Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий інститут комп'ютерних систем
Кафедра інженерії програмного забезпечення

Волков Денис Володимирович
студент групи АС-171

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Автоматизована система для купівлі-продажу сільськогосподарської продукції

Спеціальність:
121 – Інженерія програмного забезпечення

Освітньо-професійна програма:
Інженерія програмного забезпечення

Керівник:
Роговський Вадим Томовіч
канд. техн. наук, доцент

Одеса – 2022

ЗМІСТ

Міністерство освіти і науки України

Національний університет «Одеська політехніка»

Навчально-науковий інститут комп'ютерних систем

Кафедра інженерії програмного забезпечення

Рівень вищої освіти: другий (магістерський)

Спеціальність: 121 – Інженерія програмного забезпечення

Освітня програма: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Комлева Н. О.

«____» _____2022 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Волкова Дениса Володимировича, група АС-171

1. Тема роботи: Автоматизована система для купівлі-продажу сільськогосподарської продукції.
   Керівник роботи: Роговський Вадим Томовіч, канд. техн. наук, доцент затверджені наказом ректора від «20» жовтня 2022р. № 399-в.
2. Зміст роботі: аналіз сучасних рішень, комплекс завдань веб-застосування, специфікація вимог до програмного продукту, проектування програмного продукту, програмна реалізація системи, тестування програмного продукт.
3. Перелік ілюстративного матеріалу: згідно зі слайдами презентації.

4. Консультанти розділів роботи

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

5. Дата видачі завдання: «_01__» _____09_____ 2022 ____ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	Аналіз існуючих рішень	2.09.2022 – 11.09.2022	вик.
2	Вивчення комплексу задач системи	12.09.2022 – 17.09.2022	вик.
3	Специфікація вимог до системиы	18.09.2022 –29.09.2022	вик.
4	Проектування системи	30.09.2022 – 18.10.2022	вик.
5	Програмна реалізація та функціональне тестування	19.10.2022 – 10.11.2022	вик.
6	Постановка та проведення експерименту	11.11.2022 – 27.11.2022	вик.
7	Оформлення пояснювальної записки та графічного матеріалу	28.11.2022 – 5.12.2022	вик.

Здобувач вищої освіти _____ Д. В. Волков

Керівник роботи _____ В. Т. Роговський

АНОТАЦІЯ

Метою кваліфікаційної роботи магістра є підвищення безпеки системи купівлі-продажу сільськогосподарської продукції шляхом реалізації механізмів захисту від найчастіших вразливостей веб-додатків, використовуючи платформу оркестрації контейнізованих додатків.

Розробка системи ґрунтується на міграції вже існуючого веб-додатку на мікросервісну архітектуру, яка надалі буде розгорнута у кластері Kubernetes. До технологій, які використовувалися при виконанні роботи, належать Java, PostgreSQL, Spring Cloud Kubernetes, Helm, Docker, DocuSign, Spring Data, Flyway.

В результаті кваліфікаційної роботи магістра було отримано веб-додаток, який є захищеним від більшості поширених веб-вразливостей.

Ключові слова: безпека веб-додатків, веб-уразливості, Kubernetes, Spring Security, PostgreSQL, Docker, Helm, Spring.

ABSTRACT

The purpose of the master's qualification work is to improve the security of the system for the sale and purchase of agricultural products by implementing protection mechanisms against the most common web application vulnerabilities using the containerized application orchestration platform.

The development of the system is based on the migration of an already existing web application to a microservice architecture, which will be further deployed in a Kubernetes cluster. The technologies used in the execution of the work include Java, PostgreSQL, Spring Cloud Kubernetes, Helm, Docker, DocuSign, Spring Data, Flyway.

As a result of the master's qualification work, a web application was obtained that is protected from most common web vulnerabilities.

Keywords: web application security, web vulnerabilities, Kubernetes, Spring Security, PostgreSQL, Docker, Helm, Spring.

ВСТУП

Сьогодні в умовах розвитку бізнесу і посилення конкурентної боротьби між господарюючими суб'єктами, електронні торгові майданчики надають компаніям унікальні можливості по скороченню власних витрат, поліпшенню показників збуту і підвищення конкурентоспроможності.

Теоретична значимість дослідження полягає в розширенні наукових уявлень про електронні торгових майданчиках, процедурах, які вони надають, і процесі роботи в системі електронних торгів.

Незважаючи на всі існуючі переваги веб-додатків, вони часто не дотримуються принципів безпеки, що призводить до того, що вони стають привабливою мішенню для зловмисників. Наявність критичних вразливостей дозволяє злочинцям отримати доступ до особистої інформації клієнтів, що ні за яких обставин не має статися.

Найважливішою частиною веб-додатку з продажу с/г продукції є контракти на купівлю-продаж товарів між покупцем і постачальником, а також їх особиста інформація, розголошення якої може спричинити серйозну відповідальність, до кримінальної. І це вже не кажучи про можливе вимушене зупинення додатку та фінансові збитки.

Метою кваліфікаційної роботи магістра є підвищення безпеки системи купівлі-продажу сільськогосподарської продукції шляхом реалізації механізмів захисту від найчастіших вразливостей веб-додатків, використовуючи платформу оркестрації контейнізованих додатків. Для досягнення поставленої мети кваліфікаційної роботи магістра необхідно вирішити наведені нижче завдання:

- провести аналіз конкурентів;
- визначити вимоги системи, як функціональні, так і нефункціональні;
- виявити та вивчити можливі вразливості системи, що розробляється;
- провести аналіз механізмів захисту;
- реалізувати систему та інтегрувати в ній механізми захисту;

- провести тестування розробленої системи у контексті її безпеки на наявність можливих проломів;

Кваліфікаційна робота магістра складається з шести розділів, наведених нижче:

- аналіз сучасних рішень. У цьому розділі проведено аналіз предметної області, що включає аналіз існуючої проблеми, вивчені найближчі конкуренти, а також їх переваги та недоліки;
- комплекс завдань веб-застосування. Цей розділ повністю присвячений аналізу веб-вразливостей, які притаманні великій кількості веб-додатків. Для кожної з такої вразливості наведено алгоритм чи механізм вирішення проблеми в цілому, без використання контексту певних технологій;
- специфікація вимог до програмного продукту. У цьому розділі описано функціональні та нефункціональні вимоги системи. Всі вони розглядаються в контексті безпеки системи, що розробляється. Функціональні вимоги представлені у вигляді опису основних акторів та діаграм варіантів використання;
- проектування програмного продукту. Основною метою розділу є опис архітектурних рішень, ухвалених під час розробки системи;
- програмна реалізація системи. У цьому розділі висвітлено основні технології та бібліотеки, які використовувалися при програмній реалізації;
- тестування програмного продукт. Завершальний розділ покликаний показати те, що мети роботи було досягнуто. Тестування проводилося з використанням додатків з відкритим вихідним кодом.

Звіт є результатом виконання кваліфікаційної роботи магістра. У ньому наведено такі розділи:

1. анотація;
2. вступ;
3. розділи, що показують вирішення поставлених завдань;
4. висновки;
5. додаток з лістингом коду.

1 АНАЛІЗ СУЧАСНИХ РІШЕНЬ

1.1 Аналіз проблем та предметної області

В даний час системи програмного забезпечення містять і обробляють безліч інформації, більшість з якої може вважатися конфіденційною, особливо враховуючи вимоги GDPR [1] (General Data Protection Regulations). Ця ухвала прийнята для уніфікування захисту персональних даних користувачів. Воно надає принципи, яким має відповідати програмна система. Основним з яких (хоча інші принципи теж важливі, але їх можна опустити в даному контексті) є принцип цілісності та безпеки даних. Інформація, надана користувачами, повинна зберігатись у захищеному місці, що реалізує всі необхідні механізми захисту від розголошення цих даних. Не дотримання чого може призвести до великих штрафів.

Що являє собою конфіденційна інформація? Такого типу інформації можна віднести будь-який вид даних, який користувачів вважає особистою. Це можуть бути номери телефонів, електронні пошти, паспортні дані та дані кредитних карток. Загалом кажучи, будь-яку інформацію користувача слід вважати особистою, якщо він не сказав зворотного. Ніхто, крім самого власника, не може отримувати доступ до конфіденційної інформації, змінювати її або фальсифікувати.

Система повинна припиняти будь-які неправомірні дії шляхом реалізації механізмів захисту особистої конфіденційної інформації. Будь-який витік, що дозволяє у будь-який спосіб використовувати такі дані або функціональні можливості програми, вважається вразливістю, і вона повинна бути усунена.

Відсутність поваги до заходів безпеки спричиняє велику ціну. Найчастіше під ціною розуміється втрачений дохід. Прикладом такої ситуації може бути втрата грошей з банківського рахунку або використання послуги без оплати. Але також ціною може бути і імідж програми чи компанії. Воно включає довіру користувачів - найцінніший актив будь-якої поважає себе фірми. Втрата довіри може обійтися набагато дорожче, ніж шкода від експлуатації вразливості в системі.

Зловмисники для того, щоб отримати доступ до системи для виконання неправомірних дій, вивчають її на наявність слабких місць (вразливостей), які

можна використовувати у своїх цілях. Незважаючи на всі переваги, веб-програми часто не дотримуються принципів безпеки, що робить їх привабливою мішенню для зловмисників. Критичні вразливості дозволяють злочинцям отримувати прямий доступ до конфіденційних даних користувачів, які за жодних обставин не повинні бути оприлюднені.

Основною метою, яку забезпечує безпека веб-додатків, є процес захисту даних, що зберігаються в додатку, від небажаного доступу до цих даних, а також їх зміни. Це можна досягти за допомогою правильних заходів політик безпеки. Якщо зломщики зможуть отримати доступ до системи, це може призвести до поганих наслідків.

Наслідки можуть змінюватись від намірів зловмисника. Нижче наведено приклади:

- може бути отриманий доступ до обмежених даних;
- скомпрометовані облікові записи користувачів;
- збитки репутації бренду;
- втрачений дохід від продажу;
- втрата довіри клієнтів;
- може бути встановлений шкідливий код.

У системі, що розробляється, головною частиною є контракти на купівлю-продаж товарів, що здійснюються між продавцем і покупцем. Ці контракти містять особисті дані обох сторін, та його розголошення може призвести до значної відповідальності, до кримінальної. Це не беручи до уваги зупинку системи та можливі фінансові збитки. У зв'язку із цим першочерговою метою є посилення аспектів безпеки системи.

Варто розглянути часті вразливості, властиві веб-додаткам. Їх можна розділити на дві категорії: вразливість рівня програми та рівня інфраструктури, в якому розгортається програма.

До першої категорії належать такі вразливості:

- порушення аутентифікації та авторизації;
- фіксація сеансу;

- міжсайтовий скриптинг;
- підробка міжсайтових запитів;
- ін'єкції;
- розкриття конфіденційних даних;
- використання залежностей з відомими вразливістями.

До другої категорії належать:

- зараження шкідливим кодом;
- застаріле/невиправлене програмне забезпечення;
- стандартні конфігурації;
- незахищена передача даних;
- неправильне налаштування політик доступу до ресурсів;
- відсутність механізмів захисту даних.

На сьогоднішній день представлено велику кількість методів та заходів забезпечення безпеки систем в мережі Інтернет. Але мало хто замислюється про те, що безпека спирається не лише на те, як реалізовані та протестовані механізми, що забезпечують захист системи (як приклад можна навести процес автентифікації чи авторизації користувача), але також на те, як розроблено бізнес-логіку. У ній у жодному разі не повинні бути присутніми жодні вразливості.

Загалом, безпеку веб-системи можна поділити на чотири ключові аспекти (або принципи):

- конфіденційність. Основна суть полягає в тому, що особисті дані користувачів, які вони зберігають у системі, не повинні бути розголошені за жодних обставин;
- цілісність. Дані, що зберігаються в системи, не повинні бути суперечливими і не повинні змінюватися не їх власниками;
- доступність. Запити від користувачів повинні бути опрацьовані протягом певного проміжку часу;
- незаперечність. Система має підтверджувати особу користувача. У свою чергу, користувач не може заперечувати зміни даних, які зберігаються в системі.

У цій роботі розглядається веб-додаток для купівлі-продажу сільськогосподарської продукції. Розробка програм для використання людьми в інтернеті завжди є цікавим і нетривіальним завданням. По-перше, потрібно реалізувати товар так, щоб він ніс у собі сенс, був корисний користувачам і конкурентоспроможним на ринку. Якщо про всі ці аспекти успіху програми власники/розробники ще замислюються, оскільки це прямо впливає на отримання ними прибутку, то на такий аспект, як безпека програми, дивляться в останню чергу, якщо взагалі роблять це. Це якраз і є пунктом номер два – безпека програми. Дуже часто його вважають неважливим. Усі думають, що проблеми хакерських атак обійдуть їх стороною і ніхто не зламуватиме їхню систему. Це пов'язано з тим, що ця область розробки є, напевно, найскладнішою для розуміння і розробки, внаслідок її багатогранності.

Тільки такий інтернет-ресурс як OWASP [2] надає список із 10 найчастіших веб-вразливостей, які нерідко можна зустріти в інтернет-ресурсах. Але цим список не закінчується, оскільки існує ще велика кількість вразливостей, що належать до мови програмування, на якій написано застосування, до середовища, в якому воно розгорнуто тощо.

Для початку розглянемо основну суть застосування та яке відношення до неї має її безпека.

Веб-застосування LotAgro є платформою з купівлі-продажу товарів. У ньому представлені лоти та замовлення. Достатньо проста схема, яка використовується повсюдно. Продавці створюють лоти, коли вони мають товар і вони хочуть його продати. Покупці у свою чергу купують ці товари, створюючи замовлення на лот, що цікавить.

Оскільки застосування спеціалізується з продажу сільськогосподарської продукції, те й вибір товарів для купівлі та продажу відповідний. До них відносяться різні види агрокультур (зернові, бобові, олійні, фрукти, овочі, горіхи, насіння, гриби), корми для тварин, товари тваринництва, розсади, сільгосптехніка і так далі.

Кожне замовлення має кілька різних стадій. Ці стадія, або як їх ще можна назвати – статуси, є станом замовлення в даний момент. Це зроблено для спрощення взаємодії між дійовими особами. Усього існує 5 статусів замовлення: calculation, contracting, payment, delivery, done.

У системі представлені три типи користувачів: клієнт, модератор та адміністратор. Клієнт може бути як і покупцем, так і продавцем. Кожен з них визначає свої власні функції у системі. За допомогою ролей реалізується функція поділу відповідальності користувачів у системі. Кожен із цих типів має доступ до різних частин застосування, необхідних для вирішення власних бізнес-задач. Як приклад можна розглянути процес підписання контрактів про факт купівлі-продажу товару. Цей процес представлений у вигляді одного із статусів замовлення - contracting. Продавець і покупець повинні підписати власні контракти під час одного з кроків обробки замовлення (залежно від ролі користувача, це буде контракт на купівлю або продаж товару). Після цього вони матимуть доступ лише до своїх контрактів, на відміну від модератора. Його завдання полягає в тому, щоб перевірити коректність складання контрактів і те, що покупець і продавець заповнили їх. Відповідно, модератор має доступ до обох контрактів. У цьому полягає суть поділу відповідальності ролей користувачів.

Важливою особливістю програми є те, що вона зберігає конфіденційні дані клієнтів (контактна інформація, дані про обліковий запис, компанії, адреси доставки, укладені договори), розголошення яких може призвести до наслідків як для клієнтів, так і для власників системи. Це веде до того, що програма повинна бути захищена від різних атак хакерів, що не дозволяють допустити цього.

1.2 Аналіз аналогів

Розглянемо найближчих конкурентів веб-застосувань з продажу сільгосппродукції на території України. До таких аналогів відносяться лише два майданчики: Agrobiz та Agro-Sells. Можна подумати, що на просторі українського

інтернету більше немає жодних аналогів. Це не так. Просто всі вони не становлять інтерес через мізерний функціонал, тому немає сенсу їх розглядати.

Аналіз буде поділено на дві частини. Спочатку будуть розглянуті специфічні для кожного конкурента особливості, цінності, які вони реалізують, а також яких не вистачає, а потім – загальні, властиві кожному з них. Наприкінці буде надана порівняльна таблиця характеристик аналогів і програми, що розробляється.

Першим аналогом буде розглянуто майданчик Agrobiz. Скріншот програми показано на рис. 1.1.

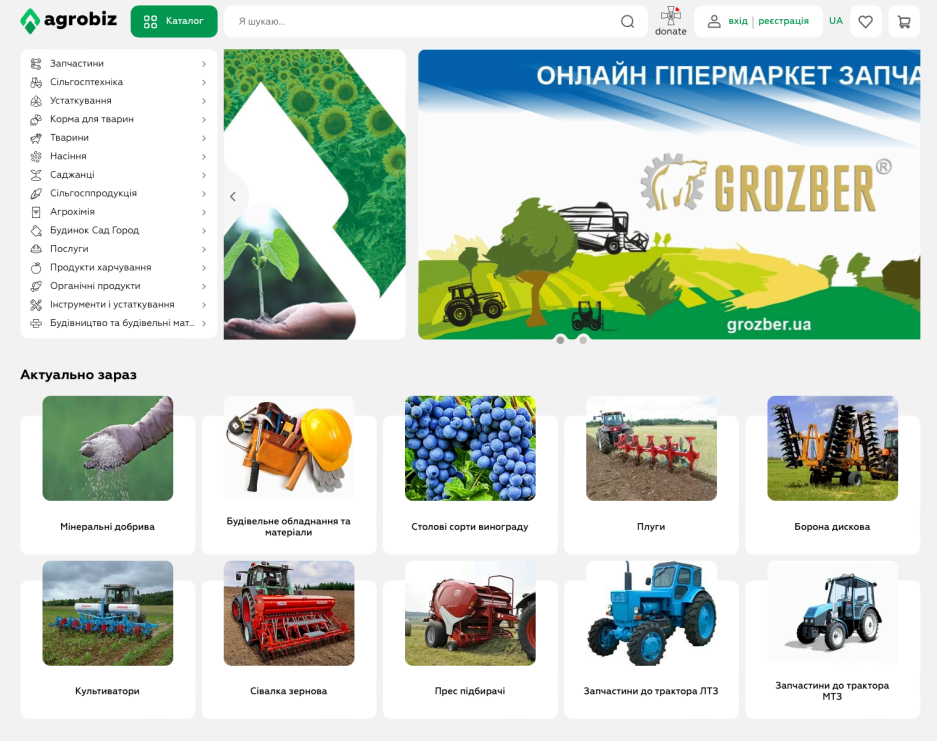


Рисунок 1.1 - Скріншот сайту Agrobiz

Вивчивши роботу майданчика, вдалося виділити такі переваги:

- привабливий дизайн. Проста палітра кольорів, немає яскравих кольорів, що відволікають погляд користувача;
- легкість навігації сторінками веб-програми. На кожній сторінці програми представлено меню навігації;
- велика кількість товарів, що надаються;
- пророоблений пошук та фільтрація товарів.

До недоліків можна віднести необхідність створення двох окремих акаунтів для купівлі та продажу товарів.

Далі розглянемо Agro-Sells. Як таких плюсів у цього додатка немає, він за всіма параметрами програє попередньому аналогу. Варто згадати, що тут присутній один аккаунт і для покупців, і для продавців, що відчутно впливає в позитивний бік на взаємодію користувача з системою. На рис. 1.2 представлений скріншот головної сторінки.

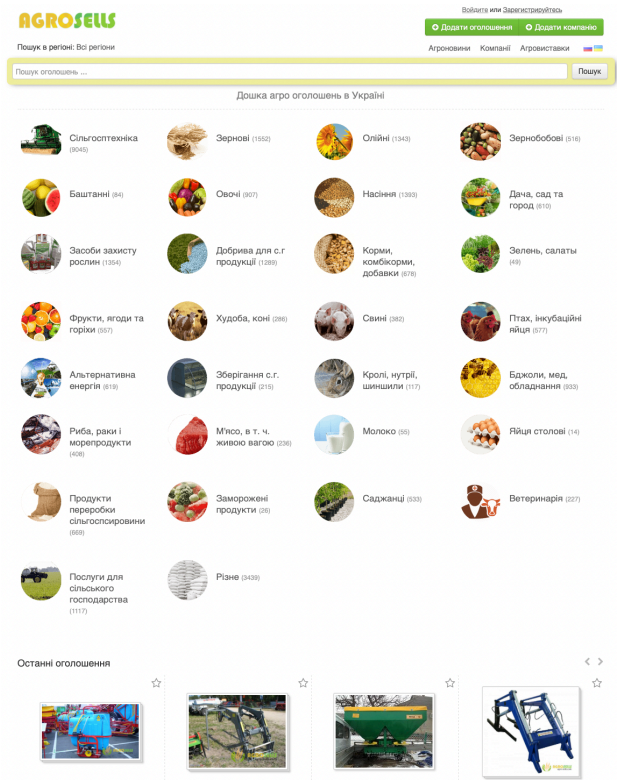


Рисунок 1.2 - Скріншот сайту Agro-Sells

Також у наведених вище аналогів є загальні недоліки. По-перше, вони не надають функціоналу для оформлення замовлення. Тобто ці майданчики просто надають контакти продавця і на цьому все. Вони не відповідають за подальше оформлення замовлення. Покупець повинен сам зв'язатися з продавцем, домовитися про оплату, доставку товарів тощо. По-друге, що випливає з попереднього пункту, документи, необхідні для оформлення замовлення, не зберігаються у системі. Щоразу, коли комусь із сторін потрібно буде знайти документи, йому доведеться шукати їх у пошті, месенджері, комп'ютері – одним

словом скрізь, але не в додатку. По-третє, після перевірки застосувань, що обговорюються, на наявність вразливостей, було з'ясовано, що вони є. В основному ці вразливості стосуються неправильної конфігурації HTTP-запитів [3]. Хоча це і не дуже критичні проломи в захисті програми, але при певному збігу обставини, вони можуть заподіяти чимало шкоди.

У табл. 1.1 показана порівняльна характеристика аналогів і програми, що розробляється. Як видно, LotAgro реалізує всі критично важливі цінності, які необхідні для перемоги в конкурентній боротьбі.

Таблиця 1.1 – порівняльна характеристика аналогів і LotAgro

Назва	Agrobiz	Agro-Sells	LotAgro
Обов'язкова реєстрація	-	-	+
OAuth2	-	-	+
Один аккаунт для покупця та продавця	-	+	+
Розгортання програми в Kubernetes	-	-	+
Автоматичне створення замовлення	-	-	+
Збереження документів про замовлення	-	-	+
Правильне конфігурування HTTP-запитів	-	-	+

Як видно з таблиці порівняння аналогів, ніхто з представлених конкурентів не реалізує критично важливі для клієнтів вимоги як функціональні, так і нефункціональні.

Підсумовуючи все перераховане вище, можна зробити висновок, що розробка веб-додатку LotAgro з купівлі-продажу сільськогосподарської продукції, а також його розгортання в середовищі оркестрації контейнерів є актуальним завданням.

1.3 Висновки до розділу

У першому розділі було проведено аналіз існуючої проблеми безпеки веб-додатків, розглянуто аналоги, визначено їхні слабкі та сильні сторони. Крім цього були визначені їхні функціональні та нефункціональні вимоги. Було підтверджено актуальність розробки автоматизованої системи купівлі-продажу сільськогосподарської продукції та розгортання її в кластері оркестрації контейнерів.

2 КОМПЛЕКС ЗАВДАНЬ ВЕБ-ЗАСТОСУВАНЬ

2.1 Алгоритми вирішення вразливостей

Як уже було сказано вище, часто веб-додатки схильні до вразливостей, тому в даному розділі будуть розглянуті процеси, що призводять до цих вразливостей, а також механізми та алгоритми захисту від них.

Як ми вже з'ясували, уразливості поділяються на два типи. До першого типу належать уразливості властиві рівню програми. Другий тип визначає вразливості які стосуються рівня інфраструктури. На рис. 2.1 для наочності наведена загальна структура застосування для розуміння на що саме спрямовані вразливості, що розглядаються.

Рисунок 2.1 - Загальна структура застосування

2.2 Механізми вирішення вразливостей рівня застосування

2.2.1 Аутентифікація та авторизація

Аутентифікація є процесом, в якому додаток ідентифікує того, хто намагається його використовувати. Коли хтось або щось використовує програму, ми хочемо встановити її особистість, щоб надати подальший доступ чи ні. Це необхідно для того, щоб надавати користувачеві певні дії та доступ до даних. Після етапу автентифікації користувача можна авторизувати в системі.

Авторизація - це процес встановлення, чи має абонент, що пройшов перевірку автентичності, права на використання певних функцій і даних.

Проблема полягає в тому, що при порушенні процесу авторизації зловмисник може якимось чином отримати доступ до функцій або даних, які не належать йому. Рис. 2.2 ілюструє проблеми відсутності процесу авторизації чи наявність вразливостей у ньому.

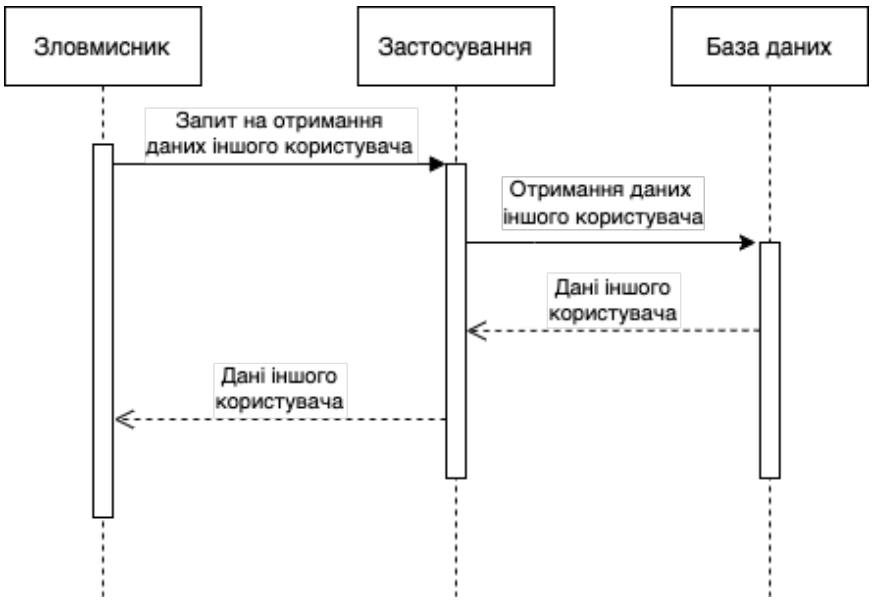


Рисунок 2.2 - Незахищений процес авторизації

Для вирішення цієї проблеми в системі повинен бути присутнім модуль або окремий сервіс, який відповідає за автентифікацію та авторизацію користувачів. Діаграма на рис. 2.3 багато в чому схожа на попередню діаграму. У ній ще додався

сервіс аутентифікації. Також зникла база даних через те, що зловмисник більше не може отримати доступ до чужих даних.

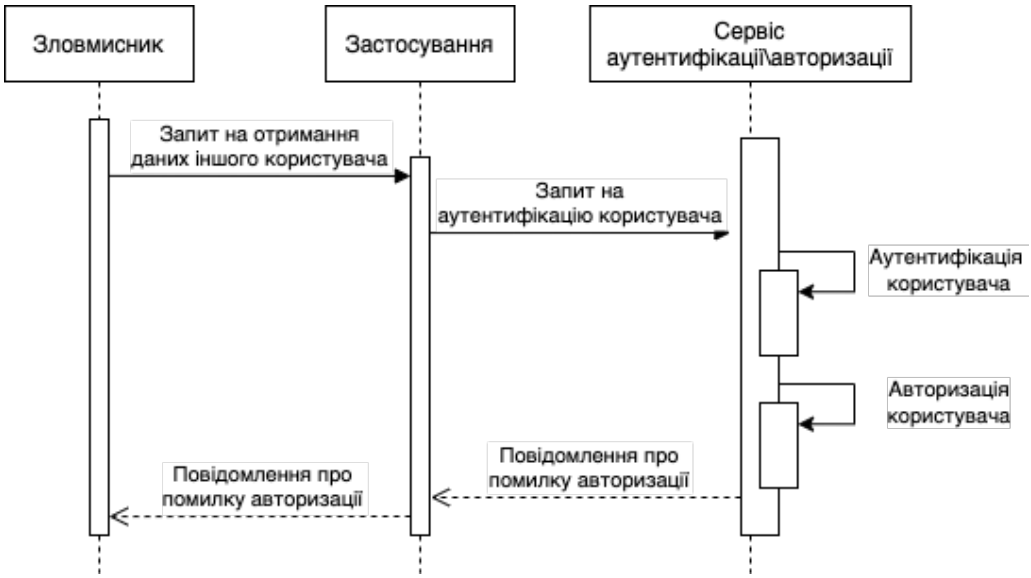


Рисунок 2.3 - Захищений процес авторизації

2.2.2 Фіксація сеансу

Вразливість, пов'язана з фіксацією сеансу, є більш специфічною та серйозною слабкістю веб-програми. Її присутність дозволяє зловмиснику видати себе за іншого дійсного користувача, використовуючи при цьому раніше згенерований ідентифікатор сеансу. Ця вразливість може виникнути, якщо під час автентифікації веб-програма не призначає унікальний ідентифікатор сеансу. Це може призвести до повторного використання існуючих ідентифікаторів сеансів. Експлуатація цієї вразливості полягає у отриманні дійсного ідентифікатора сеансу та використанні його браузером передбачуваної жертви.

Залежно від того, як реалізовано веб-додаток, зловмисники можуть використовувати цю вразливість у різний спосіб. Наприклад, якщо програма надає ідентифікатор сеансу в URL-адресі, жертву можна обдурити, змусивши клацнути зловмисне посилання. Якщо програма використовує прихований атрибут, зловмисник може обманом змусити жертву використовувати чужу форму, а потім

надіслати запит на сервер. Якщо програма зберігає значення сеансу у файлі cookie [4], зловмисник може впровадити скрипт і змусити браузер жертви виконати його.

Мірою протидії вразливості фіксації сеансу є кодування програми таким чином, щоб вона не могла прийняти токен, примусово введений у сеанс жертви.

Наступні кроки забезпечують надійний спосіб захисту веб-програм від цих атак:

- не приймати ідентифікатори сеансів у GET чи POST запитах. Це значно ускладнює атаку зловмиснику, оскільки простіше змусити жертву зробити запит без використання вразливостей браузера. Крім того, всі ідентифікатори сеансів повинні генеруватись сервером; клієнт не повинен мати потребу пропонувати новий ідентифікатор сеансу для програми;
- змінити ідентифікатор сеансу після входу до системи. Сервер повинен згенерувати новий ідентифікатор сеансу та додати його до файлу cookie після того, як користувач увійде до системи. Будь-який існуючий сеанс для користувача має бути знищений на сервері;
- забезпечити функцію виходу із системи та закінчення терміну дії старих сеансів. Користувач повинен мати можливість вибирати, коли завершити сеанс із програмою, яка має негайно завершити всі поточні сеанси на сервері, а не просто видалити файл cookie із браузера. Дані сеансу також повинні автоматично відключатися після закінчення певного періоду часу, щоб скоротити часовий інтервал, протягом якого зловмисник може використовувати скомпрометований сеанс.

2.2.3 Міжсайтовий скриптинг

Міжсайтовий скриптинг, також званий XSS, дозволяє впроваджувати скрипти на стороні клієнта до веб-служб, що надаються сервером, тим самим дозволяючи іншим користувачам запускати їх.

Для проведення атаки з використанням міжсайтових сценаріїв зловмисник впроваджує шкідливий сценарій у дані, що вводяться користувачем. Зловмисники

також можуть здійснити атаку, змінивши запит. Якщо веб-програма вразлива для XSS-атак, дані, що вводяться користувачем, виконуються як код. На рис. 2.4 показаний процес атаки.

Рисунок 2.3 – Процесс XSS атаки

Стратегії запобігання XSS-атакам включають наступне:

– ніколи не довіряйте введення користувача;
– реалізувати кодування виводу;
– виконувати перевірку введення користувача.

На рис. 2.4 показаний процес атаки.

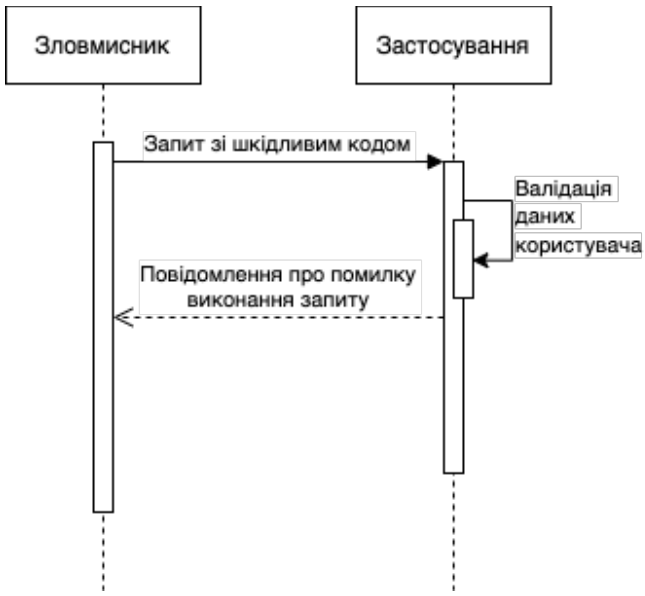


Рисунок 2.4 – Процес вирішення XSS атаки

2.2.4 Підробка міжсайтових запитів

Уразливість підробки міжсайтових запитів також поширена у веб-додатках. Атаки CSRF припускають, що URL-адреса [5], що викликає дію на певному сервері, може бути витягнута і повторно використана поза програмою. Якщо сервер довіряє виконанню без перевірки походження запиту, його можна виконати з будь-якого іншого місця. За допомогою CSRF зловмисник може змусити користувача виконувати небажані дії на сервері, приховуючи ці дії. Зазвичай, за допомогою цієї вразливості зловмисник націлений на дії, які змінюють дані в системі.

Одним із способів усунення цієї вразливості є використання токенів для ідентифікації запиту або обмеження спільного використання ресурсів між джерелами (CORS). Іншими словами, необхідно завжди перевіряти походження запиту та дозволяти виконувати ті з них, яким можна довіряти. На рис. 2.5 показано механізм захисту від CSRF атак.

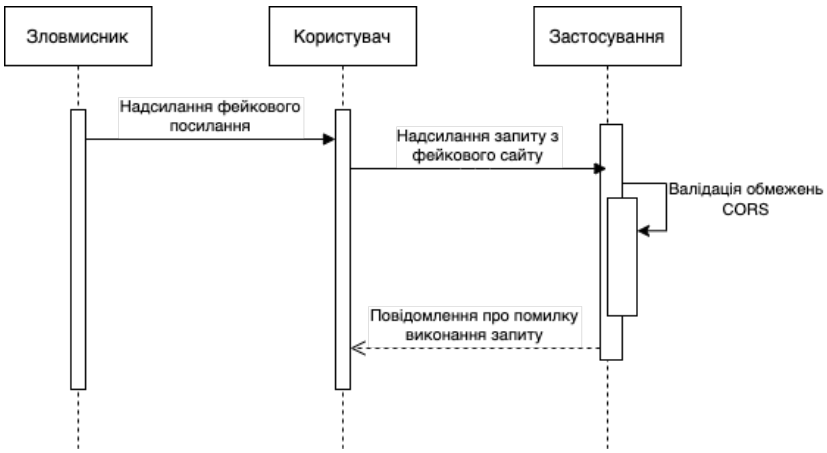


Рисунок 2.5 – Механізм захисту від CSRF атак

2.2.5 Ін'єкції

Ін'єкційні атаки на системи поширені. Під час атаки шляхом впровадження зловмисник, який використовує вразливість, вводить у систему певні дані. Ціль полягає в тому, щоб завдати шкоди системі, змінити дані небажаним чином або отримати дані, які не призначені для доступу зловмисника.

Існує багато типів ін'єкційних атак. Зрештою, всі ін'єкційні атаки впроваджують шкідливий код у систему. Прикладами може бути використання SQL [6], використання XPath [7], використання команд ОС, використання LDAP [8] тощо.

Одним із найстаріших і, можливо, добре відомих типів вразливостей, пов'язаних з ін'єкціями, є SQL-ін'єкція. Якщо ваш додаток має вразливість ін'єкції SQL, зловмисник може спробувати змінити або запустити різні SQL-запити, щоб змінити, видалити або витягти дані з вашої системи. У найпросунутіших атаках із використанням SQL-коду людина може запускати команди ОС у системі, що зумовлює повної компрометації системи.

Найголовніше правило для вирішення цього типу атак - дані, що надсилаються користувачем, не повинні брати участь у формуванні тексту SQL-запиту, принаймні безпосередньо. Досягається використання параметризованих (підготовлених) запитів.

У деяких випадках неможливо налаштувати запити. У такому випадку необхідно обмежити список допустимих значень, які можуть надходити від користувача (білий список).

2.2.6 Розкриття конфіденційних даних

Навіть якщо з точки зору складності розкриття конфіденційних даних здається найпростішим для розуміння і найменш складним з вразливостей, воно залишається однією з найпоширеніших помилок. Це відбувається через те, що розробники часто використовують гайди з Інтернету для вирішення поставлених завдань, не звертаючи уваги на те, що часто в таких гайдах проблеми безпеки опущені для спрощення. Прикладом є зберігання облікових даних у файлах проекту.

Встановивши такі значення у файлах конфігурації, ці конфіденційні значення доступні всім, хто може бачити вихідний код. Більше того, в системі керування

версіями вихідного коду зберігається історія змін. Також до розкриття конфіденційних даних відноситься інформація в лог файлах.

Ще однією великою проблемою є дані, які система повертає користувачеві. Часто трапляється ситуація, коли програма повертає занадто багато деталей, що розкривають реалізацію. Наприклад, деякі повідомлення про помилки в системі можуть розкривати IP-адресу. Зловмисник може використовувати цю адресу, щоб вивчити конфігурацію мережі та, зрештою, знайти спосіб управління інфраструктурою, в якій розгорнуто програму. Звичайно, маючи тільки цю таких даних, не можна заподіяти жодної шкоди. Але збирання різних фрагментів інформації та їх об'єднання можуть надати все необхідне для несприятливого впливу на систему.

Завжди важливо надавати користувачеві мінімум необхідної інформації, проводять валідацію всіх вихідних повідомлень.

2.2.7 Використання залежностей з відомими вразливостями

Іноді вразливими місцями є не програма, яка розробляється, а залежності, такі як бібліотеки або фреймворки, які використовуються для створення функціональності. Завжди уважно потрібно ставитися до залежностей, що використовуються і видаляти всі версії, про які відомо, що вони містять вразливості.

На щастя, існує кілька утиліт для статичного аналізу, що дозволяють визначити залежність вразливостей. При розробці будь-якого програмного забезпечення потрібно вживати всіх необхідних заходів, щоб уникнути використання будь-яких залежностей, які мають відомі вразливості. Якщо все-таки виявився факт використання такої залежності, необхідно не тільки виправити її, оновивши версію залежності або використавши іншу зі схожим функціоналом, але також з'ясувати, чи використовувалася дана функціональність в системі (в даному випадку потрібно буде вжити необхідних заходів).

2.3 Механізми вирішення вразливостей рівня інфраструктури

2.3.1 Зараження шкідливим кодом

Зловмисники на постійній основі перевіряють наявність загальнодоступних уразливостей та незахищених компонентів, які можуть бути використані для виконання експлойтів, що дозволяють проникнути до системи. Далі будуть розглянуті різні види експлойтів та методів боротьби з ними.

Криптоджекінг - це процес або запуску криптомайнера в зараженому контейнері або віртуальній машині, або запуску виділеного контейнера або віртуальної машини з програмним забезпеченням для криптомайнінгу. Іншою формою криптоджекінгу є атака з отруєнням ланцюжка поставок, при якій образи загальнодоступних контейнерів, наприклад, що зберігаються в Docker Hub, містять програмне забезпечення для криптомайнінгу. Ці контейнери замасковані під щось корисне, наприклад, під популярне доброякісне зображення, і часто працюють так, як задумано, просто з побічним процесом, який майне криптовалюту для зловмисників.

Шкідливе ПЗ рідко задовольняється першою точкою входу. Використовуючи інструменти сканування портів, такі як masscan, pnscan та zgrab, а також дампи CLI [9], шкідливі програми можуть знайти наступну мету для зараження та розповсюдження. Звідти вони можуть відкрити доступ до інших контейнерів та віртуальних машин за допомогою програмного забезпечення для керування службами, таких як SSH, або програмного забезпечення API, такого як команди API Kubernetes [10].

Одним з найпопулярніших методів заплутування є запуск шкідливого ПЗ тільки в пам'ять та очищення історії журналу. Це ускладнює виявлення та запобігання таким атакам при скануванні томів. Шкідливе програмне забезпечення включається лише тимчасово, іноді в каталог tmp або у вигляді сценарію тільки в пам'яті, а потім стирається. Це робить судову експертизу без активного моніторингу практично неможливою.

Інструменти керування вразливістю дозволяють виявляти всі відомі вразливості у базових образах та пакетах та надавати рекомендації щодо оновлення. Коли вразливості не можуть бути виправлені чи немає доступних виправлень, надання віртуальних виправлень та інших засобів захисту під час виконання може бути корисним компенсуючим елементом керування.

Найкращий спосіб захиститися від такого роду експлойтів – максимально утруднити отримання доступу до облікових даних та максимально заблокувати облікові дані.

Процеси криптомайнінгу можна зупинити у кількох місцях. При зверненні до серверів управління та контролю мережевий моніторинг може блокувати трафік на відомі шкідливі IP-адреси та домени. Платформи захисту робочого навантаження можуть блокувати запуск відомих криптомайнерів та відстежувати події до та після спроби. Якщо контейнер або віртуальна машина намагається завантажити файл або запустити образ контейнера, інструменти безпеки можуть визначити сигнатури відомих шкідливих програм для криптомайнінгу або відправити їх до пісочниці для виявлення невідомих шкідливих програм. Крім того, ці інструменти можуть обмежити розгортання лише відомими, надійними образами та заблокувати всі інші образи.

Блокування бокового руху включає блокування кожного кроку спроби. Почніть з обмеження поверхні атаки, використовуючи мікросегментацію, щоб обмежити служби, які можуть взаємодіяти один з одним. Використовуйте інструменти захисту робочих навантажень, щоб блокувати відомі інструменти сканування портів. Kubernetes дозволяє заздалегідь визначити взаємодію служби, тому навряд чи знадобиться сканування портів із контейнера. Потім заблокуйте доступ по SSH та RDP лише до відомих користувачів та процесів та заблокуйте свої API-інтерфейси Kubernetes та Docker [11].

Єдиним захистом від цих атак є активний моніторинг. Агенти можуть виявляти та блокувати шкідливі процеси та модифікації файлів у джерелі. Для більшості середовищ Kubernetes запобігання доступу до оболонки є гарною практикою та ефективним заходом проти деяких атак та очищення історії. Такі

команди, як history -c, можна заблокувати за допомогою настроюваних правил, щоб зберегти незмінну історію дій у журналах.

2.3.2 Стандартні конфігурації

Існує термін "Plug and play". Він передбачає можливість відразу ж запустити програму з мінімальними змінами в конфігураціях. З одного боку, це звучить привабливо, тому що немає необхідності витрачати час на вивчення специфікацій та посібників із конфігурування. Але також ця зручність може швидко стати вразливою.

Зазвичай під дефолтними значеннями розуміють логіни та паролі облікових записів адміністраторів системи. Але є й інші елементи конфігурації, які можуть бути небезпечні, але трохи по-іншому. До таких даних можуть належати зміни мереж, баз даних, різних сервісів інфраструктури. Велика кількість зламів систем трапляється саме через те, що дефолтні значення конфігурацій не були замінені на оновлені.

При налаштуванні будь-якої частини інфраструктури в першу чергу слід потурбуватися про зміну всіх конфігурацій з наданих свої особисті (звісно ж з урахуванням специфіки системи).

2.3.3 Незахищена передача даних

Передаючи дані по мережі, не важливо, чи це буде публічна або приватна мережа, можна зіткнутися з трьома основними проблемами безпеки:

- Як ми можемо дізнатися, чи дійсно людина, з якою ми спілкуємося, та, за кого вона себе видає?
- Як ми можемо знати, що дані не були підроблені з моменту їх надсилання?
- Як ми можемо заборонити іншим людям переглядати та отримувати доступ до даних?

Метою таких атак є крадіжка особистої інформації, такої як облікові дані для входу в систему, дані облікового запису, номери кредитних карток і так далі. Цілями зазвичай є користувачі фінансових додатків, підприємств SaaS, сайтів електронної комерції та інших веб-сайтів, де потрібний вхід до системи.

Інформація, отримана під час атаки, може бути використана для багатьох цілей, включаючи крадіжку особистих даних, несанкціоновані перекази коштів або незаконну зміну пароля.

Для додатків вважається найкращою практикою використовувати SSL/TLS [12] для захисту кожної сторінки свого сайту, а не лише сторінок, які вимагають від користувачів входу в систему. Це допомагає знизити ймовірність того, що зловмисник украде файли cookie сеансу у користувача.

TLS використовує низку криптографічних методів для вирішення кожної з цих трьох проблем. Разом вони дозволяють протоколу аутентифікувати іншу сторону з'єднання, перевіряти цілісність даних та забезпечувати зашифрований захист.

2.3.4 Відсутність механізмів захисту даних

Великий відсоток витоків даних є результатом зловмисної атаки, а викликаний недбалим чи випадковим розкриттям конфіденційних даних. Співробітники організації часто діляться, надають доступ, втрачають або неправильно поводяться з цінними даними або випадково, або тому, що вони не знають політиків безпеки.

Атаки з використанням соціальної інженерії є основним вектором, який використовується зловмисниками для доступу до конфіденційних даних. Вони пов'язані з маніпулюванням або обманом людей, щоб вони надали особисту інформацію або доступ до привілейованих облікових записів.

Також проблемою є переміщення даних у хмару, щоб полегшити спільне використання та спільну роботу. Однак коли дані переміщуються в хмару, їх складніше контролювати та запобігати втраті даних. Користувачі отримують доступ до даних із особистих пристроїв та через незахищені мережі. Занадто легко поділитися файлом з неавторизованими сторонами випадково або зловмисно.

Існує кілька технологій та методів, які можуть підвищити безпеку даних. Жоден метод не може вирішити проблему повністю, але, комбінуючи кілька з наведених нижче методів, організації можуть значно покращити свою безпеку.

2.3.4.1 Виявлення та класифікація даних

Сучасні IT-середовища зберігають дані на серверах, кінцевих точках та в хмарних системах. Спостереження за потоками даних - важливий перший крок до розуміння того, які дані можуть бути вкрадені або використані за призначенням. Щоб належним чином захистити свої дані, необхідно знати тип даних, де вони і для чого використовуються. У цьому можуть допомогти інструменти виявлення та класифікації даних.

Виявлення даних є основою визначення того, які дані у вас є. Класифікація даних дозволяє створювати масштабовані рішення для забезпечення безпеки, визначаючи, які дані є конфіденційними та потребують захисту. Рішення для виявлення та класифікації даних дозволяють помічати файли на кінцевих точках, файлових серверах та в хмарних системах зберігання, дозволяючи візуалізувати дані по всьому підприємству та застосовувати відповідні політики безпеки.

2.3.4.2 Маскування даних

Маскування даних дозволяє створити синтетичну версію даних, яку можна використовувати для тестування програмного забезпечення, навчання та інших цілей, які не потребують реальних даних. Ціль полягає в тому, щоб захистити дані, надаючи при необхідності функціональну альтернативу.

Маскування даних зберігає тип даних, але змінює значення. Дані можна змінити кількома способами, включаючи шифрування, перетасовування символів та заміну символів або слів. Хоч би який метод ви вибрали, ви повинні змінити значення таким чином, щоб їх не можна було реконструювати.

2.3.4.3 Управління доступом до ресурсів

Управління ідентифікацією та доступом (IAM) – це бізнес-процес, стратегія та технічна структура, які дозволяють організаціям керувати цифровими посвідченнями. Рішення IAM дозволяють IT-адміністраторам контролювати доступ користувачів до конфіденційної інформації.

Системи, що використовуються для IAM, включають системи єдиного входу, двофакторну автентифікацію, багатофакторну автентифікацію та керування привілейованим доступом. Ці технології дозволяють організації безпечно зберігати ідентифікаційні дані та дані профілів, а також підтримувати управління, гарантуючи застосування відповідних політик доступу до кожної частини інфраструктури.

2.3.4.4 Шифрування даних

Шифрування даних — це метод перетворення даних із незашифрованого формату в закодований формат, що не зчитається (зашифрований текст). Тільки після розшифрування зашифрованих даних за допомогою ключа дешифрування дані можуть бути прочитані чи оброблені.

У методах криптографії з відкритим ключем немає необхідності ділитися ключем дешифрування — відправник та одержувач мають власні ключі, які об'єднуються для виконання операції шифрування. Це за своєю суттю безпечніше.

Шифрування даних може завадити хакерам отримати доступ до конфіденційної інформації. Це важливо для більшості стратегій безпеки і явно вимагається багатьма стандартами відповідності.

2.3.4.5 Запобігання втраті даних

Щоб запобігти втраті даних, організації можуть використовувати низку заходів безпеки, включаючи резервне копіювання даних в інше місце. Фізична

надмірність може допомогти захистити дані від стихійного лиха, збоїв або атак на локальні сервери. Надмірність може виконуватися в локальному центрі обробки даних або шляхом реплікації даних на віддалений майданчик або хмарне середовище.

Крім базових заходів, таких як резервне копіювання, програмні рішення DLP можуть допомогти захистити ці організації. Програмне забезпечення DLP автоматично аналізує вміст для виявлення конфіденційних даних, забезпечуючи централізований контроль та застосування політик захисту даних, а також оповіщення в режимі реального часу при виявленні аномального використання конфіденційних даних, наприклад, великих обсягів даних, скопійованих за межі корпоративної мережі.

2.3.4.6 Аутентифікація та авторизація

Організації повинні впровадити надійні методи аутентифікації, такі як OAuth для веб-систем. Настійно рекомендується застосовувати багатофакторну автентифікацію, коли будь-який користувач, чи то внутрішній, чи зовнішній, запитує конфіденційні чи особисті дані.

Крім того, організації повинні мати чітку структуру авторизації, яка гарантує, що кожен користувач має ті права доступу, які йому необхідні для виконання функції або використання послуги, і не більше того. Слід використовувати періодичні перевірки та автоматизовані інструменти для очищення дозволів та видалення авторизації для користувачів, яким вони більше не потрібні.

2.3.4.7 Аудит безпеки даних

Організація повинна проводити перевірки безпеки не рідше ніж один раз на кілька місяців. Це виявляє прогалини та вразливості у системі безпеки організацій. Рекомендується проводити аудит із залученням стороннього експерта, наприклад, у рамках моделі тестування на проникнення. Однак також можна провести аудит

безпеки вдома. Найголовніше, коли аудит виявляє проблеми з безпекою, організація повинна виділити час та ресурси для їх вирішення та усунення.

2.4 Висновки до розділу

У другому розділі було проведено детальний аналіз вразливостей веб-додатків. Вони були поділені на два типи: вразливість рівня програми та вразливість рівня інфраструктури, в якій ця програма запускатиметься. Для кожного з такого типу були розглянуті вразливості, що найчастіше зустрічаються. Такий аналіз дозволив визначити причини, які ведуть до появи слабких місць у системі (для деяких з вразливостей були наведені діаграми для більш детального опису процесу їх появи). Після аналізу причин були надані механізми та алгоритми їх вирішення. Результати проведеного в цьому розділі аналізу будуть використані в подальшому, зокрема при проектуванні архітектури системи, а також при виборі технологій реалізації. Це дозволить розробити систему так, щоб уникнути присутності в ній вразливостей, які можуть призвести до небажаних наслідків.

3 СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Функціональні вимоги

Вимоги безпеки повинні бути засновані на аналізі активів та послуг, які необхідно захистити, та загроз безпеки, від яких ці активи та послуги мають бути захищені. Існують чіткі взаємозв'язки між активами та сервісами, які вразливі для загроз безпеці, що обумовлює вимоги безпеки, що вимагають механізмів безпеки, які протидіють цим загрозам безпеці та тим самим захищають активи та сервіси.

Спочатку розберемо акторів та діаграму варіантів використання поза контекстом аспектів безпеки. Це дасть розуміння основних діючих користувачів системи та дій, які можуть виконувати, з урахуванням їх ролей. У табл. 3.1 наведено перелік акторів, що діють.

Таблиця 3.1 – Актори системи

Актор	Опис актора
Клієнт	Агрегація користувачів Покупець та Продавець.
Покупець	Користувач, який здійснює покупку.
Продавець	Користувач, який здійснює продаж
Модератор	Користувач, який відповідає за модерацію лотів та замовлень

На рис. 3.1 зображено варіанти використання системи.

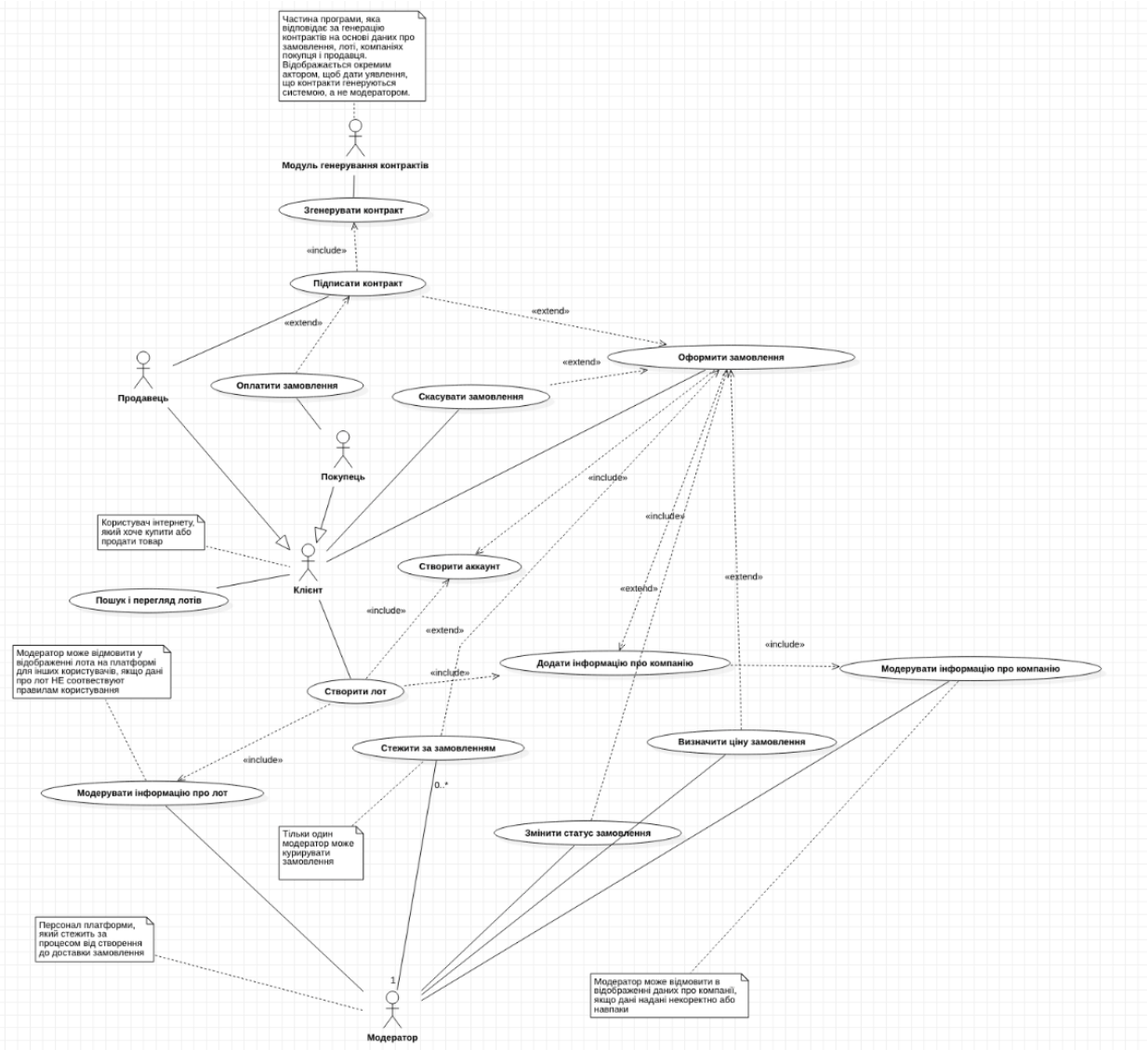


Рисунок 3.1 – Діаграма варіантів використання

Оскільки метою роботи є реалізація захисних механізмів системи, то буде показано ще одну діаграму варіантів використання (рис. 3.2), але вже у контексті аспектів безпеки. Вимоги до функціональної безпеки програмного забезпечення визначають функції безпеки, які мають забезпечувати програмне забезпечення. Очевидно, що функціональні вимоги безпеки є підмножиною загальних функціональних вимог.

Для більш повного розуміння варіантів використання розглянемо їх сценарії (табл 3.2 – 3.9). Вони дозволять розібратися у взаємодії акторів з системою.

Рисунок 3.2 – Діаграма варіантів використання у контексті безпеки

Таблиця 3.2 –Сценарії варіантів використання №1

Тип	Контроль доступу
Назва	Спроба спуфінгу за допомогою дійсного ідентифікатора користувача.
Загроза безпеці	Система аутентифікує та авторизує зловмисника, начебто він був дійсним користувачем.
Передумови	1) Зловмисник має дійсні засоби ідентифікації користувача. 2) Зловмисник використовує неприпустимі засоби автентифікації користувача.

Продовження таблиці 3.2

Тип	Контроль доступу
Взаємодії	1) Система має запросити засоби ідентифікації та аутентифікації зловмисника. 2) Неправомірний користувач надає допустимі засоби ідентифікації користувача, але недійсні засоби автентифікації користувача. 3) Система не повинна аутентифікувати та авторизувати зловмисника. 4) Система відхиляє запит зловмисника, скасувавши транзакцію.
Постумови	1) Система не повинна дозволяти зловмиснику вкрасти засоби автентифікації користувача. 2) Система не повинна аутентифікувати зловмисника як справжнього користувача. 3) Система не повинна дозволяти зловмиснику виконувати будь-які транзакції, що потребують автентифікації. 4) Система має зареєструвати збій контролю доступу.

Таблиця 3.3 – Сценарії варіантів використання №2

Тип	Контроль доступу
Назва	Спроба крадіжки особистих даних та аутентифікації
Загроза безпеці	Зловмисник краде засоби ідентифікації та аутентифікації користувача, тим самим дозволяючи зловмиснику видавати себе за дійсного користувача.
Передумови	1) Зловмисник не має дійсних засобів ідентифікації користувача. 2) Зловмисник не має дійсних засобів ідентифікації чи автентифікації користувача.

Продовження таблиці 3.3

Тип	Контроль доступу
Взаємодії	1) Система повинна запросити особу та автентифікацію користувача. 2) Користувач ідентифікує та аутентифікує себе. 3) Зловмисник намагається вкрасти засоби користувача для ідентифікації та аутентифікації. 4) Система повинна захищати особу та автентифікацію користувача під час взаємодії. 5) Система повинна ідентифікувати та аутентифікувати користувача. 6) Система має запросити у користувача вибір взаємодії.
Постумови	1) Система повинна запобігати крадіжці зловмисником засобів ідентифікації та автентифікації користувача. 2) Система повинна ідентифікувати та аутентифікувати користувача

Таблиця 3.4 – Сценарії варіантів використання №3

Тип	Цілісність
Назва	Системні дані захищені
Загроза безпеці	Зловмисник може пошкодити (наприклад, додати, змінити, видалити) конфіденційні дані, що зберігаються у системі.
Передумови	Система зберігає конфіденційні дані, які не повинні бути пошкоджені.

Продовження таблиці 3.4

Тип	Контроль доступу
Взаємодії	1) Зловмисник намагається пошкодити (наприклад, додати, змінити, видалити) конфіденційні дані, що зберігаються у системі. 2) Система повинна запобігати пошкодженню даних. 3) Система повинна повідомити співробітника служби безпеки про спробу пошкодження даних.
Постумови	Система повинна гарантувати, що конфіденційні дані не будуть пошкоджені.

Таблиця 3.5 – Сценарії варіантів використання №4

Тип	Цілісність
Назва	Система захищена від несанкціонованого доступ
Загроза безпеці	Зловмисник може запустити шкідливий код або програму
Передумови	Система розгорнута у безпечному оточенні
Взаємодії	1) Зловмисник намагається отримати доступ до системи. 2) Система повинна запобігати несанкціонованому доступу. 3) Система повинна повідомити співробітника служби безпеки про спробу несанкціонованого доступу.
Постумови	Оточення повинно гарантувати, що сторонні не можуть отримати доступ всередину і виконувати сторонній код або програми.

Таблиця 3.6 – Сценарії варіантів використання №5

Тип	Цілісність
Назва	Цілісність системних повідомлень

Продовження таблиці 3.6

Тип	Цілісність
Загроза безпеці	Зловмисник спотворює повідомлення, яке надсилається системою користувачеві.
Передумови	1) Зловмисник має засоби для перехоплення повідомлення від системи до користувача. 2) Зловмисник має засоби змінити перехоплене повідомлення. 3) Зловмисник може переслати змінене повідомлення користувачу.
Взаємодії	1) Система повинна надіслати повідомлення користувачу. 2) Система повинна гарантувати, що модифікації повідомлення будуть очевидними для користувача. 3) Зловмисник перехоплює та модифікує системне повідомлення та пересилає його користувачеві. 4) Користувач отримує пошкоджене повідомлення. 5) Система має визнати, що її повідомлення було пошкоджене. 6) Система повинна повідомити користувача, що його повідомлення було пошкоджено.
Постумови	Система повинна повідомити користувача про пошкодження системного повідомлення.

Таблиця 3.7 – Сценарії варіантів використання №6

Тип	Цілісність
Назва	Цілісність повідомлень користувача
Загроза безпеці	Зловмисник спотворює повідомлення користувача системі.

Продовження таблиці 3.7

Тип	Цілісність
Передумови	Зловмисник має засоби для перехоплення повідомлення між користувачем та системою.
Взаємодії	1) Користувач надсилає повідомлення до системи. 2) Зловмисник перехоплює, змінює та пересилає повідомлення користувача. 3) Система має розпізнати, що повідомлення користувача було пошкоджено. 4) Система повинна повідомити користувача про пошкодження повідомлення користувача.
Постумови	Система повинна повідомити користувача про пошкодження повідомлення користувача.

Таблиця 3.8 – Сценарії варіантів використання №7

Тип	Конфіденційність
Назва	Конфіденційність даних
Загроза безпеці	Зловмисник отримує доступ до особистих даних, що зберігаються у системі.
Передумови	Система зберігає особисті дані.
Взаємодії	1) Система має зробити приватні збережені дані нечитаними. 2) Зловмисник отримує доступ до особистих даних, що зберігаються в системі.
Постумови	Система повинна зберігати особисті дані у формі, яка недоступна для читання зловмисником.

Таблиця 3.9 – Сценарії варіантів використання №8

Тип	Конфіденційність
Назва	Конфіденційність системних повідомлень
Загроза безпеці	Зловмисник отримує доступ до особистого повідомлення від системи до користувача.
Передумови	Зловмисник має засоби для перехоплення повідомлення від системи користувача.
Взаємодії	1) Система має зробити особисте повідомлення нечитаним під час передачі. 2) Система має надіслати особисте повідомлення користувачеві. 3) Зловмисник перехоплює особисте повідомлення системи.
Постумови	Система має надіслати особисте повідомлення у формі, яку зловмисник не зможе прочитати.

3.2 Нефункціональні вимоги

Нефункціональні вимоги безпеки визначають якість або атрибут безпеки, який повинен мати програмне забезпечення. Існує 3 типи нефункціональних вимог безпеки: вимоги до властивостей безпеки, вимоги обмежень (негативні вимоги) та вимоги до забезпечення безпеки.

Вимоги до властивостей безпеки визначають властивості, якими має володіти програмне забезпечення. До таких вимог належать:

- програмне забезпечення має залишатися стійким до атак;
- поведінка програмного забезпечення повинна бути коректною та передбачуваною;
- програмне забезпечення має забезпечувати цілісність інформації про обліковий запис клієнта.

Негативні вимоги накладають обмеження на функції програмного забезпечення, щоб звести до мінімуму ймовірність небезпечної поведінки програмного забезпечення, зазвичай, з точки зору того, чого слід уникати або запобігати. Обмеження/негативні вимоги існують, оскільки не можна допускати, щоб функціональні можливості програмного забезпечення поводилися таким чином, що може призвести до збою програмного забезпечення в небезпечному стані або іншим чином стати вразливим до експлуатації або компрометації. Можна виділити такі вимоги:

- Сервер не повинен повертати веб-сторінку з обмеженим доступом будь-якому браузеру, який не може аутентифікуватись.
- Сервер не повинен повертати веб-сторінку з обмеженим доступом для користувача, який не має права доступу до неї.
- Програмне забезпечення не повинно приймати надто довгі вхідні дані.
- Програма не повинна приймати неприпустимі URL-адреси.

Вимоги безпеки — це правила, найкращі практики та процеси, за допомогою яких створюватимуться, розгортатимуться та керуватимуться функції безпеки програмного забезпечення. Вимоги забезпечення безпеки будуть перетворені не на елементи дизайну програмного забезпечення, а на стандарти, посібники або процедури для процесів його розробки та експлуатації. Такими вимогами є:

- Програмне забезпечення має бути побудовано відповідно до стандартів безпеки веб-сервісів SOA;
- Процеси розробки повинні відповідати рівню можливостей SSE-CMM 3 або вище.

3.3 Висновки до розділу

У третьому розділі було визначено функціональні та нефункціональні вимоги до системи у контексті безпеки. Функціональні вимоги були представлені у вигляді діаграми варіантів використання та опису прецедентів. Нефункціональні вимоги були поділені на три типи, і до кожного з них були наведені приклади вимог.

4 ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

4.1 Огляд ключових моментів системи

У цій секції розглядатимуться структура оточення, в якій розгортатиметься система, а також структура та модулі самої системи. Також буде порушено порівняння попередніх рішень та їх недоліки.

Як було сказано вище, безпека програми є ключем до успіху. У цій роботі будуть розглянуті та реалізовані такі заходи щодо безпеки веб-застосування з продажу сільгосппродукції:

- міграція бекенд-частини програми з монолітної архітектури на мікросервісну;
- розгортання веб-застосування на платформі контейнерної оркестрації програм;
- інтеграція фреймворку Spring Security [13];
- створення індивідуального фреймворку ACL.

Поговоримо про кожен із цих пунктів трохи більш розгорнуто, щоб отримати розуміння необхідності їх реалізації.

Почнемо із міграції монолітної архітектури на мікросервісну. Відмінності між ними полягають в тому, що монолітний додаток являє собою один єдиний артефакт, запустивши який, можна буде користуватися додатком. Мікросервісна архітектура ж є набором невеликих програм, кожна з яких відповідальна за свою частину бізнес-логіки. Всі разом вони утворюють цільну програму, яка виконує ту ж логіку, що і моноліт, що просто має іншу структуру. Зміна архітектури дозволить інкапсулювати бізнес-логіку програми у різних місцях, що дозволить мінімізувати ризик злому всій системі.

Для чого потрібне розгортання системи на платформі контейнерного оркестрування програм? Найпопулярнішою та широко використовуваною з таких платформ є Kubernetes. Це найважливіша частина, яка дозволить уникнути великої кількості зовнішніх вразливостей та проломів у захисті системи. Kubernetes має велику кількість налаштувань та пристроїв для ізоляції модулів системи від

несанкціонованого доступу ззовні. До таких речей належать Network Policy, RBAC, Pod Security Policy, Secrets. Так само чимало важливо, звертаючи увагу на попередній пункт, є посилення аспекту безпеки взаємодії між мікросервісами. Це зокрема дозволяє зробити Kubernetes, за допомогою технології званої Service Mesh. Реалізація всіх перерахованих особливостей Kubernetes дозволить досягти максимального захисту системи.

Spring Security – це де-факто єдиний фреймворк для всеосяжного захисту додатків, написаних на Spring [14] (кожний мікросервіс у розроблюваному додатку – це окремий додаток, написаний мовою Java [15], використовуючи як основу Spring). Він відповідальний за реалізацію методів безпеки на рівні додатку, а не на інфраструктурі, як Kubernetes. Сфери застосування варіюються від аутентифікації та авторизації користувачів у системі до зберігання даних у системі та передачі даних між різними частинами системи.

Заключним кроком є реалізація ACL. ACL є частиною системи, яка займається управлінням доступу до захищених ресурсів програми на основі ролей користувачів. Прикладом такого механізму може бути функціонал отримання даних замовлення. Як ви знаєте, до замовлення належать укладені контракти про купівлю-продаж товару. Без ACL покупець зможе отримати доступ до контракту продавця та навпаки, що є великим ризиком.

4.2 Проектування інфраструктури

Почнемо з оточення. Стандартним методом запуску застосування був його запуск на фізичному сервері. Крім простоти використання цей спосіб привносив набагато більше проблем, ніж переваг. Однією з таких проблем була відсутність поділу ресурсів між програмами на сервері. Рішенням є запуск програми у контейнері, який у свою чергу управляється Kubernetes кластером. Віртуалізація ізолює програми між собою на одному фізичному сервері, підвищуючи рівень безпеки, через те, що програми не мають доступу один до одного і використовують свої власні файлові системи, пам'ять, простір процесора і так далі. Ілюстрація

наочно показує різницю між цими двома підходами. На рис. 4.1 зображено структуру програми на фізичному сервері, на відміну від запуску в кластері (рис. 4.2).

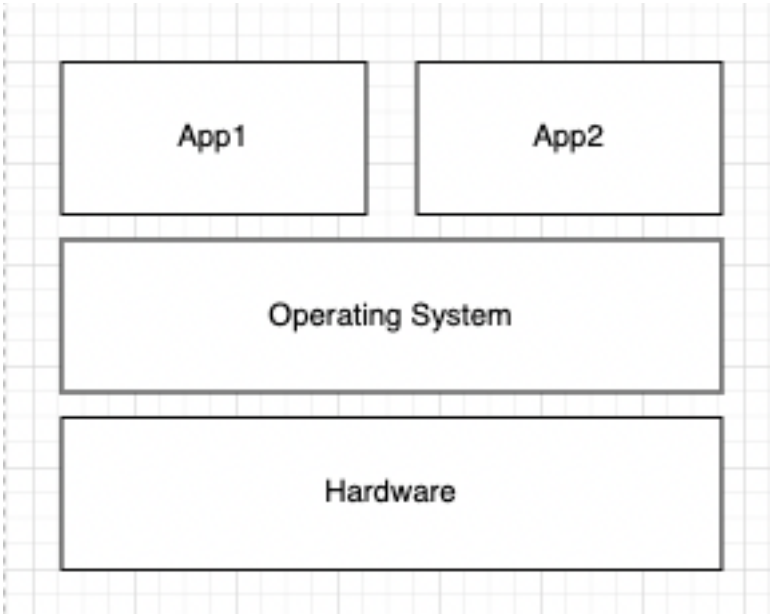


Рисунок 4.1 – Розгортання програми на фізичному сервері

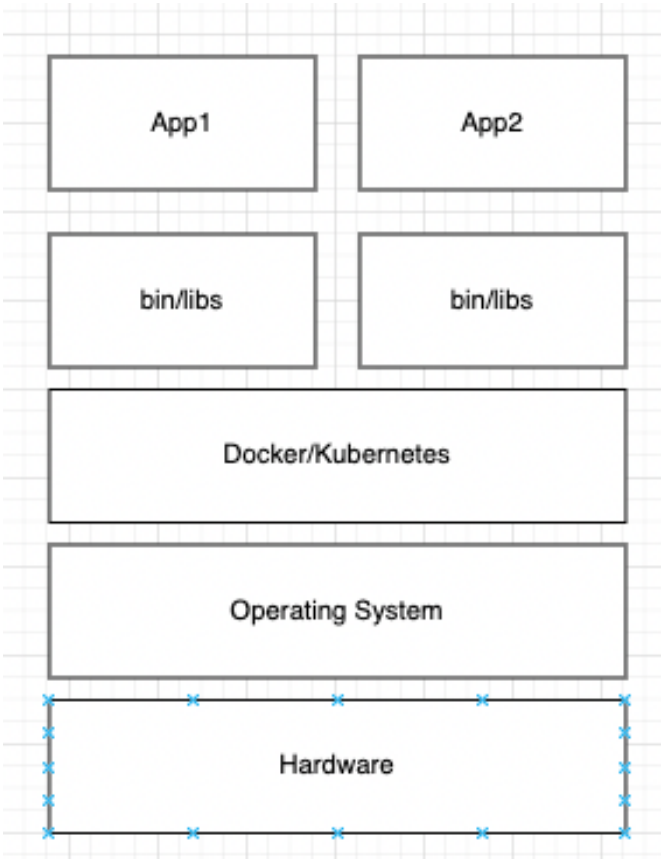


Рисунок 4.2 – Розгортання програми на кластері

Наступним кроком, після розгляду важливості використання контейнерів та їх принципів роботи, необхідно розглянути механізми Kubernetes, які забезпечують повний захист системи від атак та загроз ззовні. На рис. 4.3 зображено складники Kubernetes.

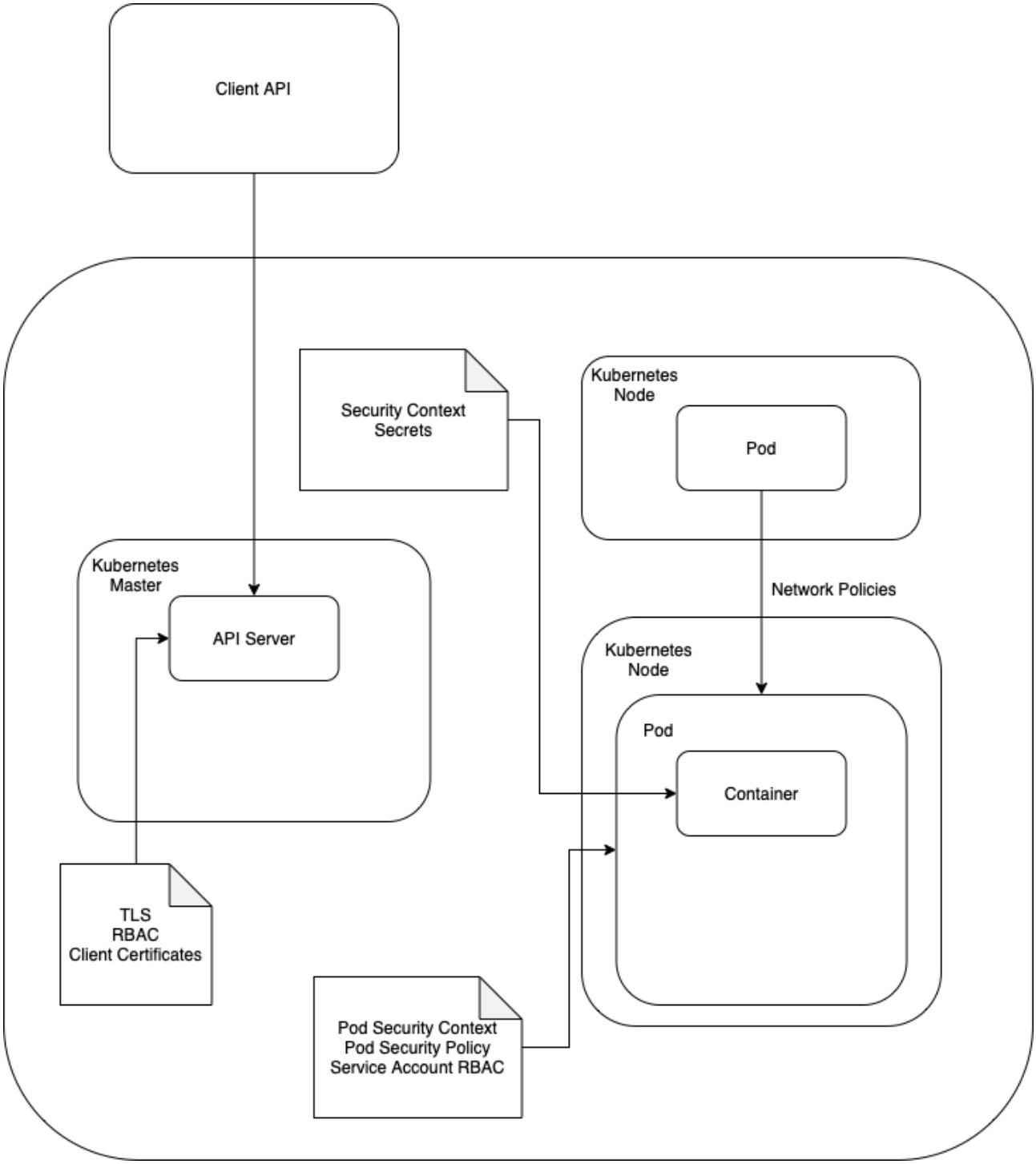


Рисунок 4.3 – Механізми захисту в Kubernetes

Розглянемо кожен із цих механізмів:

- Container. Це готовий до запуску програмний пакет, що містить все необхідне для запуску програми: код та будь-яке необхідне середовище виконання, програми та системні бібліотеки, а також значення за замовчанням для всіх основних параметрів;
- Pod. Це група з одного або декількох контейнерів із загальним сховищем та мережевими ресурсами, а також специфікацією того, як запускати контейнери;
- Kubernetes Node. Kubernetes запускає робоче навантаження, розміщуючи контейнери у модулях для запуску на вузлах. Вузол може бути віртуальною або фізичною машиною, залежно від кластера. Кожен вузол управляється площиною керування та містить служби, необхідні для запуску модулів;
- Network Policies. За замовчуванням весь вихідний та вхідний трафік дозволено всередині кластера, що робить його модулі неізольованими. Застосування політики мережі дозволяє уникнути такої ситуації, дозволяючи відхиляти невирішений трафік для певних модулів. Як приклад можна розглянути клієнт-серверну архітектуру програми. Клієнтські модулі можуть надсилати трафік лише на модулі серверної частини. Якщо зловмисник отримати доступ до клієнтських модулів, він не зможе отримати доступ до бази даних ніяким чином;
- Security Context. Контекст безпеки служить для визначення привілеїв, що використовуються у модулях кластера чи контейнерах. Для різних модулів можна використовувати різні контексти. Прикладом привілеїв може бути право доступу до зовнішніх даних або запуск процесів у привілейованому режимі;
- Secrets. Основне завдання секретів полягає у зберіганні конфіденційних даних: SSH ключі, логіни, паролі, різні види токенів тощо. Секрети дозволяють досягти гнучкості життєвого циклу контейнерів, а також простежити використання конфіденційних даних. Все це допомагає знизити можливість розкриття даних зловмисниками;

- RBAC. Даний механізм визначає допустимі дії користувача (що він може виконувати у різних кластерах та просторах імен). Для його реалізації використовуються ролі суб'єктів (користувач, група, обліковий запис служби);
- Pod Security Policy. Цей механізм контролює дії пода, які він може виконувати, а також визначає ресурси, до яких він може отримати доступ. Політики безпеки пода визначають умови, без яких під не може запуститися в кластері.

4.3 Проектування бекенду

Далі розглянемо структуру бекенд-частини системи. Вона містить у собі бізнес-логіку необхідну до виконання поставлених завдань застосування. На рис. 4.4 наведено структуру монолітної реалізації бекенда.

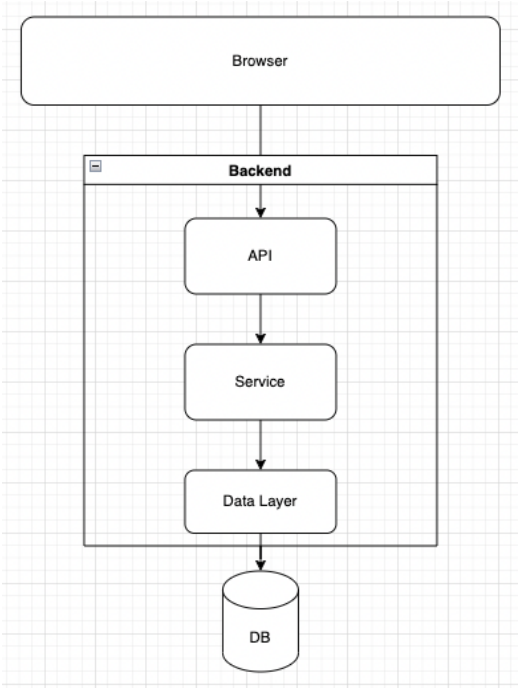


Рисунок 4.4 – Структура монолітної архітектури

Як видно, бекенд-частина являє собою один артефакт, який містить у собі всі необхідні модулі та бізнес-логіку (крім бази даних, вона запускається окремо від програми на фізичному сервері).

Після процесу міграції програми з монолітної архітектури на мікросервісну його структура буде виглядати так. Результат показаний на рис. 4.5.

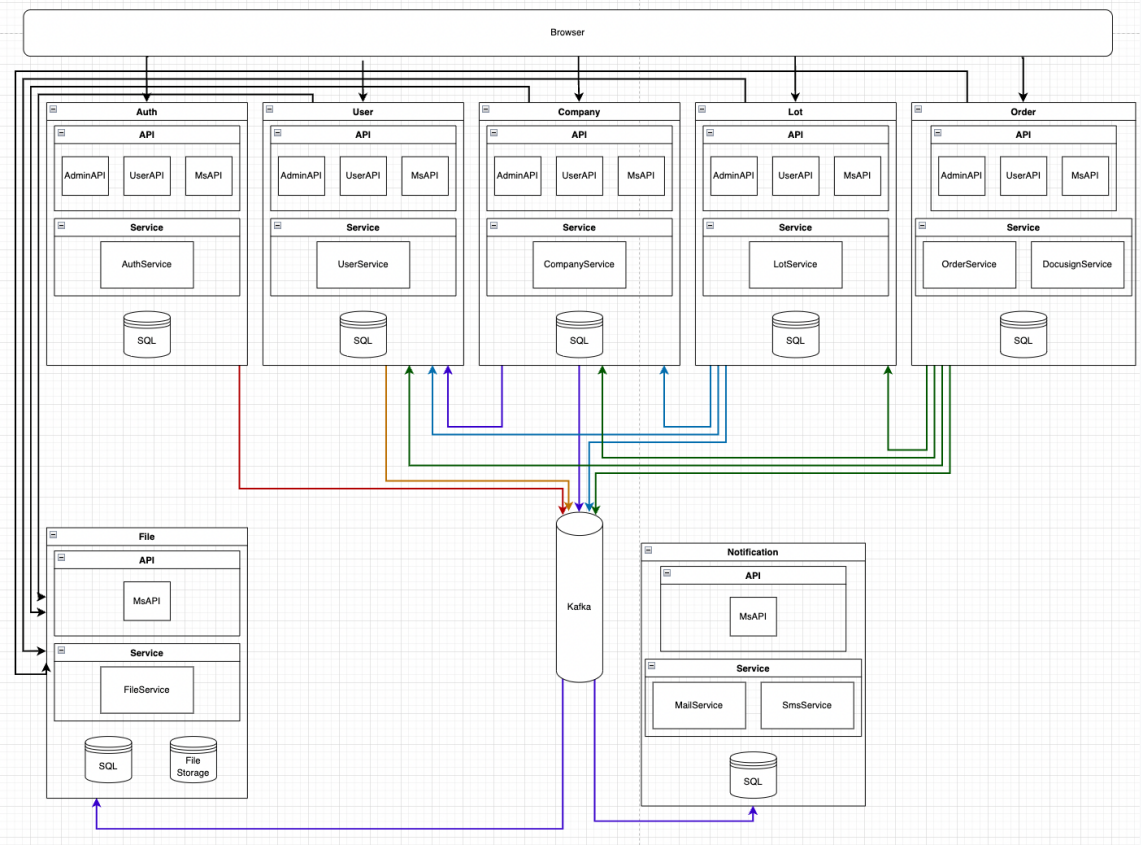


Рисунок 4.5 – Структура мікросервисної архітектури

Виглядає трохи масивно, чи не так? Давайте розберемося у змінах.

Замість однієї програми, що запускається, запускається цілих сім програм (мікросервісів).

Крім монолітного стилю при проектуванні серверної частини використовується багаторівнева архітектура. Основна ідея полягає в тому, щоб розбити застосування на компоненти (шари), які виконують чітко визначену частину роботи. Зокрема, виходить три окремих шари:

- шар контролерів. Відповідає за обробку вхідних запитів від клієнтської частини і віддачі їй відповідей;
- шар бізнес-логіки. Цей шар реалізує всю бізнес-логіку програми;
- шар доступу до даних. Даний шар здійснює запис інформації в базу даних і отримання інформації з неї ж;

Кожен із мікросервісів інкапсулює свою бізнес-логіку. Структура мікросервісу складається з трьох основних елементів: API, Service, Database (SQL).

API модуль мікросервісу визначає інтерфейси взаємодії з ним. Так як не всі інтерфейси доступні всім користувачам, було вирішено розбити модуль API на три підмодулі за типом взаємодії з мікросервісом: AdminAPI, UserAPI, MsAPI.

- AdminAPI – інтерфейс взаємодії між адміністратором, а також модератором та мікросервісом;
- UserAPI – інтерфейс взаємодії між користувачем (покупцем або продавцем) та мікросервісом;
- MsAPI – інтерфейс взаємодії між мікросервісами (виклики мікросервісом усередині кластера, які недоступні ззовні).

Service модуль відповідає за бізнес-логіку програми та обробку вхідних та вихідних даних. В основному цей тип модулів містить лише один сервіс, крім мікросервісу Order. У ньому міститься 2 сервіси OrderService та DocusignService. Можна запитати, чому сервіс електронного підпису (DocusignService) не винесений в окремий мікросервіс. Він сильно пов'язаний з предметною областю замовлень, тому немає необхідності у створенні ще одного мікросервісу.

Заключним модулем є бази даних. Усі мікросервіси використовують власні реляційні бази даних (Postgres). Окремо стоїть мікросервіс File. Тільки він відповідає за збереження файлів на файловій системі для додаткового захисту інформації.

4.4 Проектування бази даних

Заключним кроком у проектуванні цілісної системи є проектування бази даних. Вона необхідна, як і з назви, для зберігання даних користувачів.

Оскільки вибір ліг на реляційну базу даних, а не на нереляційну, то нижче буде представлено опис реляційних таблиць (табл. 4.1 – 4.13), які використовуються для зберігання різних типів даних користувачів системи.

Таблиця 4.1 – Структура таблиці account

Назва	Тип	Ключ	Індекс	Опис
id	text	первинний	-	Айді користувача
email	text	-	+	Пошта користувача
password	text	-	-	Пароль користувача у зашифрованому вигляді
email_confirmed	boolean	-	-	Пошта підтверджена
account_locked	boolean	-	-	Аккаунт заблокований
role_id	integer	зовнішній	-	Айді ролі користувача

Таблиця 4.2 – Структура таблиці role

Назва	Тип	Ключ	Індекс	Опис
id	text	первинний	+	Айді ролі
role	text	-	-	Назва ролі

Таблиця 4.3 – Структура таблиці user

Назва	Тип	Ключ	Індекс	Опис
account_id	text	первинний	+	Айді акаунту
first_name	text	-	-	Ім'я користувача
last_name	text	-	-	Прізвище користувача
phone	text	-	-	Телефонний номер користувача
country_code	text	-	-	Код країни користувача

Таблиця 4.4 – Структура таблиці company

Назва	Тип	Ключ	Індекс	Опис
account_id	text	первинний	+	Айді акаунту
name	text	-	+	Назва компанії
address_id	text	зовнішній	-	Айді адреси
email	text	-	-	Пошта компанії
reg_number	text	-	-	Реєстраційний номер компанії
auth_person_id	text	зовнішній	-	Айді контактної особи компанії

Продовження таблиці 4.4

Назва	Тип	Ключ	Індекс	Опис
bank_requisites_id	text	зовнішній	-	Айді банківських реквізитів
phone	text	-	-	Телефонний номер компанії
description	text	-	-	Опис компанії

Таблиця 4.5 – Структура таблиці address

Назва	Тип	Ключ	Індекс	Опис
id	text	первинний	+	Айді
zip	text	-	-	ZIP
address	text	-	-	Адреса
city	text	-	-	Місто
country_code	text	-	-	Код країни

Таблиця 4.6 – Структура таблиці auth_person

Назва	Тип	Ключ	Індекс	Опис
id	text	первинний	+	Айді
first_name	text	-	-	Ім'я контактної особи
last_name	text	-	-	Прізвище контактної особи

Таблиця 4.7 – Структура таблиці bank_requisites

Назва	Тип	Ключ	Індекс	Опис
id	text	первинний	+	Айді
iban	text	-	-	IBAN
Correspondent _account	text	-	-	Кореспондентський рахунок
correspondent_ swift_code	text	-	-	Свіфт-код кореспондента
swift_code	text	-	-	Свіфт-код

Таблиця 4.8 – Структура таблиці lot

Назва	Тип	Ключ	Індекс	Опис
id	text	первинний	+	Айді
account_id	text	зовнішній	+	Айді акаунту
culture_id	text	зовнішній	-	Айді культури
unit	text	-	-	Одиниця виміру
country_code	text	-	-	Код країни
currency	text	-	-	Валюта
name	text	-	+	Назва лоту
amount	double precision	-	-	Кількість
price	double precision	-	-	Ціна
description	text	-	-	Опис лоту

Таблиця 4.9 – Структура таблиці culture

Назва	Тип	Ключ	Індекс	Опис
id	text	первинний	+	Айді
name	text	-	-	Назва
culture_group_id	text	зовнішній	-	Айді культурної групи

Таблиця 4.10 – Структура таблиці culture_group

Назва	Тип	Ключ	Індекс	Опис
id	text	первинний	+	Айді
name	text	-	-	Назва
product_type_id	text	зовнішній	-	Айді типу продукту

Таблиця 4.11 – Структура таблиці product_type

Назва	Тип	Ключ	Індекс	Опис
id	text	первинний	+	Айді
name	text	-	-	Назва

Таблиця 4.12 – Структура таблиці favorite_lot

Назва	Тип	Ключ	Індекс	Опис
id	text	первинний	-	Айді
account_id	text	зовнішній	+	Айді акаунту
lot_id	text	зовнішній	-	Айді лота

Таблиця 4.13 – Структура таблиці Order

Назва	Тип	Ключ	Індекс	Опис
id	text	первинний	+	Айді
account_id	text	зовнішній	+	Айді акаунту
lot_id	text	зовнішній	-	Айді лота
amount	double precision	-	-	Кількість
status	text	-	-	Статус замовлення

4.5 Висновки до розділу

У четвертому розділі було розглянуто ухвалені архітектурні рішення для інфраструктури та бекенд-частини системи. Ці рішення було аргументовано. Наприкінці розділу було наведено структуру бази даних як реляційних таблиць.

5 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

5.1 Технології та бібліотеки

Під час розробки веб-застосування використовувалися такі технології та інструменти:

- Java;
- Maven;
- Spring Boot;
- Spring Security;
- Spring Data;
- Postgres;
- Flyway;
- Lombok;
- Docker;
- Kubernetes;
- Helm;

Java — об'єктно-орієнтована мова програмування, яка широко використовується при створенні надійних та безпечних веб-додатків, незалежно від їхньої архітектури. Має велику кількість бібліотек і фреймворку, які забезпечують високу швидкість розробки, оскільки немає необхідності в реалізації своїх власних механізмів для вирішення загальновідомих завдань (крім моментів чіткої необхідності в цьому). Java - захищена мова програмування, оскільки вона не використовує явні покажчики. Крім того, Java-програми запускаються у пісочниці віртуальної машини. JRE також надає завантажувач класів, який використовується для динамічного завантаження класу JVM [16]. Він відокремлює пакети класів локальної файлової системи від тих, що імпортуються з мережі. Java - надійна мова програмування, оскільки вона використовує суворе керування пам'яттю. Ми також можемо обробляти винятки через код Java. Крім того, ми можемо використовувати перевірку типів, щоб зробити наш код безпечнішим. Він не надає явних

покажчиків, тому програміст не може отримати доступу до пам'яті безпосередньо з коду.

Для створення проекту на Java достатньо встановити JDK для платформи, де розробляється проект. JDK – це комплект необхідних інструментів для створення програм Java. До нього належать компілятор, стандартні бібліотеки, документація, а також JRE (середовище виконання для Java). На момент розробки веб-застосунку найостаннішою актуальною версією Java є Java 18. Дистрибутив можна вибрати будь-який, оскільки відмінності між ними є тільки в плані комерційної підтримки, яка, по суті, не потрібна. Вибір ліг на дистрибутив від Amazon під назвою Java 18 Amazon Corretto. Вона і використовуватиметься у проекті.

Maven. Це інструмент управління проектами написаних на мові Java. Під управлінням мається на увазі опис об'єктної моделі проекту, дотримання стандартів розробки, життєвий цикл проекту, управління залежностями, впровадженням плагінів для виконання додаткових кроків на етапах життєвого циклу проекту.

Для використання Maven необхідно зробити дві речі:

- Знайти версію Maven, яка відповідає версії Java, та встановити її. Оскільки використовується Java 18 Amazon Corretto, то версія Maven буде 3.8.6 Amazon Corretto.
- Створити файл pom.xml у кореневій папці проекту з обов'язковими полями, до яких належить інформація про проект (версія, ідентифікатори назви та групи, версія моделі).

У pom.xml слід додати залежності без яких не вдасться створити проект: spring-boot-starter, spring-boot-starter-web, Lombok, spring-boot-starter-data-jpa, flyway-core, postrgresql. Оскільки мікросервісів буде кілька, то в кожному з них буде свій файл pom.xml. Внаслідок чого, кожен мікросервіс можна гнучко налаштовувати залежності, що використовуються, та їх версії.

Spring Boot. Де-факто найкращий фреймворк, що набагато спрощує розробку Java-додатків. Він являє собою кістяк додатку, до якого додаються різні залежності для реалізації будь-якого аспекту поведінки системи, чи то робота з базою даних або

брокером повідомлень, створення веб-додатку, реалізації взаємодії між мікросервісами.

Перехід від простого Java додаток до програми Spring Boot виконується дуже легко. Для початку потрібно додати залежність spring-boot-starter в pom.xml файл. Після цього в папку, де лежать вихідні джерела (зазвичай це папка src/main/java/{groupId}.{projectName}), необхідно додати Java-клас з анотацією @SpringBootApplication. От і все. Ця інструкція просканує компоненти проекту, додасть їх у контекст і проведе автоконфігурацію.

Spring Security. Фреймворк, що надає механізми автентифікації та авторизації клієнтів. Його плюсом є повна кастомізація (розширюваність) функціоналу, що дозволяє реалізовувати власні елементи захисту. Як приклад, двофакторна автентифікація або автентифікація за допомогою OAuth2.

Spring Data. Фреймворк для доступу до даних. Має велику кількість баз даних, що підтримуються, починаючи від реляційних і неряляційних, закінчуючи хмарними серверами.

Щоб ми могли отримати доступ до бази даних з нашої програми, необхідно додати дві залежності: spring-boot-starter-data-jpa та postgresql. Перша залежність надає код, друга є драйвером бази даних. Драйвера виконують такі завдання: відкриття з'єднань сокетів, надсилання запитів, отримання подій з бази.

Додавши необхідні залежності, можна розпочати написання коду. Для взаємодії з базою нам знадобиться два Java-класи: Repository та Entity. Repository успадковується від інтерфейсу JpaRepository. Це дозволить Spring Data знайти цей інтерфейс та автоматично створити для нього реалізацію. Розширюючи інтерфейс, ми отримуємо найактуальніші методи CRUD для стандартного доступу до даних. Entity – сутність, що представляє дані, які можна зберегти у базі даних. Сутність є таблицю, що зберігається у базі даних. Кожен екземпляр об'єкта є рядком у таблиці. Для того, щоб Java-клас сприймався як JPA сутність, необхідно позначити клас наступними інструкціями @Entity і @Table(name = “table_name”).

Postgres є об'єктно-реляційною системою управління базами даних (ORDBMS) з упором на розширюваність і відповідність стандартам. Як сервер бази даних, його

основна функція полягає в безпечному зберіганні даних і підтримці кращих практик, а також у подальшому вилученні їх на запит інших програмних додатків, будь то програми на тому ж комп'ютері або ті, які працюють на іншому комп'ютері в мережі (включаючи Інтернет). Він може справлятися з робочими навантаженнями, починаючи від невеликих одномашинних додатків і закінчуючи великими інтернет-додатками з безліччю користувачів, що одночасно працюють.

Flyway. Не завжди з першого разу виходить продумати базу даних так, щоб потім не довелося вносити зміни. Більшість часу проводиться ітеративний процес модифікації нашої схеми у міру зміни вимог. І тут на допомогу приходить Flyway. Це інструмент міграції схем баз даних. Його великим плюсом є його гарна інтеграція зі Spring Boot. Все на все потрібно додати в pom.xml залежність "flyway-core" і шляхом "src/main/resources/db/migration" додати файл з міграцією. Ще не потрібно забути про зміни у файлі application.yml. Для повноцінної роботи Flyway потрібно додати властивості, які використовуються для з'єднання Flyway та бази даних.

Lombok. Lombok є Java-бібліотекою, яка дозволяє скоротити стандартний код Java-класів. Замість такого коду застосовують різні види анотацій Lombok, які здійснюють генерацію коду під час компіляції. Використовуючи Lombok, відпадає необхідність написання методів гетерів, сеттерів, конструкторів, методів equals, hashCode.

Docker. Це технологія з відкритим кодом, яка використовується для створення імеджів мікросервісів. Кожен мікросервіс містить файл під назвою Dockerfile. Цей файл описує кроки, необхідні для складання та запуску мікросервісу.

Kubernetes. Імеджі, створені за допомогою Docker, використовуватимуться Kubernetes. Він дозволяє запускати контейнери та робочі навантаження Docker, а також допомагає вирішувати деякі операційні складності під час переходу на масштабування кількох контейнерів, розгорнутих на кількох серверах. Важливою частиною Kubernetes є конфігураційні файли. Вони описують процес запуску контейнерів, їх поведінку в кластері, а також поза ним. Зі збільшенням кількості

мікросервісів зростає кількість файлів конфігурацій. Внаслідок чого їх важко підтримувати та застосовувати.

Helm. Інструмент Helm допомагає адмініструвати програми в Kubernetes — Helm чарти допомагають визначати, встановлювати та оновлювати навіть найскладніші програми Kubernetes. Діаграми легко створювати, верифікувати, ділитися ними та публікувати. Helm нативно підтримує Kubernetes, а це означає, що вам не потрібно писати складні синтаксичні файли чи щось інше, щоб почати використовувати Helm. Просто перемістіть файли шаблону в нову діаграму, і все готово.

5.2 Висновки до розділу

У п'ятому розділі було проведено огляд технологій, фреймворк та бібліотек, на основі яких розробляється система. Усі вони були обрані з урахуванням таких аспектів:

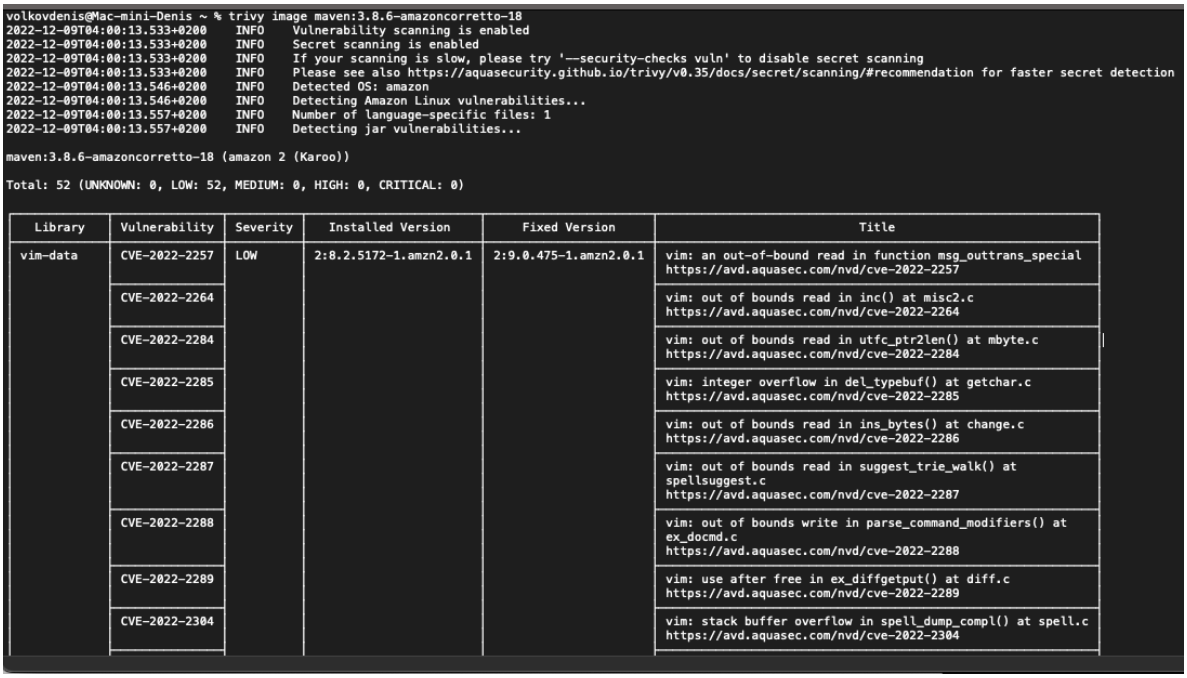
- швидкість розробки;
- велика спільнота та документація;
- дотримання аспектів безпеки;
- надійність.

6 ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

6.1 Trivy

Trivy – це інструмент сканування з відкритим вихідним кодом, який є чудовим вибором для впровадження статичного тестування безпеки програм. Він забезпечує комплексне виявлення вразливостей для операційних систем, інфраструктури у вигляді файлів коду та мовних пакетів, а також використовується для виявлення проблем із конфігурацією. Бібліотека Trivy автоматично оновлює свою базу даних вразливостей, таким чином вона завжди актуальна.

Як видно за результатами виконання перевірки Trivy, представлених на рис. 6.1 і рис. 6.2, Docker-образи, які використовуються як основа мікросервісів, містять лише помилки з рівнем LOW, що показує, що дані образи можна використовувати. Найголовніше, що немає вразливостей із рівнем вище LOW (до таких рівнів відносяться MEDIUM, HIGH, CRITICAL).



```
volkovdenis@Mac-mini-Denis ~ % trivy image maven:3.8.6-amazoncorretto-18
2022-12-09T04:00:13.533+0200    INFO    Vulnerability scanning is enabled
2022-12-09T04:00:13.533+0200    INFO    Secret scanning is enabled
2022-12-09T04:00:13.533+0200    INFO    If your scanning is slow, please try '--security-checks vuln' to disable secret scanning
2022-12-09T04:00:13.533+0200    INFO    Please see also https://aquasecurity.github.io/trivy/v0.35/docs/secret/scanning/#recommendation for faster secret detection
2022-12-09T04:00:13.546+0200    INFO    Detected OS: amazon
2022-12-09T04:00:13.546+0200    INFO    Detecting Amazon Linux vulnerabilities...
2022-12-09T04:00:13.557+0200    INFO    Number of language-specific files: 1
2022-12-09T04:00:13.557+0200    INFO    Detecting jar vulnerabilities...

maven:3.8.6-amazoncorretto-18 (amazon 2 (Karoo))

Total: 52 (UNKNOWN: 0, LOW: 52, MEDIUM: 0, HIGH: 0, CRITICAL: 0)
```

Library	Vulnerability	Severity	Installed Version	Fixed Version	Title
vim-data	CVE-2022-2257	LOW	2:8.2.5172-1.amzn2.0.1	2:9.0.475-1.amzn2.0.1	vim: an out-of-bound read in function msg_outtrans_special https://avd.aquasec.com/nvd/cve-2022-2257
	CVE-2022-2264				vim: out of bounds read in inc() at misc2.c https://avd.aquasec.com/nvd/cve-2022-2264
	CVE-2022-2284				vim: out of bounds read in utfc_ptr2len() at mbyte.c https://avd.aquasec.com/nvd/cve-2022-2284
	CVE-2022-2285				vim: integer overflow in del_typebuf() at getchar.c https://avd.aquasec.com/nvd/cve-2022-2285
	CVE-2022-2286				vim: out of bounds read in ins_bytes() at change.c https://avd.aquasec.com/nvd/cve-2022-2286
	CVE-2022-2287				vim: out of bounds read in suggest_trie_walk() at spellsuggest.c https://avd.aquasec.com/nvd/cve-2022-2287
	CVE-2022-2288				vim: out of bounds write in parse_command_modifiers() at ex_docmd.c https://avd.aquasec.com/nvd/cve-2022-2288
	CVE-2022-2289				vim: use after free in ex_diffgetput() at diff.c https://avd.aquasec.com/nvd/cve-2022-2289
	CVE-2022-2304				vim: stack buffer overflow in spell_dump_compl() at spell.c https://avd.aquasec.com/nvd/cve-2022-2304

Рисунок 6.1 – Тестування за допомогою Trivy

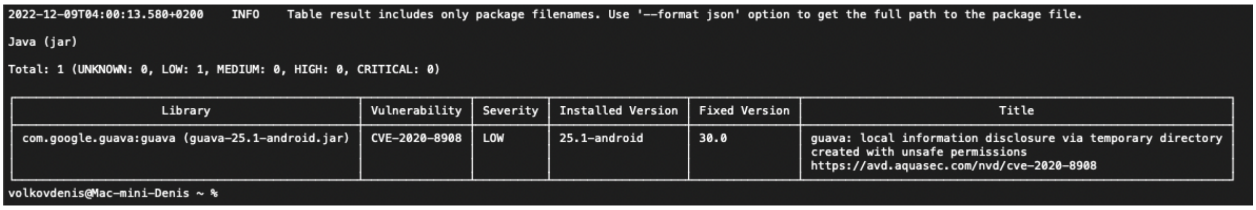


Рисунок 6.2 – Тестування за допомогою Trivy

6.2 Clair

Clair — інструмент статичного сканування вразливостей з відкритим вихідним кодом для контейнерів, розгорнутих у кластері. Інструмент має кілька режимів розгортання. Clair підтримує REST API та надає звіти про сканування у форматі HTML.

На рис. 6.3 показано результат перевірки Clair. Дана перевірка знайшла 2 вразливості рівня LOW, які не мають особливої небезпеки.

Порівнявши Clair і Trivy видно, що ці дві утиліти використовують різні бази даних вразливостей (на це вказує різна кількість знайдених вразливостей).

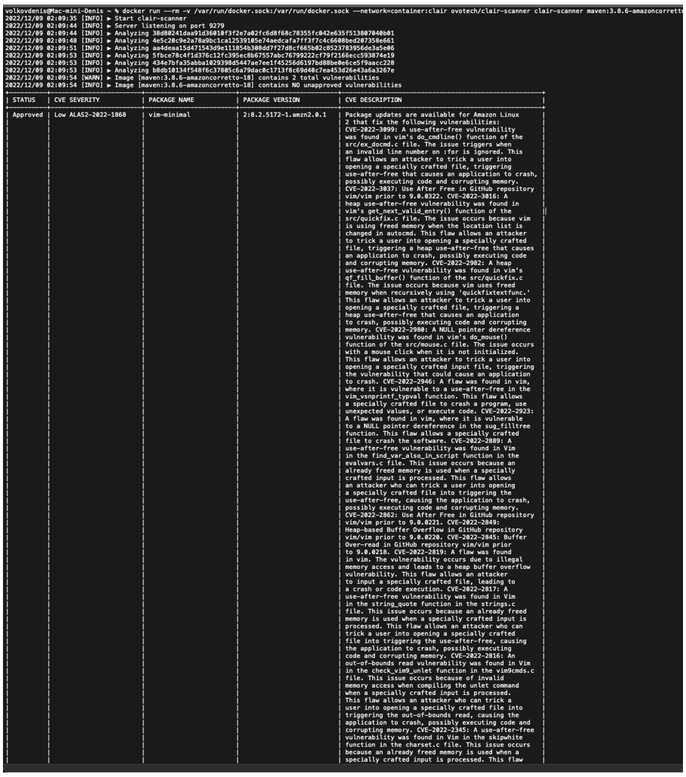


Рисунок 6.3 – Тестування за допомогою Clair

6.3 Kube-bench

Kube-bench — це інструмент із відкритим вихідним кодом, який перевіряє, чи оптимально розгортається Kubernetes відповідно до CIS Kubernetes Benchmark, який містить набір кращих практик безпеки Kubernetes. Таким чином, kube-bench найкраще підходить для сканування лише з метою порівняльного аналізу CIS.

На рис. 6.4–6.6 показані результати перевірки кластера інструментом Kube-bench.

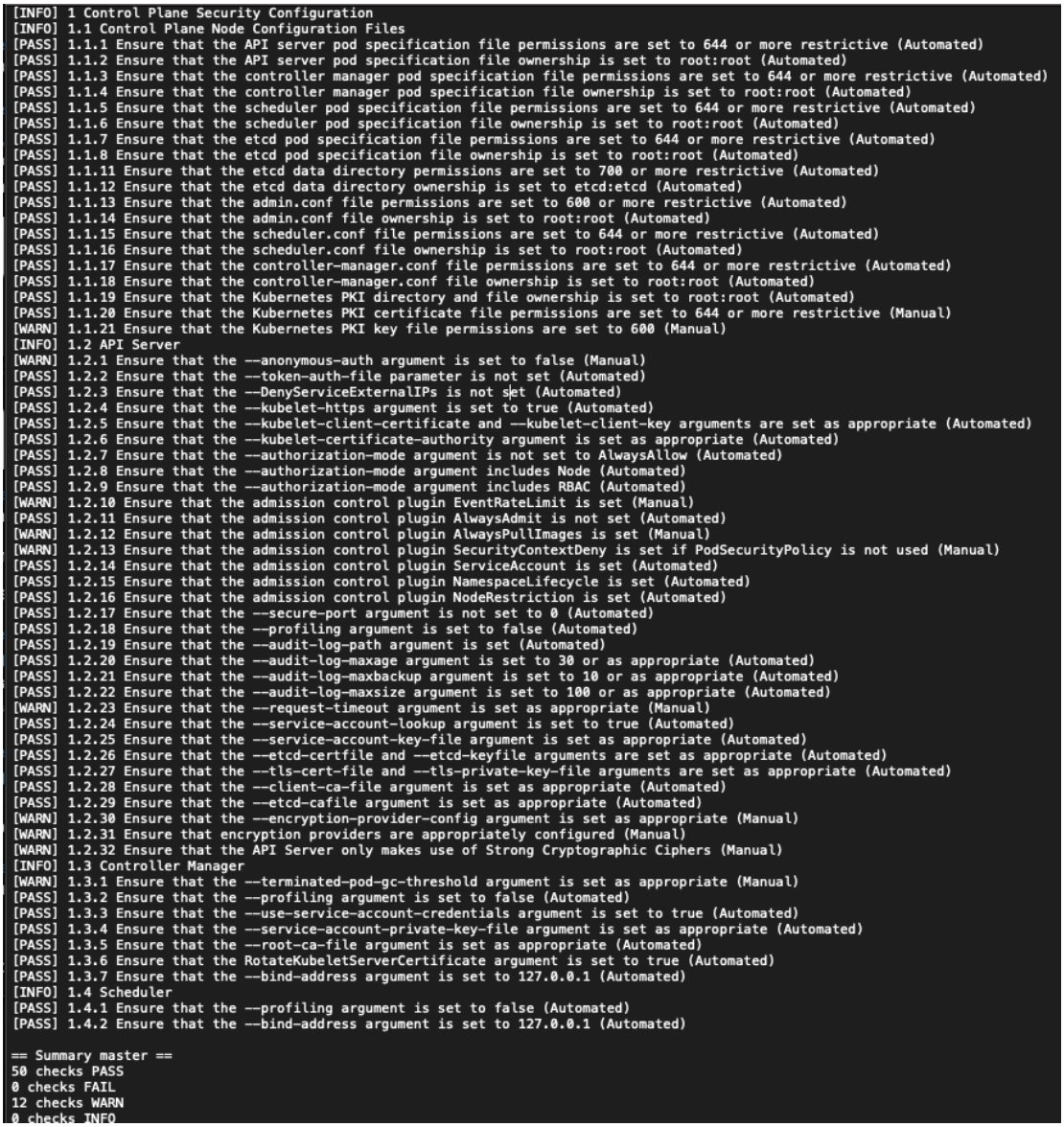


Рисунок 6.4 – Тестування за допомогою kube-bench

```
[INFO] 2 Etcd Node Configuration
[INFO] 2 Etcd Node Configuration
[PASS] 2.1 Ensure that the --cert-file and --key-file arguments are set as appropriate (Automated)
[PASS] 2.2 Ensure that the --client-cert-auth argument is set to true (Automated)
[PASS] 2.3 Ensure that the --auto-tls argument is not set to true (Automated)
[PASS] 2.4 Ensure that the --peer-cert-file and --peer-key-file arguments are set as appropriate (Automated)
[PASS] 2.5 Ensure that the --peer-client-cert-auth argument is set to true (Automated)
[PASS] 2.6 Ensure that the --peer-auto-tls argument is not set to true (Automated)
[PASS] 2.7 Ensure that a unique Certificate Authority is used for etcd (Manual)

== Summary etcd ==
7 checks PASS
0 checks FAIL
0 checks WARN
0 checks INFO

[INFO] 3 Control Plane Configuration
[INFO] 3.1 Authentication and Authorization
[PASS] 3.1.1 Client certificate authentication should not be used for users (Manual)
[INFO] 3.2 Logging
[PASS] 3.2.1 Ensure that a minimal audit policy is created (Manual)
[PASS] 3.2.2 Ensure that the audit policy covers key security concerns (Manual)

== Summary controlplane ==
3 checks PASS
0 checks FAIL
0 checks WARN
0 checks INFO
```

Рисунок 6.5 – Тестування за допомогою kube-bench

```
[INFO] 4 Worker Node Security Configuration
[INFO] 4.1 Worker Node Configuration Files
[PASS] 4.1.1 Ensure that the kubelet service file permissions are set to 644 or more restrictive (Automated)
[PASS] 4.1.2 Ensure that the kubelet service file ownership is set to root:root (Automated)
[PASS] 4.1.3 If proxy kubeconfig file exists ensure permissions are set to 644 or more restrictive (Manual)
[PASS] 4.1.4 If proxy kubeconfig file exists ensure ownership is set to root:root (Manual)
[PASS] 4.1.5 Ensure that the --kubeconfig kubelet.conf file permissions are set to 644 or more restrictive (Automated)
[PASS] 4.1.6 Ensure that the --kubeconfig kubelet.conf file ownership is set to root:root (Automated)
[PASS] 4.1.7 Ensure that the certificate authorities file permissions are set to 644 or more restrictive (Manual)
[PASS] 4.1.8 Ensure that the client certificate authorities file ownership is set to root:root (Manual)
[PASS] 4.1.9 Ensure that the kubelet --config configuration file has permissions set to 644 or more restrictive (Automated)
[PASS] 4.1.10 Ensure that the kubelet --config configuration file ownership is set to root:root (Automated)
[INFO] 4.2 Kubelet
[PASS] 4.2.1 Ensure that the --anonymous-auth argument is set to false (Automated)
[PASS] 4.2.2 Ensure that the --authorization-mode argument is not set to AlwaysAllow (Automated)
[PASS] 4.2.3 Ensure that the --client-ca-file argument is set as appropriate (Automated)
[PASS] 4.2.4 Ensure that the --read-only-port argument is set to 0 (Manual)
[PASS] 4.2.5 Ensure that the --streaming-connection-idle-timeout argument is not set to 0 (Manual)
[PASS] 4.2.6 Ensure that the --protect-kernel-defaults argument is set to true (Automated)
[PASS] 4.2.7 Ensure that the --make-iptables-util-chains argument is set to true (Automated)
[WARN] 4.2.8 Ensure that the --hostname-override argument is not set (Manual)
[WARN] 4.2.9 Ensure that the --event-qps argument is set to 0 or a level which ensures appropriate event capture (Manual)
[WARN] 4.2.10 Ensure that the --tls-cert-file and --tls-private-key-file arguments are set as appropriate (Manual)
[PASS] 4.2.11 Ensure that the --rotate-certificates argument is not set to false (Automated)
[PASS] 4.2.12 Verify that the RotateKubeletServerCertificate argument is set to true (Manual)
[WARN] 4.2.13 Ensure that the Kubelet only makes use of Strong Cryptographic Ciphers (Manual)

== Summary node ==
19 checks PASS
0 checks FAIL
4 checks WARN
0 checks INFO

[INFO] 5 Kubernetes Policies
[INFO] 5.1 RBAC and Service Accounts
[WARN] 5.1.1 Ensure that the cluster-admin role is only used where required (Manual)
[WARN] 5.1.2 Minimize access to secrets (Manual)
[WARN] 5.1.3 Minimize wildcard use in Roles and ClusterRoles (Manual)
[WARN] 5.1.4 Minimize access to create pods (Manual)
[WARN] 5.1.5 Ensure that default service accounts are not actively used. (Manual)
[WARN] 5.1.6 Ensure that Service Account Tokens are only mounted where necessary (Manual)
[WARN] 5.1.7 Avoid use of system:masters group (Manual)
[WARN] 5.1.8 Limit use of the Bind, Impersonate and Escalate permissions in the Kubernetes cluster (Manual)
[INFO] 5.2 Pod Security Standards
[WARN] 5.2.1 Ensure that the cluster has at least one active policy control mechanism in place (Manual)
[WARN] 5.2.2 Minimize the admission of privileged containers (Manual)
[WARN] 5.2.3 Minimize the admission of containers wishing to share the host process ID namespace (Automated)
[WARN] 5.2.4 Minimize the admission of containers wishing to share the host IPC namespace (Automated)
[WARN] 5.2.5 Minimize the admission of containers wishing to share the host network namespace (Automated)
[WARN] 5.2.6 Minimize the admission of containers with allowPrivilegeEscalation (Automated)
[WARN] 5.2.7 Minimize the admission of root containers (Automated)
[WARN] 5.2.8 Minimize the admission of containers with the NET_RAW capability (Automated)
[WARN] 5.2.9 Minimize the admission of containers with added capabilities (Automated)
[WARN] 5.2.10 Minimize the admission of containers with capabilities assigned (Manual)
[WARN] 5.2.11 Minimize the admission of Windows HostProcess containers (Manual)
[WARN] 5.2.12 Minimize the admission of HostPath volumes (Manual)
[WARN] 5.2.13 Minimize the admission of containers which use HostPorts (Manual)
[INFO] 5.3 Network Policies and CNI
[WARN] 5.3.1 Ensure that the CNI in use supports NetworkPolicies (Manual)
[WARN] 5.3.2 Ensure that all Namespaces have NetworkPolicies defined (Manual)
[INFO] 5.4 Secrets Management
[WARN] 5.4.1 Prefer using Secrets as files over Secrets as environment variables (Manual)
[WARN] 5.4.2 Consider external secret storage (Manual)
[INFO] 5.5 Extensible Admission Control
[WARN] 5.5.1 Configure Image Provenance using ImagePolicyWebhook admission controller (Manual)
[INFO] 5.7 General Policies
[WARN] 5.7.1 Create administrative boundaries between resources using namespaces (Manual)
[WARN] 5.7.2 Ensure that the seccomp profile is set to docker/default in your Pod definitions (Manual)
[WARN] 5.7.3 Apply SecurityContext to your Pods and Containers (Manual)
[WARN] 5.7.4 The default namespace should not be used (Manual)

== Summary policies ==
0 checks PASS
0 checks FAIL
30 checks WARN
0 checks INFO

== Summary total ==
76 checks PASS
0 checks FAIL
49 checks WARN
0 checks INFO
```

Рисунок 6.6 – Тестування за допомогою kube-bench

Результати перевірки показують, що безпека кластера відповідає всім практикам, що загально використовуються, оскільки немає жодного тесту, який би завершився помилкою.

6.4 Kubeaudit

Kubeaudit — це інструмент із відкритим вихідним кодом, який перевіряє кластер Kubernetes на відповідність загальним елементам керування безпекою, таким як:

- запускати без повноважень root;
- використовувати кореневу файлову систему лише для читання;
- не використати зайві можливості;
- не запускати контейнери у привілейованому режимі;

На рис. 6.7 показано результат перевірки kubeaudit. Єдиним попередженням, яке знайшло дана програма було повідомлення про те, що для деяких ресурсів не встановлені ліміти на використання процесора та пам'яті (це не є проблемою, тому що не впливає на безпеку системи).

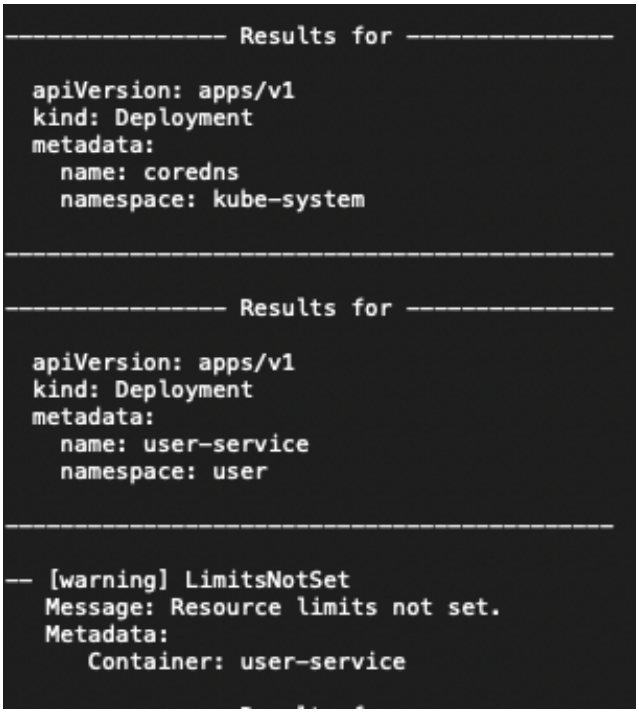
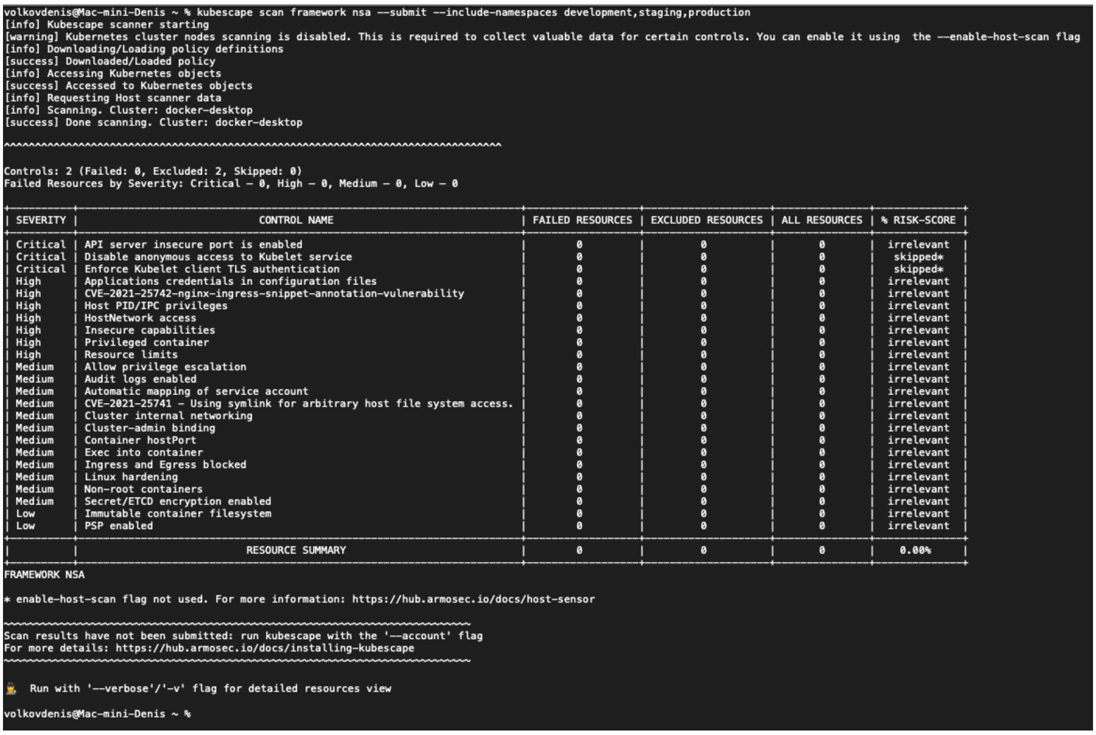


Рисунок 6.7 – Тестування за допомогою kubeaudit

6.5 Kubescape

Kubescape — це інструмент із відкритим вихідним кодом, який перевіряє, чи розгорнуть Kubernetes відповідно до всіх основних нормативних документів, включаючи NSA-CISA і MITRE ATT&CK, а також передовими практиками DevSecOps.

На рис. 6.8 представлено результат сканування.



```
volkovdenis@Mac-mini-Denis ~ % kubescape scan framework nsa --submit --include-namespaces development,staging,production
[info] Kubescape scanner starting
[warning] Kubernetes cluster nodes scanning is disabled. This is required to collect valuable data for certain controls. You can enable it using  the --enable-host-scan flag
[info] Downloading/Loading policy definitions
[success] Downloaded/Loaded policy
[info] Accessing Kubernetes objects
[success] Accessed to Kubernetes objects
[info] Requesting Host scanner data
[info] Scanning. Cluster: docker-desktop
[success] Done scanning. Cluster: docker-desktop

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Controls: 2 (Failed: 0, Excluded: 2, Skipped: 0)
Failed Resources by Severity: Critical - 0, High - 0, Medium - 0, Low - 0
```

SEVERITY	CONTROL NAME	FAILED RESOURCES	EXCLUDED RESOURCES	ALL RESOURCES	% RISK-SCORE
Critical	API server insecure port is enabled	0	0	0	irrelevant
Critical	Disable anonymous access to Kubelet service	0	0	0	skipped*
Critical	Enforce Kubelet client TLS authentication	0	0	0	skipped*
High	Applications credentials in configuration files	0	0	0	irrelevant
High	CVE-2021-25742-nginx-ingress-snippet-annotation-vulnerability	0	0	0	irrelevant
High	Host PID/IPC privileges	0	0	0	irrelevant
High	HostNetwork access	0	0	0	irrelevant
High	Insecure capabilities	0	0	0	irrelevant
High	Privileged container	0	0	0	irrelevant
High	Resource limits	0	0	0	irrelevant
Medium	Allow privilege escalation	0	0	0	irrelevant
Medium	Audit logs enabled	0	0	0	irrelevant
Medium	Automatic mapping of service account	0	0	0	irrelevant
Medium	CVE-2021-25741 - Using symlink for arbitrary host file system access.	0	0	0	irrelevant
Medium	Cluster internal networking	0	0	0	irrelevant
Medium	Cluster-admin binding	0	0	0	irrelevant
Medium	Container hostPort	0	0	0	irrelevant
Medium	Exec into container	0	0	0	irrelevant
Medium	Ingress and Egress blocked	0	0	0	irrelevant
Medium	Linux hardening	0	0	0	irrelevant
Medium	Non-root containers	0	0	0	irrelevant
Medium	Secret/ETCD encryption enabled	0	0	0	irrelevant
Low	Immutable container filesystem	0	0	0	irrelevant
Low	PSP enabled	0	0	0	irrelevant
	RESOURCE SUMMARY	0	0	0	0.00%

```
FRAMEWORK NSA

* enable-host-scan flag not used. For more information: https://hub.armosec.io/docs/host-sensor

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Scan results have not been submitted: run kubescape with the '--account' flag
For more details: https://hub.armosec.io/docs/installing-kubescape
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  Run with '--verbose'/'-v' flag for detailed resources view

volkovdenis@Mac-mini-Denis ~ %
```

Рисунок 6.8 – Тестування за допомогою kubescape

Жодної помилки не виявлено.

6.6 Висновки до розділу

У шостому розділі було проведено тестування механізмів безпеки системи, використовуючи різні утиліти тестування з відкритим вихідним кодом.

Дане тестування показало, що система не містить жодних критично важливих вразливостей, що свідчить про те, що цілі роботи були досягнуті.

ВИСНОВКИ

Як було зазначено у кожному розділі аналізованої роботи, безпека - ключовий чинник, який би успіх і конкурентоспроможність докладання. Ніхто не користуватиметься твоєю розробкою, якщо особисті дані, що зберігаються в ній, можуть бути легко вкрадені або скомпрометовані.

На підставі виконаної кваліфікаційної роботи магістра було досягнута мета - підвищення безпеки системи купівлі-продажу сільськогосподарської продукції шляхом реалізації механізмів захисту від найчастіших вразливостей веб-додатків, використовуючи платформу оркестрації контейнеризованих додатків.

Були вирішені такі завдання, необхідні для досягнення поставленої мети:

- проведено аналіз конкурентів;
- були визначені функціональні та нефункціональні вимоги системи;
- були виявлені та вивчені можливі вразливості системи;
- було проведено аналіз механізмів та алгоритмів захисту;
- було реалізовано систему, а також інтегровано механізми захисту;
- було проведено тестування системи з використанням інструментів тестування безпеки та наявності вразливостей у системі.

Тестування за допомогою інструментів, що найбільш використовуються (інструменти з відкритим вихідним кодом, які повсюдно використовуються у великих організаціях, що свідчить про їх значущість) показало, що в системі немає жодних критичних уразливостей. Це свідчить про те, що дані користувачів системи перебувають у повній безпеці, виключаючи будь-які можливості зловмисника отримати доступ до них або скомпроментувати.

СПИСОК ВИКОРАСТАНИХ ДЖЕРЕЛ

1. General Data Protection Regulation (GDPR) – Official Legal Text. *General Data Protection Regulation (GDPR)*. URL: https://gdpr-info.eu.
2. OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation. *OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation*. URL: https://owasp.org.
3. HTTP | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP.
4. What is a URL? - Learn web development | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL.
5. What are cookies? What are the differences between them (session vs. persistent)?. *Cisco*. URL: https://www.cisco.com/c/en/us/support/docs/security/web-security-appliance/117925-technote-csc-00.html.
6. SQL Introduction. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/sql/sql_intro.asp.
7. XPath | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/XPath.
8. LDAP.com. *LDAP.com*. URL: https://ldap.com.
9. What is CLI. *W3Schools Online Web Tutorials*. URL: https://www.w3schools.com/whatis/whatis_cli.asp.
10. Kubernetes Documentation. *Kubernetes*. URL: https://kubernetes.io/docs/home/.
11. Docker: Accelerated, Containerized Application Development. *Docker*. URL: https://www.docker.com.
12. Transport Layer Security (TLS) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Glossary/TLS.
13. Spring Security. *Spring | Home*. URL: https://spring.io/projects/spring-security.
14. Spring makes Java simple. *Spring*. URL: https://spring.io.
15. Oracle Java Technologies | Oracle. *Oracle | Cloud Applications and Cloud Platform*. URL: https://www.oracle.com/java/technologies/.
16. JVM vs. JRE vs. JDK: What's the Difference?. *IBM - Deutschland | IBM*. URL: https://www.ibm.com/cloud/blog/jvm-vs-jre-vs-jdk.

ДОДАТОК А. Лістинг програми

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface AccessCheckAfter {
    AccessActionType action();
}

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface AccessCheckBefore {
    AccessActionType action();
}

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Repeatable(value = AccessGrantContainer.class)
public @interface AccessGrant {
    AccessActionType action();
    Class<?> object();
    OwnershipType[] grantTo();
}

@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
public @interface AccessGrantContainer {
    AccessGrant[] value();
}

@Component
@Aspect
@Monitor
public class AccessControlCheckAspect {

    private Logger logger = LoggerFactory.getLogger(this.getClass());
    @Autowired
    private AccessControlManager controlManager;

    public AccessControlCheckAspect() {
        logger.info("AccessControlCheckAspect(): *** CONSTRUCT ***");
    }
```

```
    @PostConstruct
    private void init() {
        logger.info("init(): *** INIT ***  with [controlManager = {}]",
controlManager);
    }


@Around("@annotation(com.agrolot.accesscontrol.core.annotation.AccessCheckBefore)"
)
    public Object handleAccessCheckBefore(ProceedingJoinPoint pjp) throws
Throwable {
        logger.info("handleAccessCheckBefore(pjp = [{}])", pjp);
        checkArgs(pjp.getArgs(), getAnnotation(pjp.getSignature(),
AccessCheckBefore.class).action());
        return proceed(pjp);
    }


@Around("@annotation(com.agrolot.accesscontrol.core.annotation.AccessCheckAfter)")
    public Object handleAccessCheckAfter(ProceedingJoinPoint pjp) throws Throwable
{
        logger.info("handleAccessCheckAfter(pjp = [{}])", pjp);

        AccessCheckAfter annotation = getAnnotation(pjp.getSignature(),
AccessCheckAfter.class);
        Class<?> returnType = getReturnType(pjp.getSignature());
        Object methodReturnObject = proceed(pjp);
        if (Collection.class.isAssignableFrom(returnType))
            return controlManager.checkAccess((Collection<? extends Object>)
methodReturnObject, annotation.action());

        if (methodReturnObject instanceof AclValidated)
            controlManager.checkAccess((((AclValidated) methodReturnObject)),
annotation.action());

        if (returnType.isArray())
            return controlManager.checkAccess(castArray(methodReturnObject),
annotation.action());

        return controlManager.checkAccess(methodReturnObject,
annotation.action());
    }
```

```
    private Object proceed(ProceedingJoinPoint pjp) throws Throwable {
        try {
            return pjp.proceed();
        } catch (Throwable t) {
            logger.error("Can't proceed method: {}", t.getMessage(), t);
            throw t;
        }
    }

    private Class<?> getReturnType(Signature signature) {
        return ((MethodSignature) signature).getMethod().getReturnType();
    }

    private <T extends Annotation> T getAnnotation(Signature signature, Class<T>
annotationClass) {
        return ((MethodSignature)
signature).getMethod().getAnnotation(annotationClass);
    }


    private Object checkArgs(Object[] args, AccessActionType action) {
        for (Object arg : args) {
            if (Collection.class.isAssignableFrom(arg.getClass()))
                controlManager.checkAccess((Collection<? extends Object>) arg,
action);
            else if (arg instanceof Object[]) {
                controlManager.checkAccess(castArray(arg), action);
            } else
                //controlManager.checkAccess(((Object[])args)[0], action);
                controlManager.checkAccess(args[0], action);
        }
        return args;
    }

    private <T> T[] castArray(Object arrayObj) {
        return (T[]) arrayObj;
    }
}

@Component
@Aspect
@Monitor
@Configuration
```

```
@Transactional
public class AccessControlGrantAspect {

    @Autowired
    private AccessControlManager controlManager;
    @Autowired
    private AclSubjectUserProvider aclSubjectUserProvider;


@Around("@annotation(com.agrolot.accesscontrol.core.annotation.AccessGrantContaine
r)")
    public Object handleAccessGrantContainer(ProceedingJoinPoint pjp) throws
Throwable {
        MethodSignature methodSignature = (MethodSignature) pjp.getSignature();
        Method method = methodSignature.getMethod();

        return grantAccess(pjp, method,
method.getAnnotationsByType(AccessGrant.class));
    }

    @Around("@annotation(com.agrolot.accesscontrol.core.annotation.AccessGrant)")
    public Object handleAccessGrant(ProceedingJoinPoint pjp) throws Throwable {
        MethodSignature methodSignature = (MethodSignature) pjp.getSignature();
        Method method = methodSignature.getMethod();
        return grantAccess(pjp, method,
method.getAnnotationsByType(AccessGrant.class));
    }

    private Object grantAccess(ProceedingJoinPoint pjp, Method method,
AccessGrant[] annotations) throws Throwable {
        if (!ArrayUtils.isEmpty(annotations)) {
            for (AccessGrant annotation : annotations)
                controlManager.grantAccess(aclSubjectUserProvider.currentUser(),
annotation.object(), annotation.action(), annotation.grantTo());
        }
        try {
            return pjp.proceed();
        } catch (Throwable t) {
            throw t;
        }
    }
}
```

```
public class AccessControlException extends RuntimeException {


    public AccessControlException(String errorMessage, Object... params) {
        this(MessageFormatter.arrayFormat(errorMessage, params));
    }

    private AccessControlException(FormattingTuple formattingTuple) {
        super(formattingTuple.getMessage(), formattingTuple.getThrowable());
    }

}
@Data
public class AccessDenyException extends RuntimeException {

    private String field;
    private String code;
    private String message;

    public AccessDenyException(String field,String code,String message){
        this.field=field;
        this.code=code;
        this.message=message;
    }
}

@Component
@Monitor
@Service
@Transactional
@Configurable
public class AccessControlManagerImpl implements AccessControlManager {

    @Autowired
    private ThreadStorage threadStorage;
    @Autowired
    private AccessConditionBuilder accessConditionBuilder;

    private Object lock = new Object();

    private Map<Pair<Class<?>, AccessActionType>,
ArrayList<Pair<AccessControlRule, AccessCondition>>> ruleAndConditionTable() {
        String attrName = this.getClass().getSimpleName() + ".ruleTable";
```

```
        Map<Pair<Class<?>, AccessActionType>, ArrayList<Pair<AccessControlRule,
AccessCondition>>> ret = threadStorage.get(attrName);
        if (ret == null)
            synchronized (lock) {
                ret = threadStorage.get(attrName);
                if (ret != null)
                    return ret;

                ret = new ConcurrentHashMap<>();
                threadStorage.set(attrName, ret);
            }

        return ret;
    }

    private ArrayList<Pair<AccessControlRule, AccessCondition>>
getRuleAndCondition(Class<?> accessObjectClass, AccessActionType action) {
        ArrayList<Pair<AccessControlRule, AccessCondition>> ret =
ruleAndConditionTable().get(new Pair<>(accessObjectClass, action));
        return ret;
    }

    @Override
    public void grantAccess(AclSubjectUser subject, Class<?> accessObjectClasses,
AccessActionType action, OwnershipType[] conditionParams) {
        for (OwnershipType ownershipType : conditionParams)
            addACLRule(subject, action, accessObjectClasses, ownershipType);
    }

    @Override
    public <T> T checkAccess(T accessObject, AccessActionType action) {
        if (accessObject == null) return null;
        ArrayList<Pair<AccessControlRule, AccessCondition>> ACLpair =
getRuleAndCondition(accessObject.getClass(), action);
        if (ACLpair == null) return accessObject;
        Iterator<Pair<AccessControlRule, AccessCondition>> iterator =
ACLpair.iterator();
        do {
            Pair<AccessControlRule, AccessCondition> item1 = iterator.next();
            AccessControlRule rule = item1.getFirst();
            if (rule.getConditionParams().equals(ALL)) return accessObject;
            AccessCondition condition = item1.getSecond();
            T ret = condition.passes(accessObject, rule.getAction()) ?
```

```
accessObject : null;
            if (ret != null) return ret;
            if (!iterator.hasNext()) return (T) onDeny(accessObject,
item1.getFirst());
        }
        while (iterator.hasNext());
        return null;
    }

    @Override
    public <T> Collection<T> checkAccess(Collection<T> accessObjects,
AccessActionType action) {
        if (CollectionUtils.isEmpty(accessObjects)) return accessObjects;
        accessObjects.stream().forEach(e -> {
            checkAccess(e, action);
        });
        return accessObjects;
    }

    @Override
    public <T> T checkAccess(AclValidated<T> accessObjects, AccessActionType
action) {
        if (accessObjects.getData() == null &&
!accessObjects.getErrors().isEmpty()) return (T) accessObjects;
        checkAccess(accessObjects.getData(), action);
        return (T) accessObjects;
    }

    @Override
    public <T> List<T> filterAccessible(Collection<T> accessObjects,
AccessActionType action) {
        if (CollectionUtils.isEmpty(accessObjects)) return new ArrayList<T>();
        return accessObjects.stream().filter(e -> {
            ArrayList<Pair<AccessControlRule, AccessCondition>> ACLpair =
getRuleAndCondition(e.getClass(), action);
            Iterator<Pair<AccessControlRule, AccessCondition>> iterator =
ACLpair.iterator();
            do {
                Pair<AccessControlRule, AccessCondition> item1 = iterator.next();
                return
accessConditionBuilder.buildCondition(item1.getFirst()).passes(e, action);
            }
            while (iterator.hasNext());
```

```
        })
                .collect(Collectors.toList());
    }


    @Override
    public <T> T[] checkAccess(T[] accessObjects, AccessActionType action) {
        if (ArrayUtils.isEmpty(accessObjects))
            return accessObjects;
        checkAccess(Arrays.asList(accessObjects), action);
        return accessObjects;
    }

    private <T> T onDeny(Object accessObject, AccessControlRule rule) {
        String message = "No Access for User ID=" + rule.getSubject().getId() + "
to Object " + accessObject.getClass().getSimpleName();
        throw new AccessDenyException(accessObject.getClass().getSimpleName(),
"AccessDenyExciption", message);
    }

    private void addACLRule(AclSubjectUser subject, AccessActionType action,
Class<?> accessObjectClass, OwnershipType conditionParams) {
        AccessControlRule rule = new AccessControlRule(subject, action,
accessObjectClass, conditionParams);
        AccessCondition cond = accessConditionBuilder.buildCondition(rule);
        ArrayList<Pair<AccessControlRule, AccessCondition>> arrayList =
ruleAndConditionTable().get(new Pair(accessObjectClass, action));
        if (arrayList == null) {
            arrayList = new ArrayList<>();
            arrayList.add(new Pair(rule, cond));
            ruleAndConditionTable().put(new Pair(accessObjectClass, action),
arrayList);
            return;
        }
        arrayList.add(new Pair(rule, cond));
    }
}

public enum AccessActionType {
    WRITE, READ
}
```

```
public interface AccessCondition {

    void setSubjectUsers(AclSubjectUser user);;

    default void setParams(AccessConditionParam... param){};

    boolean passes(Object object, AccessActionType action);

}

@Data
public class AccessControlRule {
    AclSubjectUser subject;
    AccessActionType action;
    Class<?> accessObjectClass;
    AccessConditionParam conditionParams;

    public AccessControlRule(AclSubjectUser subject, AccessActionType action,
Class<?> accessObjectClass, AccessConditionParam conditionParams) {
        this.subject = subject;
        this.action = action;
        this.accessObjectClass = accessObjectClass;
        this.conditionParams = conditionParams;
    }
}

public interface AclSubjectUser{
    String getId();
}

public interface AclValidated<T> {
    Object getData();
    List<T> getErrors();
}

public interface AccessConditionBuilder {
    AccessCondition buildCondition(AccessControlRule rule);
}

public interface AccessControlManager {

    void grantAccess(AclSubjectUser subjectUser, Class<?> object, AccessActionType
action, OwnershipType[] grantTo);
```

```
    <T> T checkAccess(T accessObject, AccessActionType action);

    <T> T checkAccess(AclValidated<T> accessObject, AccessActionType action);

    <T> T[] checkAccess(T[] accessObjects, AccessActionType action);

    <T> Collection<T> checkAccess(Collection<T> accessObjects, AccessActionType
action);

    <T> List<T> filterAccessible(Collection<T> accessObjects, AccessActionType
action);

}

public interface AclSubjectUserProvider {
    AclSubjectUser currentUser();
}

@Service
@Transactional
public class LotServiceImpl implements LotService {

    @Autowired
    private AccessControlManager accessControlManager;

    @Autowired
    private LotBOMapper mapper;
    @Autowired
    private LotCustomRepository lotRepository;
    @Autowired
    private AuthService authService;
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private FileStorageHelperImpl fileStorageHelperImpl;
    @Autowired
    private CompanyService companyService;
    @Autowired
    private CurrencyService currencyService;
    @Autowired
    private FavoriteLotsService favoriteLotsService;
    @Autowired
```

```
    private VerificationService verificationService;

    @Override
    @SetOwnership
    @AccessCheckBefore(action = WRITE)
    public Validated<LotBO> save(LotBO lot) {
        Validated<LotBO> ret = new Validated<>();
        if (!isCompanyExist(ret, lot.getCompanyId())) {
            return ret;
        }
        LotEntity lotEntity = mapper.toLotEntity(lot);
        if (lotEntity.getId() != null) {
            setLotActiveAndOwnership(lotEntity, ret);
        }
        if (!ret.isValid()) {
            return ret;
        }
        lotRepository.save(lotEntity);
        if (lot.getPhoto() != null) {
            fileStorageHelperImpl.saveFile(lotEntity.getId(), FileType.LOT_PHOTO,
lot.getPhoto());
        }
        ret.setData(mapper.toLotBO(lotEntity));
        return ret;
    }

    private Validated<LotBO> setLotActiveAndOwnership(LotEntity lotEntity,
Validated<LotBO> ret) {
        LotEntity temp = lotRepository.findById(lotEntity.getId());
        if (!isLotExist(ret, temp)) return ret;
        lotEntity.setOwnership(temp.getOwnership());
        lotEntity.setIsActive(temp.getIsActive());
        return ret;
    }

    private boolean isLotExist(Validated<LotBO> ret, LotEntity temp) {
        if (temp == null) {
            ret.addError(LotValidationCode.LOT_NOT_FOUND);
            return false;
        }
        return true;
    }
```

```
private boolean isCompanyExist(Validated<LotBO> ret, String companyId) {
    if (companyService.findById(companyId) == null) {
        ret.addError(CompanyValidationCode.COMPANY_DOES_NOT_EXIST);
        return false;
    }
    return true;
}

@Override
public LotEntity findById(String id) {
    return lotRepository.findById(id);
}

@Override
public Validated<LotBO> findLotById(String id) {
    Validated<LotBO> ret = new Validated<>();
    if (id == null || id.equals("undefined")) {
        ret.addError(LotValidationCode.LOT_NOT_FOUND);
        return ret;
    }
    LotEntity lot = findById(id);
    if (lot == null) {
        ret.addError(LotValidationCode.LOT_NOT_FOUND);
        return ret;
    }
    ret.setData(mapper.toLotBO(lot));
    return ret;
}

@Override
@AccessCheckAfter(action = READ)
public Validated<LotBO> findLotByIdACL(String id) {
    return findLotById(id);
}

@Override
public List<LotEntity> findAll() {
    return lotRepository.findAll();
}

@Override
public void setPayment() {
    List<LotEntity> lotEntityList = findAll();
```

```
    for (LotEntity lotEntity : lotEntityList) {
        lotEntity.setPayment(PaymentEntity.builder()
                .percent(100)
                .type(PaymentType.ADVANCE)
                .build());
        lotRepository.save(lotEntity);
    }
}

@Override
public void setVerificationToLots() {
    List<LotEntity> lotEntityList = lotRepository.findAll();
    for (LotEntity lotEntity : lotEntityList) {
        lotEntity.setVerification(APPROVED);
        lotRepository.save(lotEntity);
    }
}

@Override
public void setIncotermToLots() {
    List<LotEntity> lotEntityList = lotRepository.findAll();
    for (LotEntity lotEntity : lotEntityList) {
        if (lotEntity.getIncoterm() == null) {
            lotEntity.setIncoterm(EXW);
            lotRepository.save(lotEntity);
        }
    }
}

@Override
public void setPreviousOwnership(String lotId) {
    LotEntity lotEntity = findById(lotId);
    if(lotEntity == null) return;
    String ownerId = lotEntity.getUserId();
    String ownerCompanyId = lotEntity.getCompanyId();
    lotEntity.getOwnership().setCreatorUserId(ownerId);
    lotEntity.getOwnership().setOwnerUserId(ownerId);
    lotEntity.getOwnership().setModifierUserId(ownerId);
    lotEntity.getOwnership().setOwnerCompanyId(ownerCompanyId);
    lotRepository.save(lotEntity);
}

@Override
```

```
    @AccessCheckAfter(action = READ)
    public Validated<LotSearchBO> search(LotSearchFilterBO filter) {
        Validated<LotSearchBO> ret = new Validated<>();
        List<LotEntity> list = lotRepository.findAll();
        List<LotBO> lots = list.stream()
                .filter(lot -> passIsActiveFilter(lot, filter.getIsActive()))
                .filter(lot ->
verificationService.passVerificationFilter(filter.getVerification(), lot))
                .filter(lot -> passFavoriteFilter(lot.getId(),
filter.getFavorite()))
                .filter(lot -> passCompanyIdsFilter(lot.getUserId(),
lot.getCompanyId(), filter.getCompanyIds()))
                .filter(lot -> passTextFilter(lot, filter.getSearchText()))
                .filter(lot -> passFilter(filter.getCountryIds(),
lot.getCountryCode()))
                .filter(lot -> passFilter(filter.getProductTypeIds(),
lot.getProductTypeId()))
                .filter(lot -> passFilter(filter.getCultureGroupIds(),
lot.getCultureGroupId()))
                .filter(lot -> passFilter(filter.getCultureIds(),
lot.getCultureId()))
                .filter(lot -> passFilter(filter.getIncoterms(),
lot.getIncoterm()))
                .filter(lot -> passFilter(filter.getPayment(), lot.getPayment()))
                .filter(lot -> passRangeFilter(filter.getAmountRange(),
lot.getAmount()))
                .filter(lot -> passFilter(filter.getUnit(), lot.getUnit()))
                .filter(lot -> passFilter(filter.getCurrencies(),
lot.getCurrency()))
                .map(lot -> mapper.toLotBO(lot))
                .filter(lot -> passPriceRangeFilter(filter.getPriceRange(),
filter.getCurrency(), lot)) // у большинства лотов нет цены
                .peek(lot -> lot.setIsFavorite(isLotFavorite(lot.getId())))
                .collect(Collectors.toList());
        sortLots(lots);
        PagingBOResponse paging = PagingBOResponse.builder()
                .number(filter.getPage().getNumber())
                .itemOnPage(filter.getPage().getItemOnPage())
                .totalSize(lots.size()).build();
        lots = lots.stream()
                .skip((filter.getPage().getNumber() - 1) *
filter.getPage().getItemOnPage())
                .limit(filter.getPage().getItemOnPage())
```

```
                .collect(Collectors.toList());
        LotSearchBO data = LotSearchBO.builder().page(paging).lots(lots).build();
        ret.setData(data);
        return ret;
    }

    @Override
    @AccessCheckAfter(action = READ)
    public Validated<LotSearchBO> search(LotSearchFilterBO filter) {
        Validated<LotSearchBO> ret = new Validated<>();
        LotQuery query = mapper.toLotQuery(filter);
        List<LotEntity> list = lotRepository.findByCustomQuery(query);
        List<LotBO> lots = list.stream()
                .filter(lot -> passIsActiveFilter(lot, filter.getIsActive()))
                .filter(lot ->
verificationService.passVerificationFilter(filter.getVerification(), lot))
                .filter(lot -> passCompanyIdsFilter(lot.getUserId(),
lot.getCompanyId(), filter.getCompanyIds()))
                .filter(lot -> passFavoriteFilter(lot.getId(),
filter.getFavorite()))
                .map(lot -> mapper.toLotBO(lot))
                .filter(lot -> passPriceRangeFilter(filter.getPriceRange(),
filter.getCurrency(), lot)) // у большинства лотов нет цены
                .peek(lot -> lot.setIsFavorite(isLotFavorite(lot.getId())))
                .collect(Collectors.toList());
        sortLots(lots);
        PagingBOResponse paging = PagingBOResponse.builder()
                .number(filter.getPage().getNumber())
                .itemOnPage(filter.getPage().getItemOnPage())
                .totalSize(lots.size()).build();
        lots = lots.stream()
                .skip((filter.getPage().getNumber() - 1) *
filter.getPage().getItemOnPage())
                .limit(filter.getPage().getItemOnPage())
                .collect(Collectors.toList());
        LotSearchBO data = LotSearchBO.builder().page(paging).lots(lots).build();
        ret.setData(data);
        return ret;
    }


    private boolean passFavoriteFilter(String lotId, Boolean favoriteFilter) {
        if (favoriteFilter == null) return true;
```

```
        if (!favoriteFilter) return true;
        var user =
userRepository.findById(authService.currentUser().getId()).get();
        if (favoriteLotsService.findAllByUserId(user.getId()) == null) return
false;
        return isLotFavorite(lotId);
    }

    private boolean passIsActiveFilter(LotEntity lotEntity, Boolean isActive) {
        String userId = authService.currentUser().getId();
        if (lotEntity.getIsActive() == null) return true;
        if (isActive == null) {
            if (!lotEntity.getIsActive() && !(lotEntity.getUserId().equals(userId)
|| authService.currentUser().getRoles().contains(UserRoleType.ORDERS_MODERATOR)))
                return false;
            return true;
        }
        if (lotEntity.getIsActive() && isActive) return true;
        if (!isActive && !lotEntity.getIsActive() &&
(lotEntity.getUserId().equals(userId) ||
authService.currentUser().getRoles().contains(UserRoleType.ORDERS_MODERATOR)))
            return true;
        return false;
    }

    private boolean isLotFavorite(String lotId) {
        UserEntity user =
userRepository.findById(authService.currentUser().getId()).get();
        if (favoriteLotsService.findAllByUserId(user.getId()) == null) return
false;
        for (String favoriteLotId :
favoriteLotsService.findAllByUserId(user.getId())) {
            if (favoriteLotId.contains(lotId)) return true;
        }
        return false;
    }


    private void sortLots(List<LotBO> lots) {
        Collections.sort(lots, new Comparator<LotBO>() {
            @Override
            public int compare(LotBO o1, LotBO o2) {
                try {
```

```
                    return
o1.getOwnership().getDateModified().compareTo(o2.getOwnership().getDateModified())
;
                } catch (Exception e) {
                    return 0;
                }
            }
        });
        Collections.reverse(lots);
    }

    private boolean passCompanyIdsFilter(String userId, String companyId,
List<String> companyIds) {
        if (authService.currentUser().getCompanyId() != null) {
            if (companyIds.size() == 0 &&
!authService.currentUser().getCompanyId().equals(companyId)) return true;
        }
        if (companyId == null && userId.equals(authService.currentUser().getId()))
return true;
        if (companyId == null &&
!userId.equals(authService.currentUser().getId())) return true;
        if (companyId == null) return false;
        if (companyIds.size() == 0 &&
!companyId.equals(authService.currentUser().getCompanyId())) return true;
        if (companyId == null &&
!companyIds.contains(authService.currentUser().getCompanyId())) return true;
        for (String id : companyIds) {
            if (companyId.equals(id)) return true;
        }
        return false;
    }

    private boolean passTextFilter(LotEntity lot, String searchText) {
        if (isBlank(searchText)) return true;
        if (lot.getName() != null &&
lot.getName().toLowerCase().contains(searchText.toLowerCase())) return true;
        //if (lot.getNameRU() != null &&
lot.getNameRU().toLowerCase().contains(searchText.toLowerCase())) return true;
        if (lot.getGroup() != null &&
lot.getGroup().toLowerCase().contains(searchText.toLowerCase()))
            return true;
        return lot.getDescription() != null &&
lot.getDescription().toLowerCase().contains(searchText.toLowerCase());
```

```
        return lot.getDescriptionRU() != null &&
lot.getDescriptionRU().toLowerCase().contains(searchText.toLowerCase());
        return false;
    }

    private <T> boolean passFilter(List<T> filterList, T value) {
        if (isEmpty(filterList)) return true;
        if (value == null) return false;
        return contains(filterList, value);
    }

    private <T> boolean passRangeFilter(T range, Float value) {
        if (range == null)
            return true;
        if (value == null)
            return false;
        if (range.getClass().equals(AmountRangeBO.class)) {
            AmountRangeBO temp = (AmountRangeBO) range;
            return (temp.getFrom() == null || temp.getFrom() <= value) &&
(temp.getTo() == null || temp.getTo() >= value);
        }
        if (range.getClass().equals(PriceRangeBO.class)) {
            PriceRangeBO temp = (PriceRangeBO) range;
            return (temp.getFrom() == null || temp.getFrom() <= value) &&
(temp.getTo() == null || temp.getTo() >= value);
        }
        return false;
    }

    private boolean passPriceRangeFilter(PriceRangeBO range, Integer currency,
LotBO lot) {
        if (range == null)
            return true;
        if (currency == null)
            return true;
        if (lot.getPrice() == null)
            return false;
        if(lot.getCurrency().equals(currency))
            return (range.getFrom() == null || range.getFrom() <= lot.getPrice())
&& (range.getTo() == null || range.getTo() >= lot.getPrice());
        if(!lot.getCurrency().equals(currency)){
            if(currency.equals(1)){
                lot.setPrice(lot.getPriceUsd());
```

```
                lot.setCurrency(currency);
                return (range.getFrom() == null || range.getFrom() <=
lot.getPriceUsd()) && (range.getTo() == null || range.getTo() >=
lot.getPriceUsd());
            }
            Float price = currencyService.convertPriceUsdToSearchPrice(currency,
lot.getPriceUsd());
            lot.setPrice(price);
            lot.setCurrency(currency);
            return (range.getFrom() == null || range.getFrom() <= price) &&
(range.getTo() == null || range.getTo() >= price);
        }
        return false;

    }

    @Override
    public Validated<LotBO> changeFavorite(String lotId) {
        Validated<LotBO> ret = getValidatedLotBOACL(lotId);
        if (!ret.isValid()) return ret;
        saveFavorite(lotId);
        return ret;
    }

    @Override
    public Validated<LotBO> inactivate(String lotId) {
        Validated<LotBO> ret = getValidatedLotBOACL(lotId);
        if (!ret.isValid()) return ret;
        Boolean isActive = ret.getData().getIsActive();
        ret.getData().setIsActive(!isActive);
        lotRepository.save(mapper.toLotEntity(ret.getData()));
        return ret;
    }

    @Override
    public Validated<LotBO> approve(String lotId) {
        verificationService.approve(lotId, LOT);
        Validated<LotBO> ret = getValidatedLotBOACL(lotId);
        if (!ret.isValid()) return ret;
        if (!isCompanyApproved(ret)) return ret;
        ret.getData().setVerification(APPROVED);
        lotRepository.save(mapper.toLotEntity(ret.getData()));
        return ret;
```

```
    }

    private boolean isCompanyApproved(Validated<LotBO> ret) {
        CompanyBO company =
companyService.findById(ret.getData().getOwnership().getOwnerCompanyId());
        if (!company.getVerification().equals(APPROVED)) {
            ret.addError(LotValidationCode.COMPANY_NOT_APPROVED);
            return false;
        }
        return true;
    }

    @Override
    public Validated<LotBO> disapprove(LotDisapproveBO lotDisapproveBO) {
        Validated<LotBO> ret = getValidatedLotBOACL(lotDisapproveBO.getLotId());
        if (!ret.isValid()) return ret;
        ret.getData().setReason(lotDisapproveBO.getReason());
        ret.getData().setVerification(DISAPPROVED);
        lotRepository.save(mapper.toLotEntity(ret.getData()));
        return ret;
    }

    @Override
    public Validated<LotBO> getValidatedLotBOACL(String lotId) {
        Validated<LotBO> ret = findLotById(lotId);
        if (!ret.isValid()) return ret;
        accessControlManager.checkAccess(ret, WRITE);
        ret.setData(ret.getData());
        return ret;
    }

    private void saveFavorite(String lotId) {

        UserBO user = authService.currentUser();
        List<String> favoriteLots =
favoriteLotsService.findAllByUserId(user.getId());
        if (favoriteLots == null) favoriteLots = new ArrayList<>();
        if (favoriteLots.contains(lotId))
            favoriteLotsService.delete(user.getId(), lotId);
        else
            favoriteLotsService.save(user.getId(), lotId);
    }
```

```
    @Override
    public Boolean deleteAll() {
        lotRepository.deleteAll();
        return true;
    }

    @Override
    public void deleteCertainLot(String lotId) {
        lotRepository.deleteById(lotId);
    }
}

public interface LotService {

    Validated<LotBO> save(LotBO lot);

    LotEntity findById(String id);

    Validated<LotBO> findLotById(String id);

    Validated<LotBO> findLotByIdACL(String id);

    List<LotEntity> findAll();

    void setPayment();

    void setVerificationToLots();

    void setIncotermToLots();

    void setPreviousOwnership(String lotId);

    Validated<LotSearchBO> search(LotSearchFilterBO lotSearchFilter);

    Validated<LotBO> changeFavorite(String lotId);

    Validated<LotBO> inactivate(String lotId);

    Validated<LotBO> approve(String lotId);

    Validated<LotBO> disapprove(LotDisapproveBO lotDisapproveBO);

}
```