порівняльна характеристика фреймверків

METRIC	**MICRONAUT 2.0 M2**	**QUARKUS 1.3.1**	**SPRING 2.3 M3**
Compile Time	1.48s	1.45s	1.33s
Test Time ./mvn test	4.3s	5.8s	7.2s
Start Time Dev Mode	420ms	866ms (1)	920ms
Start Time Production java -jar myjar.jar	510ms	655ms	1.24s
Time to First Response	960ms	890ms	1.85s
Requests Per Second (2)	79k req/sec	75k req/sec	75k req/sec
Request Per Second - Xmx18m	50k req/sec	46k req/sec	46k req/sec
Memory Consumption After Load Test (-Xmx128m) (4)	290MB	390MB	480MB
Memory Consumption After Load	249MB	340MB	430MB

Таким чином, зважаючи на порівняння між Spring тай його прямими

конкурентами, ось деякі з переваг Spring, які можуть бути важливі для розгляду цієї магістерської роботи:

Велика спільнота та підтримка: Spring має широку та активну спільноту розробників та велику кількість ресурсів, документації і статей. Це дозволяє отримати підтримку та відповіді на питання швидше, а також забезпечує більшу надійність фреймворку завдяки великій кількості використовувачів.

Екосистема: Spring має багато різноманітних модулів та пакетів, які допомагають у розробці різних типів додатків. Наприклад, Spring Security для безпеки, Spring Data для роботи з базами даних та багато інших.

Зручність управління залежностями: Spring Framework має потужний контейнер інверсії управління та внедрення залежностей, який дозволяє зручно та гнучко управляти компонентами та їх залежностями.

Забезпечена підтримка інших мов програмування: Spring підтримує не тільки Java, а також Kotlin та Groovy, що дозволяє вибирати мову програмування в залежності від ваших уподобань.

Широкі можливості тестування: Spring Framework надає великий набір інструментів для тестування, включаючи модульні, інтеграційні та взаємодійні тести, що спрощує процес тестування додатків.

Широкі можливості інтеграції: Spring має вбудовані рішення для інтеграції з іншими технологіями, такими як бази даних, месенджери, інші стеки технологій та середовища.

Стандартна платформа для підприємств: Spring є стандартом у світі розробки підприємницьких додатків на Java, і багато підприємств використовують його для розробки великих та важливих додатків.

Загалом, Spring є важливим і надійним гравцем у світі розробки на Java, і його екосистема та підтримка роблять його відмінним вибором для великих та складних проектів

1.3. Vue.js дослідження архітектурних особливостей та порівняльна характеристика з його аналогами

Vue.js - це модерний фреймворк для розробки користувацького інтерфейсу на стороні клієнта. Основні характеристики та переваги включають:

Простота та легкість вивчення: Vue JS легко вивчається і швидко інтегрується в проекти завдяки своєму простому API та документації.

Компонентна архітектура: Vue сприяє побудові користувацького інтерфейсу з використанням компонентів, що полегшує управління і підтримку коду.

Реактивність: Vue JS надає реактивність в HTML-шаблонах, що дозволяє автоматично оновлювати інтерфейс при зміні даних.

Інструменти розробника: Існують різноманітні інструменти розробника, такі як Vue Devtools, які полегшують відлагодження та аналіз коду.

Основні архітектурні особливості Vue.js включають наступне:

Компонентна архітектура: Однією з ключових особливостей Vue.js є його компонентна архітектура. Веб-сторінка розбивається на невеликі, незалежні компоненти, які можна повторно використовувати. Кожен компонент має свій власний стан, логіку та шаблон, і їх легко комбінувати для створення складних інтерфейсів.

Декларативний підхід до UI: Vue.js використовує декларативний підхід до опису користувацького інтерфейсу. Розробник описує, як повинен виглядати інтерфейс в певний момент часу, і Vue.js автоматично оновлює його відповідно до змін у даних та стані додатка.

Реактивність: Vue.js використовує систему реактивності для відстеження змін у даних та автоматичного оновлення відображення. Це дозволяє підтримувати стан інтерфейсу синхронізованим з даними і реагувати на події користувача.

Двостороннє зв'язування даних: Vue.js підтримує двостороннє зв'язування даних, що дозволяє автоматично оновлювати дані в моделі, коли вони змінюються на стороні відображення та навпаки. Це полегшує взаємодію між даними та інтерфейсом користувача.

Вирішення проблеми з областями видимості (Scoped CSS): Vue.js надає можливість використовувати CSS, який обмежений областю компонента, що дозволяє уникнути конфліктів та забезпечити ізольованість стилів.

Маршрутизація та керування станом: Vue Router - це офіційний пакет для маршрутизації в Vue.js, який дозволяє легко визначати маршрути та керувати станом додатка.

Екосистема доповнень: Vue.js має багато доповнень та сторонніх бібліотек, які розширюють його можливості. Наприклад, Vuex - це бібліотека для керування станом додатка, інтегрована з Vue.js.

Документація та спільнота: Vue.js має добре документований API та широку спільноту розробників, що допомагає розробникам швидко навчатися та вирішувати проблеми. [7]

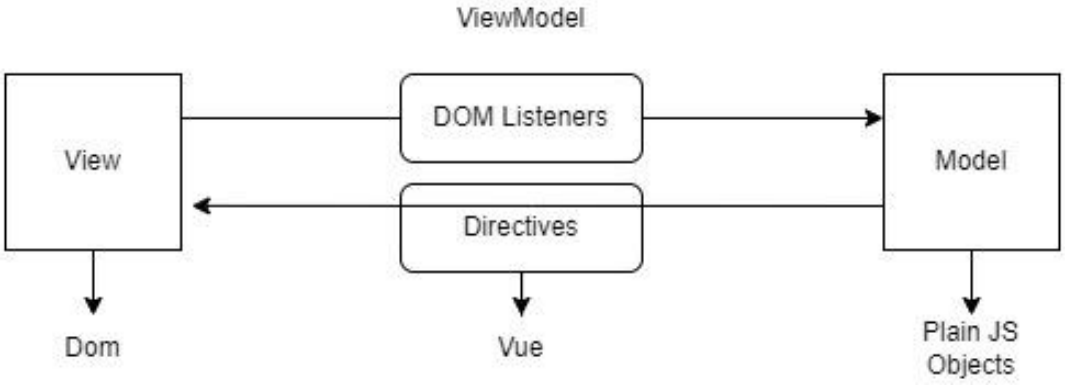


Рисунок 1.5 - Структура Vue.js додатка

Найпопулярнішими конкурентами Vue.js в сфері розробки фронтенду є:

React: Розроблений Facebook, React є одним з найпопулярніших фреймворків для створення інтерфейсів користувача. Він використовує віртуальний DOM та компонентну архітектуру, що дозволяє створювати високопродуктивні та масштабовані додатки.

Angular: Розроблений Google, Angular є повноцінним фреймворком для створення великих та складних веб-додатків. Він надає розширену функціональність та вбудовану підтримку для розробки SPA (Single Page Application).

Svelte - це новий підхід до розробки веб-додатків, який компілює код в чистий JavaScript під час збірки. Він відрізняється від інших фреймворків, так як не використовує віртуальний DOM і має низьку вагу.

Ember - це фреймворк з високим рівнем абстракції для розробки веб-додатків. Він надає велику кількість конвенцій та вбудовану підтримку для розробки SPA.

Aurelia - це фреймворк, який прагне бути простим та легким для використання. Він підтримує двостороннє зв'язування даних та має компонентну архітектуру.

Inferno - це швидкий фреймворк для створення веб-інтерфейсів. Він використовує віртуальний DOM та має низьку вагу.

Mithril - це мінімалістичний фреймворк, який зосереджується на швидкості та простоті. Він підтримує віртуальний DOM і має компонентну модель.

Preact - це легкий альтернативний фреймворк, сумісний з React, але з меншою вагою та швидкістю.

Нижче наведено порівняльну характеристику між Vue.js та React, основного конкуренту серед фреймверків. [8]

Архітектура:

Vue.js має компонентну архітектуру, де інтерфейс розбивається на невеликі компоненти з власним станом та шаблонами.

React також використовує компонентну архітектуру, де кожен компонент має власний стан та відображення.

Віртуальний DOM:

Vue.js використовує віртуальний DOM для ефективного оновлення сторінки та уникнення зайвого перерендерингу.

React також використовує віртуальний DOM для оптимізації оновлень та покращення продуктивності.

Спосіб опису інтерфейсу:

В Vue.js ви описуєте інтерфейс за допомогою шаблонів HTML, що може бути більш зрозумілим для дизайнерів та розробників.

У React ви описуєте інтерфейс за допомогою JSX, який є розширенням

JavaScript. Це дає більшу гнучкість, але може бути складнішим для початківців.

Управління станом:

Vue.js надає вбудовану систему управління станом компонентів, включаючи підтримку двостороннього зв'язування даних.

У React ви можете вибирати більше підходів до управління станом, такі як використання локального стану, Redux або Mobx для глобального стану.

Розширюваність та екосистема:

Vue.js має багато офіційних і сторонніх бібліотек і доповнень для різних задач розробки.

React також має широку екосистему, але для деяких завдань може знадобитися більше налаштувань та сторонніх бібліотек.

Спільнота та підтримка:

Vue.js має швидко зростаючу спільноту та активних розробників, але менше порівняно з React.

React має велику та дуже активну спільноту з багатьма ресурсами та сторонніми бібліотеками.

Швидкість і продуктивність:

Vue.js часто відзначається за свою високу продуктивність завдяки компактному розміру та швидкому часу розгортання додатків.

React також досить продуктивний, але потребує деякої конфігурації для досягнення максимальної продуктивності.

Vue.js ідеально вписується в архітектуру REST (Representational State Transfer) через свою природню спрямованість на розробку інтерфейсів користувача та взаємодію з веб-службами за допомогою HTTP запитів. Вотчери (watchers) та директиви Vue.js дозволяють створювати реактивні інтерфейси, які автоматично оновлюються при зміні даних на сервері. Ось детальніша характеристика та області застосування Vue.js в архітектурі REST:

Оптимізовані HTTP запити: Vue.js легко поєднується з HTTP бібліотеками, такими як Axios або Fetch, для здійснення HTTP запитів до сервера REST API. Ви можете визначати HTTP запити у ваших компонентах та відстежувати їхній стан за

допомогою асинхронних запитів.

Реактивна взаємодія з даними: Vue.js надає можливість автоматичного оновлення інтерфейсу користувача при зміні даних, отриманих з сервера. Завдяки використанню віртуального DOM та реактивності, зміни в даних автоматично відображаються в інтерфейсі без необхідності вручну оновлювати DOM.

Реактивні компоненти: Vue.js дозволяє створювати реактивні компоненти, які відображають дані, отримані з сервера, та надають користувачам можливість взаємодіяти з ними. Це дозволяє створювати інтерактивні додатки, які автоматично реагують на події користувача.

Управління станом: Vue.js надає зручні інструменти для управління станом додатка та компонентів. Ви можете використовувати Vuex, бібліотеку для керування станом, для збереження та керування даними, отриманими з сервера.

Маршрутизація: Vue Router - офіційна бібліотека для маршрутизації в Vue.js. Вона дозволяє визначати маршрути та навігацію у додатку та інтегрується з архітектурою REST API.

Компонентна архітектура: Завдяки компонентній архітектурі Vue.js, ви можете створювати компоненти, які відображають окремі об'єкти або ресурси з REST API. Це полегшує реорганізацію та повторне використання компонентів.

Розширення та плагіни: Vue.js має багато розширень і плагінів, які полегшують взаємодію з REST API та додавання нових функцій до додатків.

Vue.js часто використовується для розробки сучасних односторінкових додатків (SPA) та інтерфейсів користувача, які взаємодіють з великою кількістю даних на стороні сервера. Він підходить для створення адміністративних панелей, соціальних мереж, електронних торгових платформ, аналітичних панелей та інших веб-додатків, де необхідна реактивність та зручність у взаємодії з REST API.

Vue.js та Spring Boot - це два різних технологічних стеки, які часто використовуються в спільності для створення сучасних веб-додатків. Взаємодія між цими двома компонентами може призвести до створення потужних та ефективних додатків з наступними перевагами:

Розділення обов'язків: Vue.js відповідає за клієнтську сторону додатка, тобто

за інтерфейс користувача та взаємодію з ним, в той час як Spring Boot відповідає за серверну частину, обробку запитів, зберігання та доступ до даних. Це дозволяє розділити обов'язки та полегшує розробку та підтримку додатку.

Спрощена інтеграція: Vue.js легко інтегрується з серверним додатком Spring Boot через HTTP запити, зазвичай за допомогою REST API. Це дозволяє створювати односторінкові додатки, які спілкуються з сервером для отримання та зберігання даних.

Реактивність: Vue.js надає можливість створювати реактивні інтерфейси, які автоматично оновлюються при зміні даних на сервері. Spring Boot може надавати дані через REST API, і Vue.js динамічно оновлює відображення на клієнтській стороні без необхідності оновлення сторінки. [9]

Швидкість розробки: Spring Boot дозволяє розробникам швидко створювати серверну частину додатка за допомогою конвенцій та автоматичного конфігурування. Vue.js, з іншого боку, надає зручний інструментарій для створення клієнтського інтерфейсу. Це полегшує та прискорює розробку додатку в цілому.

Масштабованість: Спільне використання REST API між Vue.js та Spring Boot дозволяє легко масштабувати додаток. Ви можете додавати нові функціональності та компоненти на клієнтській стороні та сервері без впливу на існуючі частини додатку.

Підтримка спільноти та інструментів: Як Vue.js, так і Spring Boot мають активні спільноти розробників та велику кількість ресурсів, бібліотек та інструментів для спрощення розробки та підтримки додатків.

Загалом, взаємодія Vue.js та Spring Boot дозволяє розробникам створювати сучасні, реактивні та ефективні веб-додатки з оптимальним розділенням завдань між клієнтською та серверною частинами. Це полегшує розробку, підтримку та масштабування додатку, забезпечуючи високу продуктивність та реактивність для користувачів.

Нижче наведено схему типового REST API з використанням цих двох технологій.

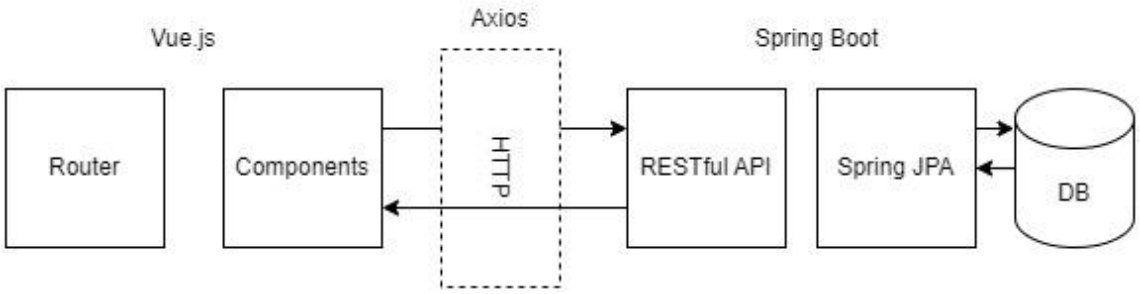


Рисунок 1.6 - Схема типового Vue.js + Spring Boot застосунку

Таким чином, Vue.js має кілька важливих переваг, які роблять його привабливим для побудови REST API та розробки сучасних веб-додатків. Ось розгорнуті висновки щодо переваг Vue.js порівняно з конкурентами:

Простота та зрозумілість: Vue.js славиться своєю простотою та зрозумілістю. Його синтаксис легко освоювати, що робить його ідеальним вибором для початківців та розробників, які шукають швидкий старт.

Реактивність: Vue.js володіє потужною системою реактивності, яка дозволяє автоматично оновлювати інтерфейс користувача при зміні даних. Це робить Vue.js ідеальним для створення динамічних та інтерактивних додатків, які вимагають живого оновлення.

Компонентна архітектура: Vue.js побудований на основі компонентної архітектури, яка дозволяє створювати невеликі та повторно використовувані компоненти. Це полегшує розробку та підтримку додатків.

Широкий спектр бібліотек і розширень: Vue.js має велику кількість офіційних та сторонніх бібліотек і розширень, які полегшують розробку і додають новий функціонал до додатків.

Інтеграція з іншими технологіями: Vue.js легко інтегрується з іншими технологіями, включаючи серверні фреймворки, такі як Spring Boot. Він може легко спілкуватися з REST API, що дозволяє створювати клієнтські інтерфейси для серверних додатків.

Швидкість і продуктивність: Vue.js відомий своєю високою продуктивністю та швидкістю завдяки використанню віртуального DOM та компактному розміру.

Багатоцільовість: Vue.js може використовуватися як для розробки SPA

(Single Page Application), так і для традиційних мультисторінкових додатків, що робить його гнучким рішенням для різних проектів.

Загалом, Vue.js володіє багатьма перевагами, які роблять його конкурентоспроможним та привабливим вибором для розробки REST API та веб-додатків. Його зрозумілість, реактивність, компонентна архітектура та інші переваги роблять його ідеальним інструментом для створення сучасних додатків з живим інтерфейсом

1.4 Дослідження архітектури баз даних, порівняння SQL та NoSQL DB

Архітектурні принципи та особливості SQL та NoSQL баз даних відрізняються через їх різні підходи до зберігання та обробки даних:

SQL (Structured Query Language) Бази Даних:

Реляційна модель даних: SQL бази даних використовують реляційну модель даних, де дані організовані у вигляді таблиць (реляційних баз) з визначеними стовпцями та зв'язками між таблицями. Ця модель надає структурованість та послідовність для даних.

ACID транзакції: SQL бази даних надають сильну підтримку транзакцій, які відповідають принципам ACID (Atomicity, Consistency, Isolation, Durability). Це забезпечує консистентність та надійність даних.

Схема даних: SQL бази даних вимагають жорсткої схеми даних, де структура таблиць та типи даних повинні бути передбаченими та фіксованими заздалегідь. Зміни схеми можуть бути складними.

Мова запитів SQL: Для взаємодії з SQL базами даних використовується SQL, мова запитів, яка дозволяє виконувати складні запити для отримання та зміни даних.

Нижче наведено типову схему SQL DB

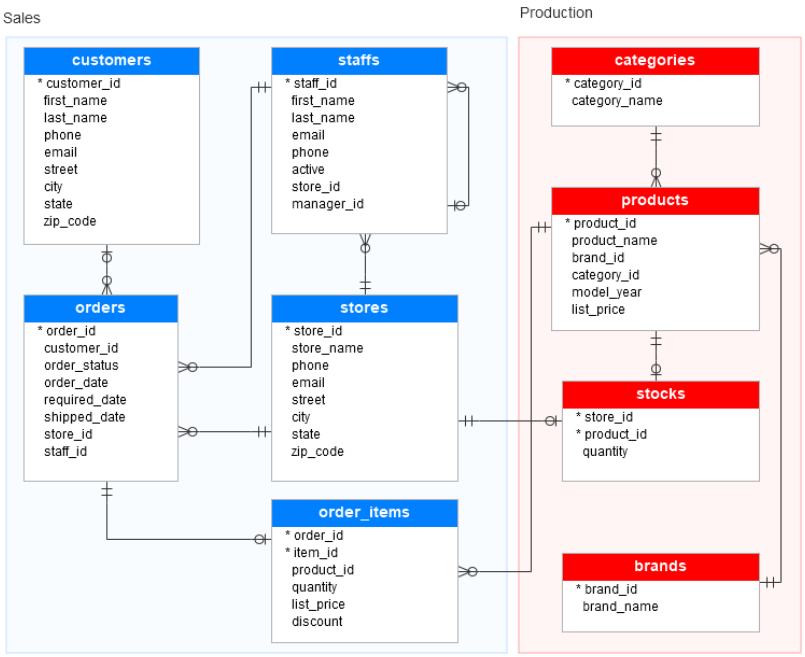


Рисунок 1.7 - Типова схема SQL DB [10]

NoSQL (Not Only SQL) Бази Даних:

Різноманітність моделей даних: NoSQL бази даних включають в себе різні моделі даних, такі як документальні (наприклад, MongoDB), стовпчасті (наприклад, Apache Cassandra), ключ-значення (наприклад, Redis), та графові (наприклад, Neo4j). Це дозволяє вибрати підходящу модель для конкретного типу даних.

Без схеми (schemaless) або динамічна схема: NoSQL бази даних дозволяють зберігати дані без жорсткої схеми або з динамічною схемою, де структура даних може змінюватися з часом без значущих обмежень.

Швидкодія та масштабованість: NoSQL бази даних зазвичай володіють високою швидкодією та здатністю до горизонтального масштабування, що робить їх ідеальними для обробки великих обсягів даних та навантаження.

BASE транзакції: NoSQL бази даних використовують принципи BASE (Basically Available, Soft state, Eventually consistent), що надає більшу гнучкість за рахунок послабленої консистентності.

Мова запитів:

Кожен вид NoSQL бази даних може використовувати власну мову запитів або API для взаємодії з даними.

Наприклад, MongoDB використовує мову запитів, яка подібна до JSON.

Огляд архітектурних принципів та особливостей SQL та NoSQL баз даних демонструє, що кожен тип бази даних має свої переваги та обмеження, і вибір між ними залежить від конкретних вимог та завдань проекту. SQL бази даних підходять для структурованих даних та додатків з високими вимогами до консистентності, тоді як NoSQL бази даних надають гнучкість та швидкодію для обробки різноманітних типів даних.

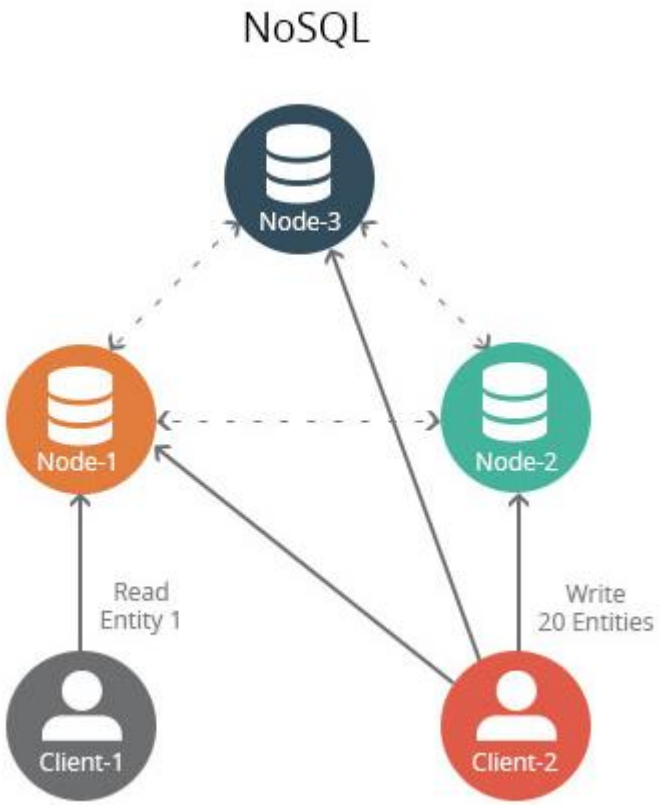


Рисунок 1.8 - Типова схема NoSQL DB [10]

Графові бази даних є одним із видів NoSQL баз даних, які спеціалізуються на зберіганні та обробці даних у формі графів. Графова модель даних ґрунтується на концепції вершин (нод) та ребер (зв'язків) для представлення та відображення структури даних. Основні особливості графових баз даних та їх переваги порівняно з реляційними базами даних включають наступне:

Основні особливості графових баз даних:

Гнучкість моделі даних: Графові бази даних дозволяють легко представляти

та моделювати взаємозв'язки між різними об'єктами. Це особливо корисно для даних, де зв'язки між об'єктами грають важливу роль.

Запити на основі шляхів: Графові бази даних дозволяють виконувати запити на основі шляхів, що дозволяє знаходити шляхи між вершинами в графі. Це корисно для задач, пов'язаних із пошуком найкоротших шляхів, аналізом мереж та іншими подібними завданнями.

Підтримка глибокого вкладення: Графові бази даних підтримують глибоке вкладення даних, що дозволяє моделювати складні структури, такі як ієрархії та мережі.

Швидкодія запитів: Велика частина операцій зчитування та аналізу графових даних виконується ефективно, оскільки бази даних побудовані для роботи зі зв'язками між даними.

Переваги графових баз даних над реляційними:

Ефективність при операціях навігації: Графові бази даних ідеально підходять для операцій навігації по великому обсязу даних та визначення взаємозв'язків між об'єктами.

Масштабованість: Графові бази даних можуть легко масштабуватися горизонтально, додаванням нових серверів або вузлів для обробки даних.

Підтримка слабко структурованих даних: Графові бази даних дозволяють зберігати та обробляти дані зі змінною структурою, що є корисним у сучасних додатках зі змінюючимся вмістом.

Висока продуктивність: Графові бази даних надають швидкість та продуктивність при роботі з великими графовими структурами.

Всі ці особливості роблять графові бази даних важливим інструментом для задач, пов'язаних із моделюванням та аналізом взаємозв'язків між даними, а також для великих проектів, де швидкодія та масштабованість грають важливу роль.

Типову схему графової бази даних наведено нижче.

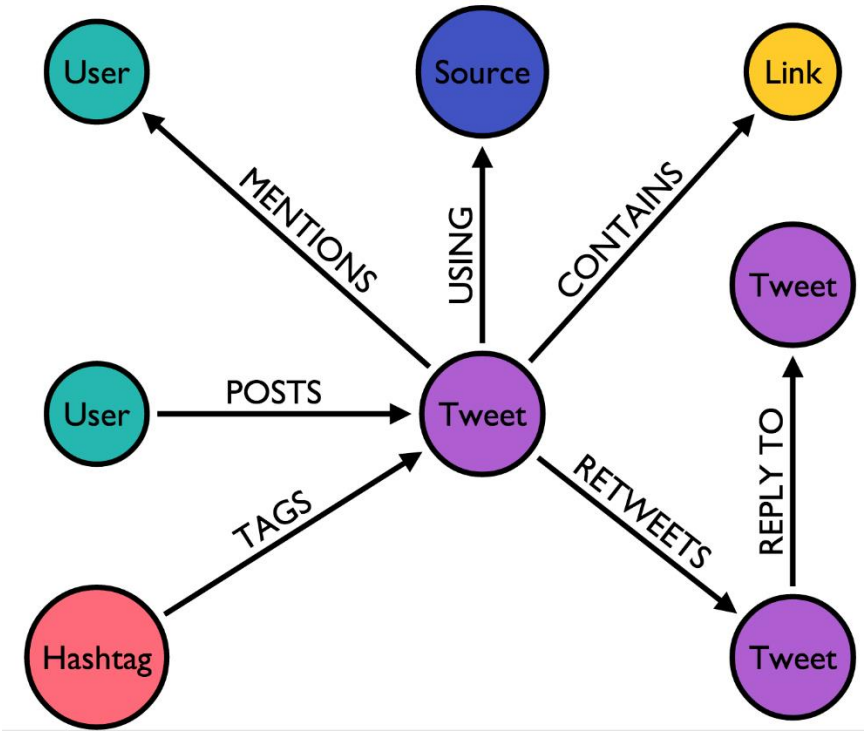


Рисунок 1.9 - Типова схема графової бази даних [10]

В даній магістерській роботі буде використовуватися NoSQL, нижче розглянемо архітектурні особливості однієї з них.

PostgreSQL є потужною та вільно розповсюджуваною реляційною системою управління базами даних (СУБД), яка відзначається багатоманіттям функцій та високою надійністю. Ось докладніше про PostgreSQL:

Архітектура PostgreSQL:

PostgreSQL використовує архітектуру "shared-nothing", що означає, що кожен процес СУБД має власну пам'ять та не ділить її з іншими процесами. Це забезпечує ізоляцію та стійкість до збоїв.

Система підтримує багатопотоковість, тобто може обробляти багато запитів паралельно за допомогою різних процесів та потоків.

В основі PostgreSQL лежить модульність, і велика кількість функцій обробки даних реалізовані як розширення, що дозволяє легко додавати нові функції та типи даних.

Основні переваги PostgreSQL:

Відкритий джерела коду: PostgreSQL є вільним програмним засобом з відкритим вихідним кодом, що робить його доступним для безкоштовного використання та модифікації.

Додаткові функції: PostgreSQL має багатий набір функцій та типів даних, включаючи підтримку геоданих, JSON та XML.

Підтримка ACID: PostgreSQL забезпечує високий рівень надійності та підтримує транзакції згідно з принципами ACID (Atomicity, Consistency, Isolation, Durability).

Підтримка реплікації: PostgreSQL має вбудовану підтримку реплікації, що дозволяє створювати резервні копії та масштабувати систему.

Розширюваність: Систему можна розширювати за допомогою розширень (extensions) та зовнішніх модулів, що дає можливість додавати новий функціонал.

Підтримка процедурної мови: PostgreSQL дозволяє використовувати різні мови програмування для створення збережених процедур та функцій.

Застосування PostgreSQL:

PostgreSQL може бути використаний у багатьох галузях, включаючи веб-розробку, геоінформаційні системи, фінансовий аналіз, аналітику даних, логістику та багато інших сфер. Він часто використовується у великих проектах, де потрібна висока надійність та масштабованість.

PostgreSQL відомий своєю надійністю, розширюваністю та можливістю адаптації до різних завдань, що робить його однією з переваг у світі СУБД.

PostgreSQL було вибрано поміж інших конкурентів, тому що вона має певні переваги, які буде наведено нижче у порівнянні з прямим конкурентом MySQL:

PostgreSQL:

ACID-сумісність: PostgreSQL має вбудовану підтримку транзакцій згідно з принципами ACID (Atomicity, Consistency, Isolation, Durability). Це забезпечує високий рівень надійності та консистентності даних.

Розширюваність: PostgreSQL дозволяє легко масштабувати систему шляхом додавання нових серверів або вузлів. Він підтримує реплікацію, яка дозволяє створювати резервні копії та розподілені системи.

Можливості геоданих: PostgreSQL має вбудовану підтримку геоданих та географічних індексів, що робить його ідеальним для розробки геоінформаційних систем та геододатків.

Розширені функції: PostgreSQL має багатий набір функцій, включаючи підтримку JSON, XML та багато інших типів даних. Ви можете створювати власні функції та типи даних.

Підтримка розширень: PostgreSQL дозволяє додавати розширення та зовнішні модулі, що розширюють можливості бази даних без зміни її основного коду.

MySQL:

Простота використання: MySQL часто вибирають для невеликих проектів та прототипування завдяки його простому використанню та налаштуванню.

Велика спільнота: MySQL має велику спільноту користувачів та активно підтримується багатьма розробниками, що робить його ідеальним для знаходження рішень на форумах та у мережі.

Швидкодія для простих завдань: MySQL може бути ефективним для простих запитів та завдань, зокрема для веб-сайтів з невеликим обсягом даних.

Висока продуктивність для певних завдань: MySQL може виконувати деякі операції швидше, ніж PostgreSQL, за умови, що це не вимагає складних запитів та розширених функцій.

Отже, PostgreSQL має переваги у високій надійності, розширюваності, підтримці геоданих та розширених функцій, що робить його ідеальним вибором для багатьох великих та вимогливих проектів. MySQL, з іншого боку, може бути кращим вибором для менших проектів та завдань, де вимоги до продуктивності не настільки високі.

Нижче наведені порівняльні характеристики тестування декількох баз даних.

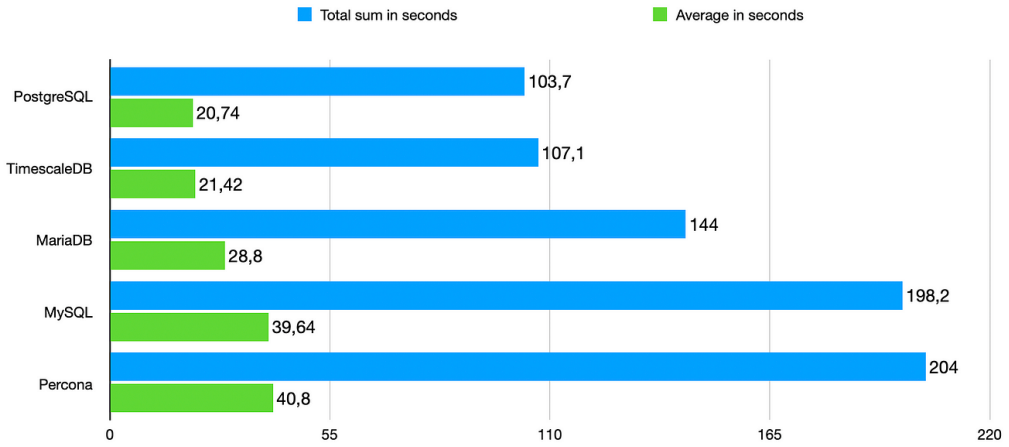


Рисунок 1.10 - Порівняльна характеристика DB [10]

1.5 Аналіз Bootstrap framework. Дослідження комбінації технологій для побудови оптимального RESTful API

Bootstrap - це потужний та популярний фреймворк для розробки веб-додатків та створення користувацького інтерфейсу. Ось детальніше про Bootstrap та його переваги для побудови RESTful API:

Основні характеристики Bootstrap:

HTML, CSS та JavaScript: Bootstrap надає готовий набір HTML-компонентів, CSS-стилів та JavaScript-плагінів, які допомагають створювати привабливий та інтерактивний інтерфейс.

Мобільно-адаптивний дизайн: Bootstrap надає мобільно-адаптивні класи та компоненти, які роблять ваші сторінки інтуїтивно користування на будь-яких пристроях.

Спільнота користувачів: Bootstrap має велику та активну спільноту користувачів та розробників, що означає, що ви можете знайти велику кількість додаткових ресурсів, документації та плагінів.

Зручний для використання: Bootstrap дозволяє швидко створювати сторінки та компоненти за допомогою готових класів та ідентифікаторів.

Переваги Bootstrap для побудови RESTful API:

Швидкість розробки: Bootstrap пропонує готовий набір компонентів, які

можна використовувати для швидкого створення інтерфейсу вашого RESTful API. Це заощаджує час та зусилля розробників.

Консистентність дизайну: Використання Bootstrap допомагає досягти консистентності дизайну на всіх сторінках вашого додатка, що робить його більш професійним та привабливим для користувачів.

Мобільна підтримка: Bootstrap надає готовий механізм мобільної адаптації, що важливо для RESTful API, оскільки користувачі можуть звертатися до нього з різних пристроїв.

Спільнота та розширення: Існує велика кількість додаткових плагінів та розширень, які розробники можуть використовувати для покращення функціональності та вигляду свого RESTful API.

Доступність та підтримка: Bootstrap підтримується та активно розвивається, що гарантує доступність оновлень та підтримку у майбутньому.

Bootstrap - це потужний інструмент для розробки інтерфейсу RESTful API, який допомагає спростити процес створення веб-додатків та забезпечує їх високий рівень професійності та якості.

Рисунок 1.11 - Основні переваги Bootstrap [11]

Bootstrap застосовується в різних сферах та в комбінаціях з різними фреймворками для створення веб-додатків. Ось де його можна використовувати і в комбінаціях, де він може бути особливо ефективним:

Розробка веб-сайтів і веб-додатків: Bootstrap є ідеальним фреймворком для

створення сучасних та інтерактивних веб-сайтів та додатків. Він допомагає розробникам створювати сторінки швидко та ефективно, забезпечуючи їх мобільною адаптацією.

Адміністративні панелі: Bootstrap часто використовується для створення адміністративних панелей та інтерфейсів управління для веб-додатків. Його готові компоненти дозволяють легко розробляти інтерфейси для керування даними та налаштуваннями.

Інтерфейси користувача для мобільних додатків: Bootstrap може бути використаний для створення користувацьких інтерфейсів для мобільних додатків, особливо для гібридних додатків, які використовують веб-технології.

Інтеграція з іншими фреймворками: Bootstrap може бути використаний разом з іншими фреймворками та бібліотеками, такими як React, Angular або Vue.js, для створення повноцінних веб-додатків. Це особливо корисно, коли вам потрібно поєднати можливості Bootstrap із функціональністю цих фреймворків.

Розробка прототипів: Bootstrap часто використовується для швидкої розробки прототипів веб-додатків. Його готові компоненти та стилі допомагають створити інтерактивні прототипи для оцінки концепцій та дизайну.

Створення лендінг-сторінок: Bootstrap добре підходить для створення лендінг-сторінок та односторінкових сайтів, оскільки він надає готові блоки та компоненти для презентації інформації та залучення користувачів.

У комбінаціях з фреймворками, такими як React, Angular або Vue.js, Bootstrap може бути особливо ефективним для розробки веб-додатків, оскільки він доповнює їхні можливості створення користувацьких інтерфейсів і допомагає зробити їх більш привабливими та швидкими.

Зважаючи на популярність та ефективність кожної технології було обрано 4 схеми за якими може буде побудованих веб-додаток.

Vue.js + Spring Boot + PostgreSQL:

Vue.js - це потужна бібліотека для створення інтерактивних користувацьких інтерфейсів. Його головна перевага полягає в простоті використання та гнучкості при розробці клієнтської частини веб-додатка. Vue.js надає інструменти для

створення відзивчивих та інтуїтивних інтерфейсів.

Spring Boot - це потужний фреймворк для створення серверної частини веб-додатків. Він надає швидкий спосіб створення RESTful API, автоматизуючи багато завдань, таких як конфігурація та керування залежностями.

PostgreSQL - це реляційна база даних, яка відзначається високою надійністю та розширюваністю. Вона ідеально підходить для зберігання даних виконавчої частини вашого додатка.

Переваги комбінації: Ця комбінація дозволяє створити повний стек для розробки веб-додатків. Vue.js забезпечує добру взаємодію з користувачем, Spring Boot робить створення RESTful API легким та швидким, а PostgreSQL гарантує надійність та безпеку зберігання даних. Такий стек часто використовується для створення сучасних веб-додатків.

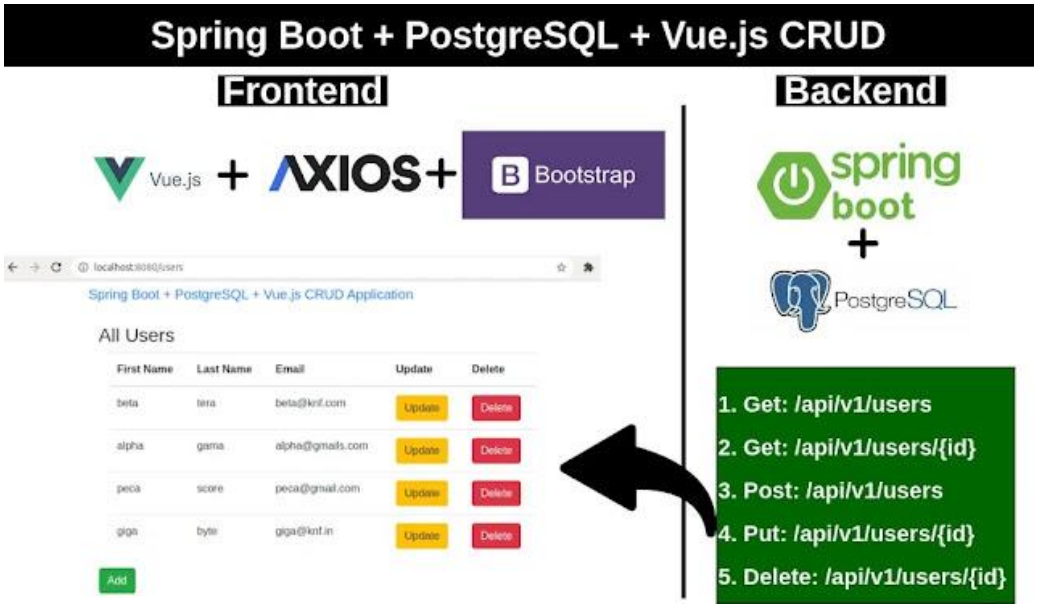


Рисунок 1.12 - Комбінована схема Vue.js + Spring Boot + PostgreSQL [11]

React + Spring Boot + PostgreSQL:

React - це інша популярна бібліотека для створення користувацьких інтерфейсів. Вона також надає ефективні інструменти для розробки клієнтської частини додатка.

Spring Boot: Тут використовується той самий Spring Boot для створення серверної частини.

PostgreSQL: Аналогічно, PostgreSQL залишається базою даних за замовчуванням.

Переваги комбінації: React та Spring Boot - це ще одна потужна комбінація для розробки веб-додатків, яка відома своєю продуктивністю та розширюваністю.

Angular + Spring Boot + PostgreSQL:

Angular - це фреймворк для створення великих та складних веб-додатків. Він надає багато вбудованих функцій та можливостей для створення потужних клієнтських додатків.

Spring Boot залишається обраною платформою для створення серверної частини.

PostgreSQL: Тут також використовується PostgreSQL для зберігання даних.

Переваги комбінації: Angular забезпечує структурованість та розширюваність клієнтської частини, що може бути корисним для великих проектів.

Vue.js + Spring Boot + MongoDB:

Vue.js: Тут знову використовується Vue.js для клієнтської частини.

Spring Boot залишається серверною платформою.

MongoDB - це документ-орієнтована NoSQL база даних, яка підходить для зберігання невеликих та середніх обсягів даних з гнучкою схемою.

Переваги комбінації: Використання MongoDB дозволяє працювати з документами без жорсткої схеми, що може бути корисним для додатків зі змінними даними.

Загалом, у всіх цих комбінаціях Spring Boot використовується як серверна платформа, оскільки він надає швидкий спосіб створення RESTful API. Vue.js, React, Angular та MongoDB або PostgreSQL можуть бути вибрані в залежності від специфічних вимог та власних уподобань розробників. Зокрема, Vue.js і Spring Boot славляться своєю простотою та продуктивністю, що робить їх популярними варіантами для розробки RESTful API.

Одже, для реалізації проекту було обрано таку схему технологій: Vue.js + Spring Boot + PostgreSQL + Bootstrap. Ця комбінація технологій має ряд значущих

переваг, які роблять її оптимальним вибором для створення RESTful API та веб-додатків. Основні переваги цієї схеми включають:

Швидкість розробки: Використання Vue.js та Bootstrap дозволяє швидко створювати інтерактивний та стильний користувацький інтерфейс для додатків. Vue.js надає зручний фронтенд, а Bootstrap - готові компоненти та стилі для дизайну.

Спрощення розробки серверної частини: Spring Boot є потужним фреймворком для створення RESTful API з великою кількістю вбудованих функцій та інструментів. Це дозволяє розробникам швидко створювати серверну частину додатків.

Мобільна адаптивність: Bootstrap забезпечує мобільну адаптивність для створених сторінок, що важливо в епоху мобільних пристроїв, коли користувачі звертаються до додатків з різних пристроїв.

Надійність і безпека: Використання PostgreSQL як реляційної бази даних дозволяє забезпечити надійність та безпеку зберігання даних. Spring Boot надає засоби для захисту RESTful API від атак та злому.

Ця схема технологій дозволить створити ефективний та продуктивний проект з високоякісним користувацьким інтерфейсом, надійним сховищем даних та зручним RESTful API для взаємодії з додатком.

Таким чином, прийняття схеми розробки на базі Vue.js + Spring Boot + PostgreSQL має численні переваги, які можна виокремити в наступний висновок:

Розділення фронтенду і бекенду: Використання Vue.js як фронтенд-фреймворку та Spring Boot як бекенд-фреймворку дозволяє чітко розділити клієнтську та серверну частини додатку. Це сприяє відокремленому розвитку, облегшує тестування та підтримку.

Ефективна розробка фронтенду: Vue.js надає інструменти для швидкої та продуктивної розробки користувацького інтерфейсу. Компонентна архітектура Vue.js дозволяє створювати перевикористовувані компоненти та забезпечує легку інтеграцію з серверним бекендом.

Масштабованість та продуктивність: Spring Boot - це потужний бекенд-фреймворк, який дозволяє створювати ефективні та масштабовані серверні додатки. Використання PostgreSQL як бази даних надає надійність та продуктивність для зберігання даних.

Безпека та автентифікація: Spring Boot має вбудовану підтримку для автентифікації і авторизації, що дозволяє легко забезпечити безпеку вашого додатку. Ви можете інтегрувати різні механізми безпеки, такі як OAuth2 або JWT.

Підтримка стандартів RESTful API: Spring Boot підтримує створення RESTful API, що є стандартним підходом до взаємодії клієнтів та серверів. Це сприяє легкому взаємодії між фронтендом і бекендом.

Сумісність і розширюваність: Vue.js, Spring Boot і PostgreSQL - це популярні та добре підтримувані технології з великою спільнотою розробників. Це робить їх ідеальними виборами для довгострокових проектів, які можна легко розширювати та підтримувати.

Загалом, поєднання Vue.js, Spring Boot та PostgreSQL надає зручну та потужну архітектуру для розробки веб-додатків, забезпечуючи швидку розробку, надійну роботу та високий рівень безпеки. Ця схема дозволяє командам розробників реалізувати різноманітні проекти, від невеликих стартапів до великих підприємств

2 ДОСЛІДЖЕННЯ SPRING BOOT ФРЕЙМВЕРКА ДЛЯ ПОБУДОВИ RESTFUL API ТА РОЗРОБКА BACKEND СХЕМИ ДОДАТКУ

2.1 Аналіз стеку технологій який було застосовано для побудови серверу додатку. Spring MVC

Сервер, також відомий як бекенд, - це складова архітектури додатка, яка відповідає за обробку запитів від клієнтів, виконання бізнес-логіки, доступ до бази даних і надання відповідей клієнтам. В основному, бекенд є центральною частиною додатка і відповідає за такі основні завдання:

Обробка запитів: Сервер приймає HTTP-запити від клієнтів (наприклад, браузерів, мобільних додатків) і відповідає на них. Запити можуть бути різного типу, такі як отримання сторінок веб-сайту, збереження даних, оновлення інформації та багато інших.

Бізнес-логіка: Бекенд містить бізнес-логіку додатка, яка визначає, як додаток повинен взаємодіяти з даними та виконувати конкретні завдання. Це може включати в себе обчислення, перевірку прав доступу, аналіз даних і інші операції, які забезпечують функціональність додатка.

Збереження та доступ до даних: Бекенд взаємодіє з базою даних або іншими сховищами даних для збереження та отримання інформації. Він керує доступом до даних, виконує запити до бази даних і обробляє результати.

Аутентифікація та безпека: Бекенд відповідає за аутентифікацію та авторизацію користувачів. Він забезпечує безпеку додатка, контролюючи доступ до ресурсів і запобігаючи несанкціонованому доступу.

Забезпечення інтеграції: Бекенд може взаємодіяти з іншими службами та зовнішніми джерелами даних, щоб отримати необхідну інформацію. Це може включати взаємодію з іншими мікросервісами, сторонніми API або системами.

Взаємодія з клієнтами: Бекенд надає відповіді клієнтам у вигляді даних або HTML-сторінок. Він генерує відповіді, які можуть бути відображені користувачам на їхніх пристроях.

Бекенд є важливою частиною багатьох додатків, особливо для веб-додатків та мобільних додатків. Він відповідає за надання функціональності, збереження даних та забезпечення безпеки. Комбінація бекенду та фронтенду (клієнтської частини) дозволяє створювати повноцінні додатки, які задовольняють потреби користувачів та підприємств.

Основні результати наукових досліджень були наведені в [11]

В першому розділі було досліджені найпопулярніші варіанти для побудови бекенду та було обрано Spring Boot, розглянемо його більш ретельно.

Spring та Spring Boot - це обидва фреймворки для розробки Java-додатків, проте вони використовуються в різних сценаріях та мають свої особливості. Ось докладна різниця між ними та переваги використання Spring Boot:

Spring:

Конфігурація: Spring потребує великої кількості налаштувань і конфігурації, що може бути досить складним завданням.

Додаткові бібліотеки: Для створення повноцінного додатка на базі Spring, вам доведеться вибирати та налаштовувати додаткові бібліотеки для різних завдань, такі як ORM (наприклад, Hibernate), контейнер сервлетів, інтеграція з базами даних і т. д.

Багато коду: Розробка додатка на Spring може вимагати написання багато коду, особливо для налаштування та керування бізнес-логікою.

Складність: Spring - потужний, але більш складний фреймворк, що може вимагати глибокого розуміння його принципів та архітектури.

Spring Boot:

Автоматизація: Spring Boot надає автоматичну конфігурацію, яка дозволяє створювати додатки з меншою кількістю налаштувань та коду.

Вбудовані бібліотеки: В Spring Boot вже включені більшість необхідних бібліотек і фреймворків, так що вам не потрібно вибирати та налаштовувати їх окремо.

Простота: Розробка на Spring Boot зазвичай вимагає менше коду і менше зусиль для налаштування.

Швидкість старту: Spring Boot має швидший час старту додатку, що полегшує розробку та тестування.

Інтеграція з іншими інструментами: Spring Boot підтримує багато зручних інструментів, таких як Spring Initializr для швидкого створення проекту, а також можливість використовувати вбудовані сервери, такі як Tomcat або Jetty.

Чому краще використовувати Spring Boot:

Spring Boot є відмінним вибором для багатьох розробників через його спрощену конфігурацію, автоматичну налаштування та зручність у використанні. Він дозволяє розробникам швидко створювати як малі прототипи, так і великі виробничі додатки з меншими зусиллями та меншею кількістю коду. Spring Boot також підтримує багато інших інструментів та технологій, що полегшує інтеграцію з екосистемою Java та різними іншими технологіями

spring	Spring Boot
Spring is the most popular Java EE framework for app building.	Spring Boot is a widely used framework for building REST APIs.
The main goal is to simplify Java EE development. However, Spring presupposes some boilerplate code to execute the minimal task.	The main goal is to deliver the easiest way of web app development and shorten the code length.
Dependency Injection (DI) and **Inversion of Control** (IOC) are the primary features of the Spring framework. Excellent tool to build loosely coupled apps.	**Spring Boot Actuator** delivers production-level features including app monitoring and metrics.
Spring is light-weight, supports the **MVC**, **WebFlux**, **ORM**, and **AOP** modules.	**Easy dependency management**. Spring Boot packages the compatible third-party dependencies for every Spring app type and delivers them to developers via starters.
Dependency management. Developers define the dependencies for the project manually in pom.xml.	**Autoconfiguration**. After choosing the starter, Spring Boot configures the app base on the jar dependencies automatically.
JDBC abstraction layer ensures an exception hierarchy, making error-handling easier.	**Embedded servlet container support**.
Spring supports all types and aspects of app development: web apps, core Java, enterprise apps, distributed apps.	**Spring Initializr**. Using this tool, it's possible to generate a skeleton of a Spring Boot project with any functionality required.
Requirements: Java 8+, Spring 5.0 and above, Servlet 3.0+ container **Testing**: TestContext, Spring MVC Test, mock objects, WebTestClient **Languages**: Groovy, Kotlin, dynamic languages	

Рисунок 2.1 - Порівняльна характеристика Spring та Spring Boot [12]

Основний технологічний стек Spring Boot включає в себе наступні компоненти та інструменти:

Spring Framework: Spring Boot базується на оригінальному Spring Framework і використовує багато його модулів та функцій. Spring Framework забезпечує основу для створення бізнес-логіки та інверсії управління.

Вбудовані сервери: Spring Boot підтримує вбудовані сервери, такі як Apache Tomcat, Jetty, та Undertow. Це дозволяє запускати додатки безпосередньо з фреймворку, без необхідності налаштовувати зовнішні сервери.

Автоматична конфігурація: Spring Boot надає автоматичну конфігурацію, яка дозволяє розробникам уникнути багатьох налаштувань за замовчуванням. Це спрощує процес розробки та дозволяє швидко почати працювати над додатком.

Управління залежностями: Spring Boot використовує систему управління залежностями Maven або Gradle для додавання бібліотек та ресурсів до проекту. Він автоматично вирішує залежності та додає їх до проекту.

Вбудовані бібліотеки: Spring Boot містить багато вбудованих бібліотек для роботи з базами даних, безпеки, веб-розробки, логування та інших аспектів розробки.

Spring Boot Starter: Це спеціальні стартери, які включають готові конфігурації та залежності для певних видів додатків, такі як веб-додатки, додатки для роботи з базами даних, додатки для безпеки і т. д. Вони спрощують старт проекту.

Spring Boot Actuator: Ця бібліотека дозволяє моніторити та управляти додатком в режимі реального часу, надаючи інформацію про стан додатка, журнали, метрики та інше.

Spring Boot DevTools: Інструменти розробки, які дозволяють автоматично перезавантажувати додаток при внесенні змін в код. Це полегшує процес розробки та тестування.

Spring Boot Data: Набір бібліотек для спрощення роботи з базами даних, включаючи Spring Data JPA, Spring Data MongoDB, Spring Data JDBC та інші.

Spring Boot Security: Модуль для налаштування безпеки в додатку, включаючи аутентифікацію та авторизацію користувачів.

Spring Boot Test: Модуль для тестування додатків, який надає різноманітні інструменти для написання та виконання тестів

Більшість з них буде використано в даній магістерській роботі, та описана нижче.

Для того, щоб побудувати максимально продуктивний та потужний додаток необхідно використовувати патерни проектування та архітектурні шаблони, зокрема наш додаток буде побудовано з використанням MVC технології.

MVC (Model-View-Controller) - це архітектурний шаблон, який використовується для розділення компонентів в програмному додатку на три основних частини: Model, View і Controller. Цей шаблон допомагає розподілити логіку програми та відображення даних, що робить програму більш підтримуваною, розширюваною та зручною для розробки.

Ось короткий опис кожної складової MVC:

Model (Модель): Модель представляє собою даний компонент, який відповідає за управління даними та бізнес-логікою додатку. Вона представляє інформацію та операції, які виконуються над даними. Модель не залежить від представлення або контролера і може повідомляти представлення про зміни в даних.

View (Представлення): Представлення відповідає за відображення даних користувачеві. Це може бути вікно програми, веб-сторінка, графічний інтерфейс або будь-який інший спосіб відображення даних користувачеві. Представлення не містить бізнес-логіки, воно лише відображає дані, які надає модель.

Controller (Контролер): Контролер відповідає за обробку запитів користувача та взаємодію з моделлю та представленням. Він приймає вхідні дані від користувача, виконує відповідні операції та оновлює модель та представлення відповідно до запитів користувача.

Розділення програми на ці три компоненти дозволяє забезпечити максимальну гнучкість та розширюваність. Наприклад, якщо ви хочете змінити

спосіб відображення даних, вам не потрібно змінювати бізнес-логіку або контролер, вам просто потрібно оновити представлення. Також, це дозволяє багатьом розробникам спільно працювати над проектом, так як кожен може працювати над однією з компонентів без впливу на інші.

MVC є одним з популярних архітектурних шаблонів у розробці програмного забезпечення і використовується в багатьох платформах та мовах програмування, включаючи Java.

Так як, ми використовуємо Spring Boot для побудови нашої бекенд частини, в нашому проекті буде застосовано Spring MVC (Model-View-Controller).

Spring MVC (Model-View-Controller) - це частина Spring Framework, призначена для розробки веб-додатків на основі архітектурного шаблону MVC. Шаблон MVC використовується для розділення веб-додатку на три основних компоненти: Model, View та Controller, що сприяє розділенню бізнес-логіки, відображення даних і обробки користувацьких запитів.

Основні компоненти Spring MVC:

Model (Модель): Модель в Spring MVC відповідає за управління даними та бізнес-логікою. Це може бути клас Java, який представляє дані, які використовуються в додатку. Модель може бути доступною для контролера для обробки і подальшого відображення у представленні.

View (Представлення): Представлення в Spring MVC представляє собою шаблон або інтерфейс, який відображає дані користувачу. Представлення може бути сторінкою HTML, JSON-відповіддю, XML-документом або іншим способом відображення інформації користувачу.

Controller (Контролер): Контролер в Spring MVC відповідає за обробку HTTP-запитів від користуачів. Він приймає запити, взаємодіє з моделлю для обробки даних та вибору відповідного представлення. Контролер визначає, яку логіку слід виконати при зверненні користувача до певного URL.

Основні особливості Spring MVC:

Централізована конфігурація: Spring MVC надає можливість конфігурувати додаток за допомогою анотацій або XML-конфігурації. Це полегшує створення і налаштування контролерів та представлень.

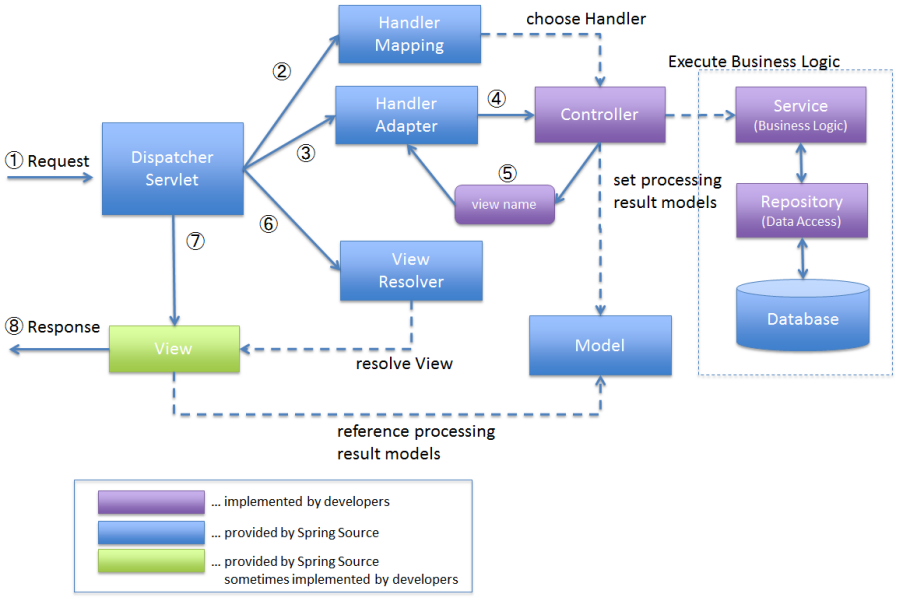


Рисунок 2.2 - Життєвий цикл запиту в Spring MVC [13]

Інтеграція з іншими Spring модулями: Spring MVC легко інтегрується з іншими модулями Spring, такими як Spring Security, Spring Data, Spring Boot і багатьма іншими, що робить його потужним і розширюваним фреймворком.

Управління станом сесії та даними: Spring MVC підтримує управління станом сесії користувача, включаючи роботу з формами та обробку даних.

Підтримка міжнародності: Фреймворк дозволяє легко використовувати різні локалізації та міжнароднізацію додатків.

Spring MVC використовується для створення різноманітних веб-додатків, включаючи класичні веб-сайти, RESTful API, системи керування контентом і багато інших. Він надає розробникам зручний спосіб структурувати свої додатки та робити їх підтримуваними та легко розширюваними.

Використання стеку технологій Spring Boot, який включає Spring MVC, є вибором, який дозволяє досягнути численних переваг у розробці веб-додатків.

Загалом, використання стеку технологій Spring Boot робить розробку веб-додатків більш зручною, ефективною та гнучкою. Цей стек надає можливість

створювати як прості веб-сайти, так і складні системи, і дозволяє розробникам швидко відгукнутися на вимоги користувачів та ринку.

2.2 Вибір системи збірки. Аналіз та вибір основних залежностей для побудови бекенд частини

Розрізняють такі основні системи збірки, як Maven і Gradle, які є популярними інструментами для збирання та управління проектами в різних мовах програмування, зокрема в Java. Ось короткий огляд кожної з них:

Maven:

Тип проектів: Maven спеціалізується на керуванні Java-проектами, але також підтримує інші мови програмування. Він розроблений для структурування та збирання проектів з використанням POM-файлів (Project Object Model).

Файлова структура: В Maven проекти організовуються в структурі з певними стандартами та конвенціями. Інформація про залежності та конфігурація збирання зберігаються в POM-файлах.

Плагіни: Maven використовує плагіни для виконання різних завдань, таких як збирання, тестування, залежності тощо.

Зручність в управлінні залежностями: Maven має розширену систему керування залежностями, яка дозволяє легко вказувати та вирішувати залежності від сторонніх бібліотек.

Gradle:

Гнучкість: Gradle надає більше гнучкості у налаштуванні проектів. Він використовує Groovy або Kotlin DSL для опису проекту, що дозволяє вам створювати більш динамічні та складні скрипти для збирання.

Паралельність та швидкість: Gradle може ефективно виконувати завдання паралельно, що забезпечує швидше збирання та тестування проектів.

Інкрементальна збірка: Gradle підтримує інкрементальну збірку, що означає, що він збирає лише ті частини проекту, які були змінені.

Залежності: Gradle також має систему керування залежностями та може використовувати репозиторії Maven.

Обидві системи мають свої переваги, і вибір між ними залежить від ваших потреб та вподобань. Maven використовується більш традиційно та має багато стандартів та конвенцій, тоді як Gradle надає більше гнучкості та швидкості в налаштуванні та виконанні завдань збирання.

В якості системи збирання проекту було вибрано Maven, тому що тві має наступні переваги:

Стандарти та конвенції: Maven використовує стандартні конвенції та структуру проекту, що полегшує спільну розробку та роботу з іншими розробниками. Якщо ваш проект відповідає цим конвенціям, Maven буде більш природнім вибором.

Широкий спектр плагінів: Maven має велику кількість плагінів, які охоплюють різні аспекти розробки, тестування та розгортання проекту. Вибір і налаштування плагінів в Maven може бути досить простим завданням.

Широке спільнота та документація: Maven має велику активну спільноту та обширну документацію, що робить вивчення та розвиток проектів більш дослідженими та підтримує вирішення проблем.

Підтримка для внутрішнього репозиторію: Maven дозволяє створювати власний внутрішній репозиторій, що полегшує управління залежностями та внутрішнім розповсюдженням артефактів.

Інтеграція з IDE: Maven добре інтегрується з багатьма популярними інтегрованими розробницькими середовищами (IDE), такими як Eclipse, IntelliJ IDEA і NetBeans

Для написання ефективного додатку було відібрано наступні залежності, які будуть описані нижче. [14]

spring-boot-starter-data-jpa - це одна з стартових залежностей в Spring Boot, яка надає підтримку для роботи з Java Persistence API (JPA) і реляційними базами даних в контексті веб-додатків. Основна мета цієї залежності - спростити налаштування та взаємодію з базами даних в Spring Boot додатках.

Основні функції та призначення spring-boot-starter-data-jpa:

Інтеграція з JPA: Залежність надає зручний спосіб підключити JPA до вашого додатка. JPA - це Java-стандарт для роботи з реляційними базами даних через об'єктно-реляційне відображення (ORM). Використання JPA дозволяє вам працювати з базами даних, використовуючи об'єктно-орієнтований підхід.

Автоматична конфігурація: Spring Boot автоматично налаштовує JPA і джерело даних на основі наявних залежностей та налаштувань. Вам не потрібно вручну налаштовувати XML-файли або Java-класи конфігурації для підключення до бази даних.

Зручна робота з сутностями: Ви можете описувати сутності (таблиці бази даних) у вигляді звичайних Java-класів з анотаціями JPA. Spring Boot автоматично створює таблиці на основі цих сутностей та дозволяє вам взаємодіяти з ними як з об'єктами.

Підтримка транзакцій: Spring Boot додає підтримку транзакцій для операцій з базою даних. Ви можете використовувати анотації Spring, такі як @Transactional, для визначення границь транзакцій та управління ними.

Підтримка різних баз даних: Ви можете використовувати spring-boot-starter-data-jpa з різними реляційними базами даних, такими як PostgreSQL, MySQL, H2, Oracle і багато інших.

Отже, spring-boot-starter-data-jpa допомагає розробникам легко і швидко підключитися до реляційних баз даних та використовувати JPA для роботи з даними. Ця залежність спрощує процес розробки та підтримки веб-додатків, що вимагають взаємодії з базами даних

spring-boot-starter-validation - це стартова залежність в Spring Boot, яка надає підтримку для валідації даних в вашому веб-додатку. Валідація - це процес перевірки та підтвердження коректності введених даних перед їх використанням у додатку. Ця залежність базується на стандарті Java Bean Validation, і вона допомагає забезпечити правильність та цілісність даних, що вводяться користувачем.

Основні функції та призначення spring-boot-starter-validation:

Анотації валідації: Залежність надає набір анотацій, які можна використовувати для позначення правил валідації на полях Java-класів. Наприклад, анотація @NotNull позначає, що поле не повинно бути пустим, а @Min та @Max визначають мінімальні та максимальні значення числових полів.

Підтримка кастомних валідаторів: Ви можете створювати власні валідатори, які реалізують логіку валідації для конкретних вимог вашого додатку.

Автоматична валідація в контролерах: При обробці HTTP-запитів у контролерах Spring Boot автоматично застосовує валідацію до вхідних даних, використовуючи анотації валідації. Це дозволяє перевіряти вхідні дані та генерувати відповіді про помилки, якщо дані не відповідають вимогам.

Підтримка локалізації помилок: Ви можете налаштовувати локалізацію повідомлень про помилки валідації, щоб забезпечити правильне повідомлення користувачам у відповідності до їхньої мови та регіону.

Інтеграція з іншими стартовими залежностями: spring-boot-starter-validation може бути легко поєднана з іншими стартовими залежностями Spring Boot для створення повноцінних веб-додатків.

Отже, spring-boot-starter-validation допомагає забезпечити правильність та цілісність даних у веб-додатку шляхом валідації вхідних даних і забезпечує більшу безпеку та надійність додатка.

spring-boot-starter-security - це стартова залежність в Spring Boot, яка надає підтримку для реалізації безпеки та автентифікації в вашому веб-додатку. Безпека важлива для захисту даних та ресурсів додатку від несанкціонованого доступу. Ця залежність дозволяє легко налаштувати та керувати механізмами безпеки в Spring Boot.

Основні функції та призначення spring-boot-starter-security:

Автентифікація і авторизація: Залежність дозволяє налаштовувати правила для автентифікації користувачів та надавати їм доступ до різних частин додатку на основі їхніх ролей та прав.

Захист ресурсів: Ви можете захистити різні ресурси додатку, такі як URL-шляхи або API-контролери, від несанкціонованого доступу. Залежність надає

можливість налаштовувати доступ до ресурсів за допомогою ролей, анотацій та фільтрів безпеки.

Управління користувачами та ролями: Ви можете зберігати користувачів та їхні ролі в базі даних або інших сховищах. Залежність допомагає управляти користувачами, їхніми паролями та ролями.

Інтеграція з іншими системами автентифікації: Ви можете інтегрувати spring-boot-starter-security з іншими системами автентифікації, такими як LDAP, OAuth, або SAML. [15]

Локалізація та кастомізація повідомлень про помилки: Залежність дозволяє налаштовувати повідомлення про помилки та локалізацію для відображення коректних повідомлень користувачам.

Захист від атак: spring-boot-starter-security надає захист від таких атак, як атаки на зміну паролів, вплив на сесії, атаки перехоплення сесій і багато інших.

Ця залежність дозволяє розробникам забезпечити високий рівень безпеки та контролю над доступом до ресурсів в їхньому веб-додатку. Вона дозволяє вам встановлювати і налаштовувати правила безпеки відповідно до конкретних вимог вашого додатку і забезпечує захист від загроз безпеці та незаконного доступу.

log4j-core - це один з компонентів Apache Log4j 2, який є популярним фреймворком для логування в Java-додатках. Log4j дозволяє розробникам структуровано та ефективно збирати, зберігати і аналізувати логи, щоб виявляти та виправляти проблеми в додатках та відстежувати їх роботу.

Основні функції та призначення log4j-core:

Логування за рівнями: Log4j дозволяє розробникам налаштовувати логування за різними рівнями важливості, такими як DEBUG, INFO, WARN, ERROR, тощо. Ви можете визначати, які повідомлення будуть збережені в логах залежно від рівня.

Апендери (Appenders): Ви можете налаштовувати апендери для визначення, куди і які дані будуть записані. Наприклад, ви можете налаштовувати апендер для запису в консоль, файл, базу даних, або надсилання логів через мережу.

Фільтри (Filters): Log4j дозволяє встановлювати фільтри для визначення, які

логи будуть записані на основі певних умов або критеріїв.

Можливість налаштування через конфігураційні файли: Ви можете налаштовувати Log4j через конфігураційні файли, такі як log4j2.xml або log4j2.properties, що дозволяє легко змінювати параметри логування без перекомпіляції додатка.

Логування в асинхронному режимі: Log4j підтримує асинхронне логування, що зменшує вплив логування на продуктивність додатка.

Підтримка розширень: Log4j має багато розширень і додаткових модулів, які дозволяють розширити його функціональність для конкретних вимог.

log4j-core допомагає розробникам забезпечити ефективне та гнучке логування для відстеження роботи додатків, виявлення помилок і аналізу діяльності системи. Він є одним з популярних інструментів для логування в Java-додатках і використовується в багатьох проектах

log4j-api - це один з компонентів Apache Log4j 2, який є інтерфейсом для взаємодії з фреймворком для логування Apache Log4j 2. Цей компонент надає API та інтерфейси для створення та керування журналами логів у додатках.

Основні функції та призначення log4j-api:

Логування за рівнями: log4j-api дозволяє розробникам логувати повідомлення з різними рівнями важливості, такими як DEBUG, INFO, WARN, ERROR, тощо. Ви можете використовувати це API для запису логів з різних частин вашого додатку.

Апендери (Appenders): API дозволяє налаштовувати апендери для визначення, куди будуть записані логи. Ви можете налаштовувати апендери для виводу логів в консоль, файл, базу даних, або інші призначені місця.

Фільтри (Filters): Ви можете встановлювати фільтри для визначення, які логи будуть записані на основі певних умов або критеріїв. Фільтри дозволяють керувати тим, які логи будуть збережені.

Підтримка розширень: log4j-api підтримує розширення та додаткові модулі, які можна використовувати для розширення функціональності фреймворку.

Підтримка логування в асинхронному режимі: Ви можете використовувати

log4j-api для асинхронного логування, що дозволяє зменшити вплив логування на продуктивність додатка.

Локалізація та кастомізація повідомлень про помилки: log4j-api дозволяє налаштовувати повідомлення про помилки та локалізацію для відображення коректних повідомлень користувачам.

log4j-api дозволяє розробникам ефективно керувати логуванням в їхніх додатках і забезпечити відстеження роботи системи, виявлення помилок і аналіз логів. Він є одним з ключових компонентів Apache Log4j 2 та використовується для створення розширених систем логування в Java-додатках.

spring-boot-starter-web - це стартова залежність в Spring Boot, яка надає базову підтримку для розробки веб-додатків. Вона включає в себе необхідні компоненти та бібліотеки для створення веб-додатків, які можуть обробляти HTTP-запити та відповіді.

Основні функції та призначення spring-boot-starter-web:

Вбудований веб-сервер: Spring Boot додає вбудований веб-сервер (зазвичай Tomcat, Jetty або Undertow), що дозволяє запускати веб-додаток без необхідності налаштування окремого веб-сервера.

Керування URL-шляхами: Ви можете використовувати анотації, такі як @RequestMapping, для визначення URL-шляхів та обробки HTTP-запитів.

Обробка HTTP-запитів та відповідей: spring-boot-starter-web надає можливість створення контролерів, які обробляють HTTP-запити та генерують HTTP-відповіді. Ви можете використовувати анотації, такі як @Controller та @RestController, для створення контролерів.

Шаблони видів (View Templates): Ви можете використовувати шаблонизатори, такі як Thymeleaf, FreeMarker або JSP, для генерації веб-сторінок та відображення даних.

Підтримка JSON та XML: spring-boot-starter-web підтримує серіалізацію та десеріалізацію даних у форматах JSON та XML для обміну даними з клієнтами.

Обробка статичних ресурсів: Ви можете налаштовувати обробку статичних ресурсів, таких як CSS, JavaScript та зображення, щоб вони були доступні з веб-

сторінок.

Підтримка WebSocket: Spring Boot надає підтримку WebSocket для створення веб-додатків з інтерактивною реальним часом.

spring-boot-starter-web є однією з основних стартових залежностей для створення веб-додатків з використанням Spring Boot. Вона спрощує розробку веб-додатків та надає засоби для роботи з HTTP-запитами та відповідями, що робить її ідеальним вибором для розробки веб-додатків на платформі Spring Boot.

spring-session-core - це модуль Spring Framework, який надає підтримку управління сесіями користувачів у веб-додатках. Він дозволяє зберігати та управляти сесіями користувачів в розподіленому середовищі і надає різні реалізації для зберігання сесій, включаючи зберігання в пам'яті, базі даних або зовнішньому сховищі.

Основні функції та призначення spring-session-core:

Управління сесіями користувачів: Цей модуль дозволяє створювати, зберігати та управляти сесіями користувачів в веб-додатках. Сесії дозволяють зберігати стан та інформацію про користувачів між HTTP-запитами.

Розподілене зберігання сесій: spring-session-core надає можливість зберігати сесії в розподіленому середовищі. Це дозволяє балансувати навантаження та забезпечувати доступ до сесій з різних вузлів додатка.

Підтримка різних сховищ для сесій: Модуль підтримує різні сховища для зберігання сесій, включаючи Redis, бази даних, Apache Geode, Apache ZooKeeper, а також підтримує зберігання в пам'яті. Ви можете вибрати сховище відповідно до ваших потреб.

Підтримка HTTP-заголовків для ідентифікації сесій: spring-session-core може використовувати HTTP-заголовки для ідентифікації сесій користувачів. Це дозволяє підтримувати безстанційні (stateless) архітектури додатків.

Підтримка безпеки сесій: Модуль дозволяє налаштовувати заходи безпеки для сесій користувачів, включаючи валідацію та заборону доступу до сесій.

Підтримка розширень: spring-session-core підтримує розширення для інтеграції з іншими бібліотеками та фреймворками, такими як Spring Security.

Цей модуль спрощує управління сесіями користувачів у веб-додатках, забезпечуючи розподілене та надійне зберігання сесій та підтримку безпеки. Він особливо корисний для веб-додатків, які працюють у розподіленому середовищі та потребують зберігання сесій у зовнішньому сховищі.

spring-boot-devtools - це модуль Spring Boot, призначений для полегшення розробки та вдосконалення досвіду розробника під час розробки веб-додатків на платформі Spring Boot. Він надає різноманітні інструменти та можливості для автоматичної перезавантаження додатку, підтримки гарячої перезавантаження (Hot Swapping) коду та багато іншого.

Основні функції та призначення spring-boot-devtools:

Перезавантаження додатку: Модуль spring-boot-devtools дозволяє автоматично перезапускати додаток після внесення змін у джерелі коду. Це полегшує інтерактивну розробку та тестування без необхідності ручного перезапуску додатку.

Гаряче перезавантаження коду (Hot Swapping): Ця функція дозволяє внесені зміни до класів і ресурсів бути завантаженими без перезапуску додатку, що прискорює процес розробки.

Автоматична підтримка інструментів розробки: spring-boot-devtools надає підтримку для різних інструментів розробки, таких як Eclipse, IntelliJ IDEA та інших. Він надає конфігурацію та плагіни для інтеграції цих інструментів.

Автоматична налаштуваність: Модуль дозволяє налаштовувати розробницький режим автоматично, забезпечуючи налаштування, такі як відключення кешування, вимкнення безпеки та інше.

Регулярні збірки (LiveReload): Модуль надає підтримку для збірки ресурсів та веб-сторінок без перезавантаження сторінки браузера. Це полегшує розробку інтерфейсу користувача.

spring-boot-devtools робить процес розробки веб-додатків на платформі Spring Boot більш ефективним і зручним для розробників. Він дозволяє швидко бачити вплив внесених змін, реагувати на них та прискорює процес тестування та вдосконалення додатку

postgresql (або postgresql dependency) - це залежність (dependency) для проектів на платформі Spring Boot, яка додає підтримку PostgreSQL бази даних. PostgreSQL - це потужна та добре відома система керування реляційними базами даних (СКБД), яка використовується для зберігання та управління даними в багатьох веб-додатках і додатках загалом.

Залежність postgresql для Spring Boot включає в себе необхідні бібліотеки та конфігурацію, які дозволяють підключитися до бази даних PostgreSQL та взаємодіяти з нею з використанням Spring Data JPA або інших технологій доступу до даних.

Основні функції та призначення залежності postgresql:

Підключення до PostgreSQL: Залежність надає необхідні засоби для підключення до сервера PostgreSQL з використанням JDBC (Java Database Connectivity). [16]

Підтримка Spring Data JPA: Якщо ви використовуєте Spring Data JPA для доступу до даних, залежність postgresql допомагає конфігурувати джерело даних та сутності для роботи з PostgreSQL.

Підтримка налаштувань для підключення: Ви можете налаштовувати параметри підключення до PostgreSQL, такі як URL бази даних, ім'я користувача, пароль та інші.

Підтримка виконання SQL-запитів: Залежність дозволяє виконувати SQL-запити до бази даних PostgreSQL та взаємодіяти з нею з використанням Spring JDBC Template.

Підтримка транзакцій: Spring Boot разом з залежністю postgresql дозволяє використовувати транзакції для забезпечення цілісності та консистентності даних в базі даних.

Використання залежності postgresql дозволяє легко інтегрувати PostgreSQL в ваш Spring Boot додаток та ефективно працювати з цією реляційною базою даних. PostgreSQL відомий свою надійність та високий рівень підтримки стандартів SQL, що робить його популярним в сферах розробки веб-додатків та інших сферах

Lombok - це бібліотека для мови програмування Java, яка дозволяє зменшити

кількість зайвого та багато разів пишучого коду, не втрачаючи при цьому читабельність і підтримку інструментів розробки. Вона пропонує аннотації для генерації стандартного бойлерплейту (boilerplate code) в коді, такого як геттери, сеттери, методи equals та hashCode, конструктори та інші.

Основні функції та призначення Lombok:

Спрощення коду: Lombok дозволяє додавати аннотації до полів класів, і він автоматично генерує необхідні методи та код. Наприклад, аннотація @Getter автоматично генерує геттер для поля, а @Setter генерує сеттер.

Мінімізація бойлерплейту: Бойлерплейт (boilerplate code) - це стандартний, повторюваний та нудний код, який не несе значущої функціональності. Lombok допомагає уникнути написання зайвого коду та зосередити увагу на бізнес-логіці додатка.

Підвищення продуктивності розробки: Завдяки Lombok розробники можуть швидше створювати моделі даних та об'єкти, не витрачаючи час на написання стандартного коду.

Покращення читабельності: Використання Lombok дозволяє зберегти код більш читабельним і менш об'ємним, оскільки він приховує деталі ініціалізації та доступу до полів.

Інтеграція з іншими інструментами: Lombok підтримується популярними інструментами розробки, такими як IntelliJ IDEA та Eclipse.

Залежно від ваших потреб, Lombok може значно спростити розробку Java-програм, зменшити кількість написаного коду та полегшити підтримку та модифікацію програм

spring-boot-starter-test - це залежність для проектів на платформі Spring Boot, яка надає інфраструктуру для написання і виконання тестів в Spring Boot додатках. Ця залежність містить різні інструменти та бібліотеки для тестування, які допомагають розробникам створювати надійні та коректні додатки.

Основні функції та призначення spring-boot-starter-test:

Підтримка тестування Spring Boot додатків: Залежність дозволяє створювати тести для Spring Boot додатків та виконувати їх в контексті Spring Boot.

Інтеграція з JUnit та іншими фреймворками тестування: spring-boot-starter-test інтегрується з популярними фреймворками тестування, такими як JUnit та TestNG, що дозволяє писати різновиди тестів, включаючи модульні, інтеграційні та функціональні.

Підтримка автоматичного конфігурування: Залежність автоматично налаштовує тестовий контекст Spring, включаючи автопідключення залежностей, налаштування джерела даних, обробників помилок та інше.

Підтримка тестування контролерів та сервісів: Ви можете створювати тести для контролерів та сервісів вашого додатка та перевіряти їх функціональність.

Підтримка тестування бази даних: Залежність дозволяє легко налаштовувати бази даних для тестів та виконувати їх без змін в реальному додатку.

Підтримка тестування з веб-запитами: Ви можете створювати тести для вашого RESTful API, включаючи тести з використанням HTTP-запитів і перевіряти їх відповіді.

spring-boot-starter-test робить тестування Spring Boot додатків більш зручним та ефективним, допомагаючи розробникам переконатися в правильності та надійності їх коду. Вона включає в себе багато інструментів для тестування, що спрощують процес тестування та дозволяють легко ідентифікувати помилки та проблеми.

spring-session-jdbc - це модуль Spring Framework, який дозволяє зберігати і керувати сесіями користувачів в базі даних за допомогою JDBC (Java Database Connectivity). Цей модуль особливо корисний при розробці веб-додатків, де потрібно зберігати та управляти сесіями користувачів.

Основні функції та призначення spring-session-jdbc:

Зберігання сесій в базі даних: Модуль дозволяє зберігати дані сесій користувачів в таблицях бази даних. Це дозволяє зберігати стан сесій навіть після перезапуску додатку та робить їх доступними для різних інстанцій додатку.

Підтримка розподіленої архітектури: Збереження сесій в базі даних дозволяє використовувати розподілену архітектуру, де додаток може мати кілька серверів або нодів, і користувачі можуть переходити між ними без втрати сесій.

Підтримка багатокористувацькості: Модуль дозволяє обслуговувати багато користувачів одночасно, зберігаючи їх сесії та дозволяючи реалізувати аутентифікацію та авторизацію.

Підтримка безпеки: Модуль дозволяє зберігати сесії в безпечному місці, де вони недоступні для несанкціонованого доступу.

Підтримка резервного копіювання та відновлення: Сесії можуть бути резервовані та відновлені в разі аварійного завершення роботи додатку.

Підтримка залежностей: Модуль інтегрується з іншими модулями Spring, такими як Spring Security, для забезпечення безпеки та авторизації.

spring-session-jdbc робить управління сесіями користувачів більш ефективним та надійним, дозволяючи зберігати та управляти сесіями в базі даних. Він особливо корисний для розробників веб-додатків, які потребують зберігати стан сесій і забезпечувати роботу з багатьма користувачами одночасно

2.3 Аналіз структури проекту, дослідження принципів ООП та SOLID під час побудови додатку. Валідація

Принципи об'єктно-орієнтованого програмування (ООП) - це основні концепції, на яких ґрунтується цей підхід до програмування. ООП спрямований на створення програм, організованих навколо об'єктів, які представляють реальні або абстрактні сутності.

Інкапсуляція (Encapsulation): Інкапсуляція вказує на об'єднання даних (змінних) і методів (функцій), що працюють з цими даними в одному класі. Це призводить до обмеження доступу до даних та методів, що дозволяє контролювати їхню цілісність і захищати дані від несанкціонованого доступу.

Спадкування (Inheritance): Спадкування дозволяє створювати нові класи на основі існуючих класів, успадковуючи їхні властивості та методи. Це сприяє відновленню та розширенню функціональності існуючих класів, зменшуючи дублювання коду.

Поліморфізм (Polymorphism): Поліморфізм означає можливість одного і того ж іменованого методу виконувати різні дії в залежності від контексту використання. Це може бути досягнуто за допомогою віртуальних методів, інтерфейсів, абстрактних класів та перевантаження методів.

Абстракція (Abstraction): Абстракція полягає в тому, що класи та об'єкти моделюють реальні або абстрактні об'єкти та процеси в програмі, і вони приховують деталі реалізації. Це дозволяє програмістам працювати з об'єктами на більш вищому рівні абстракції, що спрощує розробку та розуміння програмного коду.

Принципи SOLID - це п'ять основних принципів об'єктно-орієнтованого програмування та проектування, які допомагають створювати гнучкі, підтримувані та розширювані програми. Ці принципи були сформульовані Робертом С. Мартіном та іншими програмістами і включають наступні скорочення, які утворюють анаграму "SOLID":

Принцип одного відповідального об'єкта (Single Responsibility Principle, SRP): Кожен клас повинен мати лише одну причину зміни. Це означає, що клас повинен виконувати лише одну конкретну функцію або завдання. Якщо клас має більше однієї причини для зміни, його слід розділити на кілька менших класів.

Принцип відкритості-закритості (Open-Closed Principle, OCP): Програмні сутності (класи, модулі, функції тощо) повинні бути відкритими для розширення, але закритими для модифікації. Це означає, що ви можете додавати новий функціонал до існуючих сутностей, не змінюючи їхнього внутрішнього коду.

Принцип підстановки Барбари Лісков (Liskov Substitution Principle, LSP): Об'єкти підкласу повинні бути замінюваними за об'єктами базового класу без втрати правильності програми. Це означає, що підкласи повинні зберігати інтерфейс базового класу та виконувати всі його умови. [17]

Принцип розділення інтерфейсу (Interface Segregation Principle, ISP): Клієнти не повинні залежати від інтерфейсів, які вони не використовують. Замість того, щоб створювати великі загальні інтерфейси, слід розділяти їх на більш малі та спеціалізовані інтерфейси, щоб клієнти мали доступ лише до того функціоналу,

який їм потрібен.

Принцип інверсії залежностей (Dependency Inversion Principle, DIP): Високорівневі модулі не повинні залежати від низькорівневих модулів, а обидва типи модулів повинні залежати від абстракцій. Цей принцип сприяє створенню слабкого зв'язку між різними частинами програми і підвищує її гнучкість та розширюваність.

Використання цих принципів SOLID допомагає покращити якість програмного коду, спрощує його розробку та підтримку, а також робить програми більш гнучкими та готовими до змін.

Для досягнення цілей відповідності основним принципам які були розписані вище, була вибрана наступна структура проекту, яка показана нижче:

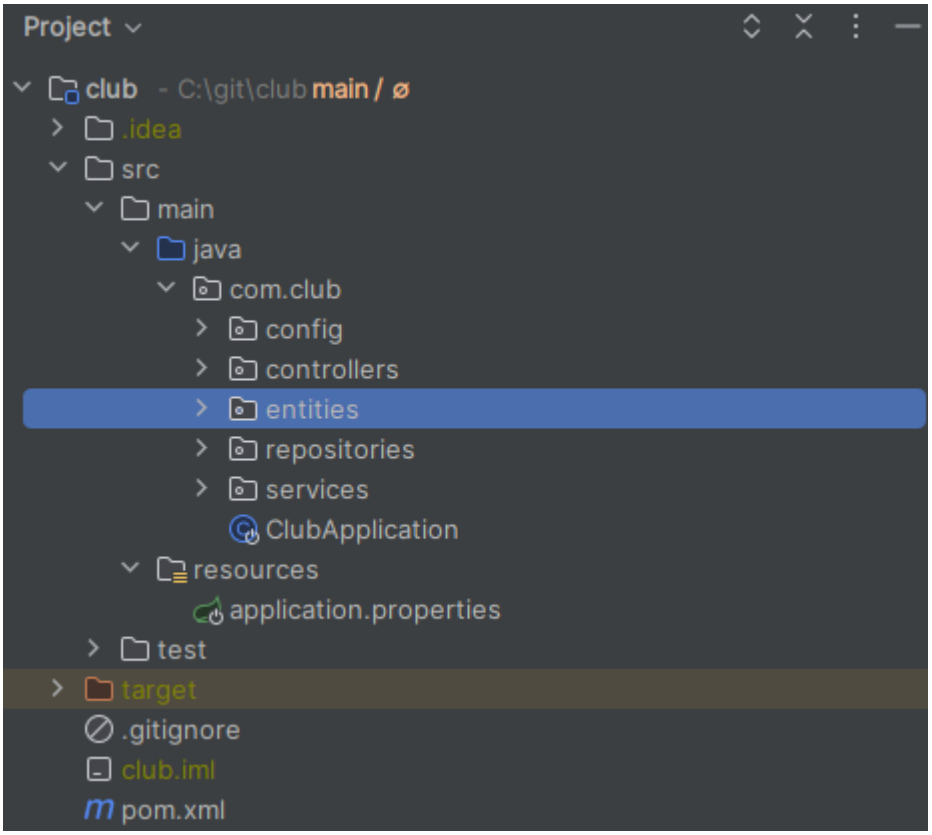


Рисунок 2.3 - Структура додатку

Кожний package відповідає за свої дані. В package «config» знаходяться основні налаштування проекту, «controllers» відповідають за оброблення запитів, «repositories» відповідають за звернення до бази даних, «services» відповідають за оброблення бізнес-логіки, а «entities» за набір сутностей, які зберігаються в базі даних.

Нижче наведено типовий приклад entity:

```java
@Entity
@Getter
@Setter
@ToString
public class CompanyProfile implements Serializable {

    @Id
    @GeneratedValue(generator = "uuid2")
    @GenericGenerator(name = "uuid2", strategy = "org.hibernate.id.UUIDGenerator")
    private String id;

    @NotBlank
    private String companyName;

    private String description;

    private Long employees;
    asternus, 05.09.2023 19:41 • company profile
    @NotNull
    private BigDecimal companyBudget;

    @OneToMany(mappedBy = "companyProfile", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JsonIgnore
    private List<WorkProfile> workProfiles;

    asternus
    public CompanyProfile() {
    }
}
```

Рисунок 2.4 - Entity

Як видно з рисунку вищу, він повністю відповідаю принципам проектування описаним вище, та використовує guid як унікальне поле. В чому переваги даного типу унікального ключа, та зв’язки між сутностями ми розглянемо в розділі 4 «Бази даних». Для генерування геттерів та сеттерів було застосовано Lombok, який було описано в розділі 2.2.

Для деяких полів була застосована валідація.

Валідація - це процес перевірки введених даних, щоб переконатися, що вони відповідають певним критеріям чи обмеженням перед їхнім подальшим використанням. Цей процес має на меті гарантувати правильність, цілісність і безпеку даних у програмах та системах. Валідація є важливою частиною розробки програмного забезпечення і використовується з численними цілями:

Запобігання помилкам та даним низької якості: Валідація допомагає запобігти введенню некоректних або некорисних даних в систему. Вона перевіряє,

чи введені дані відповідають очікуваному формату та обмеженням.

Забезпечення безпеки даних: Валідація даних допомагає уникнути атак, таких як SQL-ін'єкція або внесення шкідливого коду через введення користувача. Вона допомагає відфільтрувати або замінити потенційно небезпечний ввід.

Забезпечення коректності функціонування програми: Валідація гарантує, що програма отримує правильні дані для обробки, що дозволяє уникнути некоректної поведінки програми чи відображення некоректних результатів.

Покращення зручності для користувача: Валідація даних може покращити досвід користувача, допомагаючи надавати користувачеві зрозумілі повідомлення про помилки та вказівки щодо правильного формату даних.

Забезпечення консистентності бази даних: Валідація може бути важливою для забезпечення консистентності даних у базі даних. Вона допомагає переконатися, що дані, які вносяться в базу, відповідають вимогам схеми даних.

Запобігання перевантаженню системи: Валідація може допомогти виявити та відхилити невірні або спамові дані, що можуть завантажити систему та призвести до некоректної роботи.

Загалом, валідація є важливою складовою якісного програмного забезпечення, оскільки вона допомагає гарантувати надійність та безпеку даних, забезпечує коректне функціонування програм та покращує взаємодію з користувачами

```
package com.club.repositories;

import com.club.entities.WorkProfile;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

import java.util.List;

3 usages  asternus
@Repository
public interface WorkProfileRepository extends JpaRepository<WorkProfile, String> {

    1 usage  asternus
    @Query("from WorkProfile w where w.user is null")
    List<WorkProfile> getAllFreeWorkProfile();

}
```

Рисунок 2.5 - Репозіторій

Вище наведено типовий клас репозіторію. Він використовує технологію spring-boot-starter-data-jpa, описану в розділі 2.2. та використовує мову HQL для написання запитів до бази даних.

Statement та PreparedStatement - це два способи взаємодії з базою даних в мові програмування Java. Різниця між ними полягає в підходах до створення та виконання SQL-запитів, а також у їхній безпеці. [18]

Statement:

Використовується для виконання статичних SQL-запитів.

SQL-запити вставляються безпосередньо в код програми, що може призвести до SQL-ін'єкції, якщо дані користувача не очищені від потенційно шкідливих символів.

Не підтримує попередньо підготовлених запитів, тому кожен SQL-запит повинен бути повністю сформований при кожному виклику.

PreparedStatement:

Використовується для виконання підготовлених SQL-запитів.

SQL-запити попередньо компілюються на стороні бази даних та використовуються з параметрами, які вставляються безпосередньо в запит під час виконання.

Запити підготовлюються лише один раз і можуть бути виконані з різними значеннями параметрів, що покращує продуктивність та зменшує ризик SQL-ін'єкції, оскільки база даних відділяє код SQL від користувацьких даних.

Проблема SQL ін'єкції:

SQL ін'єкція - це вид атаки на безпеку даних, коли зловмисник вставляє шкідливий SQL-код у введені дані, які потім виконуються базою даних. Це може призвести до руйнування чи зламу системи, витоку конфіденційних даних, а також до некоректного виконання SQL-запитів.

Проблему SQL ін'єкції можна уникнути за допомогою наступних практик:

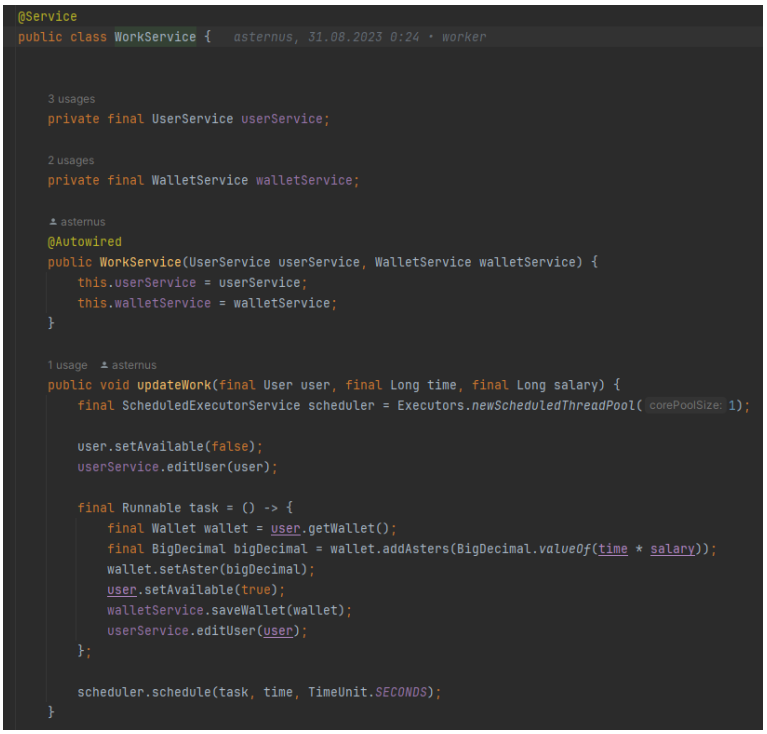
Використання PreparedStatement: Використання підготовлених запитів дозволяє параметризувати SQL-запити та вставляти дані безпосередньо в запит, замість конкатенації даних в рядку запиту.

Валідація введених даних: Перевірка та фільтрація даних, які вводить користувач, перед використанням у SQL-запитах. Наприклад, можна використовувати обмеження допустимих символів, або перевіряти на відповідність очікуваному формату.

Використання обмежень доступу: Налаштування бази даних та ролей користувачів, щоб забезпечити обмеження доступу до функціоналу бази даних і забезпечити, що користувачі не мають повного доступу до всіх таблиць чи процедур.

Використання бібліотек для роботи з базами даних: Використовувати сторонні бібліотеки та фреймворки, які мають вбудовані заходи безпеки для запитів до бази даних

В даній магістерській роботі використовуються PreparedStatement запити у всіх репозиторіях.



```
@Service
public class WorkService {   asternus, 31.08.2023 0:24 • worker


    3 usages
    private final UserService userService;

    2 usages
    private final WalletService walletService;

    asternus
    @Autowired
    public WorkService(UserService userService, WalletService walletService) {
        this.userService = userService;
        this.walletService = walletService;
    }

    1 usage   asternus
    public void updateWork(final User user, final Long time, final Long salary) {
        final ScheduledExecutorService scheduler = Executors.newScheduledThreadPool( corePoolSize: 1);

        user.setAvailable(false);
        userService.editUser(user);

        final Runnable task = () -> {
            final Wallet wallet = user.getWallet();
            final BigDecimal bigDecimal = wallet.addAsters(BigDecimal.valueOf(time * salary));
            wallet.setAster(bigDecimal);
            user.setAvailable(true);
            walletService.saveWallet(wallet);
            userService.editUser(user);
        };

        scheduler.schedule(task, time, TimeUnit.SECONDS);
    }
```

Рисунок 2.6 - Сервісний клас

Вище наведено приклад сервісного класу, який використовую Spring Boot для Dependency injection.

Dependency Injection (DI) - це паттерн проектування та метод інверсії контролю, який використовується у програмуванні для зменшення залежностей

між компонентами програми. Головною ідеєю DI є введення залежностей (dependencies) в об'єкти або компоненти програми не за допомогою самостійного створення цих залежностей всередині об'єкта, а за допомогою передачі їх ззовні.

В основі DI лежить ідея розділення створення об'єктів (ініціалізації залежностей) від їх використання. Це сприяє покращенню модульності та розширюваності програми, а також полегшує тестування коду. Залежності, такі як інші класи, сервіси чи об'єкти, ін'єктуються в об'єкт (клас) як параметри конструктору або методів (частіше конструктору), або через властивості об'єкта.

Основні переваги DI включають:

Зменшення залежностей: DI дозволяє визначити залежності один раз і використовувати їх багаторазово в різних частинах програми, що робить код менш залежним від конкретних реалізацій.

Покращення тестованості: DI спрощує тестування, оскільки залежності можна легко замінювати на штучні об'єкти (зазвичай тестові двійники), що дозволяє виконувати модульне тестування без реального взаємодії з внешніми ресурсами чи сервісами.

Покращення розширюваності: Завдяки DI, змінюючи або додаваючи нові залежності, можна легко розширювати функціональність програми, не змінюючи існуючий код.

Покращення читабельності коду: DI робить код більш зрозумілим і призначеним для людського сприйняття, оскільки він дозволяє виокремлювати та визначати залежності явно.

Покращення безпеки: DI може зменшити ризик помилок, пов'язаних із введенням даних, оскільки залежності можуть бути перевірені та валідовані перед використанням.

Загалом, DI є потужним інструментом для розробки програмного коду, який полегшує розробку, тестування та підтримку програм, а також допомагає зменшити залежності між компонентами програми

Крім того, в донному класі було застосовано технологію виклику коду згідно розкладу, що зволяє знизити кількість коду, та інтервали з заданою періодичністю.

2.4 Впровадження Spring Security

Spring Security - це фреймворк для забезпечення безпеки додатків, який розроблений для спрощення впровадження різноманітних механізмів автентифікації, авторизації та управління безпекою в додатках на платформі Spring.

В даному проекті були налаштовані наступні технології.

SecurityFilterChain - це об'єкт, який представляє собою цепочку фільтрів для обробки запитів, пов'язаних з безпекою в Spring Security. Введений у Spring Security 5.0 як частина додаткових можливостей для налаштування безпеки, SecurityFilterChain дозволяє розробникам змінювати та налаштовувати цепочку фільтрів для обробки запитів в залежності від конкретних потреб застосунку.

Основні риси SecurityFilterChain включають:

Конфігурація безпеки на рівні фільтрів: За допомогою SecurityFilterChain, ви можете налаштовувати фільтри для обробки запитів в залежності від шляху запиту, ролей користувачів, або інших критеріїв. Це дозволяє вам точно контролювати обробку запитів на рівні фільтрів.

Зручний спосіб налаштування безпеки: SecurityFilterChain дозволяє розробникам налаштовувати цепочки фільтрів за допомогою зрозумлих інтерфейсів та анотацій, що робить процес налаштування безпеки більш доступним та зрозумілим.

Підтримка багаторазових SecurityFilterChain: Ви можете визначити декілька SecurityFilterChain для обробки різних типів запитів чи ролей користувачів. Це дозволяє вам розділити логіку безпеки для різних частин додатку.

Підтримка динамічної налаштування безпеки: Ви можете динамічно змінювати конфігурацію безпеки під час роботи додатку, що дозволяє адаптувати безпеку до змінних вимог або умов.

Загалом, SecurityFilterChain дозволяє розробникам докладно налаштовувати процес обробки запитів у Spring Security, надаючи більше гнучкості та контролю над безпекою у додатку

CookieSerializer - це інтерфейс в Spring Security, який використовується для

налаштування та керування параметрами та властивостями cookie-файлів, які використовуються для автентифікації та сесійного керування в додатках, побудованих на основі Spring Security.

CookieSerializer має наступні методи:

void writeCookieValue(CookieValue cookieValue): Цей метод призначений для запису значення cookie-файла. Він приймає об'єкт CookieValue, який містить всі необхідні дані про cookie, такі як ім'я, значення, час життя, шлях і т. д.

CookieValue readCookieValue(RequestContext context): Цей метод використовується для читання значення cookie-файла з HTTP-запиту. Він приймає об'єкт RequestContext, який представляє собою контекст поточного HTTP-запиту, і повертає об'єкт CookieValue, який містить дані cookie. [19]

void setUseSecureCookie(boolean useSecureCookie): Цей метод встановлює флаг, що вказує, чи cookie-файли повинні бути відправлені через безпечне з'єднання (SSL). Якщо useSecureCookie встановлено в true, то cookie-файли будуть позначені як безпечні і відправлятимуться лише через захищені з'єднання.

boolean isSecure(RequestContext context): Цей метод визначає, чи поточний запит здійснюється через безпечне з'єднання (SSL). Він приймає об'єкт RequestContext і повертає true, якщо запит є безпечним, і false в іншому випадку.

CookieSerializer дозволяє налаштовувати параметри cookie-файлів, такі як ім'я, час життя, шлях, домен, безпечність і багато інших, щоб відповідати потребам вашого додатку та забезпечувати безпеку та правильну роботу автентифікації та сесійного керування.

Також було налаштовано Cross-Origin Resource Sharing механізм.

CORS (Cross-Origin Resource Sharing) - це механізм, який використовується в веб-розробці для контролю доступу до ресурсів на інших доменах або піддоменах веб-сторінок. Він дозволяє серверу визначити, які джерела (домени) можуть звертатися до ресурсів на сервері через JavaScript у браузері.

CORS необхідний з таких причин:

Безпека: Він допомагає запобігти атакам, які можуть виникнути внаслідок звернення до ресурсів з інших доменів, таких як Cross-Site Request Forgery (CSRF)

та Cross-Site Scripting (XSS).

Розділення обов'язків: CORS дозволяє виділити відповідальність за безпеку між сервером та браузером. Сервер встановлює політику доступу, і браузер виконує її.

Доступ до віддалених ресурсів: CORS дозволяє стороннім сайтам звертатися до віддалених API та інших ресурсів, що розміщені на інших серверах.

В Spring Framework і Spring Security, ви можете налаштувати CORS за допомогою CorsConfiguration і CorsFilter. CorsConfiguration дозволяє визначити політику CORS, включаючи список доменів, які мають доступ, дозволені HTTP-методи, заголовки і т. д. CorsFilter використовується для застосування цієї конфігурації на рівні фільтра для обробки запитів на сервері.

Щоб використовувати CORS в Spring Security, в проекті налаштовано CorsConfigurer, який додає конфігурацію CORS до вашої системи безпеки. Це допоможе забезпечити безпеку і контроль доступу до ресурсів на вашому сервері через HTTP-запити, які надходять з інших доменів.

Також в проекті було використано PasswordEncoder в Spring Security - це інтерфейс, який використовується для шифрування та перевірки паролів користувачів у системах безпеки. Його головна роль - забезпечити безпеку паролів, зберігаючи їх у захищеному вигляді.

Spring Security надає декілька реалізацій PasswordEncoder, але дві з них особливо важливі:

BCryptPasswordEncoder: Цей кодер використовує алгоритм bcrypt для збереження та перевірки паролів. Bcrypt є сучасним і дуже безпечним алгоритмом хешування паролів. Він додатково включає сіль (salt) і ітерації для підвищення безпеки. Основною перевагою BCryptPasswordEncoder є те, що він автоматично генерує та перевіряє сіль, що ускладнює атаки на паролі.

PasswordEncoderFactories.createDelegatingPasswordEncoder: Цей метод створює "делегуючий" парольний кодер, який може обробляти різні алгоритми хешування паролів в залежності від префікса, який додається до закодованого паролю. Він дозволяє використовувати різні алгоритми в залежності від потреб

вашого додатку.

Наприклад, DelegatingPasswordEncoder може розпізнавати такі префікси для паролів:

{bcrypt} для паролів, закодованих алгоритмом bcrypt.

{noop} для незакодованих паролів.

{pbkdf2} для паролів, закодованих алгоритмом PBKDF2.

2.5 CI/CD та модульне тестування додатку

CI/CD стоїть за скороченнями Continuous Integration (постійна інтеграція) та Continuous Deployment (постійне розгортання), і це підходи до розробки програмного забезпечення, які сприяють автоматизації процесів розробки, тестування та розгортання додатків.

Continuous Integration (CI):

Це практика, в якій розробники регулярно об'єднують свій код у спільний репозиторій (наприклад, в системі контролю версій, такій як Git) і виконують автоматизовану процедуру збирання та тестування коду.

Основна ідея - уникнути конфліктів між кодом різних розробників, шукати та виправляти помилки відразу, а не під час злиття коду, і забезпечити готовність коду до розгортання.

Це вимагає використання інструментів для автоматизації, таких як системи збирання (наприклад, Jenkins, Travis CI), тести (наприклад, JUnit, Selenium), та інші.

Continuous Deployment (CD):

Це практика, в якій зміни в коді, які успішно пройшли процес CI, автоматично розгортані на виробничному сервері без додаткового втручання розробників.

Ідея полягає в тому, щоб максимально скоротити час і ризик між написанням коду та його впровадженням виробничому середовищу.

CD включає в себе автоматичне тестування, налаштування серверів, підготовку середовища, та інші операції, необхідні для розгортання нової версії

додатку.

CI/CD сприяють зменшенню ризику помилок, поліпшують якість програмного забезпечення та прискорюють розробку. Вони допомагають автоматизувати процеси, забезпечити більшу стабільність та надійність додатків і забезпечити швидкий цикл розробки.

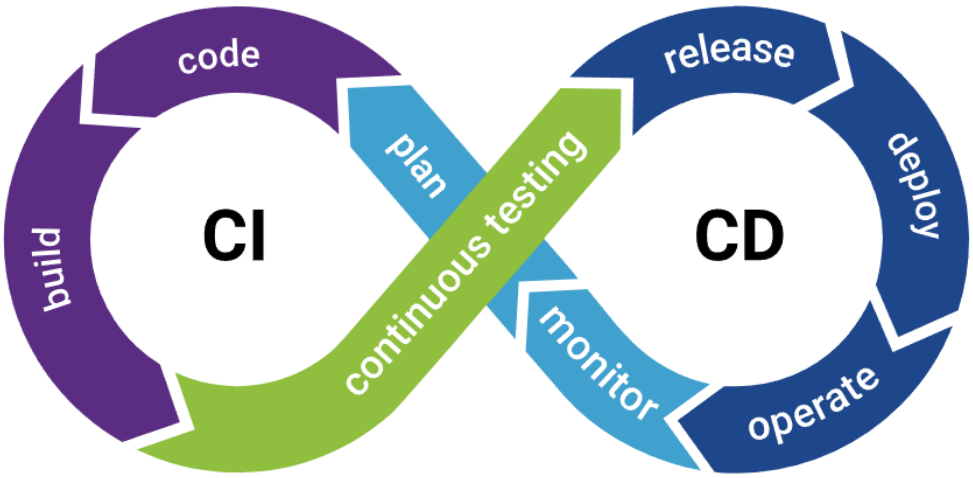


Рисунок 2.7 - Графічне представлення CI/CD. [20]

Ці підходи дозволяють розробникам розгортати новий код на виробничому сервері без затримок та ручних втручань, зменшуючи таким чином час досягнення бізнес-цілей і відгуку на зміни в продукті.

TDD (Test-Driven Development) - це методологія розробки програмного забезпечення, яка передбачає написання тестів перед написанням коду програми. TDD розглядається як ефективний спосіб підвищити якість програмного забезпечення та полегшити процес розробки. Два основні підходи до TDD включають в себе:

Outside-In TDD (ззовні-всередину TDD):

У цьому підході тести розглядаються як засіб специфікації поведінки програми ззовні. Процес Outside-In TDD розпочинається з написання тестів на високорівневий функціонал, який вимагає більше зовнішніх залежностей (наприклад, взаємодії з іншими системами або інтеграції із зовнішніми інтерфейсами).

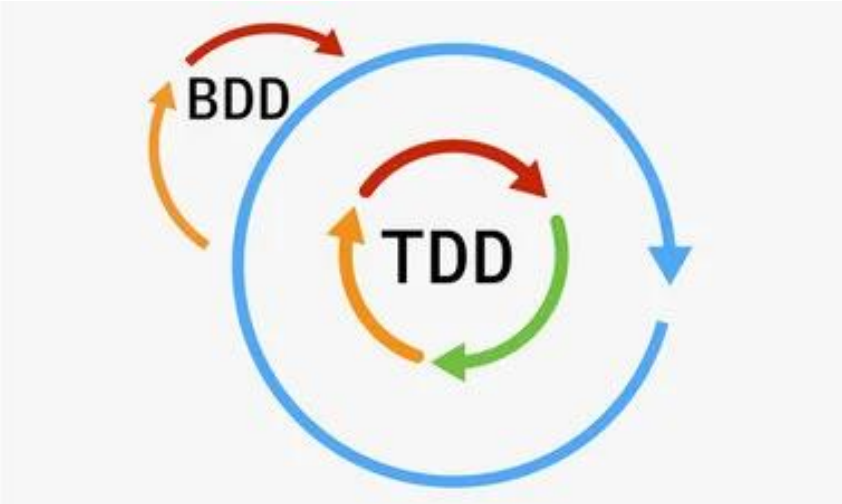


Рисунок 2.8 - Графічне представлення TDD. [21]

Кроки Outside-In TDD:

Спочатку написати тест на високорівневу функціональність або інтеграцію з іншими системами.

Запустити тест, який буде провалився, оскільки код для цієї функціональності ще не написаний.

Написати мінімальний код, необхідний для проходження тесту.

Знову запустити тест та переконатися, що він пройшов успішно.

Поступово розширювати функціональність та додавати нові тести, які вимагають нового коду.

Повторювати процес для кожної нової функціональності.

Inside-Out TDD (зсередини-назовні TDD):

У цьому підході тести розглядаються як засіб перевірки коректності індивідуальних компонентів програми (наприклад, окремих методів чи класів). Процес Inside-Out TDD розпочинається з написання тестів для низькорівневого коду, і ці тести зазвичай тестують окремі методи чи функції.

Кроки Inside-Out TDD:

Спочатку написати тест для окремого методу чи функції.

Запустити тест, який буде провалився, оскільки код для цього методу чи функції ще не написаний.

Написати код для цього методу чи функції, так, щоб тест пройшов успішно.

Поступово розширювати код і тести, додаючи нові методи та функції.

Поступово комбінувати окремі методи і функції, щоб створити більш високорівневу функціональність, і написати для неї відповідні тести.

Обидва підходи до TDD допомагають розробникам створювати код, який легко тестувати, та забезпечувати високу якість програмного забезпечення. Вибір між ними може залежати від конкретних потреб проекту та структури розробки

В даній магістрській роботі використовувася Inside-Out TDD підхід.

Таким чином, Використання Spring Boot для побудови бекенд частини веб-сайту має численні переваги, які роблять цей фреймворк відмінним вибором для розробки серверної частини додатку:

Простота та продуктивність: Spring Boot розроблений з метою спростити розробку серверної частини. Він надає конфігурацію за замовчуванням, що допомагає швидко створити базовий проект та зосередити увагу на реалізації функціональності.

Широкий спектр функціональності: Spring Boot надає набір готових модулів та бібліотек для різних завдань, таких як обробка HTTP-запитів, доступ до бази даних, робота з безпекою, обробка розсилки електронних повідомлень і багато іншого. Це дозволяє розробникам швидко реалізовувати різноманітну функціональність.

Велика спільнота і підтримка: Spring Boot має велику та активну спільноту розробників, яка регулярно вносить внески до розвитку фреймворку. Це гарантує наявність багато різних джерел інформації, документації та плагінів, які допомагають вирішити різні завдання.

Безпека: Spring Boot має вбудовану підтримку для різних аспектів безпеки, таких як автентифікація, авторизація, захист від атак і шифрування даних. Ви можете легко налаштувати механізми безпеки відповідно до ваших потреб.

Масштабованість: Spring Boot добре підходить для розробки масштабованих додатків. Ви можете легко розгорнути додаток на різних серверах та використовувати інші технології Spring, такі як Spring Cloud, для розробки мікросервісної архітектури.

Підтримка різних баз даних: Spring Boot підтримує різні системи управління базами даних, включаючи PostgreSQL, MySQL, Oracle, MongoDB та інші. Це дозволяє вибрати та налаштувати базу даних відповідно до ваших потреб.

Розширюваність та інтеграція: Spring Boot легко інтегрується з іншими технологіями і фреймворками, що дозволяє розробникам вибирати найкращі інструменти для своїх завдань і розширювати функціональність додатку

3 ПРОЕКТУВАННЯ ТА СТВОРЕННЯ ФРОНЕНД ЧАСТИНИ

3.1 Вибір стеку технологій для рощробки фронтенд додатку

Фронтенд - це частина програмного забезпечення, яка взаємодіє з користувачем і відповідає за відображення інформації на екрані та взаємодію з користувачем через інтерфейс користувача. Фронтенд зазвичай представляє собою веб-сторінку, десктопний додаток, мобільний додаток або інший інтерфейс, який користувач використовує для взаємодії з програмою або сервісом.

У контексті REST API (Representational State Transfer Application Programming Interface) фронтенд використовується для взаємодії з сервером, який надає RESTful API. Основні завдання фронтенда в цьому контексті включають:

Відображення даних: Фронтенд відповідає за отримання даних з REST API та їх відображення на веб-сторінці або іншому інтерфейсі користувача. Фронтенд може виконувати запити до API для отримання інформації, такої як тексти, зображення, дані про користувача тощо, і відображати цю інформацію для користувача.

Взаємодія з користувачем: Фронтенд надає користувачам можливість взаємодіяти з додатком або сервісом, включаючи введення даних, виконання дій та навігацію. Фронтенд може збирати введені користувачем дані і надсилати їх на сервер через REST API, а також обробляти відповіді від сервера.

Аутентифікація та авторизація: Фронтенд може забезпечувати можливість користувачам увійти в систему, автентифікуватися та отримувати доступ до обмежених ресурсів на основі авторизації. Для цього він може взаємодіяти з REST API для обробки запитів на аутентифікацію та авторизацію.

Маршрутизація: Фронтенд визначає, які сторінки або розділи програми відображати, як користувач переходить між ними та які дані потрібно завантажити для кожної сторінки. Він також може використовувати REST API для отримання необхідних даних.

Фронтенд та REST API співпрацюють разом, щоб забезпечити

функціональність та взаємодію з додатком або сервісом, надаючи користувачам можливість працювати з даними та виконувати дії через інтерфейс користувача

Як було зазначено у 2 розділі, даної магістрської роботи, було принято використовувати Vue.js. Нижче буде наведено які залежності були використані для побудови фроненд частини проекта.

@popperjs/core - це бібліотека для розміщення попапів (виринаючих віконць, спливаючих підказок, випливаючих меню тощо) у веб-додатках. Вона включає в себе алгоритми для позиціонування цих попапів відносно цільового елементу або області, де вони повинні з'являтися.

Основні особливості @popperjs/core включають:

Позиціонування попапів: Бібліотека визначає оптимальне розташування попапів відносно цільового елементу або іншої області, забезпечуючи, що вони залишаються видимими та відповідають розмірам контейнера.

Автоматичне адаптування розміру: @popperjs/core враховує розміри та розташування попапу, щоб він може адаптуватися до змін вмісту або розмірів вікна браузера. [22]

Конфігурування та налаштування: Бібліотека дозволяє налаштовувати різні параметри попапів, такі як розмір, зсуви, відступи тощо, щоб вони відповідали конкретним потребам додатка.

Різні стратегії відображення: @popperjs/core підтримує різні стратегії відображення попапів, такі як відображення попапу зверху, знизу, ліворуч, праворуч, а також варіанти центрування та інші.

Ця бібліотека стала дуже популярною у веб-розробці та використовується для створення інтерактивних інтерфейсів та для вирізування різних компонентів, таких як випливаючі меню, виринаючі віконця тощо. Вона є частиною екосистеми розробки фронтенду та може використовуватися в поєднанні з іншими бібліотеками та фреймворками, такими як React, Vue.js, або просто як самостійний інструмент.

axios - це бібліотека для здійснення HTTP-запитів з використанням JavaScript. Вона надає інтерфейс для взаємодії з веб-серверами та взаємодії з API через HTTP-

протокол. axios є однією з найпопулярніших бібліотек для виконання HTTP-запитів у веб-розробці, і вона може бути використана як на стороні клієнта (в браузері), так і на стороні сервера (в середовищі Node.js).

Основні можливості axios включають:

Виконання HTTP-запитів: axios підтримує виконання HTTP-запитів, таких як GET, POST, PUT, DELETE і інших, що дозволяє отримувати, надсилати та оновлювати дані на сервері.

Підтримка обіцянок (Promises): Використання axios зазвичай включає в себе використання обіцянок для роботи з асинхронними запитами, що робить код більш читабельним та простим у використанні.

Передача заголовків та параметрів: Ви можете встановлювати HTTP-заголовки та передавати параметри до запитів для налаштування запитів та передачі даних на сервер.

Обробка відповідей: axios дозволяє обробляти відповіді від сервера та отримувати дані у різних форматах, таких як JSON.

Обробка помилок: Бібліотека дозволяє ефективно обробляти помилки, які можуть виникати під час виконання запитів.

Скачування та відправлення файлів: Ви можете використовувати axios для завантаження файлів з сервера або для відправки файлів на сервер.

axios зручно використовувати в сполученні з іншими бібліотеками та фреймворками для створення клієнтської та серверної частин веб-додатків. Вона допомагає спростити взаємодію з API та зробити HTTP-запити в легко читабельний та підтримуваний спосіб.

Bootstrap - це відомий та широко використовуваний фреймворк для фронтенд-розробки веб-додатків і веб-сайтів. Він надає набір готових компонентів, стилів і JavaScript-функціоналу, які допомагають швидко і зручно створювати сучасні та ефективні інтерфейси користувача. Bootstrap розроблений командою від Twitter і є одним із найпопулярніших фреймворків у світі веб-розробки.

Основні характеристики та можливості Bootstrap включають:

Готові компоненти: Bootstrap надає широкий набір готових компонентів,

таких як кнопки, форми, меню, модальні вікна, каруселі, навігація та багато інших. Це дозволяє розробникам швидко і легко використовувати стандартні елементи веб-інтерфейсу.

Сітка (Grid System): Bootstrap має міцну сітку для створення респонсивних дизайнів, яка пристосовується до різних розмірів екрану. Вона дозволяє створювати додатки, які виглядають добре на різних пристроях, включаючи комп'ютери, планшети і смартфони.

Стилі та теми: Bootstrap надає стилізацію для тексту, заголовків, таблиць, форм, кнопок і багатьох інших елементів. Крім того, існують теми Bootstrap, які дозволяють швидко змінити зовнішній вигляд додатку за допомогою зміни кольорів та стилів.

Підтримка JavaScript: Bootstrap містить JavaScript-компоненти, такі як модальні вікна, каруселі, спливаючі підказки, які додають інтерактивність до веб-сторінок.

Заходи безпеки: Bootstrap включає в себе заходи безпеки, такі як захист від Cross-Site Scripting (XSS) і Cross-Site Request Forgery (CSRF), які допомагають забезпечити безпеку додатків.

Документація: Bootstrap надає докладну документацію та приклади використання, які полегшують розробникам вивчення та використання фреймворку.

Bootstrap може бути використаний як самостійний фреймворк, або в поєднанні з іншими технологіями, такими як HTML, CSS, JavaScript, і фреймворками JavaScript, такими як React або Vue.js. Це робить його популярним вибором для широкого спектру проектів веб-розробки. [23]

Cookie бібліотека для Vue - це JavaScript бібліотека, яка призначена для роботи з HTTP-кукі у Vue.js додатках. Куки (cookies) - це невеликі рядки даних, які зберігаються на боці клієнта (у веб-переглядачі) і використовуються для зберігання інформації, яка може бути використана на стороні сервера або в самому додатку.

Cookie бібліотеки допомагають спростити роботу з куками в Vue.js додатках шляхом надання зручного інтерфейсу для їх створення, читання, оновлення та

видалення. Зазвичай ці бібліотеки надають Vue компоненти або міксіни, які дозволяють вам взаємодіяти з куками відображуваного компонента.

Популярні бібліотеки для роботи з куками в Vue.js включають vue-cookies і vue-cookie-law. Вони дозволяють легко керувати куками в вашому додатку, дозволяють встановлювати терміни дії, шифрувати дані, якщо це потрібно, та інше. Ви можете встановити ці бібліотеки в свій проект і використовувати їх для зручної роботи з куками у ваших Vue.js додатках.

core-js - це бібліотека JavaScript, яка надає поліфілли (polyfills) для нових функцій та методів, які були введені в стандарті ECMAScript (ES) після версії ES5. ECMAScript - це стандарт мови JavaScript, із якого виникають всі сучасні реалізації JavaScript.

Поліфіли (або поліфілли) - це код, який дозволяє використовувати нові функції мови JavaScript у старих веб-переглядачах, які не підтримують ці функції. core-js надає такі поліфіли для забезпечення сумісності JavaScript-коду з різними веб-переглядачами.

Ця бібліотека дозволяє розробникам використовувати нові можливості мови JavaScript навіть в старих або менш підтримуваних веб-переглядачах, не вдаючись до багатьох умовних конструкцій або використання власних реалізацій. Це особливо корисно, коли ви розробляєте веб-додатки, які мають підтримувати широкий спектр веб-переглядачів.

Щоб використовувати core-js, ви можете встановити його через npm (Node Package Manager) або інший менеджер залежностей і імпортувати необхідні поліфіли у свій JavaScript-код. Таким чином, ви зможете спростити роботу з сучасними функціями мови JavaScript, навіть у веб-переглядачах, які не підтримують їх вбудовано.

CORS (Cross-Origin Resource Sharing) - це механізм безпеки в браузерах, який обмежує запити, які веб-сторінки можуть робити до інших джерел на інших доменах. Це робиться для захисту користувачів від атак, таких як Cross-Site Request Forgery (CSRF), та для забезпечення безпеки веб-додатків.

CORS бібліотеки - це бібліотеки або модулі, які допомагають розробникам

налаштовувати та керувати політикою CORS у своїх веб-додатках або на серверному боці. Вони надають інструменти для визначення того, які запити будуть допущені до ресурсів на сервері та які будуть заборонені.

Найпоширенішим способом налаштування CORS є додавання HTTP-заголовків до відповідей сервера. Для цього використовуються заголовки, такі як Access-Control-Allow-Origin, Access-Control-Allow-Methods, Access-Control-Allow-Headers і інші. CORS бібліотеки можуть надавати зручний спосіб встановлення цих заголовків та налаштування правил для обробки запитів.

Деякі популярні CORS бібліотеки та рішення включають cors для серверів Node.js, flask-cors для веб-сервера Flask, а також різні бібліотеки та розширення для різних серверних мов та фреймворків. Вони допомагають розробникам забезпечувати безпеку та контроль над доступом до своїх ресурсів через мережу Інтернет.

vue-router дозволяє визначати маршрути (routes), які відповідають різним URL-шляхам і відображають відповідні компоненти Vue.js при доступі до цих URL-шляхів. Вона також дозволяє виконувати навігацію між сторінками та передавати параметри між компонентами.

Основні функції та можливості vue-router включають:

Визначення маршрутів: Ви можете визначити, які компоненти Vue.js будуть відображатися при доступі до різних URL-шляхів.

Вкладені маршрути: Ви можете створювати вкладені маршрути, що дозволяють організувати складні структури маршрутизації.

Динамічні маршрути: vue-router підтримує динамічні маршрути, де параметри URL можуть використовуватися для відображення відповідних даних на сторінці.

Історія переходів: Бібліотека дозволяє використовувати історію переходів, таку як кнопки "Назад" і "Вперед" у веб-переглядачі.

Середовище плагінів: Ви можете легко розширювати функціональність vue-router, додаючи плагіни.

vue-router ідеально підходить для створення додатків зі складною

маршрутизацією та навігацією між сторінками. Вона добре інтегрується зі структурою Vue.js і допомагає розробникам створювати веб-додатки зі зручною маршрутизацією.

@babel/core - це ядро Babel, яке є однією з ключових компонентів Babel, популярного інструмента для трансформації та компіляції коду JavaScript. Babel використовується для перетворення сучасного коду JavaScript, написаного з використанням нових функцій та синтаксису, на код, який може бути виконаний в старих веб-переглядачах або інших середовищах.

@babel/core відповідає за наступні функції:

Парсинг (Parsing): Воно відповідає за перетворення вихідного коду JavaScript на абстрактне синтаксичне дерево (Abstract Syntax Tree або AST). AST представляє структуру та семантику коду.

Трансформація (Transformation): @babel/core дозволяє виконувати різні види трансформацій коду, такі як перетворення нового синтаксису в старий, видалення непідтримуваних функцій або додавання поліфілів для старих веб-переглядачів.

Генерація (Code Generation): Після виконання трансформацій @babel/core може генерувати новий код на основі AST.

@babel/core є основною складовою Babel, і його можна використовувати як часть інтеграції Babel в різні проекти. Розробники можуть налаштовувати правила трансформації та встановлювати різні плагіни для перетворення коду так, як це потрібно в конкретному проекті.

Використання @babel/core дозволяє розробникам писати сучасний JavaScript, зберігаючи сумісність зі старими веб-переглядачами та іншими середовищами, які не підтримують нові функції мови.

Нижче наведено структуру проекту.

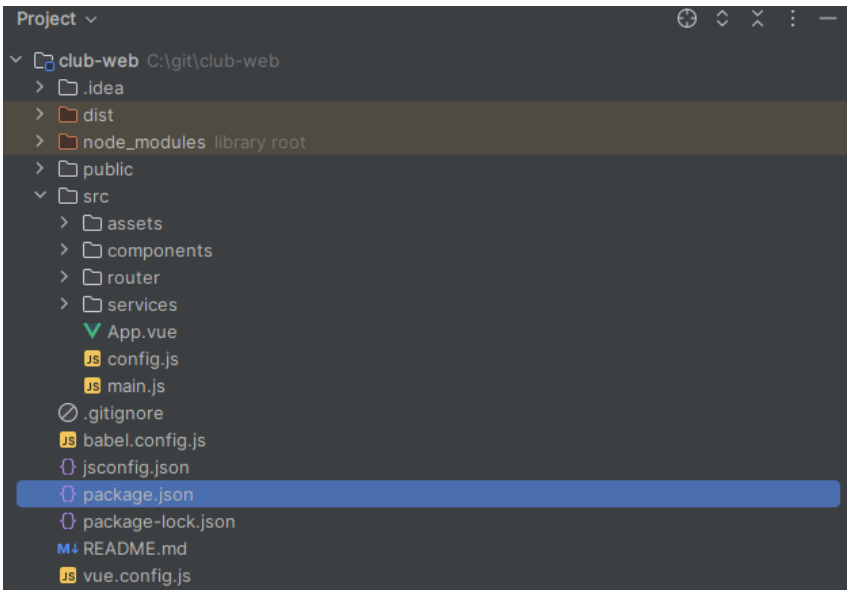


Рисунок 3.1 - Структура фротенд додатку

3.2 Використання «Axios» для забеспечення RESTful API

Axios - це бібліотека для виконання HTTP-запитів у JavaScript-додатках, таких як веб-додатки та мобільні додатки. Вона дозволяє здійснювати HTTP-запити до серверів RESTful API і взаємодіяти з ними. Ось як Axios забезпечує роботу з RESTful API:

Зроблення запитів: Axios надає простий та зручний інтерфейс для здійснення HTTP-запитів, таких як GET, POST, PUT, DELETE і багато інших. Ці запити можуть бути використані для взаємодії з різними ресурсами на сервері, які представляють різні дії у RESTful API. [24]

Обробка відповідей: Axios автоматично обробляє відповіді від сервера і надає зручний спосіб доступу до даних, отриманих від RESTful API. Ви можете отримувати дані у різних форматах, таких як JSON, XML або тексти, і подальше їхнє обробка може бути дуже простою.

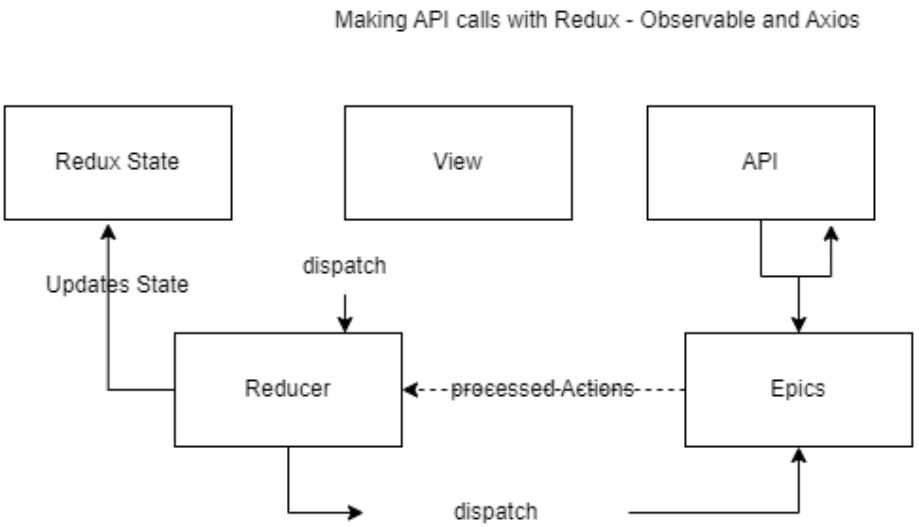


Рисунок 3.2 - Використання Axios

Налаштування параметрів запиту: Axios дозволяє вам легко налаштовувати параметри запиту, такі як заголовки, параметри запиту (query parameters), дані запиту (request data) та інші. Це дозволяє точно визначити, які дані ви надсилаєте на сервер і які параметри ви передаєте у запиті.

Обробка помилок: Axios надає інструменти для обробки помилок, які можуть виникнути під час виконання запитів. Ви можете перевіряти статус коди, обробляти помилки і визначати відповідну логіку для різних станів запиту.

Міжсайтовий обмін ресурсами (CORS): Axios дозволяє виправити проблеми з CORS, що можуть виникнути при взаємодії з віддаленим RESTful API на іншому домені. Ви можете налаштувати CORS-заголовки та різні параметри для взаємодії з сервером.

Нижче наведено приклад використання Axios для відправки запитів на бекенд частину, реалізація якої було ретельно описано в 3 розділі даної магістрської роботи.

```
export default {
  data() {
    return {
      user: {},
      wallet: {},
    };
  },
  mounted() {
    this.fetchUser();
  },
  methods: {
    async fetchUser() {
      try {
        const response = await api.get( url: "/profile");
        this.user = response.data;
        this.wallet = response.data.wallet;
      } catch (error) {
        console.error("Error fetching messages:", error);
      }
    },
  },
};
</script>
```

Рисунок 3.3 - Приклад застосування Axios для відправки запиту

Загалом, Axios робить взаємодію з RESTful API дуже зручною і забезпечує широкий набір інструментів для роботи з віддаленими ресурсами. Він є популярним вибором для багатьох розробників JavaScript, оскільки надає високий рівень функціональності та надійності.

3.3 Використання Bootstrap для побудови дизайну додатку

Bootstrap - це відомий фреймворк для розробки веб-сайтів та веб-додатків. Він надає набір заздалегідь стилізованих компонентів, CSS-стилів та JavaScript-засобів, які значно спрощують і прискорюють процес створення і розробки сучасних та адаптивних веб-додатків. Ось декілька ключових аспектів Bootstrap з точки зору побудови дизайну сайту:

Готові компоненти: Bootstrap надає багато готових компонентів, таких як кнопки, навігаційні панелі, каруселі, модальні вікна, форми, таблиці та багато інших. Ці компоненти мають стилі та розміри, які можна налаштовувати, і вони дозволяють створювати різноманітні елементи інтерфейсу без необхідності вручну писати CSS та JavaScript.

Адаптивний дизайн: Bootstrap побудований з урахуванням адаптивності, що означає, що сайти та додатки, розроблені з використанням Bootstrap, коректно відображаються на різних пристроях та розмірах екранів. Фреймворк включає в себе сітку (grid system), яка спрощує створення резинових та адаптивних макетів.

Компоненти для мобільних пристроїв: Bootstrap має вбудовані компоненти та класи, які полегшують створення мобільної дружньої версії веб-додатка або сайту.

Налаштовані теми: Bootstrap дозволяє налаштовувати кольори, шрифти та інші стилі, щоб відповідати дизайну вашого проекту. Ви можете використовувати змінні CSS (Sass або Less) для легкої кастомізації.

Підтримка браузерів: Bootstrap ретельно тестується та підтримується для багатьох сучасних браузерів, що дозволяє забезпечити сумісність та коректну роботу на різних платформах.

Зручна документація: Bootstrap має докладну та зрозумілу документацію з прикладами використання всіх компонентів та можливостей фреймворку, що робить процес розробки більш зручним.

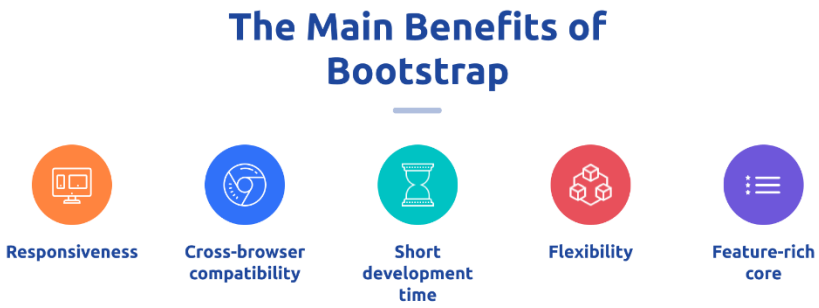


Рисунок 3.4 - Основні переваги Bootstrap [25]

Завдяки цим можливостям Bootstrap стає потужним інструментом для швидкого створення сучасних та ефективних дизайнів веб-сайтів та веб-додатків.

Таким чином, використання Vue.js разом з Bootstrap для побудови фронтенд додатку має численні переваги:

Простота та продуктивність: Vue.js надає зручний і простий спосіб розробки

інтерактивних веб-додатків. Із використанням Bootstrap, ви можете швидко створювати сучасний та ефективний дизайн без потреби великої кількості власного CSS і JavaScript коду.

Адаптивність: Bootstrap надає адаптивний дизайн, що дозволяє вашому додатку коректно відображатися на різних пристроях та розмірах екранів. Це допомагає покращити користувацький досвід для мобільних користувачів.

Зручність інтеграції: Vue.js та Bootstrap легко інтегруються один з одним. Ви можете використовувати Bootstrap компоненти в компонентах Vue.js і керувати їх станом та даними.

Багато готових компонентів: Bootstrap надає набір готових компонентів, таких як кнопки, форми, навігація, каруселі і багато інших. Це спрощує розробку та полегшує відтворення стандартних елементів інтерфейсу.

Спрощений управління стилями: Bootstrap дозволяє вам зручно керувати стилями та кольорами за допомогою класів та змінних CSS. Ви можете налаштовувати дизайн додатку, використовуючи готові засоби стилізації.

Швидка розробка: Використання Vue.js і Bootstrap спрощує розробку фронтенду, оскільки вони допомагають уникнути багатьох повторюваних завдань та прискорюють процес створення додатку.

Узагальнюючи, використання Vue.js та Bootstrap разом спрощує і прискорює розробку фронтенд додатків, забезпечуючи гнучкість та стильність дизайну, а також покращує користувацький досвід. Це потужна комбінація для будь-якої веб-розробки, яка робить процес більш продуктивним і ефективним

РОЗДІЛ 4 РОЗГОРТАННЯ БАЗИ ДАНИХ ПРОЕКТУ

4.1 Аналіз оптимальної схеми бази данних, побудова таблиць

Як відомо з першого розділу даної магістрської роботи, в якості бази даних було прийнято використовувати PostgreSQL.

Оптимальна налаштована база даних PostgreSQL зазвичай враховує різні аспекти, які допомагають досягти найкращої продуктивності та надійності для конкретного застосування. Ось ключові аспекти оптимальної PostgreSQL бази даних:

Планування і проектування схеми: Схема бази даних повинна бути ретельно спроектована, враховуючи структуру даних та типи запитів, які будуть виконуватися. Це включає в себе визначення таблиць, стовпців, індексів та зв'язків між таблицями.

Індексація: Використання індексів для швидкого доступу до даних є ключовим аспектом оптимізації PostgreSQL. Визначення правильних індексів для стовпців, які часто використовуються в запитах, може значно покращити продуктивність.

Налаштування конфігурації: PostgreSQL має численні налаштування, які дозволяють оптимізувати його роботу під конкретні потреби. Налаштування, такі як кількість одночасних з'єднань, буфери пам'яті і параметри запитів, можуть бути налаштовані для забезпечення найкращої продуктивності.

Резервне копіювання та відновлення: Регулярне резервне копіювання бази даних та забезпечення можливості швидкого відновлення в разі виникнення проблем є важливими аспектами надійності.

Моніторинг та налагодження: Використання інструментів для моніторингу та профілювання може допомогти виявити проблеми продуктивності та оптимізувати запити та конфігурацію.

Оновлення та безпека: Періодичне оновлення PostgreSQL до останньої версії та вчасне вирішення відомих безпекових проблем важливо для забезпечення

безпеки та надійності бази даних.

Реплікація та високодоступність: В залежності від потреб додатку, може бути використана реплікація для забезпечення високодоступності та збереження резервних копій даних. [26]

Планування робочого часу та архівування даних: Оптимальна база даних повинна мати стратегію планування робочого часу для оптимізації завантаження та архівування даних, які не активно використовуються.

Захист даних та доступ до них: Важливо захищати дані від несанкціонованого доступу та втрати. PostgreSQL надає інструменти для налаштування безпеки даних.

Тестування та моніторинг виробництва: Ретельне тестування та моніторинг робочого середовища допомагають виявляти та усувати проблеми, зберігаючи високу надійність та продуктивність.

Оптимізація та налагодження PostgreSQL відображається на продуктивності та надійності бази даних і, відповідно, на продуктивності веб-додатка. Оптимальна настройка бази даних враховує вимоги та потреби конкретного проекту і може значно покращити його продуктивність та надійність

В розробці існують два основних підходи для проектування баз даних називаються "перш за все таблиці" (англ. Table-First) і "перш за все сутності" (англ. Entity-First). Ось їхні основні відмінності:

Підхід «По-перше, таблиці» (Table-First):

Починається з проектування таблиць і структури бази даних.

Таблиці та структура бази даних визначаються заздалегідь без врахування логіки додатку або сутностей, які вони представляють.

Дані про сутності визначаються пізніше, і вони повинні відповідати структурі таблиць.

Цей підхід зазвичай використовується в традиційних реляційних базах даних, де таблиці та структура мають велике значення.

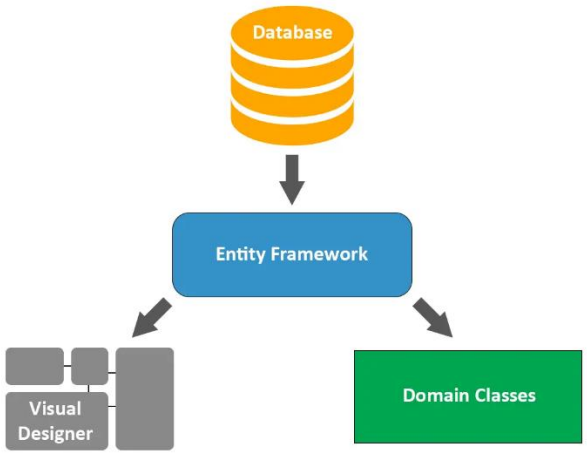


Рисунок 4.1 - «По-перше, таблиці» [27]

Підхід «По-перше, сутності» (Entity-First):

Починається з проектування сутностей (класів або об'єктів), які використовуються в додатку.

Сутності визначаються на основі логіки додатку і вимог до даних.

Після визначення сутностей, структура таблиць та бази даних налаштовується для відображення цих сутностей.

Цей підхід часто використовується в сучасних об'єктно-орієнтованих базах даних та об'єктно-реляційних відображень (ORM).

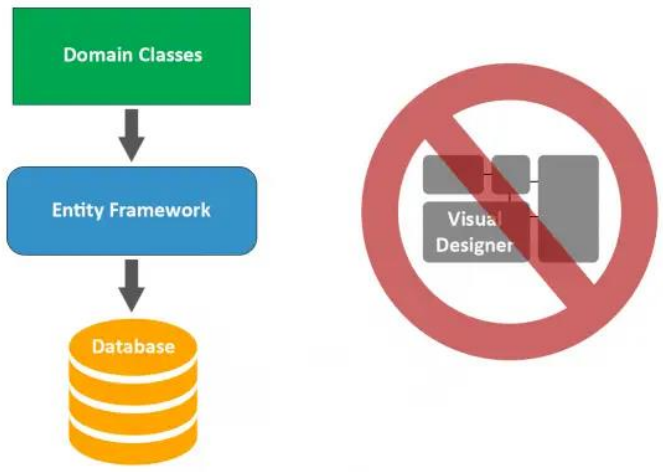


Рисунок 4.2 - «По-перше, сутності» [27]

Обираючи між цими підходами, слід враховувати потреби та характер

проекту, а також конкретний контекст використання бази даних. Підхід "По-перше, таблиці" може бути використаний для дуже структурованих даних, де структура даних визначається заздалегідь і має велике значення. Підхід "По-перше, сутності" зазвичай використовується в сучасних додатках, де логіка додатку та потреби користувачів грають ключову роль, і де важлива гнучкість та швидкість розробки.

В даній магістрській роботі було використано підхід "По-перше, сутності" (Entity-First). Для побудови таблиць було використано Hibernate, принцип роботи якого описано в розділі 2 даної магістрської роботи.

4.2 Аналіз та вибір оптимального первинного ключа

Первинний ключ (Primary Key) в базі даних - це унікальний ідентифікатор, який визначає кожен запис (рядок) в таблиці. Властивість первинного ключа полягає в тому, що вона повинна бути унікальною для кожного запису в таблиці, тобто не може існувати два записи з однаковим значенням первинного ключа.

Основні характеристики первинного ключа:

Унікальність: Кожен запис повинен мати унікальне значення первинного ключа. Це гарантує, що кожен запис може бути однозначно ідентифікованим за допомогою первинного ключа.

Обов'язковість: Поле, визначене як первинний ключ, не може мати значення NULL (порожнє значення). Кожен запис повинен мати значення первинного ключа.

Ідентифікація: Первинний ключ використовується для ідентифікації конкретного запису в таблиці. Це допомагає в здійсненні зв'язків між таблицями і забезпечує цілісність даних. [28]

Швидкість пошуку: База даних може ефективно здійснювати пошук та фільтрацію даних за значеннями первинного ключа, оскільки цей ідентифікатор визначає унікальний запис.

Зовнішні ключі: Первинний ключ часто використовується в якості посилання (зовнішнього ключа) в інших таблицях, що дозволяє створювати зв'язки між записами різних таблиць.

Прикладами типів, які можуть бути використані як первинний ключ, є цілі числа, рядки, унікальні ідентифікатори (наприклад, UUID), дати та інші унікальні

значення. Вибір типу первинного ключа залежить від конкретних потреб проекту та особливостей бази даних.

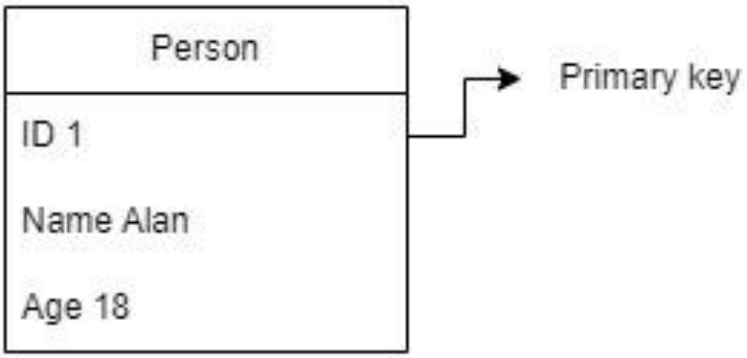


Рисунок 4.4 - Приклад первинного ключа

В даній магістрській роботі у якості первинного ключа було використано GUID (Global Unique Identifier).

GUID (Global Unique Identifier), також відомий як UUID (Universally Unique Identifier), - це текстовий або числовий первинний ключ, який має численні переваги порівняно з іншими типами первинних ключів. Ось деякі з переваг GUID як первинного ключа:

Унікальність: GUID гарантує глобальну унікальність. Кожен створений GUID у світі унікальний, що дозволяє використовувати його як первинний ключ для ідентифікації записів у великій мережі без ризику конфліктів або дублювання ключів.

Надійність: GUID може бути створений локально на пристрої без необхідності взаємодії з централізованим сервісом для генерації унікальних ідентифікаторів. Це дозволяє зберігати унікальні ключі, навіть якщо пристрій не підключений до мережі.

Конфіденційність: У випадках, коли конфіденційність важлива, текстовий GUID може бути менш помітним для зовнішніх спостерігачів порівняно з числовими ключами. Текстовий GUID може бути хешований або засолений для додаткового рівня захисту.

Підтримка розподілених систем: GUID особливо корисний для розподілених систем і мереж, де записи можуть створюватися на різних серверах або пристроях.

Кожен сервер може створити свій унікальний GUID без потреби узгодження з іншими серверами.

Підтримка реплікації та референціювання: GUID легко впроваджується в системи реплікації та референціювання, оскільки він зберігає свою унікальність навіть при копіюванні даних між серверами або базами даних.

Гнучкість: Текстовий GUID може бути легко інтерпретованим і включати в себе інформацію, яка може бути корисною для аналізу даних.

Однак важливо враховувати, що GUID мають певні недоліки, такі як більший обсяг зберігання порівняно з числовими ключами і погіршену продуктивність при використанні як індексів у великих таблицях, проте переваги є більшими за недоліки, тому його вибір є обгрунтованим.

4.3 Побудова зв’язків між сутносями в проекті

У базах даних існує кілька типів зв'язків, які визначають, як таблиці взаємодіють одна з одною. Ось основні типи зв'язків:

Зв'язок один-до-одного (One-to-One):

У цьому типі зв'язку один запис в одній таблиці пов'язаний з одним записом в іншій таблиці.

Зазвичай цей тип використовується для розділення пов'язаних але великих таблиць, які можна розмістити в окремих таблицях для покращення продуктивності і читабельності коду.

Рисунок 4.4 - Зв'язок один-до-одного (One-to-One)

Зв'язок один-до-багатьох (One-to-Many):

У цьому типі зв'язку один запис в одній таблиці пов'язаний з багатьма

записами в іншій таблиці.

Наприклад, один автор може мати багато книг, і кожна книга посилається на одного автора.

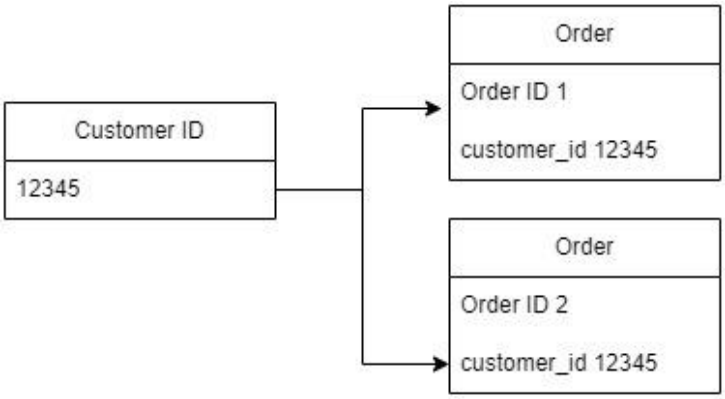


Рисунок 4.5 - Зв'язок один-до-багатьох (One-to-Many)

Зв'язок багато-до-одного (Many-to-One):

У цьому типі зв'язку багато записів в одній таблиці посилаються на один запис в іншій таблиці.

Наприклад, багато студентів можуть бути записані на одного викладача.

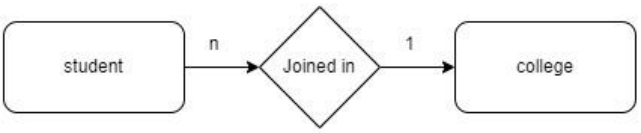


Рисунок 4.6 - Зв'язок багато-до-одного (Many-to-One)

Зв'язок багато-до-багатьох (Many-to-Many):

У цьому типі зв'язку багато записів в одній таблиці пов'язані з багатьма записами в іншій таблиці.

Наприклад, багато студентів можуть бути записані на багато курсів, і кожен студент може вивчати багато курсів.

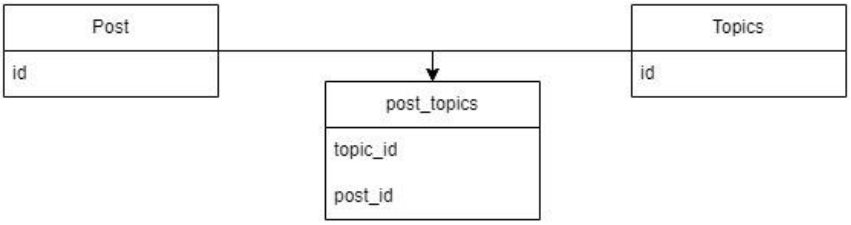


Рисунок 4.7 - Зв'язок багато-до-багатьох (Many-to-Many)

Зв'язок батько-дитина (Parent-Child):

Це тип зв'язку використовується для організації ієрархічних даних. Кожен запис посилається на батька і може мати декілька дітей.

Наприклад, дерево категорій товарів, де кожна категорія може мати багато підкатегорій.

Кожен з цих типів зв'язків визначає, як дані взаємодіють між собою і як вони організовані в базі даних.

В даній магістрській роботі були реалізовані усі типи зв’язків, за допомогою бібліотеки Hibernate. Приклад застосування було наведено у розділі 2 даної магістрської роботи на рисунку 2.4.

Таким чином, використання PostgreSQL для побудови RESTful API має численні переваги, які роблять його доцільним вибором для багатьох проектів. Ось деякі з цих переваг:

Відкритий джерело і безкоштовність: PostgreSQL є системою управління базами даних з відкритим кодом і повністю безкоштовним для використання. Це зменшує витрати на ліцензії і ресурси.

Завдовжки та надійність: PostgreSQL відомий своєю високою надійністю та стабільністю. Він здатен обробляти великі обсяги даних і запитів і має ряд механізмів для запобігання втраті даних.

Сумісність і розширюваність: PostgreSQL підтримує стандарти SQL і має широкий набір функцій і розширень. Ви можете легко інтегрувати PostgreSQL з іншими інструментами і сервісами.

Підтримка геоданих: PostgreSQL має розширення PostGIS, яке надає підтримку геоданих і географічних запитів. Це робить його ідеальним вибором для додатків, які вимагають обробки просторових даних.

Розширені можливості індексації: PostgreSQL надає широкий вибір типів індексів, включаючи більш складні, такі як GIN і GiST, що робить його ефективним для пошуку і фільтрації даних.

Можливості безпеки: PostgreSQL надає багато можливостей забезпечення безпеки даних, включаючи вбудовані механізми шифрування, аутентифікації і

авторизації. [29]

Підтримка реплікації: PostgreSQL має вбудовану підтримку реплікації, що дозволяє створити резервні копії даних та підвищити доступність додатку.

Підтримка JSON і XML: PostgreSQL підтримує роботу з документами JSON і XML, що робить його ідеальним для створення RESTful API, які взаємодіють з додатками та клієнтами, що використовують ці формати даних.

Широкий вибір бібліотек та інструментів: PostgreSQL має активну спільноту та багато різних бібліотек та інструментів для розробки RESTful API, включаючи популярний фреймворк Django з підтримкою PostgreSQL. [30 - 33]

Загалом, PostgreSQL є потужною та надійною системою управління базами даних, яка надає усі необхідні можливості для створення RESTful API для вашого додатку. Вибір PostgreSQL може допомогти забезпечити високу продуктивність, надійність та безпеку вашого API.

ВИСНОВКИ

В даній магістерській роботі було розроблено додаток, який використовує комбінацію Vue.js, Spring Boot, PostgreSQL та Bootstrap для побудови RESTful API. Ця комбінація технологій дозволила досягнути високої ефективності та надійності додатку, і ось деякі основні висновки:

Ефективність технічного стеку: Використання Vue.js для фронтенду та Spring Boot для бекенду забезпечує оптимальну розділеність відповідальностей і дозволяє командам розробників працювати паралельно над фронтендом та бекендом. Це сприяє швидкій розробці та покращує продуктивність.

Відмінна підтримка бази даних: Використання PostgreSQL як системи управління базами даних забезпечує високий рівень надійності та швидкості доступу до даних. PostgreSQL підтримує розширення для роботи з геоданими, JSON і багато інших функцій, що дозволяє розробити багатофункціональний додаток.

Зручний інтерфейс користувача: Використання Bootstrap дозволило створити зручний і адаптивний інтерфейс користувача. Шаблони та компоненти Bootstrap дозволяють швидко створити привабливий інтерфейс без значних зусиль.

RESTful API: Використання Spring Boot для створення RESTful API забезпечує простий та консистентний спосіб взаємодії з додатком через HTTP. Це спрощує інтеграцію з іншими додатками та сервісами.

Гнучкість та розширюваність: Всі компоненти технічного стеку (Vue.js, Spring Boot, PostgreSQL, Bootstrap) є гнучкими і легко розширюються. Це дозволяє легко внести зміни в додаток та розширити його функціонал.

Вже розроблений додаток: Розробка додатку з використанням цього технічного стеку є ефективним рішенням, оскільки цей стек вже використовується для інших проектів та має готовий функціонал. Це дозволяє економити час та ресурси при розробці нового додатку.

Загалом, комбінація Vue.js, Spring Boot, PostgreSQL та Bootstrap для побудови RESTful API дозволяє створити високоефективний та надійний додаток,

який легко розширюється і має зручний інтерфейс для користувачів. Цей технічний стек вже довів свою ефективність у практиці і є відмінним вибором для багатьох проектів.

ПЕРЕЛІК ПОСИЛАНЬ

1. D. Corradini, A. Zampieri, M. Pasqua, M. Ceccato Empirical Comparison of Black-box Test Case Generation Tools for RESTful APIs 2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM) (2021), pp. 226-236.

2. A. Ehsan, M.A.M.E. Abuhaliqa, C. Catal, D Mishra RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions Appl. Sci., 12 (2022), p. 4369, 10.3390/app12094369

3. https://www.geeksforgeeks.org/http-headers-keep-alive/

4. Andy Neumann, Nuno Laranjeiro, Jorge Bernardino An Analysis of Public REST Web Service APIs IEEE Transactions on Services Computing, 14 (4) (2021), pp. 957-970n.

5. Dessì D., Osborne F., Recupero D.R., Buscaldi D., Motta E., Sack H. AI-KG: an automatically generated knowledge graph of artificial intelligence ISWC 2020, Lecture notes in computer science, vol.12507, Springer (2020), pp. 127-143

6. LuoK. Enhanced grey wolf optimizer with a model for dynamically estimating the location of the prey Appl. Soft Comput. (2019)

7. KambojV.K. et al. An intensify Harris Hawks optimizer for numerical and engineering optimization problems Appl. Soft Comput. (2020)

8. A new artificial bee colony algorithm employing intelligent forager forwarding strategies Appl. Soft Comput. (2020)

9. A.C. Severin et al. Effects of a corrective heel lift with an orthopaedic walking boot on joint mechanics and symmetry during gait Gait Posture (2019)

10. https://howtodoinjava.com/interview-questions/sql-interview-questions-with-answers/

11. Сторчак К.П., Ткаленко О.М., Бурда Ю.О., Гарник Д.О. «Дослідження проблеми оптимізації продуктивності в мові програмування Java». Стаття у загальногалузевому науково-виробничому журналі «Зв’язок», м.Київ - №5(165), 2023. – С.15-20

12. Бурда Ю.О. «Дослідження технології побудови ефективного додатку з використанням restful api». Тези доповіді на I всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу україни і світу». – Київ, 28 листопада 2023 р.

13. К.П. Сторчак, О.М. Ткаленко, О.В. Полоневич, К.П. Косенкох., В.М. Чорна. «Пошук, обробка та аналіз інформації» Навчальний посібник, ДУТ, Київ-2018. – 127 с

14. Li Y and Jiang Z (2019). Assessing and optimizing the performance impact of the just-in-time configuration parameters - a case study on PyPy, Empirical Software Engineering, 24:4, (2323-2363), Online publication date: 1-Aug-2019

15. Scott Oaks " Java Performance, 2nd Edition" – Addison-Wesley, O'Reilly Media, Inc. 2020 – 714p

16. Matt Weisfeld, " The Object-Oriented Thought Process, 5th Edition" Addison-Wesley Professional 2019- 323 p

17. A.C. Carniel et al. A generic and efficient framework for flash-aware spatial indexing Inf. Syst.(2019)

18. Z. He et al. GeoBeam: a distributed computing framework for spatial data Comput. Geosci. (2019)

19. Impact of Disc Types on Database Performance 2022, 2022 IEEE 16th International Scientific Conference on Informatics, Informatics 2022 – Proceedings

20. https://www.abtasty.com/ci-cd/

21. https://www.edx.org/es/learn/software-development/ibm-test-and-behavior-driven-development-tdd-bdd

22. Tsoutsouras V., Anagnostopoulos I., Masouros D., Soudris D. A hierarchical distributed runtime resource management scheme for noc-based many-cores ACM Trans Embed Comput Syst, 17 (3) (2023)

23. Gadioli D., Vitali E., Palermo G., Silvano C. margot: A dynamic autotuning framework for self-aware approximate computing IEEE Trans Comput, 68 (5) (2019), pp. 713-728

24. Carvalho T. Programming and mapping strategies for embedded computing runtime adaptability (Ph.D. thesis) University of Porto, Faculty of Engineering, Porto, Portugal (2019)

25. https://getdevdone.com/blog/should-you-use-bootstrap-for-wordpress-theme-development.html

26. Garcia K.D., Carvalho T., Mendes-Moreira J., Cardoso J.M., de Carvalho A.C.P.L.F. A study on hyperparameter configuration for human activity recognition Álvarez F. Martínez, Lora A. Troncoso, Muñoz J.A. Sáez, Quintián H., Corchado E. (Eds.), 14th International conference on soft computing models in industrial and environmental applications, (SOCO 2019), Springer International Publishing (2019), pp. 47-56

27. https://www.ryadel.com/en/code-first-model-first-database-first-vs-comparison-orm-asp-net-core-entity-framework-ef-data/

28. Azevedo F.B. Cost reduction technique for mutation testing (Master's Thesis) Faculdade de Engenharia da Universidade do Porto, Porto, Portugal (2020)

29. Teixeira G., Bispo J.A., Correia F.F. Multi-language static code analysis on the lara framework Proceedings of the 10th ACM SIGPLAN international workshop on the state of the art in program analysis, SOAP 2021, Association for Computing Machinery, New York, NY, USA (2021), pp. 31-36

30. https://www.odinschool.com/blog/big-data/top-big-data-skills-to-future-proof-your-career-become-a-data-engineer-now

31. https://techreviewer.co/blog/postgresql-vs-mysql-which-relational-database-is-better

32. https://talentgrid.io/spring-boot-interview-questions/

33. https://www.knowledgefactory.net/2022/02/spring-boot-postgresql-vuejs-crud-example.html

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (презентація)

Державний університет інформаційно-комунікаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

КВАЛІФІКАЦІЙНА РОБОТА
на тему:
“Розробка веб-додатку користувачів на основі технологій Java Spring, Vue JS”

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Виконав: здобувач вищої освіти гр. ІСДМ-51
Юрій БУРДА
Керівник: к.т.н., доцент кафедри ІПЗАС
Оксана ТКАЛЕНКО

Київ - 2023

- ***Актуальність теми:*** Розробка веб-додатку користувачів на основі технологій Java Spring і Vue JS з використанням PostgreSQL та стандартів REST надзвичайно важлива в сучасному інформаційному та технологічному світі. Ураховуючи ці аспекти та враховуючи ваші конкретні вимоги до веб-додатків на базі Java Spring, Vue JS, PostgreSQL та REST, можна визначити актуальність теми для дослідження та розробки в цій галузі.
- ***Об`єкт дослідження:*** процес розробки веб-додатку для користувачів з використанням технологій Java Spring і Vue JS, зі спеціальним акцентом на взаємодію та інтеграцію між цими технологіями
- ***Предмет дослідження:*** Spring app.
- ***Мета дослідження:*** Метою дослідження є детальне вивчення та аналіз процесу розробки веб-додатку користувачів, використовуючи технології Java Spring і Vue JS з інтеграцією PostgreSQL та використанням стандартів REST.
- ***Завдання дослідження:***
  - Побудова оптимального REST app з використанням java Spring
  - Дослідження оптимальної структури БД
  - Визначення та побудова фронтенд частини

2

Огляд технологічного стеку та архітектурні особливості RESTful API

- RESTful API (Representational State Transfer API) - це архітектурний стиль для створення веб-сервісів, який базується на принципах та обмеженнях, які описані в статті Роя Філдінга "Архітектурний стиль REST" (Representational State Transfer)
- Ресурси (Resources): У RESTful API дані та функції представлені як ресурси, кожен з яких ідентифікується унікальним URL. Наприклад, ресурсом може бути "користувач", "замовлення" або "продукт«
- HTTP методи: REST використовує стандартні HTTP методи для взаємодії з ресурсами. HTTP методи, що використовуються в RESTful API, відіграють важливу роль у взаємодії між клієнтом і сервером.

Схема стандартного RESTful API

-

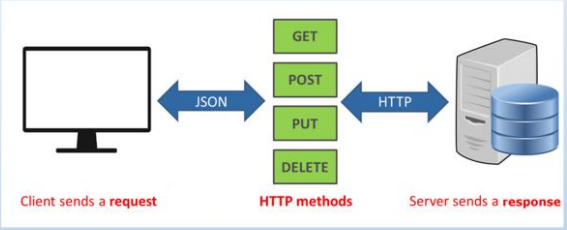


Рисунок 1.1 Схема стандартного RESTful API

Ідемпотентність та безпечність

- **Ідемпотентність** - це важливий аспект проектування та реалізації RESTful API, який визначає, як ведуть себе HTTP-запити в контексті повторних викликів. Поняття ідемпотентності означає, що певний HTTP-запит може бути виконаний багато разів, і кожен наступний виклик має той же результат, що і перший, без негативних наслідків або небажаних змін

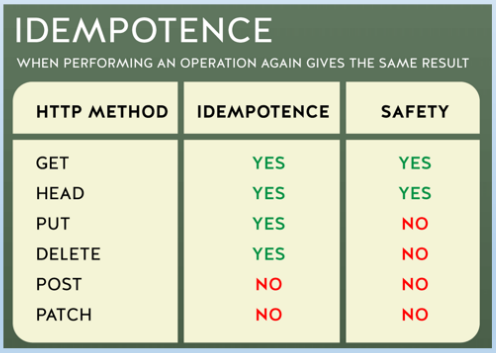


Рис. 1.2 Ідемпотентність та безпечність

Архітектура Spring

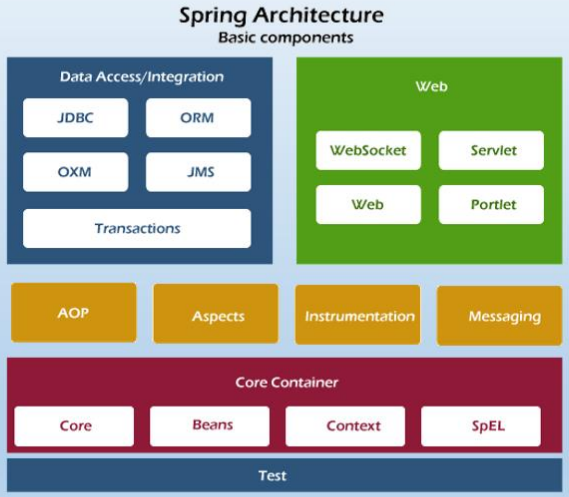


Рис 1.4 Архітектура Spring

Порівняльна характеристика фреймверків

METRIC	MICRONAUT 2.0 M2	QUARKUS 1.3.1	SPRING 2.3 M3
Compile Time	1.48s	1.45s	1.33s
Test Time ./mvn test	4.3s	5.8s	7.2s
Start Time Dev Mode	420ms	866ms (1)	920ms
Start Time Production java -jar myjar.jar	510ms	655ms	1.24s
Time to First Response	960ms	890ms	1.85s
Requests Per Second (2)	79k req/sec	75k req/sec	75k req/sec
Request Per Second -Xmx18m	50k req/sec	46k req/sec	46k req/sec
Memory Consumption After Load Test (-Xmx128m) (4)	290MB	390MB	480MB
Memory Consumption After Load Test (-Xmx18m) (4)	249MB	340MB	430MB

Ідемпотентність та безпечність

- **Ідемпотентність** - це важливий аспект проектування та реалізації RESTful API, який визначає, як ведуть себе HTTP-запити в контексті повторних викликів. Поняття ідемпотентності означає, що певний HTTP-запит може бути виконаний багато разів, і кожен наступний виклик має той же результат, що і перший, без негативних наслідків або небажаних змін

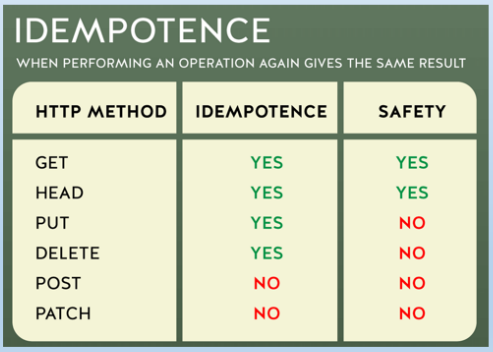


Рис. 1.2 Ідемпотентність та безпечність

Архітектура Spring

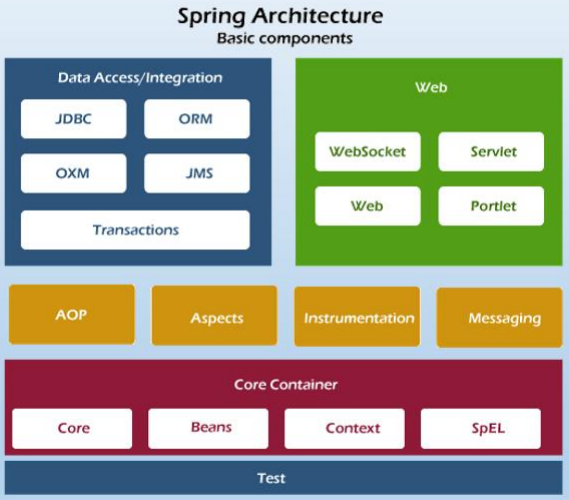


Рис 1.4 Архітектура Spring

Порівняльна характеристика фреймверків

METRIC	MICRONAUT 2.0 M2	QUARKUS 1.3.1	SPRING 2.3 M3
Compile Time	1.48s	1.45s	1.33s
Test Time ./mvn test	4.3s	5.8s	7.2s
Start Time Dev Mode	420ms	866ms (1)	920ms
Start Time Production java -jar myjar.jar	510ms	655ms	1.24s
Time to First Response	960ms	890ms	1.85s
Requests Per Second (2)	79k req/sec	75k req/sec	75k req/sec
Request Per Second -Xmx18m	50k req/sec	46k req/sec	46k req/sec
Memory Consumption After Load Test (-Xmx128m) (4)	290MB	390MB	480MB
Memory Consumption After Load Test (-Xmx18m) (4)	249MB	340MB	430MB

Структура Vue.js додатка та Схема типового Vue.js + Spring Boot застосунку

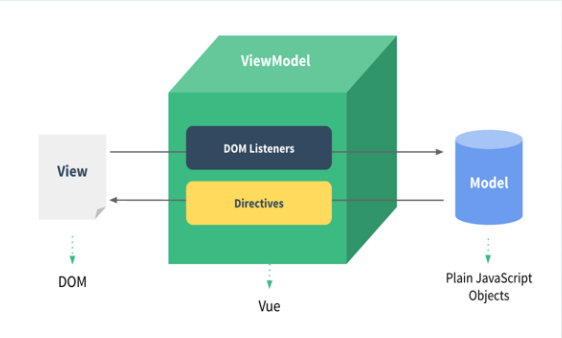


Рис 1.4 Структура Vue.js додатка

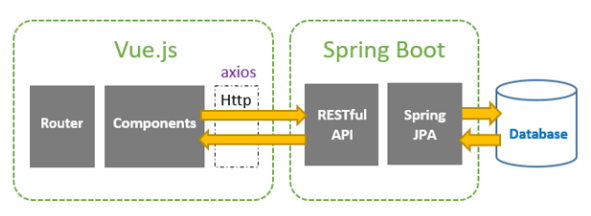


Рис 1.6 Схема Vue.js + Spring Boot застосунку

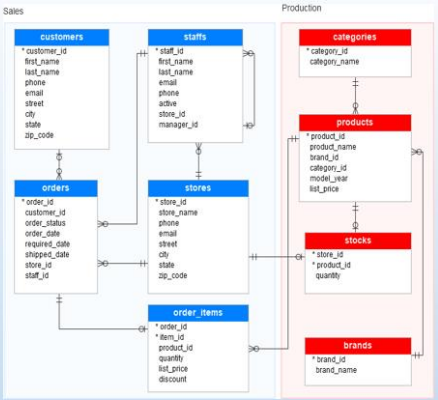


Рис 1.7 Схема SQL DB

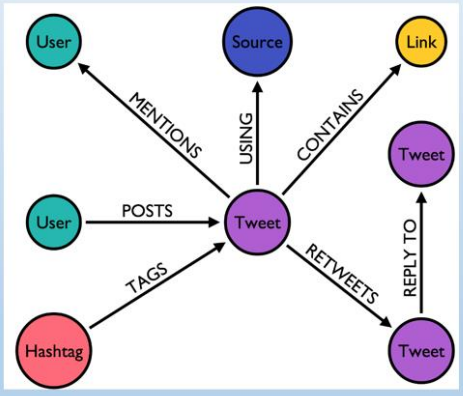


Рис 1.7 Схема графової бази даних

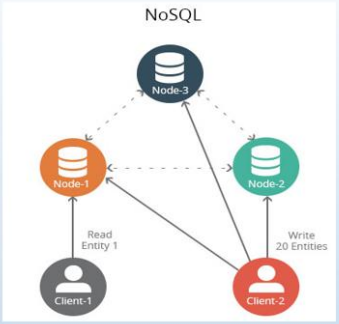


Рис 1.8 Схема NoSQL DB

Комбінована схема Vue.js + Spring Boot + PostgreSQL

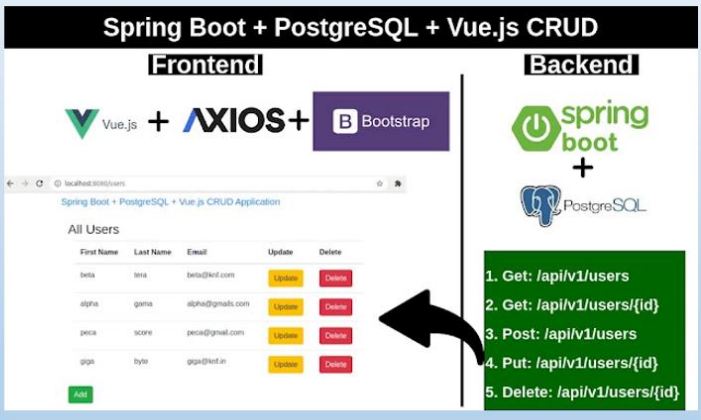


Рис 1.12 Комбінована схема Vue.js + Spring Boot + PostgreSQL

Життєвий цикл запиту в Spring MVC

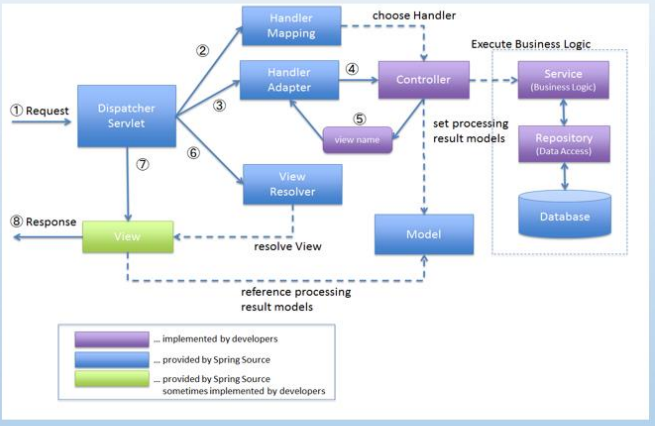


Рис 2.2. Життєвий цикл запиту в Spring MVC

Планування і проектування схеми

- Схема бази даних повинна бути ретельно спроектована, враховуючи структуру даних та типи запитів, які будуть виконуватися. Це включає в себе визначення таблиць, стовпців, індексів та зв'язків між таблицями.
- Індексація: Використання індексів для швидкого доступу до даних є ключовим аспектом оптимізації PostgreSQL. Визначення правильних індексів для стовпців, які часто використовуються в запитах, може значно покращити продуктивність.
- Налаштування конфігурації: PostgreSQL має численні налаштування, які дозволяють оптимізувати його роботу під конкретні потреби. Налаштування, такі як кількість одночасних з'єднань, буфери пам'яті і параметри запитів, можуть бути налаштовані для забезпечення найкращої продуктивності.

- Таким чином, використання PostgreSQL для побудови RESTful API має численні переваги, які роблять його доцільним вибором для багатьох проектів. Ось деякі з цих переваг:
- Відкритий джерело і безкоштовність: PostgreSQL є системою управління базами даних з відкритим кодом і повністю безкоштовним для використання. Це зменшує витрати на ліцензії і ресурси.
- Завдовжки та надійність: PostgreSQL відомий своєю високою надійністю та стабільністю. Він здатен обробляти великі обсяги даних і запитів і має ряд механізмів для запобігання втраті даних.
- Сумісність і розширюваність: PostgreSQL підтримує стандарти SQL і має широкий набір функцій і розширень. Ви можете легко інтегрувати PostgreSQL з іншими інструментами і сервісами.
- Підтримка геоданих: PostgreSQL має розширення PostGIS, яке надає підтримку геоданих і географічних запитів. Це робить його ідеальним вибором для додатків, які вимагають обробки просторових даних.

ВИСНОВКИ

- В даній магістерській роботі було розроблено додаток, який використовує комбінацію Vue.js, Spring Boot, PostgreSQL та Bootstrap для побудови RESTful API. Ця комбінація технологій дозволила досягнути високої ефективності та надійності додатку, і ось деякі основні висновки:
- Ефективність технічного стеку: Використання Vue.js для фронтенду та Spring Boot для бекенду забезпечує оптимальну розділеність відповідальностей і дозволяє командам розробників працювати паралельно над фронтендом та бекендом. Це сприяє швидкій розробці та покращує продуктивність.
- Відмінна підтримка бази даних: Використання PostgreSQL як системи управління базами даних забезпечує високий рівень надійності та швидкості доступу до даних. PostgreSQL підтримує розширення для роботи з геоданими, JSON і багато інших функцій, що дозволяє розробити багатофункціональний додаток.
- Зручний інтерфейс користувача: Використання Bootstrap дозволило створити зручний і адаптивний інтерфейс користувача. Шаблони та компоненти Bootstrap дозволяють швидко створити привабливий інтерфейс без значних зусиль.
- RESTful API: Використання Spring Boot для створення RESTful API забезпечує простий та консистентний спосіб взаємодії з додатком через HTTP. Це спрощує інтеграцію з іншими додатками та сервісами.
- Гнучкість та розширюваність: Всі компоненти технічного стеку (Vue.js, Spring Boot, PostgreSQL, Bootstrap) є гнучкими і легко розширюються. Це дозволяє легко внести зміни в додаток та розширити його функціонал.
- Вже розроблений додаток: Розробка додатку з використанням цього технічного стеку є ефективним рішенням, оскільки цей стек вже використовується для інших проектів та має готовий функціонал. Це дозволяє економити час та ресурси при розробці нового додатку.
- Загалом, комбінація Vue.js, Spring Boot, PostgreSQL та Bootstrap для побудови RESTful API дозволяє створити високоефективний та надійний додаток, який легко розширюється і має зручний інтерфейс для користувачів. Цей технічний стек вже довів свою ефективність у практиці і є відмінним вибором для багатьох проектів.

Апробація результатів дослідження:

1. Бурда Ю.О. «Дослідження проблеми оптимізації продуктивності в мові програмування Java». Стаття у загальногалузевому науково-виробничому журналі «Зв'язок», м.Київ - №5(165), 2023. – С.15-20
2. Бурда Ю.О. «Дослідження технології побудови ефективного додатку з використанням restful api». Тези доповіді на І всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу україни і світу». – Київ, 28 листопада 2023 р.

ДЯКУЮ ЗА УВАГУ!