Шелофаст М.В.
здобувач вищої освіти
Таврійський державний агротехнологічний університет
імені Дмитра Моторного
Науковий керівник: ст. викл. Тіменко А.В.

МЕТОДИКА СТВОРЕННЯ ДОДАТКІВ НА JAVA ТА SPRING BOOT

Анотація. *У статті розглянуто методику створення сучасних веб-додатків із використанням мови програмування Java та фреймворку Spring Boot. Акцент зроблено на поетапному процесі розробки, ключових інструментах та підходах, які забезпечують масштабованість і продуктивність систем. Виокремлено основні кроки: налаштування середовища, побудова архітектури, реалізація функціоналу та тестування.*

Ключові слова: *Java, Spring Boot, розробка додатків, мікросервіси, веб-технології.*

Shelofast M. Methodology for creating applications on Java and Spring Boot. *The article discusses the methodology of developing modern web applications using the Java programming language and the Spring Boot framework. The focus is on the step-by-step development process, key tools, and approaches that ensure system scalability and performance. The main stages are highlighted: environment setup, architecture design, functionality implementation, and testing.*

Key words: *Java, Spring Boot, application development, microservices, web technologies.*

Актуальність дослідження. У сучасному світі інформаційні технології відіграють ключову роль у розвитку бізнесу, науки та освіти. Кількість користувачів, які щодня взаємодіють із веб- та мобільними додатками, зростає в геометричній прогресії. Це зумовлює високі вимоги до якості програмного забезпечення: воно має бути продуктивним, масштабованим, безпечним і зручним у використанні.

Мова програмування Java уже понад 25 років утримує провідні позиції у світі завдяки своїй платформній незалежності, надійності та

розвиненій екосистемі. Вона використовується для розробки веб-додатків, серверних додатків, чат-ботів, у тому числі IoT [6]. Разом із цим, у процесі розробки програмних продуктів виникла потреба у фреймворках, які дозволяють значно прискорити створення додатків і водночас дотримуватися архітектурних стандартів. Одним із найуспішніших рішень став Spring Boot, який забезпечує просту конфігурацію, гнучку інтеграцію з іншими інструментами та підтримку сучасних підходів [7; 9].

Особливої актуальності тема набуває в контексті переходу компаній до хмарних рішень та використання мікросервісних систем, які дозволяють масштабувати сервіси незалежно один від одного. Spring Boot у поєднанні з інструментами на зразок Docker, Kubernetes, Kafka та Redis відкриває широкі можливості для побудови гнучких і високопродуктивних додатків.

Таким чином, дослідження методики створення додатків на основі Java та Spring Boot є актуальним завданням, яке має як наукову, так і практичну цінність.

Мета статті полягає у формуванні узагальненого підходу, який може бути використаний як основа для навчання студентів, підготовки молодих спеціалістів, а також як методичний орієнтир для практикуючих розробників при створенні реальних проектів.

Виклад основного матеріалу. Розробка додатків на Java та Spring Boot починається з правильного налаштування середовища. Від цього етапу залежить зручність роботи розробника або студента та швидкість виконання завдань. Необхідно встановити Java Development Kit. Для сучасних проектів рекомендовано використовувати Java версії 17 або Java 21, оскільки ці версії є довгостроково підтримуваними і мають низку покращень у сфері продуктивності та безпеки. Важливим інструментом є система керування залежностями та збірки. Найчастіше використовують Maven або Gradle.

Для командної роботи потрібно підключити систему контролю версій Git разом із платформами для співпраці – GitHub [1, с. 177], GitLab або Bitbucket. Це забезпечує збереження історії змін, можливість колективної розробки та впровадження практик CI/CD (безперервної інтеграції та доставки). Окремо варто згадати про налаштування бази даних. Найчастіше у зв’язці зі Spring Boot використовуються PostgreSQL, MySQL або Oracle, які інтегруються через Spring Data JPA. Для тестування зручно застосовувати вбудовані бази. Таким чином,

підготовка середовища є першим і надзвичайно важливим етапом, що створює фундамент для подальшої реалізації додатка.

Архітектура програмного забезпечення визначає, наскільки зручно буде розвивати, масштабувати та підтримувати додаток у майбутньому. Використання Java разом із Spring Boot дає можливість реалізовувати як монолітні, так і мікросервісні архітектури залежно від потреб проєкту.

Монолітна архітектура передбачає, що весь додаток працює як єдиний процес. Такий підхід є зручним для невеликих систем, які не потребують високого рівня масштабованості. Переваги такого підходу полагають у простоті у налаштуванні та розгортанні, менша кількість інфраструктурних складностей та зручність для навчальних або внутрішніх проектів. Проте недоліки моноліту стають відчутними при зростанні системи: складність внесення змін, обмежена масштабованість, залежність усіх модулів один від одного.

Для великих систем більш доцільною є мікросервісна архітектура. У цьому випадку кожен сервіс виконує власну бізнес-логікуі може розвиватися незалежно.

Spring Boot у поєднанні з Spring Cloud надає інструменти для побудови мікросервісної архітектури. Переваги мікросервісів виражаються в можливості незалежного масштабування сервісів, гнучкості у виборі технологій для кожного сервісу, а також підвищена відмовостійкість [8]. Разом із тим, цей підхід потребує більш складної інфраструктури, знань у сфері DevOps, а також ретельного управління транзакціями та даними між сервісами (Рис. 1).

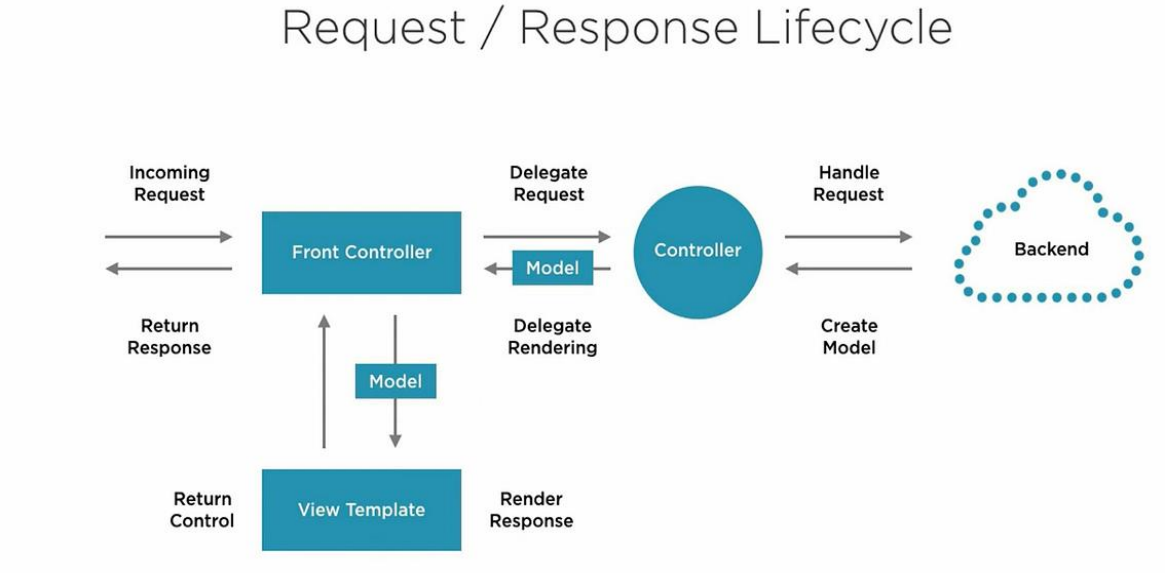

Рисунок 1. Робота контролера

Після вибору архітектури наступним етапом є безпосередня розробка функціональних можливостей додатка. Завдяки Spring Boot цей процес значно спрощується завдяки автоматичній конфігурації та готовим бібліотекам. Основний спосіб взаємодії клієнта та сервера реалізується через REST API. Використовуючи модуль Spring Web, можна швидко створювати контролери, які обробляють HTTP-запити.

Цей підхід дозволяє легко організувати обмін даними між фронтендом і бекендом. Spring Boot інтегрується з Spring Data JPA та Hibernate, що дозволяє працювати з базами даних на високому рівні абстракції. До шару репозиторія доставляється через сервісний шар. Як це реалізовано зображено на Рис. 2.

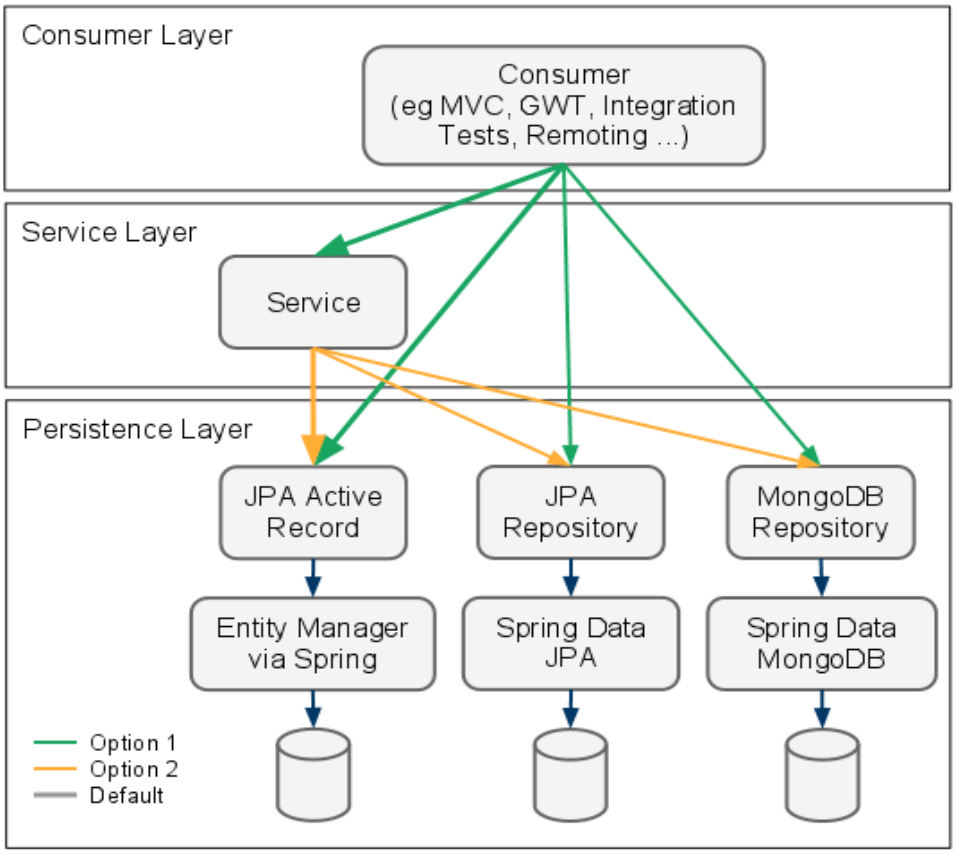

Рисунок 2. Взаємодія сервісного шару з репозиторіями

Створення репозиторію виглядає наступним чином (Рис. 3).

```
@Repository  9 usages  Maxim Shelofast
public interface PostRepository extends JpaRepository<Post, Long> {
    Page<Post> findAllByAuthorId(Long authorId, Pageable pageable);  3 usages  Maxim Shelofast
    long countByAuthorId(Long authorId);  2 usages  Maxim Shelofast
}
```

Рис. 3 Створення репозиторію

Це значно зменшує обсяг коду та спрощує реалізацію CRUD-операцій. Для оптимізації продуктивності часто використовується такий підхід як кешування Рис. 4. Це означає, що спочатку запитуємі дані будуть перевірятися в кеші, і якщо вони там є, то вони повертаються. Якщо в кеші таких даних не опинилося - ми беремо їх з основної бази даних. Найчастіше в якості кеша використовується така NoSQL база даних як Redis [3; 2]. Основна суть полягає в тому, що Redis може повертати дані дуже швидко за константну складність, так як дані зберігаються в оперативній пам’яті, на відміну від SQL баз даних.

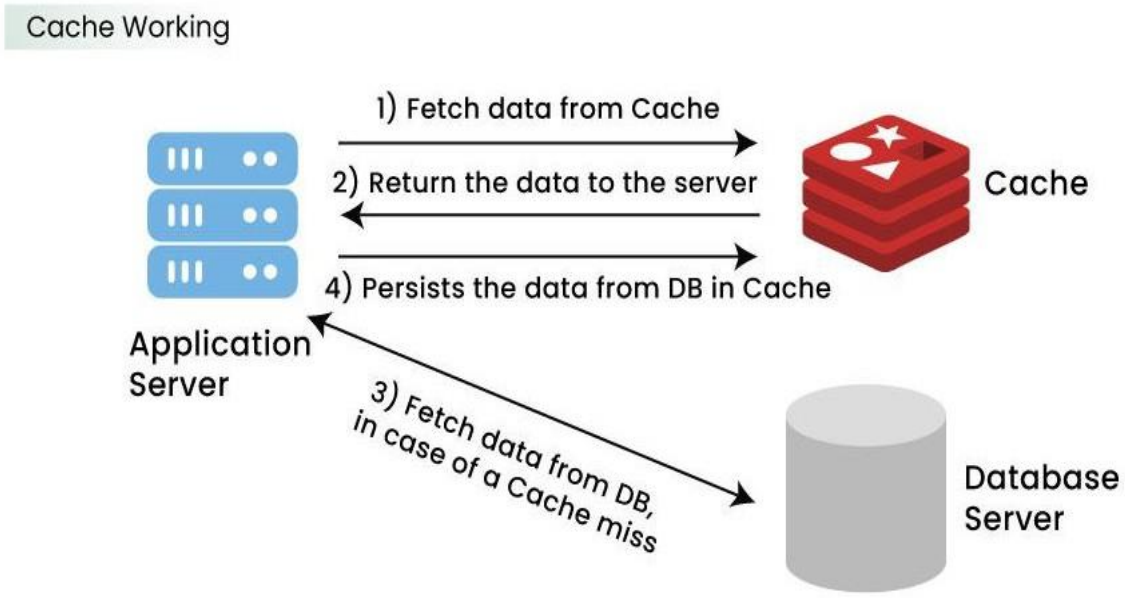

Рисунок. 4 Процес роботи кешування

У Spring Boot це реалізується завдяки анотації @Cacheable. Повторні запити до одного й того ж ресурсу обробляються швидше.

```
@Test  Maxim Shelofast
void testGetPostById_Success() {
    Post post = buildPost();
    PostResponse expected = buildPostResponseDto(CONTENT);

    when(postRepository.findById(POST_ID)).thenReturn(Optional.ofNullable(post));
    when(postMapper.toDto(post)).thenReturn(expected);

    PostResponse result = postService.getPostById(POST_ID);

    assertThat(result).isEqualTo(expected);
}
```

```
@Test  Maxim Shelofast
void testGetPostById_WhenPostNotFound_ShouldThrowException() {
    when(postRepository.findById(POST_ID)).thenReturn( value: Optional.empty());
    assertThrows(NotFoundException.class, () -> postService.getPostById(POST_ID));
}
```

Рисунок 5. Написання unit-тестів

Якість програмного забезпечення значною мірою залежить від рівня тестування. Використання Java у поєднанні зі Spring Boot надає розробникам широкий спектр інструментів для перевірки роботи додатків на різних рівнях.

Основою є JUnit 5, який дозволяє писати тести для окремих класів і методів [4]. У поєднанні з Mockito можна легко створювати mock-об'єкти та перевіряти логіку бізнес-рівня без підключення до бази даних чи зовнішніх сервісів. Основною метою є якомога більше покриття різних сценаріїв бізнес логіки, щоб гарантувати правильну роботу всього додатку (Рис. 5).

```
@Test  Maxim Shelofast
@DataSet(value = "datasets/posts-by-author-id.yml")
void shouldFindAllByAuthorId() {
    Page<Post> result = postRepository.findAllByAuthorId(AUTHOR_ID, buildPageable());
    assertThat(result.getTotalElements()).isEqualTo( expected: 2);
    assertThat(result.getContent()).containsExactlyInAnyOrderElementsOf(buildExpectedPosts());
}

@Test  Maxim Shelofast
@DataSet(value = "datasets/posts-by-author-id.yml")
void shouldCountByAuthorId() {
    long result = postRepository.countByAuthorId(AUTHOR_ID);
    assertThat(result).isEqualTo( expected: 2);
}
```

Рисунок 6. Написання інтеграційних тестів для репозиторію з використанням DBRider

Spring Boot надає спеціальні анотації, які дозволяють запускати тести в умовах, наближених до реальних. Це дає можливість перевірити взаємодію між компонентами: контролерами, сервісами, репозиторіями [5] При тестуванні репозиторія можемо часто зустріти використання бібліотеки DBRider (див Рис. 6). Порівняльна таблиця інтеграційних та юніт тестів наведена у Таблиці 1.

Таблиця 1

Порівняльна таблиця інтеграційних та юніт тестів

Характеристика	Юніт-тести	Інтеграційні тести
Основна мета	Перевірка коректності найменшої ізольованої частини коду (зазвичай метод або клас)	Перевірка взаємодії між декількома компонентами, сервісами або модулями системи
Об’єкт тестування	Один «юніт» (клас, метод)	Два або більше компонентів, що працюють разом
Ізоляція	Висока. Всі зовнішні залежності (наприклад, бази даних, API) зазвичай «мокаються» (імітуються)	Низька або середня. Тести можуть взаємодіяти з реальними залежностями або їхніми «стабами» (заглушками)
Залежності	Мінімальні або відсутні (використовуються моки/стаби)	Включають взаємодію з іншими частинами системи, базою даних, мережевими сервісами тощо
Швидкість виконання	Дуже швидкі. Можна запустити сотні або тисячі за секунди	Повільніші за юніт-тести, оскільки вимагають налаштування середовища та взаємодії компонентів
Складність написання	Зазвичай простіші, оскільки фокусуються на малій частині логіки	Потребують налаштування взаємодії між компонентами та тестового оточення
Частота запуску	Дуже часто. Зазвичай запускаються при кожному збереженні файлу або перед кожним комітом	Рідше. Зазвичай запускаються перед злиттям гілок (merge) або як частина CI/CD пайплайну

Якщо буде зроблено дуже багато інтеграційних тестів і мало юніт тестів, то може виникнути ситуація, коли додаток дуже довго збирається і розгортається, а при цьому бізнес логіка працює не так як задумано. Тому середнє значення покриття тестами буде приблизно таке: 70-80

процентів тестів покрито юніт тестами, та процентів 20 це інтеграційні тести.

Висновки. У статті було розглянуто методику створення додатків на основі Java та Spring Boot, яка охоплює всі основні етапи життєвого циклу програмного забезпечення: від підготовки середовища розробки до тестування. Аналіз показав, що Java залишається однією з найбільш надійних і затребуваних мов програмування для корпоративних систем. Spring Boot значно спрощує процес розробки, забезпечує інтеграцію з сучасними технологіями та підтримує як монолітну, так і мікросервісну архітектуру. А також, комплексне тестування є критично важливим фактором для підтримки якості та стабільності систем.

Таким чином, методика створення додатків на Java та Spring Boot може розглядатися, як універсальна основа для підготовки майбутніх фахівців у сфері розробки програмного забезпечення, та як практичний орієнтир для інженерів, що працюють над реальними проектами.

Література

1. Мосіюк О.О. Переваги використання Git у процесі професійної підготовки майбутніх учителів інформатики. *Науковий вісник Ужгородського університету. Серія: «Педагогіка. Соціальна робота»*. 2017. Т.2, № 41. С. 176–179.
2. Шаров С.В., Петровський В.В. Огляд нереляційних баз даних. *Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення: збірник тез доповідей*. 2015. № 14. С. 16–18.
3. Deinum M., Long J., Mak P., Mayr K. *Spring Boot: Reference Documentation*. URL: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/.
4. JUnit 5 User Guide. URL: https://junit.org/junit5/docs/current/user-guide/.
5. Hightower K., Burns B., Beda, J. *Kubernetes: Up and Running*. O'Reilly Media, 3rd edition, 2022.
6. Kulykovska N.A. et al. Chatbot application to support indoor temperature control. *IOP Conference Series: Earth and Environmental Science*. IOP Publishing, 2024. Т. 1415, № 1. С. 012075.
7. Long J. *Spring Boot Up and Running: Building Cloud Native Java and Kotlin Applications*. O'Reilly Media, 2021.
8. Richardson C. *Microservices Patterns: With examples in Java*. Manning Publications, 2018.
9. Walls C. *Spring in Action*. Manning Publications, 6th edition, 2022.