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

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ С.В. Казмірчук
«_____»_____ 2021 р.

На правах рукопису
УДК 004.056

ДИПЛОМНА РОБОТА

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «БАКАЛАВР»

Тема: Метод автентифікації користувачів інформаційних систем

Виконавець: М. С. Остапенко
Керівник: старший викладач О. Л.Яковенко
Нормоконтролер: старший викладач О. Л.Яковенко

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

«__» _____ 2021 р.

ЗАВДАННЯ

на виконання дипломної роботи

здобувача вищої освіти Остапенка Максима Сергійовича

1. Тема: *Метод автентифікації користувачів інформаційних систем* затверджена наказом ректора від «26» квітня 2021 р. № 652/ст.
2. Термін виконання: з 10.05.2021 р. по 20.06.2021 р.
3. Вихідні дані: дослідити методи автентифікації користувачів інформаційних систем; виявити їх переваги і недоліки алгоритмів автентифікації; реалізувати програмний модуль для автентифікації користувачів в інформаційних системах.
4. Зміст пояснювальної записки: аналіз існуючих сучасних алгоритмів автентифікації; розробка програмного забезпечення для автентифікації в системі, перевірка отриманих результатів.

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	23.04.2021	*Виконано*
4.	Збір інформації	26.04.2021	*Виконано*
5.	Дослідження сучасних методів автентифікації користувачів в інформаційних системах	01.05.2021	*Виконано*
6.	Огляд архітектури додатку та інструментів розробки	08.05.2021	*Виконано*
7.	Розробка веб-додатку для авторизації, реєстрації та автентифікації користувачів	16.05.2021	*Виконано*
8.	Написання висновків	26.05.2021	*Виконано*
9.	Оформлення презентації	02.06.2021	*Виконано*
10.	Перевірка на плагіат	03.06.2021	*Виконано*
11.	Оформлення і друк пояснювальної записки	05.06.2021	*Виконано*
12.	Отримання рецензій від рецензента	10.06.2021	*Виконано*

Здобувач вищої освіти М. Остапенко

(підпис, дата)

Керівник дипломної роботи О. Яковенко

(підпис, дата)

РЕФЕРАТ

Дипломна робота складається зі вступу, двох розділів, загальних висновків, списку використаних джерел, додатків і має 58 сторінок основного тексту, 31 рисунку, 6 сторінок додатків. Список використаних джерел містить 17 найменувань і займає 2 сторінки. Загальний обсяг роботи 62 сторінки.

Метою роботи є розробка програмного модуля автентифікації користувачів за допомогою JWT веб-токену.

В роботі вирішено задачу побудови веб-додатку з підтримкою автентифікації користувачів в системі.

В роботі розроблено програмне забезпечення для автентифікації, авторизації та реєстрації користувачів з використанням JSON Web-Token.

Розроблений метод та програмне забезпечення відносяться до галузі інформаційної безпеки і можуть бути використані для підвищення рівня цілісності та конфіденційності користувачів.

Ключові слова: автентифікація, авторизація, токен, JSON Web-Token, сервер, клієнт, метод автентифікації

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ІС	–	Інформаційна система
ЦС	–	Центр сертифікації
HTTP	–	HyperText Transfer Protocol
HTTPS	–	HyperText Transfer Protocol Secure
WWW	–	World Wide Web
ACL	–	Access Control List
URL	–	Uniform Resource Locator
USB	–	Universal Serial Bus
SSL	–	Secure Sockets Layer
TLS	–	Transport Layer Security
RSA	–	Rivest–Shamir–Adleman
API	–	Application Programming Interface
HMAC	–	keyed hash message authentication code
SSO	–	Single sign-on
IP	–	identity provider
SP	–	service provider
SWT	–	Simple web token
HTML	–	Hypertext Markup Language
JWT	–	JSON WebToken
SAML	–	Security Assertion Markup Language
SOAP	–	Simple Object Access Protocol
XML	–	extensible markup language
STS	–	Secure Token Service
OAuth	–	Open Authorization

ВСТУП

Актуальність теми. У наш час загальної комп'ютеризації та інформатизації особливу важливість і значимість набувають завдання захисту інформації та розмежування доступу до управління інформаційних систем і до їх ресурсів, захисту від несанкціонованого доступу. Ці завдання вирішуються системами контролю доступу та захисту інформації, які виконують автентифікацію на основі певної унікальної інформації, що характеризує конкретного користувача.

Мета роботи. Розробка програмного модуля автентифікації користувачів за допомогою JWT веб-токену.

Для реалізації мети потрібно:

- дослідити методи автентифікації користувачів інформаційних систем;
- виявити їх переваги і недоліки алгоритмів автентифікації;
- реалізувати програмний модуль для автентифікації користувачів в інформаційних системах.

Об'єкт дослідження. Процес автентифікації користувачів інформаційних систем.

Предмет дослідження. Метод забезпечення автентифікації користувачів інформаційних систем.

Галузь застосування. Метод може бути використаний для захисту та підвищення рівню конфіденційності даних користувачів.

Практична значимість. розроблений програмний продукт дозволяє вирішити основні задачі для будь якої системи, яка взаємодіє, зберігає та використовує дані користувачів, а саме авторизація, реєстрація та автентифікація користувачів. Цей продукт є універсальний для більшості веб-додатків, тож його можна використовувати для створення майбутніх програм для користувачів з особистими кабінетами.

РОЗДІЛ 1. СУЧАСНІ МЕТОДИ АВТЕНТИФІКАЦІЇ

Автентифікація на даний момент вважається одним із основних засобів захисту ІС, найбільш важливий елемент системи безпеки.

Ідентифікація та автентифікація можуть вважатися основою програмного та апаратного забезпечення, оскільки інші служби призначені для обслуговування іменованих об’єктів. Ідентифікація та автентифікація - це перша лінія захисту.

Ідентифікація - процедура розпізнавання користувача в системі, як правило, за допомогою наперед визначеного імені (ідентифікатора) або іншої апріорної інформації про нього, яка сприймається системою.[1]

Вона дозволяє певного користувачу або іншому апаратно-програмного компоненту назвати себе (повідомити своє ім'я).

Автентифікація - процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора.[2]

Проводячи автентифікацію, інша сторона переконується, що суб’єктом є насправді той, ким він або вона себе видає.

Автентифікація буває:

- Одностороння (клієнту потрібно доводити свою справжність серверу)
- Двостороння (взаємна). Протоколи взаємної автентифікації дають можливість обом, що беруть участь в обміні даними сторонам, переконатися в достовірності один одного і обмінятися сеансовими ключами.

Сучасні ІС - сукупність територіально розосереджених компонентів. Більшості користувачів потрібен доступ до багатьох сервісів. Бажано, щоб лише авторизовані користувачі мали до нього доступ і щоб сервери мали можливість автентифікувати запити до сервісів.

В мережевому середовищі, коли сторони територіально рознесені, в сервісу автентифікації є два основних аспекти:

- що служить автентифікатором (посвідченням)
- як організований та безпечний обмін даними ідентифікації/автентифікації (тобто протокол автентифікації).

Для підтвердження своєї автентичності, суб’єкт має пред'явити хоча б одну з нижче вказаних сутностей:

- те, що він знає (тобто криптографічний ключ, пароль або персональний ідентифікаційний номер та інші);
- те, чим він володіє (може бути особиста картка або інший аналогічний пристрій);
- те, що є частина його самого (свої біометричні характеристики, такі як голос, відбитки пальців та інші).

Процес автентифікації повинен бути простим для користувачів і одночасно надійним для забезпечення безпеки системи.

На сьогодні автентифікація дуже важлива, тому що вона дозволяє організаціям забезпечувати безпеку своїх мереж, дозволяючи лише автентифікованим користувачам (або процесам) доступ до своїх захищених ресурсів, які можуть включати комп'ютерні системи, мережі, бази даних, веб-сайти і інші мережеві додатки або служби.

Щоб дізнатися, чи потрібно автентифікованому об’єкту давати доступ до захищеного ресурсу або системи, після успішного процесу автентифікації процес або користувач потрібні провести процедуру авторизації. Тобто, якщо об’єкт автентифікований, але не пройшов процес авторизації, в цьому випадку він не зможе отримати доступ до системи.

Терміни автентифікація і авторизація часто використовуються як синоніми; хоча вони часто реалізовуються разом, ці два поняття різні. У той час як автентифікація - це процес перевірки особистості зареєстрованого користувача перед наданням доступу до захищеного ресурсу, авторизація - це процес перевірки того, що автентифікованим користувачеві надано дозвіл на доступ до ресурсів, які він намагається отримати. Процес, за допомогою якого

доступ до цих ресурсів обмежується певним числом користувачів, називається контролем доступу. Процес автентифікації завжди передує процесу авторизації.

Технології автентифікації забезпечують контроль доступу для систем, перевіряючи, чи збігаються облікові дані користувача з обліковими даними в базі даних авторизованих користувачів або на сервері автентифікації даних.

Ідентифікація користувача відбувається зазвичай за допомогою ідентифікатора користувача, а коли користувач надає свої облікові дані, які збігаються з його ідентифікатором, виконується автентифікація. Більшість користувачів найкраще знайомі з використанням пароля, який як частина інформації, яка повинна бути відома тільки користувачеві, називається фактором перевірки автентичності знань.

На теперішній час є багато способів авторизувати користувачів у системах, але спочатку потрібно чітко розуміти термінологію.

Ідентифікація — це заява про те, ким ви є. Залежно від ситуації, це може бути ім'я, адреса електронної пошти, номер облікового запису, тощо.

Автентифікація — надання доказів, що ви насправді є той, ким ідентифікувалися (від слова "authentic" — істинний, справжній)[3].

Авторизація — перевірка, що вам дозволено доступ до запитуваного ресурсу [4].

Наприклад, якщо ви намагаєтесь потрапити до закритого клубу, Вас попередньо ідентифікують (просять вказати Ваше ім’я та прізвище), засвідчують справжність (просять показати паспорт та порівняють фото) та авторизують (перевірка, чи Ваше ім’я знаходиться у списку очікуваних), після чого Вас зможуть пустити до клубу.[4]

Подібним чином ці терміни використовуються в комп'ютерних системах, де ідентифікація традиційно означає, що ви отримуєте свій профіль (identity) за допомогою username чи email; під автентифікацією - перевірка, чи знаєте Ви пароль цього облікового запису, а під авторизацією - перевірка Вашої ролі в

системі та рішення про надання доступу до запитуваної сторінки чи ресурсу. Роздивимося різні підходи за допомогою яких автентифікують користувачів:

- автентифікація за допомогою паролю;
- автентифікація за допомогою сертифікатів;
- автентифікація за допомогою ключів-доступу;
- автентифікація за допомогою токенів.

1.1 Автентифікація за допомогою паролю

Цей метод заснований на тому, що користувач повинен надати ім’я користувача та пароль для успішної ідентифікації та автентифікації системи. Пара ім’я користувача/пароль встановлюється користувачем під час входу в систему, і ім’я користувача може бути електронною адресою користувача. Для веб-додатків існує кілька стандартних протоколів для автентифікації паролем, а саме: HTTP, Форми, URL-запит, тіло запиту, HTTP header.

Описана в стандартах HTTP 1.0/1.1, HTTP authentification використовується протягом багатьох років і за цей час стала корпоративним стандартом. Розглянемо схему взаємодії з сайтом:

1. По-перше, несанкціонований клієнт запитує захищене джерело на сервері, що відповідає статусом HTTP "401 Unauthorized", також додається схему та параметри автентифікації до заголовка "WWW-Authenticate";
2. Після отримання цієї відповіді браузер автоматично відображає діалогове вікно для введення ім’я користувача та паролю, куди користувач вводить інформацію про свій обліковий запис;
3. Під час подальших викликів на сервер цього захищеного джерела браузер додає "Авторизацію" до заголовка запиту, який містить усі дані користувача, необхідні для автентифікації сервера;

4. Сервер повинен автентифікувати користувачів на основі даних, отриманих з цього заголовка, і вирішує надавати користувачеві доступ до захищеного ресурсу в залежності від ролі, ACL чи інших даних.[5]

Усі браузери та веб-сервери добре підтримують цей стандартизований процес. Схеми автентифікації, які існують на даний момент, мають різні рівні захисту:

- Базова - найпростіша схема, при якій ім'я користувача та пароль користувача переносяться в заголовок Авторизації в незашифрованому вигляді (закодований base64). Однак це відносно безпечно, якщо використовувати протокол HTTPS (рис. 1.1);
- Дайджест - схема виклику відповіді, при якій сервер надсилає унікальне значення nonce, а браузер передає пароль користувача MD5 у вигляді MD5 хешу, розрахований за допомогою nonce. Це є більш безпечнішою альтернативою базовій схемі з незахищеними зв’язками, але схильні до атак "man-in-the-middle" (та з заміною на базову схему). Крім того, при використанні цієї схеми не має можливості використовувати сучасні функції хешування, щоб зберігати паролі усіх користувачів на стороні серверу;
- NTLM (також відомий як Windows authentication) також базується на підході challenge-response, при якому в чистому вигляді пароль не може бути передано. На даний момент ця схема підтримується більшістю веб-браузерів та веб-серверів але вона не є стандартом HTTP. Ця схема вразлива до атак “path-the-hash” але в основному використовується для автентифікації користувачів Windows Active Directory у веб-програмах.
- Negotiate - це ще одна сімейна схема автентифікації для Windows, яка дозволяє клієнту вибирати між автентифікацією NTLM та Kerberos. Kerberos - це більш безпечний протокол єдиного входу. Однак це

може працювати, лише якщо клієнт і сервер знаходяться в зоні інтрамережі та є частиною домену Windows.

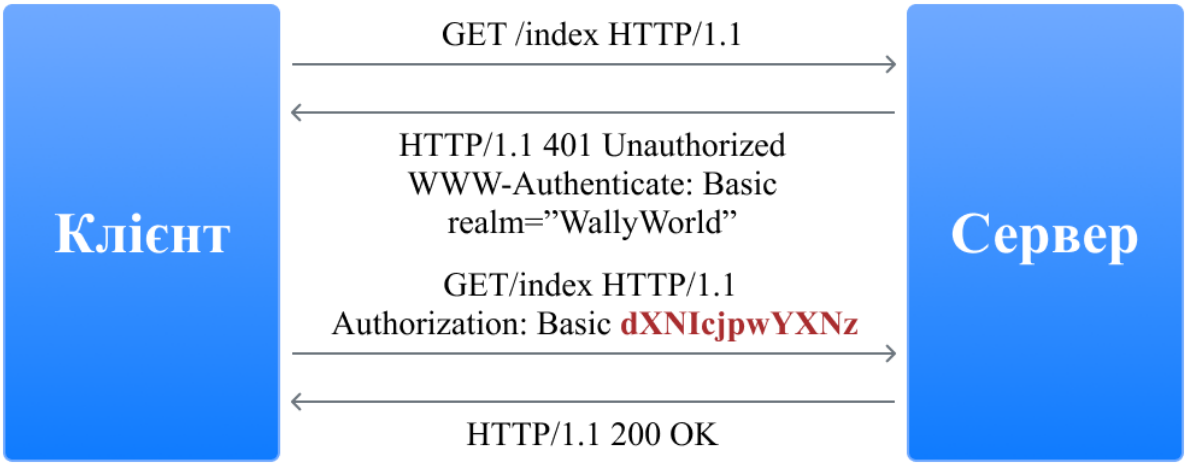


Рис. 1.1. Схема Basic автентифікації

Варто зазначити, що під час автентифікації HTTP користувач не має звичайної опції виходу з веб-програми, тільки якщо закрити всі вікна у своєму браузері.

Протокол Authentication Forms не має конкретного стандарту, тому всі його реалізації специфічні для певних систем і, точніше, для модулів автентифікації певних платформ розробки. Він працює за таким принципом: веб-програма включає форму HTML, в якій користувач повинен ввести свійusername/password і відправити ці дані на сервер через HTTPPOST запит для успішної автентифікації. У разі успіху веб-програма створює токен сеансу, який зазвичай можна знайти у файлах cookie браузера.

Для подальших онлайн-запитів токен сеансу автоматично завантажується на сервер і дозволяє програмі отримувати інформацію про поточного користувача, який схвалює запит. (рис.1.2)

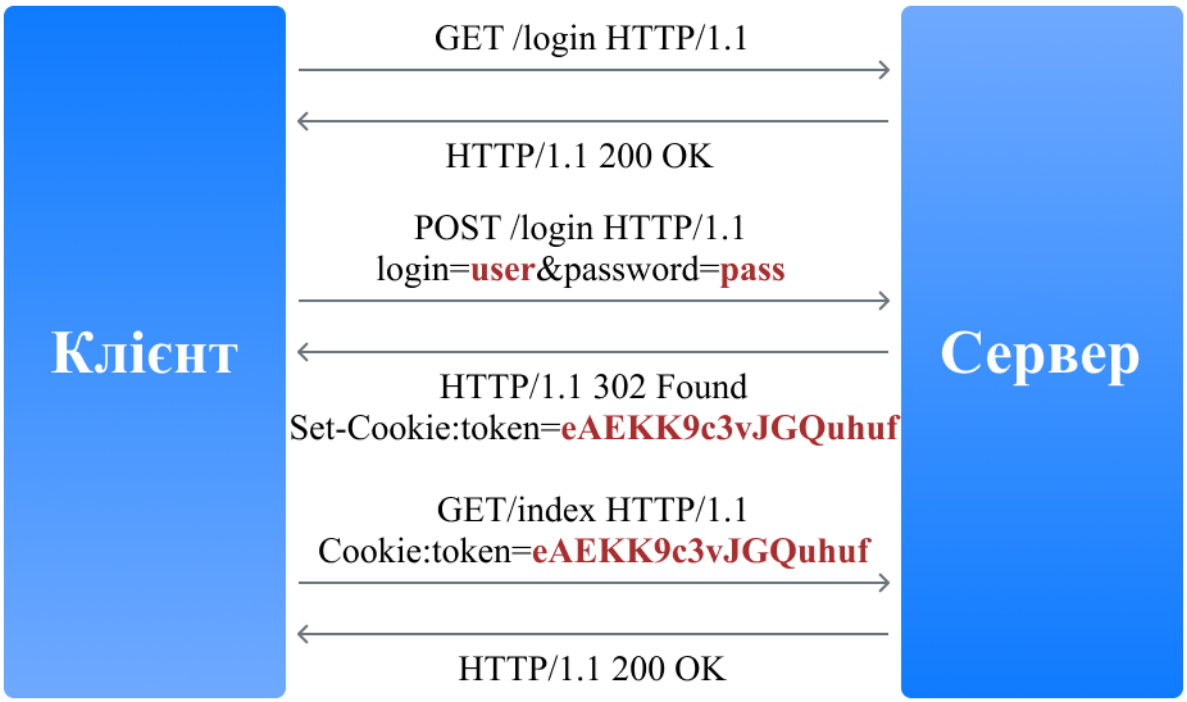


Рис. 1.2. Схема Forms автентифікації

Додаток може створити токен сеансу двома способами:

1. Як ідентифікатор автентифікації сеансу користувача, що зберігається в пам'яті сервера або в базі даних. Сеанс повинен містити всю необхідну інформацію про користувача, щоб мати змогу затвердити його запити;
2. Як зашифрований та/або підписаний об’єкт, що містить дані користувача, а також термін. Використання даного підходу дозволяє реалізувати архітектуру сервера без стану, але вимагає механізму оновлення токена сесії після закінчення терміну дії.

Слід розуміти, що перехоплення токенів сеансу часто забезпечує той самий рівень доступу, що і знання імені користувача/пароля. Отже, усі комунікації клієнт-сервер у разі автентифікації форми повинні виконуватися лише на захищеному з'єднанні HTTPS.

Інші протоколи автентифікації паролів. Описані вище протоколи успішно використовуються для автентифікації користувачів на веб-сайтах. Однак при розробці додатків клієнт-сервер із використанням веб-служб (таких як iOS або Android), крім аутентифікації HTTP, часто використовуються нестандартні

протоколи, в яких дані автентифікації передаються в інші частини запиту. Є лише кілька місць, де ви можете вказати ім’я користувача та пароль у запитах HTTP:

- URL-запит - вважається небезпечним варіантом, тоді як браузери, проксі-сервери та веб-сервери можуть запам'ятовувати рядки URL-адрес;
- Тіло запиту - безпечний варіант, але використовується лише для запитів, що містять тіло повідомлення (наприклад, POST, PUT, PATCH);
- HTTP-оптимальний варіант заголовка, тому ми можемо також використовувати стандартну авторизацію заголовка (наприклад, у базовій схемі) та інші довільні заголовки.

Простота цього методу приховує багато недоліків, а тому цей метод не надто надійний. Паролі часто можна підібрати, і користувачі зазвичай використовують прості і однакові паролі в різних системах або записують їх на папері. Якщо зловмисникові вдалося дізнатись пароль, користувач часто не знає про нього. Крім того, розробники додатків можуть допустити низку концептуальних помилок, які полегшують проникнення в облікові записи. Далі наведено перелік найпоширеніших вразливих місць автентифікації паролів:

- Веб-програма не забороняє користувачам використовувати вразливі паролі;
- Веб-програма не захищена від можливості пошуку пароля (атака brute-force);
- Веб-програма генерує та розповсюджує паролі для користувачів, але користувачу не пропонується змінити свій пароля після першої авторизації (тобто поточний пароль десь збережений);
- Веб-програма дозволяє завантажувати паролі через незахищене з'єднання HTTP або в наборі URL-адрес;

- Веб-програма не використовує функції безпечного стиснення для зберігання паролів користувачів;
- Веб-програма не дозволяє користувачам змінювати пароль або не інформує користувачів про зміну пароля;
- Веб-програма використовує функціонал відновлення пароля, який, через певні вразливості, дозволяє отримати несанкціонований доступ до інших облікових записів.
- Веб-програма не запитує повторної автентифікації користувачів під час проведення важливих процесів(наприклад, зміна пароля, адреси доставки товарів тощо);
- Веб-програма створює токени сеансів, щоб їх можна було вибрати або надати іншим користувачам;
- Веб-програма дозволяє передавати токени сеансів через незахищене з'єднання HTTP або в наборі URL-адрес;
- Веб-програма вразлива до атак фіксації сеансу (не замінює токен сеансу, коли анонімний сеанс користувача перевіряється);
- Веб-програма не встановлює прапори HttpOnly та Secure для файлів cookie браузера, що містять токени сеансу;
- Веб-програма не знищує сеанс користувача після короткого періоду бездіяльності або не дозволяє вийти з довіреного сеансу.

1.2 Автентифікація за допомогою сертифікатів

Сертифікат - це набір атрибутів, що ідентифікують власника, підписаний центром сертифікації (ЦС). Центр сертифікації виступає посередником для забезпечення справжності сертифікатів. Сертифікат також криптографічно пов’язаний із закритим ключем, що зберігається власником сертифіката, і дозволяє однозначно підтвердити право власності на сертифікат. На стороні

клієнта ви можете зберегти сертифікат разом із закритим ключем до операційної системи, браузера, файлу або окремого фізичного пристрою (смарт-картка, USB). Закритий ключ, як правило, додатково захищений паролем або PIN-кодом. Веб-програми традиційно використовують сертифікати X.509. Автентифікація сертифіката X.509 відбувається під час підключення до сервера та є частиною протоколу SSL/TLS. Браузер обре підтримують цей механізм та дозволяють користувачеві вибрати та використовувати сертифікат, якщо сайт дозволяє такий спосіб автентифікації.(рис. 1.3)

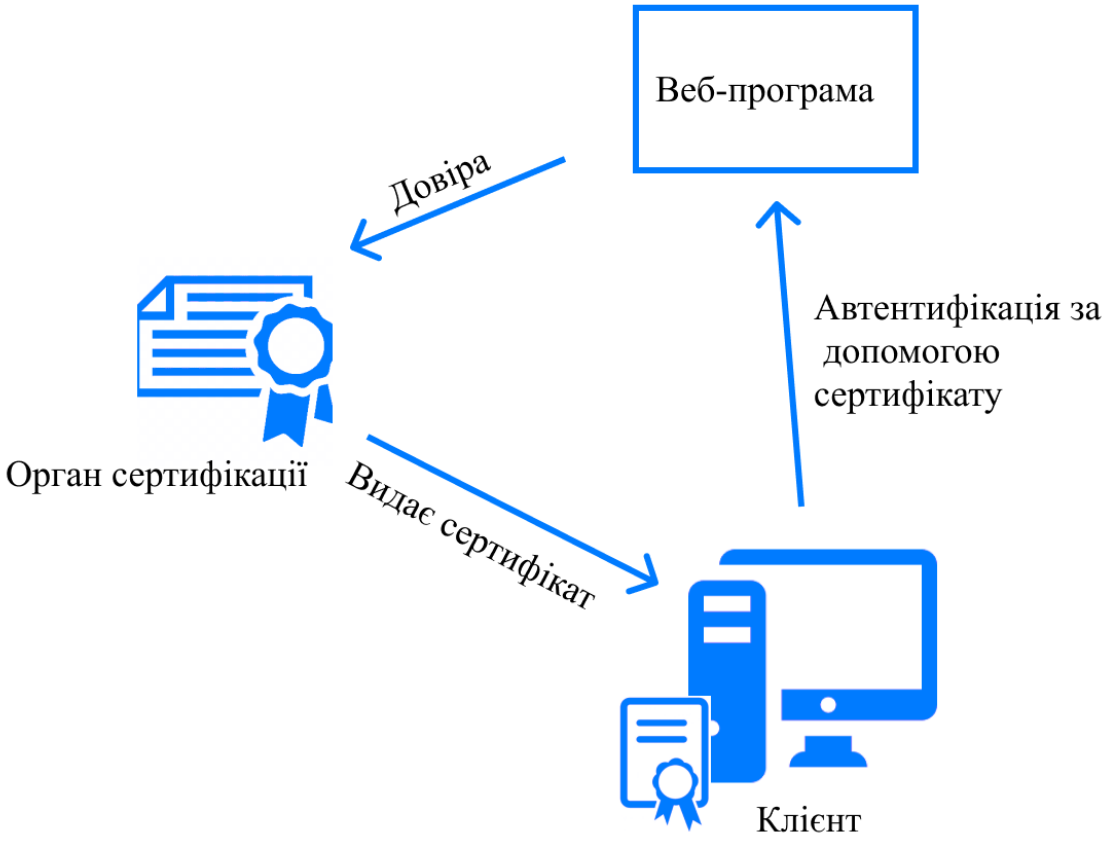


Рис. 1.3. Схема автентифікації сертифіката

Під час автентифікації сервер перевіряє сертифікат на основі таких правил:

– Сертифікат повинен бути підписаний надійним сертифікатом (верифікація ланцюга сертифікатів);

- Сертифікат повинен бути дійсним на теперішню дату (перевірка);
- Сертифікат не повинен бути анульований відповідним Органом сертифікації (перевірити списки виключень).

Після успішної автентифікації веб-програма може дозволити запит на основі даних сертифіката, таких як subject (ім’я власника), issuer (видавець), serialnumber (серійний номер сертифіката) або thumbprint (відбиток відкритого ключа сертифіката).

Використання сертифікатів автентифікації - набагато безпечніший спосіб, ніж автентифікація паролем. Це досягається створенням цифрового підпису в процесі автентифікації, про наявність якого свідчить факт використання приватного ключа в конкретній ситуації (заперечення відмови). Однак через труднощі в розповсюдженні та підтримці сертифікатів цей спосіб автентифікації недоступний для широкої громадськості.

Одноразова автентифікація за допомогою пароля зазвичай використовується на додаток до автентифікації за допомогою пароля для виконання двофакторної автентифікації (2FA). У цій концепції користувач повинен вказати два типи даних для входу: те, що він знає (наприклад, пароль), і те, що їм належить (наприклад, генератор одноразових паролів). Наявність таких двох факторів може значно підвищити рівень захисту, який може знадобитися для деяких типів веб-додатків.

Подальша автентифікація користувача під час виконання важливих дій, таких як переказ грошей, зміна налаштувань тощо, являє собою інший популярний сценарій використання одноразових паролів. Існує кілька джерел для створення одноразових паролів. Найпопулярніші з них:

- Токени (апаратні чи програмні), які мають можливість генерувати одноразові паролі на базі введеного секретного ключа та поточного часу. Секретні ключі користувачів, які є фактором власності, також зберігаються на сервері, що дозволяє перевірити введені унікальні

паролі. Приклад реалізації чіп-апаратного забезпечення - RSA SecurID; програмне забезпечення - додаток Google Authenticator;

- Коди генеруються випадковим чином і надсилаються користувачеві за допомогою SMS або іншого каналу зв'язку. У цьому випадку фактором власності є телефон користувача (точніше, SIM-карта, прив’язана до певного номера);
- Друк або scratchcart зі списком раніше створених одноразових паролів. Для кожного нового логіна потрібно вводити новий одноразовий пароль із певним номером.

У веб-програмах цей механізм автентифікації часто реалізується з використанням розширення автентифікації форми: після початкової автентифікації за паролем створюється сеанс користувача, але в межах цього сеансу користувач не має доступу до програми, поки не виконає додаткову автентифікацію одноразового пароля.

1.3 Автентифікація за допомогою ключів доступу

Цей метод найчастіше використовується для автентифікації пристроїв, служб чи інших програм під час доступу до Інтернет-послуг. Тут ключі доступу (accesskey, APIkey) використовуються як секрет - довгі унікальні рядки, які містять будь-який набір символів і по суті замінюють комбінацію ім’я користувача / пароль.

У більшості випадків сервер генерує ключі доступу на прохання користувачів, які потім зберігають ці ключі в клієнтських програмах. Створюючи ключ, можна також обмежити термін дії та рівень доступу, який отримає клієнт при автентифікації за допомогою цього ключа.

Хорошим прикладом використання автентифікації ключа є хмара Amazon Web Services. Скажімо, у користувача є веб-програма, яка дозволяє завантажувати та переглядати фотографії, і хоче використовувати службу

Amazon S3 для зберігання файлів. У цьому випадку користувач може створити ключ з обмеженим доступом до хмари через консоль AWS: просто прочитати / записати його файли в Amazon S3. Цей ключ в кінцевому підсумку може бути використаний для автентифікації хмарного веб-додатка AWS. (рис 1.4) [6].

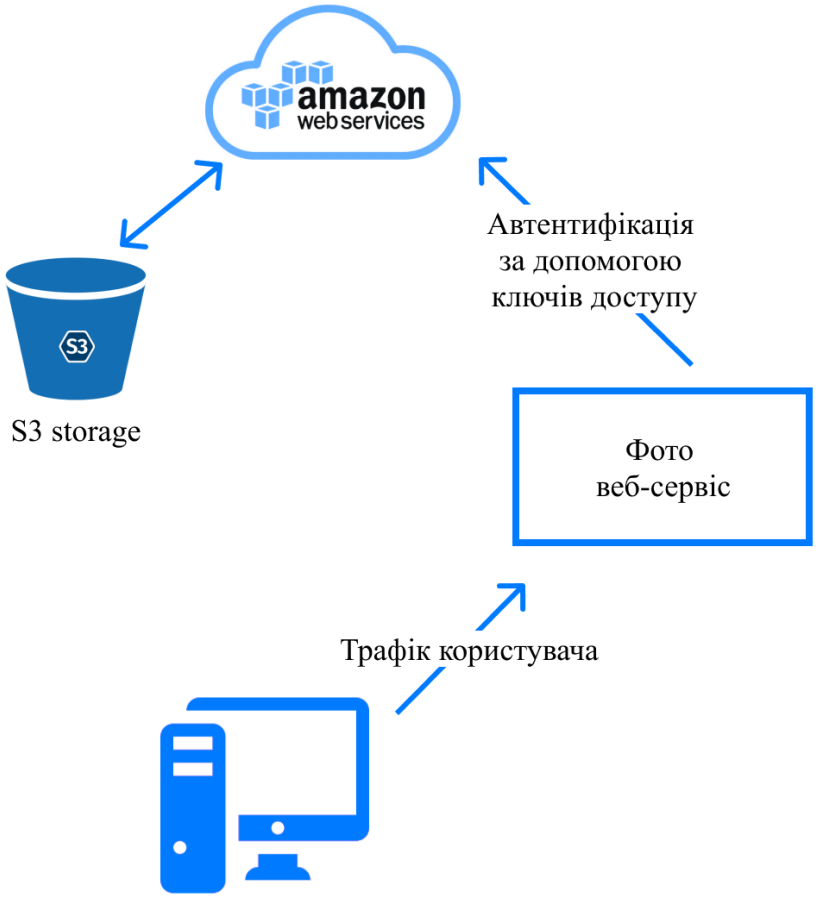


Рис. 1.4. Схема автентифікації ключів

Використовуючи ключі, ви уникаєте передачі пароля користувача стороннім програмам (у наведеному вище прикладі користувач не зберігав свій пароль у веб-програмі, а ключ доступу). Ключі мають набагато вищу ентропію, ніж паролі, що робить їх майже неможливими для пошуку. Крім того, якщо ключ було розкрито, це не компрометує основний обліковий запис користувача - просто відкликайте цей ключ і створіть новий.

З технічної точки зору, не існує єдиного протоколу: ключі можуть передаватися в різних частинах запиту HTTP: URL-query, bodyrequest або HTTP header. Як і для автентифікації за допомогою пароля, найкращим варіантом є використання HTTP header. У деяких випадках для передачі токена в заголовок використовується схема HTTP-носія (Authorization: Bearer [token]). Щоб уникнути перехоплення ключа, підключення до сервера повинно бути захищене за допомогою SSL/TLS. (рис. 1.5)

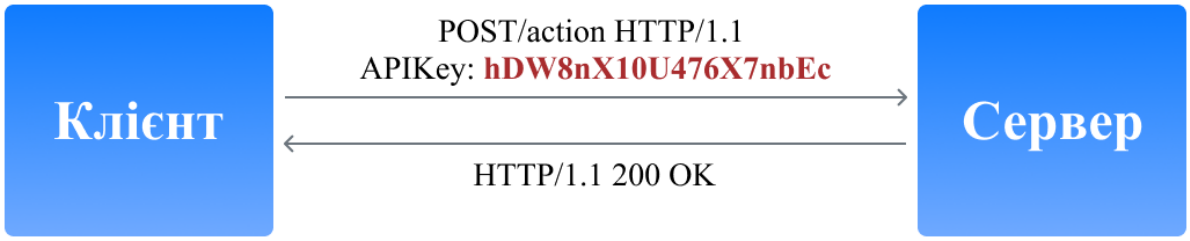


Рис. 1.5. Схема автентифікації ключа доступу, надісланого в HTTP header

Крім того, існують більш складні схеми автентифікації ключів для небезпечних з'єднань. У цьому випадку ключ, як правило, складається з двох частин: публічної та приватної. Публічна частина використовується для ідентифікації клієнта, а приватна дозволяє створювати так званий електронний підпис, який використовується для декодування запиту, надісланого на сервер. Наприклад, за аналогією зі схемою автентифікації витягування, сервер може надіслати клієнту унікальне значення nonce або timestamp, а клієнт може повернути хеш або HMAC цього значення, розрахованого за допомогою секретної частини ключа.

1.4 Автентифікація за допомогою токенів

Найчастіше використовується при виготовленні систем розподіленого єдиного входу (SSO), коли одна програма (service provider або relying party) передає функцію автентифікації користувача іншій програмі (identity provider або

authentication service). Типовим прикладом цього методу є вхід до програми через обліковий запис у соціальних мережах. Тут соціальні мережі надають послуги автентифікації, і програма довіряє функцію автентифікації користувачів соціальних мереж.

Реалізація цього методу полягає в тому, що постачальник identity provider (IP) надає надійну інформацію про користувача у вигляді токена, а програма service provider (SP) використовує його для ідентифікації, автентифікації та авторизації користувача (рис.1.6) [7].

Рис. 1.6. Схема автентифікації «активного» клієнта з токеном, надісланим за схемою Bearer

Загалом, весь процес такий:

- Клієнт автентифікується за допомогою паролю, ключа доступу, сертифікату або Kerberos у постачальника ідентифікаційних послуг.
- Клієнт вимагає від постачальника ідентифікаційних даних надати токен для певної SP-програми. Постачальник ідентифікаційних даних генерує токен і надсилає його клієнту;
- Клієнт використовує цей токен для автентифікації в SP-програмі.

Описана вище процедура відображає механізм автентифікації активного клієнта, тобто того, який може виконувати запрограмовану послідовність дій (наприклад, додатки iOS / Android). Браузер є пасивним клієнтом у тому сенсі, бо він може відображати лише сторінки, що запитуються користувачем. У цьому випадку автентифікація виконується шляхом автоматичного перенаправлення браузера між постачальником послуг веб-додатків і постачальником послуг.

Існує кілька стандартів, що визначають протокол взаємодії між клієнтами (активним та пасивним) та програмами IP / SP та формат підтримуваних токенів. Серед найбільш популярних стандартів - OAuth, OpenID Connect, SAML та WS-Federation. (рис.1.7)

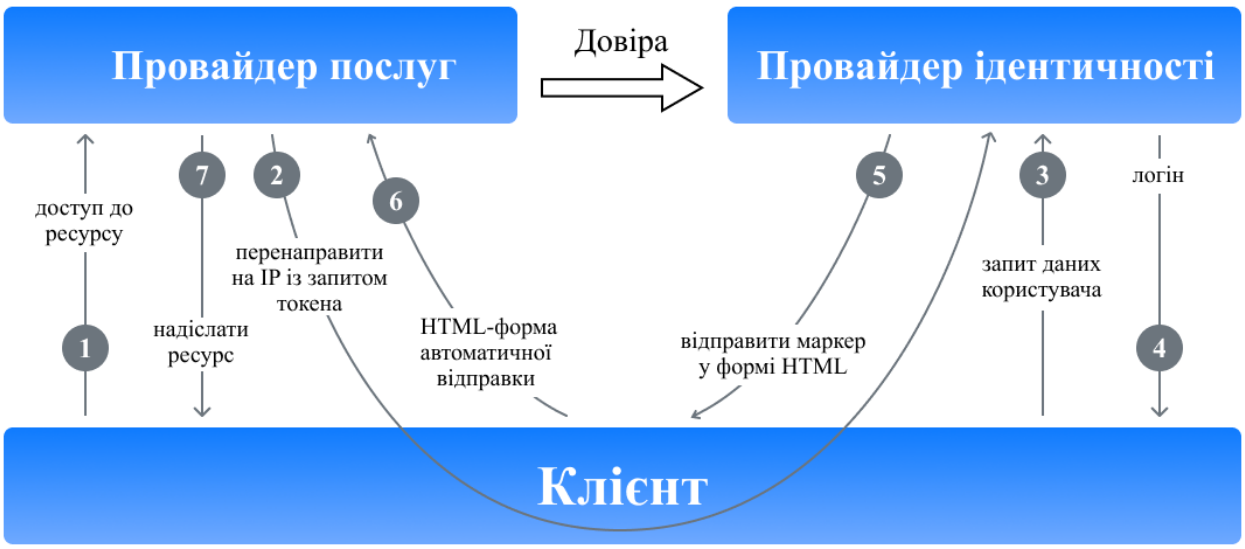


Рис. 1.7. Схема автентифікації «пасивного» клієнта з токеном, відправленим за допомогою схеми Bearer

Сам токен зазвичай являє собою структуру даних, що містить інформацію про те, хто створив токен, хто може бути одержувачем токену, термін дії, набір інформації про користувача (claims). Крім того, токен додатково підписується для запобігання несанкціонованим змінам та гарантій автентичності.

Під час автентифікації токена SP-програмою повинна виконувати такі перевірки:

- Токен був виданий заявкою надійного постачальника ідентифікаційних даних (перевірка поля issuer).
- Токен присвоюється поточною SP-програмою (перевірка поля audience).
- Токен ще не закінчився (перевірка поля expiration date).
- Токен справжній і не змінювався (перевірка підпису).
- Якщо перевірка пройшла успішно, SP-програма виконує авторизацію запиту на основі даних користувача, що містяться в токені.

1.5 Різновиди форматів токенів

Існує декілька поширених форматів веб-токенів:

- Simple web token (SWT) – найпростіший формат, який являє собою набір довільних пар ім’я/значення згідно формату HTML форм. Стандарт визначає кілька збережених назв: Publisher, Audience, ExpiresOn та HMACSHA256. Токен підписаний симетричним ключем, тому для створення або перевірки токена його повинні мати програми IP та SP.
- JSON WebToken(JWT) – містить три блоки,які розділені крапкою: адресу, набір полів (claims) та підпис. Перші два блоки представлені у форматі JSON та додатково закодовані у форматі base64. Набір полів містить будь-які пари ім’я/значення, крім того, стандарт JWT визначає кілька зарезервованих імен (iss, aud, exp та інші). Підпис може бути створений за допомогою симетричного та асиметричного алгоритмів шифрування. Крім того, існує окремий стандартний–формат зашифрованого JWT.[8]
- Security Assertion Markup Language (SAML) – визначає токени (SAML assertions) у форматі XML, що включає інформацію про емітента, сутність, необхідні умови для перевірки токена,а також набір додаткових тверджень(statements) про користувача. Токени SAML підписуються за допомогою асиметричної криптографії. Крім того, SAML-токени, на

відміну від попередніх форматів, містять механізм, який допомагає підтвердити володіння токеном, який запобігає перехопленню токенів атаками «man-in-the-middle» при використанні незахищеного з'єднання[9].

Стандарт мови Security Assertion Markup Language (SAML) описує методи взаємодії та протоколи між постачальником ідентифікації та постачальником послуг для обміну інформацією про автентифікацію та авторизацію за допомогою токенів. Спочатку версії 1.0 і 1.1 виходили між 2002 і 2003 роками, але версія 2.0 значно розширила стандарт і була опублікована в 2005 році.

Цей базовий стандарт є досить складним і підтримує безліч різних сценаріїв системної інтеграції. Основні «будівельні блоки» стандарту:

- Assertions– власна форма токенів SAML у форматі XML;
- Protocols– набір підтримуваних повідомлень між учасниками, включаючи запит на створення нового токена, отримання існуючих токенів, logout (вихід з системи), управління ідентифікаторами користувачів тощо;
- Bindings– механізми передачі повідомлень за допомогою різних транспортних протоколів. Підтримувані методи – HTTP Redirect, HTTP POST, HTTP Artifact (посилання на повідомлення), SAML SOAP, SAML URI (адреса отримання повідомлень) та інші;
- Profiles– типові сценарії застосування стандарту, що визначають набір вищезазначених операторів, що забезпечує кращу сумісність. Прикладом таких профілів є SSO Web Browser.

Окрім того, стандарт визначає формат обміну інформацією між учасниками, який включає перелік підтримуваних ролей, протоколів, атрибутів, ключів шифрування тощо. [10].

Існують також такі стандарти, як WS-Trust та WS-Federation, а також Oauth та Open ID Connect.

WS-Trust та WS-Federation є частиною групи стандартів WS- *, що описують веб сервіси SOAP/XML. Ці стандарти розробляються групою компаній, до складу якої входять Microsoft, IBM, Veri Sign та інші. Ці стандарти,

разом із SAML, досить складні і в основному використовуються у бізнес-сценаріях [9].

Інтерфейс служби авторизації Secure Token Service (STS) описується за допомогою стандарту WS-Trust. Ця послуга працює на SOAP і підтримує створення, оновлення та скасування токенів. Стандарт дозволяє використовувати токени різних форматів, але на практиці в основному використовуються токени SAML.

Стандарт WS-Federation стосується до механізмів взаємодії послуг між підприємствами, зокрема протоколів обміну токенами. Водночас WS розширює функції та інтерфейс служби STS, описані у стандарті WS-Trust.

Стандарт WS-Federation, серед іншого, визначає:

- Форма та методи обміну метаданими служби;
- Функція єдиного виходу(single sign-out);
- Служба атрибутів, яка надає додаткову інформацію про користувача;
- Служба псевдонімів, який дозволяє створювати альтернативні імена користувачів;
- Підтримка пасивних клієнтів (браузерів) із переспрямуванням.

Можна сказати, що WS-Federation дозволяє вирішувати ті самі проблеми, що і SAML, але їх підходи та реалізація дещо відрізняються.

На відміну від SAML та WS-Federation, стандарт Open Authorization(Oauth) не описує протокол автентифікації користувача. Натомість він визначає механізм, за допомогою якого одна програма отримує доступ до іншої за іменем користувача. Однак існують схеми, що дозволяють автентифікацію користувачів на основі цього стандарту.[11]

Перша версія стандарту була розроблена в 2007-2010 рр., А поточна версія 2.0 випущена в 2012 р. Версія 2.0 значно розширюється і одночасно спрощує стандарт, але не сумісна з версією 1.0. Сьогодні Oauth 2.0 дуже популярний і часто використовується для забезпечення авторизованого доступу та автентифікації третьої сторони [10].

Для кращого розуміння самого стандарту переглянемо приклад веб-програми, яка допоможе користувачам спланувати свої поїздки. Як частина функції, вона може аналізувати користувачів пошти, щоб перевірити, чи є листи, що підтверджують бронювання, і автоматично включати їх у запланований маршрут. Питання в тому, як ця веб-програма може безпечно отримувати доступ до електронних листів, таких як Gmail. Є два варіанти:

- Попросити користувача надати інформацію про свій рахунок? – поганий варіант;
- Попросити користувача створити ключ доступу? – можливо, але дуже складно.

Стандарт OAuth може вирішити цю проблему, для початку описавши як програмний додаток для подорожей, тобто клієнт, може отримати доступ до серверу з ресурсами при наявності дозволу власника. Загалом, весь процес складається з декількох етапів (рис.1.8):

- Користувач (власник ресурсу) дозволяє додатку (клієнту) отримати доступ до певного ресурсу у формі гранту.
- Додаток відправляє запит до сервера авторизації та отримує токен доступу до джерела в обмін на свій гарант. Коли програма викликається, вона додатково автентифікується за допомогою ключа доступу, який буде виданий їй під час попередньої реєстрації.
- Програма використовує цей токен для отримання необхідних даних із серверних джерел (у нашому випадку – служби Gmail).

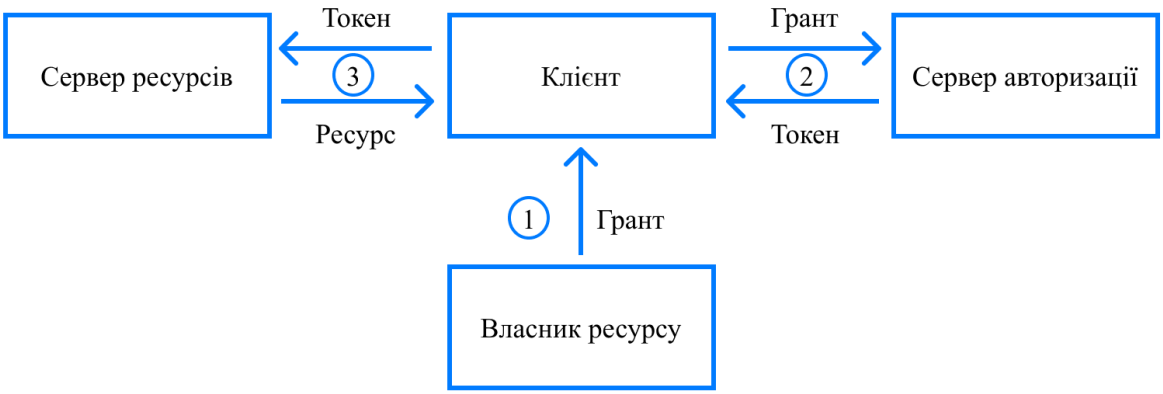


Рис. 1.8. Схема взаємодії компонентів у стандартах Oauth

Стандарт описує чотири типи грантів, що визначають можливі сценарії застосування:

- Authorization code - Це схвалення можна отримати від сервера авторизації після успішної автентифікації та підтвердження згоди на надання доступу. Цей метод найчастіше використовується у веб-додатках. Процес авторизації дуже схожий на пасивний механізм автентифікації клієнта в SAML та WS Federation;
- Implicit(Неявний) - використовується, коли програма не має можливості безпечно отримати токен із сервера авторизації (наприклад, програма JavaScript у браузері). У цьому випадку грант - це токен, отриманий від сервера авторизації, і крок 2 виключається із наведеного вище сценарію;
- Recource Owner Password Credentials(Повноваження пароля власника джерела) - грант представляє пару ім’я користувача/пароль. Її можна використовувати, якщо програма є «інтерфейсом» для сервера ресурсів (наприклад, програма є мобільним клієнтом для Gmail).
- Client Credentials(Клієнтські дані) - в цьому випадку немає користувача, і програма отримує доступ до своїх ресурсів за допомогою своїх ключів доступу (крок 1 виключено).

Стандарт не визначає формат токена, який отримує додаток: у сценаріях, на які звертається стандарт, додатку не потрібно аналізувати токен, оскільки він

використовується лише для доступу до ресурсів. Отже, ні токен, ні грант не можуть бути використані для автентифікації користувача. Однак, якщо додатку потрібно отримати надійні дані користувача, це можна зробити кількома способами:

- Як правило, API ресурсного сервера включає операцію, яка надає інформацію про користувача (наприклад, / me в Facebook API). Консоль може виконувати цю операцію кожного разу, коли отримує токен для ідентифікації клієнта. Цей прийом іноді називають псевдо-автентифікацією;
- Використовуйте стандарт OpenID Connect, розроблений як рівень облікових даних на додаток до OAuth (опублікований у 2014 році). Відповідно до цього стандарту сервер авторизації надає додатковий ідентифікаційний токен на етапі 2. Цей токен у форматі JWT міститиме набір визначених полів (assertions) з інформацією про користувача.

Варто згадати, що OpenID Connect, який замінює попередні версії стандартів OpenID 1.0 та 2.0, також містить набір додаткових доповнень для пошуку серверів авторизації, динамічної реєстрації клієнта та управління користувачами сеансів.

1.6 Висновки до розділу 1

У першому розділі дипломної роботи було розглянуто та проаналізовано значну кількість сучасних методів автентифікації користувачів у веб-програмах та стандартів авторизації. Було визначено переваги та недоліки методів автентифікації.

Також було досліджено системи хмарних обчислень, які використовують автентифікацію за допомогою ключ доступу.

Щодо автентифікації за допомогою токенів, було розглянуто їх найпоширеніші формати та стандарти користування.

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО МОДУЛЮ ДЛЯ АВТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ.

2.1 Вибір методу автентифікації

На сьогоднішній день одним із найефективніших методів автентифікації є автентифікація за допомогою токенів.

Токен - це частина даних, яка не має значення і не використовується самостійно, але в поєднанні з правильною системою токенізації стає важливим гравцем у захисті вашої програми. Автентифікація на основі токенів працює, гарантуючи, що кожен запит до сервера супроводжується підписаним токеном, який сервер перевіряє на справжність і лише потім відповідає на запит.

Використання токенів має багато переваг порівняно з традиційними методами, такими як файли cookie:

– Токени не мають статичного характеру. Токен є автономним і містить всю інформацію, необхідну для автентифікації. Це чудово підходить для масштабованості, оскільки звільняє ваш сервер від необхідності зберігати стан сесії.

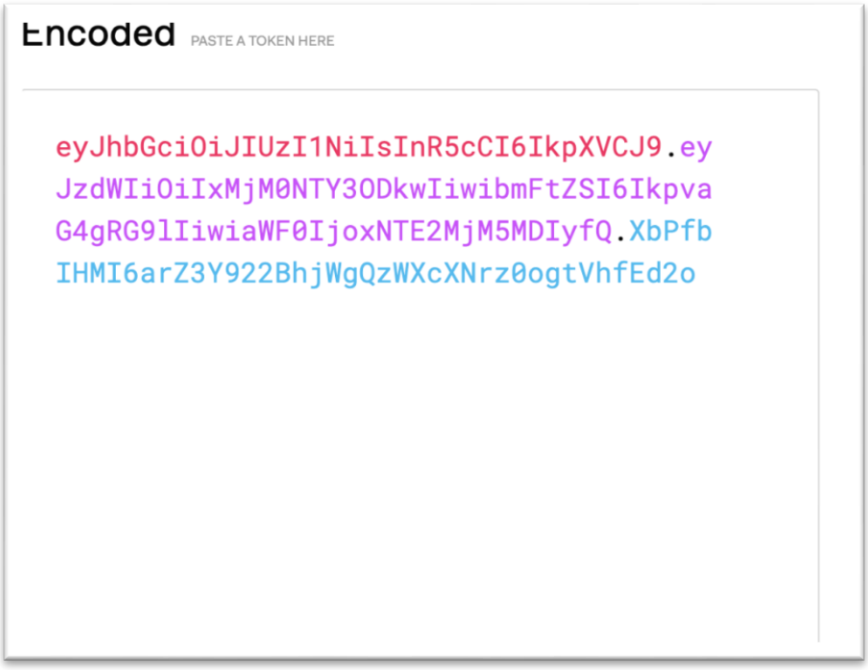


Рис. 2.1. Закодована форма JWT веб-токена

– Токени можна генерувати з будь-якого місця. Генерація токенів не відокремлюється від перевірки токена, що дозволяє вам обробляти підписання токенів на окремому сервері або навіть через іншу компанію.
– Дрібнодисперсний контроль доступу. В тілі точена (payload) ви можете легко вказати ролі та дозволи користувача, а також ресурси, до яких користувач може отримати доступ.[12]

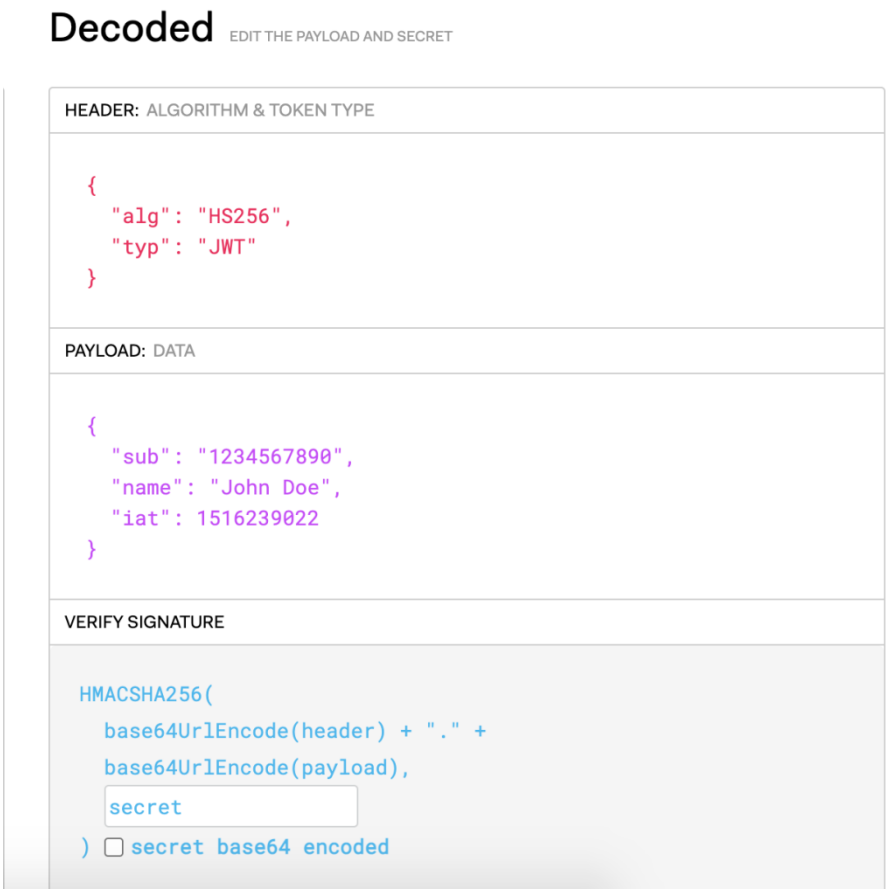


Рис. 2.2. Розкодована форма JWT веб-токена

Переваги JWT в порівняні зі SWT (Simple Web Token) та SAML (Security Assertion Markup Language) токенами:

– Більш компактний: JSON менш детальний, ніж XML, тому, коли він кодується, JWT менше(рис. 2.1, 2.2), ніж токен SAML(рис. 2.3, 2.4). Це робить JWT хорошим вибором для передачі в середовищах HTML і HTTP.

Рис. 2.3. Токен SAML

Рис. 2.4. Закодована форма SAML токена

– Більш безпечний: JWT можуть використовувати пару відкритого/приватного ключів у формі сертифіката X.509 для підписання.

JWT також може бути симетрично підписаний загальним секретом за допомогою алгоритму HMAC. І хоча токени SAML можуть використовувати пари відкритих / приватних ключів, таких як JWT, підписання XML цифровим підписом XML без введення неясних дірок у безпеці є дуже складним порівняно з простотою підписання JSON. Докладніше про алгоритми підписання JWT.

Рис. 2.5. Розкодована форма SAML токена

– Більш поширений: синтаксичні аналізатори JSON поширені в більшості мов програмування, оскільки вони безпосередньо відображаються в об’єктах. І навпаки, XML не має природного зіставлення документа з об’єктом. Це полегшує роботу з JWT, ніж твердження SAML.

– Легше обробляти: JWT використовується в Інтернеті. Це означає, що його легше обробити на пристроях користувача, особливо на мобільних.[13]

2.2 Клієнт-серверна архітектура програмного модулю

Складні веб-додатки включають в себе клієнтську та серверну частину, які реалізовують клієнт-серверну архітектуру(рис.2.6).

Клієнт-серверна архітектура- це обчислювальна модель, при якій сервер розміщує, постачає та управляє більшістю ресурсів та послуг, що споживаються клієнтом. Цей тип архітектури має один або кілька клієнтських комп'ютерів, підключених до центрального сервера через мережу або Інтернет. Ця система використовує обчислювальні ресурси. Клієнт / серверна архітектура також відома як мережева обчислювальна модель або мережа клієнт/сервер, оскільки всі запити та послуги доставляються через мережу.

Клієнтські комп'ютери забезпечують інтерфейс, що дозволяє користувачеві комп'ютера запитувати послуги сервера і відображати результати, які повертає сервер. Сервери чекають надходження запитів від клієнтів, а потім відповідають на них. В ідеалі сервер надає стандартизований прозорий інтерфейс для клієнтів, так що клієнти не повинні знати про особливості системи (тобто апаратного та програмного забезпечення), що надає послугу. Клієнти часто розташовані на робочих станціях або на персональних комп'ютерах, тоді як сервери знаходяться в інших місцях мережі, як правило, на більш потужних машинах. Ця обчислювальна модель особливо ефективна, коли клієнти та сервер мають різні завдання, які вони регулярно виконують. Наприклад, при обробці даних у лікарні, на клієнтському комп'ютері може бути запущена прикладна програма для введення інформації про пацієнта, в той час як на сервері працює інша програма, яка управляє базою даних, в якій інформація зберігається постійно. Багато клієнтів можуть одночасно отримувати доступ до інформації сервера, а

одночасно клієнтський комп'ютер може виконувати інші завдання, наприклад, надсилання електронної пошти.

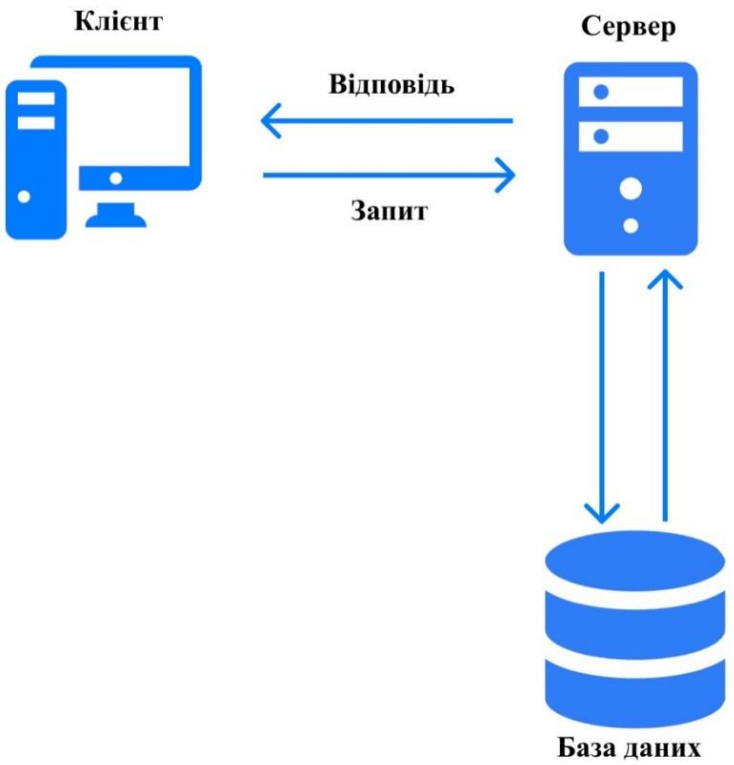


Рис. 2.6. Клієнт-серверна архітектура

Оскільки і клієнтський, і серверний комп'ютери вважаються незалежними пристроями, модель клієнт-сервер повністю відрізняється від старої моделі мейнфреймів, в якій централізований мейнфреймовий комп'ютер виконував усі завдання для пов'язаних з ним "німих" терміналів, які просто взаємодіяли з центральним мейнфреймом .

2.3 Клієнтська сторона

У веб-розробці термін "клієнтська сторона" означає все у веб-програмі, що відображається або розміщується на клієнті (пристрої кінцевого користувача).

Сюди входить те, що бачить користувач, наприклад текст, зображення та інший інтерфейс користувача, а також будь-які дії, які програма виконує в браузері користувача.

Мови розмітки, такі як HTML і CSS, інтерпретуються браузером на стороні клієнта. Крім того, багато сучасних розробників включають процеси клієнтської сторони в свою архітектуру додатків і відходять від того, щоб робити все на стороні сервера; Наприклад, бізнес-логіка для динамічних веб-сторінок *, як правило, працює на стороні клієнта в сучасному веб-додатку. Клієнтські процеси майже завжди написані на JavaScript.

Браузер на стороні клієнта інтерпретує HTML, CSS та JavaScript, які визначають, як головна сторінка веб-програми відображається для користувача. Сторінка також може реагувати на "події":

Наприклад, якщо миша користувача наводить курсор на одне із мініатюрних зображень фільму, зображення розширюється, а сусідні мініатюри трохи переміщуються в одну сторону, щоб звільнити місце для більшого зображення. Це приклад процесу на стороні клієнта; код всередині веб-сторінки реагує на мишу користувача та ініціює цю дію без зв'язку з сервером.

Клієнтська сторона також відома як front-end, хоча ці два терміни не означають точно одне і те ж. Клієнтська сторона стосується виключно місця, де виконуються процеси, тоді як інтерфейсний стосується типів процесів, які запускаються на клієнтській стороні.[14]

2.3.1 Засоби розробки клієнтської сторони

Для реалізації клієнтської сторони ефективно використовувати компонентний підхід. Для такого підходу рекомендовано використання бібліотеки для створення інтерфейсів – React. Також, щоб полегшити розробку було додано допоміжні бібліотеки для створення функціональної та візуальної частини додатку.

React - це бібліотека JavaScript, створена для побудови швидких та інтерактивних користувальницьких інтерфейсів для веб та мобільних додатків. Це відкрита, компонентна, інтерфейсна бібліотека, відповідальна лише за рівень перегляду програми. У архітектурі Model View Controller (MVC, рис. 2.7) рівень перегляду відповідає за те, як виглядає та відчуває себе програма. React був створений Джорданом Уолком, інженером-програмістом у Facebook.

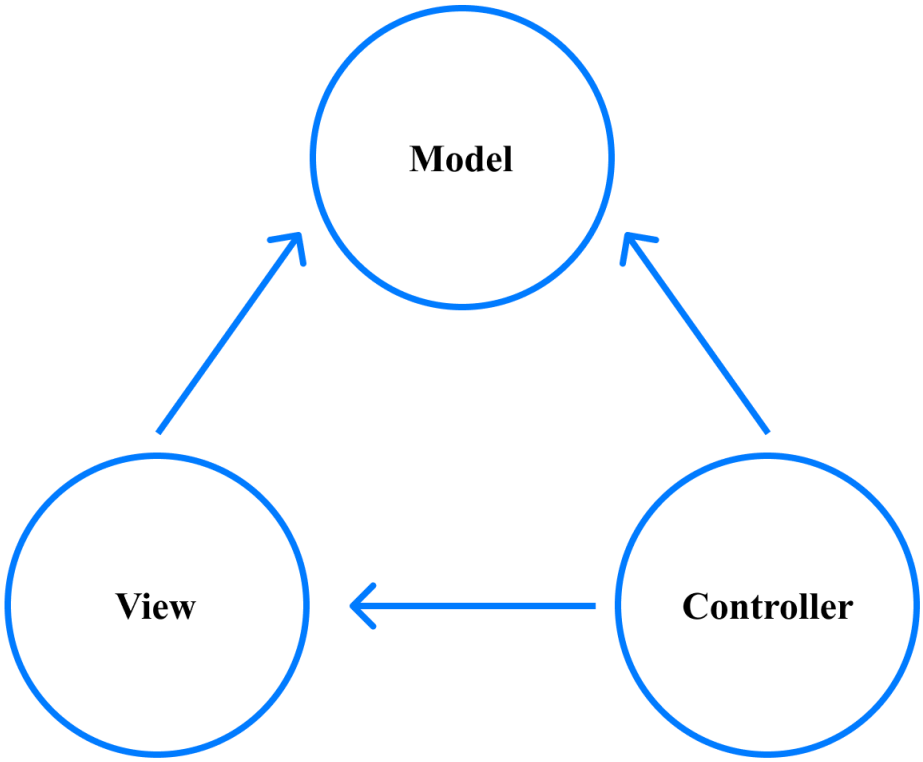


Рис. 2.7. Архітектура ModelViewController

Головні переваги використання бібліотеки React:

– Простота створення динамічних додатків: React спрощує створення динамічних веб-додатків, оскільки вимагає менше кодування та пропонує більше функціональних можливостей, на відміну від JavaScript, де кодування часто дуже швидко ускладнюється.

– Покращена продуктивність: React використовує віртуальний DOM, тим самим швидше створюючи веб-програми. Віртуальний DOM порівнює

попередні стани компонентів та оновлює лише ті елементи в Real DOM, які були змінені, замість того, щоб знову оновлювати всі компоненти, як це роблять звичайні веб-програми.

– Багаторазові компоненти: Компоненти є будівельними блоками будь-якої програми React, і одна програма, як правило, складається з декількох компонентів. Ці компоненти мають свою логіку та елементи керування, і їх можна використовувати повторно протягом усієї програми, що в свою чергу різко скорочує час розробки програми.

– Односпрямований потік даних: React слідує за односпрямованим потоком даних. Це означає, що при розробці програми React розробники часто вкладають дочірні компоненти в батьківські компоненти. Оскільки дані рухаються в одному напрямку, стає простіше налагоджувати помилки та знати, де проблема виникає у програмі на даний момент.

– Легкість вивчення: React легко засвоїти, оскільки він здебільшого поєднує основні концепції HTML та JavaScript з деякими корисними доповненнями. Однак, як і у випадку з іншими інструментами та фреймворками, вам доведеться витратити трохи часу, щоб отримати належне розуміння бібліотеки React.

– Його можна використовувати для розробки веб і мобільних додатків: ми вже знаємо, що React використовується для розробки веб-додатків, але це не все, що він може зробити. Існує фреймворк, який називається React Native, похідний від самого React, який надзвичайно популярний і використовується для створення прекрасних мобільних додатків. Отже, насправді React можна використовувати для створення як веб-, так і мобільних додатків.

– Виділені інструменти для легкої налагодження: Facebook випустив розширення Chrome, яке можна використовувати для налагодження програм React. Це робить процес налагодження веб-програм React швидшим та простішим.

HTML(HyperText Markup Language)– це мова для опису структури веб-сторінок.

HTML надає авторам засоби:

– Публікувати в Інтернеті документи з заголовками, текстом, таблицями, списками, фотографіями тощо.

– Отримувати інформацію в Інтернеті за гіпертекстовими посиланнями, натисканням кнопки.

– Розробляти форми для проведення операцій з віддаленими службами, для використання при пошуку інформації, оформленні бронювання, замовленні продуктів тощо.

– Включати таблиці розповсюдження, відеокліпи, звукові кліпи та інші програми безпосередньо до своїх документів.

Каскадні таблиці стилів (CSS) – це мова таблиці стилів, що використовується для опису презентації документа, написаного мовою розмітки, такою як HTML.CSS – це наріжна технологія Всесвітньої павутини, поряд з HTML та JavaScript.[15]

CSS описує спосіб відображення елементів HTML на екрані, папері або в інших носіях та економить багато роботи. Він може контролювати розміщення декількох веб-сторінок одночасно. Зовнішні таблиці стилів зберігаються у файлах CSS.

2.3.2 Інтерфейс програми

1. Запуск клієнтської сторони веб-програми відбувається командою *‘npm run client’*. Точка входу являється компонент App.jsx. Клієнтська частина запускається на локальному порту 3000.

Після переходу на сторінку, не авторизований користувач баче сторінку авторизації та реєстрації:

На цій сторінці користувач може здійснити авторизацію та реєстрацію шляхом вказування своєї пошти на паролю.

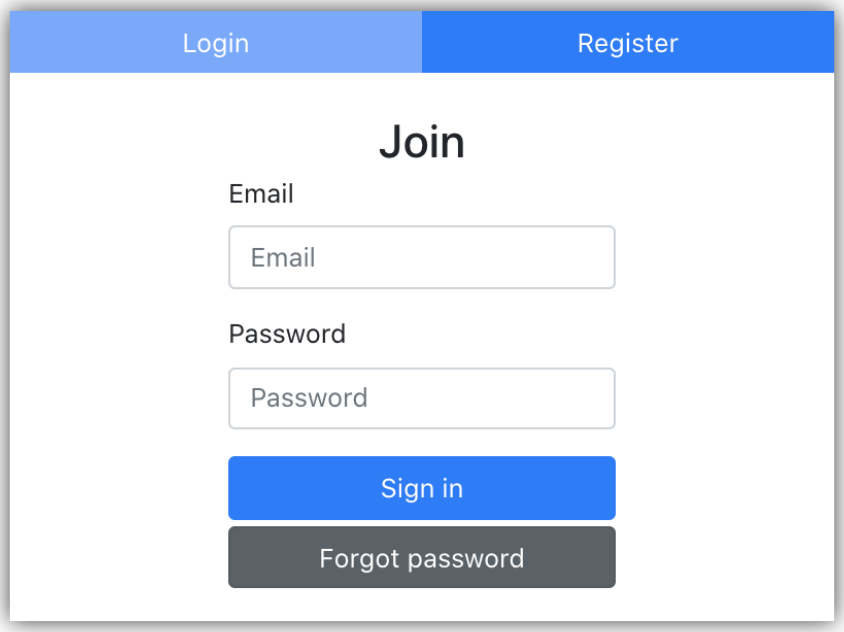


Рис. 2.8. Зовнішній вигляд сторінки входу та реєстрації

При введені неіснуючих даних користувача ми отримуємо помилку яка повідомляє про те, що ці дані не існують у БД (рис. 2.9):

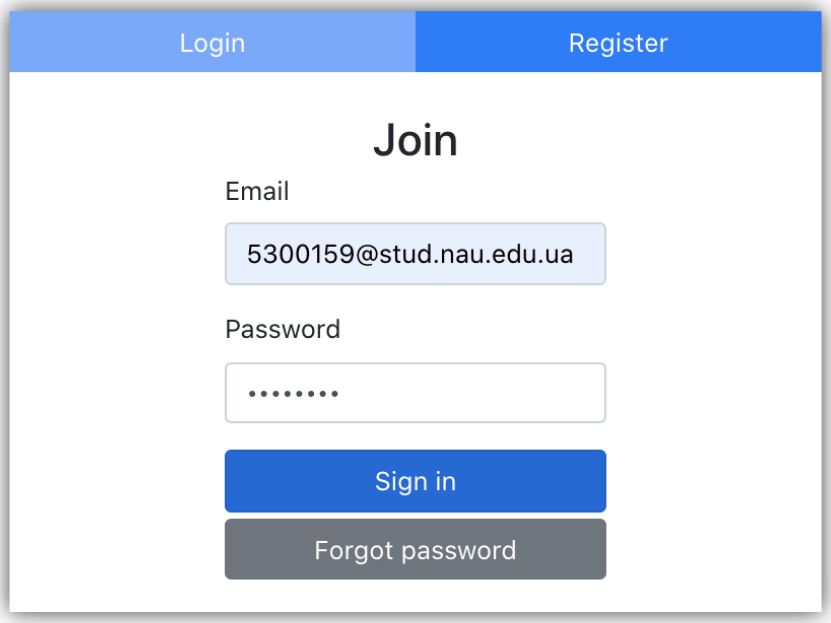


Рис. 2.9. Помилка про неіснуючого користувача

При спробі зареєструватись, якщо дані унікальні, та не існує такого користувача в БД, користувач буде зареєстрований та відразу авторизований з отриманням свого унікального JWT токена (рис. 2.10, рис. 2.11):

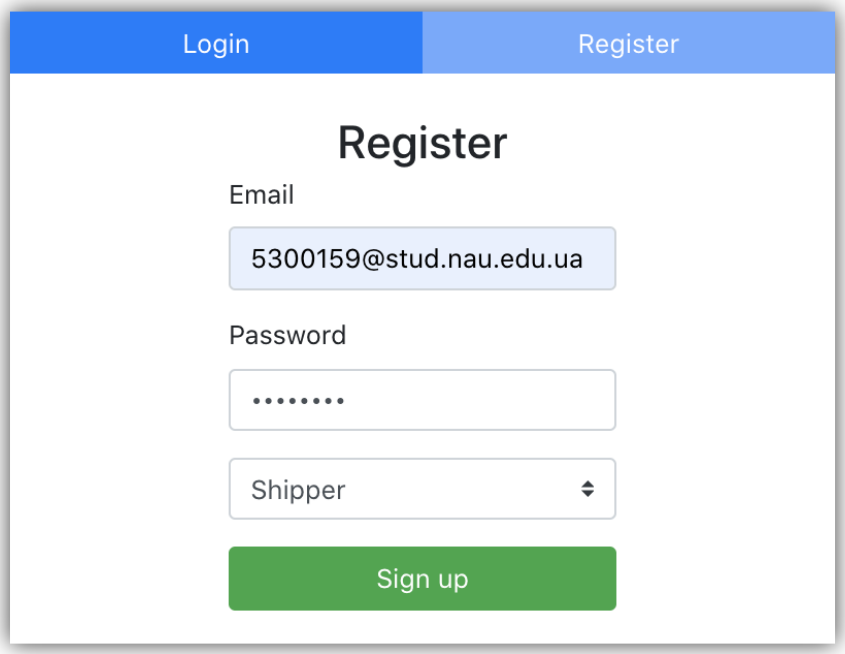


Рис. 2.10. Процес реєстрації

Рис. 2.11. Перехід на головну сторінку додатку

При спробі авторизації з невірним паролем, користувач отримує відповідне повідомлення(рис. 2.12):

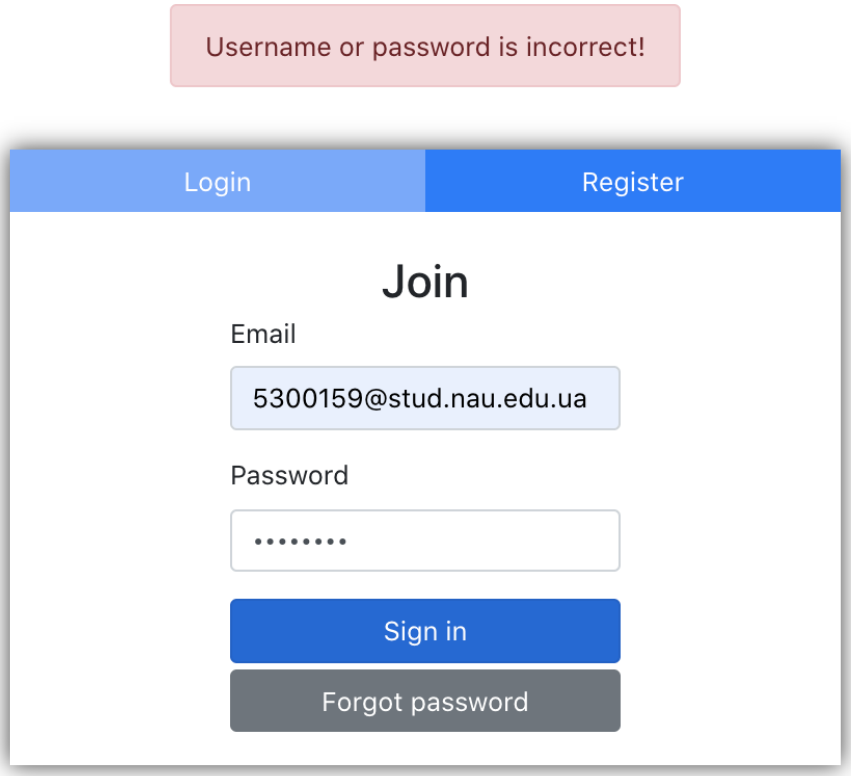


Рис. 2.12. Помилка про невірні дані

2.4 Серверна сторона

Як і на стороні клієнта, "сторона сервера" означає все, що відбувається на сервері, а не на клієнті. Раніше майже вся бізнес-логіка працювала на стороні сервера, і це включало рендеринг динамічних веб-сторінок, взаємодію з базами даних, автентифікацію ідентифікації та push-сповіщення.

Проблема розміщення всіх цих процесів на стороні сервера полягає в тому, що кожен запит із залученням одного з них повинен щоразу проходити шлях від клієнта до сервера. Це призводить до великої затримки. З цієї причини сучасні програми запускають більше коду на стороні клієнта; одним із випадків використання є рендеринг динамічних веб-сторінок в режимі реального часу, запуск скриптів у браузері, які вносять зміни до вмісту, який бачить користувач.[14]

Як і у випадку з "front-end" та "client side", серверний сервер також є терміном для процесів, що відбуваються на сервері, хоча серверний сервіс стосується лише типів процесів, а серверний - місця розташування процесів.

2.4.1 Засоби розробки серверної сторони

Node.js

Node.js - це між-платформене середовище виконання з відкритим кодом для розробки серверних та мережевих додатків. Додатки Node.js написані на JavaScript і можуть запускатися в середовищі виконання Node.js в OS X, Microsoft Windows та Linux. [16]

Node.js також надає багату бібліотеку різних модулів JavaScript, що значною мірою спрощує розробку веб-додатків, що використовують Node.js.

Особливості Node.js:

- Асинхронність та керування подіями - Усі API бібліотеки Node.js є асинхронними, тобто не блокуючими. По суті, це означає, що сервер на основі Node.js ніколи не чекає, поки API поверне дані. Сервер переходить до наступного API після його виклику, і механізм сповіщення про події Node.js допомагає серверу отримати відповідь від попереднього виклику API.
- Велика швидкість - побудована на механізмі JavaScript V8 від Google Chrome, бібліотека Node.js дуже швидко виконує код.
- Одно поточність, але масштабованість - Node.js використовує одно поточну модель із циклічним подією. Механізм подій допомагає серверу реагувати не блокуючим способом і робить сервер дуже масштабованим на відміну від традиційних серверів, які створюють обмежені потоки для обробки запитів. Node.js використовує одну потокову програму, і ця сама

програма може надавати послуги набагато більшій кількості запитів, ніж традиційні сервери, такі як Apache HTTP Server.

- Без буферизації - програми Node.js ніколи не буферизують будь-які дані. Ці програми просто виводять дані шматками.
- Ліцензія - Node.js випускається під ліцензією MIT. [16]

MongoDB

MongoDB - це орієнтована на документи база даних NoSQL, яка використовується для великого обсягу зберігання даних. Замість використання таблиць і рядків, як у традиційних реляційних базах даних, MongoDB використовує колекції та документи. Документи складаються з пар ключ-значення, які є основною одиницею даних у MongoDB. Колекції містять набори документів та функції, що є еквівалентом реляційних таблиць баз даних. MongoDB - це база даних, яка з’явилася приблизно в середині 2000-х.[17]

Особливості MongoDB:

- Кожна база даних містить колекції, які, в свою чергу, містять документи. Кожен документ може бути різним із різною кількістю полів. Розмір та зміст кожного документа можуть відрізнятися один від одного.
- Структура документа більше відповідає тому, як розробники будують свої класи та об'єкти у відповідних мовах програмування. Розробники часто говорять, що їх класи не є рядками та стовпцями, а мають чітку структуру з парами ключ-значення.
- Рядки (або документи, як їх називають у MongoDB), не повинні мати схему, визначену заздалегідь. Натомість поля можна створювати на льоту.
- Модель даних, доступна в MongoDB, дозволяє легше представляти ієрархічні відносини, зберігати масиви та інші більш складні структури.[17]

NPM - або "Node Package Manager" - це менеджер пакетів за замовчуванням для середовища виконання Node.js.

NPM складається з двох основних частин:

- інструмент CLI (інтерфейс командного рядка) для публікації та завантаження пакетів;
- Інтернет-сховище, яке розміщує пакети JavaScript;

Для розробки даного модулю було використано такі пакети, які полегшують процес розробки:

Bcrypt – бібліотека, яка пропонує розробнику велику кількість варіантів хешування паролів, для безпечного збереження їх в базі даних. Зберігати паролі в чистому вигляді дуже небезпечно, тому було прийнято рішення використовувати інструмент для хешування паролів.

Config – бібліотека, яка дозволяє зберігати один файл конфігурації для всього додатку, що є зручним для розробника.

Express - це мінімальний та гнучкий фреймворк для веб-додатків Node.js, який забезпечує надійний набір функцій для розробки веб і мобільних додатків. Це сприяє швидкому розвитку веб-додатків на основі Node.

Особливості фреймворку Express:

- Дозволяє налаштувати проміжні програми для відповіді на запити HTTP;
- Визначає таблицю маршрутизації, яка використовується для виконання різних дій на основі методу HTTP та URL-адреси;
- Дозволяє динамічно відображати HTML-сторінки на основі передачі аргументів у шаблони;

Generate-password – це зручна бібліотека для генерування випадкового паролю для користувачів, які забули свій пароль та хочуть отримати новий.

Jsonwebtoken – бібліотека для формування JWT веб-токену. Використана для здійснення автентифікації користувача.

Mongoose- це бібліотека моделювання об'єктних даних (ODM) для MongoDB та Node.js. Вона управляє взаємозв'язками між даними, забезпечує перевірку схем і використовується для перекладу між об'єктами в коді та представлення цих об'єктів у MongoDB.

2.4.2 Реалізація методу автентифікації за допомогою JWT токена

Модуль авторизації

Для авторизації користувач потрібен знаходитись на сторінці *"/auth/login"*, обрати вкладку *login* та ввести свої дані(рис. 2.13):

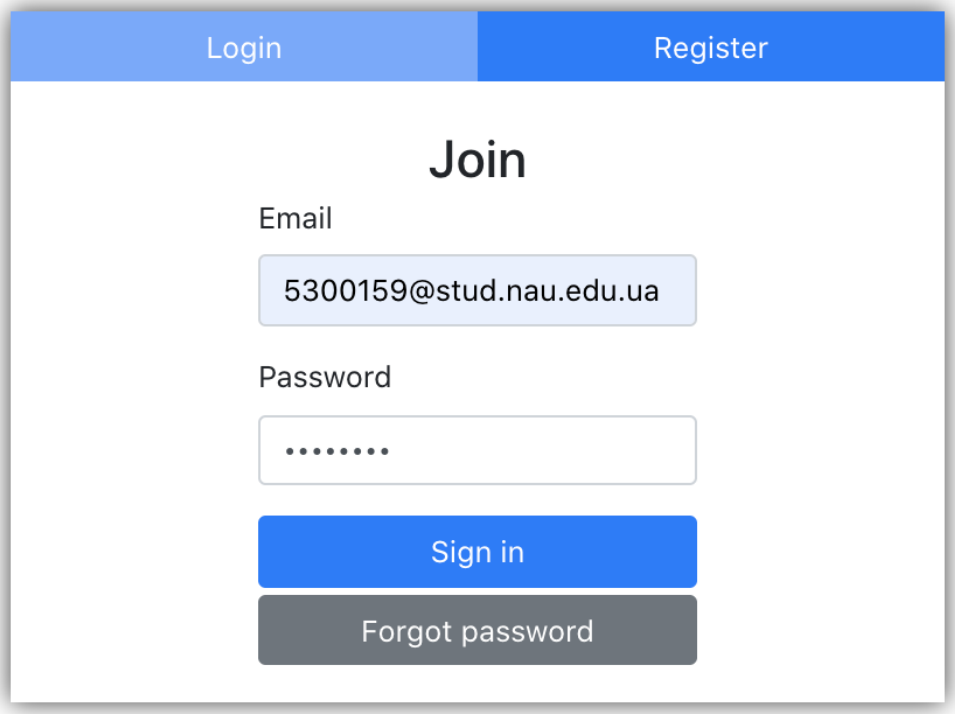


Рис. 2.13. Вікно входу

На серверній частині ця подія контролюється роутером даних, та якщо серверу надходить запит на авторизацію(рис. 2.14), він проводить цю процедуру за допомогою authController (див. Додаток А).

Рис. 2.14. Тіло запиту на вхід

AuthController в свою чергу направляє ці дані в функцію *login*, яка перевіряє введені дані на достовірність.

Якщо дані вірні, контролер генерує JWT веб токен за допомогою jwtSecret та унікального userId для цього користувача та відповідає на запит(рис. 2.15):

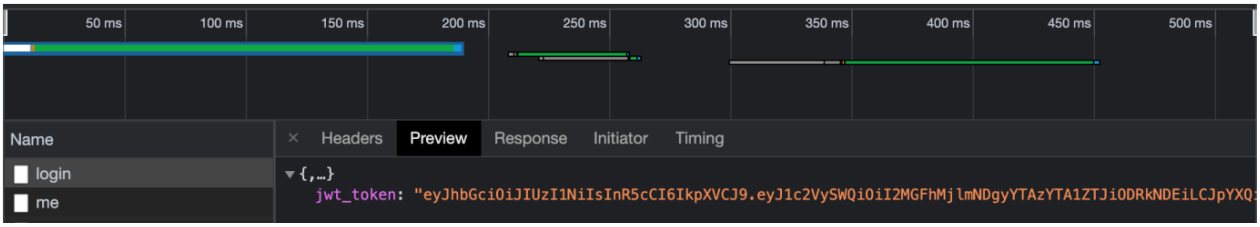


Рис. 2.15. JWT токен у відповідь на запит логіну

Токен буде дійсний 1 годину, після закінчення якої, користувачу потрібно буде провести операцію авторизації ще раз. В полі payload JWT токена знаходиться вся необхідна інформація для проведення автентифікації, а саме його унікальний userId та поле expiresIn, яке вказує коли токен стане не дійсним(рис. 2.16):

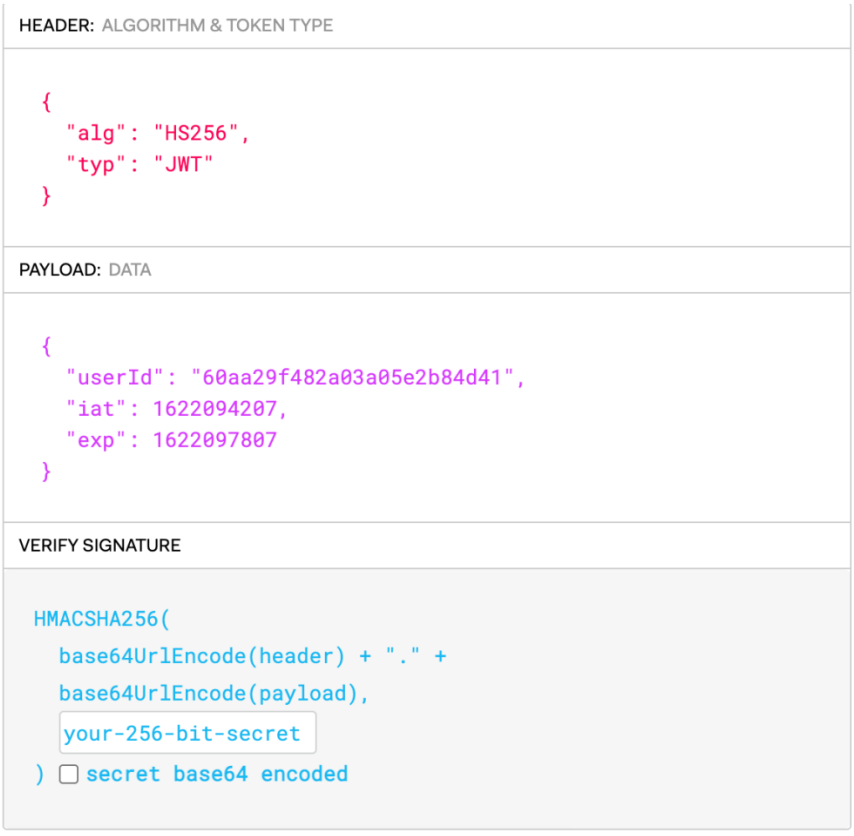


Рис. 2.16 Тіло JWT токена

Якщо дані некоректні, сервер відповідає клієнту про помилку(рис. 2.17, рис. 2.18) та просить ввести дані спочатку (див. Додаток Б).

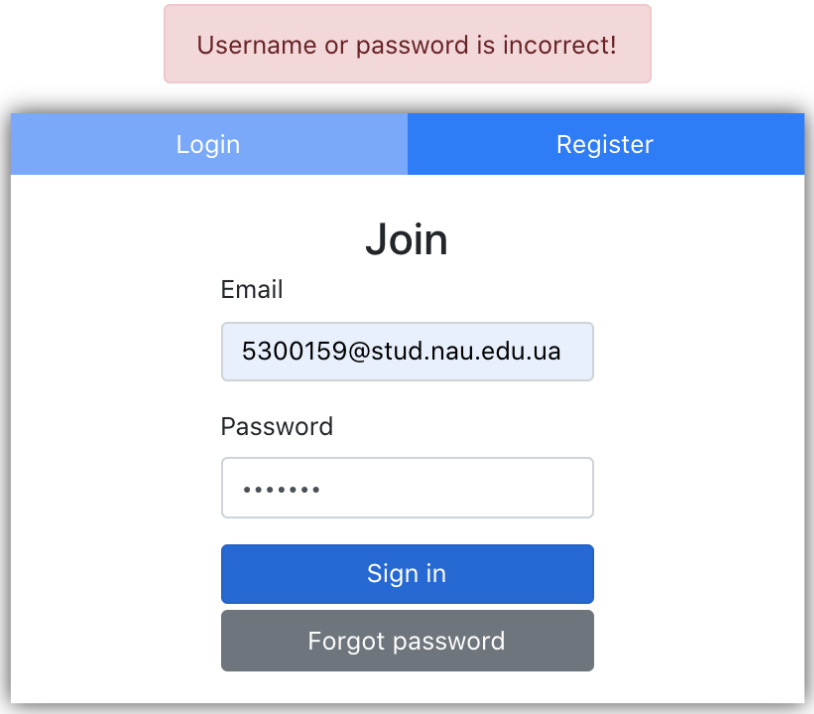


Рис. 2.17. Помилка на стороні клієнта про невірні дані

Рис. 2.18. Відповідь від сервера про невірні дані

Для реєстрації користувач потрібен знаходитись на сторінці *"/auth/register"*, обрати вкладку *register* та ввести свої унікальні дані(рис. 2.19):

На серверній частині ця подія контролюється роутером даних, та якщо серверу надходить запит на реєстрацію(рис. 2.20), він проводить цю процедуру за допомогою authController (див. Додаток В).

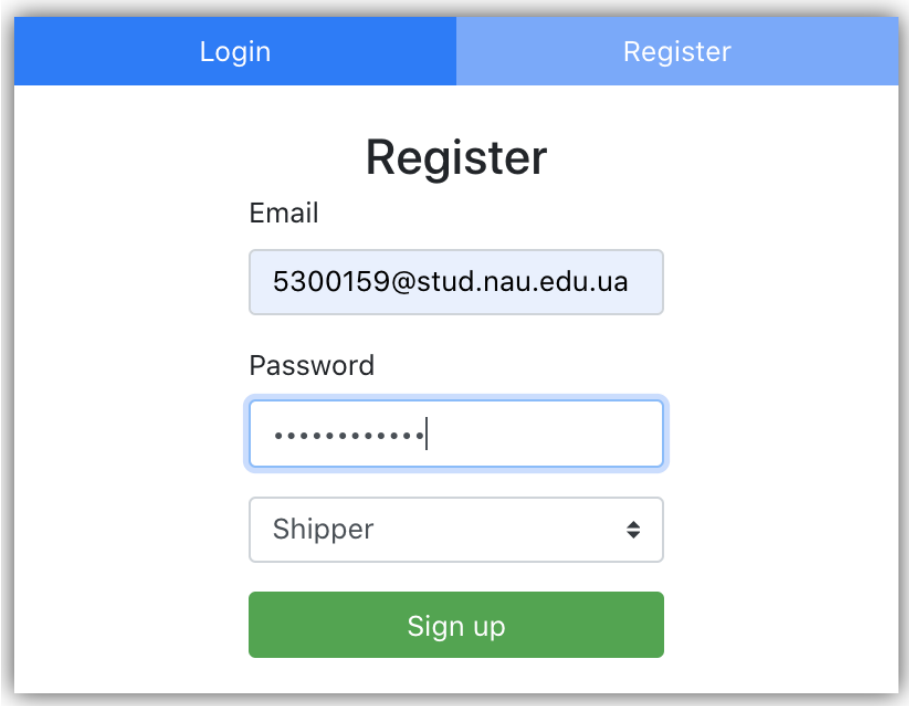


Рис. 2.19. Форма реєстрації

Рис. 2.20 Запит на реєстрацію

AuthController для реєстрації використовує модуль *register*, який валідує введені дані та відповідає користувачу сповіщенням про успішну реєстрацію(рис. 2.21) або про помилку(рис. 2.22), та описує її(див. Додаток Г).

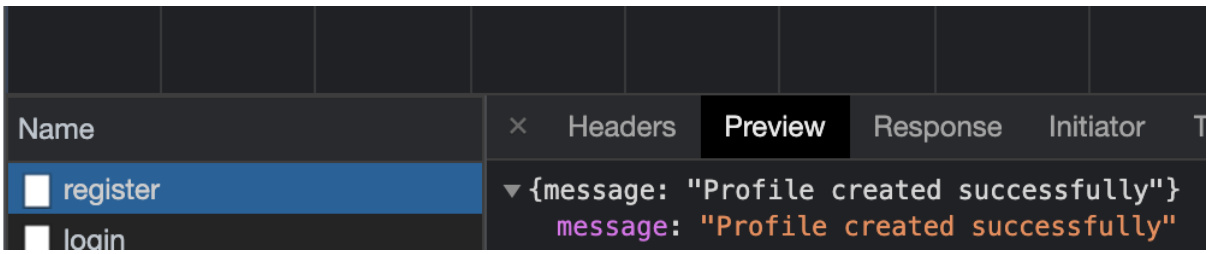


Рис. 2.21. Відповідь сервера про успішну реєстрацію

Рис. 2.22. Помилка від серверу про вже існуючого користувача

Одразу після успішної реєстрації клієнт робить запит на вхід та проводиться процедура описана в модулі авторизації.

Модуль-middleware для перевірки автентичності користувача

Перед переходом до контролерів, які здійснюють будь-які операції з даними користувача, сервер додатково проводить процедуру автентифікації користувача за допомогою *authMiddleware*(див. Додаток Ґ).

AuthMiddleware спочатку перевіряє body та headers запиту на коректність, після чого намагається автентифікувати користувача (див. Додаток Д).

Автентифікація проходить методом *.verify()* бібліотеки *jsonwebtoken.* Для цієї процедури middleware витягує jwtSecret, який знаходиться в файлі конфігурації серверу, та сам токен, який нам присилає клієнт в Authorization header запиту.

Приблизний вигляд Authorization header: «BeareryJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c». Сам токен поділений на 3 блоки, які розділені точкою (.). В перший блок розміщується інформація про тип токену та хешування. В другому знаходиться інформація про користувача(userId), а в третьому зберігається сигнатура. Перші 2 блоки знаходяться у форматі JSON, що дає змогу швидко перетворити цей формат у javascript та продовжити автентифікацію.

Якщо користувача не автентифіковано, то клієнт отримує відповідь від серверу з таким тілом у форматі JSON (рис. 2.23):

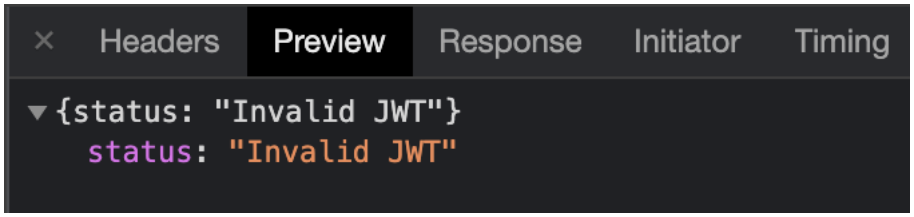


Рис. 2.23. Відповідь сервера про невірний JWT токен

Після цього користувач не зможе робити ніяких дій з даними на стороні клієнта, поки не проведе повторну авторизацію для отримання токена, додатково відбудеться переадресація на сторінку авторизації.

2.5 Висновки до розділу 2

В даному розділі було проаналізовано поняття серверної та клієнтської частини та їх принципи роботи.

Було перераховано усі інструменти для розробки програмного продукту, який забезпечує автентифікацію, авторизацію та реєстрацію користувачів.

Також у цьому розділі показано структурні схеми системи автентифікації, авторизації та реєстрації користувачів та наведено більшість можливих подій при проведенні цих операцій, які повинні контролюватися сервером та клієнтом.

ВИСНОВКИ

Під час виконання дипломної роботи було визначено актуальність теми, предмета та об’єкта дослідження, визначено мету та оцінено практичну значимість продукту. Розглянуто методи вирішення проблеми та проведено аналіз переваг та недоліків існуючих аналогів. Досягнуто поставленої мети, а саме: дослідити метод автентифікації користувачів інформаційних систем та розробити функціонал для автентифікації користувача за допомогою JWT веб-токену.

Перед початком розробки веб-продукту з підтримкою автентифікації користувача були сформовані чіткі вимоги, що має виконувати програма, обрано набір технологій для розробки.

Для розробки веб-додатку було використано набір інструментів для спрощення розробки та користування ним:

- MongoDB – система управління базою даних.
- Express – фреймворк для розробки серверної частини на платформі Node.js
- Mongoose – управління БД
- React – бібліотека для розробки клієнтської частини
- Node.js – платформа для розробки серверної частини
- Bcrypt – для хешування паролів
- Json web token – створення токенів доступу для автентифікації користувачів

Структура програми була представлена логічними схемами програми, що забезпечує коректне розуміння функціоналу веб-додатку.

Розроблений програмний продукт дозволяє вирішити основні задачі для будь якої системи, яка взаємодіє, зберігає та використовує дані користувачів, а саме:

- Авторизація користувача
- Реєстрація користувача

– Автентифікація користувача

Розроблений веб-додаток має не важкий для розуміння інтерфейс та велику швидкість роботи завдяки використанню нових та постійно-підтримуємих інструментів для розробки.

Дану веб-програму в майбутньому можна удосконалювати, налагоджуючи систему авторизації, автентифікації та реєстрації.

Цей продукт є універсальний для більшості веб-додатків, тож його можна використовувати для створення майбутніх програм для користувачів з особистими кабінетами.

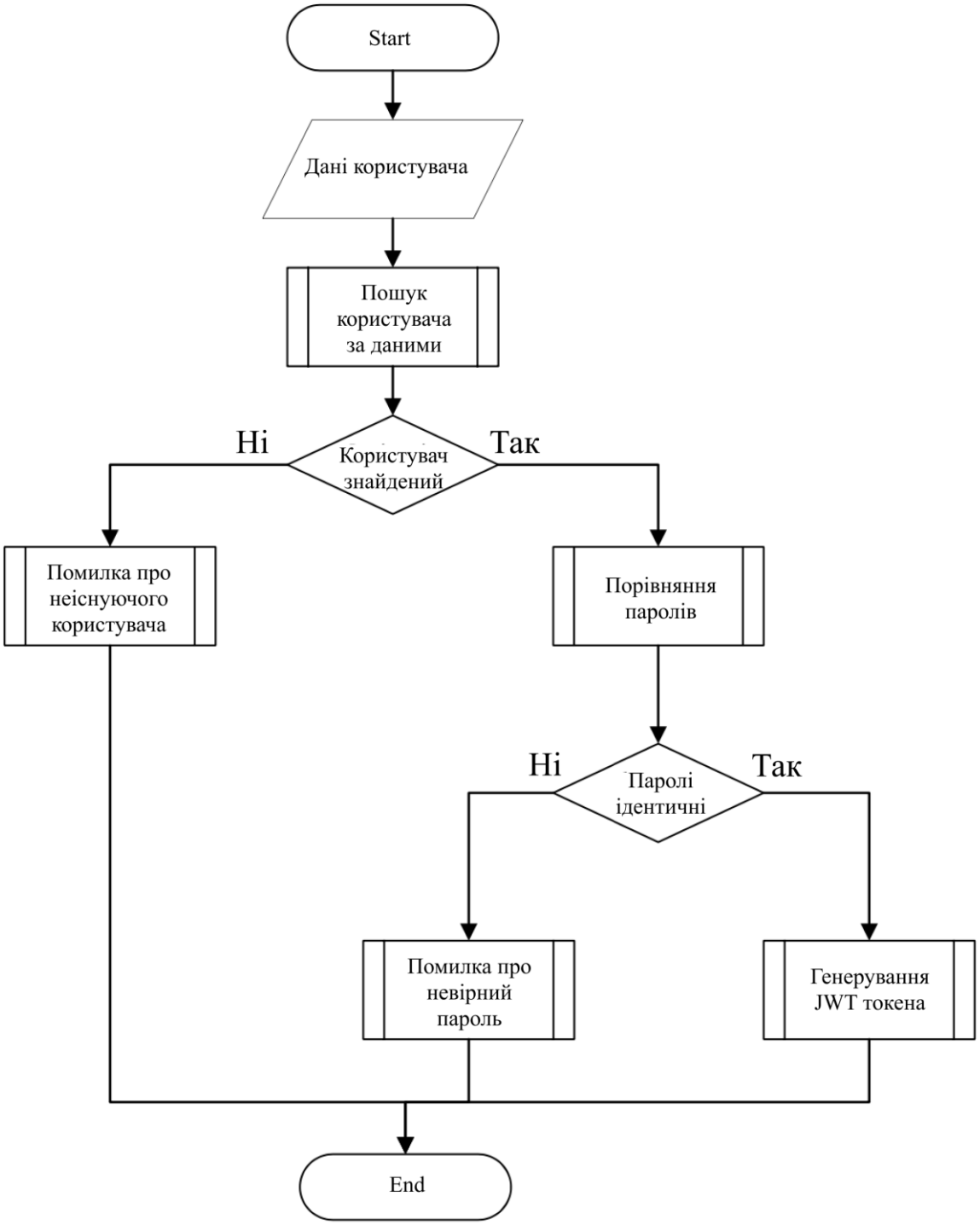
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Ідентифікація (інформаційна безпека) [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Ідентифікація_(інформаційна_безпека) Дата доступу 28.05.2021
2. Ідентифікація (інформаційна безпека) [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/АвтентифікаціяДата доступу 28.05.2021
3. Thomas D. The Current State Of Authentication: We Have A PasswordProblem. Smashing Magazine 2016.
4. Сміт. Р. Аутентифікація: від паролей до відкритих ключів (Authentication:From Passwords to Public Keys First Edition.) / Річард Э. Сміт — М.:Вильямс, 2002. — 432 с
5. Microsoft. Basic Authentication [Електронний ресурс] / Microsoft. – 2005. – Режим доступу:https://docs.microsoft.com/enus/windows/desktop/secauthn/basic-authentication-conceptsДата доступу: 28.05.2021
6. Офіційна документація Amazon Web Service [Електронний ресурс]/ Amazon — Режим доступу: https://docs.aws.amazon.com/en_us/cli/latest/userguide/cli-chap-gettingstarted.htmlДата доступу: 28.05.2021
7. Duncan D. "Two-factor authentication" / de Borde Duncan., 2009.
8. Корченко О. Метод автентифікації користувачів інформаційних систем за їх рукописним почерком з багатокроковою корекцією первинних даних / О. Корченко, А. Давиденко, О. Висоцька // Захист інформації. 2019. – Том 21, №1. – С. 40-51. DOI: 10.18372/2410-7840.21.135462.
9. Boyd R. Getting Started with OAuth 2.0. / Ryan Boyd — 1005 Gravenstein Highway North, Sebastopol, CA 954: O’Reilly Media, Inc, 2012. — 56 с.

10. Cantor S. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0 / Scott Cantor — Errata Composite: 2015. – 93
11. Сучасні стандарти ідентифікації: OAuth 2.0, OpenID Connect, WebAuthn [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/491116/ Дата доступу 28.05.2021
12. Token based authentication [Електронний ресурс] – Режим доступу:https://auth0.com/learn/token-based-authentication-made-easy/Дата доступу 28.05.2021
13. JSON Web Tokens [Електронний ресурс] – Режим доступу: https://auth0.com/docs/tokens/json-web-tokens Дата доступу 28.05.2021
14. What are client side and server side? [Електронний ресурс]— Режим доступу: www.cloudflare.com/en-gb/learning/serverless/glossary/client-side-vs-server-side/ Дата доступу: 28.05.2021
15. Каскадні таблиці стилів [Електронний ресурс]— Режим доступу: http://www.kievoit.ippo.kubg.edu.ua/kievoit/2013/139/139.htmlДата доступу: 28.05.2021
16. Node.js – Introduction [Електронний ресурс]— Режим доступу: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htmДата доступу: 28.05.2021
17. What is MongoDB [Електронний ресурс]— Режим доступу: https://www.guru99.com/what-is-mongodb.htmlДата доступу: 28.05.2021

Додаток А

Модуль авторизації



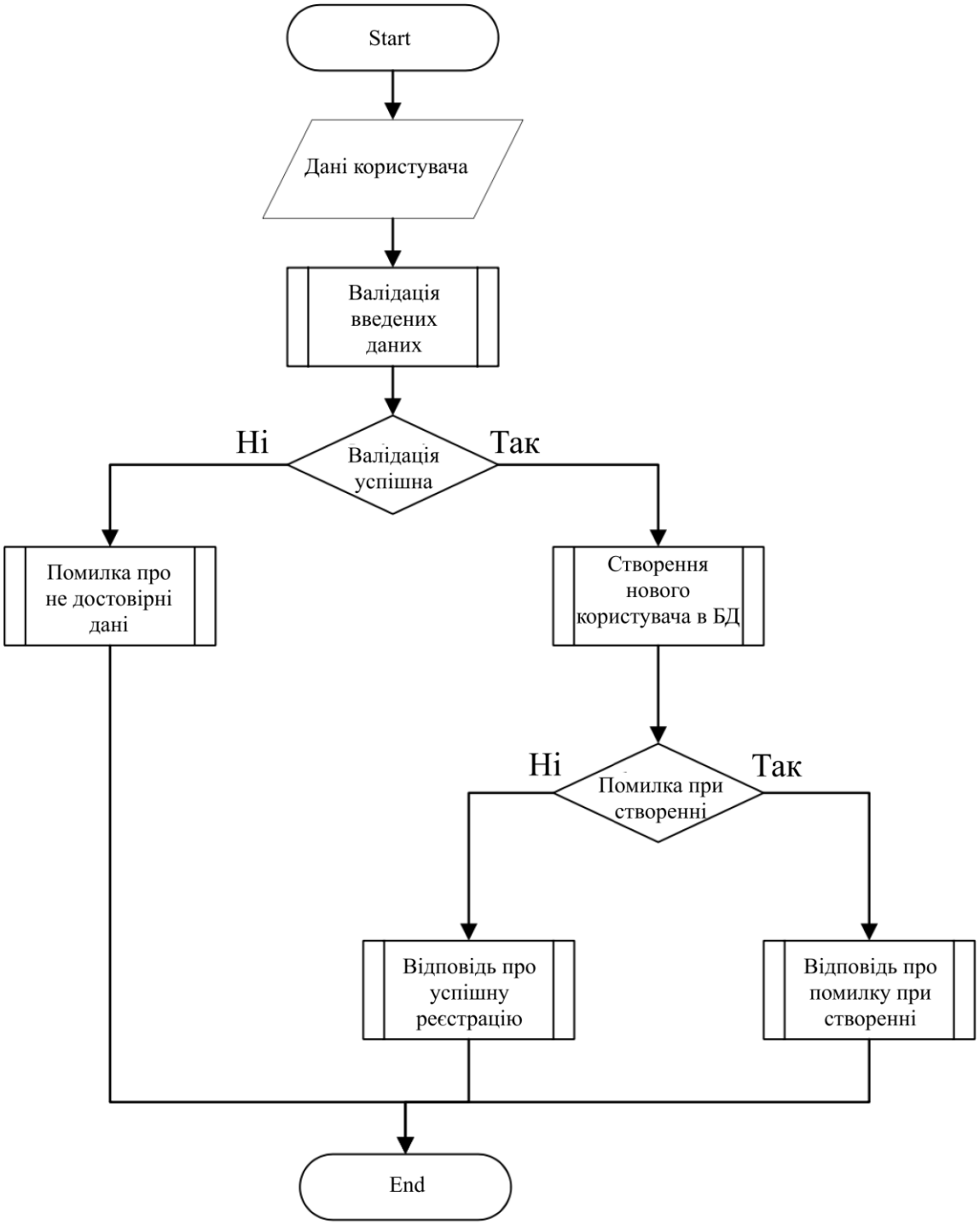
Додаток Б

Реалізація модулю авторизації

```
module.exports.login = async (req, res) => {
const { email, password } = req.body;

try {
const user = await User.findOne({ email }).catch(err => {
    res.status(500).json({ message: err.message });
  });

if (!user) {
    res.status(400).json({ message: "Can't find this user" });
  }
const isMatch = await bcrypt.compare(password, user.password);

if (!isMatch) {
    res.status(400).json({ message: 'Username or password is incorrect!' });
  }
  res.status(200).json({
jwt_token: jwt.sign({ userId: user._id }, config.get('jwtSecret'), {
expiresIn: '1h',
    }),
  });
 } catch (err) {
  res.status(500).json({ message: err.message });
 }
};
```

Додаток В

Модуль реєстрації



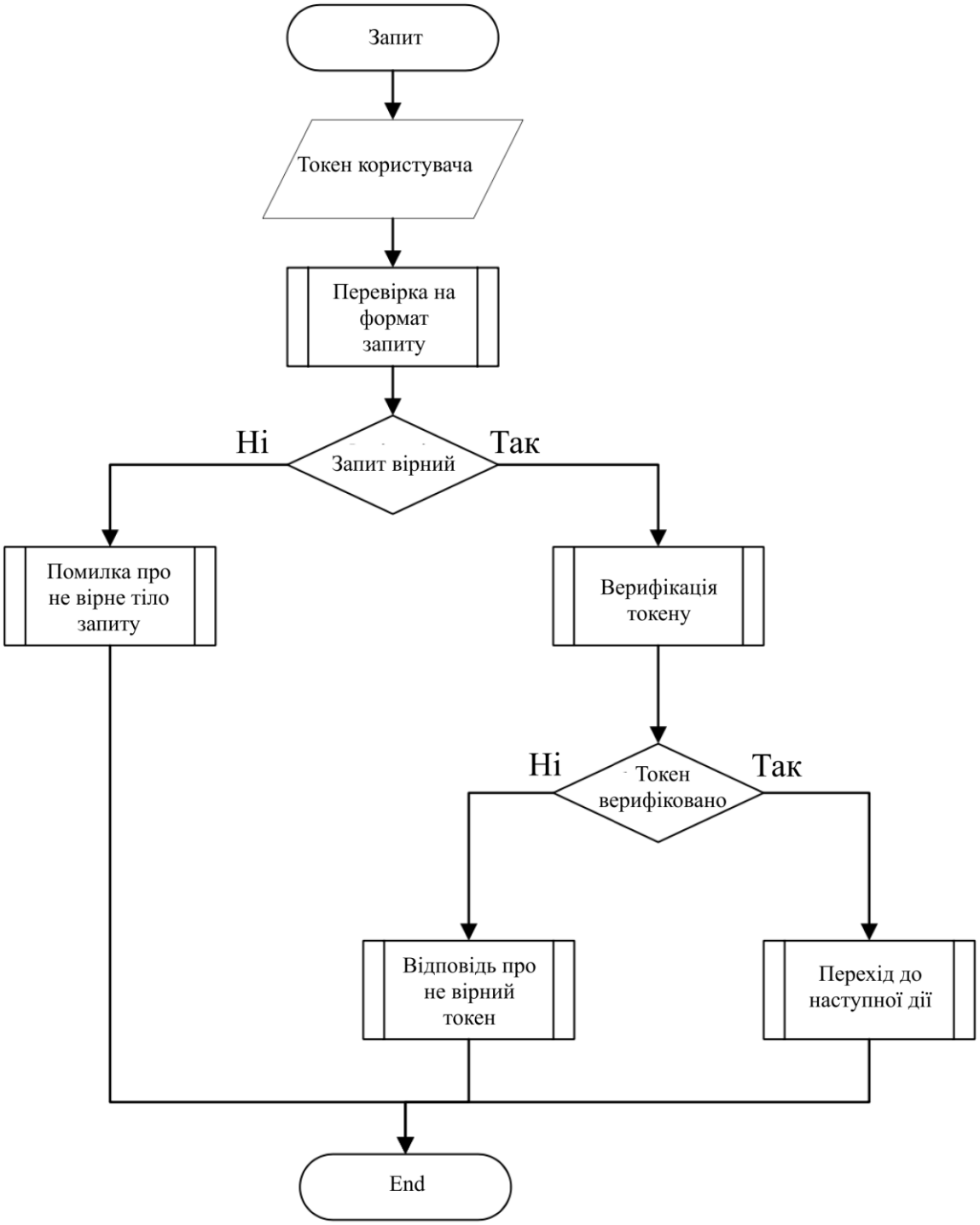
Додаток Г

Реалізація методу реєстрації

```
module.exports.register = async (req, res) => {
const { email, password } = req.body;
let { role } = req.body;
  role = role.toUpperCase();
const notExistsRole = !roles[role];
if (notExistsRole) {
return res.status(400).json({ message: 'This role are not exists' });
  }

const created_date = Date.now();
const hashedPassword = await bcrypt.hash(password, 10);
const user = newUser({
    email,
password: hashedPassword,
role: role.toUpperCase(),
    created_date,
  });

  user.save().then(() => {
      res.status(200).json({ message: 'Profile created successfully' });
    })
    .catch(err => {
      res.status(400).json({ message: err.message });
    });
};
```

Додаток Ґ

Модуль-middleware для перевірки автентичності користувача



Додаток Д

Реалізація автентифікації користувача

```
module.exports = async (req, res, next) => {
const authHeader = req.headers['authorization'];

if (!authHeader) {
return res.status(401).json({ message: 'No authorization header found' });
  }

const [, jwtToken] = authHeader.split(' ');
try {
const jwtSecret = config.get('jwtSecret');
    req.user = await jwt.verify(jwtToken, jwtSecret);
next();
  } catch (err) {
return res.status(401).json({ status: 'Invalid JWT' });
  }
};
```