Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра прикладної математики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Вибір ефективних механізмів автентифікації для розроблення Web-застосунків з різною архітектурою та навантаженням
(тема)

Виконав:
здобувач _2_ року навчання, групи _САУм-23-1
Гаврисюк О.Б.
(прізвище, ініціали)
Спеціальність
124 Системний аналіз
(код і повна назва спеціальності)
Тип програми _освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма
Системний аналіз і управління
(повна назва освітньої програми)
Керівник доц. Поляков А.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ПМ _____ Сидоров М.В.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет _інформаційно-аналітичних технологій та менеджменту
Кафедра _прикладної математики
Рівень вищої освіти _другий (магістерський)
Спеціальність _124 Системний аналіз
(код і повна назва)
Тип програми _освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма _Системний аналіз і управління
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри ПМ _____
(підпис)
“ _25_ ” _листопада_ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Гаврисюку Олександру Богдановичу
(прізвище, ім’я, по батькові)

1. Тема роботи Вибір ефективних механізмів автентифікації для розроблення Web-застосунків з різною архітектурою та навантаженням

затверджена наказом по університету від 22 листопада 2024 р. № 1228 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 6 січня 2025 р.

3. Вихідні дані до роботи набір даних подій на платформі електронної комерції

4. Перелік питань, що потрібно опрацювати в роботі
1. Системний аналіз предметної області
2. Вибір і обґрунтування методу розв’язання
3. Програмна реалізація
4. Результати обчислювального експерименту
5. Аналіз можливих застосувань

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

1. Актуальність теми роботи
2. Постановка задачі
3. Системний аналіз предметної області
4. Метод чисельного аналізу
5. Результати обчислювального експерименту

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Підбір та вивчення технічної літератури за темою роботи	25 листопада – 1 грудня 2024 р.	виконано
2	Вибір та обґрунтування методу	2 – 8 грудня 2024 р.	виконано
3	Розробка алгоритму і програми	9 – 22 грудня 2023 р.	виконано
4	Проведення аналітичних досліджень та розрахунків	23 – 29 грудня 2024 р.	виконано
5	Робота над текстом пояснювальної записки	30 грудня 2024 р. – 9 січня 2025 р.	виконано
6	Представлення роботи на рецензію в ЕК	10 січня 2025 р.	виконано

Дата видачі завдання _25_ листопада 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Поляков А.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 94 с., 1 табл., 15 рис., 1 дод., 10 джерел.

АУТЕНТИФІКАЦІЯ, АВТОРИЗАЦІЯ, БЕЗПЕКА, ВЕБ-ДОДАТОК, ПРОТОКОЛИ.

Об’єкт дослідження – Методи аутентифікації та авторизації у фронт–енд веб-додатках.

Мета роботи – Дослідження методів аутентифікації та авторизації, які можна безпечно реалізувати у фронт–енд веб-додатках, з акцентом на токенізовані рішення.

Методи дослідження – Аналіз сучасних веб-технологій, протоколів OpenID Connect та OAuth 2.0, а також практичне вивчення їх впровадження у реальних клієнтських проєктах.

Реферат відображає основні аспекти дипломної роботи, спрямованої на дослідження сучасних підходів до аутентифікації та авторизації у веб-додатках. Робота включає:

У кваліфікаційній роботі описано значення розвитку веб-технологій і виникнення парадигм фронт–енд та бек–енд. Вказано, що еволюція JavaScript дала змогу створювати автономні веб-додатки, які потребують нових підходів до забезпечення безпеки. Основна частина роботи присвячена дослідженню та порівнянню сучасних методів аутентифікації та авторизації, зокрема рішень на основі токенів. Описано протоколи OpenID Connect та OAuth 2.0, що є стандартами в сучасній веб-розробці.

Практична частина демонструє впровадження цих рішень у трьох реальних проєктах, що доводить їх ефективність і зручність.

Рекомендації щодо використання результатів роботи стосуються впровадження токенізованих рішень у веб-додатках із високими вимогами до безпеки. веб-додатки різного призначення, включаючи системи електронної

комерції, CRM–системи та освітні платформи. Сприяє підвищенню рівня безпеки у веб-додатках та вдосконаленню процесів аутентифікації й авторизації.

Результати, отриманні у кваліфікаційній роботі можуть бути використані щодо подальшого вдосконалення методів аутентифікації та авторизації, включаючи можливість використання біометричних даних та покращення алгоритмів захисту токенів.

ABSTRACT

Introductory note: 94 pages, 1 table, 15 figures, 1 appendix, 10 sources.

AUTHENTICATION, AUTHORIZATION, PROTOCOLS, SECURITY, WEB APPLICATION.

Object of research – Authentication and authorization methods in front–end web applications.

Purpose of work – Research of authentication and authorization methods that can be safely implemented in front–end web applications, with an emphasis on tokenized solutions.

Research methods – Analysis of modern web technologies, OpenID Connect and OAuth 2.0 protocols, as well as practical study of their implementation in real client projects.

The abstract reflects the main aspects of the diploma thesis aimed at studying modern approaches to authentication and authorization in web applications. The work includes:

The qualification work describes the significance of the development of web technologies and the emergence of front–end and back–end paradigms. It is indicated that the evolution of JavaScript has made it possible to create autonomous web applications that require new approaches to ensuring security. The main part of the work is devoted to the study and comparison of modern authentication and authorization methods, in particular token–based solutions. The OpenID Connect and OAuth 2.0 protocols, which are standards in modern web development, are described.

The practical part demonstrates the implementation of these solutions in three real projects, which proves their effectiveness and convenience.

Recommendations for using the results of the work concern the implementation of tokenized solutions in web applications with high security requirements. web applications for various purposes, including e–commerce systems, CRM systems and

educational platforms. Contributes to increasing the level of security in web applications and improving authentication and authorization processes.

The results obtained in the qualification work can be used to further improve authentication and authorization methods, including the possibility of using biometric data and improving token protection algorithms.

ЗМІСТ

ВСТУП

Актуальність теми. Зростання використання веб-додатків: Зі стрімким розвитком Інтернету та цифрових технологій веб-додатки стали основними інструментами для надання різноманітних послуг користувачам по всьому світу. Від онлайн–банкінгу до електронної комерції та хмарних сервісів – веб-додатки є невід'ємною частиною сучасного бізнесу та повсякденного життя. Однак для забезпечення безпеки користувачів та даних необхідно ефективно впроваджувати механізми автентифікації.

Проблеми безпеки та захисту даних: З кожним роком збільшується кількість кіберзлочинів і атак на веб-додатки, що ставить під загрозу конфіденційність користувачів та цілісність даних. Недосконала система автентифікації може стати вразливим місцем для таких атак. Саме тому важливо вибрати такі механізми автентифікації, які забезпечать надійний захист і мінімізують можливість зломів.

Різноманітність архітектур веб-додатків: Сучасні веб-додатки мають різну архітектуру, включаючи монолітні, мікросервісні та серверні безкоштовні додатки (serverless). Кожна з цих архітектур потребує специфічних підходів до вибору механізмів автентифікації. Наприклад, для мікросервісної архітектури може бути доцільним використання системи управління токенами (наприклад, OAuth 2.0 або JWT), в той час як для монолітних додатків потрібні більш традиційні методи аутентифікації.

Навантаження на веб-додатки: Веб-додатки з високим навантаженням (наприклад, онлайн–магазини з великою кількістю транзакцій або соціальні мережі з мільйонами користувачів) потребують ефективних механізмів автентифікації, які не будуть впливати на продуктивність системи. У таких випадках важливо вибирати механізми, здатні обробляти велику кількість запитів одночасно, зберігаючи при цьому безпеку та високу швидкість доступу.

Популярність нових технологій: В останні роки активно розвиваються нові підходи до автентифікації, такі як багатофакторна автентифікація (MFA),

біометричні методи та використання токенів (JWT, OAuth 2.0, OpenID Connect). Зважаючи на це, постійно виникає потреба в аналізі нових технологій і виборі найбільш ефективних рішень для конкретних потреб веб-додатків. Нормативні вимоги та стандарти: З розвитком технологій безпеки виникають нові вимоги до захисту персональних даних і конфіденційності користувачів, зокрема в контексті таких нормативних актів, як GDPR. Це створює додаткову актуальність для вибору відповідних механізмів автентифікації, які не тільки забезпечують безпеку, але й відповідають вимогам законодавства. Підвищення зручності користувача: У той час як безпека є головним пріоритетом, важливо також забезпечити, щоб вибраний механізм автентифікації був зручним для кінцевого користувача. Занадто складні або обтяжливі процеси автентифікації можуть знизити якість користувацького досвіду, що в свою чергу може призвести до відмови від використання веб-додатків.

Отже, вибір ефективних механізмів автентифікації для веб-додатків з різною архітектурою та навантаженням є важливою задачею, що включає в себе оптимізацію безпеки, зручності та ефективності роботи додатків, що, у свою чергу, безпосередньо впливає на їх успішність і довіру користувачів.

Мета і завдання кваліфікаційної роботи. Метою кваліфікаційної роботи є ми спершу покажемо, як ми проводили дослідження та який метод і процедуру використовували, а також обговоримо дослідницьке питання і як ми розробили пошукову стратегію. Ми розглянемо, які бази даних ми використовували та скільки результатів було отримано. Також буде обговорено, які критерії ми застосовували для виключення нерелевантних статей.

Далі ми представимо результати та резюме кожної релевантної статті з вказівкою класифікації, яку вони використовували. Після того як усі статті будуть підсумовані та проаналізовані, ми проведемо обговорення та зведемо їх у таблицю з різними класифікаціями методів аутентифікації.

Об’єктом дослідження є Механізми автентифікації (їх типи, методи, стандарти).

Предметом дослідження порівняння, оцінка та вибір ефективних механі-

змів автентифікації для веб-додатків з різною архітектурою та навантаженням, а також аналіз їх безпеки, ефективності та впливу на користувацький досвід..

Методи дослідження. У роботі використовуються статистичний аналіз, метод порівняльного аналізу, метод експертної оцінки.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Системний аналіз задачі «Вибір ефективних механізмів автентифікації для розроблення Web-додатків із різною архітектурою та навантаженням»

Об’єкт аналізу – «Механізми автентифікації (їх типи, методи, стандарти)»

Предмет аналізу – «порівняння, оцінка та вибір ефективних механізмів автентифікації для веб-додатків з різною архітектурою та навантаженням, а також аналіз їх безпеки, ефективності та впливу на користувацький досвід».

Точка зору: дослідник.

Призначення системи:

– Забезпечити ефективне управління пристроями та їх параметрами в режимі реального часу.

– Аналізувати та обробляти дані для покращення експлуатаційної ефективності і попередження можливих несправностей.

– Підвищити безпеку та операційну ефективність через автоматизацію моніторингу та управлінських функцій.

– Забезпечити масштабованість та гнучкість для інтеграції нових функцій або пристроїв.

Мета системи – є вивчення, порівняння та вибір ефективних механізмів автентифікації для сучасних веб-додатків з урахуванням різних архітектурних підходів (монолітні, мікросервісні, serverless тощо) та умов змінного навантаження. Оцінка ефективності автентифікації базуватиметься на критеріях безпеки, продуктивності, зручності для користувачів та масштабованості в умовах високого навантаження.

Виконаємо класифікацію системи.

Класифікація системи може базуватися на кількох важливих аспектах, таких як архітектура системи, типи механізмів автентифікації, вимоги до

навантаження, рівень безпеки та інші характеристики. Ось як можна класифікувати систему:

За типом архітектури

а) монолітна архітектура:

1) всі компоненти (сервери, бази даних, автентифікація) працюють як єдине ціле;

2) автентифікація зазвичай зберігається на сервері у вигляді сесій або токенів (якщо використовуються API);

3) вимоги до масштабованості можуть бути обмежені;

б) мікросервісна архітектура:

1) система складається з окремих незалежних сервісів, кожен з яких відповідає за конкретну задачу;

2) автентифікація вимагає централізованого управління, наприклад, через спеціалізований сервіс або API-менеджер (Auth0, OAuth 2.0);

3) забезпечується більша гнучкість, але складність інтеграції між сервісами може бути вищою;

в) Serverless–архітектура:

1) автентифікація зазвичай здійснюється через сторонні сервіси (наприклад, Firebase Authentication, AWS Cognito);

2) висока масштабованість, але може бути обмежена певними функціональними можливостями.

За типом механізму автентифікації

а) базова автентифікація:

1) простий механізм, де користувач вводить ім’я користувача та пароль;

2) використовується, як правило, для не дуже складних додатків або при використанні сесій;

б) токен–орієнтована автентифікація:

1) використовуються механізми, як-от JWT (JSON Web Token), для авторизації запитів без необхідності збереження сесії на сервері;

2) підходить для розподілених систем і мікросервісних архітектур;

в) OAuth 2.0:

1) стандарт, що дозволяє стороннім додаткам отримувати обмежений доступ до ресурсів користувача без надання йому пароля;

2) використовується для інтеграції з іншими сервісами (наприклад, через Facebook або Google Login);

г) OpenID Connect:

1) розширення протоколу OAuth 2.0, що забезпечує автентифікацію користувача разом із авторизацією;

2) використовується для реалізації єдиного входу (Single Sign–On, SSO);

д) багатофакторна автентифікація (MFA):

1) задіяння кількох методів автентифікації, таких як пароль + SMS, біометрія, або токен;

2) підвищує рівень безпеки;

е) біометрична автентифікація:

1) використовує фізичні характеристики користувача, такі як відбитки пальців або розпізнавання обличчя, для ідентифікації;

2) вимагає спеціалізованих пристроїв, але є дуже безпечною.

За рівнем безпеки

а) низький рівень безпеки:

1) простий механізм автентифікації (наприклад, базова автентифікація або прості токени);

2) використовується в не надто критичних додатках або сервісах;

б) середній рівень безпеки:

1) використовуються більш захищені методи, такі як OAuth 2.0, JWT або SSO;

2) підходить для додатків, які мають помірні вимоги до безпеки;

в) високий рівень безпеки:

1) використання багатофакторної автентифікації або біометричних

даних;

2) потрібно для критичних або високозахищених додатків, наприклад, банківських або фінансових.

За вимогами до навантаження

а) низьке навантаження:

1) для невеликих додатків або інтерфейсів, де кількість одночасних користувачів обмежена;

2) прості методи автентифікації, такі як сесії або базова автентифікація;

б) середнє навантаження:

1) додатки з більшою кількістю користувачів або інтеграцією з кількома сервісами;

2) токен–орієнтовані методи автентифікації (JWT, OAuth 2.0), можливе використання мікросервісної архітектури;

в) високе навантаження:

1) для високонавантажених систем з великою кількістю одночасних користувачів, де необхідно швидко та ефективно обробляти запити;

2) використовуються масштабовані рішення з використанням серверів автентифікації або сторонніх рішень (Firebase, Auth0).

За технологією збереження даних

а) сесії:

1) зберігання даних автентифікації на сервері, що може бути менш масштабованим;

2) використовується у монолітних архітектурах або для невеликих додатків;

б) зовнішні сервіси для автентифікації:

1) використання сторонніх сервісів для збереження та управління автентифікацією (Auth0, Firebase Authentication);

2) зручно для додатків, які потребують високої масштабованості та швидкої інтеграції.

За типом клієнта

а) веб-клієнт:

1) використання стандартних механізмів автентифікації для браузерних додатків (сесії, JWT, OAuth);

б) мобільний клієнт:

1) використання автентифікації, орієнтованої на мобільні платформи (Firebase Authentication, OAuth 2.0 для мобільних додатків);

в) API-клієнт:

1) рішення для автентифікації в API-системах з високими вимогами до продуктивності (OAuth, JWT).

Класифікація механізмів автентифікації для веб-додатків дозволяє систематизувати підходи залежно від різних архітектурних, безпекових та навантажувальних вимог. Вибір правильного механізму залежить від таких факторів, як тип системи (монолітна, мікросервісна, serverless), вимоги до безпеки, навантаження, а також тип клієнта.

1.2 Змістовна та формальна постановка задачі

1.2.1 Змістовна постановка задачі

Сьогодні використовується багато технік автентифікації, і для полегшення їх вивчення ми сформулювали наступне дослідницьке питання: ДП. На які класифікації поділяються різні техніки автентифікації? Це питання має на меті прояснити існуючі класифікації різних технік автентифікації, доступних на сьогодні. Отримавши відповідь на це питання, ми розширимо дослідження, щоб охопити різні методи автентифікації та їх зручність використання.

1.2.2 Формальна постановка задачі

Розробити класифікацію сучасних технік автентифікації, що дозволить чітко визначити основні підходи, їх принципи роботи та зручність використання:

а) визначити існуючі класифікації методів автентифікації;

б) проаналізувати техніки автентифікації відповідно до цих класифікацій;

в) оцінити зручність використання кожного методу та його придатність для різних сценаріїв.

1.3 Постановка задач дослідження

а) Аналіз існуючих технік автентифікації

1) вивчити та систематизувати сучасні методи автентифікації, які використовуються в інформаційних системах;

2) виявити ключові принципи та механізми роботи цих методів;

б) класифікація технік автентифікації

1) розробити класифікацію технік автентифікації на основі їх основних характеристик (наприклад, фактори автентифікації, рівень безпеки, зручність використання тощо);

2) виділити основні групи технік автентифікації та описати їх особливості;

г) оцінка зручності використання

1) дослідити зручність використання різних технік автентифікації для кінцевих користувачів;

2) визначити переваги та недоліки кожної техніки з точки зору практичної реалізації;

д) розробка рекомендацій

1) надати рекомендації щодо вибору оптимальних технік

автентифікації залежно від контексту їх застосування (корпоративні системи, персональні пристрої, веб-додатки тощо);

е) висновки та перспективи

1) підсумувати результати дослідження та визначити напрями для подальшого розвитку автентифікаційних систем;

є) мета дослідження:

1) проаналізувати сучасні техніки автентифікації, розробити їх класифікацію та оцінити їхню зручність використання для різних категорій користувачів та систем.

2 ВИБІР ТА ОБГРУНТУВАННЯ МЕТОДУ РОЗВ`ЯЗАННЯ

2.1 Односторінкова програма та традиційна веб-програма

Веб-додатки – це комп'ютерні програми, які доступні через веб-браузер і виконуються в ньому. Вони використовують мову розмітки гіпертексту (HTML) для відображення вмісту, каскадні таблиці стилів (CSS) для оформлення вмісту та JavaScript (JS) для створення інтерактивних елементів на веб-сторінках. Веб-додатки завжди розміщуються на сервері, який називається бекенд-сервісом і ідентифікується унікальним ідентифікатором ресурсу (URI). Цей сервер чекає на запити, і коли отримує їх, він надає ресурси веб-додатку запитуючому фронтенд-додатку в браузері. Після отримання запитуваного вмісту від бекенд-сервісу, браузер розбирає його та відображає веб-сторінку користувачу в браузері.

Враховуючи цей процес, веб-додатки можна поділити на дві категорії.

2.2 Традиційний веб-додаток

Традиційний веб-додаток (рисунок 2.1) характеризується тим, що бекенд-сервіс обробляє та надає кожну сторінку, яка розміщується на веб-сервері. HTML-документи забезпечують Гіперпосилання (теги a) використовуються для надання посилань на різні сторінки веб-вмісту. Кожен клік на тег a перенаправляє браузер до бекенд-сервісу, який обробляє запит, отримує та формує запитувану сторінку і надсилає її назад до веб-браузера.

Традиційний веб-додаток має кілька переваг, які перелічені нижче:

– First Contentful Paint (FCP), що визначається як час, який браузер витрачає на перше відображення веб-сторінки після її запиту з бекенд-сервісу, є низьким. Це пояснюється тим, що бекенд-сервіс надсилає лише невелику частину всього веб-додатку, що відповідає ресурсам першої сторінки;

– він забезпечує легку оптимізацію для пошукових систем (SEO), що допомагає веб-пошуковим системам індексувати сторінки, покращуючи шанси на їх включення в список результатів пошуку, коли користувач шукає будь-які терміни, що мають відношення до веб-додатку;

– він має мінімальні вимоги до браузерів для відображення веб-сторінки. Особливо коли в браузері вимкнено JavaScript, традиційний веб-додаток може забезпечити більшість основних функцій.

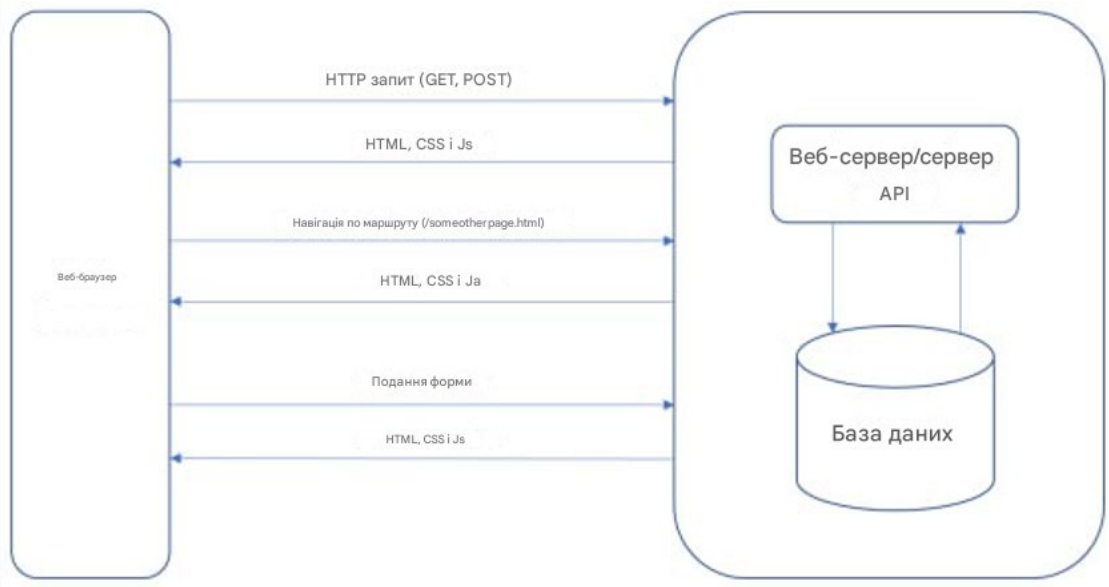

Рисунок 2.1 – Традиційна архітектура веб-додатків

Попри перелічені переваги, традиційні веб-додатки стали застарілою системою. Це був основний підхід до розробки веб-додатків. в старі часи. Він має кілька недоліків порівняно з SPA (односторінковими веб-додатками), що є основним підходом до розробки веб-додатків у сучасному світі:

– високі вимоги до мережі, що означає, що кожна сторінка вимагає запиту до бекенд-сервісу. В умовах обмеженої мережевої пропускної здатності (рисунок 2.2) це призводить до погіршення користувацького досвіду;

– більше зусиль для забезпечення безпеки, оскільки кожна сторінка веб-додатку повинна бути захищена;

– низька продуктивність в порівнянні з SPA. Час взаємодії з користувачем

також збільшується, оскільки користувач повинен щоразу чекати, поки браузер завантажить нову веб-сторінку.

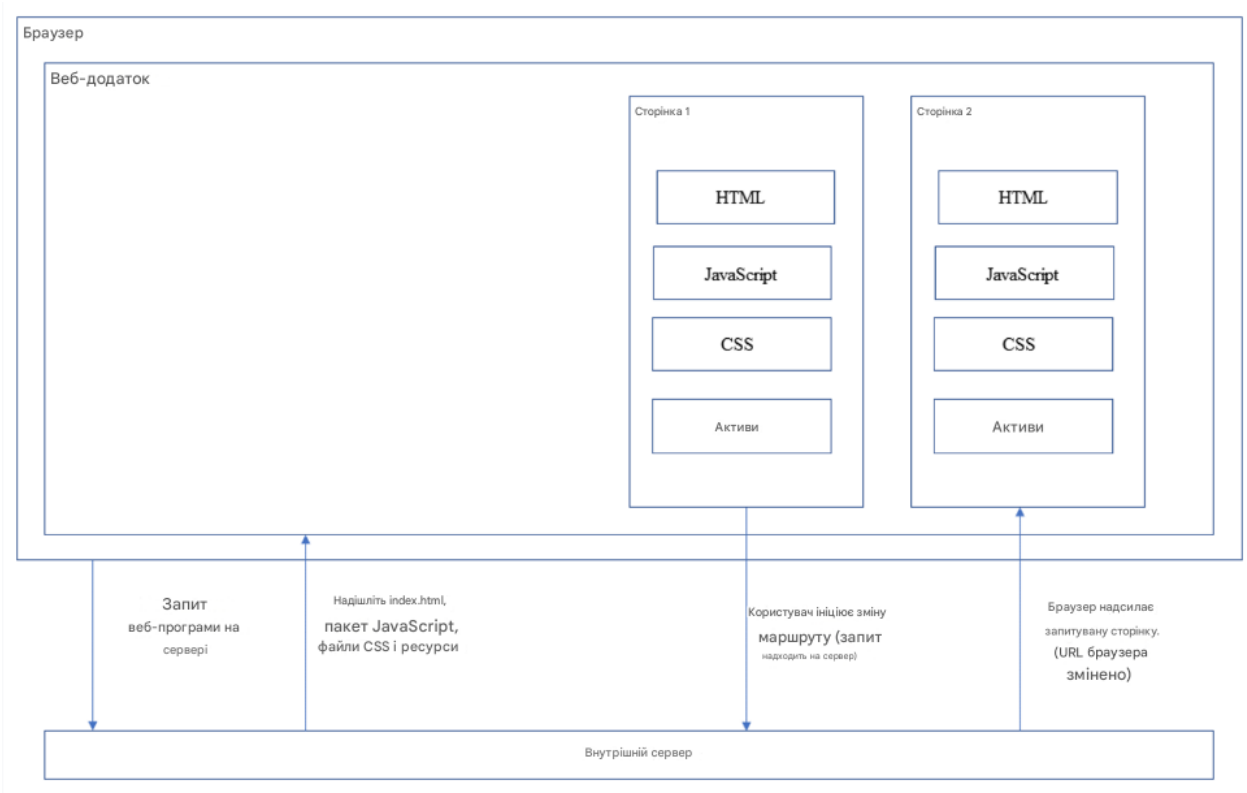

Рисунок 2.2 – Зміна маршруту традиційної веб-програми

Коли браузер робить початковий запит до бекенд-сервера, сервер відповідає сторінкою index.html та всіма скриптами, таблицями стилів і ресурсами, які з нею пов'язані. Коли користувач клікає на тег **a**, браузер ініціює новий запит до URL, вказаного в атрибуті href тега **a**. Якщо посилання абсолютне, то для запиту використовується точний URL. Якщо ж посилання відносне, браузер будує повний URL, беручи домен веб-сторінки та додаючи значення посилання до нього. Посилання для зміни сторінки зазвичай надаються як відносні URL. Коли бекенд-сервер отримує запит на зміну маршруту, він обробляє запит, формує нову сторінку і надсилає HTML–сторінку разом з усіма скриптами, таблицями стилів та іншими ресурсами, пов'язаними з цією сторінкою. Ця операція виконується для кожної сторінки, на яку користувач переходить, а також після кожної відправки форми.

2.3 Single–Page Application (SPA)

Як вказує назва, SPA (односторінковий веб-додаток) складається з однієї веб-сторінки, яка завантажується лише один раз в веб-браузері. Архітектура SPA (рисунок 2.3) слідує шаблону Model–View–Controller (MVC). View (представлення) відображає сторінки або секції, які видно користувачу. Model (модель) представляє дані, які пов'язані з View. Controller (контролер) виступає в ролі моста між Views і Models. Важливою характеристикою SPA є відображення URI на Views. Контролер співвідносить URI з конкретним View, який відображається разом з відповідними даними, коли цей запит здійснюється [1, 2].

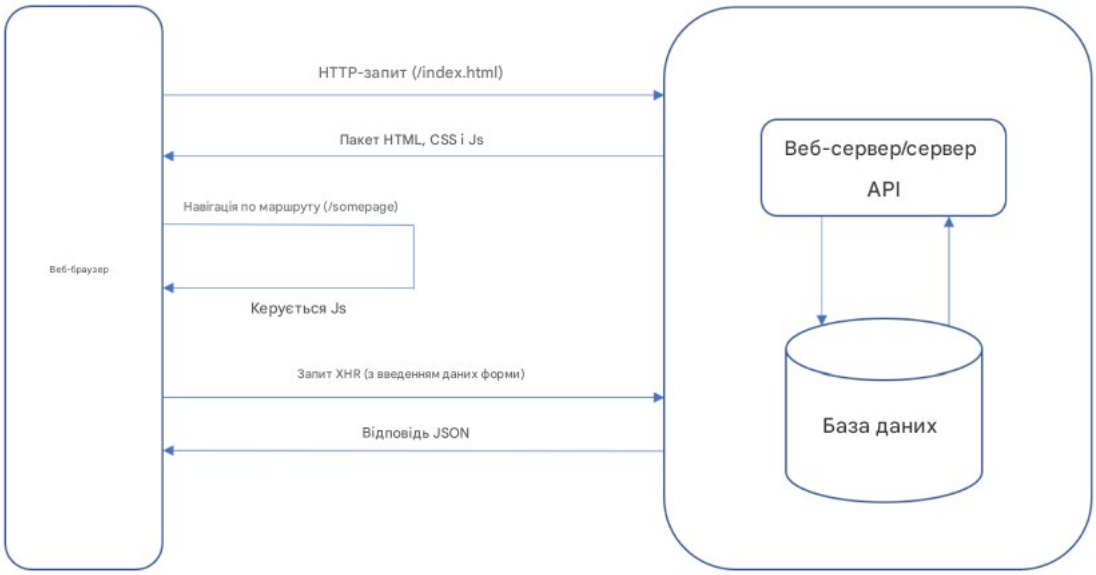

Рисунок 2.3 – Архітектура односторінкової програми

SPA має кілька переваг порівняно з традиційними веб-додатками, які перелічені нижче:

– швидша продуктивність та покращений досвід користувача. Оскільки вся веб-сторінка керується JavaScript, це забезпечує швидшу реакцію на

взаємодію користувача та швидше завантаження сторінок;

– менше навантаження на мережу. First Contentful Paint (FCP) вищий, ніж у традиційного веб-додатку, оскільки весь пакет JavaScript завантажується в браузер на початку. Однак наступні переходи між маршрутами потребують меншої взаємодії з сервером. з бекенд-сервісом. Маршрути обробляються контролером, а дані, пов'язані зі сторінкою в активному маршруті, асинхронно отримуються з бекенд-сервісу за допомогою XHR-запитів (XMLHttpRequest);

– офлайн-можливості, оскільки браузер може кешувати весь додаток. Оскільки SPA може працювати без бекенд-сервера, веб-розробники часто обирають підхід, коли вони обслуговуються через мережі доставки контенту (CDN). SPA використовують покращену здатність і підтримку браузерів для асинхронних викликів JavaScript та XML (AJAX), що дозволяє здійснювати запити до серверної частини безпосередньо з JavaScript.

Крім того, розвиток і широке впровадження REST API зробило SPA привабливим варіантом для веб-розробників при створенні веб-додатків. Безстанова природа REST API дозволяє відокремити веб-додаток від API-сервісу, що дозволяє кожному з них виконувати свої завдання незалежно. Для складних додатків SPA часто розробляються та обслуговуються разом з бекенд-сервером в межах одного домену. Бекенд-сервер хостить SPA та реалізує маршрути REST API, які використовуються для асинхронного отримання та відправки даних з SPA. У контексті розробки веб-додатків фронтенд-розробник здебільшого займається створенням клієнтської частини додатку, яку користувач може бачити. Цю частину часто називають публічним клієнтом, оскільки весь код знаходиться в браузері користувача і доступний для інспекції будь-кому. Бекенд-додаток підтримує фронтенд-сервіси, забезпечуючи доступ до ресурсів та їх управління, що не є видимим для кінцевих користувачів. Фронтенд- і бекенд-сервіси також можуть існувати на різних кінцевих точках. Термін фронтенд-сервер часто використовують для позначення сервісу, який надає публічно доступні URI через мережу для доступу до SPA. Коли користувач ініціює зміну маршруту, JavaScript-пакет обробляє запит,

завантажує запитувану сторінку і виводить її в браузер. Також оновлюється URL браузера, щоб відобразити зміну маршруту. Зміна маршруту не викликає нового запиту до сервера. JavaScript-пакет використовує дві різні стратегії для досягнення цього. Одна з стратегій відома як Hash-стратегія, де JS-пакет використовує символ хеш (#) в URL для ініціації зміни маршруту. За замовчуванням браузер не надсилає новий запит до сервера, якщо будь-яка частина рядка URL після хешу змінюється. Наприклад, зміна маршруту для завантаження сторінки панелі керування змінює URL з https://www.example.com на https://www.example.com/#/dashboard. Інша стратегія полягає у використанні новіших HTML5 history API. Використовуючи функції pushState та replaceState, JS-пакет може змінювати URL браузера без ініціації нового запиту до сервера.

2.4 Аутентифікація

Автентифікація – це процес ідентифікації сутності безпечною мовою. Ця сутність може бути користувачем, пристроєм або сервісом. Автентифікація в інформаційних системах необхідна, щоб захищені сервіси та ресурси не були доступні для когось чи чогось без належного дозволу. Простими словами, автентифікація (рисунок 2.4) відповідає на питання "Хто ви?", а авторизація – на питання "Чи дозволено вам зробити цей запит або отримати доступ до цього ресурсу?". Процес відповіді на ці питання потребує безпечної перевірки сутності, про яку йде мова.

В архітектурі веб-додатка роль автентифікації стає актуальною, коли клієнт або користувач хоче отримати доступ до API, яке надається сервером API. Нижче ми наводимо деякі з поширених механізмів автентифікації, які використовуються для веб-додатків.

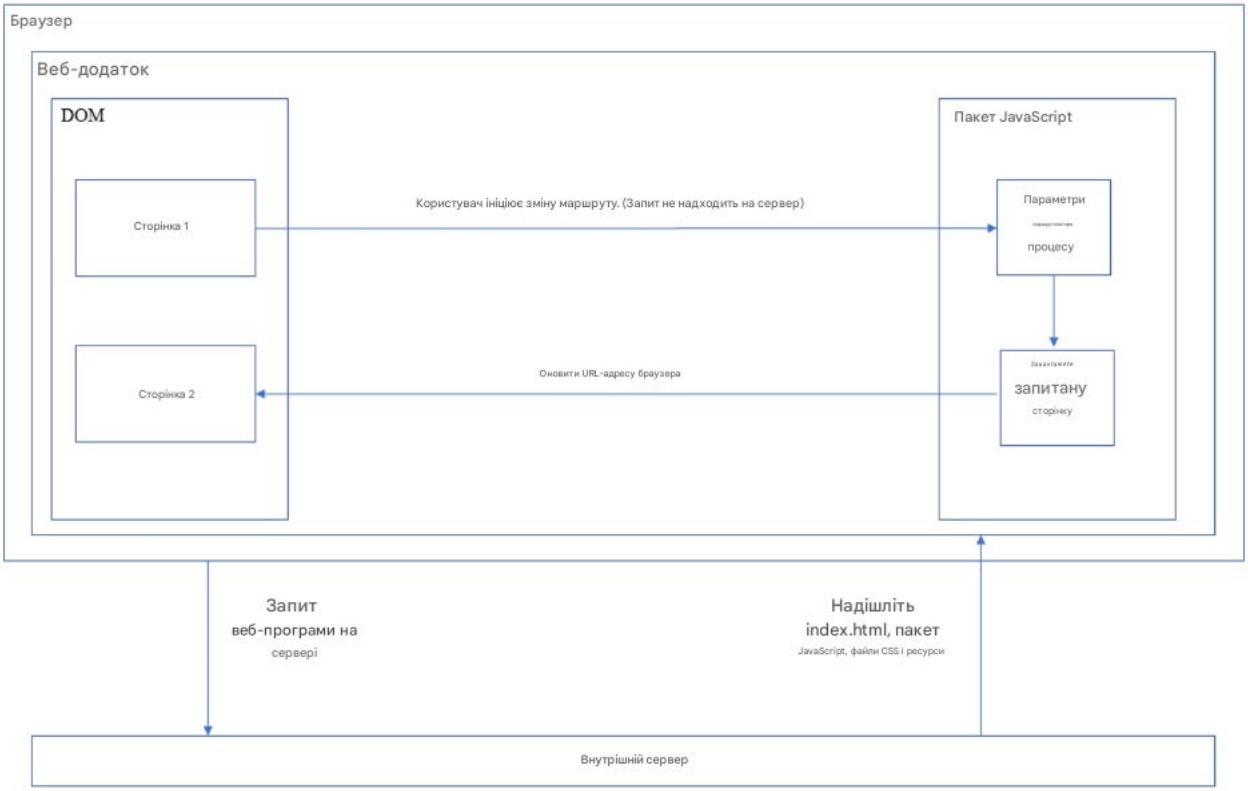

Рисунок 2.4 – Зміна маршруту односторінкової програми

2.5 Базова автентифікація HTTP

Інтернет–інженерний комітет (IETF) визначає кілька механізмів автентифікації та авторизації, які можуть використовувати веб-клієнти. Найпростіша форма автентифікації користувача полягає в передачі імені користувача та пароля, наданих користувачем, у заголовку HTTP Authorization. Це базується на механізмі виклику-відповіді, де веб-клієнт відповідає на виклик сервера при доступі до кінцевої точки. Є два методи, які можна застосувати в цій формі автентифікації.

Перший метод називається Basic scheme (основна схема), де веб-клієнт надсилає ім'я користувача та пароль, закодовані за допомогою base64, після їх з’єднання в такому форматі: username:password [3]. Другий метод називається digest scheme (схема підсумовування), де клієнт відповідає на виклик сервера у

вигляді хешованого рядка, що складається з імені користувача, пароля та унікального значення nonce сервера [4]. Спочатку для хешування використовувався алгоритм MD5, але пізніші версії специфікації дозволяють використовувати більш сильний алгоритм SHA-256. Хоча автентифікація за допомогою підсумовування пропонує кращу безпеку, ніж базова автентифікація, вона не є широко використовуваною у веб-клієнтській автентифікації.

Далі ми детальніше пояснимо, як реалізується базова автентифікація HTTP. Сервер запитує користувача надання його автентифікаційних даних. Це здійснюється за допомогою поля заголовка HTTP www–authenticate. Значення для цього поля є парами name=value, розділеними комами. Одне з значень, що вказується в цьому заголовку, – це realm. Realm визначає простір, зарезервований на сервері, в межах якого здійснюється автентифікація клієнтського запиту. Він ідентифікує область і межі, в яких захищені ресурси сервера.

Клієнт, який хоче автентифікуватися на сервері, повинен надати запитувану інформацію в полі заголовка HTTP Authorization. У цьому процесі користувач надає ім’я користувача та пароль у вигляді простого тексту для клієнтської програми. Клієнтська програма використовує ці дані для створення рядка, закодованого в base64, після об'єднання імені користувача та пароля за допомогою двокрапки між ними. Також можливе використання проксі-сервера для пересилання автентифікаційного виклику до клієнта за допомогою заголовка Proxy-Authenticate. Клієнт відповідає на цей виклик через заголовок Proxy-authorization. У заголовках запиту та відповіді схема автентифікації повинна бути вказана, в цьому випадку вона передається як Basic.

Цей механізм авторизації сам по собі не забезпечує жодних переваг у плані безпеки, тому він є вразливим до атак. Кодування пароля в base64 іноді помилково вважається механізмом шифрування, але насправді цей код можна декодувати назад, щоб отримати значення у вигляді простого тексту. Натомість він залежить від безпеки та шифрування каналу, через який передається дані.

Одним із таких способів є використання HTTPS, що є зашифрованим протоколом HTTP з використанням SSL/TLS, який став стандартом для безпечної веб-комунікації.

2.6 Ключі API

API ключі є випадковими та безпечними спільними секретами між клієнтом та API-постачальником. Клієнтами можуть бути окремі користувачі або проекти, що використовують API.

Коли API ключі використовуються для автентифікації користувачів, API сервер генерує унікальний секрет для кожного користувача. У цьому контексті API сервер є тим самим, що й бекенд сервер. Він відповідає за генерацію API ключів, їх зберігання в базі даних та перевірку кожного вхідного запиту API на наявність правильного API ключа. API ключ має бути випадковим, щоб його не можна було вгадати. Хорошим підходом є використання UUID (Універсальний унікальний ідентифікатор користувача), який є випадковим рядком, що генерується застосунком для кожного зареєстрованого користувача. Щоб уникнути підробки ключа, його можна за бажанням підписати спільним секретним ключем.

API сервер надсилає згенерований API ключ користувачеві через HTTP редиректпісля первинної автентифікації або реєстрації користувача. Залежно від реалізації, API ключі можуть бути передані клієнту як cookie або як параметр редиректу. Якщо API ключ передається як cookie, секрет зберігається в браузері користувача для домену, що відповідає API серверу. Кожен наступний запит до API серверу буде включати cookie в параметрах запиту, і він використовується API сервером для автентифікації користувача. Якщо ключ передається як параметр редиректу, клієнтська програма повинна зберігати API ключ у сховищі браузера. З клієнтської програми фронтенду API ключ передається разом з кожним запитом до веб-API.

API ключі можуть бути передані як параметр запиту в URL запиту HTTP GET або як заголовок x-API-key в запиті HTTP POST:

GET https://api.example.com?api_key=<API-KEY>

Або

GET https://api.example.com

x-API-key: <API-KEY>

Хоча API ключі забезпечують простий спосіб автентифікації клієнтів і підтримки їхніх сесій, вони вважаються відносно ненадійними. Оскільки це спільний секрет, витік ключа може дозволити зловмисникам видавати себе за зареєстрованих користувачів. Альтернативно, API ключі зазвичай використовуються для ідентифікації проєктів, а не окремих користувачів. У цьому випадку між клієнтськими пристроями та API сервером існує фронтенд-сервіс, який виступає в ролі проксі між ними. Оскільки фронтенд сервер не доступний користувачам, він може безпечно зберігати API ключ. Він додає API ключ до кожного переданого запиту до API сервера і передає відповідь назад клієнту. Фронтенд сервер також підтримує власну реалізацію автентифікації та ідентифікації окремих користувачів.

Коли API сервери роздають API ключі веб-розробникам, розробник має зареєструвати застосунок, після чого йому надається унікальний API ключ. Цей API ключ відображається з повідомленням, що він буде показаний тільки один раз, і розробник має зберегти його в безпеці. Поширеною помилкою веб-розробників є вбудовування API ключа безпосередньо в код на фронтенд сервері. Це призводить до того, що API ключ стає доступним для публіки, якщо код буде опублікований у відкритому репозиторії. Краще використовувати змінні середовища для зберігання API ключів та використовувати ці змінні в коді. Одним із найпопулярніших підходів для розміщення веб-застосунків є використання платформи як послуги (PaaS) від хмарного провайдера. Більшість постачальників PaaS мають вбудовану конфігурацію змінних середовища, яка дозволяє розробнику задавати API ключі безпосередньо в панелі управління хмарного провайдера. Природно, ще більшою помилкою буде вбудовувати API

ключі прямо в код.

Вбудовування API ключа застосунку в JavaScript на стороні клієнта або інші способи його відкриття для веб-браузера є серйозною помилкою, оскільки це може призвести до того, що ключ стане доступним для публіки. Якщо код опублікований у відкритому доступі, наприклад, в репозиторіях на GitHub або інших сховищах, зловмисники можуть отримати доступ до цього ключа і використовувати його для зловмисних цілей. Це робить систему вразливою до атак, тому важливо зберігати API ключі в безпечних середовищах, таких як змінні середовища або спеціалізовані конфігураційні налаштування на сервері.

2.7 Аутентифікація на основі токенів

Токенна автентифікація стала популярним і ефективним рішенням останнім часом, особливо в сучасних веб-застосунках, які вимагають масштабованості та здатності працювати з різноманітними клієнтами, такими як мобільні додатки та сторонні сервіси. На відміну від станових систем автентифікації, які вимагають від сервера зберігати дані сесії для кожного користувача, токенна автентифікація дозволяє серверу працювати в безстанному режимі, тобто йому не потрібно зберігати жодної сесійної або стану клієнта.

а) Огляд потоку токенної автентифікації;

б) перенаправлення до постачальника ідентифікації (IdP):

1) клієнтський застосунок перенаправляє користувача до окремого Постачальника Ідентифікації (IdP) для автентифікації. IdP зазвичай відповідає за управління процесом автентифікації, таким як перевірка особистості користувача через облікові дані (ім'я користувача/пароль) або додаткові фактори, наприклад, одноразовий пароль (OTP), надісланий на мобільний пристрій користувача;

в) автентифікація користувача (рисунок 2.5):

1) dP перевіряє особистість користувача. Це може включати базову автентифікацію (ім'я користувача та пароль) або багатофакторну автентифікацію (наприклад, OTP, надісланий на мобільний пристрій);

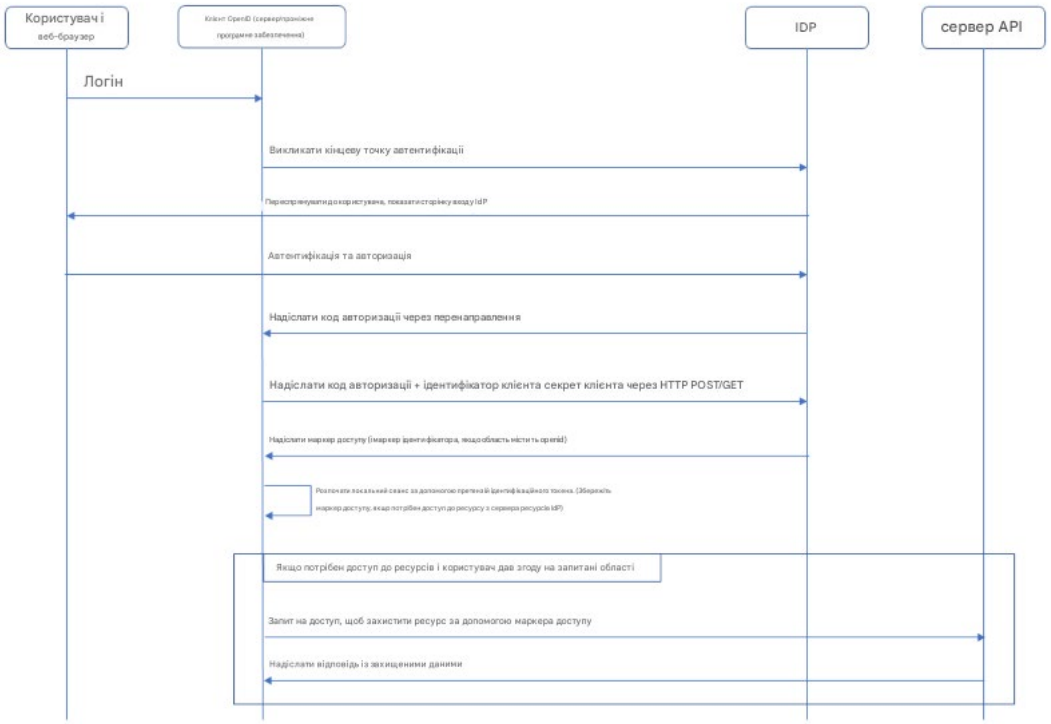

Рисунок 2.5 – Потік коду авторизації

г) видача токенів:

1) після успішної автентифікації користувача, IdP видає два основних типи токенів;

д) зберігання токенів:

1) клієнтський застосунок зберігає access токен (зазвичай у сховищі браузера, як localStorage або sessionStorage, або в безпечному cookie) і використовує його для автентифікації наступних запитів до захищених ресурсів;

е) доступ до захищених ресурсів:

1) при здійсненні запитів до бекенд API, клієнтський застосунок включає access токен у заголовок HTTP, зазвичай у заголовку Authorization.

Токенна автентифікація забезпечує зручність і безпеку, дозволяючи

зберігати автентифікаційний стан на стороні клієнта, а також зменшує навантаження на сервер, оскільки не потрібно зберігати стан сесії для кожного користувача.

Клієнт OpenID повинен надати токен доступу, щоб отримати інформацію з цього кінцевого пункту. ID токени є корисними, оскільки вони відображають поточний стан входу користувача через кілька додаткових заяв (claims), таких як ідентифікатор видавця, ідентифікатор суб'єкта, аудиторія, для якої призначений ID токен, час його дії та час автентифікації кінцевого користувача. Іншим важливим параметром запиту автентифікації є response_type, який визначає, які токени будуть повернуті. Значення response_type має містити id_token, якщо потрібен ID токен.

OpenID Connect (OIDC) визначає два підходи залежно від типу застосунку. У архітектурі, де більшість логіки застосунку обробляється на сервері, таких як традиційний веб-застосунок або односторінковий застосунок (SPA), що підтримується бекенд–мідваром, OIDC визначає потік авторизації через код. У цьому підході більшість логіки автентифікації обробляється бекенд–мідваром або сервером, який є клієнтом OIDC.

У загальних рисах, клієнт OIDC отримує код авторизації у відповідь на запит автентифікації, надісланий до кінцевого пункту автентифікації IdP. Потім клієнт OIDC використовує цей код авторизації, щоб отримати ID токен і токен доступу з кінцевого пункту для токенів. Цей потік визначений в специфікації OAuth 2.0. Авторизаційний сервер завжди повинен повертати токен доступу. OIDC базується на цьому, вимагаючи, щоб авторизаційний сервер повернув ID токен разом з токеном доступу.

Залежно від архітектури застосунку, потреба в токені доступу може змінюватися. Наприклад, для застосунку, який використовує авторизаційний сервер лише для підтвердження особи кінцевого користувача, заяви (claims), що містяться в ID токені, будуть достатніми. Такий застосунок може не потребувати токен доступу, оскільки він не має доступу до ресурсів користувача з ресурсного сервера. Токен доступу все одно буде видано, якщо

використовується потік авторизації через код, і він може бути використаний лише для тих областей (scopes), які були надані. У цьому випадку токен доступу можна використати для доступу до кінцевого пункту /userinfo, щоб отримати інформацію про ті самі заяви, що містяться в ID токені.

Потік авторизації через код детально описується наступними кроками [7]:

– Клієнт ініціює автентифікацію з IdP за допомогою взаємодії з користувачем, наприклад, шляхом натискання кнопки.

– Клієнт викликає кінцеву точку /authentication на IdP. Специфікації OIDC визначають, що клієнт повинен направити користувача на сторінку автентифікації на сервері авторизації (IdP). Запит на /authentication містить необхідні параметри, такі як client_id, redirect_uri, scope, response_type (зазначений як code), та інші необхідні параметри.

– Користувач надає свої облікові дані на сторінці авторизації, яку надає IdP (зазвичай це сторінка входу з полями для введення логіна та пароля).

– IdP перевіряє автентичність користувача. Якщо облікові дані вірні, сервер авторизації може додатково застосувати двофакторну автентифікацію або інші методи для забезпечення безпеки.

– IdP повертає авторизаційний код. Після успішної автентифікації IdP перенаправляє користувача на redirect_uri, зазначений у початковому запиті клієнта. У рядку запиту цього URL передається authorization code, який є короткочасним кодом для отримання токенів.

– Клієнт отримує авторизаційний код. Клієнт отримує цей код на стороні браузера або мобільного застосунку через редирект.

– Клієнт відправляє авторизаційний код на кінцеву точку /token. Клієнт відправляє POST-запит до кінцевої точки /token на сервері авторизації. У цьому запиті містяться наступні параметри:

– Сервер авторизації перевіряє запит. Він перевіряє правильність кодів, redirect_uri та інші дані. Якщо все вірно, сервер авторизації генерує два токени: ID токен та access token.

– Клієнт отримує токени. У разі успішної перевірки сервер повертає у

відповіді два токени:

– Клієнт використовує токен доступу для доступу до захищених ресурсів. Тепер клієнт може використовувати access token для доступу до API або інших ресурсів, які захищені на сервері. Токен доступу додається до HTTP-запитів, наприклад, як заголовок Authorization: Bearer <Access_Token>.

Цей потік дозволяє клієнту отримати доступ до ресурсів, не зберігаючи стан автентифікації на сервері, і зберігаючи безпеку через використання короткочасних кодів та токенів.

Ключовим етапом цього потоку є автентифікація клієнта через кінцеву точку токену. OIDC визначає кілька методів автентифікації клієнта, які повинні підтримуватися IdP. У кожному з них клієнт відправляє різну інформацію до IdP в полі client_secret, яке згадувалося раніше. Ці методи детально описуються нижче [7]:

– client_secret_basic. У цьому методі клієнт використовує механізм HTTP Basic Authentication для автентифікації на кінцевій точці токену IdP. Клієнт повинен відправити client_secret, який він отримав від IdP під час реєстрації. Це значення передається в заголовку авторизації HTTP запиту за форматом:

Authorization: Basic <Base64(client_id:client_secret)>

Цей метод є простим і широко використовуваним;

– client_secret_post. Цей метод схожий на client_secret_basic, але замість того, щоб передавати client_id та client_secret в заголовку запиту, ці параметри передаються як частина тіла HTTP POST запиту. Це може виглядати так:

POST https://idp.example.com/token

Content–Type: application/x–www–form–urlencoded

client_id=<client_id>&client_secret=<client_secret>&code=<authorization_code>&redirect_uri=<redirect_uri>

Цей метод також широко підтримується, але він може бути менш безпечним порівняно з попереднім, оскільки параметри передаються в тілі запиту;

– client_secret_jwt. У цьому методі клієнт створює Hashed Message

Authentication Code (HMAC) за допомогою алгоритму, такого як HMAC SHA-256. HMAC створюється з використанням client_secret, який отримує клієнт від IdP. Клієнт також має включити claims у JWT (JSON Web Token).

Токен у вигляді JWT є більш захищеним варіантом, оскільки він включає в себе підпис та додаткову інформацію, яка допомагає перевірити цілісність і автентичність запиту.

Порівняння методів:

– client_secret_basic є найпростішим методом, де секрет передається через стандартний механізм HTTP Basic Authentication, що робить цей метод легко інтегрованим, але він менш безпечний, якщо канал зв'язку не зашифрований;

– client_secret_post додає гнучкість, але також може бути менш безпечним, оскільки передача секретів у тілі запиту може бути схильна до атак, якщо канал зв'язку не захищений;

– client_secret_jwt є найбільш безпечним методом, оскільки забезпечує криптографічну перевірку автентичності за допомогою HMAC і JWT, що робить цей метод ідеальним для сучасних додатків з підвищеними вимогами до безпеки.

Ці методи забезпечують різні рівні безпеки, і вибір між ними залежить від конкретних вимог до безпеки та зручності інтеграції.

OpenID Connect також визначає Implicit flow, який спеціально орієнтований на веб-додатки, що працюють у браузері, такі як односторінкові додатки (SPA). У цьому потоці ID token безпосередньо надсилається клієнту від Identity Provider (IdP) без попереднього отримання авторизаційного коду. response_type у цьому потоці має бути id_token. Якщо разом з ID токеном потрібен доступ до access token, тоді response_type має бути id_token token.

Оскільки HTTP є безстаневим протоколом, IdP має надіслати токени назад до веб-додатку через HTTP редирект. Це викриває ID та access token через історію браузера, що може бути піддано спостереженню зловмисними програмами, наприклад, плагінами у браузері. Самі ID tokens не використовуються для доступу до захищених ресурсів або для автентифікації.

Вони лише підтверджують статус автентифікації кінцевого користувача і містять інформацію про нього. Тому ризик витоку ID токенів є меншим, ніж ризик витоку access token, хоча такий ризик все одно існує.

У зв'язку з цим Implicit flow не рекомендується, якщо клієнт потребує access token разом з ID token. Цей ризик також присутній при використанні Authorization code flow в браузерних додатках, оскільки client secret не можна надійно зберігати в браузері – код і дані браузера доступні кожному користувачеві.

Щоб уникнути цих ризиків, IETF (Internet Engineering Task Force) запропонувала Authorization code flow разом з Proof Key for Code Exchange (PKCE) для браузерних і нативних додатків, які не можуть безпечно отримати та зберігати client secret [11]. PKCE виконує роль client secret у Authorization code flow, забезпечуючи додатковий рівень безпеки для обміну коду на токени.

Як працює PKCE (рисунок 2.6) в Authorization Code Flow:

– PKCE був розроблений для покращення безпеки в сценаріях, коли клієнт не може безпечно зберігати client secret, як це відбувається в браузерних та мобільних додатках;

– клієнт генерує code verifier (довільно згенеровану строку) і використовує її для створення code challenge (відповідної хешованої версії);

– під час запиту на авторизацію клієнт передає code challenge та вказує метод хешування (наприклад, SHA-256);

– після того, як клієнт отримає авторизаційний код від IdP, він надсилає цей код разом з code verifier на кінцеву точку токену;

– IdP перевіряє code verifier з отриманим code challenge і, якщо вони збігаються, надсилає ID token і access token клієнту.

Цей процес гарантує, що лише той клієнт, який ініціював запит на авторизацію, може отримати доступ до токенів, навіть якщо його запит було перехоплено, тому що хешоване значення code verifier можна обчислити лише на стороні клієнта.

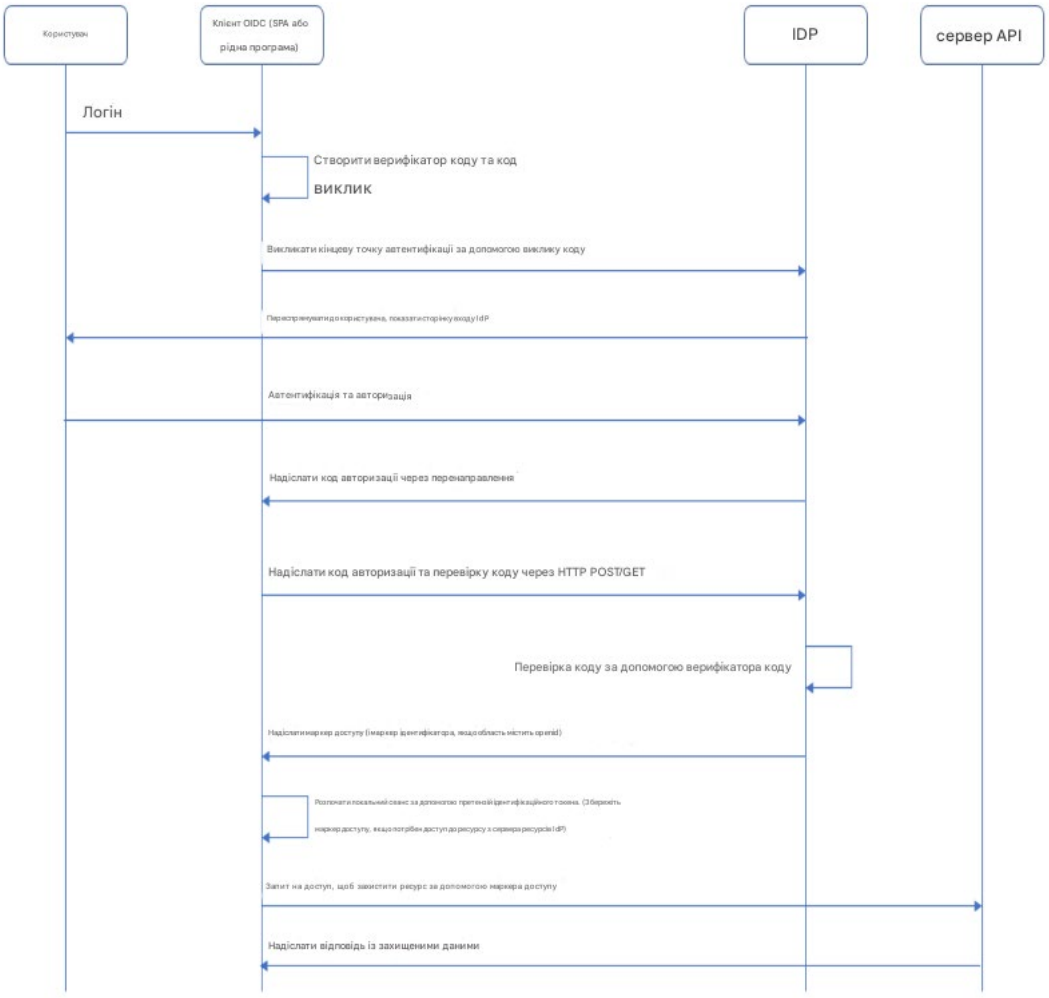

Рисунок 2.6 – Потік коду авторизації з PKCE

Переваги використання PKCE:

– підвищує безпеку при використанні Authorization Code Flow, особливо в середовищах, де клієнт не може зберігати client secret безпечно (наприклад, браузер або мобільні додатки);

– використання code challenge та code verifier забезпечує, що навіть якщо авторизаційний код буде перехоплений, його не можна буде використовувати без відповідного code verifier, який відомий лише клієнту.

Цей механізм дозволяє знизити ризики, пов'язані з витоком або компрометацією client secret у клієнтських додатках, і є стандартом для сучасних веб-додатків.

Частина специфікації OAuth 2.0, яка, як розширення, застосовується до

специфікації OIDC. Якщо початковий запит на автентифікацію не містить значення openid в параметрі scope, то повертається лише код доступу. Однак, якщо значення openid додається до параметра scope, то повертається ID токен разом з токеном доступу. Щоб реалізувати автентифікацію за допомогою цього підходу, фронт-енд додаток має виконати наступні кроки:

– застосунок ініціює автентифікацію з IdP за допомогою дії користувача, наприклад, натискання кнопки;

– застосунок генерує два артефакти: code_verifier та code_challenge. code_verifier – це випадковий рядок довжиною 128 біт, що складається з алфавітно–цифрових символів і кількох спеціальних символів. code_challenge генерується з code_verifier шляхом його шифрування за допомогою криптографічного хеш–алгоритму, зазвичай SHA256, і кодується у формат base64URL;

– застосунок відправляє code_challenge разом із redirect_uri до серверу автентифікації в запиті на автентифікацію через HTTP редірект. Параметр scope має містити значення openid, якщо потрібно отримати ID токен. Параметр response_type має бути встановлений як code;

– сервер автентифікації виконує автентифікацію користувача, зазвичай через ім’я користувача та пароль. Після автентифікації сервер запитує у користувача згоду на надання доступу до ресурсів користувача запитуючому клієнтському застосунку. Після отримання згоди сервер авторизації зберігає code_challenge та відправляє криптографічно згенерований випадковий авторизаційний код клієнтському застосунку через redirect_uri;

– клієнтський застосунок отримує авторизаційний код і відправляє POST–запит до серверу авторизації, запитуючи ID токен та токен доступу. У цьому запиті клієнт має надіслати code_verifier та авторизаційний код;

– сервер перевіряє code_verifier з code_challenge, який він зберіг у попередньому запиті. Додатково він перевіряє авторизаційний код. Якщо перевірка успішна, сервер відповідає на POST-запит ID токеном та токеном доступу, а також видаляє code_verifier як використаний;

– клієнтський застосунок розбирає ID токен і створює персоналізовану панель управління з інформацією про претензії. Він зберігає токен доступу в своєму внутрішньому сховищі для використання під час майбутніх запитів до ресурсів.

2.8 Session Management

ID токен і токен доступу, які видаються провайдером ідентифікації, часто мають обмежений термін дії через міркування безпеки. Це створює ситуацію, коли клієнт повинен запитувати нові токени у авторизаційному сервері після того, як поточні токени стануть недійсними. Клієнт OIDC повинен керувати сесіями для обох токенів – ID токеном і токеном доступу. Токени можуть мати різний час дії. Важливо розрізняти та ідентифікувати призначення двох токенів для сценаріїв використання додатка.

Аудиторія ID токену – це клієнт OIDC, а аудиторія токену доступу – це сервер ресурсів. ID токен надає клієнту інформацію про особу користувача та його статус аутентифікації. Ця інформація надається у полі claims в тілі ID токену. Цей токен використовується для генерації персоналізованого профілю з полями претензій.

Токени доступу (рисунок 2.7), в свою чергу, використовуються клієнтом для доступу до ресурсів від імені користувача. Специфікація OIDC визначає сесії користувача як безперервний період часу, протягом якого клієнт OIDC має дійсний аутентифікований статус з IdP. Управління сесією користувача стосується перевірки цього аутентифікованого статусу. Якщо сесія закінчується, усі існуючі токени стають недійсними, і нові токени видаються після того, як користувач Користувач повинен повторно пройти аутентифікацію. Це стосується також токену доступу, навіть якщо токен доступу все ще є дійсним відповідно до часу його дії.

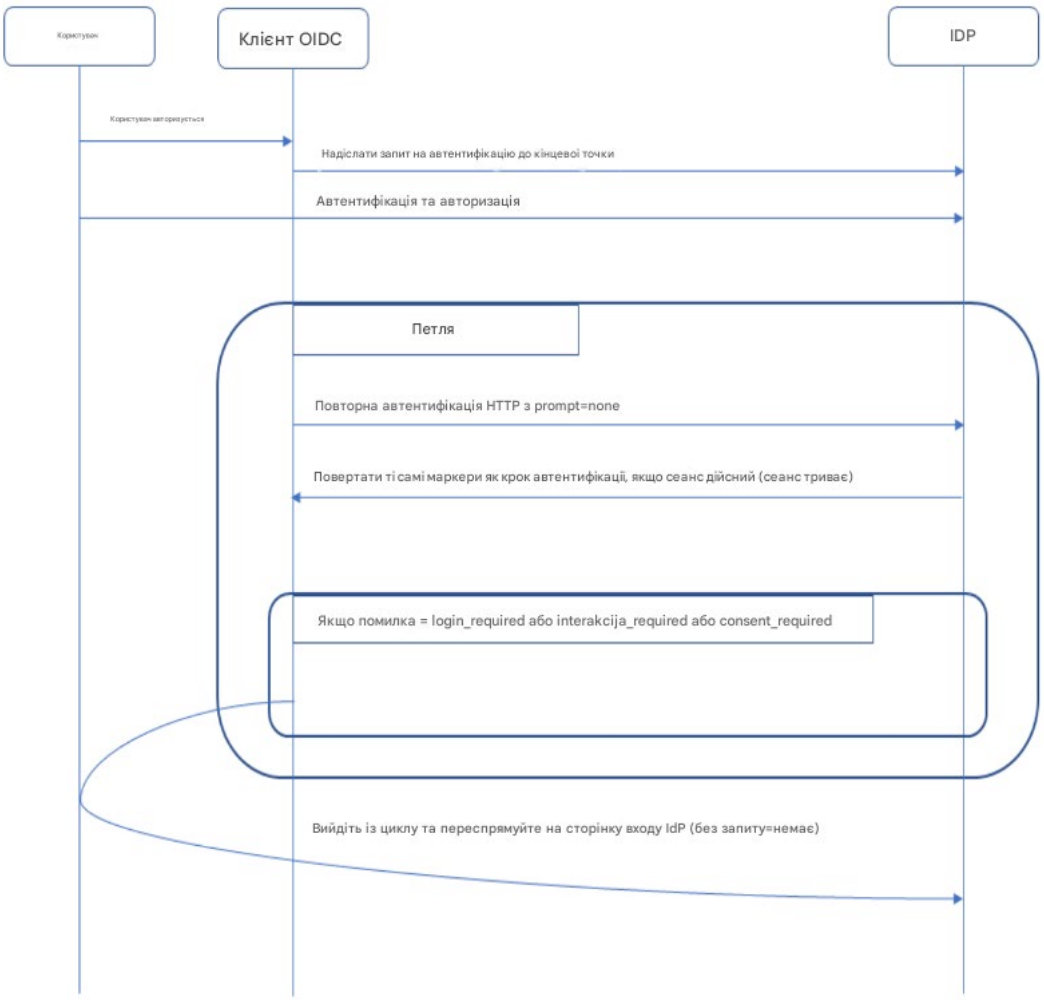

Рисунок 2.7 – Керування сеансами за допомогою
тихої повторної автентифікації.

Існують два основних підходи до підтримки сесії. Один із них називається тихою повторною аутентифікацією [7]. Веб-додаток дотримується звичайного процесу аутентифікації, надаючи ті самі параметри до кінцевої точки авторизації IdP. Цей підхід вимагає від клієнта періодично запитувати в провайдера OpenID статус сесії кінцевого користувача. Наприклад, у OpenID Connect параметр prompt=none в запиті вказує провайдеру ідентифікації, що аутентифікація не повинна вимагати взаємодії з кінцевим користувачем. Якщо сесія в IdP все ще є дійсною, IdP відповідає з запитуваними токенами.

Крім того, OIDC визначає коди помилок у відповіді, які IdP повинен включити в параметр error_description, якщо тихе підтвердження аутентифікації

не вдалося. Ці коди помилок включають login_required, interaction_required та consent_required.

login_required – Цей код надсилається, коли користувач не увійшов до системи з IdP.

interaction_required – Цей код надсилається, коли користувач увійшов до системи з IdP та авторизував додаток в IdP, але IdP потрібно виконати додаткову взаємодію з користувачем перед оновленням ID токена.

consent_required – Цей код надсилається, коли користувач увійшов до системи з IdP, але IdP вимагає згоду кінцевого користувача перед тим, як надати додатку доступ до ідентичності користувача та ресурсів.

Клієнт OIDC повинен реагувати на вищезгадані відповіді, перенаправляючи до авторизаційної точки входу IdP без параметра prompt=none, оскільки IdP вимагає взаємодії з користувачем для завершення аутентифікації.

Другий підхід полягає у використанні прихованих iframe, що підключаються до IdP, який регулярно опитує веб-клієнт для отримання статусу сесії [12]. Це здійснюється через міждоменною комунікацію між клієнтом OIDC та IdP. Клієнт OIDC завантажує невидимий iframe з URL–адресою, наданою IdP. Це дозволяє IdP працювати в своєму власному контексті безпеки всередині клієнта OIDC. Клієнт OIDC використовує API Window.postMessage для надсилання повідомлень до iframe IdP, який відповідає за допомогою того ж API. Повідомлення в кожному iframe отримуються за допомогою слухача подій message, який спрацьовує, коли iframe отримує повідомлення через API postMessage.

OIDC визначає два метаданих–ендпоінти у вигляді URL–адрес, які IdP повинен реалізувати для підтримки цієї функціональності. Перший URL, check_session_iframe, надається як джерело для невидимого iframe, що завантажує сторінку, реалізовану IdP, в iframe (рисунок 2.8). Другий URL, end_session_endpoint, надається IdP, щоб клієнт OIDC міг перенаправити на нього, коли хоче завершити сесію, наприклад, коли користувач хоче вийти.

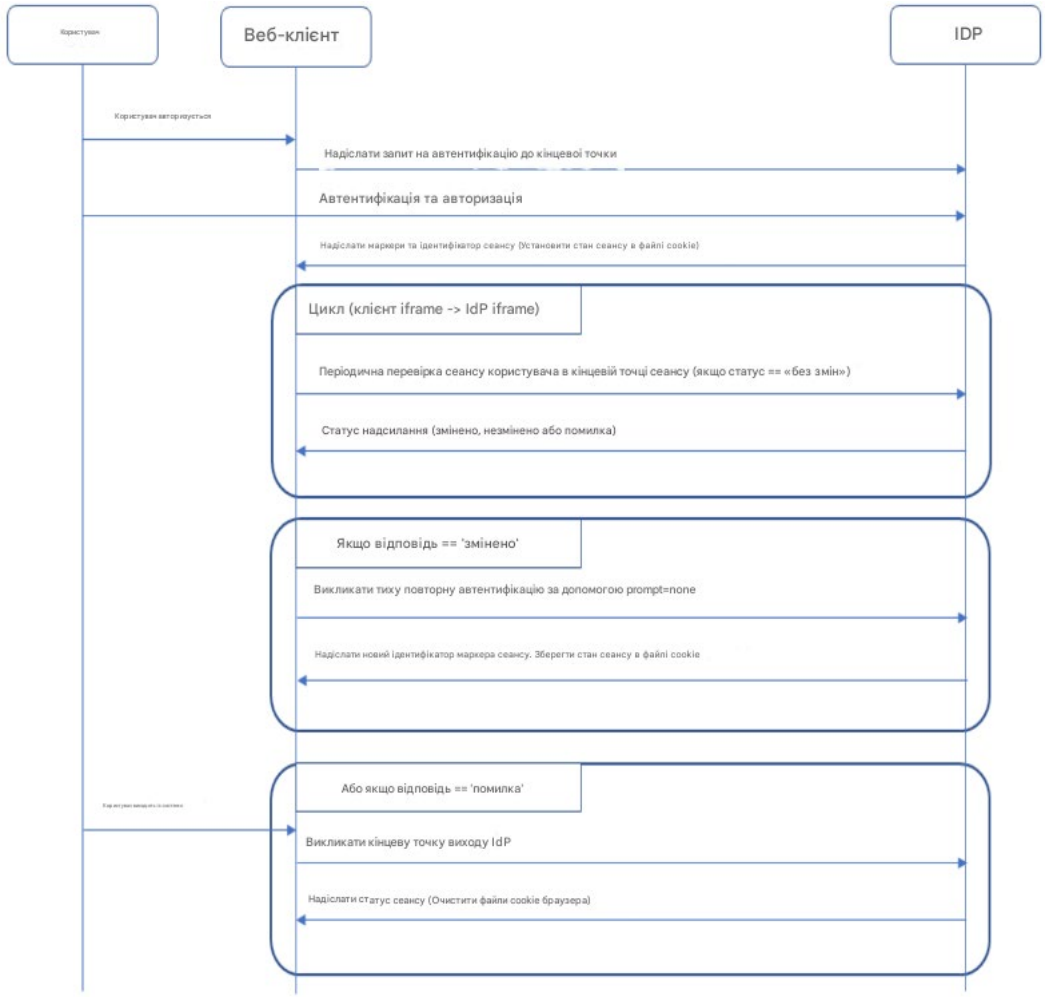

Рисунок 2.8 – Керування сеансами за допомогою прихованих iframe

Для відстеження стану сесії OIDC визначає параметр, званий session_state. Він обчислюється з client_id, origin URL, стану браузера і солі, які потім криптографічно хешуються. Він обчислюється IdP і надсилається клієнту OIDC разом із початковою відповіддю аутентифікації. Стан браузера підтримується IdP і зберігається у вигляді cookies або в локальному сховищі браузера. Це залежить від реалізації IdP, яке оновлюється під час значущих подій, таких як вхід, вихід тощо. Не рекомендується, щоб IdP часто змінював стан браузера, оскільки це призведе до зміни session_state і додаткового трафіку в мережі для відображення цієї зміни у клієнті OIDC. Стан браузера зберігається в домені IdP, тому він недоступний поза його доменом.

Це через сторінку, яка виконується в невидимому iframe під власним

доменом. Origin URL посилається на початкову URL–адресу відповіді аутентифікації. Зазвичай це є посиланням на URL–адресу домену IdP. [12]

Клієнт OIDC завантажує невидимий iframe з URL–адресою зі свого власного домену для комунікації з iframe IdP. Цей iframe надсилає client_id та session_state з кожним запитом перевірки сесії, що надсилається до iframe IdP. Клієнт OIDC iframe повинен знати ідентифікатор iframe IdP для того, щоб надіслати запит. IdP використовує client_id, надісланий клієнтом OIDC, origin URL IdP та стан браузера, збережений у браузері, для перерахунку session state. Потім він порівнює його з session state, переданим клієнтом OIDC. Залежно від результату порівняння, IdP відповідає однією з трьох можливих значень статусу:

– changed (змінено) – Якщо два стани сесій не співпадають, IdP надсилає цю відповідь. Отримавши цю відповідь, клієнт OIDC має ініціювати silent re–authentication з IdP, використовуючи параметр prompt=none. Клієнт OIDC повинен надіслати старий ID token як параметр id_token_hint у запиті. Якщо IdP відповість ID token для того ж користувача, клієнт OIDC зберігає ID token та оновлює значення session state, отримане разом з відповіддю. Якщо клієнт OIDC не отримує ID token або отримує ID token для іншого користувача, це означає кінець сесії, і клієнт OIDC повинен обробити це як вихід (logout) для поточного користувача;

– unchanged (не змінено) – Якщо два стани сесій співпадають, IdP відповідає цією відповіддю. У цьому випадку клієнт OIDC не потребує додаткових дій, оскільки це означає, що сесія користувача все ще є дійсною;

– error (помилка) – Якщо IdP не може розрахувати session state з параметрів, отриманих через запит клієнта OIDC, він відповідає цією відповіддю. Веб-застосунок повинен обробити цей випадок як вихід (logout) для сесії користувача. Після отримання цієї помилки, OIDC вимагає, щоб клієнт OIDC не ...не виконувати повторну аутентифікацію з параметром prompt=none, щоб уникнути потенційних нескінченних запитів до IdP;

– щоб завершити сесію з IdP під час виходу користувача, клієнт OIDC

перенаправляє браузер на URL end_session_endpoint. Клієнт OIDC повинен передати поточний ID token як параметр id_token_hint. Додатково, клієнт OIDC може також передати URL для перенаправлення після виходу як post_logout_redirect_uri. Якщо цей параметр надано, IdP перенаправить браузер на цей URL після того, як користувача буде виведено з системи. URL повинен бути зареєстрований у IdP перед його використанням;

– специфікація OAuth 2.0 ввела refresh tokens, які можна використовувати для запиту нового access token від IdP після того, як поточний токен стане недійсним [8]. Оскільки специфікація OpenID Connect була розроблена на основі OAuth 2.0, вона ввела параметр openid scope, який необхідно передавати, якщо разом з access token потрібен ID token. Цей параметр scope можна передавати як у початковому запиті на аутентифікацію, так і при використанні refresh token. Refresh tokens зазвичай мають тривалий термін дії, що дозволяє веб-застосунку підтримувати активну сесію без необхідності повторної аутентифікації та авторизації користувача;

– оскільки refresh tokens можуть використовуватись для безпосереднього отримання access tokens, дуже важливо їх безпечно зберігати. В публічному клієнті, такому як односторінковий веб-застосунок (SPA), refresh token може зберігатися в localStorage браузера або в cookies. Крім того, refresh tokens надаються для одноразового використання, що означає, що разом з access token авторизаційний сервер також надсилає новий refresh token. Це все ще не запобігає доступу до refresh token зловмисним програмам у браузері, особливо якщо вони зберігаються в localStorage;

– щоб підтримувати сесію користувача, веб-застосунок повинен запитувати нові токени до того, як поточні стануть недійсними. Всі токени є непрозорими для веб-застосунку, що означає, що веб-застосунок не може прочитати access token та отримати інформацію про його термін дії. Часто авторизаційні сервери налаштовуються на відправлення параметра expires_in, який містить час, через який токени стануть недійсними. Веб-застосунок має зчитати та зберегти це значення і реалізувати автоматичний запит заздалегідь,

до того як токени стануть недійсними.

2.9 Контроль доступу

Контроль доступу в SPA (Single Page Application).

У традиційних веб-додатках контроль доступу зазвичай реалізується на бекенді, де кожен маршрут перевіряється на наявність прав доступу користувача до цього маршруту. Однак у SPA, де вся програма працює в браузері користувача, впровадження стратегії контролю доступу на фронтенді стає важливою задачею.

З точки зору безпеки, контроль доступу в браузерних додатках має на меті спрощення і покращення досвіду користувача при роботі з додатком. Оскільки браузерні додатки можуть бути перевірені і змінені, перевірки доступу мають проводитися на бекенді. Проте, впровадження додаткових перевірок доступу на фронтенді має кілька переваг, як зазначено нижче:

Переваги додаткових перевірок доступу на фронтенді:

– запобігання непотрібним асинхронним AJAX–запитам до сервера. Веб-додаток завантажує певні частини інтерфейсу і заповнює їх даними асинхронно. Програма оптимізована для продуктивності, оскільки уникатиме запитів на сервер для отримання даних, до яких користувач не має доступу;

– запобігання відображенню елементів інтерфейсу для операцій, які користувач не може виконувати. Веб-додаток працює краще завдяки покращеному часу завантаження та реакції сторінки, а також забезпечує узгодженість в користувацькому досвіді, видаляючи можливі точки помилок;

– покращення користувацького досвіду через відображення коректних повідомлень при виконанні несанкціонованих дій. Користувачам надаються чіткі повідомлення, якщо вони намагаються виконати дію, на яку не мають прав.

Метод контролю доступу: рольовий контроль доступу (RBAC).

Найбільш поширеним методом контролю доступу в веб-додатках є Role–Based Access Control (RBAC) (рисунок 2.9). Коли налаштовуються сервери авторизації, кожному користувачеві призначається роль, група або дозвіл. Користувач може отримати роль, яка визначає набір дозволів, які він може виконувати. Наприклад, користувач з роллю "Адміністратор" може мати дозвіл на додавання або видалення користувачів. Проте, користувач з роллю "Гість" може мати лише дозвіл на читання. Користувачі також можуть бути призначені до груп, кожна з яких має набір дозволів, доступних її членам.

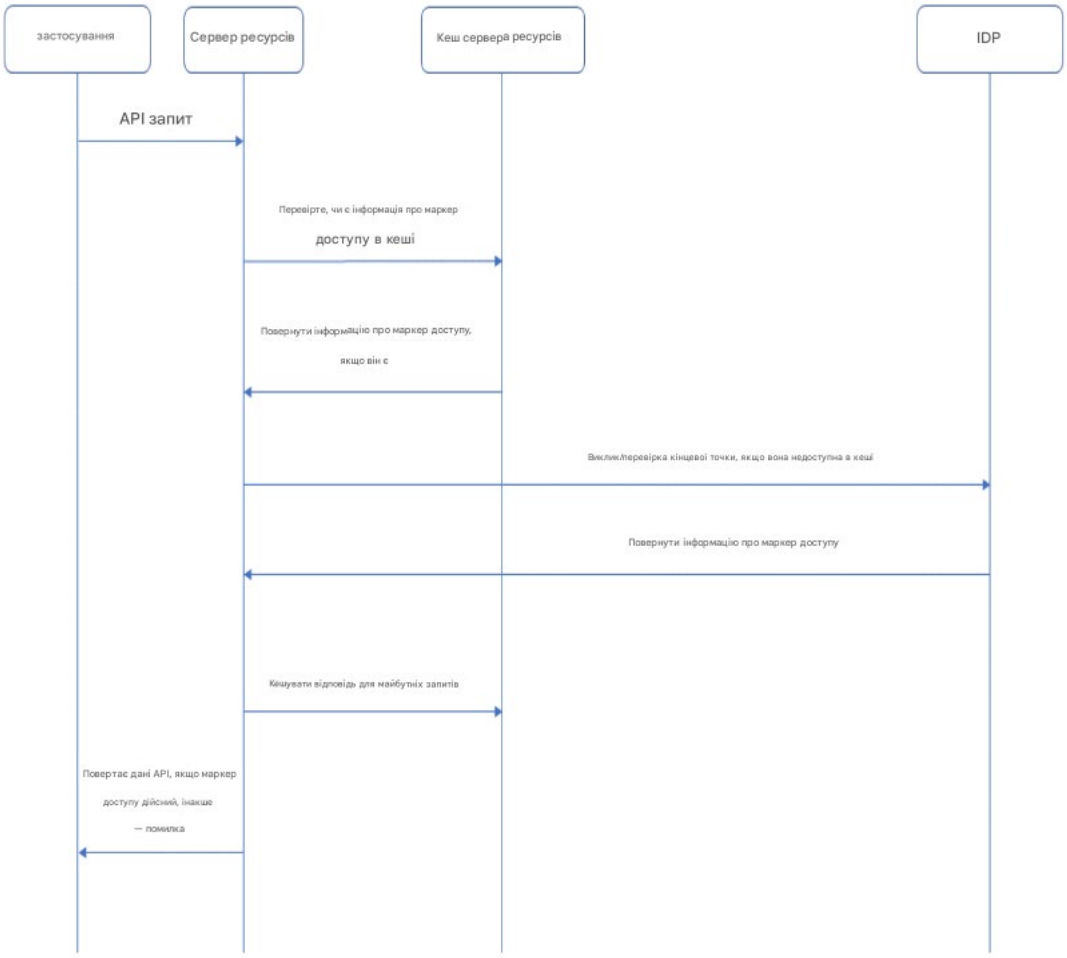

Рисунок 2.9 – Перевірка підтвердження маркера доступу

Імплементація контролю доступу через налаштування scope. Контроль доступу можна реалізувати за допомогою кастомних scope (обсягів доступу),

які вказують на дозволені дії для конкретного ресурсу. Зазвичай вони представлені у форматі ресурс:дія. Коли фронтенд-застосунок запитує токен доступу від сервера авторизації, він повинен включити параметр scope в запит, що є списком рядків, розділених пробілами. Сервер авторизації відповідає з дозволеними правами у параметрі scope токена доступу.

Крім того, сервер авторизації може відобразити інтерфейс для користувача, де той може погодитися або відмовитися від дозволів, запитуваних клієнтським додатком. Залежно від дії користувача, сервер може змінити набір дозволів перед тим, як відправити список scope назад в веб-додаток. Список доступних custom scopes визначається розробником API і зазвичай надається через документацію API.

Як реалізувати контроль доступу в SPA.

У SPA маршрути та елементи інтерфейсу повинні бути захищені на основі прав доступу, отриманих через scope. Це можна зробити, зберігаючи список всіх ролей і дозволених прав доступу в веб-додатку. На основі отриманого scope фронтенд-застосунок може впроваджувати route guards (охоронці маршрутів), що запобігають доступу користувача до захищених маршрутів. Наприклад, користувач з роллю "Гість" не повинен мати змоги отримати доступ до маршруту /admin. Крім того, елементи інтерфейсу повинні бути приховані або деактивовані на загальних сторінках на основі ролей користувачів. Наприклад, на сторінці Dashboard може бути кнопка "Додати користувача", яка має бути видима тільки для користувачів з роллю Адміністратор.

Це дозволяє застосунку дотримуватися політик безпеки, контролюючи доступ як на рівні маршруту, так і на рівні інтерфейсу.

Перевірка доступу на сервері ресурсів

Кожен запит на доступ перевіряється на сервері ресурсів для валідації. Це необхідно, оскільки зловмисник може обійти перевірки доступу на фронтенді веб-додатка. Зазвичай сервери ресурсів взаємодіють з серверами авторизації для перевірки запиту на доступ. Сервери авторизації зберігають Access Control

Lists (ACL), які містять інформацію про користувачів разом з їхніми ролями та дозволами.

OAuth 2.0 визначає точку доступу /introspect, яку сервери ресурсів можуть використовувати для валідації токена доступу [14]. Рекомендується не робити цю точку доступу публічною, оскільки вона може повернути чутливу інформацію про клієнта. Зазвичай її використовують на приватному сервері, до якого має доступ лише сервер ресурсів. Крім того, рекомендується захищати цю точку доступу механізмами аутентифікації, наприклад, HTTP Basic Authentication.

Як працює точка доступу /introspect:

– відбувається запит на точку доступу /introspect повертає JSON–об'єкт, що містить інформацію про токен доступу та клієнта;

– сервер ресурсів перевіряє поле scope, щоб визначити, чи має токен доступ до запитуваного ресурсу. Якщо дозволу недостатньо, сервер повертає помилку "неавторизований доступ" (unauthorized error).

Кешування запитів до /introspect.

Для підвищення продуктивності сервер ресурсів може кешувати значення, отримані з точки доступу /introspect. Час життя кешу визначається компромісом між безпекою та продуктивністю:

– коротші часи життя кешу покращують безпеку, оскільки токени перевіряються частіше;

– більші часи життя кешу можуть привести до ситуацій, коли сервер ресурсів використовує застарілі або прострочені токени, що знижує рівень безпеки.

Таким чином, для забезпечення безпеки, сервери ресурсів повинні ретельно контролювати доступ і здійснювати перевірки на кожному кроці, незалежно від того, чи були ці перевірки вже виконані на фронтенді.

2.10 Одноразова аутентифікація (SSO) з OpenID Connect (OIDC)

Single Sign–On (SSO) дозволяє користувачеві використовувати однакові облікові дані для кількох додатків в межах організації. Облікові дані користувача реєструються та зберігаються в IdP (Identity Provider) організації. Кожен додаток в організації має довірчі відносини з IdP. Для кожного додатка користувачі перенаправляються на сторінку входу IdP, де вони аутентифікуються. Після успішної аутентифікації IdP ділиться статусом аутентифікації з додатком. У випадку з OIDC, цей статус аутентифікації передається через ID токен [15] [16].

JWT (JSON Web Token) зазвичай використовується для токенів. Поле payload ID токена містить кілька claims (тверджень), які представляють інформацію про користувача. Ця інформація використовується фронтенд додатком для створення персоналізованого досвіду для користувача. Перед тим, як доступитися до поля payload, фронтенд додаток повинен декодувати ID токен і перевірити підпис JWT. Згідно з стандартами, payload токена закодовано у форматі base64URL [17]. Отже, необроблена інформація в payload доступна просто шляхом декодування у base64URL.

Перевірка автентичності токена.

Для перевірки автентичності отриманого токена фронтенд додаток повинен пройти кілька кроків:

– перевірка підпису JWT;

– вибір алгоритму підпису;

– перевірка підпису за допомогою RS256.

Висновки:

– SSO дозволяє користувачам використовувати єдині облікові дані для доступу до кількох додатків у межах організації, зберігаючи при цьому безпеку через передачу статусу аутентифікації за допомогою ID токенів.

– JWT використовується для передачі інформації про аутентифікацію, і

він має бути перевірений на фронтенді за допомогою алгоритмів підпису, таких як RS256 для публічних додатків.

– Важливо правильно реалізувати перевірку підпису токенів для забезпечення безпеки при взаємодії з відкритими веб-додатками, такими як SPA.

Коли Identity Provider (IdP) генерує JWT (JSON Web Token), він підписує його своїм приватним ключем. Токен потім передається клієнту, який для перевірки підпису використовує публічний ключ IdP. Для отримання цього публічного ключа, IdP повинен надати кінцеву точку, що містить конфігураційну інформацію про нього.

OIDC визначає стандартну кінцеву точку для цього, яка складається з базового URL IdP, доповненого шляхом /.well–known/OpenID–configuration [18] [19]. Наприклад, якщо адреса IdP – https://www.example.com, то конфігураційна URL буде виглядати так:

https://www.example.com/.well–known/OpenID–configuration

Відповідь на цей запит містить JSON-об'єкт з полем jwks_uri, яке вказує на місце, де можна знайти JSON Web Key Set (JWKS) IdP. JWKS містить масив JSON Web Keys (JWK), кожен з яких є криптографічним ключем, використаним IdP для підписування JWT. Кожен JWK має поле kid, що ідентифікує ключ, і публічний ключ для перевірки підпису.

Клієнт використовує значення kid з JWT для пошуку відповідного публічного ключа у JWKS та здійснює перевірку підпису.

Перевірка полів JWT.

Окрім перевірки підпису, існують ще три важливі поля, які повинні бути перевірені:

– Issuer (iss) – Ідентифікує IdP, що видав токен;

– Subject (sub) – Унікальний ідентифікатор користувача в рамках IdP (OIDC Provider), який аутентифікується;

– Audience (aud) – Ідентифікує клієнта, для якого призначений токен. Це має бути client ID OIDC клієнта (додатка, який отримує токен). Оскільки токен

повинен бути перевірений саме цим клієнтом, його client ID має міститися в полі aud.

OIDC також дозволяє додавати спеціальні значення до поля aud, якщо це потрібно для конкретних додатків.

Використання отриманих даних з JWT.

Після того, як токен успішно перевірено, фронтенд додаток може використовувати subject identity (sub) та інші claims з payload токена для створення локального профілю користувача. Специфікація OpenID Connect визначає кілька стандартних claims, пов'язаних з особистістю користувача, які можуть бути використані додатком.

Для того, щоб отримати ці claims у ID токену, при аутентифікації клієнт повинен вказати відповідні scope значення в запиті. Наприклад:

scope=openid profile email address phone

Список scope значень розділяється пробілами:

– openid – вказує на використання протоколу OpenID для аутентифікації та очікування ID токену в відповіді;

– profile – вказує, що додаток хоче отримати доступ до публічного профілю користувача, включаючи ім'я, стать, фото, день народження;

– email, address, phone – вказують на доступ до відповідних даних користувача.

Фронтенд розробник повинен ретельно обирати запитувані scope значення, щоб не порушити конфіденційність користувача та мінімізувати обсяг запитуваної інформації.

Мінімально необхідний набір claims. Користувач отримує запит на згоду, коли він аутентифікується через IdP, щодо дозволу веб-додатку доступати їхні дані. Тому запитуваний scope може не повністю відповідати тому набору claims, який насправді буде повернуто.

2.11 Single Logout

OIDC надає специфікації, що підтримують єдиний вихід (single logout). Основна ідея єдиного виходу полягає в тому, щоб завершити сесію користувача для всіх веб-додатків, в які користувач увійшов через один і той самий IdP. Це дозволяє здійснити вихід один раз, замість того, щоб виходити з кожного веб-додатку окремо.

При виході користувача з веб-додатку, додаток має обробляти кілька сценаріїв, щоб забезпечити успішний вихід. По-перше, необхідно очистити змінні сесії з власного контексту безпеки, який підтримується для цього веб-додатку. Потім, додаток повинен виконати вихід через IdP, що очистить сесію на боці IdP. Залежно від випадків використання, веб-додаток може виконати локальний або глобальний вихід.

У випадку локального виходу, веб-додаток виходить з поточного активного додатку, що ініціював вихід. У випадку глобального виходу, IdP ініціює вихід з усіх додатків, де є активна сесія користувача.

Для локального виходу OIDC-клієнт (рисунок 2.10) може реалізувати простий механізм виходу, заснований на логіці тайм-ауту. Кожен ID токен містить поле expiration, яке вказує час, до якого токен є дійсним. OIDC-клієнт запускає внутрішній таймер, який ініціює вихід, коли час дії ID токену закінчується. Однак цей механізм не працюватиме, якщо OIDC-клієнт використовує refresh токени для збереження сесії користувача або використовує періодичну тиху аутентифікацію для підтримки користувача в стані авторизованого.

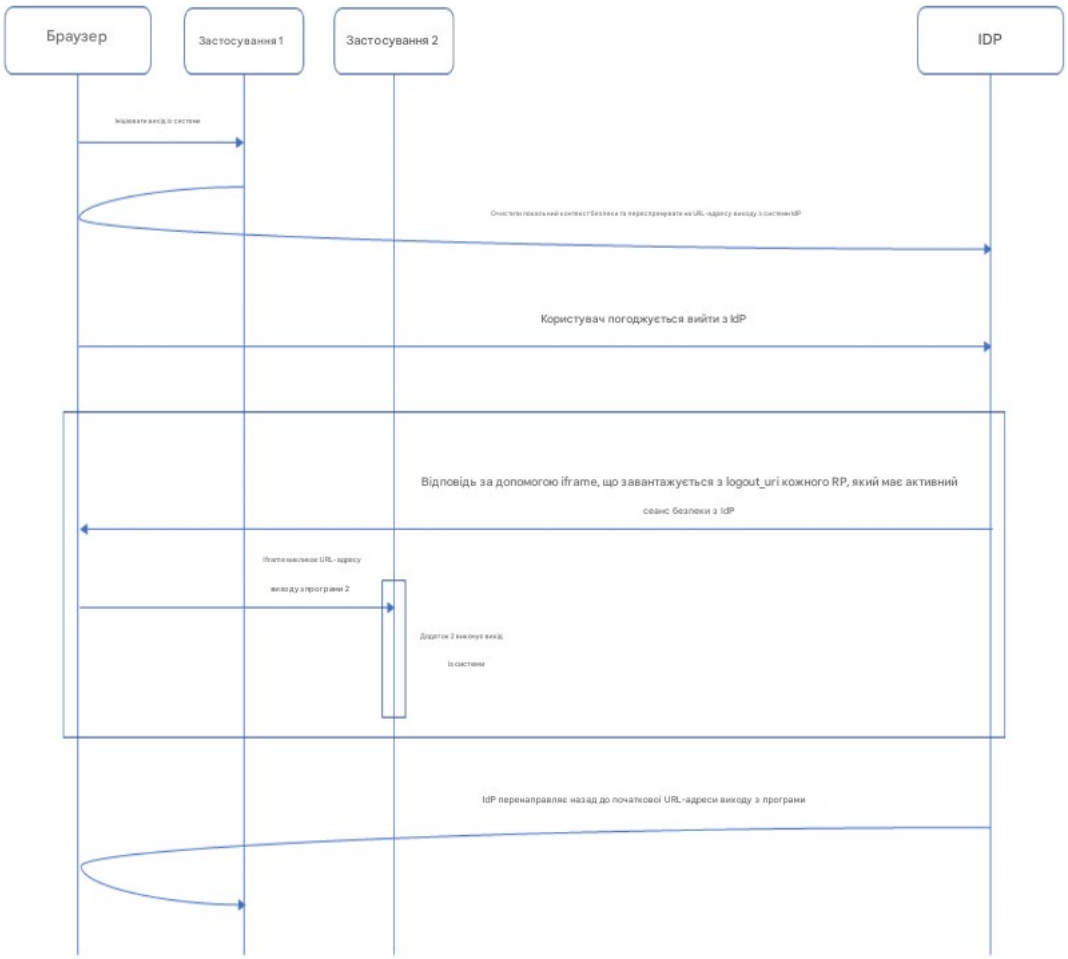

Рисунок 2.10 – Вихід із переднього каналу OIDC

Специфікація управління сесією OIDC описує, як веб-додаток може виконати локальний вихід [12]. Розділ управління сесією описує, як OIDC-клієнт підтримує сесію з IdP. Для виходу, ініційованого OIDC-клієнтом, клієнт спочатку очищує свою власну сесію з користувачем. Потім він перенаправляє браузер на визначену IdP кінцеву точку, яка очищає сесію, що підтримується IdP для користувача. IdP повинен реалізувати кінцеву точку виявлення сервісу, як зазначено в OIDC (рисунок 2.11), через яку OIDC-клієнти можуть знайти URL для виходу.

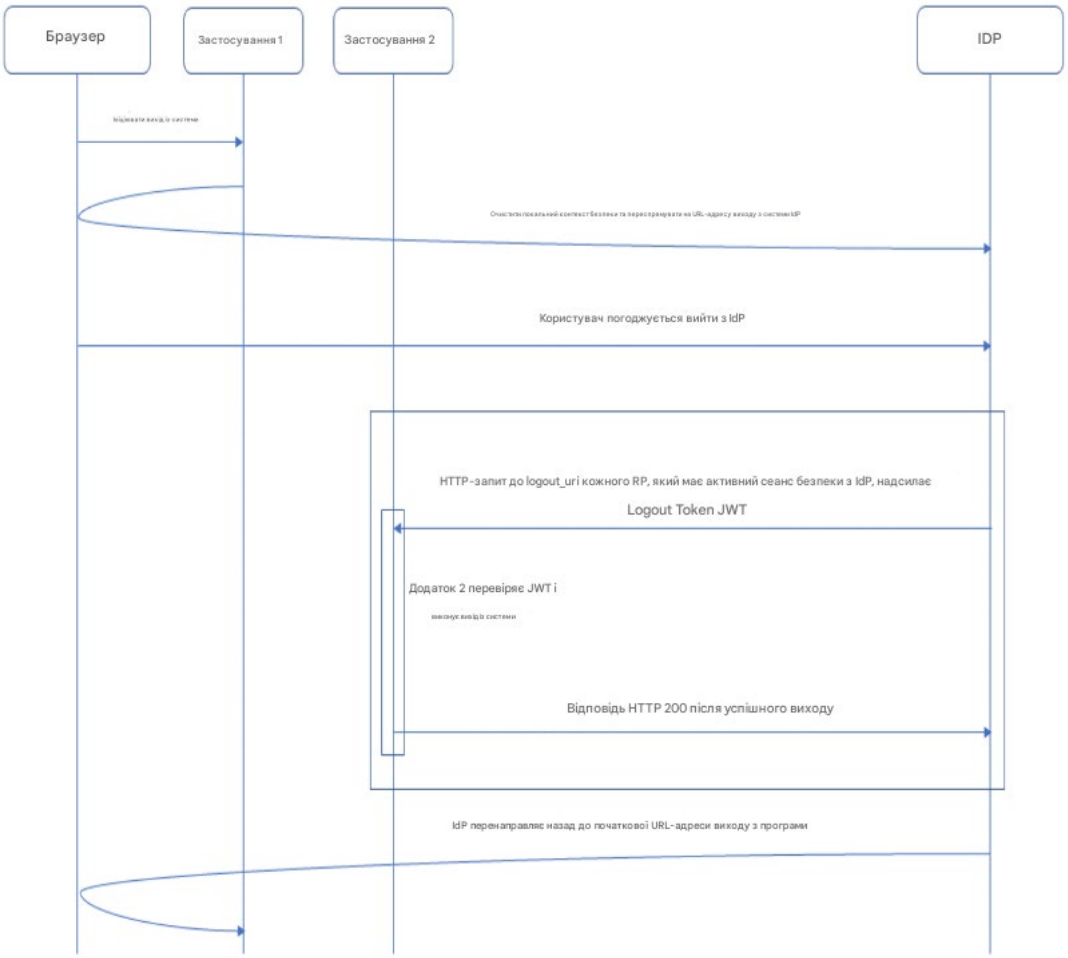

Рисунок 2.11 – Вихід із зворотного каналу OIDC

OIDC-клієнт відправляє параметри id_token_hint, post_logout_redirect_uri та state. На кінцевій точці виходу IdP має запитати користувача, чи бажає він вийти також з IdP. Це викликає глобальний вихід залежно від вибору користувача. Для виходу, ініційованого IdP, OIDC–клієнт повинен періодично запитувати IdP про статус сесії користувача. Це робиться за допомогою двох невидимих iframe: один створюється OIDC–клієнтом на його власному домені з сторінкою, що містить функції управління сесією, а інший створюється OIDC–клієнтом із URL сесії, наданим IdP. Ці два iframe спілкуються між собою через window.postMessage API. На кожен запит IdP відповідає одним із трьох значень статусу: changed, unchanged або error. Якщо статус changed, RP (ресурсний сервер) має ініціювати тиха повторну аутентифікацію з IdP. Якщо IdP не відповідає з ID токеном або відповідає з токеном для іншого користувача, OIDC–клієнт має обробити це як вихід. Якщо статус error, OIDC–клієнт також

має обробити це як вихід без спроби виконати тиху повторну аутентифікацію.

Для глобального виходу OIDC визначає два механізми: Front channel logout [22] та Back channel logout [23]. Front channel logout використовує браузер для обробки редіректів, що виконують кроки виходу. Цей механізм використовується клієнтами односторінкових додатків, які обробляють управління сесіями на фронтенді. Back channel logout виконується IdP через пряме спілкування з OIDC-клієнтом. У цьому випадку клієнт є проміжним сервером на бекенді, який обробляє аутентифікацію, авторизацію та управління сесією з IdP. Для фронтенд-розробника важливо розуміти механізм front channel logout [24, 25].

Рисунок 2.10 показує потік виходу через фронтальний канал між IdP та OIDC.

OIDC-клієнти – це односторінкові веб-додатки (SPA), які працюють у браузері користувача. У випадку front–channel logout, IdP не має прямого способу зв’язатися з клієнтською або серверною частиною інших додатків B і C для того, щоб вивести користувача з цих додатків. Замість цього браузер користувача повинен надіслати ці повідомлення про вихід на сервери додатків B і C. Щоб реалізувати це, кожен додаток має зареєструвати front_channel_logout URI в IdP. Користувач ініціює глобальний вихід із додатка A через браузер, що запускає механізм глобального виходу в IdP. Відповідь IdP на браузер містить сторінку глобального виходу, яку реалізує IdP, яка інформує сервери додатків B і C про вихід. Це здійснюється шляхом відображення iframe на сторінці глобального виходу. Згідно з політикою того ж походження (same origin policy), сторінка глобального виходу створює новий iframe для кожного з додатків із їх зареєстрованим front_channel_logout URI. Сервери потім повідомляють клієнтську частину додатків B і C в браузері користувача та також видаляють стан сесії зі своїх серверів. Нарешті, IdP виконує редірект назад на post–logout URL додатка A.

IdP ініціює back–channel глобальний вихід шляхом надсилання запиту на вихід до кожного додаткового клієнта, який має встановлену сесію, на їх

зареєстрований back_channel_logout URI. IdP надсилає токен для виходу, який відомий як Security Event Token (SET) [26]. SET – це JWT токен, який представляє набір подій безпеки та ідентифікації, що відбуваються для клієнта. Специфікація OIDC визначає набір необхідних полів, які повинні бути присутніми в SET, щоб ідентифікувати його як токен виходу. Ось ці поля:

– iss – ідентифікатор видавця SET, яким є ідентифікатор IdP;

– aud – аудиторія SET, тобто client ID додатка, для якого призначений цей токен;

– iat – час видачі (issued at time), який вказує на момент, коли SET був виданий;

– jti – унікальний ідентифікатор JWT, який унікально ідентифікує SET;

– events – це поле містить JSON–об’єкт, де ключем повинно бути значення http://schemas.openid.net/event/backchannel–logout, а значенням – порожній JSON-об’єкт {}.

Запит на back–channel logout від IdP до клієнта виглядає наступним чином:

POST /backchannel_logout HTTP/1.1 Host: client.com Content–Type: application/x–www–form–urlencoded logout_token=

Отримавши запит на вихід через back–channel з SET, клієнт повинен перевірити токен SET. Кроки перевірки схожі на кроки перевірки ID токена. Крім того, клієнт повинен переконатися, що токен SET містить поле events з правильним JSON-об’єктом, як зазначено вище, і що до цього часу він не отримував інший токен для виходу з таким самим значенням jti. Якщо будь-який з кроків перевірки не проходить, клієнт повинен відповісти помилкою HTTP 400 Bad Request.

Після того як перевірка пройдена успішно, клієнт виконує свій процес виходу, специфічний для реалізації. Клієнт може використовувати значення sub або iss в SET для пошуку будь-яких змінних сесії в своїй сесії та видалення їх. Після успішного виходу клієнт відповідає IdP запитом HTTP 200.

2.12 Token storage

Зберігання токенів у сервері бекенду є природно більш безпечним, ніж у фронтенд-додатку, що працює в веб-браузері. Веб-додаток на стороні клієнта, який працює у браузері, називається публічним клієнтом, оскільки його вихідний код і дані повністю видимі користувачу через браузер. Сервери бекенду не дозволяють доступу до свого коду для публіки. До нього мають доступ лише розробники, які працюють в середовищі розробки. Функціональність сервера бекенду надається через публічні API, які можуть викликатися будь-яким публічним клієнтом. Для кожного запиту до API сервер бекенду відповідає даними, які запросив користувач. Публічний клієнт не має доступу до того, як надаються ці дані або звідки вони надходять. Тому зберігання чутливої інформації в середовищі бекенду вважається безпечним.

З іншого боку, оскільки весь вихідний код фронтенд-додатку видимий для всіх, необхідно вжити додаткових запобіжних заходів при зберіганні та використанні чутливої інформації. Спільні секретні ключі, які використовуються в асиметричній криптографії, ніколи не зберігаються в публічних клієнтах.

Токен доступу, який використовується в операціях OIDC, є токеном типу bearer. Будь-хто, хто має доступ до нього, може зробити дійсне запит до API, яке приймає цей токен. Крім того, він повинен бути відправлений у заголовку HTTP-запиту як Bearer. Для веб-додатків, що мають сервер бекенду, токени безпечно зберігаються на сервері, який модифікує кожен вихідний API запит, вставляючи токени, і передає відповіді API назад у веб-додаток. Для веб-додатків, які не мають серверу бекенду, як–от статичні веб-додатки, вставка токенів доступу в кожен запит має здійснюватися за допомогою JavaScript коду.

Токени доступу і refresh токени можуть зберігатися в веб-браузері у localStorage або sessionStorage. Браузери можуть зберігати пари ключ:значення в цих місцях зберігання. SessionStorage зберігає дані лише протягом однієї сесії. Він дозволяє доступ до даних лише під час поточної сесії.

У межах вікна або вкладки, в яких були створені дані. SessionStorage очищається, як тільки користувач закриває браузер. LocalStorage зберігає дані навіть після закриття браузера. Ці дані можуть бути доступні через різні вкладки та вікна. Браузери зберігають ці значення за політикою per–origin, що означає, що вони не дозволяють доступ до цих сховищ з інших доменів. Однак будь-який JavaScript, що працює на домені додатку, може отримати доступ до цього сховища. Це створює загрозу безпеці, відому як Cross–Site Scripting (XSS). Уразливість XSS може бути також в сторонніх бібліотеках, які використовуються у веб-додатку. [27]

Щоб уникнути ризиків, пов’язаних із зберіганням токенів у сховищі браузера, постачальники OIDC (OP) можуть вибрати зберігання токенів у cookies (куки). Куки також зберігаються в браузері на основі політики домену, що означає, що веб-сторінка може отримати доступ лише до куки, яка асоційована з її доменом. Куки надають низку прапорців, які можна встановити для запобігання їх компрометації. OP може встановити прапорець Secure для куки, щоб вона надсилалась лише з запитами через HTTPS, а також прапорець httpOnly, щоб JavaScript не міг отримати доступ до куки через document.cookie DOM API. Також можна встановити прапорець SameSite, який забезпечує, що куки надсилаються лише за запитами з того ж походження (origin). Однак браузери мають обмеження за розміром для куки – до 4 кілобайт. Тому будь-які дані або токени, більші за 4 кілобайти, не можна зберігати в куки.

2.13 Software Development Kits

Програмні комплекти розробки (SDK) – це інструменти, які дозволяють розробникам використовувати певну функціональність. SDK об'єднують операції та функціональність, що стосуються певної логіки, і надають API, які можна викликати для їх виконання. SDK (рисунок 2.12) містять інструкції для розробників щодо їх використання на підтримуваних платформах. Вони легко

інтегруються з існуючими додатками, що надає зручний спосіб для розробників додавати нову функціональність до своїх веб-додатків. Один з прикладів – SDK для аутентифікації та авторизації. Кроки, необхідні для виконання аутентифікації та авторизації, визначені специфікаціями. Перехід від цих специфікацій до коду має забезпечити однакову функціональність в кожному додатку.

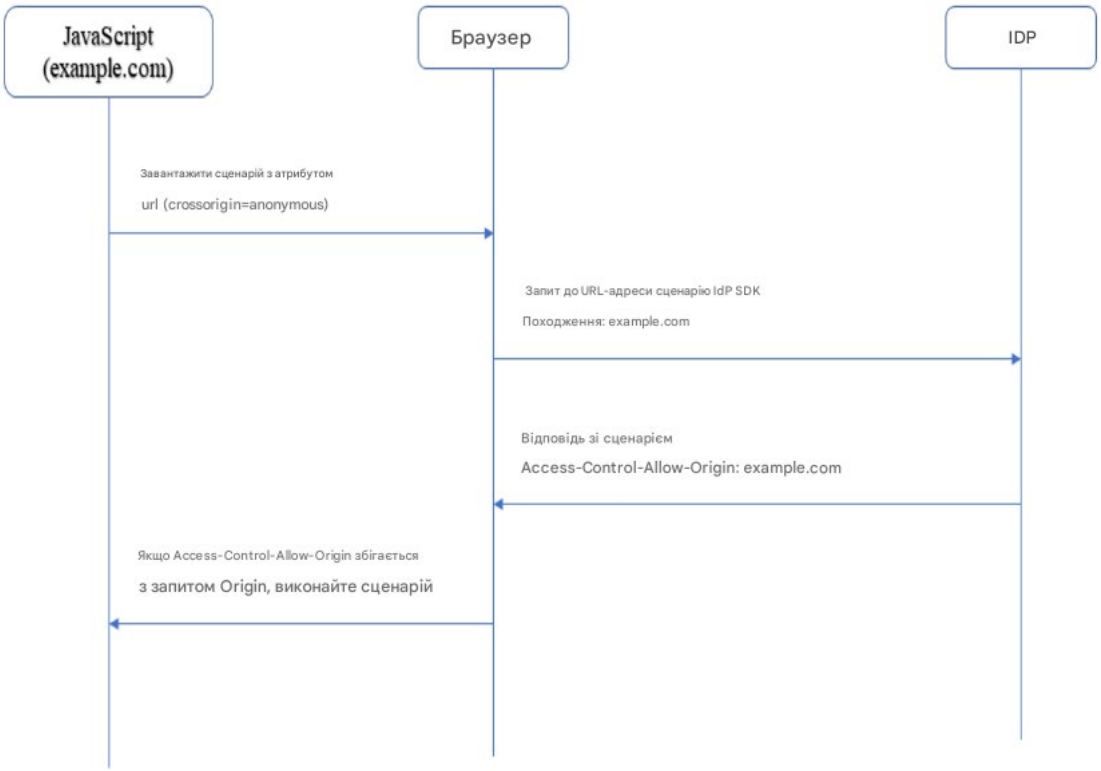

Рисунок 2.12 – Завантаження скрипту за допомогою CORS

Об'єднання цих кроків в одному місці та надання їх через API має кілька переваг, як зазначено нижче:

– розробникам не потрібно писати однаковий код для кожного додатку, який слідує тим самим крокам аутентифікації та авторизації. Це значно знижує витрати часу та коштів, необхідних для розробки та випуску додатку;

– розробникам не потрібно турбуватися про кроки, які виконуються для

досягнення певної операції. Розробники додатків не повинні піклуватися про методи реалізації певних операцій, таких як зберігання токенів і управління сесіями. Зберігання та отримання токенів обробляється SDK. Створення iframe для управління сесією та періодичне опитування IdP обробляються SDK. SDK містить надійні механізми обробки помилок, які охоплюють всі можливі межі та крайні випадки. Розробники SDK відповідають за підтримку SDK та забезпечення того, щоб він працював належним чином без будь-яких помилок чи вразливостей;

– впровадження нових функцій і підтримка існуючих функцій можуть виконуватися в одному місці, а не в кожному додатку, що їх використовує. Наявність єдиного коду полегшує налагодження та виправлення помилок;

– це забезпечує єдиний досвід роботи в подібних додатках. Завдяки однаковій конфігурації розробник може бути впевнений, що операції працюватимуть подібним чином у всіх додатках;

– SDK слідують єдиним шаблонам у своїх API та використанні. Ця абстракція основних операцій дозволяє легше мігрувати між різними платформами.

Постачальники ідентифікації (IdP) впроваджують та випускають URL–ендпоінти в різних версіях. Для забезпечення сумісності нові версії публікуються за іншими URL. SDK підтримують URL–ендпоінти, які використовуються з конкретним IdP. Оскільки міграція до новішої версії URL обробляється безпосередньо SDK, розробник додатку не повинен турбуватися про це. Без SDK міграція має бути здійснена для кожного додатку, що може бути трудомістким і схильним до помилок. Крім того, IdP може змінювати операції з новими URL. Без SDK розробнику додатку доведеться змінювати логіку операцій у кожному додатку.

Більшість постачальників ідентифікації мають доступні JavaScript SDK для використання з веб-клієнтами. Кожен SDK починається з завантаження JavaScript-бандлу на веб-сторінку. Це здійснюється за допомогою тега <script>, в якому атрибут src вказує на URL бандлу, наданий постачальником

ідентифікації. Це завантажує SDK під час завантаження сторінки і відкриває глобальну змінну, через яку доступні API SDK.

Веб-додатки повинні враховувати політику Cross–Origin Resource Sharing (CORS) при доступі до ресурсів з зовнішніх серверів. CORS – це механізм, який реалізують веб-браузери для запобігання доступу JavaScript-коду до даних, які походять з іншого джерела, ніж сторінка, на якій працює цей код. Наприклад, SDK обробляють помилки і відображають повідомлення про помилки через журнали помилок для налагодження, що є необхідним для розробника, який використовує SDK. Коли скрипт SDK завантажується з зовнішнього сайту, браузер відображає повідомлення Script Error у журналі замість правильного повідомлення про помилку. Це відбувається через те, що перевірка політики того ж походження у браузері забороняє обмін помилками. Логування даних через різні домени. Перевірка на те, чи є запит із того ж джерела, не виконується браузерами, коли скрипт завантажується з зовнішнього сервера за допомогою тега <script>. Це можна обійти за допомогою двох кроків. По–перше, атрибут crossorigin у тегу <script> встановлюється в значення anonymous, що вказує браузеру дозволити політику того ж походження для запиту [29]. По–друге, зовнішній сервер IdP, що хостить скрипт SDK, відповідає на запит, включаючи заголовок Access–Control–Allow–Origin з значенням Origin, яке відповідає значенню запиту. Коли ці значення збігаються, браузер дозволяє обмін даними з зовнішнього скрипту. Крім атрибута crossorigin, тег <script> також може містити атрибути defer і async [29]. Атрибут defer гарантує, що скрипт буде виконано після того, як вся сторінка буде розібрана, а атрибут async забезпечує, що скрипт буде виконано у фоновому режимі. Ці два атрибути гарантують, що завантаження та виконання скрипту не блокують розбір та рендеринг основної HTML–сторінки. Кожен SDK вимагає початкової конфігурації після завантаження скрипту. Конфігурація зазвичай включає в себе ідентифікатор клієнта, області токенів та інші поля, пов’язані з IdP. Більшість SDK вимагають API-ключ у конфігурації, як це робить Google з JavaScript SDK.

Хоча це не є безпечним для аутентифікації користувачів, Google використовує API-ключі для обмеження кількості запитів, які можуть бути надіслані до його API, і це не пов’язано з аутентифікацією користувачів. Іншому веб-додатку можна заборонити використовувати API-ключ, використовуючи заголовок HTTP referrer. Це поле заголовка містить інформацію про останню веб-сторінку, яку відвідав користувач. Для API-запитів воно міститиме значення правильного домену веб-додатку, який викликав API.

Поле заголовка referrer є обмеженим і не може бути змінене програмно через JavaScript. Розробник також має налаштувати прийняті значення для поля referrer у порталі конфігурації SDK. Використання API є досить простим. Вони використовують асинхронне програмування за допомогою зворотних викликів (callbacks) або обіцянок (promises). Зворотні виклики або обіцянки в JavaScript – це функції, які передаються як параметри в функцію SDK. Функція SDK викликає зворотний виклик з визначеними параметрами після того, як вона виконає свій код. Кожен SDK визначає, які параметри необхідно передати разом зі зворотним викликом під час виклику API. Наприклад, JavaScript SDK Facebook передає результат виклику входу в об’єкті відповіді. SDK керує зберіганням токенів, їх відкликанням і поновленням, якщо це підтримується. Для кожного виклику, пов’язаного з аутентифікацією та авторизацією, SDK автоматично додає токени до API-запитів. Загальною функціональністю, яку надають усі SDK, є аутентифікація користувача та надання ID токена веб-додатку, що викликає API. Більшість SDK постачають заздалегідь створену кнопку входу, оформлену відповідно до унікального дизайну IdP. Вони пропонують підхід «plug–and–play», що дозволяє веб-додаткам підтримувати сервіси ідентифікації IdP з мінімальними налаштуваннями. Розробники також можуть використовувати API для входу для досягнення того ж результату. Поширеною практикою для відображення сторінки входу IdP є відкриття спливаючого вікна замість перенаправлення. У таких випадках спливаюче вікно та оригінальне вікно додатка спілкуються за допомогою API postMessage.

Оригінальний додаток слухає подію повідомлення, яку відправляє спливаюче вікно. Повідомлення містить токени та інформацію про процес входу. З міркувань безпеки оригінальний додаток завжди має перевіряти джерело кожного запиту, отриманого через подію повідомлення. Він повинен приймати і обробляти тільки ті повідомлення, які надіслані з відомого та довіреного джерела.

2.14 Credentials Manager

Менеджер облікових даних вбудований у користувацький агент і допомагає управляти обліковими даними користувачів. Консорціум World Wide Web (W3C) визначає API для керування обліковими даними, яке користувацькі агенти повинні надавати, через яке веб-додатки можуть взаємодіяти з Менеджером облікових даних [30]. За допомогою Менеджера облікових даних фронтенд-розробники можуть спростити аутентифікацію в своїх додатках, одночасно забезпечуючи покращений досвід користувача, зменшуючи частоту взаємодії з користувачем.

API для керування обліковими даними надає кілька переваг, які детально описані нижче:

– користувачам не потрібно пам'ятати свої ім’я користувача та пароль, оскільки вони зберігаються в Менеджері облікових даних. Це дозволяє користувачам встановлювати безпечні паролі;

– користувачі можуть використовувати збережені облікові дані для автоматичного входу в систему. Оскільки сесії зазвичай закінчуються через день, користувачі не стикаються з формою входу щоразу, коли відвідують веб-сторінку. Якщо облікові дані користувача не змінюються, це дозволяє створити практично постійну сесію;

– мменеджер облікових даних підтримує збереження даних для федеративних логінів поряд із паролями. Це дозволяє веб-додаткам інтегрувати

федеративних постачальників ідентифікації в додаток, при цьому використовуючи Менеджер облікових даних;

– Web Authentication API, побудоване на основі API для керування обліковими даними, дозволяє використовувати фізичні об'єкти, такі як USB–фоб або біометричні зчитувачі, як засоби ідентифікації. Це усуває використання текстових паролів, що підвищує рівень безпеки.

Credentials Management API (рисунок 2.13)

API для керування обліковими даними надає інтерфейси для взаємодії з Менеджером облікових даних. На цей момент його підтримує 83% усіх браузерів. Веб-додатки можуть використовувати ці API через об'єкт navigator.credentials.

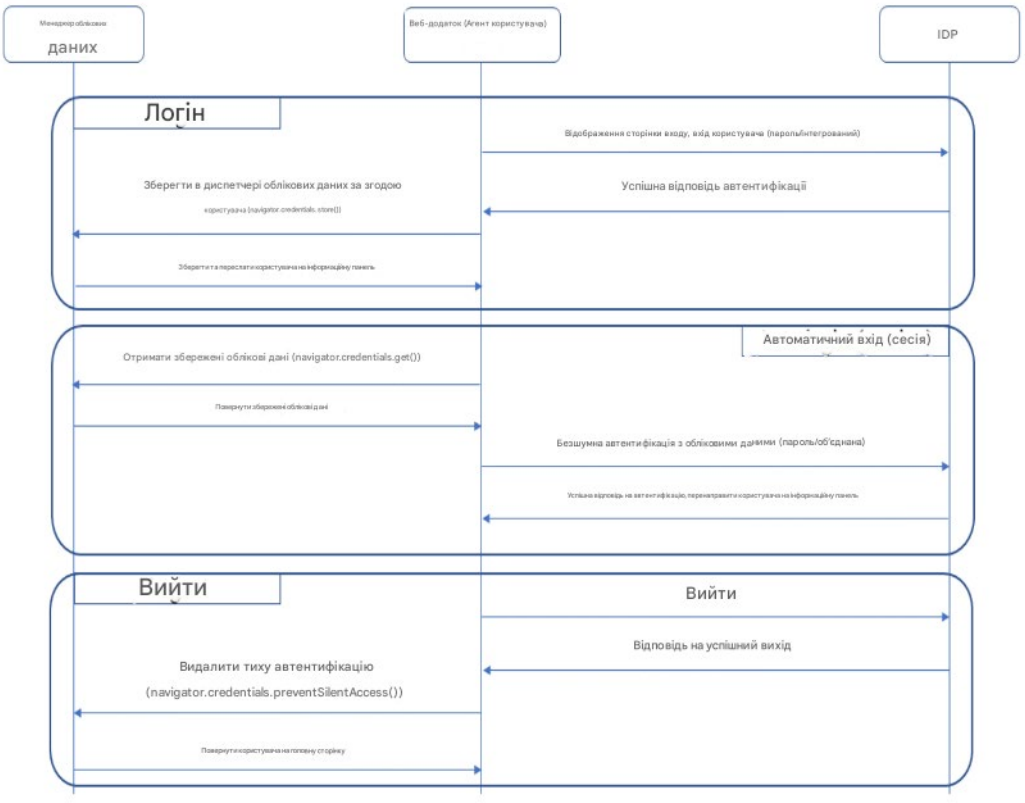

Рисунок 2.13 – Використання Credentials Management API

API пропонує чотири основні функції:

– navigator.credentials.create – цей API створює об'єкт Credential, що

містить облікові дані користувача. Це може бути або PasswordCredential (облікові дані на основі пароля), або FederatedCredential (федеративні облікові дані). PasswordCredential вимагає ID користувача та пароль, а FederatedCredential вимагає ID користувача та постачальника федеративної ідентифікації. Ім'я користувача та пароль автоматично визначаються через елементи введення в HTML, за допомогою атрибута autocomplete. Специфікація HTML визначає значення атрибута autocomplete як username для імені користувача або ID, new–password для нового пароля (який використовується у формі реєстрації, де користувач створює новий пароль), та current–password для поточного пароля користувача;

– navigator.credentials.store – цей API зберігає облікові дані в сховищі Менеджера облікових даних. Він викликається після створення об'єкта Credential за допомогою create API. Користувач завжди отримує сповіщення перед збереженням пароля, і облікові дані можна зберігати лише за згодою користувача;

– navigator.credentials.get – цей API отримує облікові дані, що зберігаються в сховищі Менеджера облікових даних;

– navigator.credentials.preventSilentAccess – цей API забороняє автоматичний вхід за допомогою облікових даних користувача. Зазвичай цей метод викликається після того, як користувач вийшов із додатку, щоб автоматичний вхід не виконувався при наступному відвідуванні веб-сторінки.

Ці API працюють у захищеному контексті користувача. Захищений контекст веб-додатка визначається за походженням (origin) найактивнішого документа веб-сторінки. Це запобігає доступу до даних через менеджер облікових даних з iframe. Кожен об'єкт Credential зберігається в контексті безпеки, який перевіряється кожного разу, коли доступ до нього запитується через get API. Менеджер облікових даних дозволяє зберігати кілька облікових даних для одного веб-додатка. Якщо існують кілька облікових записів, користувач завжди отримає запит на вибір облікового запису, за допомогою якого він хоче увійти під час автоматичного входу.

2.15 Web Authentication API

API Web Authentication (WebAuthn) побудована на основі Credential Management API, що надає облікові дані на основі публічних ключів для веб-додатків. Вона використовує автентифікатори для створення сильно підтверджених та обмежених облікових даних на основі публічних ключів і дозволяє зберігати та використовувати ці облікові дані з постачальниками ідентифікації (IdP), що їх підтримують.

Автентифікатор – це абстрактна сутність, яка надає та підтверджує ідентичність користувача. W3C визначає Модель автентифікатора WebAuthn, яка встановлює протоколи, яким має відповідати автентифікатор повинен слідувати, щоб його можна було використовувати веб-браузером з WebAuthn API [31]. Автентифікатор може бути програмним компонентом, що використовує довірений модуль безпеки (TPM) платформи, або апаратним компонентом, таким як Windows Hello чи сенсори відбитків пальців на мобільному пристрої. Це також може бути незалежний апаратний модуль, що відповідає стандарту FIDO–CTAP і може використовуватися через Bluetooth або Near Field Communications (NFC) [32].

WebAuthn API (рисунок 2.14) використовує два основних методи, які описані нижче:

– navigator.credentials.create – цей метод ініціює автентифікатор для створення та збереження нового набору облікових даних на основі публічних ключів для користувача.

– navigator.credentials.get – цей метод отримує існуючі облікові дані на основі публічних ключів із сховища.

Два основних компоненти в цьому процесі – це сервер IdP (постачальник ідентифікації) та автентифікатор. Сервер має підтримувати реєстрацію та автентифікацію через асиметричне криптографічне шифрування. Автентифікатор зберігає ідентифікаційну інформацію користувача. У випадку автентифікації, що базується на паролі, ідентифікаційна інформація

зберігається в мозку користувача.

Рисунок 2.14 – Реєстрація за допомогою API веб-автентифікації

Реєстрація веб-клієнта з сумісним IdP відбувається у кілька етапів, як описано нижче [31]:

– сервер спочатку надсилає виклик реєстрації на запит, зроблений веб-клієнтом. Виклик генерується випадковим чином сервером і використовується як ідентифікатор операції. Крім того, сервер надсилає інформацію про користувача та ID клієнта;

– клієнт викликає метод navigator.credentials.create і передає параметри виклику, отримані від сервера, як об'єкт PublicKeyCredentialCreationOptions. Цей об'єкт містить ID клієнта, інформацію про користувача, виклик, надісланий сервером, а також об'єкт pubKeyCredParams, який містить алгоритми, що повинні використовуватися для облікових даних;

– браузер внутрішньо викликає метод authenticatorMakeCredential на автентифікаторі після перевірки даних клієнта та додавання додаткових даних до об'єкта clientDataJSON. До цього об'єкта також додається походження

запиту, яке сервер використовує для перевірки запитів з однаковим походженням. Хеш SHA-256 об'єкта clientDataJSON, відомий як clientDataHash, передається під час виклику методу автентифікатора;

– автентифікатор запитує у користувача методи ідентифікації, такі як Windows Hello або сенсор відбитків пальців. Після перевірки автентифікатор створює нову пару асиметричних ключів і підтверджує публічний ключ, підписуючи його власним приватним ключем. Приватний ключ вбудований в автентифікатор під час виробництва. Його можна перевірити за допомогою сертифіката кореня довіри через ланцюг сертифікатів;

– браузер отримує об'єкт PublicKeyCredential, який містить створений автентифікатором rawId, clientDataJSON, дані підтвердження, створені автентифікатором, та публічний ключ. Об'єкт відправляється назад на сервер IdP для завершення процесу реєстрації;

– сервер перевіряє інформацію в об'єкті PublicKeyCredential. Найважливіше – це перевірка оригінального виклику, ID клієнта та походження. Наприкінці перевіряється підтвердження на основі clientDataHash, і якщо валідація успішна, сервер зберігає облікові дані для майбутнього використання.

Аутентифікація користувача з веб-клієнта відбувається за подібними кроками до процесу реєстрації, з кількома змінами, як описано нижче:

– сервер надсилає виклик на запит аутентифікації від веб-клієнта. На цьому етапі не передається жодної іншої інформації;

– веб-клієнт викликає метод navigator.credentials.create API Менеджера облікових даних. Він передає виклик у об'єкті PublicKeyCredentialRequestOptions методу create, який містить виклик, надісланий сервером, разом з деякими додатковими необов'язковими даними, такими як ID клієнта;

– браузер внутрішньо викликає API authenticatorGetCredential автентифікатора, передаючи ID клієнта та хеш SHA-256 (clientDataHash) об'єкта clientDataJSON, який містить дані, надані при виклику create. Браузер виконує

первинну перевірку даних перед створенням об'єкта clientDataJSON;

– автентифікатор використовує ID клієнта для пошуку відповідного облікових даних та запитує у користувача підтвердження та дозвіл. Після перевірки та підтвердження автентифікатор створює підтвердження, підписуючи clientDataHash приватним ключем користувача, згенерованим на етапі реєстрації. Автентифікатор надсилає це підтвердження та дані автентифікатора назад до браузера в об'єкті PublicKeyCredential;

– браузер надсилає це підтвердження, хеш clientDataHash і дані автентифікатора назад на сервер. Сервер використовує збережений публічний ключ користувача для перевірки підтвердження. Крім того, він перевіряє виклик, який був надісланий на першому етапі, походження запиту та ID клієнта. Після успішної перевірки сервер авторизує користувача та надсилає аутентифікаційні токени веб-клієнту.

2.16 Аутентифікація та авторизація в нативних додатках

Розробка нативних (мобільних) додатків традиційно використовує власні середовища розробки та мови програмування. Два найбільш популярних мобільних операційних системи на сьогоднішній день – це Android та iOS. Обидві системи надають свої власні SDK для розробників додатків, що дозволяє їм отримати доступ до всього екосистеми. Однак останнім часом зростає популярність інструментів, що не залежать від ОС для створення нативних додатків. Це дає можливість розробникам націлюватися на обидві платформи, заощаджуючи значний час розробки, використовуючи один набір інструментів та середовище розробки. Багато компаній прийняли цей підхід, щоб забезпечити своїм користувачам доступ до додатків незалежно від операційної системи, яку вони використовують. Серед бібліотек, що існують сьогодні, найбільш популярними для розробки нативних додатків є React Native та Ionic. Розробники використовують веб-технології, такі як HTML, JavaScript

та CSS, для створення додатків, що підвищує адаптивність і знайомство з розробкою нативних додатків. Це дозволяє командам або організаціям використовувати своїх існуючих фронтенд-розробників для проектів розробки нативних додатків, замість того, щоб наймати спеціалістів з розробки нативних додатків для Android або iOS. У цьому контексті важливо згадати механізми аутентифікації та авторизації на основі токенів, що використовуються та рекомендуються для нативних додатків. [33] OAuth 2.0 встановлює основні правила, яких слід дотримуватись для нативних додатків [34]. Нативні додатки повинні використовувати веб-браузер для аутентифікації з IdP (рисунок 2.15) (постачальником ідентичності).

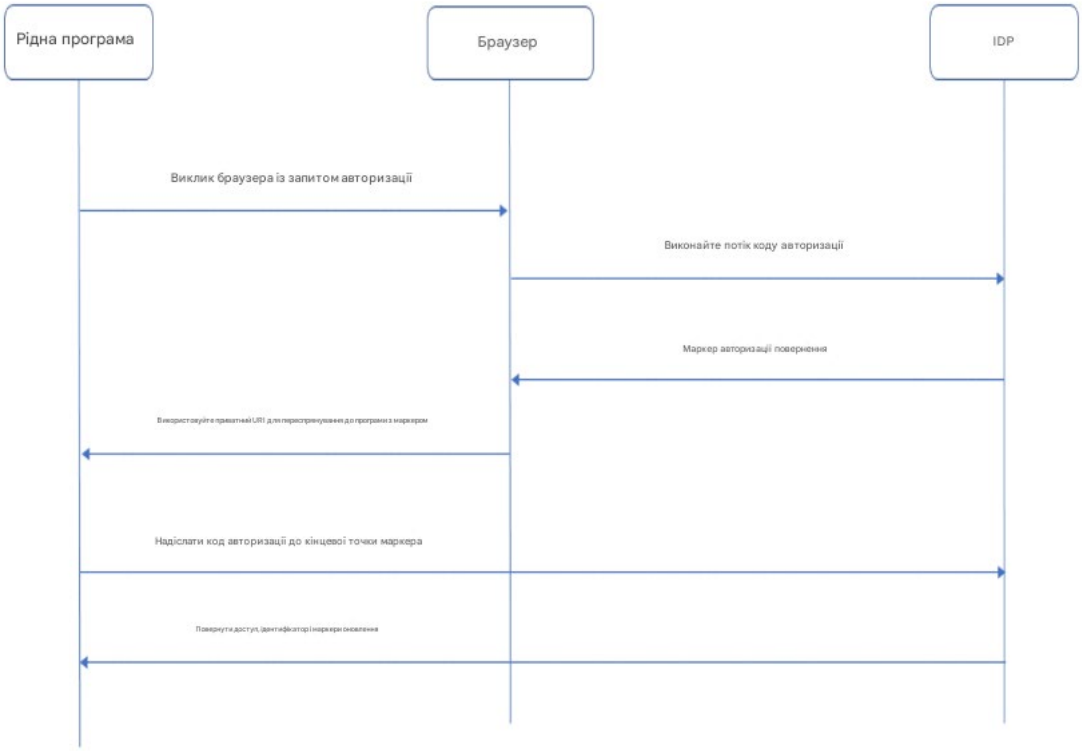

Рисунок 2.15 – Взаємодія рідної програми, браузера та IdP

Це здійснюється за допомогою специфічних для платформи API, які дозволяють одному додатку взаємодіяти з іншим. Використання браузерного додатка є вигідним, оскільки це дозволяє додаткам використовувати єдину систему аутентифікації (SSO). Більшість IdP зберігають стан сеансу

користувача в куках браузера. Завдяки цьому користувач може бути автоматично авторизований на будь-яким іншим додатком, який використовує той самий IdP. З іншого боку, якщо нативний додаток обирає виконати кроки аутентифікації самостійно без браузера, він зіткнеться з обмеженнями при взаємодії щодо стану сеансу користувача з іншими додатками.

Операційні системи дозволяють нативним додаткам використовувати веб-перегляди, які є екземплярами браузера, що можуть бути створені всередині контексту додатка. Використання веб-переглядів настійно не рекомендується, оскільки нативний додаток має повний доступ до контексту веб-перегляду. Він може записувати будь-яку діяльність, що відбувається всередині веб-перегляду, а також читати куки. Замість веб-переглядів нативні додатки повинні використовувати зовнішні надійні браузери.

Перенаправлення з браузера до нативного додатку за допомогою авторизаційного коду здійснюється за допомогою URI–схем для приватного використання. Кожному додатку операційна система надає URI–схему. Як правило, ці схеми є зворотними до доменних імен веб-сайтів. Рекомендується, щоб нативні додатки використовували подібну схему, наприклад, додаток із доменом app.example.com повинен використовувати URI–схему, як–от com.example.app. Операційні системи дозволяють додаткам взаємодіяти за допомогою цих приватних URI–схем. Перенаправлення після аутентифікації з IdP повинно бути за приватною URI–схемою. Коли браузер намагається перенаправити, використовуючи таку URI, операційна система обробляє URI та відкриває додаток, який зареєстрований для цієї URI.

Нативні додатки повинні використовувати Implicit flow з PKCE. Секретний перевірник, який генерує клієнтський додаток у цьому підході, допомагає захистити повторне використання авторизаційного коду, якщо він перехоплений іншим додатком, зареєстрованим на ту саму приватну URI. Тільки додаток, який має секретний перевірник, отримає токени.

Висновки за розділом 2

У даному розділі було проведено детальний аналіз методів розробки та управління сучасними веб-додатками, а також обґрунтування вибору оптимального підходу до їх впровадження. Розглянуто ключові аспекти, що визначають функціональність, безпеку та зручність використання програм.

Особливу увагу приділено порівнянню односторінкових додатків (SPA) та традиційних веб-програм. Визначено, що SPA надають значні переваги в контексті швидкості взаємодії та користувацького досвіду, оскільки не вимагають повного перезавантаження сторінок. Водночас традиційні додатки залишаються актуальними для проєктів, де критично важлива обробка на серверній стороні.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Отримання даних

Для збору інформації, яка могла б допомогти відповісти на дослідницьке питання, були створені пошукові рядки. Якщо база даних дозволяла, використовувалися булеві оператори AND та OR. Оператор OR дозволяє включати альтернативні слова чи синоніми, а оператор AND об’єднує два або більше слова чи фрази.

X складалася із синонімів або слів, які можуть бути пов’язані з класифікацією:

X: {Classification OR taxonomy OR grouping}

Y складалася із синонімів або слів, які можуть бути пов’язані з техніками автентифікації, причому всі слова об’єднані оператором "OR".

Y: {Authentication techniques OR authentication methods}

Нарешті, оператор "AND" було додано між X та Y, щоб знайти релевантну літературу, пов’язану з класифікацією технік автентифікації. Пошук був обмежений заголовками та анотаціями публікацій.

Пошуковий рядок виглядав так:

((((("Document Title":classification OR taxonomy OR grouping) AND "Document Title":"authentication techniques" OR "authentication methods") OR "Abstract":classification OR taxonomy OR grouping) AND "Abstract":"authentication techniques" OR "authentication methods")

Використання цього рядка дало лише вісім результатів, що не допомогло значно просунутися у відповіді на дослідницьке питання. Тоді пошук було розширено до всіх метаданих кожної публікації.

Для цього був використаний наступний пошуковий рядок:

((classification OR taxonomy OR grouping) AND ("authentication techniques" OR "authentication methods"))

3.2 Процес включення

Пошукові запити, розроблені в попередньому розділі, були співставлені із заголовками та анотаціями публікацій, опублікованих за останнє десятиліття (2002–2013). У результаті пошуку за трьома джерелами даних було знайдено 69 статей. Ми переглянули анотації та вступи цих статей і відхилили 45 як нерелевантні. Решта 25 статей були визначені як релевантні для дослідження.

З цих 25 статей чотири виявилися дублікатами, тобто ті самі статті були знайдені у різних джерелах даних. Ці дублікати були вилучені, що залишило 21 статтю. Дві статті були недоступні у електронних джерелах даних, тому залишилося 18 повнотекстових статей.

Ми також знайшли ще 10 статей в інших джерелах, таких як посилання на роботи зі схожих тем або стандартний пошук в інтернеті. Це включало керівництва, книги тощо, що довело загальну кількість до 29 статей. З них 10 були визнані нерелевантними, і 19 статей були відібрані як потенційно корисні для дослідження.

Зрештою, ці 19 статей були проаналізовані шляхом повного прочитання їхнього змісту, щоб визначити, чи мають вони цінність для відповіді на дослідницьке питання. У результаті чотири статті були визнані нерелевантними, і лише 15 статей виявилися безпосередньо пов’язаними з класифікацією технік автентифікації.

3.3 Алгоритм розв’язання задачі

Більшість виключених результатів були зосереджені на конкретній техніці, а не на класифікаціях технік. Оскільки слово "класифікація" також використовується як термін у галузі інженерії даних, деякі статті були вилучені, оскільки вони стосувалися аналізу даних, а не автентифікації користувачів. Інші статті були зосереджені на автентифікації кібернападів, що не є темою нашого дослідження.

Висновки за розділом 3

У цьому розділі розглянуто програмну реалізацію розробленого рішення, яка включає отримання даних, процес їх обробки та застосування алгоритму для розв’язання поставленої задачі.

На етапі отримання даних було реалізовано механізми збору інформації з використанням сучасних інструментів та технологій, які забезпечують швидкість та надійність передачі даних. Особлива увага приділена оптимізації цього процесу для мінімізації затримок та помилок при обробці великих обсягів інформації.

Процес включення передбачає інтеграцію отриманих даних у загальну систему. Розглянуто способи зберігання та структуризації даних з урахуванням специфіки завдань, що вирішуються. Було впроваджено підходи, які дозволяють ефективно управляти даними, забезпечуючи їхню доступність та безпеку.

4 РЕЗУЛЬТАТИ ОБЧИСЛЮВАЛЬНОГО ЕКСПЕРИМЕНТУ ТА ЇХ АНАЛІЗ

4.1 Постановка експерименту

Статичний аналіз автентифікації (Focardi; 2005). У цій статті автор обговорює протоколи автентифікації, серед іншого. Сутність автентифікації зводиться до надійного підтвердження особи. Автори класифікують техніки автентифікації на три групи. Важливим моментом є те, що ці техніки часто комбінуються для спрощення використання або підвищення безпеки.

Класифікації, використані в цій роботі:

– те, що ви знаєте;

– те, що ви маєте;

– те, що властиве вам.

Підвищення безпеки системи Pass Points із використанням змінного допуску (M. Samuel; 2010). У цій статті автор описує техніку автентифікації Pass Points, яка базується на використанні графічних зображень. Як показано на рисунку 1, техніки автентифікації класифікуються на три категорії:

– токен-базована автентифікація;

– біометрична автентифікація;

– автентифікація на основі знань.

Автентифікація на основі знань ділиться на дві підкатегорії:

– автентифікація на основі тексту;

– автентифікація на основі зображень.

Користувацька автентифікація за допомогою динаміки клавіатури з використанням фіксованого тексту (Mariusz, Piotr, Khalid; 2010). У цій статті пропонується підхід до ефективної автентифікації користувачів за допомогою динаміки клавіатури, використовуючи короткий фіксований текст, наприклад, під час процесу входу в систему. Автори обговорюють сучасний стан динаміки клавіатури, а також запропоновану методику, експериментальні результати та

перспективи.

Класифікація технік автентифікації в цій статті:

– автентифікація на основі пам’яті;

– токен-базована автентифікація;

– біометрична автентифікація.

Використання Байєсівської моделі (таблиця 4.1) для оцінки надійності рішень у мультимодальних біометричних системах (Maple, Schetinin; 2006). У цій статті обговорюється, як використання моделей рішень у Байєсівській методології може покращити автентифікацію в мультимодальних біометричних системах.

Таблиця 4.1 – Порівняння аспектів зручності використання Байесівської моделі

	Пін	Пароль	Патерн розблокування	Розблокування обличчям	Надійний замок
Сенсорний екран	0	–	+	+	+
Тривалість	+	–	+	+	0
Складність	+	–	0	+	0
Надійність	+	+	+	–	+

Класифікація методів автентифікації:

– токен–базовані (те, що ви маєте);

– знання–базовані (те, що ви знаєте);

– біометричні.

Біометрична автентифікація поділяється на дві підкатегорії:

– статична (фізичні атрибути);

– динамічна (поведінкові атрибути).

4.2 Аналіз продуктивності різних механізмів автентифікації

Динамічна автентифікація: потреба, а не вибір (Saxena; 2008). У цій статті автори обговорюють різні схеми автентифікації, їхні переваги та недоліки, а також пропонують реалізовану схему двофакторної динамічної одноразової автентифікації за допомогою мобільного пристрою. Автор розділяє схеми автентифікації паролів на два типи: на основі хеш–функцій та на основі відкритих ключів, залежно від обчислювальної складності.

Класифікація автентифікації:

– те, що хтось має (смарт-картка, токен або посвідчення особи);

– те, що хтось знає (пароль або PIN-код);

– те, ким хтось є (відбиток пальця);

– будь-яка комбінація цих факторів.

Автентифікація на основі питань із використанням контекстних даних (Nosseir, Connor, Revie, Terzis; 2006). У цій статті автори пропонують схему автентифікації на основі питань, яка підходить для ситуацій із низьким рівнем ризику. Вони проводять експеримент для дослідження того, чи можуть історії, створені «розумними» середовищами на основі контексту їх мешканців, використовуватися для розрізнення між справжніми користувачами та зловмисниками. Автори зазначають, що техніки автентифікації класифікуються за механізмами:

– хто ви є;

– що ви маєте;

– що ви знаєте.

Повторна автентифікація користувача через рухи миші (Pusara, Brodley; 2004). У цій статті автори пропонують підхід до повторної автентифікації користувачів на основі даних, отриманих від рухів комп’ютерної миші. Автори зазначають, що автентифікація може досягатися за допомогою:

– те, що користувач знає (наприклад, паролі доступу, PIN-коди);

– те, що користувач має (наприклад, токени доступу, значки ідентифіка-

ції, смарт-карти, бездротові ідентифікаційні агенти);

– те, ким користувач є (наприклад, відбиток пальця, зразок голосу, візерунок райдужки).

Автентифікація AwareLESS: автентифікація на основі непомітного введення (Manabe, Fukumoto; 2007). У цій статті автори пропонують методику автентифікації "awareLESS" для підвищення безпеки портативних пристроїв. Вони зазначають, що користувач може бути автентифікований за допомогою кількох факторів:

– того, що він має (наприклад, IC-картка, ID-мітка);

– біологічних чи поведінкових характеристик (наприклад, відбиток пальця або жести);

– того, що він знає (наприклад, пароль або PIN-код).

Автори також зазначають, що техніки, засновані на власності, є вразливими до крадіжки або втрати.

Автентифікація електронних учнів із використанням мультимодальної біометричної технології (Asha, Chellappan; 2008). У цій статті автори обговорюють автентифікацію в системах електронного навчання за допомогою мультимодальної біометричної технології. Вони розрізняють два базові типи біометричних систем:

– одномодальні біометричні системи;

– мультимодальні біометричні системи.

Одномодальні системи використовують лише одну біометричну характеристику для ідентифікації. Мультимодальні системи залучають декілька біометричних характеристик одночасно.

4.3 Вплив архітектури додатку на ефективність автентифікації

Огляд біометричної автентифікації (Bhattacharyya, Ranjan, Farkhod, Choi; 2009). У цій статті автори надають огляд технік біометричної автентифікації та

можливостей у цій сфері. Біометрична система може виконувати дві функції: перевірку (verification) та автентифікацію (authentication). Техніки біометричної автентифікації повинні бути достатньо надійними, щоб забезпечувати обидві функції одночасно. Також розробляються інші біометричні стратегії, такі як аналіз ходи (gait), сітківки ока, вен долоні, вушного каналу, теплового зображення обличчя, ДНК, запаху та відбитків долоні.

Автори класифікують біометричні системи на два типи:

– фізіологічні;

– поведінкові.

Фізіологічні системи вважаються більш надійними, оскільки індивідуальні особливості людини, що використовуються цими системами, не змінюються під впливом психоемоційного стану. До фізіологічних характеристик належать: відбитки пальців, розпізнавання райдужної оболонки ока, геометрія долоні, ДНК, розпізнавання обличчя, візерунок вен долоні.

Поведінкові методи враховують дії людини, надаючи користувачеві можливість контролювати свої дії. Біометрія, заснована на цих методах, враховує високий рівень внутрішніх змін (настрій, стан здоров'я тощо), тому ці методи корисні лише при постійному використанні. До поведінкових характеристик належать голос, хода, ритм друкування.

Графічний пароль із використанням токенів, біометрії та автентифікації на основі знань для мобільних пристроїв (Patil, Shimpi; 2013). У цій статті автори обговорюють графічний пароль як перспективну альтернативу текстовим паролям. Відповідно до психології людини, зображення запам'ятовуються легше, ніж текст. Автори пропонують нову гібридну систему графічних паролів, яка поєднує техніки впізнавання та запам'ятовування, забезпечуючи численні переваги порівняно з існуючими системами.

Класифікація методів автентифікації, запропонована авторами:

– методи, засновані на токенах;

– методи, засновані на біометрії;

– методи, засновані на знаннях.

Біометрія поділяється на дві категорії:

– контактна;

– безконтактна.

Рекомендації з електронної автентифікації (NIST; 2011). У цьому документі автори описують класичну парадигму для систем автентифікації, яка включає три основні фактори:

– щось, що ви знаєте (наприклад, пароль);

– щось, що ви маєте (наприклад, бейдж ідентифікації або криптографічний ключ);

– щось, ким ви є (наприклад, відбиток пальця або інші біометричні дані).

Також зазначено, що багатофакторна автентифікація використовує більше ніж один із перелічених факторів. Надійність систем автентифікації здебільшого визначається кількістю використаних факторів. Системи, які використовують два фактори, вважаються надійнішими за ті, що використовують лише один; а системи, що включають усі три фактори, є найнадійнішими.

Безпечна автентифікація для Інтернет–банкінгу (Hiltgen, Kramp, Weigold; 2006). У цій статті автори представляють два рішення для автентифікації в Інтернет-банкінгу за принципом «виклик-відповідь» – одне на основі коротких рядкових паролів, а інше на основі сертифікатів. Автори описують, як ці рішення можна розширити у разі виникнення складніших атак на контент.

Класифікація методів автентифікації для Інтернет-банкінгу виконується відповідно до їхньої стійкості до двох типів поширених атак:

– офлайн-атаки, спрямовані на крадіжку облікових даних;

– онлайн-атаки, спрямовані на порушення каналу зв’язку.

Автентифікація користувачів на мобільних пристроях (Sethi, Manzoor, Sethi; 2012). У цій статті автори обговорюють, як автентифікація забезпечує надійне підтвердження особи заявника. Вони зазначають, що техніки часто поєднуються для спрощення використання або підвищення безпеки. Автори класифікують техніки автентифікації на три групи:

– щось, що ви знаєте;

– щось, що ви маєте;

– щось, ким ви є.

Контекстно–залежна мобільна біометрична автентифікація на основі методів опорних векторів (Witte, Rathgeb, Busch; 2013) У цій статті автори пропонують контекстно–залежну мобільну біометричну систему. Як показано на рисунку 3, вони пропонують систему, у якій оцінки довіри, отримані з (біометричних) контекстних даних, використовуються для прийняття рішень або налаштування подальших систем автентифікації.

Класифікація методів автентифікації:

– методи на основі біометрії;

– методи на основі знань.

Висновки за розділом 4

У цьому розділі, присвяченому результатам обчислювального експерименту та їх аналізу, було проведено оцінку ефективності розробленого рішення на основі практичного тестування. Експериментальні дослідження дозволили перевірити працездатність алгоритмів, визначити їх продуктивність та оцінити точність отриманих результатів у різних сценаріях використання.

Аналіз отриманих даних показав, що запропоноване рішення демонструє високу швидкість виконання завдань та стабільність навіть при значних обсягах вхідної інформації. Також було підтверджено відповідність результатів очікуваним, що свідчить про правильність реалізації алгоритмів.

Особлива увага приділялася оцінці ключових параметрів, таких як час виконання, використання системних ресурсів та масштабованість системи. Порівняння з аналогічними рішеннями продемонструвало конкурентні переваги розробленої системи, включаючи її ефективність та гнучкість у налаштуванні під специфічні вимоги.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено комплексне дослідження технік автентифікації, їх класифікацій та особливостей використання в сучасних інформаційних системах. Результати роботи дозволяють зробити такі висновки:

а) Різноманітність технік автентифікації. У процесі аналізу було виявлено, що сучасні методи автентифікації можна умовно розділити на три основні категорії:

1) щось, що ви знаєте (паролі, PIN-коди, секретні питання).

2) щось, що ви маєте (токени, смарт-карти, мобільні пристрої).

3) щось, ким ви є (біометричні дані, такі як відбитки пальців, голос або райдужка ока). Деякі техніки комбінуються для підвищення рівня безпеки, наприклад, у схемах багатофакторної автентифікації.

б) Класифікації методів автентифікації. У різних дослідженнях було запропоновано класифікації, які базуються на різних підходах: знаннях, біометрії, токенах або контекстних даних. Це дозволяє систематизувати існуючі техніки та обрати оптимальний метод залежно від конкретного сценарію використання.

в) Переваги та недоліки різних методів. Кожна категорія технік має свої сильні та слабкі сторони. Наприклад:

1) паролі є простими у використанні, але вразливі до атак, спрямованих на вгадування або крадіжку облікових даних;

2) біометрія забезпечує високий рівень надійності, проте може вимагати спеціалізованого обладнання та викликає занепокоєння щодо конфіденційності;

3) токени надають зручність, але вразливі до крадіжок або втрат.

г) Перспективи розвитку. У сучасному світі зростаючих кіберзагроз перспективними є мультимодальні системи автентифікації, які використовують декілька факторів для підвищення рівня безпеки. Також важливим напрямом є використання контекстно-залежних даних для адаптації методів автентифікації

до потреб користувача.

Практичне застосування результатів. Проведене дослідження та розроблена класифікація можуть бути використані для створення нових систем автентифікації або вдосконалення існуючих. Запропоновані рекомендації щодо вибору технік залежно від рівня ризику та специфіки середовища дозволяють оптимізувати процес автентифікації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Stallings W. Cryptography and Network Security: Principles and Practice. Pearson Education, 2018. 840 p.

2. Rathgeb C., Uhl A. Biometric Authentication and Multimodal Systems. Springer, 2011. 236 p.

3. RFC 6749. The OAuth 2.0 Authorization Framework. Internet Engineering Task Force (IETF). URL: https://www.rfc-editor.org/info/rfc6749 (дата звернення: 16.10.2024).

4. OpenID Foundation. OpenID Connect Core. URL: https://openid.net/specs/openid-connect-core-1_0.html (дата звернення 14.11.2024).

5. NIST Special Publication 800–63B. Digital Identity Guidelines: Authentication and Lifecycle Management. NIST, 2017. 93 p.

6. Bhattacharyya D., Ranjan R., Alisherov F., Choi M. Biometric Authentication: A Review. *International Journal of u- and e-Service, Science, and Technology*, 2009. Vol. 2, pp. 13–28.

7. Jain A. K., Ross A., Prabhakar S. An Introduction to Biometric Recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 2004. ol. 14, pp. 4–20.

8. Dasgupta D., Roy A., Nag A. Advances in User Authentication. Springer, 2017. 360 p.

9. Abid A., Shuja A., Malik N. Context-Aware Authentication Systems: A Survey. *Journal of Network and Computer Applications*. 2018. Vol. 108. P. 79–104.

10. Multifactor Authentication Market Trends. Global Market Insights Report. URL: https://www.gminsights.com (дата звернення: 29.12.2024).