МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. ПОТЕБНІ
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Порівняльний аналіз використання сучасних фрейворків безпеки при розробці Java-застосунків**

Виконав: студент 2 курсу, групи 8.1213-іпз-2
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
В.В. Козирь
(ініціали та прізвище)
Керівник доцент, к.т.н. Г.П.Коломоєць
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дискус»
Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Т.В. Критська
"_02_" __вересня_____2024_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Козирю Вячеславу Віталійовичу
(прізвище, ім'я, по батькові)

1.Тема роботи Порівняльний аналіз використання сучасних фрейворків безпеки при розробці Java-застосунків

керівник роботи Коломоєць Геннадій Павлович, доцент, к.т.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _28.02.2024р._№_361-е_

2. Строк подання студентом кваліфікаційної роботи 1 грудня 2024 р.
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження фрейворків безпеки при розробці Java-застосунків;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 02.09.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області. Огляд засобів безпеки платформи	02.09-07.09.24	виконано
2	Формулювання основної задачі та визначення цілей дослідження	08.09-10.09.24	виконано
3	Визначення механізмів безпеки, реалізація яких буде порівнюватися	11.09-15.09.24	виконано
4	Розробка загальної архітектури інформаційної системи для порівняння фрейворків	16.09-19.09.24	виконано
5	Реалізація основних функцій системи інформаційної системи для порівняння фрейворків	20.09-02.10.24	виконано
6	Тестування розроблених функцій системи під час взаємодії з різними типами інверторів	03.10-08.10.24	виконано
7	Налаштування взаємодії з базою даних системи	09.10-19.10.24	виконано
8	Реалізація шаблонів веб-застосунку та інтеграція з API	20.10-01.11.24	виконано
9	Тестування загальної роботи системи	02.11-09.11.24	виконано
10	Демонстрація результатів реалізації системи науковому керівнику та узгодження етапів дослідження	10.11-14.11.24	виконано
11	Проведення дослідження роботи з порівння фрейворків	10.11-15.11.24	виконано
12	Опис результатів дослідження	16.11-19.11.24	виконано
13	Оформлення звіту	20.11-27.11.24	виконано

Студент _____ В.В. Козирь
(підпис) (прізвище та ініціали)

Керівник роботи _____ Г.П. Коломоєць
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 98

Рисунків: 30

Таблиць: 10

Джерел: 42

Козирь В.В. Порівняльний аналіз використання сучасних фрейворків безпеки при розробці Java-застосунків: кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник Г.П. Коломоєць. Запоріжжя : ЗНУ, 2024. 98 с.

Метою і завданням дослідження є порівняльний аналіз сучасних фреймворків безпеки, що застосовуються при розробці Java-застосунків, з метою визначення їх переваг, недоліків та оптимальних сфер застосування.

У процесі дослідження було розглянуто особливості різних фрейворків безпеки Java. Також, було розглянуто можливості використання фрейворків та сформовано список їхніх особливостей та недоліків. В результаті була розроблена інформаційна система перегляду коміксів, що складається з елементів для взаємодії з різними фрейворками, API та веб-застосунку. Вона включає в себе основні функціональні можливості аналогічних систем та підтримує можливість використання різних фрейворків на основі різних профілів.

Ключові слова: безпека, фрейворк, комікси, автентифікація, інформаційна система, веб-застосунок.

SUMMARY

Pages: 98

Figures: 30

Tables: 10

References: 42

Kozyr V.V. Comparative Analysis of the Use of Modern Security Frameworks in Java Application Development: Master’s Qualification Thesis, Specialty 121 "Software Engineering" / Scientific Supervisor H.P. Kolomoiets. Zaporizhzhia : ZNU, 2024. 98 p.

The aim and objectives of the study are to conduct a comparative analysis of modern security frameworks used in the development of Java applications to determine their advantages, disadvantages, and optimal areas of application.

The study examined the features of various Java security frameworks. It also explored the potential applications of these frameworks and compiled a list of their features and drawbacks. As a result, an information system for comic book browsing was developed, comprising a service for interacting with frameworks, an API, and a web application. This system includes the core functionalities of similar systems and supports the use of various frameworks based on different profiles.

Keywords: security, framework, comics, authentication, information system, web application.

Зміст

ВСТУП 4

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ЗАСОБІВ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ В JAVA-ЗАСТОСУНКАХ 10

1.1 Огляд необхідності дослідження фреймворків безпеки для Java-застосунків 10

1.2 Опис основ безпеки Java 14

1.3 Опис основних функцій безпеки 21

Висновки до першого розділу 28

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ФРЕЙМВОРКІВ ОРГАНІЗАЦІЇ БЕЗПЕКИ РОБОТИ ЗАСТОСУНКІВ JAVA 30

2.1 Java Authentication and Authorization Service 30

2.2 Spring Security 35

2.3 Apache Shiro 39

Висновки до другого розділу 43

РОЗДІЛ 3 РОЗРОБКА ЗАСТОСУНКУ З РЕАЛІЗАЦІЯМИ МЕХАНІЗМІВ БЕЗПЕКИ ФРЕЙМВОРКАМИ JAAS, SPRING SECURITY ТА APACHE SHIRO 45

3.1 Вимоги до системи для реалізації механізмів безпеки 45

3.2 Побудова діаграм варіантів використання 48

3.3 Архітектура програми 53

3.4 Інтерфейс програми 61

3.4 Реалізація застосунку з використанням JAAS 68

3.5 Реалізація застосунку з використанням Spring Security 72

3.6 Реалізація застосунку з використанням Apache Shiro 76

3.7 Висновки з розділу 3 81

4. ПОРІВНЯННЯ ФРЕЙМВОРКІВ JAAS, SPRING SECURITY ТА APACHE SHIRO НА ОСНОВІ РЕАЛІЗОВАНОГО ЗАСТОСУНКУ 83

4.1 Аналіз продуктивності та ефективності 83

4.2 Зручність використання та впровадження 86

4.3 Рівень безпеки та можливості розширення 87

4.4 Переваги та недоліки кожного фреймворку 89

4.5 Висновки з розділу 4 92

ВИСНОВКИ 93

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 95

ВСТУП

Актуальність теми

З розвитком сучасних технологій забезпечення безпеки стало невід'ємною частиною процесу розробки програмного забезпечення. Особливо це стосується застосунків, створених мовою Java, яка використовується у різних галузях – від корпоративних систем до мобільних додатків. Java вирізняється високим рівнем надійності, що робить її популярним вибором для розробки масштабних проєктів, де безпека є пріоритетом.

Однією з важливих задач, що стоять перед розробниками, є захист застосунків від можливих загроз та атак. Це вимагає використання фреймворків безпеки, які дозволяють ефективно захищати дані, що обробляються програмою, і забезпечувати конфіденційність, цілісність та доступність. Існує безліч сучасних фреймворків, які надають інструменти для забезпечення безпеки Java-застосунків. Серед них можна виокремити такі популярні рішення, як JAAS [2], Spring Security [3], Apache Shiro[4]. Кожен з них має свої унікальні можливості, переваги та обмеження, які впливають на вибір розробника.

Застосування фреймворків безпеки в Java-застосунках дозволяє ефективно вирішувати різноманітні задачі: від автентифікації та авторизації до управління сеансами та захисту від атак типу SQL-ін'єкцій і XSS (Cross-Site Scripting). Ці фреймворки створені з урахуванням сучасних підходів до кібербезпеки, що робить їх надійними засобами захисту даних та інформаційних потоків.

Актуальність дослідження порівняльного аналізу таких фреймворків полягає у необхідності вибору найбільш відповідного рішення залежно від специфіки проекту. Оцінка ефективності, простоти інтеграції, продуктивності та можливостей розширення допоможе розробникам краще зрозуміти, який

інструмент використовувати для забезпечення належного рівня безпеки. Зважаючи на стрімке зростання кількості кіберзагроз і підвищені вимоги до безпеки програмного забезпечення, аналіз сучасних фреймворків є своєчасним і важливим кроком у забезпеченні захищеності Java-застосунків.

Мета і завдання дослідження

Метою дослідження є порівняльний аналіз сучасних фреймворків безпеки, що застосовуються при розробці Java-застосунків, з метою визначення їх переваг, недоліків та оптимальних сфер застосування.

Досягнення цієї мети передбачає вирішення наступних завдань:

1. Аналіз основних загроз і викликів, які стоять перед Java-застосунками в сучасних умовах.
2. Дослідження особливостей архітектури та функціоналу найбільш поширених фреймворків безпеки, таких як Spring Security, Apache Shiro та JAAS.
3. Порівняння ефективності роботи фреймворків при вирішенні завдань автентифікації, авторизації та захисту від типових атак.
4. Оцінка продуктивності та впливу інтеграції фреймворків на швидкодію Java-застосунків.
5. Визначення критеріїв вибору фреймворку залежно від типу застосунку та вимог безпеки.
6. Проведення експериментів для тестування практичної ефективності кожного з фреймворків.
7. Розробка рекомендацій щодо інтеграції та налаштування фреймворків для оптимальної роботи.

Об’єкт дослідження

Об’єктом дослідження є сучасні фреймворки безпеки, що використовуються для розробки Java-застосунків.

Предмет дослідження

Предметом дослідження є методи, засоби та принципи роботи фреймворків безпеки для забезпечення захисту Java-застосунків.

Методи дослідження

Для вирішення поставлених завдань використовуються наступні методи дослідження:

1. Аналіз літератури — огляд наукових статей, технічної документації та практичних керівництв, що стосуються використання фреймворків безпеки.
2. Емпіричні дослідження — тестування фреймворків на реальних прикладах Java-застосунків для оцінки їх ефективності та продуктивності.
3. Порівняльний аналіз — вивчення та порівняння характеристик фреймворків з точки зору функціоналу, можливостей налаштування та використання.
4. Експериментальні методи — проведення практичних експериментів для тестування систем захисту від атак типу SQL-ін'єкцій, XSS та інших загроз.
5. Синтез отриманих даних — узагальнення результатів дослідження для розробки рекомендацій щодо вибору фреймворку.

Наукова новизна дослідження

Наукова новизна дослідження полягає у проведенні порівняльного аналізу сучасних фреймворків безпеки для Java-застосунків з акцентом на їхню продуктивність, ефективність захисту та простоту використання. У роботі вперше комплексно досліджуються можливості інтеграції цих фреймворків у різні типи застосунків, від малих проектів до масштабних корпоративних систем.

Практичне значення роботи

Практичне значення роботи полягає у розробці рекомендацій для розробників щодо вибору та налаштування сучасних фреймворків безпеки. Отримані результати можуть бути використані для підвищення рівня захисту Java-застосунків без збільшення витрат на розробку і супровід, забезпечуючи ефективну інтеграцію засобів безпеки у проект.

Апробація одержаних результатів.

Результати дослідження представлено на науковій конференції "Молода "Молода наука-2024" [36].

Глосарій

Фреймворк безпеки — набір бібліотек та інструментів, призначених для забезпечення захисту програмного забезпечення від кіберзагроз.

Spring Security — потужний фреймворк для захисту Java-застосунків, що надає можливості для автентифікації та авторизації з широкими можливостями налаштування.

Apache Shiro — легкий і гнучкий фреймворк для забезпечення безпеки, що надає базові можливості управління доступом та автентифікації.

OWASP ESAPI — набір бібліотек від Open Web Application Security Project для підвищення рівня безпеки веб-застосунків, що включає засоби захисту від XSS, SQL-ін'єкцій та інших поширених загроз.

SQL-ін'єкція — вид кібератаки, що спрямована на виконання несанкціонованих SQL-запитів до бази даних з метою отримання доступу до даних або їхнього знищення.

XSS (Cross-Site Scripting) — атака на веб-застосунок, що дозволяє вставляти шкідливий скрипт для виконання на стороні користувача.

Кіберзагрози — потенційні дії або події, що можуть завдати шкоди інформаційній безпеці системи.

Автентифікація — процес перевірки ідентичності користувача або системи.

Авторизація — процес надання прав доступу користувачу після проходження автентифікації.

JAAS – Java Authentication and Authorization Service.

API – Application Programming Interface (Інтерфейс програмування застосунків).

LDAP – Lightweight Directory Access Protocol (Легкий протокол доступу до каталогів).

OTP – One-Time Password (Одноразовий пароль).

CSRF – Cross-Site Request Forgery (Міжсайтове підроблення запиту).

JWT – JSON Web Token (Токен веб-автентифікації у форматі JSON).

ORM – Object-Relational Mapping (Об'єктно-реляційне відображення).

JNDI – Java Naming and Directory Interface (Інтерфейс іменування та доступу до каталогів Java).

RDBMS – Relational Database Management System (Система управління реляційними базами даних).

EJB – Enterprise JavaBeans (Компонентна архітектура Java для створення багаторазових бізнес-застосунків).

BCrypt – Алгоритм шифрування паролів, що використовує адаптивний метод хешування.

OAuth2 – Open Authorization 2.0 (Відкрита авторизація 2.0).

HTTPS –Захищений протокол передавання гіпертексту.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ЗАСОБІВ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ В JAVA-ЗАСТОСУНКАХ

1.1 Огляд необхідності дослідження фреймворків безпеки для Java-застосунків

Безпека програмного забезпечення є одним із найважливіших аспектів сучасної розробки, особливо для застосунків, які обробляють конфіденційні дані або виконують критичні функції. Мова програмування Java, завдяки своїй платформонезалежності, масштабованості та розширюваності, стала широко використовуваною для створення веб-додатків, корпоративних систем, мобільних платформ і навіть інтеграційних рішень. Проте із збільшенням кількості атак і підвищенням вимог до захисту даних постала необхідність забезпечення безпеки Java-застосунків на найвищому рівні.

Розробники Java-застосунків стикаються з численними загрозами, які можуть призвести до значних втрат даних та загрожувати безпеці користувачів. До найпоширеніших загроз належать SQL-ін'єкції, атаки типу XSS (Cross-Site Scripting), підміна сеансів (Session Hijacking), витік даних, а також інші види кібератак. Ці загрози мають потенціал завдати значної шкоди як самим застосункам, так і їхнім користувачам, що робить необхідним впровадження комплексних заходів захисту [1].

SQL-ін'єкції — одна з найбільш небезпечних атак, спрямованих на зміну або виконання несанкціонованих SQL-запитів через уразливості в системах обробки введення користувача. Це дозволяє зловмисникам отримати доступ до конфіденційної інформації або змінити бази даних [2]. Атаки XSS спрямовані на вставку шкідливих скриптів у веб-сторінки, які потім виконуються в браузерах користувачів, що може призвести до викрадення сеансів, облікових даних або шкідливої взаємодії з іншими елементами сторінки [3].

Підміна сеансів становить загрозу для збереження ідентифікації користувача в системі, що призводить до можливості захоплення управління сесією та доступу до конфіденційної інформації [4]. Також не менш серйозною загрозою є витік даних, що виникає через недостатні заходи захисту під час передачі або зберігання інформації [5].

Використання сучасних фреймворків безпеки стало необхідною відповіддю на ці виклики. Вони надають інструменти для впровадження захисних механізмів, зокрема автентифікації, авторизації, управління сесіями, шифрування даних та захисту від найпоширеніших атак. Spring Security є одним із найбільш використовуваних фреймворків для забезпечення безпеки Java-застосунків. Він пропонує гнучку архітектуру, яка дозволяє легко інтегрувати його з іншими компонентами екосистеми Spring та налаштовувати автентифікацію, авторизацію і управління ролями [6]. Його перевагою є підтримка сучасних стандартів безпеки, таких як OAuth2 та JWT, що забезпечує безпечну передачу даних і авторизацію у веб-застосунках [7].

Apache Shiro — легкий фреймворк безпеки, відомий своєю простотою впровадження і гнучкістю. Його основними перевагами є інтуїтивно зрозуміле налаштування та підтримка таких функцій, як автентифікація, авторизація, управління сесіями і шифрування даних. У той же час, Shiro може виявитися менш гнучким порівняно зі Spring Security у великих проектах, оскільки не пропонує такої кількості інтеграцій та можливостей [8].

JAAS (Java Authentication and Authorization Service) — це вбудований у платформу Java API, який забезпечує засоби для автентифікації та авторизації користувачів у Java-застосунках. JAAS дозволяє впроваджувати багаторівневий контроль доступу, інтегруючи різні джерела даних для підтвердження облікових записів і ролей.

JAAS підтримує модульну архітектуру автентифікації, яка дозволяє розробникам налаштовувати стратегії автентифікації за допомогою різних реалізацій LoginModule. Для авторизації використовується модель на основі ролей і дозволів, що забезпечує гнучкість у визначенні політик доступу.

Основною перевагою JAAS є його тісна інтеграція з платформою Java і підтримка стандартів безпеки, що спрощує його використання у Java-застосунках. Однак, як і у випадку з іншими базовими інструментами, його функціонал може вимагати значного доопрацювання для складних проектів із специфічними вимогами безпеки.

Аналіз і порівняння цих фреймворків є критично важливими для того, щоб розробники могли вибрати найбільш підходящий інструмент для своїх потреб. Наприклад, для великих корпоративних систем, де важлива гнучкість і можливість інтеграції, оптимальним вибором може бути Spring Security [11]. Для проектів, де потрібна швидка та проста реалізація безпеки, варто звернути увагу на Apache Shiro [12]. У випадках, коли необхідні інтегровані механізми автентифікації та авторизації з гнучкою підтримкою різних джерел облікових даних, JAAS стане найкращим рішенням [14].

Узагальнюючи інформацію розглянемо таблицю 1.1 з загрозами і фреймворками для них.

Таблиця 1.1 - Аналіз основних загроз, викликів та методів захисту, що забезпечуються фреймворками безпеки для Java-застосунків.

Загроза	Spring Security	Apache Shiro	JAAS
SQL-ін'єкції	Захист через інтеграцію з методами перевірки введення та використання ORM	Немає вбудованих інструментів для захисту, потребує додаткової обробки	JAAS не надає прямого захисту; слід використовувати JDBC з підготовленими запитами.
XSS (Cross-Site Scripting)	Захист через налаштування фільтрації та перевірки вхідних даних	Обмежений захист, потребує додаткових бібліотек	Захист відсутній; необхідно застосовувати додаткові бібліотеки для санітаризації вводу.
Підміна сеансів	Підтримка захисту від атак на сесію, контроль через Spring Session	Захист сесій через управління сесіями	Підтримка механізмів автентифікації для верифікації сесій.
Витік даних	Використання шифрування та контролю доступу	Підтримка шифрування даних, але менше налаштувань	Використання JAAS у поєднанні зі шифруванням для обмеження доступу до даних.
CSRF (Cross-Site Request Forgery)	Вбудовані інструменти для захисту від CSRF	Потребує додаткової реалізації	Немає вбудованого захисту; слід застосовувати токени для запобігання CSRF.
Атаки типу Brute Force	Можливість впровадження обмежень на кількість спроб входу	Підтримує базові механізми захисту від атак	Підтримка гнучких політик автентифікації, що можуть обмежувати кількість спроб входу.

Таблиця 1.1 показує, які методи захисту надають популярні фреймворки безпеки при вирішенні основних загроз для Java-застосунків. Вона допомагає зрозуміти, які аспекти безпеки можна забезпечити без додаткових бібліотек, а де можуть знадобитися додаткові засоби чи конфігурації. Проведення детального аналізу ефективності, продуктивності та зручності використання фреймворків дозволяє уникнути помилок при інтеграції системи безпеки, підвищити надійність Java-застосунків та знизити ризики втрат даних [14]. Такий підхід відповідає сучасним вимогам до розробки безпечного

програмного забезпечення, враховуючи зростаючу складність та витонченість кіберзагроз [15].

Узагальнюючи розглянуту інформацію, проведення дослідження фреймворків безпеки для Java-застосунків є актуальним і затребуваним завданням, яке сприяє підвищенню захисту програмного забезпечення від потенційних загроз. Це дослідження дозволяє розробникам обрати найбільш підходяще рішення для свого проекту, забезпечуючи оптимальний баланс між безпекою, продуктивністю та зручністю інтеграції.

1.2 Опис основ безпеки Java

Безпека Java-застосунків ґрунтується на поєднанні найкращих практик програмування, використанні вбудованих механізмів безпеки та впровадженні зовнішніх фреймворків для захисту від сучасних загроз. Основи безпеки Java включають кілька ключових принципів, які забезпечують захищеність програмного забезпечення, його стабільну роботу та захист даних користувачів.

Автентифікація є важливим процесом перевірки ідентифікації користувача перед наданням доступу до системи. У Java-застосунках автентифікація забезпечує підтвердження особи користувача шляхом перевірки введених даних, таких як ім'я користувача і пароль, або за допомогою більш сучасних методів, таких як одноразові паролі (OTP), біометрія та двофакторна автентифікація (2FA). Java Enterprise Edition (JEE) пропонує вбудовані модулі для управління автентифікацією через стандартні засоби безпеки, як-от HttpServletRequest.login() та JASPIC (Java Authentication SPI for Containers) [1]. Типи автентифікації розглянемо на рисунку 1.1.

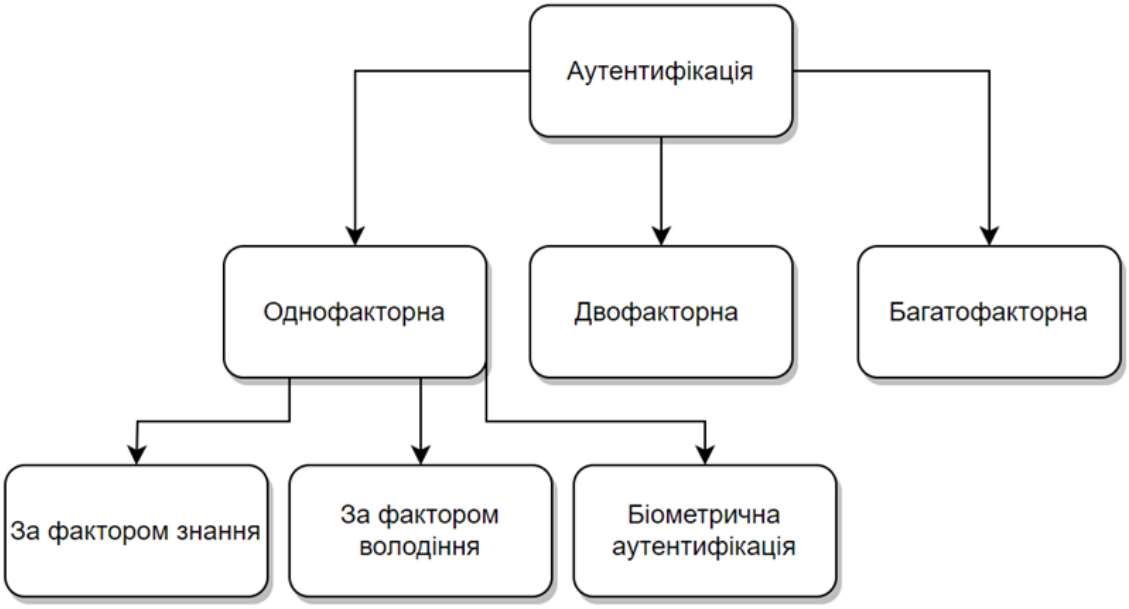


Рисунок 1.1 – Типи автентифікації користувачів

Spring Security є одним із найпоширеніших фреймворків для впровадження автентифікації, що підтримує різні механізми, такі як OAuth2, JWT (JSON Web Token), LDAP (Lightweight Directory Access Protocol) та інші [2]. Цей фреймворк забезпечує високу гнучкість і дозволяє створювати надійні системи захисту з мінімальними зусиллями.

Авторизація визначає права доступу користувача до ресурсів системи після його автентифікації. Основні підходи до авторизації включають використання ролевої моделі доступу, яка дозволяє призначати права на основі ролей користувача, або списків контролю доступу (ACL), які визначають рівні доступу для кожного користувача окремо. Apache Shiro пропонує просту і гнучку модель авторизації, яка дозволяє легко налаштовувати права доступу і інтегруватися з різними джерелами даних [3]. Порівняльний аналіз механізмів автентифікації та аторизації у фреймворках розглянемо далі у таблиці 1.2.

Таблиця 1.2 - Порівняльна таблиця механізмів автентифікації та авторизації у фреймворках

Фреймворк / Підхід	Типи автентифікації	Типи авторизації	Додаткові можливості
Java EE (JASPIC)	Ім'я користувача та пароль, інтеграція з сервлетами	Рольова модель доступу	Стандартні засоби для веб-контейнерів
Spring Security	OAuth2, JWT, LDAP, базова автентифікація, 2FA	Рольова модель доступу, ACL	Підтримка сучасних протоколів, кастомізація
Apache Shiro	Ім'я користувача та пароль, токени	Рольова модель, ACL	Гнучка конфігурація, простота налаштувань
OWASP ESAPI	Підтримка автентифікації через кастомні механізми	ACL, налаштовувана авторизація	Інструменти для додаткового захисту від вразливостей
Keycloak	Одиночний вхід (SSO), OAuth2, OpenID Connect	Рольова модель, багаторівнева авторизація	Інтеграція з різними системами і додатками

Захист даних є критично важливим аспектом забезпечення безпеки Java-застосунків, оскільки він спрямований на запобігання несанкціонованому доступу до інформації під час її передачі мережею та при зберіганні. У цьому контексті використовуються різні методи шифрування, що дозволяють зберегти конфіденційність та цілісність даних.

1. Вбудовані бібліотеки для шифрування в Java. Java пропонує потужний набір інструментів для реалізації криптографічних операцій, серед яких основним є Java Cryptography Extension (JCE). Ця бібліотека надає можливість використовувати:

- Симетричне шифрування, де один ключ використовується для шифрування та розшифрування даних. Прикладом є алгоритм AES (Advanced Encryption Standard), який забезпечує високий рівень захисту при швидкому виконанні операцій [3].

- Асиметричне шифрування, що використовує пару ключів — відкритий для шифрування та закритий для розшифрування. RSA (Rivest–Shamir–Adleman) є одним із найпоширеніших асиметричних алгоритмів, що використовується для передачі даних з високим рівнем захисту [4]. Узагальнюючи методи шифрування можемо побачити на рис. 1.2.

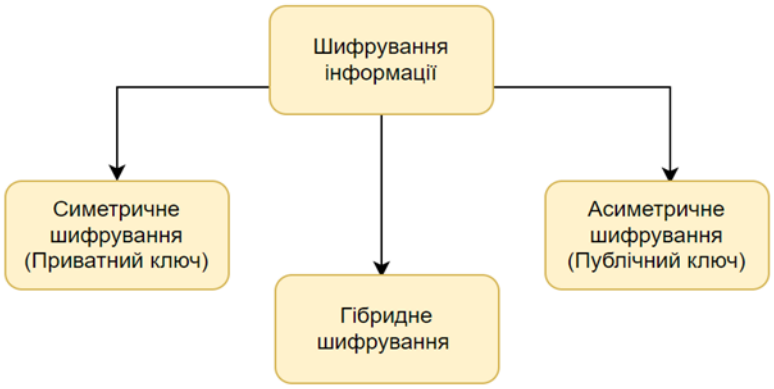


Рисунок 1.2 – Методи шифрування інформації

JCE також підтримує хешування даних, що дозволяє перевіряти їх цілісність. Алгоритми, такі як SHA-256 (Secure Hash Algorithm), забезпечують створення хешів, що допомагає виявляти будь-які зміни у даних під час передачі [5].

2. Використання зовнішніх бібліотек для шифрування. На додаток до вбудованих можливостей, Java підтримує інтеграцію зі сторонніми бібліотеками для підвищення рівня захисту даних:

- OWASP ESAPI (Enterprise Security API) — бібліотека, розроблена спеціально для підвищення рівня безпеки веб-застосунків. Вона пропонує інструменти для роботи з шифруванням, управлінням паролями, захистом від ін'єкцій та інших атак. Зокрема, ESAPI підтримує зручні методи для

симетричного та асиметричного шифрування, що спрощує процес впровадження захищеного зберігання даних [6].

- Bouncy Castle — ще одна популярна криптографічна бібліотека, що пропонує розширену підтримку алгоритмів шифрування та хешування. Вона додає підтримку нових криптографічних протоколів і можливість впровадження користувацьких налаштувань для специфічних завдань [7].

3. Методи захисту даних під час передачі. Для захисту даних, що передаються мережею, використовуються протоколи шифрування, такі як TLS (Transport Layer Security). Він забезпечує захищене з'єднання між клієнтом і сервером, захищаючи передані дані від перехоплення або модифікації. У Java можна налаштувати TLS-з'єднання за допомогою Java Secure Socket Extension (JSSE) [8].

4. Управління ключами та безпека паролів. Ефективне шифрування також залежить від безпечного управління ключами. Java KeyStore (JKS) є стандартним механізмом для зберігання криптографічних ключів, сертифікатів та інших секретів. Зовнішні бібліотеки, такі як ESAPI, пропонують інструменти для зберігання ключів та управління ними, забезпечуючи їх безпеку за допомогою шифрування [9].

Для управління паролями можна використовувати багаторазове хешування із застосуванням випадкових значень (сіль). Цей метод значно підвищує безпеку паролів, запобігаючи їхньому відновленню за допомогою попередньо обчислених таблиць [10].

5. Практичні рекомендації з реалізації захисту даних

- використовуйте алгоритми шифрування з надійним рівнем захисту, як-от AES-256.

- Налаштуйте безпечні протоколи передачі даних (TLS) для всіх мережевих з'єднань.

- Забезпечте надійне управління ключами за допомогою JKS або спеціалізованих рішень для управління секретами [11].

- Використовуйте багаторазове хешування для захисту паролів, додаючи випадкові значення для кожного пароля.

Захист даних у Java-застосунках вимагає поєднання різних методів та інструментів, щоб забезпечити їхню безпеку як під час зберігання, так і під час передачі. Дотримання сучасних практик та використання надійних бібліотек є ключем до успішного захисту конфіденційної інформації.

Безпечне управління сесіями є основою для захисту веб-додатків від атак, спрямованих на підміну сеансів (session hijacking) та зловживання сесіями. Java-застосунки можуть використовувати функції Java Servlet API для управління сесіями користувачів. Ці функції забезпечують підтримку сесій через механізми, такі як зберігання ідентифікатора сесії в cookie-файлах та прив’язка сесії до конкретного користувача [5].

Spring Security пропонує більш гнучкі можливості управління сесіями, включаючи:

- обмеження кількості одночасних сеансів для одного користувача, що знижує ризик зловживань при вході з різних пристроїв.

- Політики завершення сесій після певного періоду неактивності, що зменшує ризик несанкціонованого доступу до системи.

- Захист від фіксації сесій (session fixation), що унеможливлює зловмиснику використання дійсної сесії іншого користувача після його автентифікації [5].

Такі функції допомагають уникати потенційних атак на сесії та забезпечують більш високий рівень безпеки застосунків.

Сучасні веб-додатки піддаються різним видам атак, серед яких SQL-ін'єкції, XSS (Cross-Site Scripting) та CSRF (Cross-Site Request Forgery).

Для захисту від SQL-ін'єкцій рекомендується використовувати підготовлені запити (prepared statements), що унеможливлюють вставку шкідливого коду у запити до бази даних. ORM-системи (об'єктно-реляційні мапери), такі як Hibernate та JPA, автоматично створюють запити, що захищені від ін'єкцій, оскільки вони параметризовані [6].

XSS-атаки можуть призвести до виконання шкідливого JavaScript-коду у браузері користувача. Для захисту рекомендується проводити перевірку та очищення вхідних даних, а також використовувати вбудовані фільтри фреймворків.

CSRF-атаки спрямовані на виконання несанкціонованих дій від імені користувача без його відома. Для захисту від цих атак використовується впровадження CSRF-токенів, які додаються до кожного запиту та перевіряються сервером. Spring Security має вбудовану підтримку генерації та перевірки CSRF-токенів, що значно спрощує їхнє використання [8].

Правильне управління доступом до ресурсів є важливою складовою безпеки Java-застосунків. Spring Security надає широкий спектр можливостей для визначення політик доступу, зокрема:

- конфігурація доступу на основі анотацій (@PreAuthorize, @Secured), що дозволяє обмежити доступ до методів або контролерів.
- XML-конфігурація, що забезпечує гнучке налаштування правил доступу без зміни вихідного коду.
- Рольова модель доступу, де права доступу визначаються на основі ролей, призначених користувачу. Це підвищує ефективність управління та дозволяє легко змінювати рівні доступу для груп користувачів [9].

Apache Shiro також забезпечує налаштування управління доступом за допомогою ролей і дозволів, що дозволяє створювати прості або складні системи контролю доступу [10].

Захищені конфігурації є важливими для зниження ризиків, пов'язаних із можливими вразливостями. Основні практики включають:

- використання HTTPS для забезпечення захищеного з'єднання та шифрування переданих даних. У Java-застосунках це досягається через налаштування сервера та використання TLS/SSL-сертифікатів [11].
- Управління винятками для уникнення виводу повідомлень про помилки, що можуть розкривати внутрішню структуру застосунку. Це знижує

ризик витоку інформації, яку зловмисники можуть використовувати для подальших атак.

- Аналіз коду на предмет вразливостей за допомогою інструментів, таких як SonarQube, що дозволяє розробникам виявляти слабкі місця в коді та виправляти їх до розгортання застосунку [12].

Дотримання міжнародних стандартів безпеки дозволяє забезпечити високу якість та захищеність програмного забезпечення. Основними стандартами є:

- OWASP Top Ten — це перелік найпоширеніших загроз безпеці веб-додатків, який допомагає розробникам уникати поширених помилок у безпеці. Впровадження рекомендацій OWASP допомагає запобігти SQL-ін'єкціям, XSS, неправильному управлінню сесіями та іншим загрозам [13].
- ISO/IEC 27001 — міжнародний стандарт управління інформаційною безпекою, який описує найкращі практики для захисту даних та управління ризиками. Впровадження стандартів ISO забезпечує комплексний підхід до управління безпекою в організації [14].

Розробка Java-застосунків повинна враховувати ці стандарти для забезпечення відповідності сучасним вимогам безпеки, що дозволяє уникати критичних вразливостей і забезпечувати захист даних користувачів на високому рівні.

1.3 Опис основних функцій безпеки

Сучасні Java-застосунки також використовують складні методи автентифікації, такі як OAuth2 та JWT (JSON Web Token). Ці протоколи забезпечують безпечний вхід через сторонні сервіси та дозволяють створювати системи без збереження стану сесій. Використання OAuth2 дозволяє користувачам авторизуватися через відомі сервіси, як-от Google чи Facebook, що підвищує зручність і безпеку. JWT, у свою чергу, дозволяє створювати токени, які містять зашифровану інформацію про користувача та

використовуються для підтвердження автентифікації без необхідності зберігати сесію на сервері [1].

Авторизація – це процес, який визначає рівень доступу користувача до ресурсів системи після успішної автентифікації. У Java-застосунках основним підходом до управління доступом є рольова модель. Ця модель передбачає, що кожному користувачу присвоюється одна або кілька ролей, які визначають його права доступу до різних ресурсів. Наприклад, роль "адміністратор" може мати повний доступ до системи, тоді як роль "користувач" обмежується певними функціями. Такий підхід забезпечує гнучке та зручне управління доступом, дозволяючи легко додавати та змінювати права [2].

Іншим підходом є використання списків контролю доступу (ACL). ACL надають можливість детальніше налаштовувати права доступу для окремих користувачів або груп. На відміну від рольової моделі, ACL визначають доступ до кожного ресурсу на основі списку дозволів, що описують, хто і які дії може виконувати з певним ресурсом. Це підвищує рівень контролю над доступом та забезпечує точніше налаштування політик безпеки, особливо у великих системах з різнорідними ресурсами.

Управління сесіями є важливим аспектом безпеки Java-застосунків, оскільки забезпечує захист користувацьких сесій від різних атак, таких як підміна сеансів (session hijacking) та несанкціонований доступ до інформації. Управління сесіями включає в себе низку функцій, які дозволяють забезпечити надійний захист користувачів і запобігти потенційним загрозам.

Однією з ключових функцій управління сесіями є захист від фіксації сесій. Підміна сесії може статися, коли зловмисник перехоплює активний ідентифікатор сесії (session ID) і використовує його для несанкціонованого доступу до системи від імені законного користувача. Для запобігання цій загрозі Java-застосунки можуть застосовувати механізм зміни ідентифікатора сесії після успішної автентифікації користувача. Це означає, що після входу в систему session ID автоматично оновлюється, знижуючи ймовірність успішної атаки з використанням старого ідентифікатора [3].

Обмеження кількості активних сесій для одного користувача є ще однією важливою функцією. Це дозволяє запобігти одночасному входу одного облікового запису на кількох пристроях або з різних місць, що може бути ознакою зловживання або атаки. Наприклад, якщо користувач вже увійшов у систему, при спробі авторизуватися з іншого пристрою стара сесія може бути завершена або заблокована, залежно від налаштувань безпеки. Такий підхід підвищує рівень захисту, особливо для систем, де необхідний суворий контроль за одночасним доступом.

Автоматичне завершення сесії після певного періоду неактивності є важливим методом запобігання несанкціонованому доступу. Якщо користувач не проявляє активність протягом заданого періоду, сесія автоматично завершується, що мінімізує ризик використання відкритих сесій сторонніми особами. Це особливо актуально для додатків, що використовуються в загальнодоступних місцях або на спільних пристроях. Налаштування тайм-ауту для сесій допомагає запобігти ситуаціям, коли сесії залишаються відкритими після того, як користувач завершив роботу.

У Java-застосунках управління сесіями можна здійснювати за допомогою Java Servlet API, який надає базові функції для роботи з сесіями. Також розширені можливості управління сесіями підтримуються Spring Security, який забезпечує гнучке налаштування політик безпеки, включаючи обмеження одночасних сесій, налаштування тайм-аутів, а також захист від фіксації сесій. Цей фреймворк надає розробникам можливість швидко інтегрувати комплексний захист сесій без необхідності написання додаткового коду.

Шифрування з використанням симетричного ключа - це процес, у якому один і той самий секретний ключ використовується як для шифрування, так і для дешифрування даних. Цей ключ повинен бути відомий як відправнику, так і одержувачу повідомлення. Однією з переваг симетричного шифрування є його простота та швидкість операцій. Під час шифрування та дешифрування використовується лише один ключ, що спрощує реалізацію алгоритму та

зменшує час обробки даних. Проте, є певні недоліки у використанні симетричних ключів. Найбільш значущим є потреба безпечно передавати цей ключ всім сторонам, які мають 9 доступ до зашифрованих даних. Це може бути складно, оскільки неправильна передача чи викрадення ключа може привести до доступу до конфіденційної інформації. Також, зміна ключа вимагає зміни ключа на всіх пристроях, що може бути не зручно у великих системах або у випадку, коли ключ використовується для тривалих періодів.

Шифрування з асиметричним ключем використовує пару ключів: відкритий ключ та закритий ключ. Відкритий ключ може бути розповсюджений публічно, тоді як закритий ключ зберігається в таємниці. Дані шифруються за допомогою відкритого ключа, а дешифруються за допомогою закритого ключа. Прикладами асиметричних алгоритмів є RSA, Diffie-Hellman та криптографія на еліптичних кривих.

Однією з основних переваг асиметричного шифрування є можливість безпечного обміну ключами без необхідності попереднього встановлення спільного секрету. Це робить його ідеальним для захищеної комунікації відкритими мережами, де важко забезпечити безпечний обмін секретним ключем між сторонами.

Проте асиметричні алгоритми зазвичай мають недолік у порівнянні з симетричними алгоритмами — вони можуть бути повільнішими через складніші обчислення, що потрібні для операцій шифрування та дешифрування. Це може бути проблемою у великих мережах або у випадку потреби в обробці великої кількості даних.

Хешування — це процес перетворення даних будь-якого розміру у фіксований хеш-код заданої довжини. Хеш-функції, такі як SHA-256 та MD5, використовуються для цього процесу. Вони не призначені для шифрування даних, але можуть бути використані для перевірки цілісності даних та автентифікації.

У процесі хешування, будь-який вхідний набір даних, будь то текстовий рядок, файл або документ, перетворюється у випадково згенеровану

послідовність байтів, що відома як хеш-код. Цей хеш-код є унікальним для кожного вхідного набору даних. Навіть найменша зміна в вхідних даних призводить до зміни хеш-коду.

Використання хеш-функцій для перевірки цілісності даних полягає у тому, щоб порівняти обчислений хеш-код даних із збереженим значенням хеш-коду. Якщо хеш-коди співпадають, це означає, що дані залишилися незмінними, що свідчить про їхню цілісність. Хеш-функції також використовуються у криптографії для генерації цифрових підписів та ідентифікації даних.

Цифровий підпис — це криптографічний засіб автентифікації особи або повідомлення, який гарантує, що повідомлення не було підроблене або змінено під час трансляції. Для створення цифрового підпису використовується комбінація відкритого ключа та повідомлення, яке потрібно підписати. Підпис може бути перевірений будь-хто за допомогою відкритого ключа, що робить його доступним для відкритої перевірки. Цифрові підписи забезпечують надійний механізм перевірки автентичності та цілісності даних в цифровому середовищі.

Криптографія на основі ідентичності, з іншого боку, забезпечує безпечний спосіб автентифікації користувачів та пристроїв без необхідності введення паролів або іншої секретної інформації. Вона ґрунтується на використанні цифрових підписів та відкритих ключів, що дозволяє користувачам доводити свою особистість шляхом надання цифрового підпису, який може бути перевірений за допомогою відкритого ключа. Це забезпечує високий рівень безпеки та впевненість у правильності автентифікації без необхідності введення секретних даних.

Захист Java-застосунків від поширених атак, таких як SQL-ін'єкції, XSS (Cross-Site Scripting) та CSRF (Cross-Site Request Forgery), є важливим завданням для збереження цілісності системи та конфіденційності даних.

SQL-ін'єкції виникають, коли шкідливий код вводиться в SQL-запити через вхідні дані користувачів. Запобігання цій загрозі здійснюється за

допомогою підготовлених запитів, які ізолюють дані від самого SQL-коду, забезпечуючи безпечне виконання запитів. Об'єктно-реляційні мапери (ORM), такі як Hibernate та JPA, автоматично формують запити, що захищені від SQL-ін'єкцій, і тим самим знижують ризик атак [5]. XSS-атаки спрямовані на виконання шкідливих скриптів у браузері користувача, що може призвести до викрадення сесій або інших небажаних наслідків. Для захисту у контексті JAAS використовується очищення вхідних даних та кодування вихідних даних за допомогою бібліотек, таких як Apache Commons Text, що забезпечують запобігання виконанню шкідливих скриптів.

CSRF-атаки відбуваються, коли шкідливий запит виконується від імені авторизованого користувача без його відома. Захист від таких атак реалізується за допомогою CSRF-токенів, що додаються до кожного запиту і підтверджують його дійсність. Spring Security забезпечує вбудовану підтримку цього методу, що спрощує реалізацію CSRF-захисту [5]. Типи захисту з інструментами розглянемо у таблиці 1.4.

Таблиця 1.4 – Типи загроз, методи захисту й відповідні Java-інструменти

Тип загрози	Методи захисту	Інструменти
SQL-ін'єкції	Підготовлені запити, використання ORM	Hibernate, JPA
XSS	Очищення вхідних даних, використання безпечних бібліотек	JAAS з бібліотеками (наприклад Apache Commons Text)
CSRF	Впровадження CSRF-токенів	Spring Security

Для забезпечення захищеного з'єднання між клієнтом і сервером використовується HTTPS/TLS, що шифрує передані дані і запобігає їх перехопленню. Це досягається шляхом налаштування серверних сертифікатів SSL, які забезпечують автентифікацію сервера та шифрування даних.

Налаштування TLS у Java-застосунках виконується за допомогою Java Secure Socket Extension (JSSE). Сертифікати можна отримати через авторитетні сертифікаційні центри або використовувати власні сертифікати

для внутрішніх проектів [6]. Розглянемо наступною таблицю 1.5 яка описує компоненти для забезпечення безпечного з'єднання

Таблиця 1.5 - Компоненти для забезпечення безпечного з'єднання

Компонент	Функція	Інструменти
HTTPS/TLS	Шифрування даних, захист від перехоплення	JSSE
Сертифікати SSL	Автентифікація сервера	Сертифікаційні центри, OpenSSL

Моніторинг діяльності користувачів і системи дозволяє своєчасно виявляти та реагувати на потенційні загрози. Журналювання дій користувачів допомагає ідентифікувати підозрілі активності, що може свідчити про спробу атаки. Для збирання логів і аналітики часто використовуються ELK Stack (ElasticSearch, Logstash, Kibana)[37] або Splunk[38], які забезпечують централізоване зберігання і аналіз даних. Розглянемо таблицю 1.6. з функціями логування та інструменти моніторингу.

Таблиця 1.6 - Функції логування та інструменти моніторингу

Функція логування	Опис	Інструменти
Журналювання дій	Виявлення підозрілих активностей	ELK Stack, Splunk
Централізований моніторинг	Збір та аналіз логів	ELK Stack, Splunk

Правильне управління помилками дозволяє уникати витоку інформації про внутрішню структуру системи. Логування помилок має відбуватися без розкриття деталей користувачу. Для відображення загальних повідомлень про помилки рекомендується використовувати спеціально налаштовані сторінки, що не розкривають внутрішню логіку роботи застосунку [8]. Компоненти управління помилками та винятками розглянемо детальніше у таблиці 1.7.

Таблиця 1.7 – Управління помилками та винятками

Компонент	Опис	Рекомендації
Логування помилок	Збереження деталей для аналізу	Не відображати внутрішні дані користувачу
Спеціальні сторінки	Загальні повідомлення для користувачів	Використовувати кастомні сторінки помилок

Дотримання міжнародних стандартів безпеки є ключовим аспектом забезпечення захищеності Java-застосунків. OWASP Top Ten [39] пропонує список найпоширеніших загроз для веб-додатків і рекомендації щодо їх запобігання, тоді як ISO/IEC 27001 [40] описує найкращі практики управління інформаційною безпекою.

Використання та впровадження цих функцій та практик дозволяє створити надійне програмне забезпечення, що відповідає сучасним стандартам захисту та забезпечує безпеку даних користувачів і систем.

Висновки до першого розділу

Проведене дослідження засобів забезпечення безпеки в Java-застосунках демонструє важливість інтеграції сучасних методів і підходів для захисту програмного забезпечення. Аналіз необхідності дослідження фреймворків безпеки показав, що через зростаючі загрози і складність атак, розробники повинні бути обізнаними щодо ефективних засобів безпеки, які можуть бути використані для підвищення захищеності систем.

Опис основ безпеки Java підкреслив важливість комплексного підходу до захисту, що включає автентифікацію, авторизацію, управління сесіями, захист даних, а також використання надійних протоколів для безпечної передачі інформації. Розуміння цих основ допомагає створювати застосунки, які відповідають сучасним вимогам і стандартам безпеки.

Опис основних функцій безпеки акцентував увагу на реалізації ключових компонентів захисту, таких як захист від веб-загроз, безпечне

з'єднання, логування та моніторинг, управління помилками та винятками, а також дотримання міжнародних стандартів безпеки. Впровадження цих функцій дозволяє знизити ризики потенційних атак, забезпечити захист конфіденційних даних і підвищити рівень довіри користувачів до програмного забезпечення.

Загалом, комплексний підхід до забезпечення безпеки в Java-застосунках є необхідним для створення надійного та захищеного програмного забезпечення. Дотримання найкращих практик, використання сучасних фреймворків та слідування міжнародним стандартам допомагає знизити вразливості систем та забезпечити їхню стійкість до різних видів загроз.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ФРЕЙМВОРКІВ ОРГАНІЗАЦІЇ БЕЗПЕКИ РОБОТИ ЗАСТОСУНКІВ JAVA

2.1 Java Authentication and Authorization Service

Java Authentication and Authorization Service (JAAS) [2] є потужним фреймворком для забезпечення автентифікації та авторизації у Java-додатках, який був інтегрований у JDK 1.4 як частина Java 2 Platform, Standard Edition (J2SE). Основною метою розробки JAAS було забезпечення більш високого рівня безпеки шляхом відокремлення логіки автентифікації та авторизації від основного коду програми. Це відокремлення дозволяє створювати безпечніші та більш гнучкі програми, оскільки процеси автентифікації та управління доступом можуть змінюватися без потреби змінювати саму бізнес-логіку.

JAAS побудований на модульному підході, що надає можливість розробникам використовувати різні модулі автентифікації. Модулі можуть бути легко додані або замінені, що значно підвищує гнучкість програми. Кожен модуль автентифікації реалізує інтерфейс LoginModule, який забезпечує базові методи для процесів входу (login()), перевірки (commit()), відкату (abort()) та виходу з системи (logout()). Цей інтерфейс дозволяє створювати модулі, які можуть виконувати широкий спектр завдань – від базової автентифікації за допомогою логіна та пароля до більш складних методів, таких як інтеграція з LDAP[41] або Kerberos [42].

Конфігураційні файли відіграють важливу роль у JAAS, забезпечуючи можливість налаштування процесів автентифікації та авторизації без зміни коду програми. Конфігураційний файл містить інформацію про модулі автентифікації, що використовуються, та їхні параметри, дозволяючи змінювати механізми автентифікації в залежності від потреб безпеки. Наприклад, можна легко налаштувати застосунок на використання сертифікатної автентифікації для підвищеного рівня безпеки або переключитися на перевірку за допомогою зовнішніх сервісів через LDAP.

Процес автентифікації в JAAS починається з ініціалізації об'єкта LoginContext. Об'єкт LoginContext приймає ім'я конфігурації, яке визначає, які модулі автентифікації мають бути викликані. Це забезпечує можливість використання різних схем автентифікації для різних частин програми. Далі метод login() викликає процес автентифікації, у якому LoginContext звертається до зазначених модулів автентифікації. На рис. 2.1 представлена структура основних елементів JAAS, де видно, як LoginContext взаємодіє з модулями автентифікації та конфігураційними файлами.

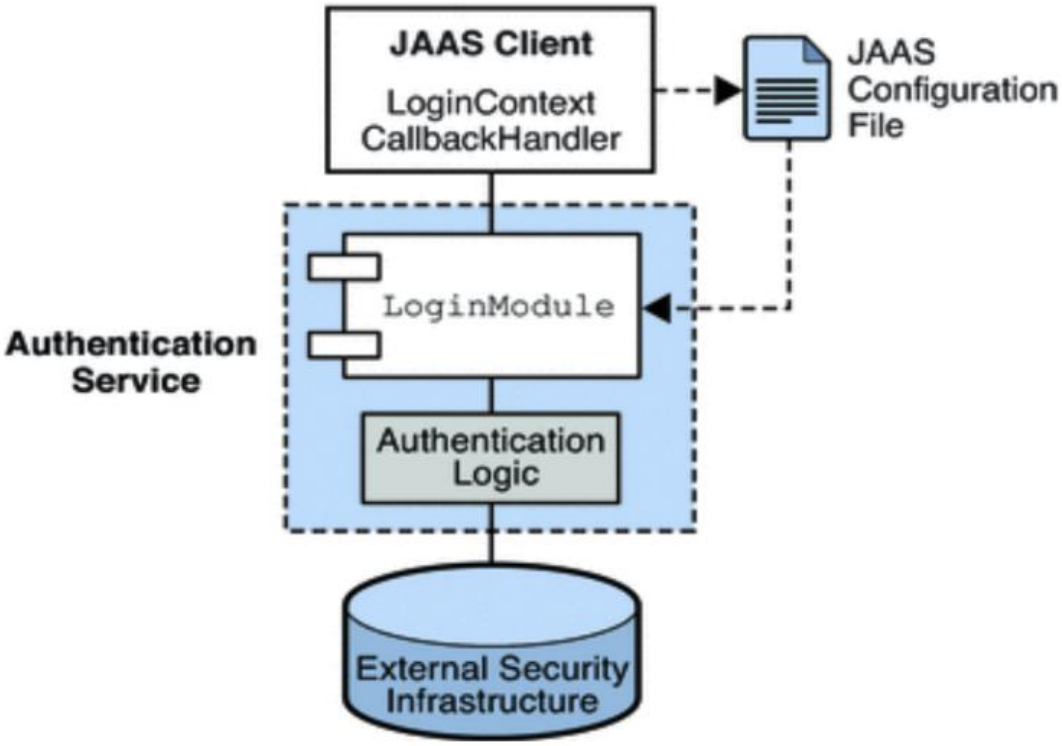


Рисунок 2.1 – JAAS елементи

Об'єкт Subject створюється після успішної автентифікації і представляє автентифікованого користувача. Цей об'єкт може містити декілька об'єктів Principal, що описують різні аспекти ідентичності користувача, такі як ім'я користувача або його роль у системі. Авторизація здійснюється через політики контролю доступу, що визначають, які дії дозволені певним користувачам або групам користувачів. Перевірка прав доступу виконується за допомогою класу

AccessController, який забезпечує перевірку, чи має поточний Subject необхідні привілеї для виконання певних дій.

На рисунку 2.2 продемонстровано приклад використання JAAS у системі черги повідомлень.

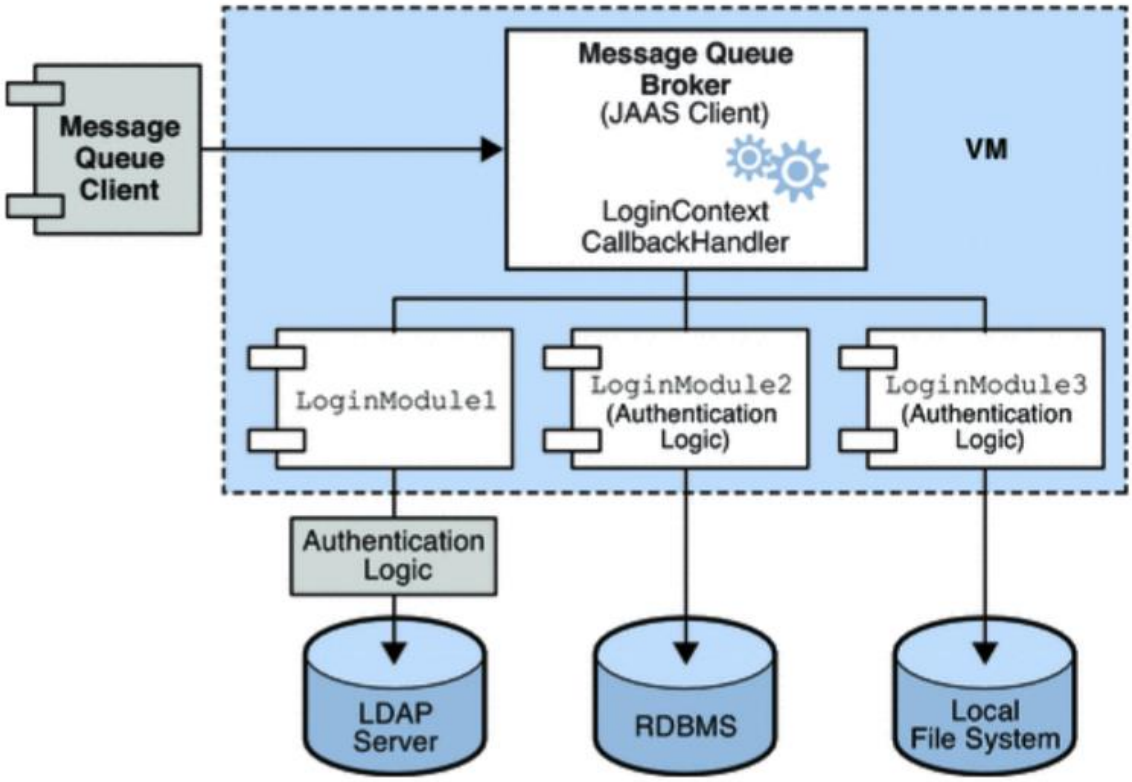


Рисунок 2.2 - Як черга повідомлень використовує JAAS

Message Queue Broker використовує LoginContext для автентифікації клієнтів, а також може звертатися до різних модулів автентифікації, таких як інтеграція з LDAP-серверами, реляційними базами даних (RDBMS) або локальними файловими системами. Такий підхід забезпечує гнучкість і масштабованість системи, дозволяючи підтримувати різні джерела для автентифікації та авторизації.

Інтеграція з іншими компонентами Java Platform, такими як Java Naming and Directory Interface (JNDI) та Java Database Connectivity (JDBC), додає можливість розширеної взаємодії з зовнішніми системами, що робить JAAS придатним для різних типів додатків – від веб-додатків до розподілених

систем. Це відкриває можливості для створення більш складних і безпечних рішень, які використовують централізоване управління ідентифікацією та доступом.

JAAS також підтримує шифрування та цифрові підписи, що є важливим для додатків, які працюють з конфіденційною інформацією. Це робить його придатним для використання у фінансових системах, банківських додатках та системах електронної комерції, де безпека даних є критичною. Переваги і недоліки розглянемо детальніше у таблиці 2.1.

Таблиця 2.1 – Переваги і недоліки JAAS

Переваги JAAS	Недоліки JAAS
Модульна архітектура, що дозволяє легко додавати та змінювати модулі	Складність налаштування та інтеграції у складних системах
Відокремлення логіки безпеки від бізнес-логіки додатка	Обмеженість у підтримці сучасних веб-протоколів, таких як OAuth2
Підтримка різних методів автентифікації, включаючи LDAP і Kerberos	Потребує додаткових зусиль для інтеграції з іншими фреймворками
Гнучкість у налаштуванні через конфігураційні файли	Може бути менш ефективним для сучасних веб-додатків
Інтеграція з іншими компонентами Java Platform, такими як JNDI та JDBC	Недостатня підтримка новітніх стандартів безпеки без додаткових рішень

Для прикладу розглянемо діаграму класів яка відображає основні компоненти Java Authentication and Authorization Service (JAAS) і їх взаємодії. Вона включає ключові класи та інтерфейси, що використовуються для забезпечення автентифікації та авторизації у Java-застосунках. Розглянемо детальніше на рис.2.3

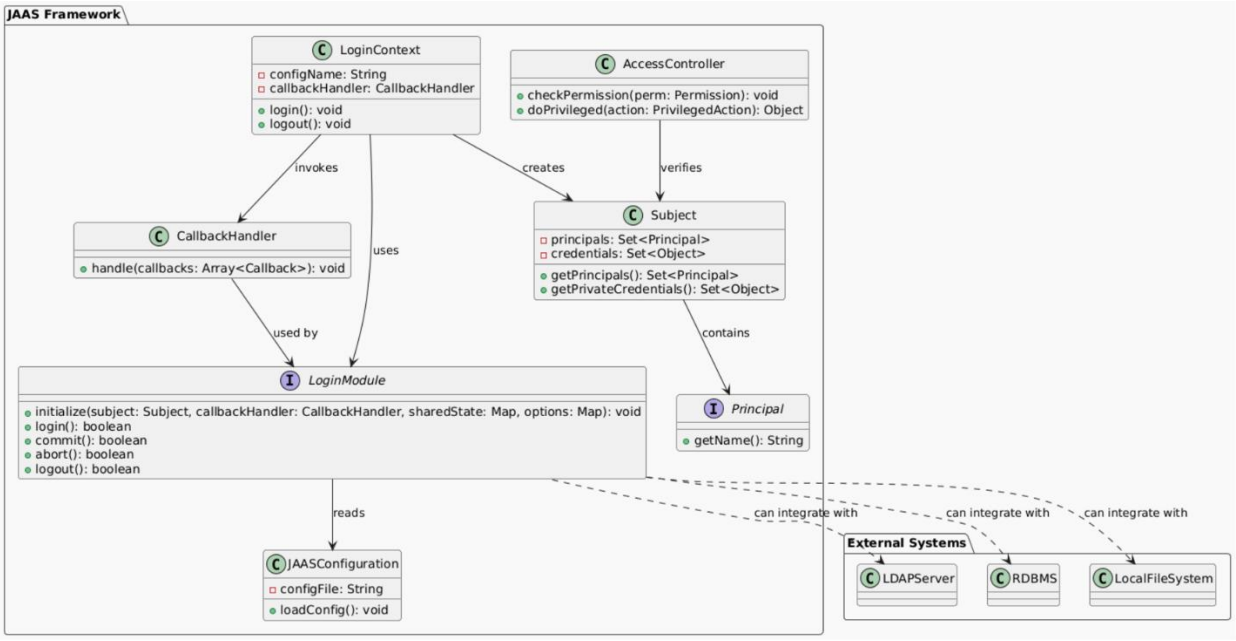


Рис.2.3 – Діаграма класів JAAS

На діаграмі зображено клас LoginContext, який відповідає за управління процесом автентифікації, приймаючи конфігурацію і викликаючи методи для ініціації автентифікації та завершення сеансу. LoginContext використовує об'єкт CallbackHandler для обробки введених даних і взаємодіє з модулем LoginModule для виконання основних завдань автентифікації.

Інтерфейс LoginModule забезпечує методи для ініціалізації, входу, підтвердження, відкату і виходу з системи, що дозволяє реалізовувати різні стратегії автентифікації. Цей інтерфейс використовується класом LoginContext і взаємодіє з конфігураційним класом JAASConfiguration, який завантажує налаштування для процесу автентифікації.

Клас Subject представляє автентифікованого користувача і містить колекції об'єктів Principal, які описують ідентифікаційні дані користувача. Subject також містить облікові дані користувача. Клас AccessController забезпечує перевірку дозволів і використовується для авторизації, підтверджуючи, чи має об'єкт Subject необхідні привілеї для виконання певних дій.

Зовнішні системи, такі як LDAPServer, RDBMS (реляційна база даних) та LocalFileSystem, можуть бути інтегровані з модулями LoginModule для

забезпечення автентифікації з різних джерел. Це підкреслює гнучкість та масштабованість JAAS у роботі з різними системами безпеки.

2.2 Spring Security

Spring Security – це потужний і гнучкий фреймворк, що надає комплексні засоби для забезпечення безпеки додатків, побудованих на основі екосистеми Spring. Інтеграція з такими компонентами, як Spring MVC, дозволяє легко використовувати фреймворк у веб-додатках, надаючи розробникам широкі можливості для налаштування політик безпеки. Spring Security підтримує великий набір сценаріїв для захисту додатків, включаючи автентифікацію, авторизацію, управління сесіями, захист від атак та шифрування даних. Логотип фреймворку можемо побачити на рис.2.4.



Рисунок 2.4 – Логотип фреймворку Spring Security

Основні цілі Spring Security – забезпечення захисту додатків від різноманітних загроз, серед яких несанкціонований доступ, CSRF-атаки (Cross-Site Request Forgery), XSS-атаки (Cross-Site Scripting) та захист даних під час їхньої передачі. Завдяки своїй архітектурі Spring Security дозволяє легко інтегрувати різні методи автентифікації, зокрема базову автентифікацію, формову автентифікацію, автентифікацію на основі токенів (JWT), а також інтеграцію з зовнішніми системами, такими як LDAP і OAuth2.

Діаграму яка відображає процеси автентифікації та авторизації у Spring Security, демонструючи основні компоненти та їхню взаємодію під час обробки клієнтських запитів, можемо побачити на рис.2.5.

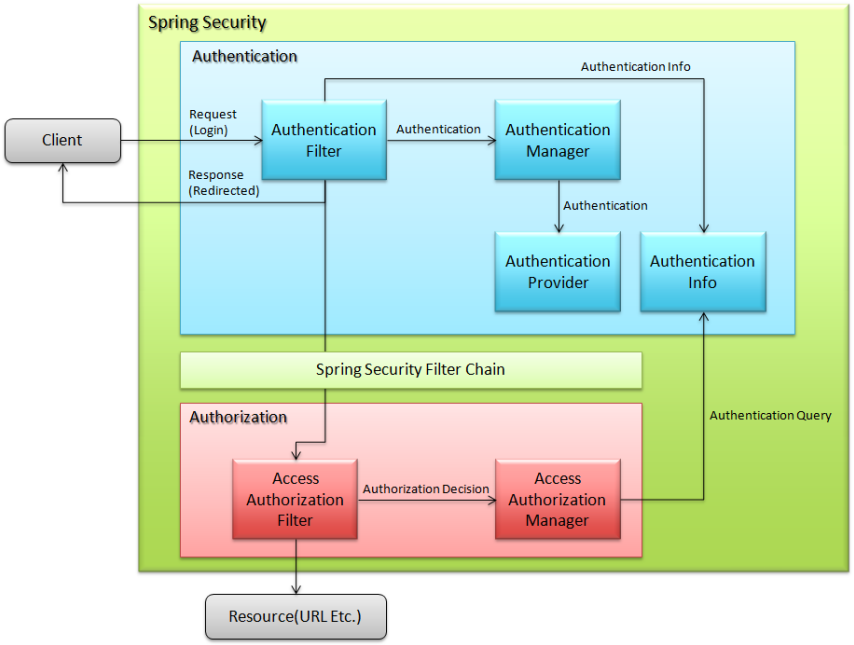


Рисунок 2.5 – Процеси автентифікації та авторизації у Spring Security

Запит передається до Authentication Manager, який є основним координатором процесу автентифікації. Authentication Manager взаємодіє з різними Authentication Provider, що реалізують логіку перевірки облікових даних користувача. Провайдери автентифікації можуть бути налаштовані для роботи з різними джерелами, такими як бази даних, LDAP або сервіси OAuth2, забезпечуючи гнучкість та розширюваність системи. Після успішної перевірки даних створюється об'єкт Authentication Info, який містить інформацію про автентифікованого користувача і може використовуватись для подальшої авторизації.

Діаграма включає також Spring Security Filter Chain, що складається з набору фільтрів, які забезпечують різні рівні захисту та перевірки запитів клієнтів. Один із цих фільтрів – Access Authorization Filter – відповідає за перевірку прав доступу користувача до запитуваних ресурсів.

Процес авторизації контролюється Access Authorization Manager, який ухвалює рішення щодо дозволу або заборони доступу до ресурсу на основі політик безпеки та інформації про користувача. Якщо авторизація пройшла успішно, клієнт отримує доступ до ресурсу, наприклад, веб-сторінки або API.

Перейдемо до розгляду далі Spring Security, гнучкість у налаштуванні – одна з головних переваг Spring Security. Використовуючи анотації, такі як @PreAuthorize, @PostAuthorize, @Secured і @RolesAllowed, розробники можуть визначати правила доступу до методів і класів додатка, що дозволяє налаштувати політики безпеки до бізнес-логіки додатка. Наприклад, за допомогою анотації @PreAuthorize("hasRole('ADMIN')") можна забезпечити доступ до певного методу тільки користувачам із роллю адміністратора.

Налаштування безпеки у Spring Security може здійснюватися за допомогою конфігураційних класів, де розробники використовують методи configure, щоб визначати правила авторизації та автентифікації. Це дозволяє швидко змінювати та налаштовувати параметри безпеки, додаючи нові механізми перевірки або адаптуючи існуючі.

Шифрування паролів – важлива функція, яка забезпечує підвищений рівень захисту даних користувачів. Spring Security підтримує автоматичне шифрування паролів за допомогою потужних алгоритмів, таких як BCrypt. BCrypt забезпечує стійкість до атак грубої сили, завдяки використанню адаптивного алгоритму, що може регулювати складність хешування залежно від вимог безпеки.

Захист від CSRF-атак – це ще одна ключова функція Spring Security, яка автоматично додає CSRF-токени до форм і перевіряє їх під час обробки запитів. Це забезпечує захист додатка від підроблених запитів, гарантуючи, що всі дії, які змінюють стан додатка, походять від авторизованих користувачів.

Інтеграція з OAuth2 дозволяє створювати безпечні додатки, які підтримують автентифікацію через сторонні сервіси, такі як Google, Facebook, GitHub та інші. Це спрощує процес автентифікації користувачів, оскільки їм не потрібно створювати нові облікові записи для доступу до додатка. OAuth2 забезпечує безпечну передачу даних про автентифікацію між клієнтом і сервером, а також дозволяє легко впроваджувати соціальний вхід.

Конфігурація безпеки через XML дозволяє налаштовувати параметри безпеки додатка без втручання у код. Це особливо корисно у великих корпоративних системах, де політики безпеки можуть змінюватися часто, і підтримка окремих XML-файлів для налаштувань є більш зручною.

Архітектура Spring Security є дуже розширюваною, що дозволяє розробникам створювати власні фільтри безпеки та компоненти автентифікації. Це означає, що фреймворк може бути адаптований під специфічні вимоги проекту, такі як власні схеми автентифікації або спеціальні методи авторизації. Власні фільтри можуть бути додані до ланцюжка фільтрів безпеки, що дозволяє розробляти рішення, які відповідають найвищим стандартам безпеки. Розглянемо архітектуру детальніше на рис.2.6.

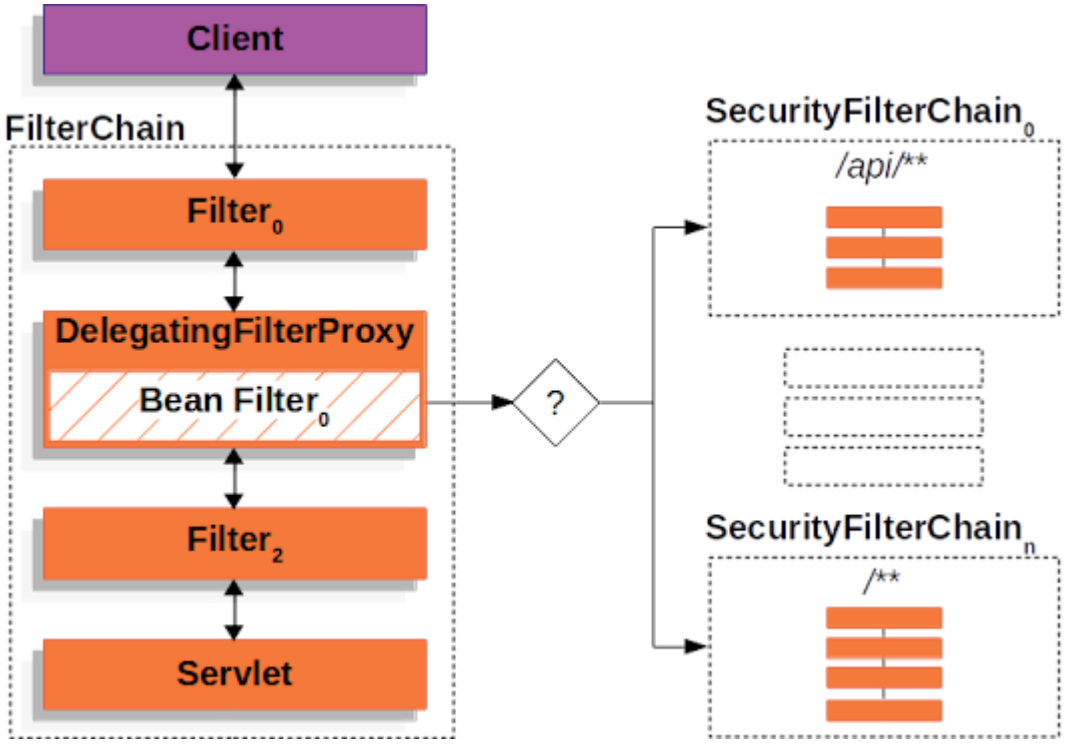


Рисунок 2.6 – Архітектура Spring Security

Сховища даних підтримуються Spring Security для зберігання інформації про користувачів та їхні ролі. Фреймворк може працювати з базами даних через JDBC або інтегруватися з LDAP для централізованого управління обліковими записами. Це забезпечує високий рівень контролю за правами

доступу користувачів і дозволяє централізовано керувати автентифікацією в організаціях з великою кількістю користувачів. Також розглянемо лістинінг 2.1. коду прикладу конфігурації для багатоетапної автентифікації.

Лістинінг 2.1 – Конфігурація для багатоетапної автентифікації

```
@Override
protected void configure(HttpSecurity http) throws Exception {
     http
       .authorizeRequests()
       .anyRequest().authenticated()
       .and()
       .formLogin()
       .and()
       .addFilterBefore(new CustomAuthFilter(),
UsernamePasswordAuthenticationFilter.class);
}
```

Підтримка багатоетапної автентифікації дозволяє підвищити безпеку, використовуючи кілька послідовних методів перевірки, таких як паролі та одноразові коди (OTP). Це особливо корисно для додатків, де потрібен високий рівень безпеки, наприклад, банківських систем або корпоративних порталів.

2.3 Apache Shiro

Apache Shiro – це потужний і гнучкий фреймворк для забезпечення безпеки Java-додатків, що надає простий, інтуїтивно зрозумілий API для автентифікації, авторизації, управління сесіями та криптографії. Основна мета Shiro – спрощення впровадження складних механізмів безпеки, що дозволяє розробникам інтегрувати його у веб-додатки, мобільні додатки та десктопні програми.

Однією з найбільших переваг Apache Shiro є його модульність та гнучкість. Фреймворк підтримує різні методи автентифікації, такі як логін за

паролем, одноразові паролі (OTP) та інтеграцію із зовнішніми сервісами, такими як LDAP, Active Directory та OAuth2. Це дозволяє розробникам вибирати та комбінувати механізми автентифікації, які найкраще відповідають вимогам проекту.

Механізм авторизації Shiro дозволяє контролювати доступ до ресурсів за допомогою ролей та дозволів. Авторизація може бути реалізована як статично (через конфігураційні файли), так і динамічно (через програмний код), що забезпечує високу гнучкість у керуванні доступом. Це важливо для програм, де доступ до певних функцій або ресурсів змінюється залежно від умов чи статусу користувача. Управління сесіями – ще одна важлива функція Apache Shiro, що забезпечує можливість зберігати інформацію про сесії користувачів як у пам’яті, так і в зовнішніх сховищах, таких як бази даних або Redis. Ця особливість дозволяє масштабувати додатки та підтримувати стабільну роботу навіть при великому навантаженні. Shiro може нативно керувати сесіями у будь-якому середовищі, включаючи програми, які не використовують контейнери Web/Servlet або EJB. Це забезпечує безперебійну роботу у самостійних додатках та інших нестандартних середовищах.

Компонент SessionDAO у Shiro відповідає за операції постійного зберігання сесій, такі як створення, читання, оновлення та видалення (CRUD). Це дозволяє налаштовувати джерела даних для збереження сесій та легко адаптувати їх під специфічні вимоги програми.

CacheManager у Shiro забезпечує створення та управління кешами для підвищення продуктивності роботи з даними безпеки. Використання кешування дозволяє Shiro швидко отримувати доступ до даних автентифікації, авторизації та управління сесіями, що особливо важливо для програм з великим числом запитів.

Криптографія є важливим елементом архітектури Shiro. Фреймворк надає прості у використанні API для хешування паролів, шифрування даних та генерації випадкових чисел. Це дозволяє розробникам легко впроваджувати криптографічний захист даних, використовуючи стандартні шифри та

алгоритми, без складних конфігурацій. Ця функціональність робить Shiro ідеальним вибором для програм, де необхідний високий рівень захисту даних.

Realms у Shiro діють як «містки» між фреймворком та джерелами даних додатка, такими як бази даних або LDAP. Вони дозволяють Shiro отримувати інформацію про користувачів, ролі та дозволи для автентифікації та авторизації. Realms є гнучкими у налаштуванні: розробник може додавати нові джерела даних або використовувати кілька realms для розширеної функціональності.

На рис. 2.7 показано архітектуру Apache Shiro та його ключові компоненти, що взаємодіють під час обробки запитів на автентифікацію та авторизацію.

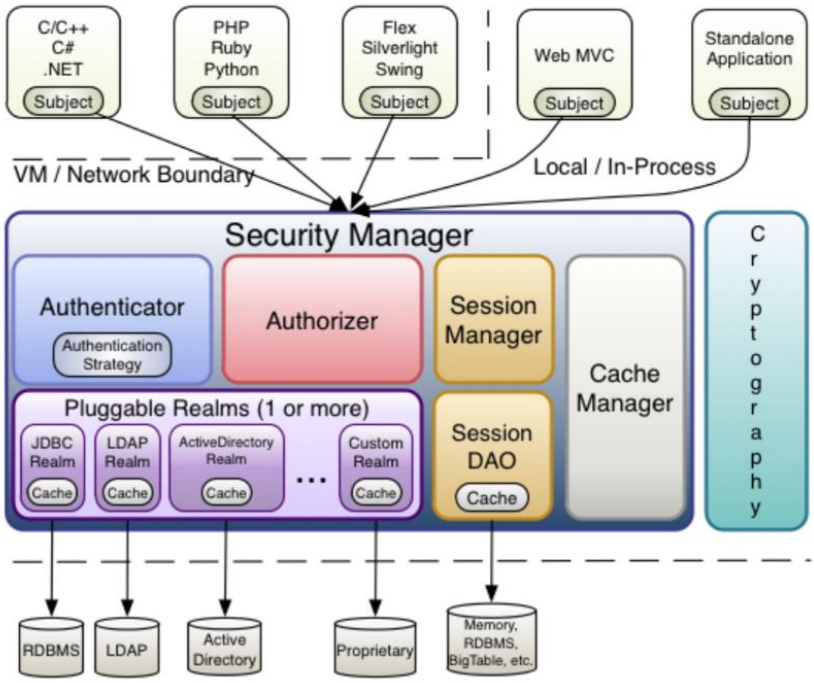


Рисунок 2.7 – Архітектура Apache Shiro

Архітектура включає Subject, який є представленням користувача, що взаємодіє з програмним забезпеченням, та SecurityManager, який координує всі операції безпеки. Authenticator відповідає за перевірку облікових даних користувачів та їх автентифікацію через один або декілька realms. Authorizer визначає, чи має користувач необхідні дозволи для доступу до певних

ресурсів. SessionManager управляє життєвим циклом сесій користувачів, а CacheManager забезпечує збереження та швидкий доступ до даних.

Ця архітектура забезпечує високий рівень безпеки, зручність у використанні та можливість масштабування для будь-яких типів Java-додатків.

Узагальнюючи розглянемо таблицю 2.2, яка описує ключові компоненти Apache Shiro та їх функції.

Таблиця 2.2 - Ключові компоненти Apache Shiro та їх функції

Компонент	Опис	Функція
Subject	Представлення користувача або сутності, що взаємодіє з програмою.	Взаємодія з програмою від імені користувача.
SecurityManager	Центральний об'єкт архітектури Shiro, який координує роботу всіх компонентів безпеки.	Управління операціями безпеки.
Authenticator	Відповідає за процес автентифікації та перевірку облікових даних.	Виконання автентифікації користувачів.
Authorizer	Виконує перевірку прав доступу користувача до ресурсів.	Контроль доступу та перевірка дозволів.
SessionManager	Управляє сесіями користувачів, забезпечуючи їх створення та контроль життєвого циклу.	Управління сесіями у веб- та не-веб-середовищах.
SessionDAO	Відповідає за постійне зберігання сесій у зовнішніх джерелах, таких як бази даних або Redis.	CRUD-операції для сесій.
CacheManager	Керує кешами для оптимізації доступу до даних безпеки.	Збереження даних для покращення продуктивності.
Cryptography	Надає засоби для хешування паролів, шифрування та генерації випадкових чисел.	Забезпечення криптографічного захисту даних.
Realms	Виступають мостами між Shiro та джерелами даних для автентифікації та авторизації користувачів.	Отримання облікових даних та ролей користувачів.

У висновку можна зазначити, що Apache Shiro є потужним та гнучким фреймворком для забезпечення безпеки Java-додатків, який пропонує

комплексний підхід до автентифікації, авторизації, управління сесіями та криптографії. Архітектура Shiro дозволяє розробникам легко інтегрувати фреймворк у свої проєкти завдяки його модульності та простоті використання. Ключовими компонентами, такими як SecurityManager, Subject, Authenticator, Authorizer, SessionManager, Realm, SessionDAO, CacheManager і Cryptography, забезпечується чітка та ефективна робота системи безпеки.

Ці компоненти працюють разом, забезпечуючи високий рівень контролю над автентифікацією та авторизацією користувачів, управлінням сесіями і збереженням даних у безпечному вигляді. Використання кешування та криптографії підвищує продуктивність і захист додатків. Модульний підхід та розширюваність архітектури Shiro дозволяють адаптувати його до специфічних вимог проєктів, що робить його незамінним інструментом для створення безпечних та надійних Java-додатків.

Висновки до другого розділу

У висновку до другого розділу можна зазначити, що аналіз фреймворків для організації безпеки роботи Java-застосунків дозволив детально дослідити їхні ключові особливості, переваги та функціональні можливості. Java Authentication and Authorization Service (JAAS), Spring Security та Apache Shiro є найпоширенішими рішеннями, кожне з яких має свої унікальні підходи до забезпечення безпеки.

JAAS забезпечує базову функціональність автентифікації та авторизації, що дозволяє легко інтегрувати його у Java-додатки та розділити логіку безпеки від основного коду. Його модульна структура забезпечує гнучкість у виборі методів автентифікації та управлінні доступом до ресурсів.

Spring Security, у свою чергу, є найпотужнішим та найбільш конфігурованим фреймворком, що інтегрується з іншими компонентами Spring. Він пропонує широкий спектр функцій для налаштування автентифікації, авторизації, управління сесіями та захисту від веб-загроз.

Завдяки своїй масштабованості та підтримці різних механізмів захисту Spring Security ідеально підходить для великих корпоративних додатків, де важливі високий рівень безпеки та підтримка складних сценаріїв доступу.

Apache Shiro є гнучким рішенням, що дозволяє розробникам легко налаштовувати автентифікацію, авторизацію, управління сесіями та криптографію. Його простота використання, модульність та підтримка різних джерел даних роблять його зручним вибором для широкого кола проектів, включаючи веб-додатки, мобільні додатки та десктопні програми.

У підсумку, вибір відповідного фреймворку для забезпечення безпеки Java-застосунків залежить від специфічних вимог проекту. JAAS підходить для базових завдань безпеки, Spring Security – для проєктів з комплексними вимогами, а Apache Shiro – для проектів, які потребують гнучких і легких у впровадженні рішень.

РОЗДІЛ 3 РОЗРОБКА ЗАСТОСУНКУ З РЕАЛІЗАЦІЯМИ МЕХАНІЗМІВ БЕЗПЕКИ ФРЕЙМВОРКАМИ JAAS, SPRING SECURITY ТА APACHE SHIRO

3.1 Вимоги до системи для реалізації механізмів безпеки

Функціональні вимоги є ключовою частиною розробки інформаційних систем, оскільки вони визначають основні можливості та завдання, які система повинна виконувати для задоволення потреб користувачів. У даному випадку функціональні вимоги були розроблені для інформаційної системи, що призначена для перегляду, обліку та управління коміксами. Основними завданнями цієї системи є забезпечення зручного інтерфейсу для читачів, збереження структурованої бази даних про комікси, авторів, видавців, художників та перекладачів, а також надання можливостей для командної роботи над перекладами.

Ця система має бути корисною для широкого кола користувачів, включаючи читачів, перекладачів, видавців і адміністраторів. Вимоги також передбачають інтеграцію функцій безпеки для захисту даних і забезпечення контрольованого доступу до ресурсів. У таблиці нижче представлені функціональні вимоги у структурованому форматі (див.табл. 3.1).

Таблиця 3.1 – Функціональні вимоги програмної системи

№	Функціональна вимога	Деталізація
1	Ведення довідника коміксів	- Додавання, редагування, видалення, перегляд коміксів - Зберігання метаданих (назва, жанр, опис, обкладинка, дата випуску) - Прив’язка до авторів, художників та перекладачів
2	Ведення довідника глав та сторінок	- Створення структури глав для кожного коміксу - Завантаження зображень сторінок із зазначенням порядку
3	Ведення довідників авторів, видавців, художників	- Додавання інформації про авторів, видавців та художників - Прив’язка цих осіб до відповідних коміксів
4	Ведення обліку читачів	- Реєстрація, авторизація та управління профілями - Налаштування профілю

Нефункціональні вимоги є невід'ємною частиною розробки будь-якої інформаційної системи, адже вони визначають її якісні характеристики, що впливають на продуктивність, масштабованість, безпеку, надійність та зручність використання. У контексті інформаційної системи для перегляду коміксів нефункціональні вимоги забезпечують створення ефективного, захищеного та стабільного середовища, яке відповідає очікуванням користувачів і вимогам сучасних технологічних стандартів.

До нефункціональних вимог цієї системи належать критерії щодо продуктивності (час завантаження сторінок), безпеки (автентифікація та шифрування даних), надійності (відновлення після збоїв), сумісності (робота на різних пристроях), а також зручності використання (інтуїтивний

інтерфейс). Ці вимоги формують основу для забезпечення якості кінцевого продукту та підтримки його довгострокової експлуатації.

Інформаційна система для перегляду коміксів повинна відповідати сучасним стандартам кібербезпеки для забезпечення захисту даних користувачів, контрольованого доступу до функціоналу системи та забезпечення довіри до її роботи. Основними аспектами безпеки є автентифікація користувачів, шифрування конфіденційної інформації, використання цифрового підпису та система ролей і прав доступу. Це дозволить запобігти несанкціонованому доступу, захистити дані та забезпечити їх цілісність. Детальніше розглянемо у таблиці 3.2.

Таблиця 3.2 – Вимоги до безпеки програмної системи

№	Вимога	Деталізація
1	Автентифікація користувачів	- Підтримка багатофакторної автентифікації - Зберігання хешованих паролів з використанням сучасних алгоритмів (наприклад, bcrypt)
2	Система ролей користувачів та прав доступу	- Наявність ролей: адміністратор, читач, - Визначення дозволів для кожної ролі (доступ до перегляду, редагування)
3	Шифрування конфіденційних даних	- Шифрування персональних даних (паролі) у базі даних

Розроблені функціональні та нефункціональні вимоги до інформаційної системи для перегляду коміксів забезпечують її ефективну роботу, зручність для користувачів та високий рівень безпеки. Вимоги охоплюють ключові аспекти управління даними, доступу, захисту конфіденційної інформації та взаємодії користувачів із системою. Їх чітке формулювання створює основу для якісної розробки системи, що відповідає сучасним стандартам і очікуванням користувачів.

3.2 Побудова діаграм варіантів використання

Візуальне моделювання в UML (Unified Modeling Language) представляє собою процес поступового деталізованого проєктування програмної системи. Цей процес починається з найбільш загальної та абстрактної концептуальної моделі, яка відображає вихідну систему, і завершується створенням логічної, а потім і фізичної моделі майбутньої програмної системи. Таким чином, UML дозволяє представити складний процес моделювання системи на кількох рівнях абстракції, починаючи з розуміння високорівневих вимог і закінчуючи деталізацією компонентів для їхньої реалізації.

Одним із перших етапів цього процесу є створення діаграми варіантів використання (use case diagram), яка описує функціональне призначення системи. Ця діаграма відповідає на питання: що система має робити в процесі свого функціонування? Діаграма варіантів використання є вихідним концептуальним поданням системи і служить основою для її подальшої деталізації під час проєктування і розробки.

Розробка діаграми варіантів використання переслідує такі важливі цілі:

1. визначення загальної межі та контексту предметної області
На перших етапах проєктування діаграма допомагає чітко визначити, якими будуть межі системи і як вона взаємодіє із зовнішнім середовищем. Це забезпечує загальне розуміння предметної області, в якій працюватиме система.

2. Формулювання загальних вимог до функціональної поведінки системи. Діаграма дозволяє на високому рівні описати ключові функціональні можливості, які система повинна забезпечувати для користувачів і зовнішніх сутностей.

3. Розробка вихідної концептуальної моделі системи
Діаграма виступає концептуальною моделлю, яка в подальшому деталізується.

Вона стає основою для створення логічних моделей (структура компонентів) і фізичних моделей (архітектура і реалізація).

4. Підготовка документації для взаємодії розробників із замовниками та користувачам. Діаграма слугує зрозумілим інструментом для обговорення функціональності системи між технічними спеціалістами, бізнес-замовниками та кінцевими користувачами. Вона допомагає уникнути непорозумінь у вимогах.

Суть діаграми варіантів використання полягає в тому, щоб представити проєктовану систему у вигляді сукупності взаємодій між різними сутностями, які називаються акторами, та системою. Актори, або дійові особи, є ключовими елементами цієї діаграми і виступають у ролі зовнішніх сутностей, що взаємодіють із системою. Це можуть бути люди, наприклад, користувачі або адміністратори; технічні пристрої, такі як сенсори або периферійне обладнання; програми, що передають дані чи взаємодіють через API; або навіть інші системи, які взаємодіють з моделюємим середовищем. Важливо, що вибір акторів залежить від цілей розробки і визначається розробником залежно від вимог системи.

Актори виступають ініціаторами певних дій чи подій, які запускають процеси в системі. Для опису цих процесів використовується концепція варіантів використання, або use cases. Варіант використання описує конкретний функціональний сервіс, який система надає актору під час взаємодії. Наприклад, це може бути автентифікація користувача, формування звіту або завантаження файлів. Кожен варіант використання визначає набір дій або кроків, які система виконує під час діалогу з актором. При цьому діаграма варіантів використання зосереджується на описі функціональних можливостей системи і не деталізує технічну реалізацію цих процесів.

У діаграмі актора і варіант використання пов’язує взаємодія, яка відображає, як саме певна зовнішня сутність використовує функціонал системи. Такий підхід дозволяє відобразити взаємозв’язки між акторами і системою, їхню роль у виконанні бізнес-процесів, а також забезпечує загальне

розуміння функціональності системи. Розглянемо діаграму варіантів використання для незареєстрованого користувача на рис.3.1.



Рисунок 3.1 – Діаграма використання системи для незареєстрованого користувача

Основним актором є "Незареєстрований користувач", який взаємодіє з системою через кілька основних варіантів використання. Першою дією, яку може виконати користувач, є реєстрація, що дозволяє створити обліковий запис для подальшої роботи з системою. Після реєстрації користувач може здійснити авторизацію, щоб отримати доступ до додаткового функціоналу.

Незареєстрований користувач також має можливість переглядати каталог коміксів. Ця дія включає два підваріанти використання: перегляд глав коміксу та перегляд детальної інформації про сам комікс. Окрім цього, у межах варіанту "Перегляд інформації про комікс" розширюється функціонал із можливістю перегляду інформації про художника, перекладача та автора, що створює додаткову деталізацію.

На діаграмі відображено зв’язки між варіантами використання. Зокрема, використано відношення типу «include» між переглядом каталогу і переглядом глав коміксу, що означає обов’язковість виконання цієї дії. Також показано розширення функціоналу за допомогою відношення «extend» між переглядом

інформації про комікс та переглядом інформації про художника, автора і перекладача, що дозволяє доповнити основну функціональність системи.

Наступний варіант використання розглянемо на рис.3.2.

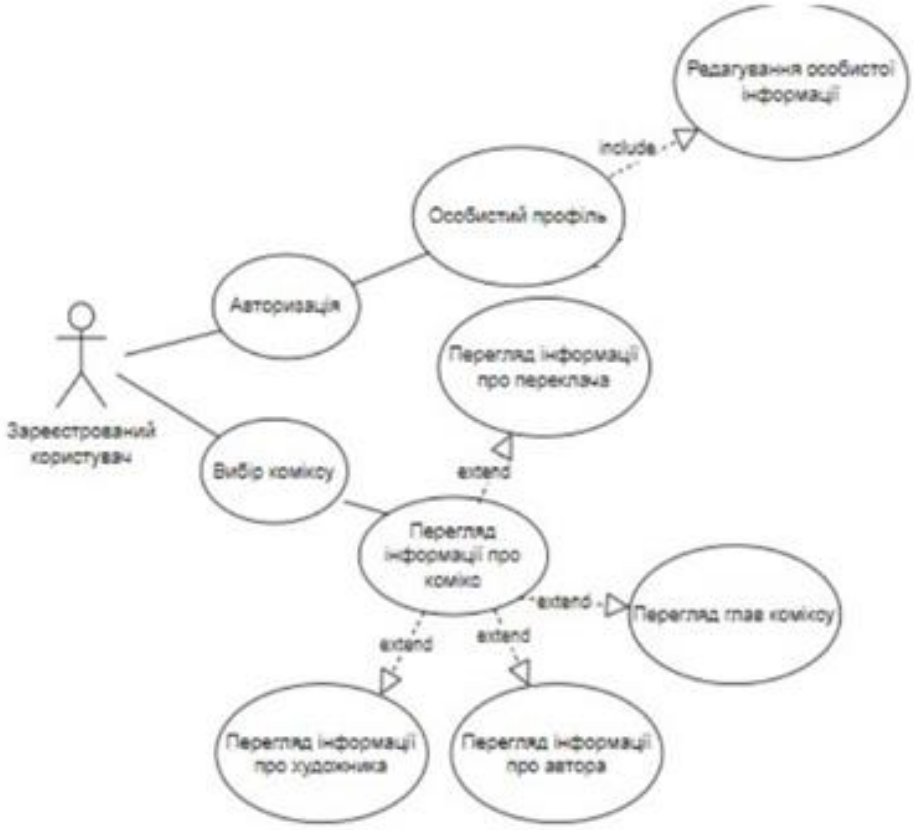


Рисунок 3.2 - Діаграма варіантів використання для зареєстрованого користувача

Основним актором є "Зареєстрований користувач", який має доступ до розширених можливостей після авторизації. Після входу в систему користувач отримує доступ до свого особистого профілю, де він може редагувати свої особисті дані. Взаємодія з профілем є центральною частиною функціональності для зареєстрованих користувачів, яка включає як перегляд інформації, так і її оновлення.

Однією з ключових дій є вибір коміксу, що дозволяє користувачу переглядати детальну інформацію про вибраний комікс, а також його глави. У рамках перегляду коміксу користувач може отримувати додаткові деталі,

зокрема інформацію про художника, автора або перекладача. Ця функціональність забезпечується через розширення основного варіанту використання "Перегляд інформації про комікс".

Користувач також має доступ до роботи з коміксами. Ця можливість включає додавання нового коміксу до своїх закладок. Цей функціонал забезпечує гнучкість у роботі з вмістом системи.

На діаграмі показано зв’язки між варіантами використання, включаючи відношення "include", яке вказує на обов’язковість виконання певних дій для доступу до функцій, і "extend", що розширює функціональність основних варіантів використання додатковими можливостями.

Останній варіант розглянемо на рис.3.3.



Рисунок 3.3 - Діаграма варіантів використання для адміністратора

Основним актором є "Адміністратор", який після авторизації отримує доступ до особистого профілю та широкого спектра дій. Адміністратор може редагувати особисту інформацію, працювати з коміксами, включаючи додавання нових коміксів і глав, а також редагування та видалення інформації про них.

Окрім цього, адміністратор має функцію роботи з базою даних, яка включає управління каталогом (додавання нових жанрів, редагування інформації) та контроль різної інформації в системі. Через функціонал "Робота з коміксами" адміністратор може переглядати глави, вибирати комікси та переглядати необхідну інформацію. Взаємозв'язки "include" і "extend" у діаграмі вказують на залежності між основними та додатковими функціями, забезпечуючи чітку структуру взаємодії адміністратора з системою.

3.3 Архітектура програми

Архітектура проєкту побудована на основі шаблону MVC (Model-View-Controller) з використанням Spring Boot для створення веб-застосунку з підтримкою фреймворків безпеки. Проєкт використовує Thymeleaf як шаблонізатор, Spring Security, Apache Shiro, а також JAAS для реалізації механізмів автентифікації та авторизації.

Шари архітектури

- Presentation Layer (Шар представлення): Відповідає за відображення даних користувачу через веб-інтерфейс. Реалізується з використанням Thymeleaf.
- Service Layer (Шар бізнес-логіки): Містить основний бізнес-код застосунку.
- Repository Layer (Шар доступу до даних): Взаємодіє з базою даних через JPA.
- Security Layer (Шар безпеки): Відповідає за налаштування безпеки за допомогою профілів та механізмів автентифікації.

Основні залежності у файлі pom.xml забезпечують функціональність представлення, доступу до даних та інтеграції з механізмами безпеки. Залежності включають Thymeleaf, Spring Security, Apache Shiro, підтримку LDAP, MySQL для збереження даних, а також механізми інтеграції з JAAS.

Spring Boot Starter Thymeleaf додає шаблонізатор для побудови веб-інтерфейсу, а Spring Security надає основний функціонал безпеки. Apache Shiro та LDAP використовуються для альтернативної авторизації та автентифікації. Thymeleaf Extras Spring Security інтегрує механізм безпеки в HTML-шаблони.

Лістинг 1 – Вміст файлу pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
parent</artifactId>
        <version>3.4.0</version>
        <relativePath/> <!-- lookup parent from repository
-->
    </parent>
    <groupId>org.example</groupId>
    <artifactId>Dipl</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Dipl</name>
    <description>Dipl</description>
    <url/>
    <licenses>
        <license/>
    </licenses>
```

```
        <developers>
            <developer/>
        </developers>
        <scm>
            <connection/>
            <developerConnection/>
            <tag/>
            <url/>
        </scm>
        <properties>
            <java.version>23</java.version>
        </properties>
        <dependencies>
            <dependency>
                <groupId>org.thymeleaf</groupId>
                <artifactId>thymeleaf</artifactId>
                <version>3.1.2.RELEASE</version>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-
thymeleaf</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-
web</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-
test</artifactId>
```

```
                    <scope>test</scope>
                </dependency>
                <dependency>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-data-
jpa</artifactId>
                </dependency>
                <dependency>
                    <groupId>mysql</groupId>
                    <artifactId>mysql-connector-java</artifactId>
                    <version>8.0.33</version>
                </dependency>
                <dependency>
                    <groupId>org.thymeleaf.extras</groupId>
                    <artifactId>thymeleaf-extras-
springsecurity6</artifactId>
                    <version>3.1.2.RELEASE</version>
                </dependency>
                <dependency>
                    <groupId>javax.servlet</groupId>
                    <artifactId>javax.servlet-api</artifactId>
                    <version>4.0.1</version> <!-- Ensure to match
the version with your servlet container (e.g., Tomcat) -->
                </dependency>
                <dependency>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-
jdbc</artifactId>
                </dependency>
            </dependencies>

            <profiles>
                <profile>
                    <id>spring-security</id>
                    <dependencies>
                        <dependency>
```

```
     <groupId>org.springframework.boot</groupId>
                              <artifactId>spring-boot-starter-
security</artifactId>
                         </dependency>
                    </dependencies>
               </profile>
               <profile>
                    <id>apache-shiro</id>
                    <dependencies>
                         <dependency>
                              <groupId>org.apache.shiro</groupId>
                              <artifactId>shiro-
spring</artifactId>
                              <version>2.0.2</version>
                         </dependency>
                         <dependency>
                              <groupId>org.apache.shiro</groupId>
                              <artifactId>shiro-web</artifactId>
                              <version>2.0.2</version>
                         </dependency>
                         <dependency>
                              <groupId>org.apache.shiro</groupId>
                              <artifactId>shiro-core</artifactId>
                              <version>2.0.2</version>
                         </dependency>
                    </dependencies>
               </profile>
               <profile>
                    <id>jaas</id>
                    <dependencies>
                         <dependency>

     <groupId>org.springframework.security</groupId>
                              <artifactId>spring-security-
ldap</artifactId>
```

```
                    <version>6.4.1</version>
                </dependency>
            </dependencies>
            <!-- JAAS не потребує додаткових залежностей -
->
        </profile>
    </profiles>

    <build>
        <plugins>
            <plugin>

<groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-
plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

У файлі pom.xml визначені профілі для керування залежностями та налаштуваннями у відповідності до обраного механізму безпеки. Використання профілів дозволяє налаштувати різні механізми безпеки без необхідності змінювати основний pom.xml.

- Профіль spring-security додає основний модуль Spring Security.
- Профіль apache-shiro додає залежності, які відповідають за підтримку Apache Shiro.
- Профіль jaas налаштовує механізм JAAS, зокрема через LDAP.’

Основні шари архітектури Spring-застосунку

Архітектура застосунку будується на використанні чітко розділених шарів, які виконують різні завдання та відповідають структурі проєкту:

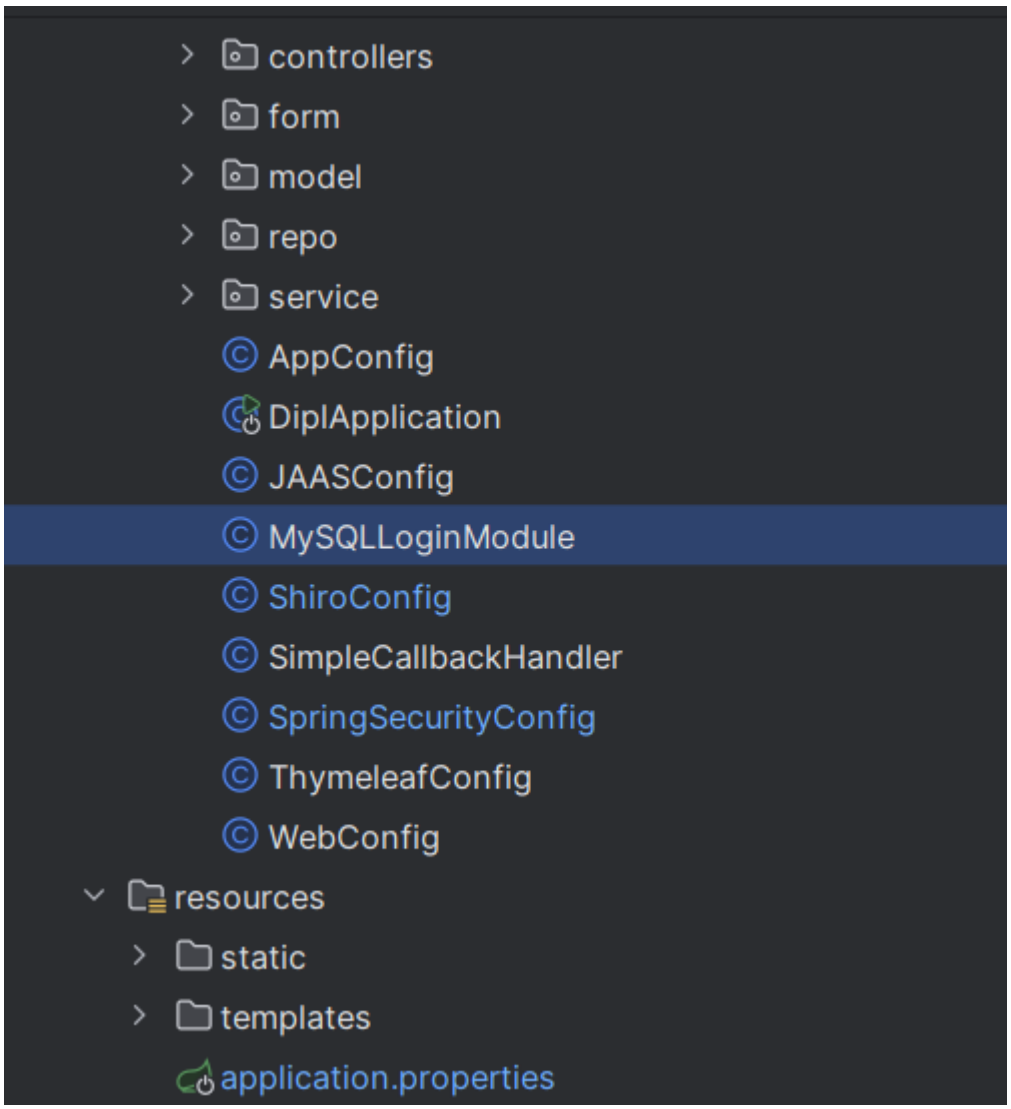


Рисунок 3.4 – Структура проєкту

Структура каталогів проєкту:

- controller - Шар представлення (Presentation Layer): містить контролери, які обробляють HTTP-запити.

- service - Шар бізнес-логіки (Service Layer): містить бізнес-логіку застосунку.

- repository - Шар доступу до даних (Repository Layer): містить інтерфейси, що працюють з базою даних через JPA.

- model - Містить сутності (Entity-класи), які відповідають за взаємодію з базою даних.

- Файли з придставкою config - конфігурування застосунку (Spring Boot та налаштування фрейворків).

- resources
    - templates - HTML-шаблони на основі Thymeleaf.
    - static – файли для статичного відображення (корстинки та CSS стилі).
    - application.properties - Налаштування застосунку.

- DiplApplication.java - Головний клас, який запускає Spring Boot-застосунок.

Для тестування роботи з різних фреймворків безпеки у застосунку використовуються профілі Maven. Це дозволяє легко перемикатися між різними механізмами автентифікації чи авторизації та підключати лише необхідні для них залежності щоб виключити елементи конфліктування

Профілі у pom.xml:

- spring-security
    - Додає залежності для Spring Security.
    - Підходить для стандартної автентифікації та авторизації через механізм Spring Security.
- apache-shiro
    - Додає залежності для Apache Shiro.
    - Забезпечує альтернативний механізм автентифікації.
- jaas
    - Додає залежності для JAAS.
    - Забезпечує альтернативний механізм автентифікації

Нижче наведено діаграму класів моделей програми (див. рис. 3.4):

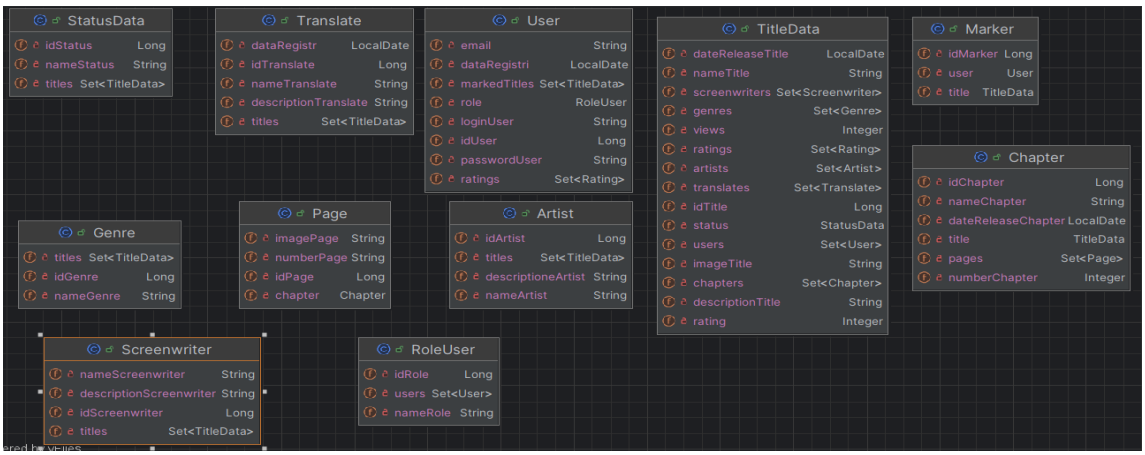


Рисунок 3.5 – Діаграма класів моделей програми

Нижче наведено діаграму класів контролерів програми (див. рис. 3.5):

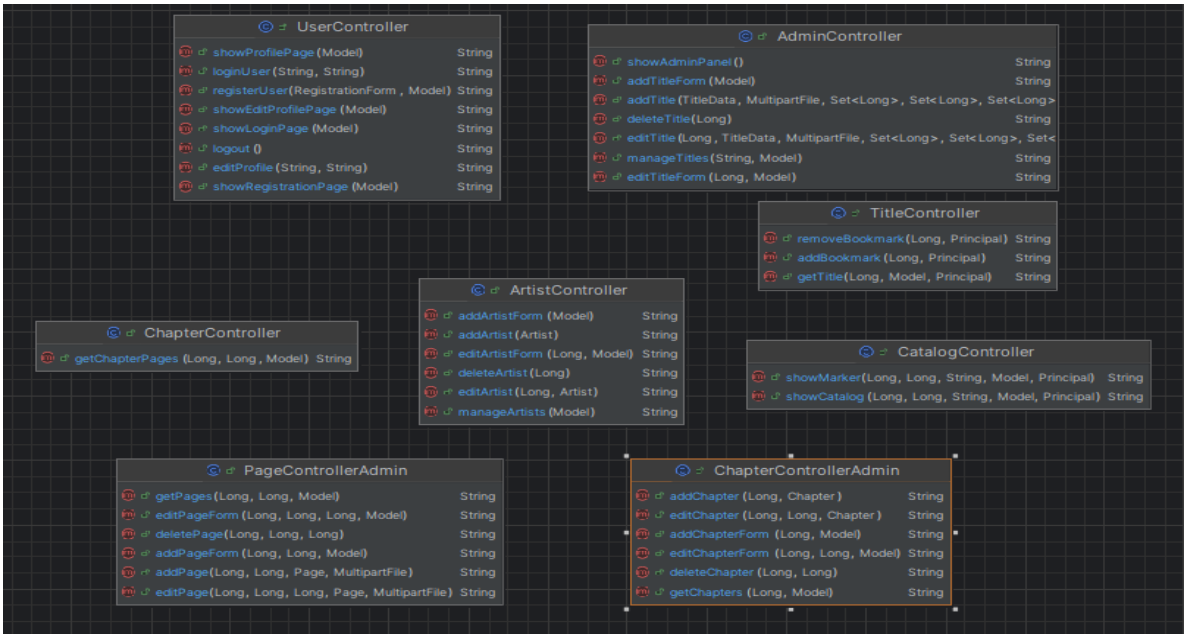


Рисунок 3.6 – Діаграма класів контролерів програми

3.4 Інтерфейс програми

Головне вікно програми представлено рисунку 3.7:

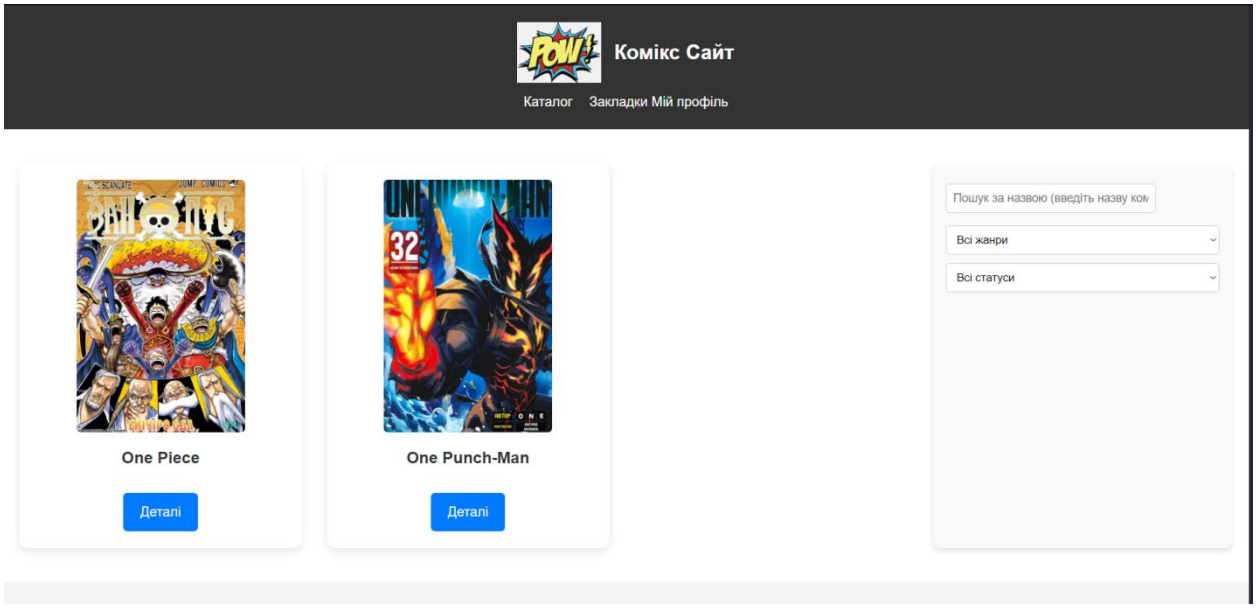


Рисунок 3.7 – Головне вікно програми

Сторінки для перегляду коміксу та глави наведено на рисунках 3.8 – 3.9

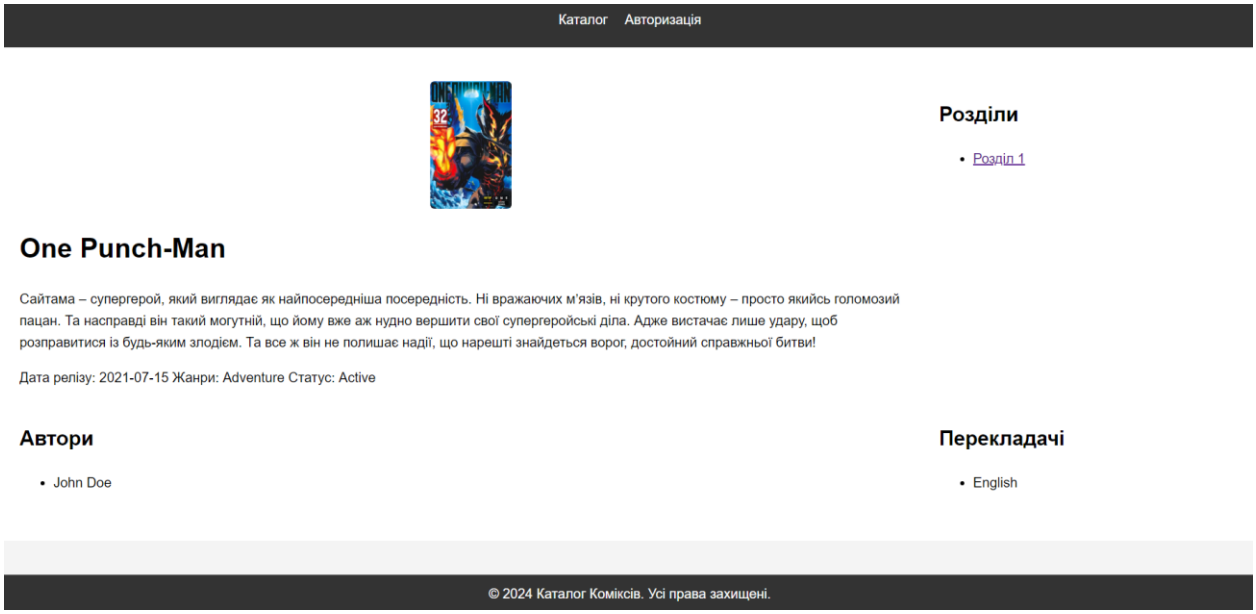


Рисунок 3.8 – Головна сторінка коміксу

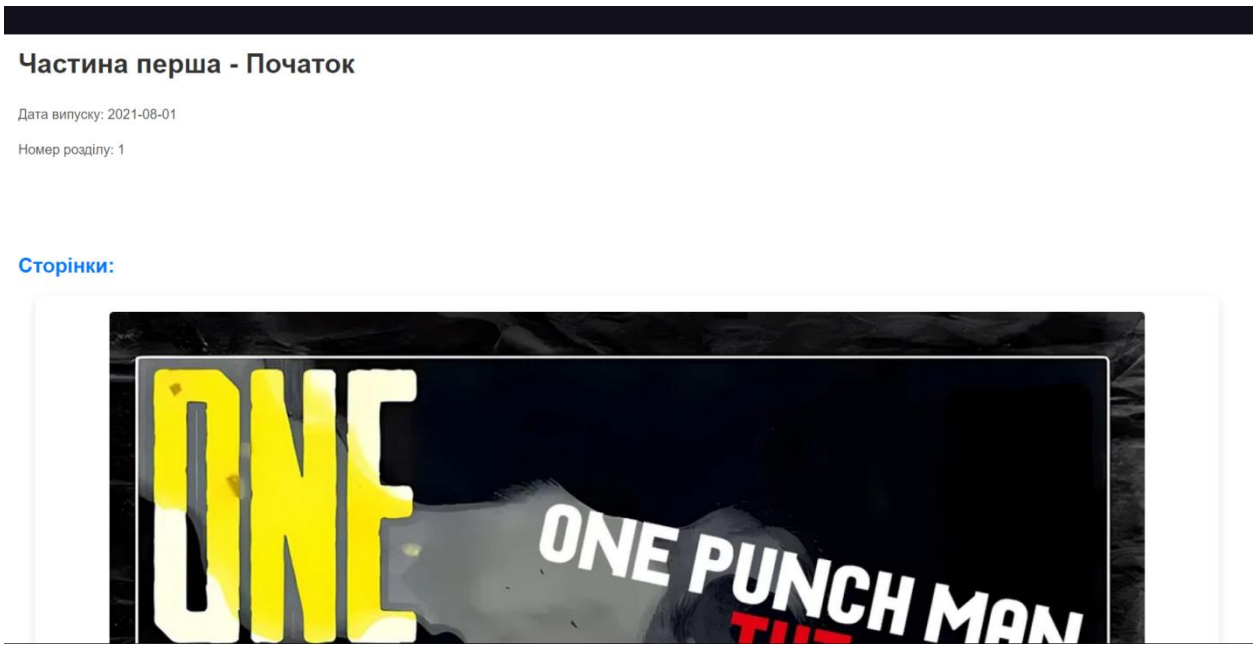


Рисунок 3.9 – Сторінка перегляду глави

Сторінки авторизації, реєстрації, профілю, закладок, забороненого доступу наведено на рисунках 3.10 – 3.14

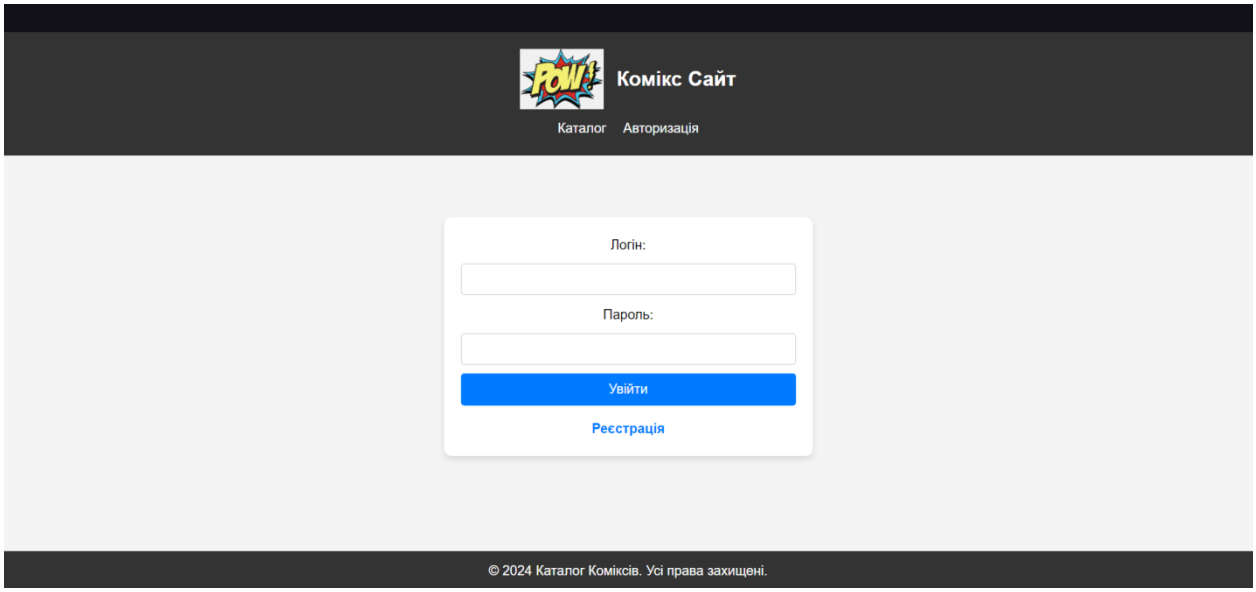


Рисунок 3.10 – Сторінка авторизації

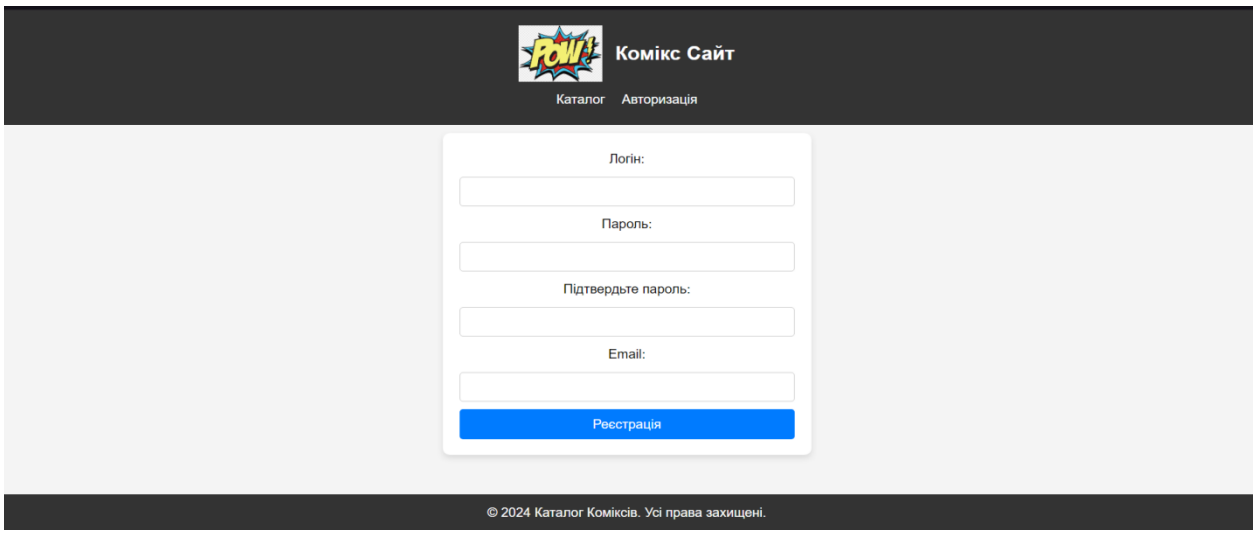


Рисунок 3.11 – Сторінка реєстрації

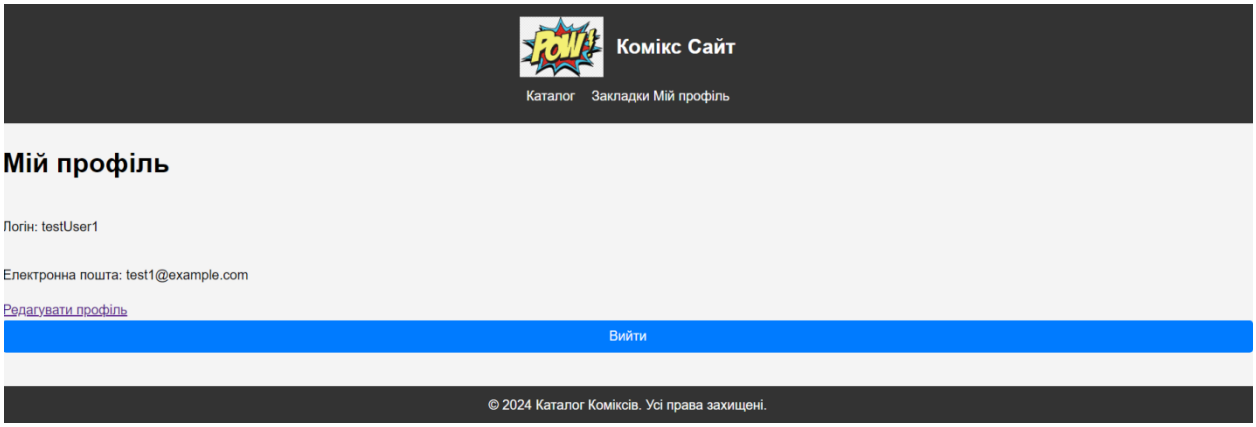


Рисунок 3.12 – Сторінка профілю користувача

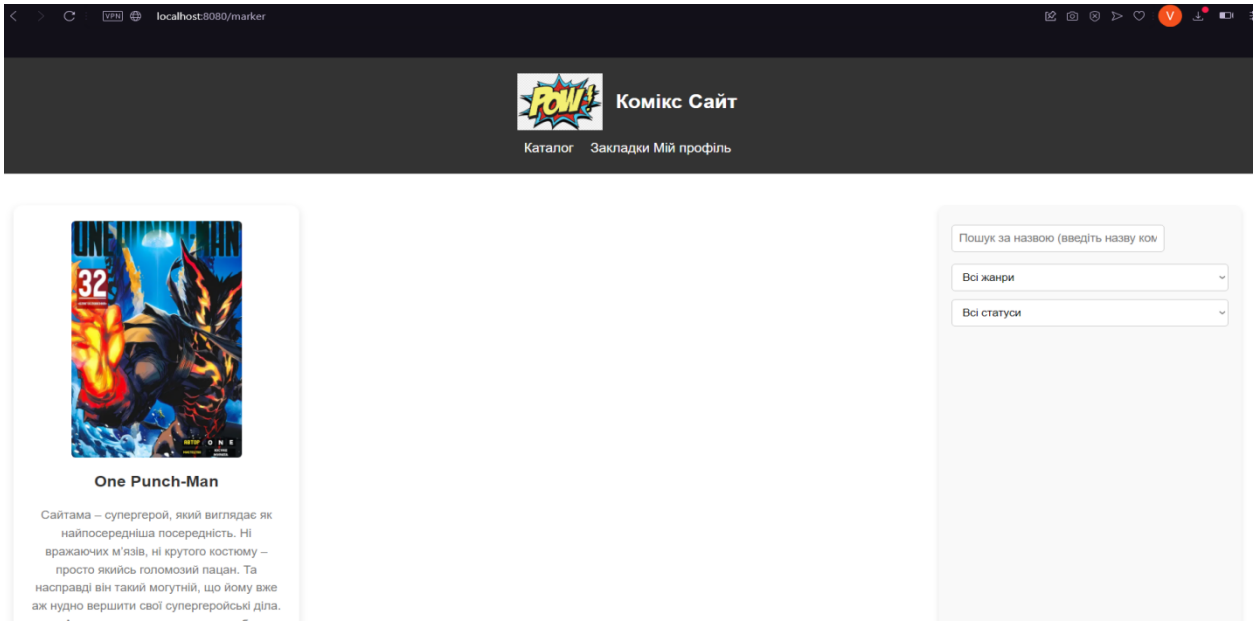


Рисунок 3.13 – Сторінка закладок користувача

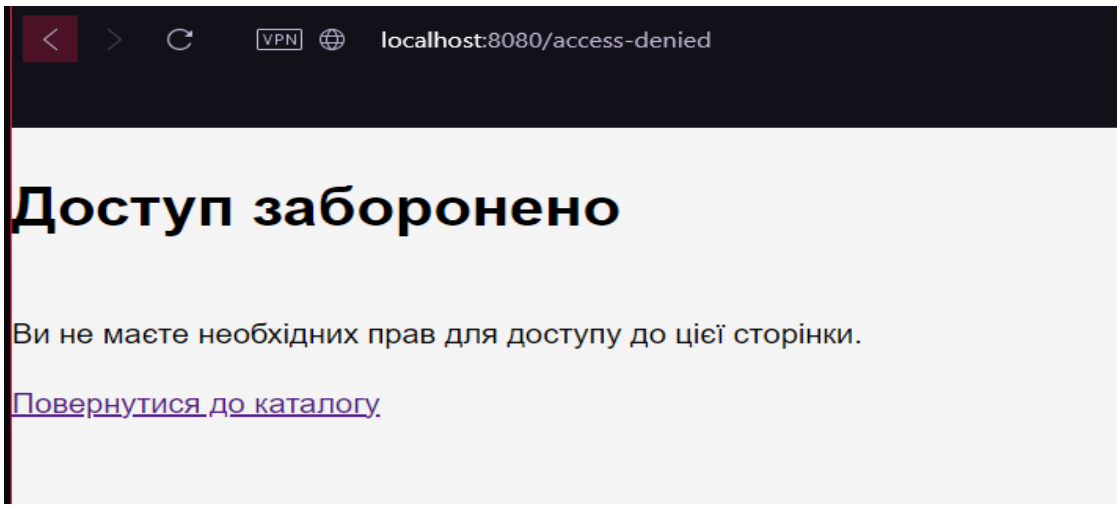


Рисунок 3.14 – Сторінка забороненого доступу

Для переходу до панелі адміністратора користувач з роллю "admin" має кнопку для переходу на панель. Сторінки адміністративного інтерфейсу представлені на ілюстраціях 3.15 – 3.21

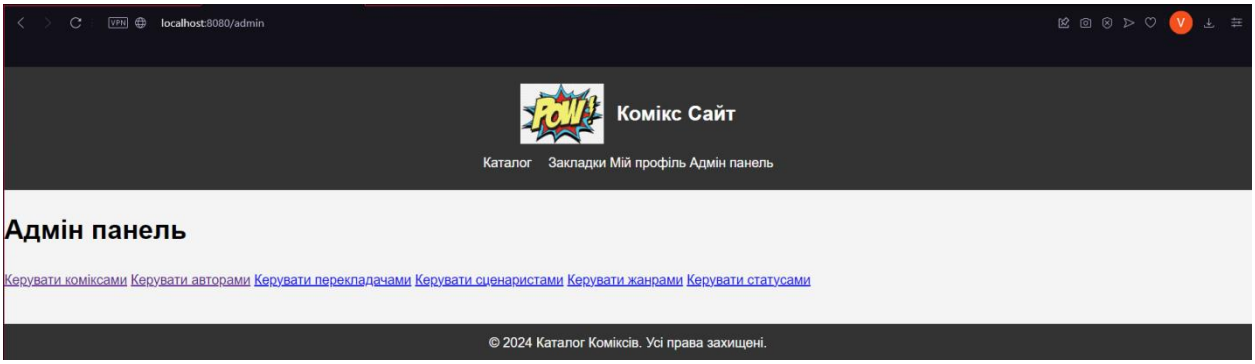


Рисунок 3.15 – Адмін панель

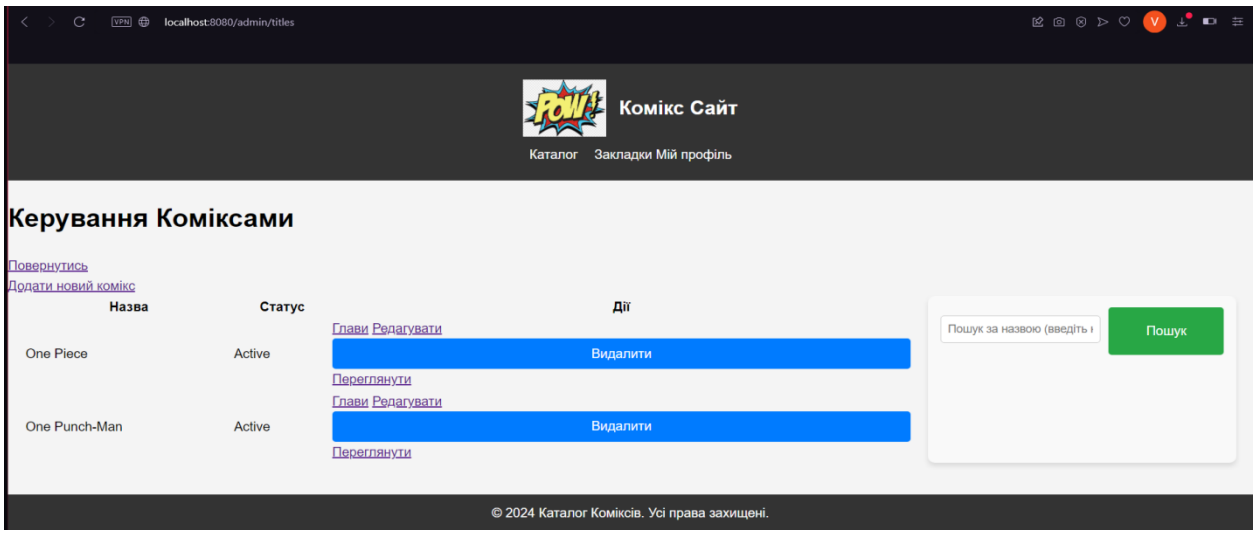


Рисунок 3.16 – Сторінка керування коміксами

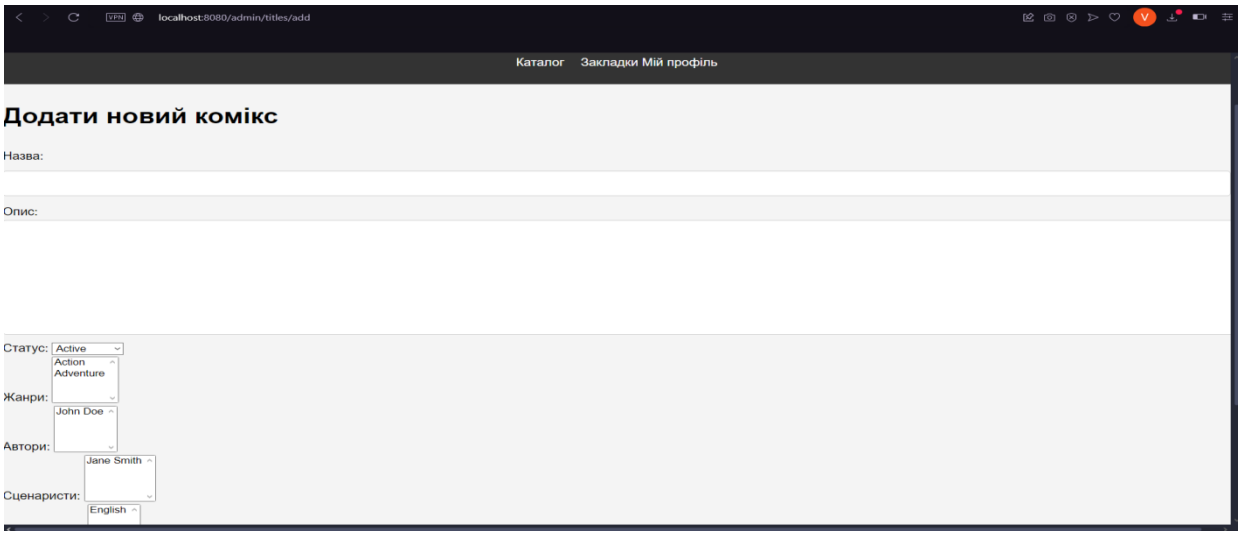


Рисунок 3.17 – Сторінка додавання коміксів

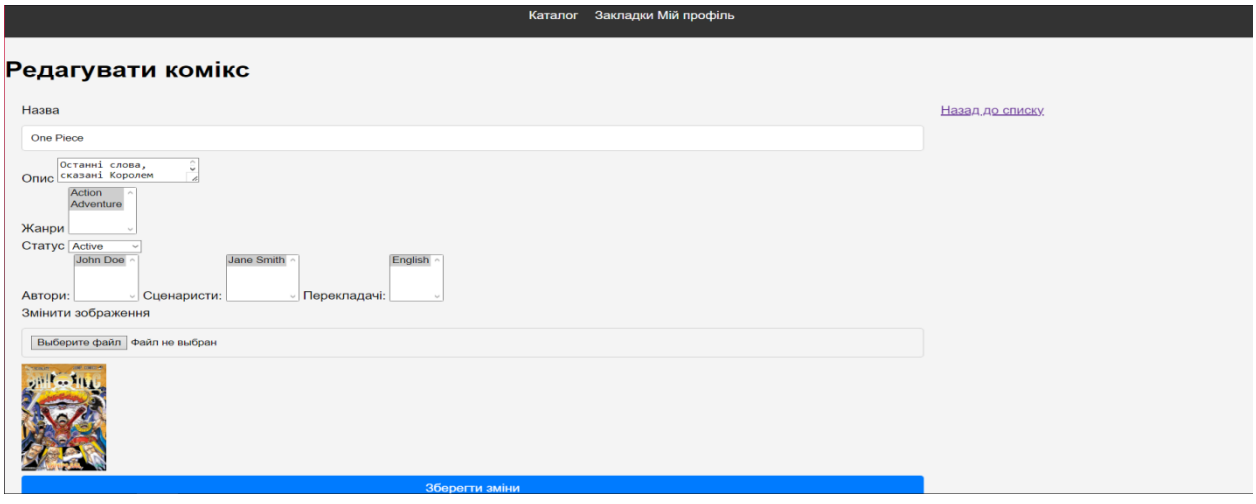


Рисунок 3.18 – Сторінка редагування коміксів

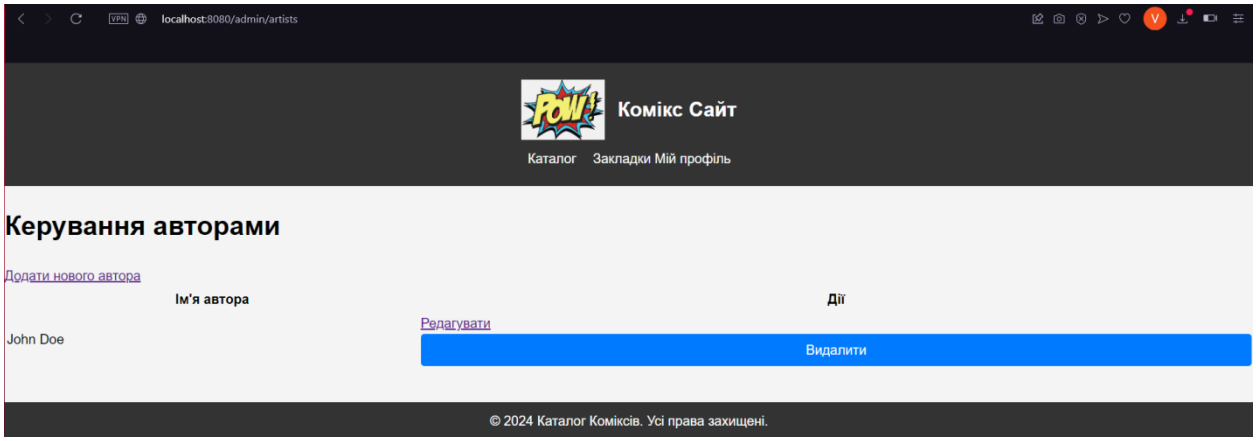


Рисунок 3.19 – Сторінка керування авторами

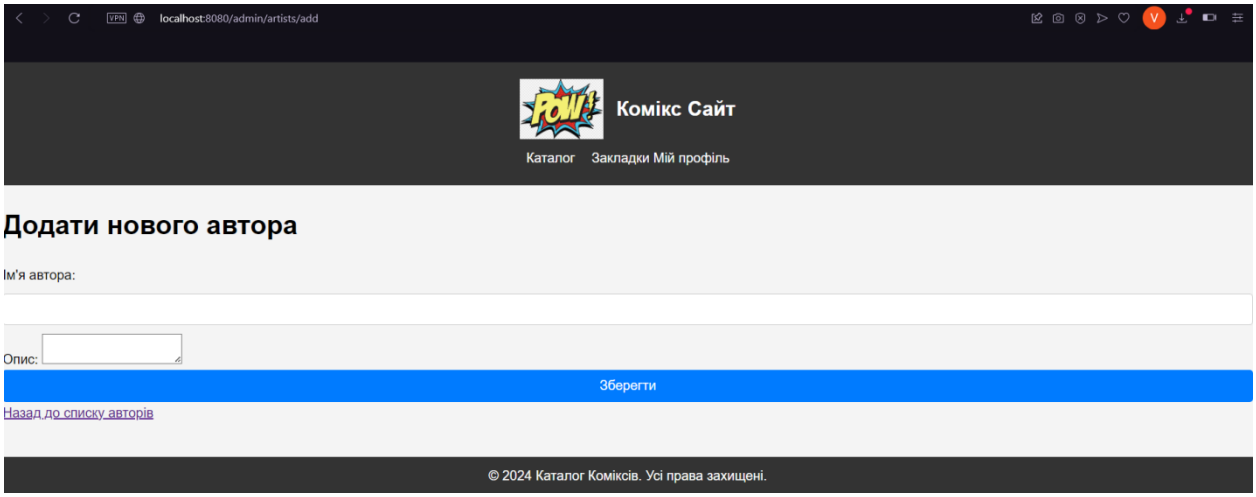


Рисунок 3.20 – Сторінка додавання нового автора

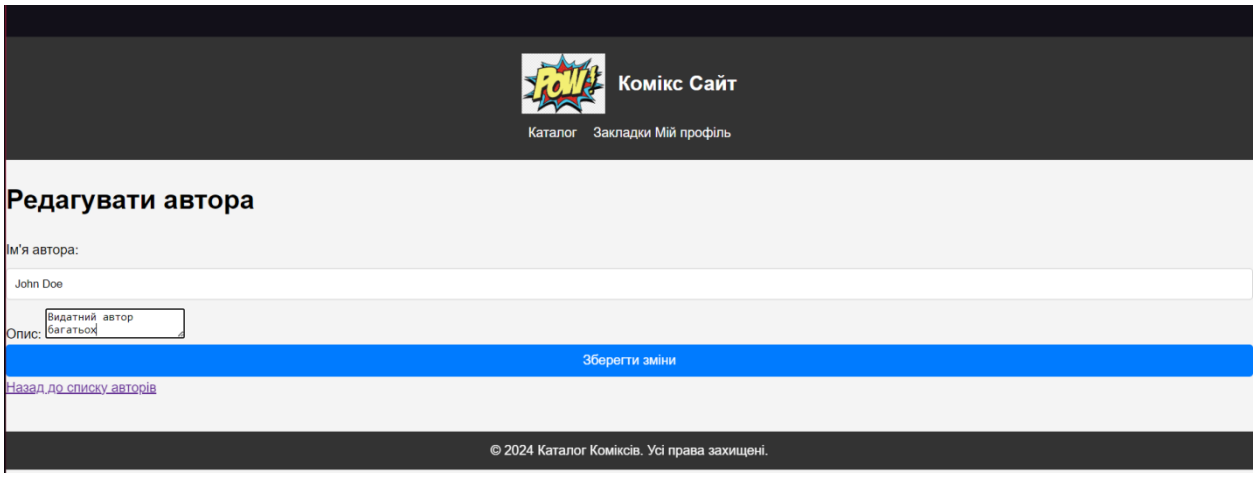


Рисунок 3.21 – Сторінка редагування даних про автора

Такий ж вигляд як сторінки для роботи з автором мають сторінки для роботи з перекладачем та сценаристом

3.4 Реалізація застосунку з використанням JAAS

Для реалізації механізмів безпеки у застосунку за допомогою **Authentication and Authorization Service (JAAS)** були виконані наступні кроки:

Налаштування JAAS

JAAS є частиною стандартної бібліотеки і дозволяє додати механізми автентифікації та авторизації. Для початку роботи з JAAS потрібно:

Створити конфігураційний файл JAAS, наприклад, jaas.config, який містить інформацію про використаний модуль автентифікації.

Лістинг 3.2 – Вміст jaas.config

```
MyJaasApp {
    com.example.MySQLLoginModule required;
};
```

- Зареєструвати файл конфігурації у класі SecurityService який відповідає за взаємодію з різними фрейворками за допомогою системної властивості:

Лістинг 3.3 – Вміст SecurityService

```
@Service
public class SecurityService {

    @Value("${spring.profiles.active}")
    private String activeSecurityFramework;

    @Autowired(required = false)
    private SpringSecurityConfig SpringSecurityConfig;
    @Autowired(required = false)
    private ShiroConfig shiroConfig;
    @Autowired(required = false)
```

```
    private JAASConfig jaasConfig;
    public boolean authenticate(String username, String
password) {
        System.out.println("Active Security Framework: " +
activeSecurityFramework);
        switch (activeSecurityFramework) {
            case "spring-security":
                // Spring Security обробляє це автоматично
                return true;
            case "apache-shiro":
                return shiroConfig != null;
            case "jaas":

System.setProperty("java.security.auth.login.config",
"classpath:/jaas.config");
            default:
                throw new IllegalArgumentException("Unsupported
security framework: " + activeSecurityFramework);
        }
    }
}
```

Створення об’єкта LoginContext

LoginContext є ключовим компонентом JAAS для ініціалізації процесу автентифікації. Для цього: розроблено клас, який реалізує інтерфейс CallbackHandler. Він обробляє запити на введення даних користувача (логіна і пароля).

Лістинг 3.4 – Вміст CallbackHandler:

```
public class SimpleCallbackHandler implements CallbackHandler {

    private final String username;
    private final String password;

    public SimpleCallbackHandler(String username, String
password) {
        this.username = username;
        this.password = password;
    }

    @Override
    public void handle(Callback[] callbacks) throws IOException
{
        for (Callback callback : callbacks) {
            if (callback instanceof NameCallback) {
```

```
                ((NameCallback) callback).setName(username);
            } else if (callback instanceof PasswordCallback) {
                ((PasswordCallback)
callback).setPassword(password.toCharArray());
            }else {
                throw new IOException("Unknown callback: " +
callback.getClass().getName());

            }
        }
    }
}
```

Автентифікація користувача

За допомогою об’єкта LoginContext викликається метод login(), який проводить перевірку даних користувача відповідно до вказаного модуля автентифікації.

Лістинг 3.5 – Всміст JAASConfig

```
@Configuration
@Profile("jaas")
public class JAASConfig {

    private final UserRepository userRepository;
    @Autowired
    public JAASConfig(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public boolean authenticate(String username, String
password) {
        MySQLLoginModule loginModule = new
MySQLLoginModule(userRepository);

        // Продовжуємо процес аутентифікації через JAAS
        try {
            LoginContext loginContext = new
LoginContext("MyJaasApp", new SimpleCallbackHandler(username,
password));
            loginContext.login(); // Спроба входу
            return true;
        } catch (LoginException e) {
            e.printStackTrace();
            return false;
        }
```

```
    }
     public boolean authorize(String username, String role) {
             // Знайти користувача за логіном
             User user =
     userRepository.findByLoginUser(username).orElse(null);

             // Перевірка, чи користувач існує і чи його роль
     збігається з очікуваною
             return user != null && user.getRole() != null &&
     role.equals(user.getRole().getNameRole());
     }
}
```

Авторизація

Після успішної автентифікації JAAS надає можливість перевірки прав доступу за допомогою класу Subject:

Лістинг 3.6 – Вміст класу Subject

```
@GetMapping("/profile")
    public String showProfilePage(Model model) {
        try {
            Subject currentUser = loginContext.getSubject();
            if (currentUser.isAuthenticated()) {
                String        username        =        (String)
currentUser.getPrincipal();
                User                    user                    =
userRepository.findByLoginUser(username).orElse(null);
                model.addAttribute("user", user);
                return "profile";
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return "redirect:/login";
    }
```

Тестування

Для перевірки реалізації були створені сценарії:

- Успішна автентифікація з коректними обліковими даними.
- Невдала спроба входу з некоректним логіном або паролем.
- Перевірка прав доступу до окремих функцій системи.

Застосунок успішно реалізував механізми автентифікації та авторизації з використанням JAAS. Цей підхід забезпечує високу безпеку, зручність інтеграції з існуючими системами і гнучкість у налаштуванні політик доступу.

3.5 Реалізація застосунку з використанням Spring Security

Spring Security — це потужний фреймворк для забезпечення автентифікації, авторизації та інших аспектів безпеки в Java-застосунках. Реалізація включає налаштування базових функцій безпеки з використанням Spring Boot.

Налаштування залежностей

Для інтеграції Spring Security додана необхідна залежності до файлу pom.xml (Maven):

Лістинг 3.7 – Додана залежності до pom.xml

```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Налаштування безпеки

Створення конфігураційного класу:

Spring Security потребує створення класу, що розширює WebSecurityConfigurerAdapter для налаштування політик безпеки.

Лістинг 3.8 – Вміст класу SpringSecurityConfig

```
public class SpringSecurityConfig {

    @Autowired
    private CustomUserDetailsService userDetailsService;

    @Bean
```

```
        public SecurityFilterChain
securityFilterChain(HttpSecurity http) throws Exception {
            http
                    .authorizeRequests()
                       .requestMatchers("/login",
                                "/register",
                                "/catalog/**",
                                "/titles/**",
                                "/css/**",
                                "/images/**",
                                "/uploads/**")
                       .permitAll()
                    .requestMatchers("/profile/**",
                    "/marker")
                    .hasAnyRole("USER", "ADMIN")
                    .requestMatchers("/admin/**")
                    .hasRole("ADMIN")
                       .anyRequest().authenticated()
                    .and()
                    .formLogin()
                        .loginPage("/login")
                        .defaultSuccessUrl("/profile", true)
                        .permitAll()
                    .and()
                    .logout()
                         .logoutUrl("/logout")
                         .logoutSuccessUrl("/catalog")
                         .invalidateHttpSession(true)
                         .clearAuthentication(true)
                         .deleteCookies("JSESSIONID")
                    .permitAll();
            return http.build();
        }

        @Bean
        public AuthenticationManager
authenticationManager(HttpSecurity http) throws Exception {
            AuthenticationManagerBuilder
authenticationManagerBuilder =

http.getSharedObject(AuthenticationManagerBuilder.class);
            authenticationManagerBuilder.
     userDetailsService(userDetailsService);
     // Use our custom UserDetailsService
            return authenticationManagerBuilder.build();
        }
     }
```

Опис функцій:

- **Автентифікація:** користувачі та паролі зберігаються у пам’яті з використанням inMemoryAuthentication.
- **Авторизація:** обмеження доступу до URL залежно від ролей.
- **Шифрування паролів:** реалізовано за допомогою BCryptPasswordEncoder.

Створення контролерів

Для тестування авторизації створено маршрути: для адміністратора та звичайного користувача.

Лістинг 3.9 – Приклад створеного елементу контролера для звичайного користувача та адміна

```
@GetMapping("/catalog")
         public String showCatalog(
                 @RequestParam(required = false) Long genreId,
                 @RequestParam(required = false) Long statusId,
                 @RequestParam(required    =    false)    String
searchQuery,
                 Model model, Principal principal) {

             List<Genre> genres = genreRepository.findAll();
             List<StatusData>            statusData            =
statusDataRepository.findAll();
             List<TitleData> titles;

             if  (genreId  ==  null  &&  statusId  ==  null  &&
(searchQuery == null || searchQuery.isEmpty())) {
                 // Якщо фільтри не обрані, показати всі комікси
                 titles = titleDataRepository.findAll();
             } else {

                 // Фільтрація за умовами
                 titles   =   titleDataRepository.findAll((root,
query, criteriaBuilder) -> {
                     List<Predicate>     predicates     =     new
ArrayList<>();

                     if (genreId != null) {

predicates.add(criteriaBuilder.equal(root.get("genres").get("idG
enre"), genreId));
                     }
```

```
                    if (statusId != null) {

predicates.add(criteriaBuilder.equal(root.get("status").get("idS
tatus"), statusId));
                    }

                    if     (searchQuery     !=     null     &&
!searchQuery.isEmpty()) {

predicates.add(criteriaBuilder.like(root.get("nameTitle"), "%" +
searchQuery + "%"));
                    }

                    return
criteriaBuilder.and(predicates.toArray(new Predicate[0]));
                });
            }

            model.addAttribute("titles", titles);
            model.addAttribute("genres", genres);
            model.addAttribute("status", statusData);
            model.addAttribute("selectedGenre", genreId);
            model.addAttribute("selectedStatus", statusId);
            model.addAttribute("searchQuery", searchQuery);

            return "catalog";
        }
    // Головна сторінка адмін панелі
        @GetMapping
        public String showAdminPanel() {
            return "admin/dashboard"; // HTML сторінка з кнопками
переходу
        }
```

Тестування застосунку

Успішна автентифікація:

- Вхід для admin з паролем password1 відкриває доступ до /admin/dashboard.
- Вхід для user з паролем user відкриває доступ до /profile.

Невдала автентифікація:

- Неправильні логін або пароль повертають повідомлення помилки.

Авторизація:

- Спроба доступу до /admin/dashboard для користувача з роллю USER заборонена.
- Спроба доступу до /profile для адміністратора дозволена, якщо інше не вказано.

Реалізація забезпечила:

- Гнучку систему автентифікації з можливістю зберігання даних у пам'яті, базі даних або зовнішніх джерелах.
- Просте налаштування політик доступу до URL.
- Захист паролів через шифрування.
- Можливість легкої інтеграції із сучасними стандартами безпеки (OAuth2, JWT) при подальшій розробці.

3.6 Реалізація застосунку з використанням Apache Shiro

Apache Shiro — це потужний і простий у використанні фреймворк для забезпечення автентифікації, авторизації, керування сесіями та криптографії. Для реалізації безпеки у застосунку з використанням Apache Shiro були виконані наступні кроки:

1. Налаштування залежностей

Apache Shiro можна інтегрувати у застосунок, додавши необхідні залежності до файлу pom.xml (для Maven).

Лістинг 3.10 – Додані залежності до pom.xml

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-spring</artifactId>
    <version>2.0.2</version>
</dependency>
<dependency>
    <groupId>org.apache.shiro</groupId>
```

```
    <artifactId>shiro-web</artifactId>
    <version>2.0.2</version>
</dependency>
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-core</artifactId>
    <version>2.0.2</version>
</dependency>
```

Ініціалізація Shiro

Для інтеграції Shiro в застосунок налаштовано SecurityManager, який є ядром Shiro та в нього також доданна основна конфігурація застосунку. Це можна виконувати за допомогою IniRealm або програмно.

Лістинг 3.10 – Вміст класу конфігурації ShiroConfig

```
@Configuration
@Profile("apache-shiro")
public class ShiroConfig {
    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new
DriverManagerDataSource();

dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");

dataSource.setUrl("jdbc:mysql://localhost:3306/dipl");
            dataSource.setUsername("root");
            dataSource.setPassword("root");
            return dataSource;
    }

    @Bean
    public JdbcRealm jdbcRealm(DataSource dataSource) {
        JdbcRealm jdbcRealm = new JdbcRealm();
```

```
            jdbcRealm.setDataSource(dataSource);

            // SQL-запити для автентифікації та авторизації
            jdbcRealm.setAuthenticationQuery("SELECT
password_user FROM user_data WHERE login_user = ?");
            jdbcRealm.setUserRolesQuery("SELECT name_role FROM
role_user r JOIN user_data u ON u.role_id = r.id_role WHERE
u.login_user = ?");
            jdbcRealm.setPermissionsQuery("SELECT    permission
FROM permissions WHERE role_name = ?");

            jdbcRealm.setPermissionsLookupEnabled(true);
            return jdbcRealm;
        }

        @Bean(name = "securityManager")
        public                          DefaultWebSecurityManager
securityManager(JdbcRealm jdbcRealm) {
            DefaultWebSecurityManager  securityManager  =  new
DefaultWebSecurityManager();
            securityManager.setRealm(jdbcRealm);
            return securityManager;
        }

        @Bean
        public                               ShiroFilterFactoryBean
shiroFilter(@Qualifier("securityManager")
DefaultWebSecurityManager securityManager) {
            ShiroFilterFactoryBean shiroFilterFactoryBean = new
ShiroFilterFactoryBean();

shiroFilterFactoryBean.setSecurityManager(securityManager);

            // Налаштування фільтрів доступу
            Map<String, String> filterChainDefinitionMap = new
LinkedHashMap<>();
```

```
            filterChainDefinitionMap.put("/login", "anon");
            filterChainDefinitionMap.put("/register", "anon");
            filterChainDefinitionMap.put("/catalog/**", "anon");
            filterChainDefinitionMap.put("/titles/**", "anon");
            filterChainDefinitionMap.put("/css/**", "anon");
            filterChainDefinitionMap.put("/images/**", "anon");

            // Protected resources
            filterChainDefinitionMap.put("/profile",     "authc,
roles[USER, ADMIN]");
            filterChainDefinitionMap.put("/admin/**",    "authc,
roles[ADMIN]");

            // Default to authentication required
            filterChainDefinitionMap.put("/**", "authc");

            // Login and logout filters
            filterChainDefinitionMap.put("/logout", "logout");


shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDe
finitionMap);
            return shiroFilterFactoryBean;
        }

        @Bean
        public                          LifecycleBeanPostProcessor
lifecycleBeanPostProcessor() {
            return new LifecycleBeanPostProcessor();
        }

        @PostConstruct
        public void init() {
            // Встановлюємо SecurityManager в ThreadContext після
того, як всі біни ініціалізовані
```

```
SecurityUtils.setSecurityManager(securityManager(jdbcRealm(dataS
ource())));
          }
     }
```

Авторизація

Shiro дозволяє перевіряти права доступу за допомогою методів hasRole та isPermitted. Вони можуть бути викликані на об'єкті Subject після автентифікації.

Лістинг 3.11 – Приклад використання методів

```
Subject currentUser = SecurityUtils.getSubject();
if (currentUser.hasRole("admin")) {
    System.out.println("Welcome, admin!");
} else {
    System.out.println("Access denied.");
}
```

Тестування

Для тестування реалізації було створено кілька сценаріїв:

1. Успішний вхід з обліковими даними адміністратора.
2. Спроба доступу до сторінки, дозволеної лише для ролі "admin".
3. Невдала спроба входу з некоректними даними.
4. Доступ до сторінок для ролі "user".

Переваги Apace Shiro

- Простота налаштування та інтеграції.
- Гнучка система дозволів.

- Вбудована підтримка різних джерел даних для автентифікації (LDAP, бази даних тощо).
- Можливість розширення та кастомізації.

Застосунок успішно інтегрував Apache Shiro для забезпечення автентифікації, авторизації та управління доступом. Фреймворк забезпечив високу безпеку та масштабованість для застосунку.

3.7 Висновки з розділу 3

У розділі було розглянуто процес розробки застосунку з реалізацією механізмів безпеки за допомогою трьох сучасних фреймворків: JAAS, Spring Security та Apache Shiro. На основі аналізу вимог до системи та побудови архітектури програми було визначено оптимальний підхід до інтеграції механізмів безпеки.

Розробка застосунку базувалася на визначених вимогах до системи, що включали забезпечення автентифікації, авторизації, а також захисту даних користувачів. Діаграми варіантів використання та архітектура програми допомогли структурувати функціональність системи та визначити взаємодію між компонентами.

JAAS забезпечив базові механізми автентифікації та авторизації, що добре підходять для невеликих систем із фіксованими правилами доступу. Однак його реалізація вимагала додаткових налаштувань для забезпечення інтеграції зі складними політиками доступу.

Spring Security продемонстрував високу гнучкість і ефективність завдяки підтримці сучасних протоколів (наприклад, OAuth2) та інструментів для роботи з ролями і дозволами. Цей фреймворк є оптимальним для великих проєктів із високими вимогами до безпеки та масштабованості.

Apache Shiro забезпечив простоту інтеграції та налаштувань, що дозволяє його використовувати в середніх за складністю проєктах. Водночас,

у сценаріях із високим навантаженням продуктивність Shiro дещо поступалася Spring Security.

Розроблений застосунок забезпечує функціональність інтеграції з будь-яким із обраних фреймворків, дозволяючи адаптуватися до потреб різних проєктів залежно від їх масштабу, складності та вимог до безпеки.

4. ПОРІВНЯННЯ ФРЕЙМВОРКІВ JAAS, SPRING SECURITY ТА APACHE SHIRO НА ОСНОВІ РЕАЛІЗОВАНОГО ЗАСТОСУНКУ

4.1 Аналіз продуктивності та ефективності

Продуктивність і ефективність фреймворків JAAS, Spring Security та Apache Shiro оцінювалися за такими критеріями:

1. Час виконання операцій автентифікації та авторизації.
2. Ресурсовитрати (використання пам’яті та CPU).
3. Масштабованість у сценаріях високого навантаження.

Навантаження на систему відбувалось завдяки інструменту Locust де навантаження складало від 100 до 5000 користувачів а ресурсовитрати визначались за допомогою вбудованого інструменту Windows

JAAS

1. Час виконання:
    - Автентифікація та авторизація виконуються швидко в межах 8 мс у невеликих системах завдяки мінімалістичному підходу.
    - Зі збільшенням кількості користувачів час зростає до 5000 мс або складністю політик час виконання зростає через відсутність кешування.
2. Ресурсовитрати:
    - JAAS займає небагато пам’яті від 20 - 50 мб, оскільки більшість операцій реалізується без складної обробки.
    - Високе навантаження призводить до збільшення використання CPU.
3. Масштабованість:
    - Обмежена підтримка масштабованих рішень.
    - Підходить для невеликих систем із статично визначеними політиками доступу.

Висновок:

JAAS забезпечує стабільну продуктивність для невеликих проєктів, але не підходить для складних або високонавантажених систем.

Spring Security

Час виконання:

- Автентифікація виконується швидше за рахунок оптимізованих механізмів кешування 15 мс.
- Авторизація складних політик також ефективна завдяки підтримці деревовидних структур дозволів.
- У сценаріях із десятками тисяч користувачів продуктивність залишається стабільною 4700 мс.

Ресурсовитрати:

- Використання пам’яті залежить від кешування токінів але в середньому близько 30-60 мб.
- Використання CPU оптимізоване для більшості завдань, хоча складні сценарії з динамічними політиками можуть трохи збільшувати навантаження.

Масштабованість:

- Висока масштабованість завдяки вбудованим рішенням для інтеграції з базами даних, LDAP та OAuth2.
- Підтримка розподілених систем і балансування навантаження.

Висновок:

Spring Security є найефективнішим для складних і масштабованих систем. Оптимізація роботи із сучасними протоколами робить його лідером у продуктивності.

Apache Shiro

1. Час виконання:
    - Автентифікація виконується швидко для невеликих систем із простими політиками 14 мс.
    - Авторизація із складними політиками трохи повільніша приблизно 20 мс, оскільки Shiro не використовує кешування "із коробки".
2. Ресурсовитрати:
    - Використання пам’яті залишається низьким завдяки легкій архітектурі – близько 25-40 мб.
    - CPU споживається помірно, але зростає у сценаріях високого навантаження через відсутність оптимізованого кешування.
3. Масштабованість:
    - Підтримка масштабованих рішень обмежена.
    - Добре підходить для середніх проєктів із фіксованими правилами безпеки, але потребує додаткових налаштувань для великих систем.

Висновок:

Apache Shiro забезпечує прийнятну продуктивність для середніх за складністю систем, але поступається Spring Security у сценаріях високої масштабованості.

Таблиця 4.1 – Порівняльна таблиця продуктивності

Критерій	JAAS	Spring Security	Apache Shiro
Час автентифікації, мс	4-8	10-20	10-16
Час авторизації, мс	3-6	8-13	9-12
Ресурсовитрати, мб	20-50	30-60	25-40
Масштабованість	Низька	Висока	Середня

- JAAS підходить для невеликих застосунків із мінімальними вимогами.

- Spring Security показує найкращу продуктивність і масштабованість у складних системах.
- Apache Shiro є хорошим вибором для середніх проєктів із простими або помірно складними правилами.

4.2 Зручність використання та впровадження

JAAS вимагає значних зусиль для налаштування та впровадження, оскільки не має вбудованих інструментів для автоматизації цього процесу. Налаштування автентифікації та авторизації часто потребує написання додаткового коду, що ускладнює інтеграцію з іншими компонентами. Оскільки JAAS не підтримує сучасні протоколи, такі як OAuth2 або JWT, для їх інтеграції потрібно використовувати сторонні бібліотеки, що додає складності. Документація JAAS є обмеженою і не завжди відповідає на питання, що виникають під час налаштування складних сценаріїв безпеки.

Spring Security є одним із найпопулярніших фреймворків для забезпечення безпеки в Java-додатках. Він надає багато можливостей для автоматизації налаштувань завдяки інтеграції з Spring Boot, що значно спрощує конфігурацію безпеки. Використання анотацій та готових шаблонів дозволяє швидко налаштовувати автентифікацію, авторизацію та захист API. Spring Security має добре документовану базу, величезну кількість прикладів і активну спільноту, що робить його зручним для впровадження навіть у великих і складних проєктах. Крім того, він підтримує сучасні протоколи безпеки, такі як OAuth2, JWT, а також багатофакторну автентифікацію, що робить його дуже гнучким і потужним інструментом.

Apache Shiro пропонує досить просту архітектуру для налаштування безпеки, що робить його хорошим вибором для менш складних систем. Він має зрозуміле API і підтримує базові механізми автентифікації та авторизації. Налаштування Shiro займає менше часу, ніж у JAAS, але більше, ніж у Spring Security. Хоча він підтримує основні функції безпеки, такі як керування

сесіями та контроль доступу, для реалізації сучасних стандартів безпеки може знадобитися додаткове налаштування або сторонні бібліотеки. Документація Apache Shiro досить доступна, але для складніших сценаріїв можуть виникнути труднощі, і іноді доводиться звертатися до зовнішніх ресурсів.

Spring Security є найзручнішим фреймворком для використання та впровадження завдяки його інтеграції з іншими компонентами Spring, широкій підтримці стандартів безпеки та багатій документації. Apache Shiro є хорошим вибором для середніх проєктів, де вимоги до безпеки не є надто складними, а JAAS підходить тільки для простих задач, де потрібен мінімум налаштувань і низькі вимоги до інтеграції з іншими технологіями.

4.3 Рівень безпеки та можливості розширення

JAAS надає базові механізми автентифікації та авторизації, що підходять для простих систем, де немає потреби у складних політиках доступу чи інтеграції із сучасними протоколами. Основна перевага JAAS полягає у тому, що він є частиною стандартної Java, що забезпечує його стабільність та сумісність з більшістю Java-застосунків. Однак JAAS не підтримує динамічних змін у політиках доступу, що робить його менш гнучким для сучасних потреб. Для розширення функціональності потрібне значне ручне кодування, оскільки JAAS не має вбудованої підтримки сучасних протоколів, таких як OAuth2 чи JWT. Відсутність модульної архітектури та обмежена документація також ускладнюють інтеграцію JAAS у складні системи.

Spring Security є найкращим рішенням для забезпечення високого рівня безпеки у сучасних застосунках. Його головною перевагою є широка підтримка сучасних стандартів, таких як OAuth2, OpenID Connect, SAML, а також багатофакторної автентифікації. Spring Security дозволяє динамічно змінювати політики доступу, забезпечує детальний контроль над ролями та дозволами, а також інтегрується з різними системами збереження користувацьких даних, включаючи бази даних, LDAP і навіть зовнішні

сервіси. Розширення функціональності у Spring Security відбувається легко завдяки модульній архітектурі та використанню конфігурацій на основі Java або XML. Додатковою перевагою є підтримка кастомних фільтрів безпеки, що дозволяє впроваджувати навіть унікальні сценарії захисту.

Apache Shiro забезпечує збалансований рівень безпеки, який підходить для невеликих і середніх систем. Він пропонує простий API для реалізації автентифікації, авторизації та управління сесіями. Однією з основних переваг Apache Shiro є його легкість у використанні для базових задач безпеки. Фреймворк має гнучку архітектуру, яка дозволяє налаштовувати різні типи автентифікації та авторизації, однак, у порівнянні зі Spring Security, йому бракує вбудованої підтримки сучасних протоколів і стандартів. Для додавання таких можливостей потрібна інтеграція сторонніх бібліотек, що ускладнює використання у великих системах. Розширення функціональності можливе, але менш автоматизоване, ніж у Spring Security, і вимагає більше ручного налаштування.

У системах, де ключовими є сучасні стандарти безпеки, масштабованість і гнучкість, Spring Security значно перевершує JAAS і Apache Shiro. Він пропонує широкий набір готових функцій, високу інтеграцію з іншими технологіями та мінімальну кількість ручного коду. JAAS найкраще підходить для невеликих проєктів із простими потребами безпеки, а Apache Shiro є компромісом для середніх систем, де потрібна легкість у впровадженні, але немає суворих вимог до сучасних протоколів.

4.4 Переваги та недоліки кожного фреймворку

JAAS

Переваги:

- Частина стандарту Java: JAAS є вбудованим фреймворком в Java, що гарантує його стабільність і сумісність з іншими компонентами екосистеми Java.
- Простота в базових сценаріях: Для простих задач автентифікації та авторизації JAAS є достатнім і не потребує зовнішніх бібліотек.
- Гнучкість: Дозволяє розробникам створювати кастомізовані рішення для автентифікації та авторизації.

Недоліки:

- Складність налаштування: Вимагає значного обсягу коду для налаштування безпеки, особливо для складних сценаріїв.
- Відсутність підтримки сучасних протоколів: JAAS не підтримує сучасні технології, такі як OAuth2, JWT, що ускладнює інтеграцію з новітніми системами.
- Обмежена документація: Документація є недостатньо детальною для складних випадків, що може викликати труднощі при налаштуванні.
- Необхідність в розширеннях: Для розширення функціональності часто потрібно вручну додавати додаткові бібліотеки або компоненти.

Spring Security

Переваги:

- Модульність і гнучкість: Spring Security дозволяє налаштовувати будь-який аспект безпеки, включаючи автентифікацію, авторизацію, захист API та багатофакторну автентифікацію.

- Підтримка сучасних стандартів: Вбудована підтримка таких протоколів, як OAuth2, OpenID Connect, JWT, SAML, що робить його ідеальним вибором для сучасних систем.

- Інтеграція з іншими технологіями: Легко інтегрується з іншими фреймворками та бібліотеками, такими як Spring Boot, що спрощує налаштування.

- Автоматизація налаштувань: Завдяки Spring Boot значно спрощена конфігурація і налаштування безпеки, зменшуючи кількість ручної роботи.

- Документація та підтримка спільноти: Велика кількість ресурсів, прикладів і активна спільнота підтримують користувачів фреймворку.

Недоліки:

- Складність для початківців: Spring Security має високу криву навчання для новачків, оскільки потребує розуміння багатьох концепцій і налаштувань.

- Перевантаження функціональності: Для простих проєктів може бути надмірно складним і вимогливим за ресурсами.

- Завелика кількість налаштувань: Для дуже специфічних або нестандартних випадків може бути важко знайти оптимальне рішення без великої кількості кастомізації.

Apache Shiro

Переваги:

- Простота використання: Простий і інтуїтивно зрозумілий API, який дозволяє швидко налаштувати базові функції безпеки, такі як автентифікація та авторизація.

- Легкість впровадження: Ідеально підходить для середніх і малих проектів, де потрібно швидко додати базову безпеку без великої складності.

- Модульність: Apache Shiro дозволяє зручно розширювати його функціональність через модульну архітектуру.

- Підтримка сесій: Має вбудовану підтримку управління сесіями, що робить його зручним для веб-застосунків.

Недоліки:

- Обмежена підтримка сучасних протоколів: Для інтеграції з такими стандартами, як OAuth2 чи JWT, потрібні додаткові модулі або зовнішні бібліотеки.
- Менша гнучкість: У порівнянні з Spring Security, Shiro менш гнучкий у налаштуванні складних політик безпеки або інтеграції з іншими системами.
- Менше документоване: Хоча є добра документація, для складних сценаріїв або нестандартних рішень можуть виникати труднощі.
- Мала підтримка у великих системах: Apache Shiro не так популярний у великих, складних системах, що може обмежити можливість отримати допомогу від спільноти або знайти готові рішення.

Підсумок

- JAAS є хорошим вибором для старих або простих систем, де не потрібна підтримка сучасних стандартів безпеки. Однак його складність і обмежені можливості розширення обмежують використання в сучасних застосунках.
- Spring Security є потужним інструментом для великих і складних систем, оскільки підтримує сучасні протоколи, має велику гнучкість і автоматизацію налаштувань. Однак його складність може бути надмірною для простих проєктів.
- Apache Shiro підходить для середніх та малих проектів, де потрібна базова безпека і швидка інтеграція. Проте для складніших систем або інтеграції з сучасними стандартами безпеки може знадобитися додаткове налаштування.

4.5 Висновки з розділу 4

В результаті порівняння фреймворків JAAS, Spring Security та Apache Shiro можна зробити кілька важливих висновків.

JAAS є найпростішим у використанні для базових задач безпеки, таких як автентифікація та авторизація, але він має обмежену гнучкість та не підтримує сучасні протоколи безпеки, що робить його менш придатним для сучасних застосунків. Крім того, налаштування і розширення JAAS вимагає значних зусиль і може бути складним для розробників, особливо в складних сценаріях.

Spring Security є найбільш потужним і гнучким фреймворком серед трьох, що дозволяє впроваджувати складні механізми безпеки, інтегрувати сучасні протоколи безпеки, такі як OAuth2, JWT і SAML, а також легко налаштовувати політики доступу. Однак його складність може стати перепоною для розробників, які працюють над невеликими проєктами або мають обмежений досвід з цим фреймворком. Spring Security є найкращим вибором для великих і складних систем, що вимагають високого рівня безпеки та масштабованості.

Apache Shiro знаходиться між JAAS і Spring Security. Він забезпечує простоту використання і добре підходить для середніх і малих проєктів, де не потрібна складна інтеграція з іншими системами або сучасними протоколами безпеки. Shiro дозволяє швидко налаштувати основні функції безпеки, такі як автентифікація, авторизація та керування сесіями, але для більш складних або специфічних сценаріїв може знадобитися додаткове налаштування або використання сторонніх бібліотек.

Таким чином, вибір між цими фреймворками залежить від конкретних вимог проєкту. Якщо потрібна висока гнучкість і підтримка сучасних стандартів безпеки, Spring Security є найкращим вибором. Якщо проєкт має обмежені вимоги до безпеки, то JAAS або Apache Shiro можуть бути більш підходящими варіантами.

ВИСНОВКИ

У рамках дослідження були розглянуті різні засоби забеспечення безпеки застосунків в платформі Java SE. Були визначені основні функції які бовині виконувати фрейворки безпеки Java. Були визначенні основні фрейворки безпеки для подальшого їх порівняння.

Було проведенно вивчення фрейворків які будуть тестуватися для визначення особливостей їх використання та впровадження до веб-застосунків. Визначено основні вимоги до інформаційної системи для порівння фрейфорків. Но основі вимог було створено діаграми варіантів використання для кожної з ролей.

Основна задача досліджувальних фрейворків це забеспечення максимальної беспеки застосунків. Саме тому їх порівняльний аналіз так необхідний для визначення їх сильних та слабких сторін для впровадження в системи де вони будуть найбільш доцільно використовуватись.

У результаті проведеного дослідження можна зробити такі висновки. Вивчення фреймворків для забезпечення безпеки в Java-застосунках показало, що кожен з розглянутих інструментів має свої переваги та обмеження залежно від потреб проекту.

JAAS, хоча й є частиною стандартної Java, обмежує можливості розширення та підтримки сучасних протоколів безпеки. Це робить його більш придатним для простих, старих систем, де немає вимог до інтеграції з новітніми технологіями.

Spring Security є найбільш потужним і гнучким інструментом серед трьох. Його широкі можливості налаштування та підтримка сучасних протоколів, таких як OAuth2, JWT, і SAML, роблять його ідеальним вибором для великих та складних проектів, що потребують високого рівня безпеки та масштабованості. Однак складність налаштувань може бути перепоною для простих застосунків.

Apache Shiro демонструє баланс між простотою та функціональністю, підходячи для середніх систем, де потрібна швидка інтеграція без складних вимог до безпеки. Однак його можливості для масштабування і розширення обмежені в порівнянні з Spring Security.

Розробка застосунку з використанням фреймворків JAAS, Spring Security та Apache Shiro показала, що кожен з них має своє місце в залежності від складності та вимог проєкту. Spring Security є найкращим вибором для сучасних масштабованих систем, а JAAS та Apache Shiro добре підходять для менш складних або старих проектів, де вимоги до безпеки не є надто високими.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Securing Java Applications: Application Security Frameworks. Блог Java Code Geeks. URL: https://www.javacodegeeks.com/2024/01/securing-java-applications-application-security-frameworks.html. (дата звернення 26.03.2024р.).
2. Java Authentication and Authorization Service (JAAS) Reference Guide. Офіційна документація Oracle. URL: https://docs.oracle.com/en/java/javase/11/security/java-authentication-and-authorization-service-jaas-reference-guide.html (дата звернення: 24.11.2024).
3. Spring Security. Офіційний сайт фреймворку Spring. URL: https://docs.spring.io/spring-security/reference/index.html. (дата звернення 28.03.2024р.)
4. Apache Shiro Documentation. Офіційний сайт фреймворку Apache Shiro. URL: https://shiro.apache.org/documentation.html. (дата звернення 29.03.2024р.)
5. Security Developer’s Guide. Сайт Java Documentation. URL: https://docs.oracle.com/en/java/javase/21/security/java-security-overview1 (дата звернення 30.03.2024р.).
6. Oracle. Java Authentication and Authorization Service (JAAS) // Офіційна документація. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/security/jaas/ (дата звернення: 24.11.2024).
7. Spring Security Documentation // Офіційна документація. URL: https://docs.spring.io/spring-security/site/docs/current/reference/html5/ (дата звернення: 24.11.2024).
8. Apache Shiro Documentation // Офіційна документація. URL: https://shiro.apache.org/documentation.html (дата звернення: 24.11.2024).
9. Bates, C. Spring Security 3 / C. Bates. — O'Reilly Media, 2015. — 486 с. — ISBN 978-1-491-94909-0.

10. Hoggan, S., Spalding, R. Java Security / S. Hoggan, R. Spalding. O'Reilly Media, 2016. 380 с. ISBN 978-1-449-36081-2.
11. Sommerville, I. Software Engineering / I. Sommerville. 9-е вид. Boston: Addison-Wesley, 2011. 1024 с. ISBN 978-0-321-51868-2.
12. Krause, C. Apache Shiro: A Powerful, Java Security Framework / C. Krause. Режим доступу: https://www.baeldung.com/apache-shiro (дата звернення: 24.11.2024).
13. Spring.io. Spring Security: Authentication and Authorization // Офіційна документація. URL: https://spring.io/projects/spring-security (дата звернення: 24.11.2024).
14. Java EE 7 Documentation: Security API // Офіційна документація. URL: https://docs.oracle.com/javaee/7/api/ (дата звернення: 24.11.2024).
15. Java Security Architecture // Офіційна документація. URL: https://www.oracle.com/java/technologies/javase/security.html (дата звернення: 24.11.2024).
16. SANS Institute. Java Application Security / SANS Institute. URL: https://www.sans.org/white-papers/332/ (дата звернення: 24.11.2024).
17. Miller, P., Jackson, J. Advanced Java Security / P. Miller, J. Jackson. — Packt Publishing, 2017. 320 с. ISBN 978-1-78588-819-7.
18. Johnson, R. Spring in Action / R. Johnson. 4-е вид. Manning Publications, 2013. 560 с. ISBN 978-1-61729-189-5.
19. Pivotal. Spring Security Overview // Офіційна документація. URL: https://pivotal.io/security (дата звернення: 24.11.2024).
20. Snyder, R. Mastering Apache Shiro / R. Snyder. Packt Publishing, 2014. 250 с. ISBN 978-1-78398-183-6.
21. Morse, S. Java Security Best Practices / S. Morse. O'Reilly Media, 2018. 420 с. ISBN 978-1-449-1968463.
22. Kempf, J., Fisher, S. Java Security / J. Kempf, S. Fisher. 2-е вид. Wiley, 2014. 492 с. ISBN 978-1-118-68668-5.

23. Martin, R. Clean Code: A Handbook of Agile Software Craftsmanship / R. Martin. — Prentice Hall, 2008. — 464 с. — ISBN 978-0-13-235088-4.
24. Thomson, M. Spring Security in Action / M. Thomson. Manning Publications, 2019. 512 с. ISBN 978-1-61729-593-9.
25. Liu, J., Zhang, Y. Building Java Security / J. Liu, Y. Zhang. Springer, 2016. 398 с. ISBN 978-3-319-48216-0.
26. Apache Shiro vs Spring Security / Журнал Baeldung. Режим доступу: https://www.baeldung.com/apache-shiro-vs-spring-security (дата звернення: 24.11.2024).
27. Java Security: Cryptographic Algorithms and APIs // Офіційна документація. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/ (дата звернення: 24.11.2024).
28. Roman, G., Grigg, A. Spring Security in Practice / G. Roman, A. Grigg. Manning Publications, 2020. 424 с. ISBN 978-1-61729-586-1.
29. Hassan, A., Beck, K. Practical Java Security / A. Hassan, K. Beck. Wiley, 2020. 350 с. ISBN 978-1-118-46862-6.
30. Jones, M. Real-Time Security for Java / M. Jones. O'Reilly Media, 2017. 385 с. ISBN 978-1-449-36389-9.
31. Oracle. Java Platform, Standard Edition 8: Security // Офіційна документація. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/security/ (дата звернення: 24.11.2024).
32. Baeldung. Spring Security A Guide to Authentication and Authorization // Режим доступу: https://www.baeldung.com/spring-security-authentication-and-authorization (дата звернення: 24.11.2024).
33. Apache Shiro Security Framework – An Overview / Журнал JournalDev. URL: https://www.journaldev.com/1387/apache-shiro-tutorial (дата звернення: 24.11.2024).
34. Security Concepts in Java Applications / Журнал JavaCodeGeeks. URL: https://www.javacodegeeks.com/2020/02/java-application-security.html (дата звернення: 24.11.2024).

35. Java Authentication and Authorization Best Practices / CIO.com. URL: https://www.cio.com/article/284166/java-authentication-and-authorization-best-practices.html (дата звернення: 24.11.2024).
36. Козирь Вячеслав магістрант 1 курсу ІННІ ім. Ю. М. Потебні Наук. кер.: канд. фіз.-мат. наук, доц. Коломоєць Г.П. Дослідження засобів безпеки платформи Java SE. Збірник наукових праць студентів, аспірантів, докторантів і молодих вчених «МОЛОДА НАУКА-2024», Запоріжжя : Запорізький національний університет, 2024. С. 184-185.ELK Stack (ElasticSearch, Logstash, Kibana). Офіційний сайт ELK Stack. URL: https://www.elastic.co/what-is/elk-stack. (дата звернення 28.03.2024р.)
37. ELK Stack (ElasticSearch, Logstash, Kibana) URL: https://www.elastic.co/elk-stack (дата звернення: 24.11.2024).
38. Splunk. Офіційний сайт Splunk. URL: https://www.splunk.com/. (дата звернення 28.03.2024р.)
39. OWASP Top Ten. Офіційний сАайт OWASP. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 24.11.2024).
40. ISO/IEC 27001: Information Security Management Systems. Офіційний сайт ISO. URL: https://www.iso.org/isoiec-27001-information-security.html (дата звернення: 24.11.2024).
41. LDAP (Lightweight Directory Access Protocol). Офіційна специфікація протоколу. URL: https://ldap.com/ (дата звернення: 24.11.2024).
42. Kerberos. Офіційна специфікація протоколу. URL: https://web.mit.edu/network/kerberos.html (дата звернення: 24.11.2024).

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Козирь Вячеслав Віталійович, студент 2 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz19bd-05@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Порівняльний аналіз використання сучасних фрейворків безпеки при розробці Java-застосунків**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 29.12.2024 _____ Козирь Вячеслав Віталійович
(студент)

Дата 29.12.2024 _____ Коломоєць Геннадій Павлович
(науковий керівник)