Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Механізми безпеки у мікросервісній архітектурі

Текстова частина до кваліфікаційної роботи

за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
асистент Андрощук М.В.
(прізвище та ініціали)
_____
(підпис)
“_____” _____ 2024 р.

Виконав студент _____
Загривий О.С.
(прізвище та ініціали)
“_____” _____ 2024 р.

Київ 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри інформатики,
доцент, кандидат наук
_____ Гороховський С.С.
(підпис)
“_____” _____ 2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту Загривому Олегу Сергійовичу факультету інформатики 4-го курсу

Тема: Механізми безпеки у мікросервісній архітектурі

Зміст ТЧ до кваліфікаційної роботи:

1. Аналіз предметної області
2. Основні механізми забезпечення безпеки в мікросервісній архітектурі
3. Планування розробки мікросервісного застосунку
4. Розробка застосунку

Дата видачі: „____ ” ____ 2023 р.

Керівник _____ (підпис)

Завдання отримав _____ (підпис)

Тема: Механізми безпеки у мікросервісній архітектурі.

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на кваліфікаційну роботу.	31.10.2023	
2.	Огляд та аналіз матеріалів за темою роботи, дослідження механізмів безпеки.	30.11.2023	
3.	Планування та проектування розробки застосунку.	20.12.2023	
4.	Будування застосунку.	28.02.2024	
5.	Тестування застосунку на захищеність від різних видів атак.	15.03.2024	
6.	Виправлення вразливостей застосунку.	30.04.2024	
7.	Написання розділу 1 «Аналіз предметної області»	03.05.2024	
8.	Написання розділу 2 «Основні механізми забезпечення безпеки в мікросервісній архітектурі»	06.05.2024	
9.	Написання розділу 3 «Планування розробки мікросервісного застосунку»	10.05.2024	
10.	Написання розділу 4 «Розробка застосунку»	13.05.2024	
11.	Коригування роботи.	16.05.2024	
12.	Створення презентації.	17.05.2024	
13.	Захист кваліфікаційної роботи.	27.05.2024	

Загривий О. С. _____

Андрощук М. В. _____

“_____” _____ _____ р.

ЗМІСТ

АНОТАЦІЯ

Ця робота присвячена дослідженню сучасних механізмів безпеки мікросервісних архітектур з створенням власного промислового мікросервісного застосунку. У роботі досліджено основні механізми безпеки, такі як аутентифікація, авторизація, шифрування даних, багаторівневий захист, захист від різних атак та інші. Для реалізації застосунку використано мову розробки Java, технології Docker для контейнеризації, Spring Boot та Spring Cloud для побудови мікросервісів, Spring Security та інші для забезпечення безпеки. Результатом роботи є реалізація безпечного мікросервісного застосунку, готового до впровадження у виробниче середовище.

ВСТУП

Сучасний розвиток програмної індустрії привертає увагу до мікросервісної архітектури як ефективного підходу до побудови промислових програмних систем. Ця архітектурна концепція дозволяє будувати великі системи з набору невеликих та незалежних компонентів, що прискорює розробку, підтримку та масштабування. Відокремленість сервісів дає розробникам можливість працювати над індивідуальними частинами системи без впливу на інші компоненти. Це створює можливість для різних команд розробників працювати над різними проектами, які можуть мати різні вимоги та технології.

Проте разом із зростанням популярності мікросервісів з'являються нові виклики та вимоги щодо їх безпеки. Чим більше сервісів у системі, тим більше потенційних моментів небезпеки, оскільки кожен сервіс може стати точкою входу для зловмисників. Складність та розподіленість мікросервісних архітектур створюють потенційні ризики безпеки, пов'язані з комунікацією між сервісами, керуванням доступом та захистом даних.

Виходячи з тенденцій індустрії, метою даної роботи було визначено дослідити різні аспекти безпеки в мікросервісних архітектурах та розробити власний промисловий мікросервісний застосунок, який буде захищений від максимальної кількості потенційних загроз.

Робота складається з чотирьох розділів.

У першому розділі проведено аналіз актуальності мікросервісної архітектури у сучасному світі. Коротко розглянуто важливість та складність впровадження заходів безпеки у розподілених системах.

Другий розділ присвячено основним механізмам забезпечення безпеки в мікросервісних архітектурах. Розглянуто різні рівні захисту, а також методи аутентифікації, авторизації, логування, моніторингу безпеки, захисту баз даних, та інші механізми безпеки.

У третьому розділі розглянуто питання планування розробки мікросервісного застосунку, такі як вибір мов програмування та технологій, вибір бази даних, модель бази даних та інші.

У четвертому розділі надано детальний огляд процесу розробки застосунку, починаючи з монолітної архітектури та завершуючи фінальним станом застосунку у вигляді мікросервісів. Описано імплементацію різноманітних заходів забезпечення безпеки та захисту від різних видів атак.

Створено програмний продукт – мікросервісний застосунок, який забезпечено широким спектром заходів безпеки.

РОЗДІЛ 1: АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Актуальність мікросервісної архітектури

У сучасному цифровому світі організації все більше звертаються до мікросервісної архітектури як ефективного інструменту для розробки програмного забезпечення. Цей підхід дозволяє розділити застосунок на невеликі, незалежні компоненти, що сприяє збільшенню гнучкості, масштабованості та швидкості розробки. Однак, разом із зростанням популярності мікросервісної архітектури зростають і виклики щодо забезпечення її безпеки.

Дослідження компанії O'Reilly 2020 року[1] підтвердило цю тенденцію зростання популярності мікросервісів, показавши, що 77% респондентів вже впровадили мікросервісну архітектуру, з яких 92% зазнали успіху у її використанні. Ці цифри свідчать про широке прийняття мікросервісної архітектури серед організацій різних розмірів та галузей. Звіт також виявив, що 29% організацій повідомили про міграцію або впровадження більшості своїх систем з використанням мікросервісів. Це підтверджує тенденцію до переходу від монолітних архітектур до розподілених систем, що складаються з невеликих незалежних компонентів.

1.2. Важливість забезпечення безпеки в мікросервісній архітектурі

У контексті мікросервісної архітектури, забезпечення безпеки є однією з найбільш важливих та актуальних проблем. Оскільки мікросервіси представляють собою розподілені компоненти програмного забезпечення, які взаємодіють між собою через мережу, вони стають потенційними точками атак і вразливостей.

Розглянемо, наприклад, той факт, що завдяки масштабуванню, у великих мікросервісних системах можуть бути сотні або навіть тисячі екземплярів

мікросервісів, кожен з яких може мати свої власні вражаючі поверхні та рівні доступу до даних. Це ускладнює виявлення та виправлення потенційних вразливостей, а також контроль за безпекою системи в цілому.

Крім того, мікросервісна архітектура підвищує складність контролю за доступом до даних і конфіденційністю інформації. Переходячи від монолітних застосунків до розподілених систем, компанії вводять більше точок доступу до даних, що може збільшити ризик неправильного використання чи несанкціонованого доступу до конфіденційної інформації.

1.3. Специфіка реалізації безпеки в мікросервісних архітектурах порівняно з іншими підходами

Мікросервісна архітектура відрізняється від традиційних монолітних систем підходом до розподіленості та масштабованості. У зв'язку з цим, підхід до забезпечення безпеки в мікросервісних системах також має свої особливості.

Порівняно з іншими підходами, мікросервісна архітектура може потребувати більшої уваги до безпеки через більшу кількість компонентів та їхню розподіленість. Наприклад, у монолітних системах не потрібно контролювати налаштування шляхів та безпеки спілкування між сервісами. Шифрування повідомлень, пошук служб (сервісів) та масштабування окремих компонент також не є хвилюваннями розробників додатків монолітних архітектур.

1.4. Складність впровадження безпеки в розподілених системах

Хоча мікросервісна архітектура може забезпечити багато переваг для розробки програмного забезпечення, вона також створює додаткові виклики у сфері забезпечення безпеки. Однією з головних складнощів є управління безпекою в умовах розподіленості та гетерогенності сервісів.

У мікросервісних системах кожен сервіс може мати власні правила та механізми безпеки, що робить управління цими правилами складним завданням. Необхідність координації між різними командами розробників, адміністраторами системи та аналітиками безпеки ускладнює процес впровадження та підтримки безпеки в мікросервісній архітектурі.

Крім того, через розподіленість системи та широкий спектр технологій, що використовуються в мікросервісах, може виникати проблема забезпечення єдності підходів до безпеки та виконання стандартів безпеки в усіх сервісах системи.

Таким чином, хоча мікросервісна архітектура надає багато переваг, впровадження та підтримка безпеки в таких системах може бути складним завданням, яке вимагає уваги та професіоналізму з боку команди розробників та адміністраторів.

1.5. Аналіз впливу безпеки на продуктивність та масштабованість мікросервісних систем

У контексті мікросервісної архітектури, забезпечення безпеки є важливим аспектом, який може впливати на продуктивність та масштабованість системи. Додаткові заходи забезпечення безпеки, такі як шифрування даних, автентифікація та авторизація користувачів, можуть вимагати додаткових обчислювальних ресурсів та збільшити навантаження на систему. Однак, необхідність в цих заходах може переважати їхній вплив на продуктивність, забезпечуючи високий рівень безпеки даних та захист користувачів.

Дослідження впливу безпеки на продуктивність та масштабованість мікросервісних систем включає аналіз різних методів забезпечення безпеки та їхній потенційний вплив на швидкодію та реагування системи на збільшення навантаження. Наприклад, деякі варіанти впровадження безпеки менш ефективні або навіть не потрібні коли продукт розгорнутий у приватній мережі, але можуть впливати на продуктивність.

1.6. Рівні захищеності систем

За американським стандартом NCSC є різні рівні захищеності комп’ютерних систем, які відображають їхній рівень безпеки та вимоги до захисту інформації. Ці рівні, від A1 до C2, служать основою для класифікації систем залежно від їхньої важливості та рівня конфіденційності.

Система рівня A1 визначається як система з найвищим рівнем захисту. Ці системи застосовуються переважно в сфері військових додатків та мають найсуворіші вимоги до безпеки. Вони характеризуються високим рівнем технічних і організаційних заходів для захисту інформації від несанкціонованого доступу та зміни.

Натомість система рівня C2 вважається менш суворою з точки зору захисту. Хоча вона також має високий рівень безпеки, вимоги до неї менш жорсткі порівняно з системою рівня A1. Наприклад, для доступу до системи рівня C2 вимагається індивідуальний логін та пароль, що забезпечує ідентифікацію користувача. Однак цей рівень може не потребувати такого ж рівня технічних заходів та контролю, що й система рівня A1.

РОЗДІЛ 2: ОСНОВНІ МЕХАНІЗМИ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ В МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

2.1. Захист на різних рівнях

В мікросервісній архітектурі важливо забезпечити безпеку на різних рівнях, включаючи рівень мережі, рівень транспорту та рівень додатку, що відповідають рівням 3, 4 та 7 OSI[3] (Open Systems Interconnection).

Для захисту мережевої інфраструктури можуть використовуватися різні засоби, такі як файрволи та віртуальні приватні мережі (VPN).

- Файрволи дозволяють контролювати трафік, який входить та виходить з мережі, і встановлювати правила безпеки для обмеження доступу. Ці правила можуть включати захист від різних атак, наприклад, DDoS/DoS (Distributed Denial-of-Service / Denial-of-Service).
- Встановлення відокремлених мережевих зон (DMZ) використовується для розділення публічних та приватних ресурсів.
- VPN забезпечують шифрування трафіку між вузлами мережі, забезпечуючи конфіденційність та цілісність даних.

Захист транспортного рівня:

TLS / SSL (Transport Layer Security / Secure Sockets Layer) - це протоколи шифрування, які забезпечують безпеку передачі даних через мережу Інтернет. Вони використовуються для захисту конфіденційності та цілісності даних під час їх передачі між вузлами мережі.

Використання сертифікатів також допомагає з автентифікацією серверів та забезпеченням конфіденційності даних під час передачі інформації між ними.

Для захисту на рівні додатку:

- Використання протоколу HTTPS замість HTTP - це один з ключових аспектів захисту на рівні додатку у мікросервісній архітектурі. HTTPS забезпечує шифрування та аутентифікацію даних, що передаються між клієнтом та сервером, зменшуючи ризик перехоплення або модифікації даних під час їх передачі.
- Встановлення механізмів автентифікації та авторизації, таких як JWT або OAuth, для контролю доступу до функціональності додатку.
- Використання механізмів логування та моніторингу для виявлення та відстеження спроб несанкціонованого доступу.

2.2. Аутентифікація та авторизація

Аутентифікація та авторизація є двома ключовими аспектами безпеки, які грають важливу роль у захисті мікросервісних систем від несанкціонованого доступу. Розглянемо кожен з цих аспектів докладніше:

2.2.1. Аутентифікація

Аутентифікація - це процес підтвердження ідентичності користувача або системи. У мікросервісній архітектурі існує низка методів аутентифікації, включаючи:

- Локальна аутентифікація: Використовується для перевірки ідентичності користувача на основі логіну та пароля, збережених локально.
- Зовнішня аутентифікація: Користується сторонніми службами аутентифікації, такими як OAuth, LDAP або OpenID Connect, що дозволяє користувачам використовувати свої дані для аутентифікації безпосередньо в системі.

- Одноразовий вхід (Single Sign-On, SSO): Механізм, який дозволяє користувачам отримувати доступ до декількох сервісів з однієї автентифікаційної сесії.

2.2.2. Авторизація

Авторизація - це процес надання користувачеві або системі доступу до певних ресурсів або функцій після успішної аутентифікації. У мікросервісній архітектурі авторизація може відбуватися на рівні кожного сервісу, а також на рівні API gateway. Деякі з методів авторизації включають:

- Ролева модель (Role-Based Access Control, RBAC): Система, яка надає доступ до ресурсів на основі ролі користувача.
    - Плюси: простота у використанні та реалізації, легкість надання ролей та змін прав доступу, гнучкість у визначенні прав доступу за допомогою ролей.
    - Мінуси: може виникнути складність при реалізації більш складних прав доступу, важко управляти доступом для індивідуальних користувачів, які мають унікальні потреби.
- Політики доступу (Access Control Policies): Визначення прав доступу до ресурсів на основі різних умов і правил.
    - Плюси: гнучкість у визначенні прав доступу, можливість точно визначати права доступу для різних об'єктів та ресурсів.
    - Мінуси: з ростом системи може стати складніше керувати та підтримувати політики доступу, потребує додаткового часу та зусиль для розробки та налагодження складних правил.
- Token (OAuth, SAML, Bearer): Спосіб передачі авторизаційних даних у вигляді підписаного та зашифрованого токену.
    - Плюси: висока безпека, токени можуть бути шифрованими та підписаними, що забезпечує конфіденційність та цілісність даних;

можливість розширення: наприклад, OAuth дозволяє використовувати сторонні постачальники ідентифікації такі як Keycloak.

- Мінуси: вимагає додаткового часу та навчання для налагодження та управління токенами, використання сторонніх постачальників може сворювати проблеми зі сумісністю та безпекою

2.3. Логування та моніторинг безпеки

Логування та моніторинг відіграють ключову роль у виявленні та реагуванні на потенційні загрози безпеки у мікросервісній архітектурі.

Логування полягає у фіксації подій та дій, що відбуваються у системі, включаючи авторизацію, аутентифікацію, доступ до ресурсів, помилки та інші події.

Моніторинг включає постійний аналіз зібраних журналів подій для виявлення аномалій, несподіваних змін та потенційних загроз безпеки.

Для логування та моніторингу можуть використовуватися різноманітні інструменти, включаючи системи журналювання подій (наприклад, ELK Stack - Elasticsearch, Logstash, Kibana), системи моніторингу мережі (наприклад, Nagios, Zabbix), та спеціалізовані сервіси моніторингу безпеки (наприклад, Security Information and Event Management (SIEM) системи).

Важливо визначити стратегію логування та моніторингу, включаючи вибір відповідних інструментів та встановлення правильних параметрів журналювання. Системи логування та моніторингу повинні мати автоматичні сповіщення про аномалії та потенційні загрози безпеки, а також мануальні перевірки для аналізу та кореляції подій. Проведення регулярного аудиту логів та моніторингу системи для виявлення та виправлення потенційних проблем безпеки – одна з ключових застав захищеної системи.

2.4. Захист баз даних

Захист баз даних є критичним аспектом в мікросервісній архітектурі, оскільки бази даних зберігають важливу та конфіденційну інформацію. Нижче розглянемо основні аспекти захисту баз даних у контексті мікросервісної архітектури:

- Використання різних рівнів доступу та ролей користувачів для обмеження доступу до конфіденційної інформації.
- Використання технологій шифрування на рівні баз даних (наприклад, Transparent Data Encryption для SQL Server або Oracle Advanced Security для Oracle) або на рівні додатків (наприклад, шифрування в рівні додатків за допомогою фреймворків шифрування даних).
- Використання параметризованих запитів та інших технік для запобігання SQL-ін'єкціям та іншим видам атак на бази даних.
- Використання механізмів фільтрації та валідації введених даних для запобігання вразливостям, пов'язаним із обробкою даних.

2.5. Compliance, Standards

Важливою частиною забезпечення безпеки мікросервісних архітектур є дотримання та використання регламентів та стандартів. Деякі приклади можуть становити:

Глобальні:

SOC 2 (Service Organization Control 2) - це набір стандартів безпеки, розроблений Американською асоціацією фахівців у галузі облікових та фінансових технологій (AICPA), спрямований на організації, які надають послуги (такі як хмарні обчислення, обробка платіжних карток тощо). SOC 2 встановлює критерії оцінки і аудиту безпеки, конфіденційності та доступності сервісів, які надаються цими організаціями. Стандарт SOC 2 включає п'ять

ключових компонентів (Trust Service Criteria), включаючи безпеку, доступність, конфіденційність, обробку даних і моніторинг.

OWASP Top 10 (Open Web Application Security Project Top 10) - це список найбільш критичних уразливостей веб-додатків, який складається та оновлюється спільнотою Open Web Application Security Project. Цей список визначає найбільш актуальні загрози та ризики для безпеки веб-додатків, такі як крос-сайтовий скриптінг (XSS-атаки), вразливості введення даних, вразливості аутентифікації та авторизації тощо. OWASP Top 10 допомагає розробникам, адміністраторам та аналітикам безпеки усвідомити найбільш актуальні загрози та вжити заходів для їх запобігання.

Регіональні:

GDPR (Загальний регламент з питань захисту персональних даних) є нормативним актом Європейського Союзу, спрямованим на захист приватності персональних даних громадян. У контексті мікросервісної архітектури, відповідність GDPR вимагає від розробників та адміністраторів системи реалізації механізмів безпеки, які забезпечують захист персональних даних від несанкціонованого доступу, витоку та зміни. Це включає в себе захист даних у спокійному та транзитному станах, впровадження механізмів контролю доступу, аудиту та забезпечення прав користувачів на їхні дані.

HIPAA (Закон про портативність та страхування доступу до медичної інформації) - законодавчий акт, що регулює захист медичної інформації у Сполучених Штатах. Він накладає вимоги щодо зберігання, обробки та передачі медичних даних з метою запобігання несанкціонованому доступу та збереження конфіденційності. У контексті мікросервісної архітектури, відповідність HIPAA означає, що система повинна використовувати безпечні методи зберігання та передачі медичної інформації, застосовувати механізми контролю доступу та шифрування даних.

Загальною метою дотримання таких стандартів є забезпечення конфіденційності, цілісності та доступності особистих даних користувачів, що важливо для збереження довіри користувачів до системи і відповідності регіональним нормативним вимогам.

2.6. CI/CD

Налаштування безпеки CI/CD також є важливим етапом у забезпеченні безпеки мікросервісних систем. Continuous Integration (CI) та Continuous Deployment (CD) дозволяють автоматизувати процеси розробки, тестування та розгортання програмного забезпечення.

Статичний та динамічний аналіз коду - це важливі інструменти для виявлення потенційних вразливостей та помилок у програмному забезпеченні. Статичний аналіз проводиться без запуску програми та вивчає її код, щоб виявити можливі проблеми на етапі компіляції або під час розробки. Динамічний аналіз, натомість, виконується під час роботи програми та вивчає її поведінку, щоб виявити потенційні вразливості та помилки під час виконання.

Інтеграція статичного та динамічного аналізу коду в процес CI/CD дозволяє виявляти та виправляти проблеми безпеки на ранніх етапах розробки, а не на етапі production, що сприяє підвищенню рівня безпеки мікросервісних систем.

2.7. Відновлення після відмови

Відновлення роботи сервісів після відмови є ключовим аспектом забезпечення надійності та доступності мікросервісних систем. При розробці механізмів відновлення слід враховувати різні можливі сценарії відмов, включаючи технічні помилки, витоки ресурсів, атаки на систему та інші.

Стратегії відновлення можуть включати автоматичне перезавантаження сервісів, перенаправлення трафіку на резервні екземпляри, відновлення даних з резервних копій, автоматичне масштабування та інші методи.

Проведення регулярних тестів відновлення сервісів допомагає перевірити ефективність розроблених стратегій та впевнитися у їх працездатності в реальних умовах. Також важливо мати моніторингові системи, які вчасно виявляють відмови та сповіщають про них операторів для подальшої реакції та відновлення роботи сервісів.

2.8. Шифрування

Статичне шифрування та транзитне шифрування є важливими механізмами забезпечення безпеки в мікросервісних системах.

Статичне шифрування використовується для захисту даних у спокійний період, коли вони зберігаються в базі даних або на файловій системі. Під час статичного шифрування дані шифруються за допомогою статичних ключів, які зберігаються у безпечному сховищі, такому як keystore. Такий підхід дозволяє захистити дані від несанкціонованого доступу, навіть якщо хакер має фізичний доступ до системи.

Транзитне шифрування використовується для захисту даних під час їх передачі між різними компонентами системи. Під час транзитного шифрування дані шифруються перед тим, як вони покинуть джерело, і розшифровуються лише після того, як вони прибудуть у призначене місце. Це дозволяє уникнути можливості перехоплення або зміни даних під час їх транспортування.

Ці механізми грають важливу роль у забезпеченні конфіденційності та цілісності даних в мікросервісних системах, а їх використання допомагає зменшити ризик витоку чутливої інформації та забезпечити безпеку під час обробки та передачі даних.

2.9. Аналіз інцидентів

Аналіз інцидентів – це процес дослідження та оцінки подій, що порушують безпеку, з метою з'ясування їхніх причин, наслідків та розробки заходів для запобігання їхньому повторенню у майбутньому. У мікросервісних архітектурах, де система складається з багатьох незалежних компонентів, аналіз інцидентів стає ще більш важливим, оскільки один інцидент може мати каскадний ефект і вплинути на багато сервісів.

Процес аналізу інцидентів включає наступні кроки:

1. Виявлення інциденту: спостереження за аномальними подіями у системі або отримання сповіщень про порушення безпеки.
2. Збір інформації: збір даних про подію, включаючи час, місце та характер інциденту, а також будь-яку доступну інформацію про можливі причини та наслідки.
3. Аналіз причин: ретельний аналіз факторів, що спричинили інцидент, включаючи помилки в коді, неправильну конфігурацію, атаки ззовні або внутрішні порушення безпеки.
4. Виявлення наслідків: оцінка можливих наслідків інциденту для системи, даних та користувачів, а також вартість відновлення та втрати репутації.
5. Розробка заходів: впровадження заходів для усунення виявлених проблем, відновлення безпеки та запобігання подібним інцидентам у майбутньому.
6. Поширення знань: документація інформації про виявлені проблеми та вжиті заходи, її поширення серед команди для запобігання подібним інцидентам у майбутньому.

2.10. Зберігання резервних копій

Зберігання резервних копій – процес створення та зберігання копій даних системи з метою відновлення їх у випадку втрати, пошкодження або інших негативних подій.

Створення резервних копій включає наступні кроки:

1. Визначення обсягу даних: визначення даних, які потрібно резервувати, включаючи бази даних, конфігураційні файли, код програм тощо.
2. Вибір стратегії резервного копіювання: дизайн стратегії створення резервних копій, включаючи частоту створення, зберігання та розташування копій даних.
3. Автоматизація процесу: встановлення автоматизованих процесів створення та зберігання резервних копій для забезпечення регулярності та надійності.
4. Тестування та перевірка: періодичне тестування копій для перевірки їхньої цілісності та можливості відновлення даних.
5. Зберігання в безпечному місці: резервні копії зберігаються в безпечному місці, що захищене від природних катастроф, крадіжок та інших небезпек.

2.11. Фізичний захист

Фізичний захист є важливою складовою будь-якої інформаційної системи, включаючи мікросервісну архітектуру. Цей аспект безпеки стосується захисту фізичного обладнання, де знаходяться компоненти мікросервісів, а також контролю доступу до цих приміщень та обладнання.

Основні аспекти фізичного захисту включають:

- Контроль доступу: Забезпечення контролю доступу до приміщень, де знаходяться сервери та мережеве обладнання, через використання систем пропускного контролю, біометричних ідентифікаторів, камер відеоспостереження тощо.

- Захист обладнання: Забезпечення безпеки серверів, комутаторів, маршрутизаторів та іншого мережевого обладнання шляхом розміщення їх у захищених приміщеннях з обмеженим доступом.
- Резервне живлення: Встановлення систем резервного живлення, таких як генератори, акумуляторні батареї або безперебійні джерела живлення, для забезпечення неперервної роботи систем у разі відмови основного живлення.
- Захист від природних катастроф: Забезпечення захисту обладнання від природних катастроф, таких як пожежі, повені, землетруси тощо, шляхом розміщення даних у безпечних приміщеннях та використання відповідного обладнання захисту.
- Фізична безпека працівників: Надання інструктажу та підтримки працівників з питань фізичної безпеки, зокрема стосовно обробки та зберігання конфіденційної інформації, заходів захисту від крадіжок та вторгнень.

РОЗДІЛ 3: ПЛАНУВАННЯ РОЗРОБКИ МІКРОСЕРВІСНОГО ЗАСТОСУНКУ

3.1. Вибір мови програмування та технологій

3.1.1. Дослідження використання мов у мікросервісній розробці

З огляду на рівні стандарту NCSC, такий застосунок буде мати рівень C2, тобто, авторизація за логіном та паролем.

За результатами опитувань JetBrains 2022 року[16], 34% респондентів (найбільший відсоток) зазначили що використовують Java для розробки мікросервісів. З огляду на цю інформацію, а також широкі можливості Java, багату екостистему фреймворків та бібліотек, розробка мікросервісного застосунку була запланована на мові програмування Java з використанням фреймворку Spring Boot.

3.1.2. Java

Мова програмування Java відома своєю стабільністю, надійністю та великим спектром можливостей для створення різноманітних програмних застосунків. Використання Java дозволяє забезпечити високу ефективність та масштабованість мікросервісних застосунків, а також використовувати широкий спектр фреймворків та бібліотек для покращення розробки та підтримки проєкту.

3.1.3. Spring та Spring Boot

Spring та його екосистема, зокрема Spring Boot, є популярними виборами для розробки мікросервісних застосунків на мові Java. Spring надає гнучкість та простоту у реалізації складних бізнес-логік, а також забезпечує інтеграцію з іншими технологіями та сервісами. Spring Boot, з своєю конвенцією над

конфігурацією та автоматичним управлінням, спрощує розгортання мікросервісних застосунків та забезпечує швидкість розробки. Обидва фреймворки є популярними серед розробників та мають активну спільноту, що робить їх привабливими виборами для створення мікросервісних архітектур.

Колекція бібліотек Spring Cloud розширює можливості Spring, надаючи набір інструментів для створення розподілених систем, таких як керування конфігурацією, виявлення сервісів, circuit breaker (механізм забезпечення надійності мікросервісів у випадку відмови), інтелектуальна маршрутизація, мікропроксі, control bus (передача сигналів керування і команд), мікросервіси з коротким терміном життя, контрактне тестування та інші.

3.2. Вибір бази даних

Під час планування розробки мікросервісного застосунку було прийнято рішення використовувати різні бази даних для середовищ розробки та продуктивного впровадження. Ось докладний аналіз вибору баз даних:

3.2.1. Development - H2 Database:

- Легкість використання: H2 Database - це легка вбудована база даних, яка дозволяє швидко розпочати розробку без необхідності налаштування складної серверної бази даних.
- Швидкість розробки: H2 дозволяє швидко створювати тестові дані та проводити експерименти без зайвого налаштування.
- Вбудованість: H2 можна легко вбудувати в застосунок, що спрощує розгортання та тестування.

3.2.2. Production - PostgreSQL:

- Надійність і стійкість до відмов: PostgreSQL має вбудовані механізми для забезпечення надійності даних та стійкості до відмов. Він підтримує транзакційність та механізми відновлення, що забезпечує цілісність та стійкість даних у випадку аварій або відмов системи. У H2 Database, як вбудованій базі даних, може бути складніше забезпечити такий рівень надійності та стійкості.
- Масштабованість: PostgreSQL розроблений з урахуванням потреб великих обсягів даних та високого рівня навантаження. Він підтримує розподілену архітектуру та може ефективно обробляти великі обсяги транзакцій та запитів. H2 Database, незважаючи на свою легкість та швидкодію, може мати обмеження при обробці великих обсягів даних або великої кількості одночасних з'єднань.
- Рівень безпеки: PostgreSQL має більше розширених можливостей забезпечення безпеки користувачів, включаючи різні методи автентифікації (наприклад, пароль, Kerberos, LDAP) та можливості керування доступом. H2 Database має менше засобів для забезпечення безпеки, особливо в складних середовищах.

3.3. Модель бази даних

Модель БД була визначена для достатнього розподілення задач сервісів, тобто, актуальності мікросервісів. Було обрано модель «книжковий магазин», тобто зберігання, замовлення і оплата замовлень книжок. Ця модель включає в себе кілька таблиць, які відображають основні сутності та їх взаємозв'язки. Нижче наведено опис структури цієї моделі:

1. Таблиця "users" містить дані про користувачів системи. Кожний запис включає унікальний ідентифікатор користувача (id), ім'я користувача (username), пароль (password), електронну пошту (email), роль користувача (role).

2. Таблиця "books": Ця таблиця містить дані про книги в системі. Кожен запис включає унікальний ідентифікатор книги (id), назву (title), автора (author), опис (description), ISBN, ціну, кількість доступних та утримуваних книг.
3. Таблиця "payments" зберігає інформацію про оплату за замовлення. Кожний запис містить ідентифікатор оплати (id), ідентифікатор замовлення (order_id), дату оплати (payment_date), суму оплати (amount), метод оплати (payment_method) та статус оплати (status).
4. Таблиця "orders" - дані про замовлення. Кожен запис містить унікальний ідентифікатор замовлення (id), ідентифікатор користувача (user_id), дату замовлення (order_date).
5. Таблиця "order_items" зберігає інформацію про замовлені книги. Кожний запис містить унікальний ідентифікатор (id), ідентифікатор замовлення (order_id), ідентифікатор книги (book_id), кількість книг (quantity).

3.4. Шлюз

Шлюз (gateway, гейтвей) – це проміжний сервер, який відповідає за керування трафіком між клієнтами та мікросервісами. Це ключовий компонент, який забезпечує централізований доступ до мікросервісів, маршрутизацію запитів до них та захист системи від зовнішніх атак.

Різні шлюзи:

- Spring Cloud Gateway - це гнучкий та легкий шлюз, який надає високий рівень налаштувань та інтеграції з екосистемою Spring.
- Zuul - це шлюз, також розроблений Netflix (як і Spring Cloud Gateway), який також забезпечує керування трафіком та маршрутизацію запитів.
- Kong - це API-центрований шлюз, який надає широкі можливості керування API та ієрархією сервісів.

- NGINX - це сервер-посередник, який також може виконувати функції шлюзу. Він відомий своєю високою продуктивністю та швидкодією в обробці HTTP-запитів.
- Amazon API Gateway - це керований сервіс AWS, який надає можливість керувати та масштабувати API безпеки в хмарному середовищі Amazon Web Services.

Було обрано Spring Cloud Gateway, через сумісність з Spring, та також такі причини:

- Spring Cloud Gateway надає можливість централізованого керування конфігурацією та політиками безпеки для всіх сервісів.
- Вбудовані функціональність автентифікації та авторизації у Spring Cloud Gateway значно спрощують реалізацію цих механізмів в проекті.
- Spring Cloud Gateway легко масштабується, а також надає гнучкість у налаштуванні.

3.5. Аутентифікація

Було розглянуто декілька варіантів автентифікації користувачів, з них є такі:

- JWT (JSON Web Tokens) є популярним механізмом аутентифікації, який використовується для створення та передачі токенів доступу між клієнтом і сервером. Вони базуються на відкритому стандарті JSON та мають високий рівень безпеки.
- OAuth 2.0 - це протокол авторизації, який дозволяє користувачам надавати доступ до своїх ресурсів без передачі свого пароля. Він широко використовується для забезпечення безпеки API та веб-додатків. У прикладі даного застосунку, фреймворк Spring Security підтримує OAuth 2.0.

- Keycloak - це відкрита система управління ідентифікацією та доступом, яка надає централізоване керування користувачами, ролями та дозволами. Keycloak зазвичай використовує SSO (Single Sign-On), але ця система також підтримує різні протоколи аутентифікації, включаючи OAuth та SAML. Keycloak також можна як розгорнути незалежно, так і додати вбудований сервер у Spring Boot додаток.

Було обрано JWT через його простоту в реалізації та ефективність у передачі даних про авторизацію між клієнтом і сервером. JWT можна налаштувати мануально (шифрування та підпис), а також важко підробити, що дозволяє легко перевіряти валідність та контент JWT.

3.6. TLS / SSL

Транспортний рівень безпеки (TLS) є протоколом шифрування, який забезпечує захищену та конфіденційну комунікацію. В основі TLS лежить криптографічний протокол, який забезпечує аутентифікацію та шифрування даних під час їх передачі між двома або більше пристроями. Є два основні варіанти TLS:

1. Односторонній TLS (TLS termination): Цей підхід включає в себе аутентифікацію тільки сервера, забезпечуючи захист комунікації від прослуховування та модифікації. TLS термінація використовується для розшифрування трафіку на проміжному сервері, такому як шлюз або балансувальник навантаження (load balancer), перед його подальшою передачею до кінцевого сервера.
2. Взаємний TLS (mTLS): У взаємному TLS обидва боки - як сервер, так і клієнт - аутентифікуються один перед одним, використовуючи сертифікати TLS. Це забезпечує двосторонню перевірку підтвердження та встановлює безпечне з'єднання між ними. Також mTLS може

передбачати аутентифікацію між мікросервісами, що допомагає запобігти атакам типу «Man-in-the-middle». Але впровадження mTLS може бути складнішим та витратнішим через потребу в конфігурації та додаткових перевірок у запитах між сервісами.

З огляду на те, що зазвичай у мікросервісній архітектурі сервіси що не повинні інтерактувати з клієнтом знаходяться у приватній або захищеній мережі, а шлюз або альтернатива – відкриті для доступу, було обрано TLS termination. Основні переваги включають зменшення навантаження на внутрішні сервіси, покращення продуктивності порівняно з mTLS та спрощення управління сертифікатами TLS.

3.7. Postman

Postman є одним з найпопулярніших інструментів для розробки, тестування та взаємодії з API в індустрії програмного забезпечення. Він надає зручний та потужний інтерфейс для створення, відправлення та отримання HTTP-запитів і відповідей, що робить його важливим інструментом для розробників, тестувальників та архітекторів.

Postman обраний для автоматизації процесів тестування та відлагодження API. Використання Postman дозволить швидко створювати, відправляти та тестувати HTTP-запити на стороні клієнта, а також перевіряти відповіді сервера для впевненості в їхній правильності та коректності.

Однією з ключових функцій Postman є його можливість робити запити за допомогою HTTPS протоколу, що забезпечує шифрування даних під час їх передачі через мережу. Це робить Postman корисним інструментом для тестування захищених API, що працюють з HTTPS, що допомагає з тестуванням у сфері цього дослідження, наприклад, після налагодження TLS.

Postman також можна використовувати для тестування JWT (JSON Web Tokens) через заголовок Authorization у відправлених запитах. Це дозволяє

зручно взаємодіяти з API, які використовують JWT для автентифікації та авторизації користувачів, дозволяючи випробовувати та тестувати їхні функції безпосередньо в середовищі Postman.

РОЗДІЛ 4: РОЗРОБКА ЗАСТОСУНКУ

4.1. Створення моноліту

Першим кроком у розробці мікросервісного застосунку зазвичай є створення монолітного застосунку. Це програмна система, де всі компоненти та функціональність об’єднані в одну структуру. Основна мета моноліту – швидка розробку базового функціоналу без ускладнень, які зазвичай супроводжують мікросервісні архітектури.

На цьому етапі було створено REST API endpoints, розділено рівні на controller, service та repository. Кожен з цих рівнів виконує свою функцію: контролери (controller) відповідають за обробку HTTP-запитів та передачу даних на відповідні сервіси, сервіси (service) реалізують бізнес-логіку та взаємодіють з базою даних через репозиторії (repository), які відповідають за взаємодію з базою даних. Крім того, було впроваджено DTO (Data Transfer Object), що дозволяє ефективно передавати дані між різними шарами застосунку та захищати його від ненадійного змінення даних ззовні, або ж отримання надлишкових даних.

Також було створено SQL scripts, що заповнюють БД даними при її створенні.

4.2. Перехід від моноліту до мікросервісів

Наступним кроком є перехід від монолітної архітектури до мікросервісної. Це включає розбиття моноліту на окремі сервіси, кожен з яких відповідає за свою частину функціональності. Процес переходу зазвичай включає наступні кроки:

- Аналіз функціональності: Оцінка, які частини моноліту можна розбити на окремі мікросервіси. Це може вимагати ретельного вивчення функціональності та взаємодії компонентів в моноліті.
- Розбиття на сервіси: Розбивання функціональності на окремі сервіси з урахуванням принципів декомпозиції моноліту. Кожен сервіс повинен

бути незалежним та мати чітко визначені межі взаємодії з іншими сервісами.

- Тестування та впровадження: Проведення тестування окремих сервісів та їх впровадження в робоче середовище з наступним моніторингом та оптимізацією.

4.2.1. Розбиття на сервіси

У процесі переходу моноліт було розбито на декілька сервісів:

- Auth-service: відповідає за збереження інформації про користувачів, у майбутньому відповідає за надання дозволів, через створення та надання токенів авторизації.
- App: цей сервіс виконує функцію складу книг, та управління цим складом.
- Order-service: обробка та зберігання інформації про замовлення, тобто дата, кількість книг та інші деталі.
- Payment-service: відповідає за оплату замовлень. Це номер замовлення, статус оплати, дата та метод (якщо такі присутні).

Таке розділення має сенс, оскільки кожен сервіс спеціалізується на своїй функціональності, що дозволяє розподілити відповідальності та полегшити масштабування системи.

4.2.2. Забезпечення коректної роботи сервісів

Під час переходу до мікросервісів було виконано важливий крок - забезпечення коректність роботи, та відсутності помилок, особливо у задачах що використовують функціонал декількох сервісів.

Один з прикладів – завершення оплати. Під час виконання такої функції, застосовуються сервіси Payment-service, Order-service, та App. Тому, для

забезпечення коректної роботи, необхідно було впровадити транзакційні операції, які гарантують атомарність операцій над даними у всіх зазначених сервісах. Також важливо було провести тестування всього функціоналу, що використовується в різних сервісах, та здійснювати моніторинг за виникненням помилок.

Також важливо звернути увагу на БД, де під час розбиття деякі з таблиць потребували зміни типів та/або назв параметрів (наприклад один з Foreign Key на OrderItem, що стосується book_id, тепер простий параметр).

4.3. Контейнеризація

Контейнеризація є ключовою технологією у процесі розробки мікросервісних застосунків. Основним інструментом для контейнеризації є Docker, який дозволяє упаковувати кожен сервіс та його залежності в окремий ізольований контейнер. Docker надає зручний і стандартизований спосіб опису та управління контейнерами. Кожен контейнер містить в собі усі необхідні файли, бібліотеки та конфігурації, що дозволяє забезпечити консистентність середовища в різних стадіях розробки, тестування та розгортання. Основні переваги використання Docker у розробці мікросервісних застосунків:

- Ізоляція сервісів: Кожен сервіс розглядається як окремий контейнер, що дозволяє уникнути конфліктів між залежностями та забезпечити їх ізоляцію.
- Стандартизація середовища: Docker дозволяє описувати середовище кожного сервісу за допомогою Dockerfile, що спрощує управління конфігурацією та розгортанням.
- Легкість розгортання та масштабування: Контейнери можна швидко розгортати та масштабувати в будь-якому середовищі, незалежно від того, чи це локальний комп'ютер для розробки чи хмарна інфраструктура.

- Підтримка мікросервісної архітектури: Docker ідеально підходить для реалізації мікросервісної архітектури, оскільки дозволяє розбити додаток на невеликі, незалежні компоненти.
- Безпека та стабільність: Docker забезпечує ізольованість кожного контейнера, що допомагає уникнути впливу помилок одного сервісу на інші, а також зменшує ризик вразливостей системи. Також це дозволяє створити приватну мережу, і відкрити її частину назовні, що також допомагає позбутися зайвих відкритих компонент.

Під час переходу до контейнеризації, було визначено таку схему: контейнери сервісів створюються за допомогою декількох Dockerfile файлів, які було створено так, що вони копіюють .jar файл (зібраний сервіс) в контейнер і встановлюють його.

Розгортання контейнеризованих мікросервісів може бути здійснене різними способами в залежності від потреб проєкту, ресурсів та вимог до інфраструктури. Ось деякі з найпоширеніших варіантів розгортання мікросервісної архітектури:

1. Хмарне розгортання: Один з найпоширеніших варіантів, коли мікросервіси розгортаються у хмарному середовищі, такому як Amazon Web Services (AWS), Microsoft Azure, або Google Cloud Platform (GCP). Це дозволяє забезпечити масштабованість, високу доступність та гнучкість в управлінні ресурсами.
2. Контейнеризація з Kubernetes: Kubernetes - це оркестратор контейнерів, який дозволяє автоматизувати процеси розгортання, масштабування та управління контейнерами. Мікросервіси можуть бути розгорнуті у контейнерах і управлятися за допомогою Kubernetes, що забезпечує високу доступність та надійність системи.
3. Самостійні сервери: Мікросервіси можуть бути розгорнуті на самостійних серверах або віртуальних машинах, які керуються

власними інструментами управління ресурсами, такими як Docker Compose або Ansible. Цей підхід може бути використаний у випадках, коли потрібна повна контрольованість над інфраструктурою.

4. Сервери на місці (On-premises): У деяких випадках мікросервіси можуть бути розгорнуті на власних серверах у власних приміщеннях. Цей підхід дозволяє зберігати дані у внутрішній мережі організації та забезпечити високий рівень контролю над інфраструктурою.

З огляду на швидке та ефективне налаштування та розгортання для такого типу мікросервісного додатку, було обрано розгортання за допомогою Docker Compose, де було відкрито відповідні порти для міжсервісної комунікації.

Також, у Docker Compose було додано сервіс «postgres», що створює у одному контейнері всі БД що потрібні для сервісів, і створено нові SQL scripts для створення PostgreSQL таблиць для кожної з БД.

З огляду на потрібність БД (сервісу postgres) також додано залежність інших сервісів від правильно працюючого сервісу postgres (depends_on: postgres: condition: service_healthy).

Тобто, Docker Compose конфігурація створює контейнери сервісів за допомогою попередньо визначених Dockerfile, додає їм environment параметри, наприклад, секрети «SERVER_SSL_KEY_STORE_PASSWORD» або «SECRET_SHARED», відкриває визначені порти (або не відкриває порти якщо не визначено), та створює приватну мережу з визначених контейнерів.

4.4. Шлюз

4.4.1. Впровадження шлюзу

У ролі шлюзу було обрано Spring Cloud Gateway, нового сервісу додатку під назвою gateway-service, який відіграє ключову роль у розгортанні

мікросервісної архітектури. Впровадження шлюзу є важливим кроком, оскільки він відповідає за маршрутизацію запитів до правильних сервісів, забезпечення безпеки та контроль за доступом. Також, він дозволяє закрити попередньо відкритий доступ до сервісів, забезпечуючи вищий рівень безпеки.

Після налаштування шлюзу, можна забезпечити перенаправлення трафіку між сервісами через шлюз, що сприяє збільшенню контролю над комунікацією між різними частинами системи. Такий підхід дозволяє краще управляти трафіком, виявляти та реагувати на проблеми, а також підвищує загальну безпеку системи.

4.4.2. Налаштування TLS

Для забезпечення безпеки комунікації використовується TLS termination. Одним із етапів є створення та налаштування keystore – файлу, що містить сертифікати та приватні ключі, які використовуються для забезпечення безпеки комунікації за допомогою TLS (Transport Layer Security) або SSL (Secure Sockets Layer). У цьому файлі зберігаються конфіденційні дані, такі як сертифікати, які підтверджують ідентичність веб-сайту або сервісу, та відповідні приватні ключі, які використовуються для розшифрування інформації, що передається.

Після налаштування на шлюзі, раніше створений keystore додається до сервісів, які мають потребу у комунікації з іншими сервісами. Це необхідно для забезпечення того, що вони можуть взаємодіяти через HTTPS з використанням шлюзу, який передає трафік за захищеним каналом.

4.5. Аутентифікація та авторизація

Початково реалізовано створення та перевірка JWT токенів на auth-service. Для цього використовується метод «generateToken», який генерує JWT токен з вказаними клеймами та даними користувача. При генерації також додаються

ролі користувача, які зберігаються у токені. Ключ для підпису токена отримується з .env файлу та передається до сервісу через docker-compose. Далі цей ключ використовується для шифрування та перевірки підпису токену.

Далі реалізовано додавання секретного ключа до gateway для перевірки JWT токенів без необхідності звертатися кожен раз до auth-service. Це забезпечує ефективність та швидкодію системи, оскільки перевірка токенів відбувається на місці.

Після цього виконано функцію фільтрації маршрутів у gateway в залежності від ролей, які містяться у JWT токені. Шляхи login та register, що відповідають за вхід та регістрацію відповідно, доступні всім, а інші шляхи блокуються для користувачів, які не мають відповідних дозволів, що дозволяє забезпечити безпеку та контроль доступу до ресурсів.

Наступним створено реалізацію logout, тобто, виходу, на шлюзі за допомогою внесення до чорного списку (blacklisting) токенів. Користувач може вийти з системи шляхом додавання його токену до списку блеклісту, що забезпечує безпеку та захист від несанкціонованого використання токенів. При вході в систему також виконується перевірка токену на наявність у блеклісті, що забезпечує безпеку користувачів, блокуючи спроби використати вже заблокований токен.

4.6. Actuator

На сервісі автентифікації (auth-service) було встановлено модуль Spring Boot Actuator. Цей модуль надає додаткову інформацію про стан сервісу, що допомагає забезпечити його ефективну роботу завдяки моніторингу. Також, було змінено логіку помилкового введення паролю та був доданий додатковий шлях до Actuator, який відображає кількість невірних спроб введення пароля за останній день у відповідності до імен користувачів. Це допомагає виявляти та моніторити облікові записи, які можуть бути під загрозою атак типу «bruteforce».

4.7. Захист від атак

Для забезпечення безпеки системи та захисту від різних атак, було перевірено систему на вразливість до цих атак та використано ряд заходів.

4.7.1. DDoS/DoS

DDoS (розподілена атака на відмову в обслуговуванні) або DoS (атака на відмову в обслуговуванні) - це тип атаки, коли зловмисники намагаються перевантажити сервер або мережу, відправляючи велику кількість запитів, що призводить до відмови в обслуговуванні для законних користувачів. Атаки DDoS можуть бути спрямовані на будь-який компонент інфраструктури, включаючи мережеві пристрої, сервери та додатки.

Зазвичай захист від DDoS включає в себе застосування таких методів, як фільтрація трафіку, використання CDN (мережа доставки контенту), захист на рівні мережі та інші техніки, які спрямовані на виявлення та блокування шкідливого трафіку.

Для захисту від DDoS/DoS в додатку був розроблений спеціальний механізм на рівні шлюзу. Цей механізм відслідковує та аналізує кількість запитів від користувачів, і в разі перевищення певного порогу, автоматично блокує подальші запити від цього користувача на період часу, за замовчуванням – 1 хвилина. Такий механізм дозволяє ефективно захистити систему від масштабних атак DDoS, забезпечуючи нормальне функціонування системи та захищаючи її від перевантаження сервера.

4.7.2. Bruteforce

Атака методом перебору паролів, відома також як «Bruteforce», це метод, коли зловмисники намагаються вгадати пароль, шляхом послідовної спроби

різних комбінацій символів. Цей вид атаки може бути використаний для отримання несанкціонованого доступу до системи.

Для захисту від такої атаки, на сервісі автентифікації (auth-service) було встановлено механізм, що складається з структури даних «LoadingCache» з бібліотеки Guava, у яку заноситься інформація про невдалу спробу, та блокує користувача за його IP адресою на 24 години після 5 невдалих спроб введення пароля. Такий захист допомагає запобігти несанкціонованому доступу до системи та забезпечує безпеку аккаунтів користувачів.

4.7.3. SQL Injection

SQL Injection - це тип атаки на системи, що використовують бази даних, коли зловмисник використовує вразливості в програмному коді, що спілкується з базою даних, для виконання шкідливих SQL-запитів. Ця атака полягає у вставці зловмисного SQL-коду в параметризовані запити або введені дані веб-форми.

Для захисту від атак SQL Injection було використано підхід, який базується на використанні методів Spring Data JPA. Цей підхід полягає у використанні методів з назвами, які автоматично перетворюються в параметризовані запити, що робить їх type-safe та автоматично екранує спеціальні символи.

Цей механізм дозволяє уникнути вразливостей, пов'язаних з SQL Injection, та забезпечити безпеку бази даних, надійність та цілісність інформації. Використання параметризованих запитів у поєднанні з автоматичним екрануванням символів дозволяє уникнути виконання шкідливих SQL-запитів та зберегти дані користувачів у безпеці.

4.7.4. Session fixation

Session fixation - це атака, при якій зловмисник намагається зафіксувати або "фіксує" ідентифікатор сесії вразливого користувача. Після цього він може

використовувати цей ідентифікатор сесії для отримання доступу до системи під користувачем.

У системі атаку Session fixation заблоковано завдяки використанню JWT, який є зашифрованим, також створено механізм чорного списку, що ускладнює використання на випадок отримання JWT.

4.7.5. Sensitive data exposure

Sensitive data exposure - це вразливість, яка може призвести до небезпечного розголошення конфіденційної інформації, такої як паролі, особиста інформація чи фінансові дані.

Цю вразливість було заблоковано шляхом заборони доступу до ресурсів, які містять конфіденційну інформацію, для користувачів з недостатніми правами. Крім того, використано шар DTO (Data Transfer Object), який контролює доступ до sensitive даних, що додатково забезпечує безпеку цієї інформації.

4.7.6. Man-in-the-Middle

Атака «Man-in-the-Middle» - це тип атаки, при якій зловмисники вступають у зв'язок між двома комунікуючими сторонами, вдаючись за оригінальну сторону. Це дозволяє зловмисникам перехоплювати та модифікувати передачу даних між сторонами без їхнього відома.

Частковий захист від атаки типу «Man-in-the-Middle» забезпечується шляхом обмеження доступу лише до шлюзу, в той час як всі інші сервіси перебувають у приватній мережі. Крім того, комунікація між клієнтами та сервісами здійснюється за допомогою протоколу HTTPS в поєднанні з TLS.

4.7.7. Zero-day exploits

Zero-day exploit – атака на систему або забезпечення, що використовує вразливість, яка ще не відома розробникам або вендорам. Ці атаки особливо небезпечні, бо від них важко захиститися, вони надходять без попередження і захисту.

Найкращим підходом є використання стабільного програмного забезпечення та його залежностей, обмеження версій (не використання версій вище встановленого ліміту або використання конкретних версій), а також швидка реакція на виявлення вразливостей і їх виправлення.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було проаналізовано мікросервісну архітектуру та її вразливості безпеки. На основі цього аналізу був розроблений промисловий мікросервісний застосунок, який враховував кращі практики та заходи безпеки. При розробці цього застосунку, були використані такі технології як Spring Boot, Spring Cloud, TLS та інші.

Після побудови застосунку був проведений докладний аналіз його вразливостей. Цей аналіз дозволив виявити слабкі місця та потенційні загрози для безпеки системи, наприклад, вразливості до bruteforce та DoS атак. На основі цих виявлених проблем були розроблені та впроваджені різноманітні заходи забезпечення безпеки, спрямовані на мінімізацію ризиків, захист системи від потенційних атак і мінімізацію втрати продуктивності.

Розроблений застосунок можна покращити у декілька способів, один з них це впровадження повної системи логування та моніторингу. Це дозволить ефективно виявляти помилки та критичні ситуації, що виникають в роботі системи, та оперативно реагувати на них. Іншим є проведення статичного та динамічного аналізу коду з метою виявлення потенційних вразливостей та їх подальшого усунення. Також можна впровадити відмовостійкість та протестувати її під час масштабування сервісів.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Loukides M. Microservices Adoption in 2020 [Електронний ресурс] / M. Loukides, S. Swoyer – Режим доступу до ресурсу: https://www.oreilly.com/radar/microservices-adoption-in-2020/.

2. National Computer Security Center. Red Book / National Computer Security Center.

3. Forcepoint. What is the OSI Model? [Електронний ресурс] / Forcepoint – Режим доступу до ресурсу: https://www.forcepoint.com/cyber-edu/osi-model.

4. Understanding Denial-of-Service Attacks [Електронний ресурс] – Режим доступу до ресурсу: https://www.cisa.gov/news-events/news/understanding-denial-service-attacks.

5. Delegated Credentials for (D)TLS [Електронний ресурс] / R.Barnes, S. Iyengar, N. Sullivan, E. Rescorla – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/draft-ietf-tls-subcerts-15.

6. OAuth 2.0 [Електронний ресурс] – Режим доступу до ресурсу: https://oauth.net/2/.

7. What is LDAP? [Електронний ресурс] – Режим доступу до ресурсу: https://www.gracion.com/server/whatldap.html.

8. What is OpenID Connect [Електронний ресурс] – Режим доступу до ресурсу: https://openid.net/developers/how-connect-works/.

9. The Difference Between LDAP and SAML SSO [Електронний ресурс] – Режим доступу до ресурсу: https://jumpcloud.com/blog/difference-ldap-saml-sso.

10. Ferraiolo D. Role-Based Access Controls [Електронний ресурс] / D. Ferraiolo, R. Kuhn – Режим доступу до ресурсу: https://csrc.nist.gov/pubs/conference/1992/10/13/rolebased-access-controls/final.

11. What is ELK Stack? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/elk-stack/.

12. SOC 2® - SOC for Service Organizations: Trust Services Criteria [Електронний ресурс] – Режим доступу до ресурсу: https://www.aicpa-cima.com/topic/audit-assurance/audit-and-assurance-greater-than-soc-2.
13. OWASP Top 10:2021 [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/Top10/.
14. General Data Protection Regulation [Електронний ресурс] – Режим доступу до ресурсу: https://gdpr-info.eu/.
15. Health Insurance Portability and Accountability Act of 1996 (HIPAA) [Електронний ресурс] – Режим доступу до ресурсу: https://www.cdc.gov/phlp/publications/topic/hipaa.html.
16. JetBrains. Microservices [Електронний ресурс] / JetBrains – Режим доступу до ресурсу: https://www.jetbrains.com/lp/devecosystem-2022/microservices/.
17. Spring Framework Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-framework/reference/.
18. Official Spring Framework site [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/.
19. Official Spring Boot documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/.
20. Spring Cloud [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-cloud.
21. H2 Database Engine [Електронний ресурс] – Режим доступу до ресурсу: https://www.h2database.com/html/main.html.
22. PostgreSQL: The World\'s Most Advanced Open Source Relational Database [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/.
23. Spring Cloud Gateway [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-cloud-gateway.
24. JWT Handbook [Електронний ресурс] – Режим доступу до ресурсу: https://auth0.com/resources/ebooks/jwt-handbook.
25. Keycloak [Електронний ресурс] – Режим доступу до ресурсу: https://www.keycloak.org/.

26. Postman API Platform [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com/.

27. Docker: Accelerate how you build, share, and run applications [Електронний ресурс] – Режим доступу до ресурсу: https://www.docker.com/.

28. Production-Grade Container Orchestration with Kubernetes [Електронний ресурс] – Режим доступу до ресурсу: https://kubernetes.io/.

29. Docker Compose overview [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/compose/.

30. Spring Boot Actuator Web API Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-boot/docs/current/actuator-api/htmlsingle/.

31. Burnett M. Blocking Brute Force Attacks [Електронний ресурс] / Mark Burnett – Режим доступу до ресурсу: https://web.archive.org/web/20161203020306/http://www.cs.virginia.edu/~csadmin/gen_support/brute_force.php.

32. Microsoft. SQL Injection [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: https://web.archive.org/web/20130802094425/http://technet.microsoft.com/en-us/library/ms161953(v=sql.105).aspx.

33. OWASP. Session fixation [Електронний ресурс] / OWASP – Режим доступу до ресурсу: https://owasp.org/www-community/attacks/Session_fixation#:~:text=Session%20Fixation%20is%20an%20attack,specifically%20the%20vulnerable%20web%20application..

34. OWASP. A3:2017-Sensitive Data Exposure [Електронний ресурс] / OWASP – Режим доступу до ресурсу: https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure.

35. Fact Sheet: Machine-in-the-Middle Attacks [Електронний ресурс] – Режим доступу до ресурсу: https://www.internetsociety.org/resources/doc/2020/fact-sheet-machine-in-the-middle-attacks/.

36. IBM. What is a zero-day exploit? [Електронний ресурс] / IBM – Режим доступу до ресурсу: https://www.ibm.com/topics/zero-day.