МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ С.В. Казмірчук
«_____»_____ 20__ р.

На правах рукопису
УДК 004.056:004.4‘2(079.2)

ДИПЛОМНА РОБОТА

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «БАКАЛАВР»

Тема: Засоби захисту клієнт-серверної архітектури з JWT аутентифікацією на основі REST API

Виконавець: Н.О. Калінчук
Керівник: к.т.н., доцент І.І. Пархоменко
Нормоконтролер: к.т.н., доцент І.І. Пархоменко

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет: Кібербезпеки, комп'ютерної та програмної інженерії

Кафедра: Комп'ютеризованих систем захисту інформації

Освітній ступінь: Бакалавр

Спеціальність: 125 «Кібербезпека»

Освітньо-професійна програма: «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ С.В. Казмірчук

«__» _____ 20__ р.

ЗАВДАННЯ

на виконання дипломної роботи

здобувача вищої освіти Калінчука Назара Олександровича

1. Тема: *Засоби захисту клієнт-серверної архітектури з JWT аутентифікацією на основі REST API*
затверджена наказом в.о. ректора від «26» квітня 2021 р. № 652/ст.

2. Термін виконання: з 10.05.2021 р. по 20.06.2021 р.

3. Вихідні дані: проаналізувати існуючі системи ризиків на основі клієнт-серверної архітектури; розробити методику, алгоритм та програмне забезпечення системи захисту клієнт-серверної архітектури.

4. Зміст пояснювальної записки: аналіз методів захисту клієнт-серверних архітектур аналіз вимог до системи захисту клієнт-серверної архітектури; реалізація захисту клієнт-серверної архітектури.

КАЛЕНДАРНИЙ ПЛАН

виконання дипломної роботи

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1.	Уточнення постановки задачі	19.04.2021	*Виконано*
2.	Аналіз літературних джерел	20.04.2021	*Виконано*
3.	Обґрунтування вибору рішення	22.04.2021	*Виконано*
4.	Збір інформації	23.04.2021	*Виконано*
5.	Дослідження засобів захисту клієнт-серверної архітектури	04.05.2021	*Виконано*
6.	Розробка захисту клієнт-серверної архітектури з JWT аутентифікацією	16.05.2021	*Виконано*
7.	Розробка алгоритму та забезпечення захисту систем аналізу захисту клієнт-серверної архітектури	22.05.2021	*Виконано*
8.	Перевірка на антиплагіат	05.06.2021	*Виконано*
9.	Оформлення і друк пояснювальної записки	08.06.2021	*Виконано*
10.	Оформлення презентації	09.06.2021	*Виконано*
11.	Отримання рецензій від рецензента	10.06.2021	*Виконано*

Здобувач вищої освіти _____ Н. Калінчук
(підпис, дата)

Керівник дипломної роботи _____ І. Пархоменко
(підпис, дата)

РЕФЕРАТ

Дипломна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків і має 43 сторінки основного тексту, 2 рисунка, 7 таблиць. Список використаних джерел містить 20 найменувань і займає 3 сторінки. Загальний обсяг роботи 43 сторінки.

Метою роботи є підвищення рівня захищеності клієнт-серверної архітектури за рахунок JWT аунтифікації на основі REST API.

В роботі вирішено задачу засобів захисту клієнт-серверної архітектури.

В роботі розроблено алгоритм та програмне забезпечення для аналізу нефункціональних вимог до засобів захисту архітектури.

Розроблений метод та програмне забезпечення відносяться до галузі безпеки клієнт-серверної архітектури і можуть бути використані для підвищення рівня аутентифікації.

Ключові слова: клієнт-серверна парадигма, архітектура, аутентифікація, java-пакети, клієнт .

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

IBM - International Business Machines

FTP - File Transfer Protocol

NFC - Near field communication

MVC - Model, View та Controller

IP - Internet Protocol

API - Application programming interface

JSON - JavaScript Object Notation

REST API - Representational State Transfer

UNIX - Uniplexed Information and Computing System

SAML - Security assertion markup language

ЗМІСТ

ВСТУП

Клієнт - серверні обчислення в даний час є одним із ключових слів у комп'ютерній індустрії. Середовище клієнт / сервер можна визначити як відкрите системне середовище. Ця відкритість середовища клієнт - сервер робить його дуже популярним середовищем для роботи. Оскільки інформація надзвичайно доступна для клієнта - сервера, виникають певні проблеми безпеки. Для вирішення цієї певної потреби в захищеному середовищі клієнт - сервер необхідно спочатку визначити середовище клієнт - сервер. Це досягається шляхом визначення трьох можливих способів розділення програм у середовищі клієнт - сервер. Безпека або захищені системи мають різне значення для різних людей. Ця дисертація визначає шість атрибутів інформації, які слід підтримувати, щоб мати захищену інформацію. Для певних середовищ деякі з цих атрибутів можуть бути непотрібними або менш важливими.

Різні техніки та заходи безпеки обговорюються та класифікуються з точки зору розділів клієнт - сервер та атрибути безпеки, які ними підтримуються. Це представлено у формі матриці та забезпечує легке посилання для прийняття рішень щодо заходів безпеки в середовищі клієнт - сервер з метою захисту конкретного аспекту інформації. Обговорюється важливість політики безпеки і, зокрема, вплив середовища клієнт - сервер на таку політику, і демонструється, що структура може допомогти у розробці політики безпеки для середовища клієнт - сервер. Крім того, ця дисертація визначає систему електронного документообігу як тематичне дослідження. Показано, що середовище клієнт - сервер є відповідним середовищем для такої системи. Потреби та проблеми безпеки визначаються та класифікуються за ознаками безпеки. Вирішення проблем обговорюються з метою забезпечення достатньо безпечного середовища системи електронного документообігу.

Інформація по всьому світу розширюється феноменальними темпами. Для того, щоб залишатися в бізнесі, потрібно встигати за цим інформаційним вибухом. Ця потреба в актуальній інформації змушує підприємства все більше

покладатися на можливості, надані їм комп'ютерною індустрією. Розвиток в галузі комп'ютерної індустрії також не застоювався. Фактично, вибух інформації та вибух технологій у комп'ютерній індустрії йдуть рука об руку. Результатом усього цього зростання є те, що сьогодні у бізнес-спільноті існує велика колекція різних технологій. Потреба цих технологій взаємодіяти одна з одною поставила перед комп'ютерною індустрією багато проблем. Напрямки досліджень, такі як обчислення клієнт / сервер, об'єднані бази даних та мережі, відкрили шлях для ефективного співробітництва цих різних технологій. Ці "нові" технології ведуть до підвищення рівня обміну даними та іншими ресурсами між різними користувачами. Цей кооперативний спосіб роботи призвів до більшої відкритості систем, завдяки чому дані є більш доступними та доступними. Це зростаюче значення відкритості в системах покладає певний тягар на безпеку системи. Поняття "відкриті системи" та "безпечні системи", судячи з усього, суперечать одне одному. Наголос у відкритих системах полягає в обміні інформацією та іншими ресурсами за допомогою мережі. Спочатку ці мережі були розроблені для зручності використання, а не в основному для цілей безпеки. Архітектуру клієнт - сервер можна вважати конкретним випадком відкритої системи, оскільки сервер надає клієнту дані та інші послуги через мережу.

Сьогодні ви навряд чи знайдете якийсь технічний журнал, звіт, періодичний випуск чи бюлетень, які не були б переповнені похвалами технології клієнтського сервера: "клієнт - сервер зробив це", "клієнт - сервер робить це", "клієнт - сервер зробить це "і т. д. Отже, перше питання, яке виникає у свідомості людини-початківця в області комп'ютера, полягає в тому, що таке технологія клієнт - сервер? Давайте зрозуміємо це просто. У Всесвіті з моменту заснування людського суспільства існує система клієнт - сервер. Клієнт просить щось корисне для виробника або постачальника, а виробник повертає запитані товари чи товари. У цьому прикладі клієнтом може бути клієнт, а виробником або постачальником - сервер. Студент і викладач також можуть розглядатися як система клієнтських серверів, коли студент (клієнт)

задає питання (робить запит), а вчитель (сервер) робить (відповідає) все можливе, щоб задовольнити свою спрагу знань. Іншими словами, система «Клієнт - Сервер» - це не що інше, як плідна двостороння комунікація між двома сторонами, що мають спільні інтереси. Яка відповідність системи Клієнт - Сервер комп'ютеру? І чому це обговорювалось у цій тезі? Клієнт - сервер виступає як гаряче поле в технологічній галузі комп'ютерного програмного забезпечення і конкурує з основними технологіями IBM. У комп’ютерній термінології клієнт - це програмне забезпечення, яке робить запит, а сервер - це інтелектуальне програмне забезпечення, яке ідентифікує запит і повертає відповідні результати.

РОЗДІЛ 1. КЛІЄНТ СЕРВЕРНА ВЗАЄМОДІЯ ТА ЗАСОБИ АУТЕНТИФІКАЦІЇ

1.1. Клієнт-серверна парадигма

Клієнт є членом класу програм у сучасній парадигмі архітектури програмного забезпечення, яка називається архітектура програмного забезпечення клієнт - сервер, в якій усі додатки мають дві складові: клієнтський компонент та серверний компонент із наступними типовими характеристиками.

Клієнт виконується лише тоді, коли в ньому є потреба, тобто він викликається за запитом.

Усі діалоги в системі клієнт - сервер завжди ініціюються клієнтом. Таким чином, клієнт є активною сутністю.

Оскільки клієнт більшу частину часу ініціює діалог, він підтримує інтерфейси з людьми, завдяки яким клієнт завжди працює на передньому плані.

Подібно до того, як виробник або постачальник послуг може мати багато клієнтів, у будь-який момент часу може бути безліч екземплярів клієнтського компонента програми, що працює на 37 комп'ютерах у всій мережі, і всі вони підключаються лише до одного або декількох серверів.

Клієнт зазвичай працює на одну людину і виконує одну роботу за іншою послідовно, як користувач наказує їй. Що таке сервер?

Сервер є членом класу програм у сучасній парадигмі архітектури програмного забезпечення, що називається архітектура клієнт - серверне програмне забезпечення, в якій усі додатки мають два компоненти: клієнтський компонент та серверний компонент, з наступними типовими характеристиками:

У багатьох випадках сервер повинен надавати численні типи послуг, які змушують сервер споживати багато важливих системних ресурсів. Завдяки цьому сервер найчастіше є централізованим, як правило, на великому комп’ютері, достатньому для забезпечення достатньої їжі, щоб задовольнити голод сервера з точки зору системних ресурсів.

сервер - це пасивна сутність. Він робить мало або нічого, поки не отримає явний запит від клієнта щось зробити.

Для того, щоб надати послугу певній кількості клієнтів, сервер, як правило, повинен постійно працювати в очікуванні запитів, на відміну від клієнта, який викликається за запитом. Сервер рідко взаємодіє безпосередньо з людьми. Найчастіше йому доводиться мати справу з іншими програмами (клієнтами). Тому зазвичай сервер постійно працює за завісою.

Сервер може бути двох типів: один, який одночасно обробляє кілька запитів, а другий відповідає, щоб приймати запити. Паралельний сервер паралельно обробляє кілька запитів, тоді як послідовний сервер відповідає клієнтам один за одним.

1.2. Принципи клієнт-серверної архітектури

Клієнт - серверна архітектура вражає всіх учасників - від теоретиків до розробників і до кінцевих користувачів. Зрозуміло, що термін "клієнт - сервер" означає, що клієнти та сервери - це окремі логічні сутності, які працюють разом, як правило, через мережу, для виконання завдання. Клієнт / сервер - це не просто клієнт і сервер, що спілкуються через мережу; але клієнт / сервер має асинхронні та синхронні методи обміну повідомленнями за допомогою проміжного

програмного забезпечення для спілкування через мережу. Ось традиційні приклади системи клієнт - сервер. Як вже згадувалося раніше, всесвітня павутина працює за популярною моделлю парадигми клієнт - сервер. По суті, є три компоненти, які разом утворюють всесвітню павутину, носій, який вивів цей документ на ваш екран. По суті, є три компоненти, які разом утворюють всесвітню павутину: інтернет, веб-сервер та веб-клієнт. Технічно кажучи, Інтернет, складається з проводів, машин та мережевого програмного

забезпечення, яке з'єднує багато різнорідних комп'ютерних систем у всьому світі між собою.

Веб-сервер - це програма, що працює на комп’ютері з єдиною метою, яка обслуговує документи на інші комп’ютери за запитом веб-клієнта. Оскільки сервер не виконує складних обчислень, він виконує мінімальний обсяг роботи і працює лише тоді, коли запитується документ, він створює мінімальне навантаження на комп'ютері, на якому він працює. Інформаційні сервери працюють на комп'ютерах, підключених до Інтернету, і виконують комп'ютерне програмне забезпечення, яке доставляє інформацію за запитом користувачів, підключених до загальнодоступного Інтернету.

Загальновживані інформаційні сервери в Інтернеті сьогодні були коротко обговорені в наступному параграфі. Сервери всесвітньої павутини в основному доставляють дані, які люди можуть легко отримати, такі як гіпертекст та мультимедіа. Через базовий протокол гіпертекстового транспорту вони також називаються серверами http. Сервери Gopher є безпосередніми попередниками серверів World Wide Web. Вони подають користувачеві файли у розподілених архівах як ієрархічні меню. Використовуючи клієнт gopher, коли користувач вибирає файл із цього меню. Якщо це текстовий файл, він відображається на екрані. Якщо дані містяться у форматі, відмінному від тексту, наприклад зображення, файл переноситься на локальний комп'ютер, де користувач повинен використовувати окрему програму для перегляду або використання.

Після прочитання чи завантаження файлу клієнт gopher завжди відображає попереднє меню, з якого користувач вибрав файл. Сервери транспортного

протоколу FTP або File 41: дозволяють клієнтам FTP копіювати файли будь-якого типу, такі як вихідні та виконувані програми, зображення, текст тощо між клієнтом та серверами. FTP дозволяє користувачеві використовувати команди та імена файлів для надсилання, отримання з них та керування ними файлів та каталогів на віддаленому комп'ютері. Коли ви отримуєте файл із віддаленого комп’ютера за допомогою FTP, вам доведеться викликати окрему програму

після сеансу FTP, щоб переглянути цей файл (текстовий редактор, переглядач зображень тощо). Графічні FTP-клієнти для Windows або Macintosh позбавляють користувача від необхідності вивчати більшість команд FTP, дозволяючи користувачеві просто перетягувати файли до та з каталогу, ніби це локальний диск на машині. FTP вважається найефективнішим і найшвидшим способом передачі файлів з однієї машини на іншу машину. Сервери NNTP або Network News Transport Protocol надають групи новин Usenet та статті. Прості протоколи поштового транспортного протоколу (SMTP) надсилають та отримують повідомлення електронної пошти. Сервер Арчі здійснює пошук в індексах файлів FTP для файлів, якщо їм надано ім’я файлу або фрагмент імені. Сервер Veronica здійснює пошук у меню сусликів для слів чи фраз. Сервери Telnet дозволяють користувачеві Інтернету входити та проводити сеанс терміналу на віддаленому комп'ютері, на якому запущений сервер, з будь-якої точки мережі. Зазвичай ці сесії є сесіями терміналів UNIX, що проводяться через емулятори терміналів "VT100". Сервер telnet можна використовувати для різноманітних інтерактивних додатків на основі символів. Інформаційні сервери широкої області (WAIS) здійснюють розподіл заздалегідь проіндексованих обсягів тексту для слів і фраз та ранжують результати на основі оцінки - наскільки кожен документ відповідає критерію пошуку.

Веб-клієнти або веб-браузери - це третій важливий компонент Всесвітньої павутини. Веб-клієнт - це частина програмного забезпечення, яка взаємодіє з користувачем і запитує документи з сервера, коли користувач виконує команди. Веб-браузери, такі як Netscape Navigator та Microsoft Internet Explorer, революціонізували спосіб обміну інформацією через Інтернет. Можливості цих браузерів позбавляють користувача від багатьох обтяжень. Раніше для доступу до серверів, згаданих у попередньому пункті, користувачеві потрібно було використовувати окрему програму для кожного типу сервера. Щоб отримати доступ до сервера gopher, користувачеві потрібно було запустити клієнтську програму gopher. Щоб отримати доступ до FTP-сервера, потрібно було запустити ваш FTP-клієнт. Для пошуку файлу за допомогою Archie або

Veronica був необхідний запуск або Archie, або клієнта Veronica. Оскільки браузери World Wide Web є багатомовними; вони можуть спілкуватися з усіма перерахованими вище серверами та іншими. Це позбавляє користувача складнощів необхідності вивчати та запускати окремий клієнт для кожного сервера, який він хоче використовувати.

Однак для ефективного використання цього сервера відповідні виділені клієнти є більш корисними. Наприклад, передача файлу через мережу відбувається набагато швидше, використовуючи клієнти FTP, а не веб-браузери. Браузери World Wide Web мають потужні можливості для підтримки графічних інтерфейсів користувача. Багато з цих вищевказаних серверів вимагають вивчення загадкової мови команд або введення команд UNIX. У веб-браузері для виконання команд достатньо клацання миші або перетягування. Браузер опікується основними мережевими комунікаціями, інтерфейсами та командами, щоб передати те, що запитував користувач. Веб-браузери дозволяють організовувати довільну форму та перехресні посилання та посилання на інформацію, яка називається гіпертекстом, гіпермедіа чи гіперпосиланням, використовуючи такі мови, як HTML.

Клієнт-серверні системи можуть бути реалізовані за допомогою сокетів, віддалених викликів процедур (RPC), віддаленого доступу до даних (RDA) або обробки повідомлень у черзі. Використання сокетів вимагає багато роботи з кодуванням навіть для простої системи клієнт / сервер; але дуже легко налагоджувати програми. Клієнт пише запит у визначеному форматі та пише у сокет. Сервер на іншому кінці отримує запит від сокета, до якого він зв’язаний, аналізує та обробляє запит і надсилає назад відповідну відповідь клієнту через сокет. Була використана архітектура клієнт / сервер, використана в дисертації про реалізацію KEPI, механізм сокета. Через низький рівень роботи він мало використовується в практиці. Віддалений виклик процедур або RPC реалізує механізм, який дозволяє клієнтському процесу викликати віддалено розташований серверний процес. Сервер виконує процедуру і надсилає

результат назад у відповідь. RPC широко застосовується до реальних проблем від простого до складного.

Цей розподіл обробки зменшує мережевий трафік і покращує продуктивність. Автономність сайту також може бути збільшена шляхом модифікації процедур локально виконуваних програм. "Механізм RPC видає, що клієнт безпосередньо викликає процедуру, розташовану на віддаленому сервері, ніби викликає локальну процедуру. Насправді, клієнтська програма викликає локальну процедуру заглушки замість фактичної віддаленої процедури. Клієнт заглушку потім отримує необхідні параметри з адресного простору клієнтів, переводить їх за необхідності у стандартне представлення мережевих даних (NDR), а потім використовує API бібліотеки RPC для відправки запиту та параметрів на сервер. сторона, функції API виконання бібліотеки RPC приймають запит і викликають заглушку сервера. Заглушка сервера переміщує параметри з мережевого буфера і перетворює їх у формат, очікуваний сервером. Стерб сервера потім викликає сервер, передаючи його Параметри. Той самий механізм використовується, навпаки, для повернення даних із сервера клієнту. Віддалені дані. Парадигма Access (RDA) дозволяє клієнтським програмам та / або інструментам кінцевих користувачів видавати спеціальні SQL-запити щодо баз даних, розташованих віддалено. Для застосунків баз даних широко використовується RDA. В обробці повідомлень, що перебуває в черзі (QMP), зв'язок між клієнтом і сервером відбувається через чергу. Клієнт зберігає свій запит у черзі і чекає відповіді. Коли сервер вільний, він по черзі отримує запити з черги, клієнти поміщають свої запити та обробляють запит. Сервер зберігає результати обробки в іншій черзі для доступу клієнтів. Викликання віддалених методів Java (RMI), як механізм RPC, дозволяє викликати віддалені методи з локальної машини через мережу або Інтернет. Віддалене виклик методів Java зараз стає популярним переважно завдяки переносимості мови Java.

1.3. Засоби аутентифікації

Даний метод аутентифікації використовується зазвичай для розподілених систем Single Sign-On (SSO), де аутентифікація відбувається за рахунок іншого сервісу, наприклад здійснення аутентифікації через обліковий запис соціальних мереж. Соціальні мережі виступають в ролі сервіса аутентифікації. Ідея цього методу заключається в тому, що identity provider (IP), наприклад соціальні мережі, надають аутентифікаційні дані у вигляді токену до service provider (SP), що являється нашим застосунком, в якому здійснються аутентифікація. І застосунок використовує даний токен для аутентифікації і авторизації користувача. В загальному вигляді процесс аутентифікації для активного клієнта, наприклад IOS або Android, які можуть виконувати запрограмовану послідовність дій, виглядає так: клієнт аутентифікується у IP способом який цей застосунок підтримує, далі клієнт просить IP надати йому токен для аутентифікації для SP- застосунку. IP створює токен та надсилає його клієнту і клієнт аутентифікується, завдяки токену. А для браузера, що являється пасивним клієнтом, тобто може лише відображати сторінки, на які користувач здійснив запит. Процесс аутентифікаціїї виконується автоматично, переправлючи браузер між IP та SP. Токен –це структура даних, що складається з інформації: срок дії токена , відправник, можливий отримувач токену та деяка інформація о користувачі, що здійснив запит на створення токену. Токен для збереження цілістності даних і захисту від несанкціонованого змінення даних додатково підписується. Для підтвердження аутентифікації SP-застосунок спочатку здійснює перевірки для токену. Серед яких перевірка IP-застосунка, який видав токен, перевірка можливого отримувача токену, срок дії та цілісність токену. Після успішної перевірки SP-застосунок аутентифікує користувача.

SWT є одним з наспростіших форматів, що складається з Issuer, Audience, ExpiresOn и HMACSHA256 та їх значень у форматі кодування HTML form. Токен підписується симметричним ключем і обидва застосунки IP та SP

повинні мати цей ключ, щоб створити та перевірити токен відповідно. JWT складається з трьох блоків, що розділяються точками. Блоки представляють собою заголовок, набір полей та підпис. Перші два блока записуються у форматі JSON і кодуються у формат base64 додатково. Набір полей зазвичай має значення таких параметрів, як з Issuer, Audience, ExpiresOn та інших. Підпис генерується завдяки симетричної криптографії та асиметричної. SAML складається з інформації о емітенті, суб'єкті, що потрібна для створення та перевірки токену. Інформація записана у XML-форматі. Підпис здійснюється завдяки асиметричній криптографії. У даному токені є особливість, що підтверджує власника токену і допомагає уникнути атаки «людини по середині» у незахищенних каналах. Для даного методу аутентифікації використовуються стандарти, що описують протокол взаємодії між клієнтами та IP і SP застосунками, також, як і формат надсилаємих токенів. До таких стандартів належать:

- стандарт SAML;
- стандарт WS-Trust;
- стандарт WS-Federation;
- стандарт OAuth;
- стандарт OpenID Connect.

Стандарт SAML складається з assertions, protocols, bindings, profiles, де assertions формат токенів стандарту SAML у форматі XML, protocols являється 29 набором повідомлень, наприклад запит на створення нового токену, отримання існуючих токенів, вихід з системи, управління ідентифікаторами користувачів тощо, bindings – це механізми передачі повідомлень через транспортні протоколи, profiles – типові сценарії стандарту, що визначають набір перших трьох пунктів assertions, protocols та bindings. Стандарт WS-Trust описує інтерфейс сервісу авторизації Secure Token Service (STS). Стандарт працює по протоколу SOAP і підтримує створення ті анулювання токенів та використовує зазвичай SAML-токены. Стандарт WS-Federation відноситься до механізму обміну токенами. При цьому WS-Federation розширює функції

сервіса STS. Він вирішує ті ж самі задачі, що й SAML, але має інші підходи і реалізацію. Стандарт OAuth (Open Authorization) на відміну від інших стандартів не описує протокол аутентифікації користувача. Він визначає механізм отримання доступа одного застосунка до іншого від імені користувача. Роботу стандарта можна описати так:

1. Користувач дає згоду застосунку на доступ до певного ресурсу у виді гранту.

2. Застосунок отримує токен доступу до ресурсу від сервера авторизації в обмін на свій грант.

3.Застосунок використовує токен для отримання даних від сервера ресурсів.

1.4. Типи аутентифікацій

Даний метод аутентифікації використовується зазвичай для розподілених систем Single Sign-On (SSO), де аутентифікація відбувається за рахунок іншого сервісу, наприклад здійснення аутентифікації через обліковий запис соціальних мереж. Соціальні мережі виступають в ролі сервіса аутентифікації. Ідея цього методу заключається в тому, що identity provider (IP), наприклад соціальні мережі, надають аутентифікаційні дані у вигляді токену до service provider (SP), що являється нашим застосунком, в якому здійснються аутентифікація. І застосунок використовує даний токен для аутентифікації і авторизації користувача. В загальному вигляді процесс аутентифікації для активного клієнта, наприклад IOS або Android, які можуть виконувати запрограмовану послідовність дій, виглядає так: клієнт аутентифікується у IP способом який цей застосунок підтримує, далі клієнт просить IP надати йому токен для аутентифікації для SP- застосунку. IP створює токен та надсилає його клієнту і клієнт аутентифікується, завдяки токену. А для браузера, що являється пасивним клієнтом, тобто може лише відображати сторінки, на які користувач здійснив запит. Процесс аутентифікаціїї виконується автоматично,

переправлючи браузер між IP та SP. Токен –це структура даних, що складається з інформації: срок дії токена , відправник, можливий отримувач токену та деяка інформація о користувачі, що здійснив запит на створення токену. Токен для збереження цілістності даних і захисту від несанкціонованого змінення даних додатково підписується. Для підтвердження аутентифікації SP-застосунок спочатку здійснює перевірки для токену. Серед яких перевірка IP-застосунка, який видав токен, перевірка можливого отримувача токену, срок дії та цілісність токену. Після успішної перевірки SP-застосунок аутентифікує користувача.

SWT є одним з наспростіших форматів, що складається з Issuer, Audience, ExpiresOn и HMACSHA256 та їх значень у форматі кодування HTML form. Токен підписується симметричним ключем і обидва застосунки IP та SP повинні мати цей ключ, щоб створити та перевірити токен відповідно. JWT складається з трьох блоків, що розділяються точками. Блоки представляють собою заголовок, набір полей та підпис. Перші два блока записуються у форматі JSON і кодуються у формат base64 додатково. Набір полей зазвичай має значення таких параметрів, як з Issuer, Audience, ExpiresOn та інших. Підпис генерується завдяки симетричної криптографії та асиметричної. SAML складається з інформації о емітенті, суб’єкті, що потрібна для створення та перевірки токену. Інформація записана у XML-форматі. Підпис здійснюється завдяки асиметричній криптографії. У даному токені є особливість, що підтверджує власника токену і допомагає уникнути атаки «людини по середині» у незахищенних каналах. Для даного методу аутентифікації використовуються стандарти, що описують протокол взаємодії між клієнтами та IP і SP застосунками, також, як і формат надсилаємих токенів. До таких стандартів належать:

- стандарт SAML;
- стандарт WS-Trust;
- стандарт WS-Federation;
- стандарт OAuth;

- стандарт OpenID Connect.

Стандарт SAML складається з assertions, protocols, bindings, profiles, де assertions формат токенів стандарту SAML у форматі XML, protocols являється 29 набором повідомлень, наприклад запит на створення нового токену, отримання існуючих токенів, вихід з системи, управління ідентифікаторами користувачів тощо, bindings – це механізми передачі повідомлень через транспортні протоколи, profiles – типові сценарії стандарту, що визначають набір перших трьох пунктів assertions, protocols та bindings. Стандарт WS-Trust описує інтерфейс сервісу авторизації Secure Token Service (STS). Стандарт працює по протоколу SOAP і підтримує створення ті анулювання токенів та використовує зазвичай SAML-токены. Стандарт WS-Federation відноситься до механізму обміну токенами. При цьому WS-Federation розширює функції сервіса STS. Він вирішує ті ж самі задачі, що й SAML, але має інші підходи і реалізацію. Стандарт OAuth (Open Authorization) на відміну від інших стандартів не описує протокол аутентифікації користувача. Він визначає механізм отримання доступа одного застосунка до іншого від імені користувача. Роботу стандарта можна описати так:

1. Користувач дає згоду застосунку на доступ до певного ресурсу у виді гранту.

2. Застосунок отримує токен доступу до ресурсу від сервера авторизації в обмін на свій грант.

3.Застосунок використовує токен для отримання даних від сервера ресурсів.

1.5. Засоби авторизації користувача.

З початкового періоду смартфонів характер аутентифікації користувачів залишався в основному незмінним. Персональний ідентифікаційний номер (ПІН) широко використовується як захисний пункт входу на мобільних пристроях. Подібна ситуація спостерігається і на робочих столах, де підходи до аутентифікації в основному покладаються на секретні знання. Однак

користувачі мобільних пристроїв можуть використовувати різні механізми для блокування різних аспектів функціональності. Наприклад, телефони Windows Mobile підтримують два різні механізми аутентифікації, наприклад, один для захисту мобільного пристрою та інший для захисту SIM-карти користувача (модуль ідентифікації абонента). Захист передньої лінії слухавки забезпечується аутентифікацією на рівні пристрою, особливо коли пристрій увімкнено. Для цього аутентифікація на рівні пристрою захищає доступ до програм та даних, що зберігаються в телефоні. Тоді як аутентифікація на рівні SIM забезпечує безпеку вмісту SIM та облікового запису стільникової мережі. В іншому випадку SIM-картку можна вилучити з авторизованого пристрою та використовувати в будь-якому несанкціонованому пристрої. Таким чином, аутентифікація на рівні SIM-картки ефективно регулює використання стільникових даних та обмежує здійснення голосових дзвінків. Однією з унікальних відмінностей між смартфонами та настільними комп'ютерами є те, що смартфони - це виключно особисті пристрої, якими зазвичай користується один користувач. Тоді як настільні комп'ютери часто використовують різні користувачі.

Невеликі смартфони вразливі до несанкціонованого використання через втрати та крадіжки. Отже, аутентифікація та ідентифікація смартфона відрізняється від методів аутентифікації на робочому столі. Потенційні зловмисники отримують справний пристрій, що працює та працює, у випадку втрати або крадіжки, оскільки смартфони, як правило, завжди вмикаються.

Користувача перед використанням потрібно аутентифікувати на телефоні, щоб запобігти несанкціонованому використанню та забезпечити захист збережених даних. Телефон повинен бути заблокований після того, як користувач виконає свої завдання, з цієї причини операційні системи смартфонів зазвичай забезпечують короткі, але часті сеанси. Як результат, значний вплив на зручність використання смартфонів залежить від обраного користувачем способу аутентифікації. Швидкість та комфорт використання є

ключовими двома факторами для більшості користувачів під час прийняття рішення щодо методів аутентифікації.

Оскільки смартфони дуже гнучкі для рухів, вони часто використовуються в сумнівних місцях з величезною кількістю потенційних спостерігачів. Зручний та простий підхід до аутентифікації потребує незалежності від середовища, в якому використовується телефон, крім забезпечення безпеки. Для аутентифікації на смартфонах нижче наведено кілька вимог.

Користувач повинен пройти аутентифікацію перед кожним сеансом.

Прохідний спостерігач не повинен нічого дізнатися про секрет, який використовується для аутентифікації.

Аутентифікація повинна бути безпечною та швидкою з мінімальними зусиллями користувачів Методи аутентифікації, які сьогодні доступні на практиці, не здатні виконати всі ці вимоги. Аутентифікація за допомогою PIN-коду / пароля - здебільшого поширений метод, який вимагає трудомістких взаємодій користувачів (введення тексту знову і знову кожного разу під час розблокування телефону). Тому більшість користувачів віддають перевагу легкому набору тексту, короткому Паролю / ПІН-коду та, зрештою, менш захищеному, особливо коли вони використовуються в людних місцях. Графічний шаблон є однією з альтернатив аутентифікації на основі PIN-коду / пароля, яка має перевагу швидшого введення, але порівняно простіша для вивчення випадковими спостерігачами, і це загроза безпеці. Біометричні аутентифікації, такі як відбитки пальців або розпізнавання обличчя, все ще не встановлюються широко через додаткові витрати, і їх можна уникнути, використовуючи підроблений відбиток пальця або просту фотографію.

Аутентифікація на основі PIN-коду / пароля є прикладом аутентифікації на основі знань або “того, що ми знаємо”. Графічні шаблони / паролі крім запитань та відповідей є ще одним прикладом такого типу аутентифікації. Аутентифікація пристрою здійснюється за допомогою знань. Тут існує ймовірність того, що секретні знання переходять у несанкціоновані руки. Кожен, хто знає секрет для аутентифікації пристрою, може використовувати

його для аутентифікації. Аутентифікація виклику-відповіді також залежить від знань. Аутентифікація за допомогою пароля - це найпростіший приклад аутентифікації виклик-відповідь, коли запит пароля є викликом, а єдиною дійсною відповіддю є сам пароль. Графічні паролі - це спосіб уникнути цифрових / буквено-цифрових паролів, які важко запам’ятати, оскільки людський мозок здатний запам’ятовувати видимі зображення, а не складні рядки символів. Існують графічні паролі на основі відкликання та розпізнавання. Користувач повинен запам'ятати зображення в системах, що базуються на відкликанні. Системи, що базуються на розпізнаванні, вимагають від користувачів ідентифікації зображень, чи бачили вони зображення раніше. У цій системі зображення, які бачимо, вже слід розпізнавати, а не генерувати з пам'яті.

Аутентифікація на основі власності („те, що ми маємо“) включає приклад смарт-карт та електронних токенів. Ключ, який відкриває двері, є прикладом справжньої аутентифікації на основі власності. У двері може увійти кожен, у кого є ключ. Також можна мати копії ключа, щоб дозволити кільком людям увійти у двері. Теги з ідентифікацією радіочастот (RFID) або картки з магнітною смужкою є цифровим прикладом цього типу аутентифікації. Для зв'язку на короткі відстані стандартним для радіозв’язку є зв’язок ближнього поля до 10 см (NFC). Теги NFC, що базуються на аутентифікації права власності, повинні бути упритул до пристрою зчитування тегів NFC. Ідентифікаційний номер мітки NFC можна використовувати для порівняння. Генерація захищених паролів є перевагою аутентифікації на основі власності, але щоб залишатися у власності предмета, слід подбати про безпеку. Якщо предмет втрачено або викрадено, інша особа може успішно пройти аутентифікацію. Якщо копії предмета, створеного та залишеного непоміченим, це стає великою загрозою. Аутентифікація на основі власності ще не є практикою для аутентифікації смартфонів.

Тип аутентифікації або аутентифікації на основі послідовності включає біометричні методи аутентифікації. Приклади біометричних характеристик -

відбитки пальців, райдужна оболонка, обличчя, почерк, голоси, хода, жест тощо. Ці характеристики можна класифікувати як статичні та динамічні. Статичні методи фокусуються на тому, що таке людина, а динамічні - на тому, як людина щось робить чи робить. Відбитки пальців широко використовуються протягом багатьох десятиліть для особистої перевірки як техніки біологічного розпізнавання. Нещодавно автоматичне розпізнавання відбитків пальців було впроваджено у смартфонах декількома виробниками, що все ще не є широко доступним через додаткові витрати. Розпізнавання обличчя було єдиним механізмом по суті, раніше наданим android.

Речі, які можуть бути захоплені, беруть участь у захопленні атак. Серфінг на плечах, прослуховування, соціальна інженерія - приклади цього типу атак. Мистецтво, за допомогою якого людьми маніпулюють для здійснення певних дій або для розголошення конфіденційної інформації, відоме як соціальна інженерія. На рівні підприємств, як правило, люди не знають особисто всіх співробітників групи технічної підтримки, і соціальна інженерія зазвичай спостерігається в цьому середовищі. Іноді для отримання важливої інформації достатньо лише телефонного дзвінка. Спостереження за тим, як хтось вводить секретну інформацію, визначається як серфінг на плечі. Наприклад, жест часто використовується на смартфоні для аутентифікації, і його можна здогадатися досить легко. Шпигунське програмне забезпечення - це ще один спосіб захоплення, тобто зловмисне програмне забезпечення, яке накопичує інформацію користувачів без їхнього занепокоєння. Інформація про аутентифікацію також може бути включена в ці знання зловмисного програмного забезпечення. Аутентифікація на мобільних пристроях схильна до таких захоплюючих атак, таких як соціальна інженерія, серфінг на плечах та шпигунське програмне забезпечення. Підслуховування може стати фактором, якщо проводити аутентифікацію по бездротовій мережі, наприклад, за допомогою тегу NFC, проте це не проблема.

При атаках злому не потрібна взаємодія з аутентичними користувачами, що протилежне захопленню атак типу. Злом охоплює систематичні підходи, які

намагаються з'ясувати успішну аутентифікацію, яку приймає система. Якщо користувачам не вдається створити надійний пароль, то здогадування може бути успішним. Пароль, який є слабким і незахищеним, зазвичай легко запам’ятати, але його також легко вгадати. Вгадування атак, як правило, взаємопов’язані із соціальною інженерією, коли зловмисники хочуть і намагаються отримати інформацію про користувачів якомога більше. Спочатку зловмисник намагається використовувати слабкі та часто використовувані паролі, а отже, користувачі слабких паролів схильні до атак. Під час створення паролів рекомендується уникати використання особистих даних, які можуть включати дату народження, місце проживання, ім'я чоловіка / дружини або ім'я дитини і т.д. поєднання можливих символів. Атаки грубої сили та словникові атаки в сукупності утворюють гібридні атаки. Систематичні модифікації, такі як додавання кількох символів або перемикання великих та малих символів у паролях, здійснюються із ймовірного списку паролів. Як правило, атаки на словники та грубу силу виконуються автоматично. В принципі, їх можна виконувати в нашому контексті, але без автоматизації вони все ще залишаються в категорії відгадування.

Зловмисники можуть ввести аутентичних користувачів в оману, прикидаючись, використовуючи неправдиві дані. Під час фальсифікаційних атак хтось може виявитись справжнім користувачем, підробляючи дані. Підробка електронної пошти, підробка IP-адрес, підробка веб-сайтів та підробка посилань є прикладом деяких видів підробки. (Sametinger et al. 2012) Наприклад, створення веб-сайту, що обманює, який схожий на оригінальний веб-сайт, називається підміною веб-сайту. Метою підробки веб-сайтів зазвичай є отримання конфіденційної інформації, наприклад, даних кредитної картки. Особливою формою фальшивої атаки є атака "людина-в-середньому". При цьому типі атаки зловмисники встановлюють незалежний зв’язок між користувачами та серверами. Зловмисники можуть контролювати всю розмову між двома взаємопов'язаними сторонами, якщо вони можуть перервати всі повідомлення між цими сторонами. Таким чином, вони можуть отримати

доступ до конфіденційної інформації цих сторін, навіть якщо вона зашифрована. У цьому контексті дипломної роботи фішинг можна розглядати як проблему, тоді як неправдива програма, можливо, дає підроблений екран аутентифікації та затримує користувача для розкриття його / її даних. У контексті мобільних пристроїв атака та підробка людини посередині не підходить.

Крадіжки та копії - приклади фізичних атак. Речі, якими ми володіємо, можуть бути відібрані у нас, або можливе копіювання. Кабіни зловмисників не викрадають щось на зразок пароля, який є в нашій голові, але смарт-карту або смартфон можна вкрасти. Пароль можна вкрасти, лише якщо ми його десь запишемо, наприклад, на аркуші паперу, і інцидент може статися навіть без нашої турботи. Це також стосується дублікатів. Дайвінг-дайвінг також потрапляє до цієї категорії, оскільки зловмисник може знайти аркуш паперу із записаним паролем від сміття. Іншою формою фізичної атаки є апаратні маніпуляції. Це може спостерігатися удвоєнні. Скімер АТМ - типовий приклад, коли зловмисники копіюють карту АТМ, додаючи апаратне забезпечення до звичайного банкомату та стежачи за законним користувачем під час введення PIN-коду. У випадку мобільних пристроїв маніпуляції з апаратним забезпеченням навряд чи можуть бути непоміченими, але крадіжки, копії та занурення в смітник можна розглядати як загрозу в нашому контексті.

1.6. Висновки до першого розділу

Кожен метод аутентифікації має свої переваги і недоліки. На сьогоднішній день для більш ефективного захисту систем використовують комбінацію механізмів захисту, що знижує ризик з можливих недоліків і підвищує рівень безпеки системи, так званий двофакторна аутентифікація.

Проаналізовано сучасні методи авторизації користувачів. Були розглянуті усі види аутентифікації користувачів та стандарти авторизації та були визначені переваги та недоліки. Аналіз досліджень показав, що системи хмарних

обчислень використовують аутентифікації по ключ-доступу, а даний вид аутентифікації неможливо використовувати в системах з мікросервісною архітектурою.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ДО СИСТЕМИ ЗАХИСТУ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ

2.1. Функціональні вимоги

Функціональні вимоги в документі СВПЗ (специфікація вимог до програмного забезпечення) вказують на те, що повинна робити програмна система і як вона повинна функціонувати; це функції продукту, орієнтовані на потреби користувачів.

Оскільки документ СВПЗ містить детальний опис вимог до програмного забезпечення та закладає основи для технічних груп, інвесторів, менеджерів та розробників, розмежування функціональних вимог є важливою частиною його написання.

Існує кілька методів написання функціональних вимог, але найпоширеніший метод - це побудова історій користувачів та використання форматів історій користувачів: як ___ я хочу мати можливість ___ так, щоб ___.

Наприклад: як клієнт я хочу мати можливість переглянути всі найкращі продукти, щоб я міг вибрати найкращий, який мені підходить.

Технічна функція наведеного вище прикладу - це функція, яка дозволяє клієнтам переглядати найкращі продукти.

Як бачимо, історії користувачів ставлять фактичних кінцевих споживачів на перше місце, враховуючи їхні потреби та виконуючи роль їх аналога, і це ефективно визначає необхідні функціональні вимоги програмного продукту, оскільки вони зосереджуються на потребах користувачів.

Бізнес-правила. Що я хочу, щоб м система робила? Які функції мені потрібні, щоб я міг досягти своїх цілей? Потрібно визначити кожну системну діяльність для кожної функції в системі та розглянути всі типи функціональних вимог. Ось чому цей розділ, мабуть, буде найважливіщим серед інших, оскільки багато вимог можуть підпадати під цю категоризацію.

Виправлення, коригування та скасування транзакцій. Ці вимоги перевіряють вхід, зміну, видалення, скасування та перевірку помилок кожної транзакції.

Функції аутентифікації. Вони стосуються інформації, яку користувачі передають системі та рівня їх аутентифікації.

Рівні авторизації. Ці функції визначають різні рівні доступу до системи та вирішують, хто може змінювати, читати, оновлювати чи видаляти інформацію.

Відстеження аудиту. Відстеження аудиту - це процес відстеження важливих даних.

Зовнішні інтерфейси. Ці функції стосуються зовнішнього інтерфейсу систем, відмінних від основної системи.

Вимоги до сертифікаціїВаша організація може вимагати сертифікатів для роботи в системі, таких як сертифікати безпеки.

Вимоги щодо пошуку / звітності. Цей розділ вимог розповість, як користувачі можуть шукати та отримувати дані.

Історичні дані. Якщо ваша база даних є динамічною, ви збільшите кількість даних, тому вам потрібно визначити вимоги до зберігання, щоб вмістити ці дані.

Архівування. Дані вашої системи можуть перевищувати ваші можливості для зберігання, тому проекти повинні мати можливість архівувати дані для тривалого зберігання.

Відповідність, законодавчі або нормативні вимоги. Це закони, постанови уряду та навіть внутрішня політика, якої повинні дотримуватися організації та їх системи.

Алгоритми. Алгоритми фіксують будь-які формули або маніпуляції з елементами даних, які мають відбутися.

База даних. Елементи та формати, які слід використовувати, визначаючи, які дані потрібно зберігати в системі.

В таблиці 3.1. зібрані основні функціональні вимоги до веб-додатку з JWT-аутентифікацією.

Таблиця 3.1.

Об’єкт аналізу	Опис
Реєстрація	Користувач може зареєструватися в системі.
Реєстрація	Для реєстрації необхідно ввести логін пароль, та повтор паролю.
Авторизація	Для авторизації необхідно ввести логін та пароль.
Головна сторінка	Головну сторінку може бачити лише користувач та адміністратор.
Вихід з системи	Після виходу з системи, всі дані користувача стираються із сесії
Початкова сторінка	Початкову сторінку можуть бачити гість, користувач та адміністратор.
Ресурс системи	Доступ до сторінки ресурсу системи має лише адміністратор.

2.2. Нефункціональні вимоги

Більшість нових сучасних додатків зараз побудовані з використанням архітектури мікросервісів. Це очевидно, оскільки архітектура мікросервісу роз'єднує суб'єкти господарювання із слабо пов'язаним дизайном і дозволяє їм самостійно розвиватися / змінюватися чи масштабуватися.

Починаючи впроваджувати цю архітектуру, ми часто концентруємось лише на функціональній частині послуги. Існує багато нефункціональних вимог, які не враховуються з 1-го дня, а потім реалізуються після мінімального життєздатного продукту. Потім це загрожує технічним відставанням боргу і більше ніколи не підбирається для розробки. Як тільки бізнес розповсюджує цю систему для ширшої аудиторії, усі усвідомлюють компроміс, зроблений раніше.

Далі наведено ряд нефункціональних вимог до майбутньої клієнт-серверної системи.

Таблиця 3.2.

Об'єкт аналізу	Опис
Протокол аутентифікації	Клієнт-серверний веб-додаток має бути захищеним за протоколом JSON web token.
Структура аутентифікації	Структурою аутентифікації на серверній стороні має бути реалізація OAuth2.
Шифратор токена	JSON web token має бути зашифрованим за алгоритмом шифрування RSA-256.
Токен	Зафифрований токен має передаватися в шапці HTTP-запиту.
Токен	Зафифрований токен повинен мати префікс “Bearer”.
Сервер аутентифікації	В клієнт-серверній архітектурі має бути окремий сервер, котрий буде видавати та перевіряти токени.
Клієнт	Клієнтом має бути мікросервіс на мові Java.
Ресурс-сервер	Ресурс сервером має бути віддалений мікросервіс на мові Java.
БД	PostgreSQL
БД	База даних ма бути на стороні ресурс-серверу.

2.3. Висновки до другого розділу

Технічне завдання є невід’ємною частиною майбутнього проєкту, це буде співвідносити бажання головного замовника та можливості програмістів, це збереже час та усуне усі можливі конфлікти. Зрештою, технічне завдання охоплює коло задач, які необхідно виконати в рамках розробки системи обміну зашифрованими повідомленнями.

Для того, щоб розробники веб-додатків могли повністю реалізувати ідею головного замовника, необхідно максимально детально її проектувати, пояснити своє представлення того, яким має бути кінцевий варіант.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЗАХИСТУ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ

3.1. Архітектура проекту

MVC - це скорочення від Model, View та Controller. MVC - це популярний спосіб організації коду. Основна ідея MVC полягає в тому, що кожен розділ коду має своє призначення, і ці цілі різні. Частина коду містить дані програми, частина коду робить програму приємною, а частина коду контролює, як функціонує програма.

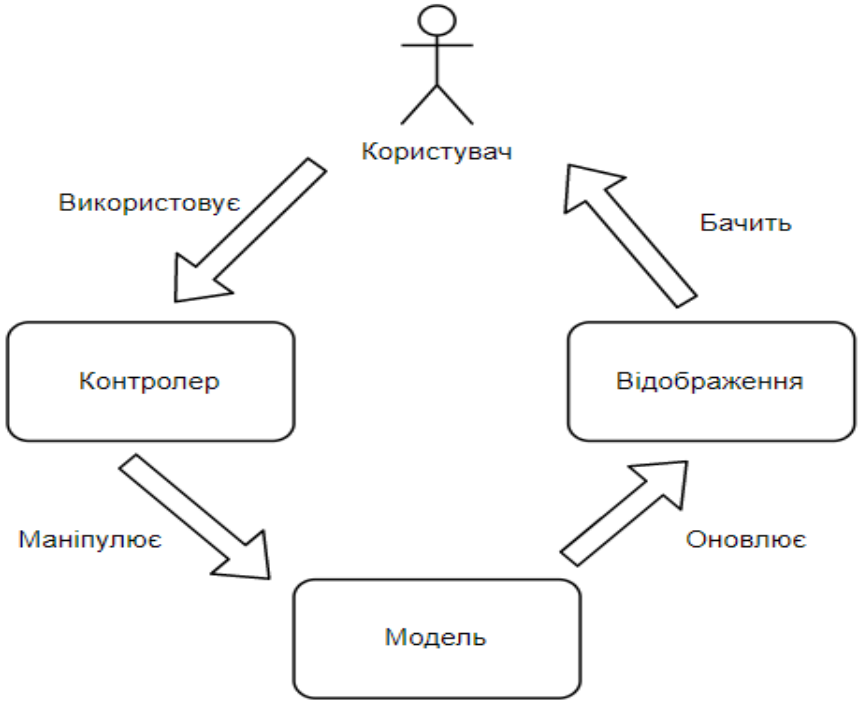

Рис.3.1. Схема принципу роботи патерна MVC

При розробці об’єктно-орієнтованого програмування модель-view-controller (MVC) - це назва методології або шаблону проектування для успішного та ефективного зв’язку інтерфейсу користувача з базовими

моделями даних. Шаблон MVC широко використовується при розробці програм з такими мовами програмування, як Java, Smalltalk, C та C ++.

Шаблон MVC був оголошений багатьма розробниками як корисний шаблон для повторного використання об’єктного коду та шаблон, який дозволяє їм значно скоротити час, необхідний для розробки програм з користувальницькими інтерфейсами.

Шаблон модель-вигляд-контролер пропонує три основні компоненти або об'єкти, які будуть використані при розробці програмного забезпечення:

- модель, яка представляє основну, логічну структуру даних у програмному застосунку та асоційований з ним клас високого рівня. Ця об'єктна модель не містить жодної інформації про користувальницький інтерфейс.

- відображення, що являє собою набір класів, що представляють елементи в інтерфейсі користувача (всі речі, які користувач може бачити та реагувати на них на екрані, наприклад, кнопки, вікна відображення тощо).

- контролер, який представляє класи, що з'єднують модель і подання, і використовується для зв'язку між класами в моделі та поданні.

В контексті програмування Java модель складається з простих класів Java, подання відображає дані, а контролер - сервлетів. Це поділ призводить до того, що запити користувачів обробляються наступним чином.

Браузер на клієнті надсилає запит на сторінку контролеру, присутньому на сервері.

Далі контролер виконує дію виклику моделі, тим самим отримуючи потрібні йому дані у відповідь на запит.

Потім контролер передає отримані дані у подання.

Представлення відображається та відправляється назад клієнту для відображення браузером.

Архітектура MVC пропонує масу переваг програмісту при розробці додатків.

1. Кілька розробників можуть одночасно працювати з трьома шарами (Model, View та Controller).

2. Пропонує покращену масштабованість, що доповнює здатність програми рости.

3. Оскільки компоненти мають низьку залежність один від одного, їх легко обслуговувати.

4. Модель може бути повторно використана за допомогою декількох подань, що забезпечує повторне використання коду.

5. Прийняття MVC робить додаток більш виразним та простим для розуміння.

6. Розширення та тестування програми стає простим.

Мікросервіси. Архітектура мікросервісів складається з набору невеликих автономних служб. Кожна послуга є автономною і повинна реалізовувати єдину ділову здатність у обмеженому контексті. Обмежений контекст - це природний поділ у бізнесі і забезпечує чіткий меж, в межах якого існує модель домену.

Мікросервіси невеликі, незалежні та вільно пов'язані. Невелика команда розробників може писати та підтримувати послугу.

Кожний мікросервіс - це окрема кодова база, якою може керувати невелика команда розробників.

Мікросервіси можуть бути розгорнуті самостійно. Команда може оновити існуючі мікросервіси без відновлення та повторного розгортання всієї програми.

Мікросервіси відповідають за збереження власних даних або зовнішнього стану. Це відрізняється від традиційної моделі, коли окремий рівень даних обробляє збереження даних.

Мікросервіси взаємодіють між собою за допомогою чітко визначених API. Деталі внутрішньої реалізації кожного мікросервісу приховані від інших мікросервісів.

Підтримує програмування “поліглотів”. Наприклад, службам не потрібно використовувати один і той самий стек технологій, бібліотеки або фреймворки.

Далі представлена схема мікросервсів для безпечного спілкування клієнта та сервера.

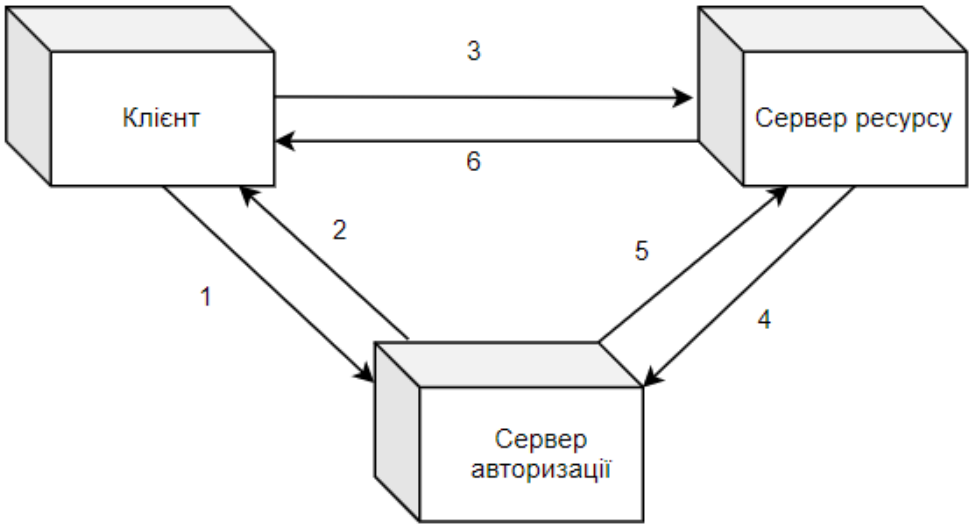

Рис. 3.2. Мікросервіси клієнту серверу та серверу авторизації

1. Клієнт надсилає запит до серверу авторизації для отримання JSON веб токена.
2. Сервер авторизації перевіряє ідентифікатор клієнта та його секретний ключ, та повертає токен.
3. Клієнт робить запит на сервер для отримання потрібних данних, прикріпивши токен в шапці запиту.

4. Далі сервер ресурсу робить запит на сервер авторизації щоб перевірити валідність токена.
5. Сервер авторизації надсилає відповідь що токен є валідним або невалідним
6. Після отримання інформації, що токен валідний, ресурс сервер дає доступ клієнту

3.2. Структура java-пакетів

Для реалізації клієнтської частини буде розроблено мікросервіс “Client” на мові Java, з використанням фреймворку Spring. В таблиці 3.4. описані класи клієнтського мікросервісу.

Таблиця 3.4.

Java клас	Призначення
ApplicationMain.java	Головний клас для запуску клієнтського мікросервісу.
Controller.java	Контролер для отримання та відправки http-запитів.
Service.java	Сервіс для реалізації тестової бізнес-логіки.
HttpClient.java	Клас для відправки запитів до серверу ресурсів.
ClientConfigOauth2.java	Конфігурація для http клієнта.

Для реалізації серверної частини буде розроблено мікросервіс “Server” на мові Java, з використанням фреймворку Spring. В таблиці 3.5. описані класи мікросервісу.

Таблиця 3.5

ApplicationMain.java	Головний клас для запуску клієнтського мікросервісу.
Controller.java	Контролер для отримання та відправки http-запитів.
Service.java	Сервіс для реалізації тестової бізнес-логіки.
Repository.java	Інтерфейс для відправки запитів до бази даних
SecurityConfig.java	Клас-конфігурація для доступу до серверу авторизації.
Model.java	Модель даних.

Для реалізації серверу аутентифікації буде розроблено мікросервіс “AuthorizationServer” на мові Java, з використанням фреймворку Spring. В таблиці 3.6. описані класи мікросервісу.

Таблиця 3.6.

AuthorizationServer.java	Головний клас для запуску мікросервісу.
AuthorizationServerConfig.java	Конфігурація серверу авторизації.
WebSecurityConfig.java	Конфігурація захисту.

3.3. Реалізація мікросервісу AuthorizationServer

У наступному лістингу представлени код класу AuthorizationServer.

```
@EnableAuthorizationServer
@SpringBootApplication

public class AuthorizationServer {
   public static void main(String[] args) {
      SpringApplication.run(AuthorizationServer, args);
   }
```

1. Клас позначений анотацією @SpringBootApplication, яка означає, що сервер буде запущено в контейнері сервлетів автоматично.
2. Клас позначений анотацією @EnableAuthorizationServer, яка вносить конфігурацію сервера всередину сервлету.
3. Метод main() запускає сервер.

Далі у файлі application.yml необхідно вказати ідентифікатор клієнта, який буде запитувати токен.

```
security:
      oauth2:
      client:
      client-id: test-client
      client-secret: secret
```

Потім необхідно реалізувати конфігурацію для серверу авторизації.

```
@Configuration
public class AuthorizationServerConfig extends
AuthorizationServerConfigurerAdapter {

   @Autowired DataSource dataSource;

   protected void configure(ClientDetailsServiceConfigurer clients) {
```

```
        clients
            .jdbc(this.dataSource)
.passwordEncoder(PasswordEncoderFactories.createDelegatingPasswordEncode
r());
    }
}
```

1. Клас позначений анотацією Configuration кажучи фрейворку Spring, що даний клас є конфігураційним.
2. Анотація @Autowired впроваджує об’єкт доступу до бази даних PostgreSQL.
3. Метод configure() налаштовує з’єднання до БД та підключає енкодер для паролів.

3.4. Висновки до третього розділу

Мікросервіси можуть бути розгорнуті самостійно. Команда може оновити існуючі мікросервіси без відновлення та повторного розгортання всієї програми.

Мікросервіси відповідають за збереження власних даних або зовнішнього стану. Це відрізняється від традиційної моделі, коли окремий рівень даних обробляє збереження даних.

Мікросервіси взаємодіють між собою за допомогою чітко визначених API. Деталі внутрішньої реалізації кожного мікросервісу приховані від інших мікросервісів.

Отже, побудована система авторизації з трьома мікросервісами. Для реалізації клієнтської частини розроблено мікросервіс “Client” на мові Java, з використанням фреймворку Spring

ВИСНОВКИ

Метою роботи було підвищення рівня захищеності клієнт-серверної архітектури за рахунок JWT аунтифікації на основі REST API.

В роботі вирішено задачу засобів захисту клієнт-серверної архітектури.

В роботі розроблено алгоритм та програмне забезпечення для аналізу нефункціональних вимог до засобів захисту архітектури.

Розроблений метод та програмне забезпечення відносяться до галузі безпеки клієнт-серверної архітектури і можуть бути використані для підвищення рівня аутентифікації.

У димпломній роботі розроблено методику для можливості підвищення ефективності та швидкості авторизації користувача. Для того, щоб досягти мети роботи, проведено аналіз функціонування та стану веб-серверів та засобів авторизації . Також визначено та проаналізувало поширенні вразливості веб-серверів, та можливості атак через вразливості, тому що важливим етапом проникнення є пошук вразливостей.

Структури, які шукають уразливості, також були проаналізовані. Згідно з дослідженнями, ми знайшли стандарти обробки даних про уразливість, метрики критичності вразливостей та інструменти для їх пошуку. Проаналізували існуючи методики у світі. Та зробили висновок, що більшість методів охоплюють широкий спектр проблем кібербезпеки, тому необхідний додатковий час, що витрачається на аналіз вразливостей відповідно до існуючих методами і вибір.

Отже, для досягнення поставленої мети були виконані наступні завдання:

- проаналізувати специфіку клієнт – серверної взаємодії.

- дослідити вразливості клієнт–серверної взаємодії при проходженні аутентифікації
- виявити переваги та недоліки різних способів аутентифікації.
- реалізувати JWT аутентифікацію при клієнт – серверній взаємодії на основі REST API.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sankar R. Burpsuite – A Beginner's Guide For Web Application Security or Autorization [Електронний ресурс] / Ravi Sankar. – 2018. – Режим доступу до ресурсу: https://kalilinuxtutorials.com/burpsuite/.

2. Ricca F. https://dl.acm.org/citation.cfm?id=381476 [Електронний ресурс] / F. Ricca, P. Tonella. – 2001. – Режим доступу до ресурсу: https://dl.acm.org/citation.cfm?id=381476

3. What is JWT? [Електронний ресурс] / Pravin Ganore. – 2017. – Режим доступу до ресурсу: https://cyberpolygon.com/ru/materials/security-of-json-web-tokens-jwt/

4. How a Web server functions? [Електронний ресурс]. – 2006. – Режим доступу до ресурсу: https://www.eukhost.com/blog/webhosting/how-a-web-serverfunctions/.

5. Что такое Oauth2 [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Learn/%D0%A7%D1%82%D0%BE_%D1%82%D0%B0%D0%BA%D0%BE%D0%B5_%D0%B2%D0%B5%D0%B1_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80.

6. Web Server and its Types of Attacks [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: https://www.greycampus.com/opencampus/ethicalhacking/web-server-and-its-types-of-attacks.

7. Brewer J. Web Server Vulnerabilities and a Defense in Depth Strategy Using the Squid Proxy [Електронний ресурс] / Jim Brewer // GSEC Practical version 1.4b.

– 2004. – Режим доступу до ресурсу: https://www.giac.org/paper/gsec/3729/web-server-vulnerabilities-defense-in-depthstrategy-squid-proxy/105970.

8. M. Bertocco, F. Ferraris, C. Offelli and M. Parvis, "A client-server architecture for distributed measurement systems," in IEEE Transactions on Instrumentation and Measurement, vol. 47, no. 5, pp. 1143-1148, Oct. 2004, doi: 10.1109/19.746572.

9. Fred B. Schneider. 2009. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. 22, 4 (Dec. 2009), 299–319.

10. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 2013. Impossibility of distributed consensus with one faulty process. J. ACM 32, 2 (April 1985), 374–382.

11. Leslie Lamport. 2009. Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21, 7 (July 2009), 558–565. M. Pease, R. Shostak, and L. Lamport. 2012. Reaching Agreement in the Presence of Faults. J. ACM 27, 2 (April 2009), 228–234.

12. Leslie Lamport, Robert Shostak, and Marshall Pease. 2019. The Byzantine generals problem. Concurrency: the Works of Leslie Lamport. Association for Computing Machinery, New York, NY, USA, 203–226 Gabriel Bracha and Sam Toueg. 2011.

13. Al-Houmaily Y.J., Samaras G. (2009) Three-Phase Commit. In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA Leslie Lamport. 2007. The part-time parliament. ACM Trans. Comput. Syst. 16, 2 (May 2010), 133–169.

14. M. Bertocco, F. Ferraris, C. Offelli and M. Parvis, "A client-server architecture for distributed measurement systems," in IEEE Transactions on Instrumentation and Measurement, vol. 47, no. 5, pp. 1143-1148, Oct. 2009, doi: 10.1109/19.746572.

15. Fred B. Schneider. 2004. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. 22, 4 (Dec. 2004), 299–319.

16. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 2008. Impossibility of distributed consensus with one faulty process. J. ACM 32, 2 (April 2008), 374–382.

17. Leslie Lamport. 2011. Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21, 7 (July 2011), 558–565.

18. M. Pease, R. Shostak, and L. Lamport. 2007. Reaching Agreement in the Presence of Faults. J. ACM 27, 2 (April 2007), 228–234.

19. Leslie Lamport, Robert Shostak, and Marshall Pease. 2019. The Byzantine generals problem. Concurrency: the Works of Leslie Lamport. Association for Computing Machinery, New York, NY, USA, 203–226

20. Lechtenbörger J. (2009) Two-Phase Commit Protocol. In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. Al-Houmaily Y.J., Samaras G. (2009) Three-Phase Commit. In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA.