Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
завідувач кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
“14” червня 2022 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної роботи
бакалавра
(назва освітнього ступеня)

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 «Кібербезпека»
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: «Підсистема авторизація з використанням мобільних пристроїв для Web-застосунків»

Виконавець: студент 4 курсу, групи КБ-41

_____ **Анна ДЕНИСЕНКО**
(підпис) (ім’я прізвище)

	Прізвище, ініціали	**Підпис**
Керівник роботи	Іван ПАРХОМЕНКО	

Нормоконтроль	Сергій ДАКОВ	

Київ 2022

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідувач кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
«01» листопада 2021 р.

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студентці **КБ-41** **Анні Денисенко**
(група) (прізвище ім'я по-батькові)

Тема дипломної роботи Підсистема авторизації з використанням мобільних пристроїв для Web- застосунків

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №5 від 29.10.2021 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Структури, архітектури, засоби функціонування web-застосунків, процес надання доступу до web-застосунків, алгоритми хешування та шифрування

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Нормативно-правова база у сфері захисту інформації, архітектура web-застосунків, механізми взаємодії web-застосунків з мобільними застосунками, процес надання доступу до інформації та поширені загрози, види методів автентифікації, асиметричне шифрування, архітектура підсистеми, мобільний застосунок

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Архітектура підсистеми авторизації до web-застосунку з використанням мобільних пристроїв на основі асиметричного шифрування

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 29.10.2021 року

Завдання видав _____ (підпис) Іван ПАРХОМЕНКО (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Анна ДЕНИСЕНКО (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	29.10.2021 – 4.11.2021	*виконано*
2	Аналіз літератури	28.01.2022 –20.02.2022	*виконано*
3	Розгляд архітектури web-застосунків	24.02.2022 – 04.03.2022	*виконано*
4	Дослідження механізмів взаємодії web- та мобільних застосунків	05.03.2022 – 24.03.2022	*виконано*
5	Огляд процесу надання доступу до застосунків	25.03.2022 – 07.04.2022	*виконано*
6	Дослідження методів автентифікації	07.04.2022 – 12.04.2022	*виконано*
7	Дослідження вразливостей процесу надання доступу	12.04.2022 – 16.04.2022	*виконано*
8	Опис розроблюваного рішення	17.04.2022 – 20.04.2022	*виконано*
9	Побудова підсистеми авторизації з використанням мобільних пристроїв	21.04.2022 – 09.05.2022	*виконано*
10	Реалізація мобільного застосунку для підсистеми	10.05.2022 – 04.06.2022	*виконано*
11	Оформлення пояснювальної записки	05.06.2022 – 08.06.2022	*виконано*

Завдання видав _____ (підпис) Іван ПАРХОМЕНКО (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Анна ДЕНИСЕНКО (ім'я, прізвище)

Термін подання дипломної роботи до ЕК 06 червня 2022 року

УДК 004.056.5

РЕФЕРАТ

Пояснювальна записка дипломної роботи складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел та додатків. Основний текст займає 71 сторінок, включає в себе зміст, вступ, три розділи дипломної роботи, висновки та список джерел. Крім того, робота містить 2 додатки із загальною кількістю сторінок 4. У пояснювальній записці дипломної роботи міститься 19 рисунків і 3 таблиці.

Метою роботи є побудова підсистеми авторизації для web-застосунків з використанням мобільних пристроїв.

Об'єктом дослідження є процес надання доступу до web-застосунків.

Предметом дослідження є набір механізмів, що реалізують процес надання доступу до web-застосунків.

Методи дослідження:

- аналіз відкритих джерел;
- порівняння методів автентифікації;
- моделювання підсистеми авторизації;

Практичною цінністю є розроблений програмний модуль для мобільного застосунку для двохфакторної автентифікації з синхронізацією з web-застосунком.

Новизна: розроблена архітектура підсистеми авторизації для web-застосунків на основі асиметричного шифрування з використанням мобільних пристроїв, як елементу двохфакторної автентифікації.

Ключові слова: web-застосунок, архітектура застосунків, мобільні застосунки, автентифікація, вразливості, захист персональних даних, облікові дані, мобільні пристрої, Kotlin.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕННЬ

АС – автоматизована система;

БА – базова автентифікація;

ЕОМ – електронно-обчислювальна машина;

ЗЗІ – засоби захисту інформації;

ЗІ – захист інформації;

ІБ – інформаційна безпека;

ІС – інформаційна система;

ІТ – інформаційні технології;

КСЗІ – комплексна система захисту інформації;

КЦД – конфіденційність, цілісність, доступність;

НД – нормативний документ;

НСД – несанкціонований доступ;

ОІД – об’єкт інформаційної діяльності;

ПЗ – програмні засоби, програмне забезпечення;

PKI – Public Key Infrastructure;

REST – Representational State Transfer;

RPC – Remote Procedure Call;

SOAP – Simple Object Access Protocol;

СЗІ – система захисту інформації.

ВСТУП

З середини XX століття життя людей кардинально змінилось, оскільки з'явилися перші комп'ютери (електронно обчислювальні машини). Вони, звісно, максимально відрізнялися від сучасних комп'ютерів, проте зробили вкрай важливу річ – дали поштовх до розвитку інформаційних технологій. По мірі прогресу, все більше людей почали володіти девайсами різного роду. Ледве не кожна людина, що проживає сьогодні в майже будь-якій країні, має власний або спільний з кимось комп'ютер. За даними Світового банку, станом на кінець 2021 року, 97% населення усієї планети використовує мобільні телефони, в тому числі смартфони, цікаво, що ця кількість перевищує кількість людей, що мають доступ до базових санітарних послуг. У 2000 році тільки 12% людей користувалися мобільними телефонами, тобто за 21 рік частка зросла більше ніж у 8 разів [1]. Digital 2021– звіт компаній We Are Social та Hootsuit вказує, що станом на зараз у світі проживає приблизно 7.83 мільярда людей, з яких доступ до Інтернету має 4.66 мільярдів, тобто приблизно 54% населення. З 2020 року по 2021 рік приріст користувачів Інтернету становить 7.5%, що свідчить про вкрай стрімку тенденцію збільшення загальної кількості користувачів [2]. Можна зробити припущення, що цей відсоток приросту з кожним роком буде тільки збільшуватися, оскільки смартфони стають все доступнішими для людей і навіть за $100 можна придбати простенький телефон з можливістю доступу до Інтернету.

Очевидним є також те, що наше суспільство вже давно та остаточно перетворилося з індустріального на інформаційне. Люди створюють роботів на нескладних мікросхемах для того аби замінити важку працю. Все це тому, що людина заради своєї зручності та економії часу перекладає величезну кількість повсякденних задач на ті чи інші електронні девайси. Інформаційне суспільство стало новою епохою, в якій люди вже просто не можуть навіть уявити своє існування без постійного підживлення мозку будь-якою інформацією. Ми, в якійсь мірі, стали залежними від інформації, наразі людину від статті про квантову

криптологію відділяє всього-на-всього кілька хвилин пошуку в Google. Для цього навіть з'явився окремий термін – гуглити. Google обробляє близько 63 тисяч пошукових запитів за 1 секунду, або 3.8 мільйона за 1 хвилину або 5.5 мільярдів запитів за день.

За статистикою на початок 2022 року, в США користувач в середньому за день відвідує близько 100 web-сторінок. Наразі web-застосунки та сайти є ключовою точкою контакту людей з інформацією в інформаційному просторі, і споживачі добре знають, чого очікують від них. За 50 мілісекунд відвідувачі формують думку про web-застосунок, тому так важливо створювати системи, які справляють позитивне враження [3].

Незмінним є те, що чим більше людина взаємодіє з інформаційним простором, тим більше персональних даних так чи інакше фігурують у цих взаємовідносинах. Багато Інтернет ресурсів та web-застосунків потребують створення акаунтів, обумовлюючи це персоналізованим підходом до кожного користувача. Певно у кожного користувача Інтернету на сьогодні є створений обліковий запис у Google, який дає змогу авторизуватися у більшість web-застосунків. В даному випадку Google гарантує конфіденційність персональних та облікових даних, які передаються, хоча навіть це не зупиняє зловмисників від зливу паролів до облікових записів користувачів. Не дивлячись на постійні заклики створювати складні паролі, на сьогодні найбільш популярним паролем являється «123456», який використовують понад 23 мільйони людей [4]. Цей факт є гарною мотивацією для створення додаткових заходів безпеки під час процесу авторизації, аби убезпечити облікові дані, та інформацію до якої отримується доступ. Посилення безпеки на цьому етапі можливе за допомогою різних способів, одним з яких є підсистема авторизації, яка по суті являється використанням багатофакторної автентифікації. Тож ***актуальність роботи*** полягає в розробці рішення для процесу авторизації, що одночасно забезпечує достатній рівень безпеки та є зручним як для web-застосунків, так і для їх користувачів.

Метою роботи є побудова підсистеми авторизації для web-застосунків з використанням мобільних пристроїв.

Для досягнення зазначеної мети дипломної роботи поставлено наступні **завдання**:

- дослідити архітектуру web-застосунків та механізми взаємодії з мобільними застосунками;
- провести аналіз процесу надання доступу, методів автентифікації та їх вразливостей;
- побудувати архітектуру підсистеми авторизації із використанням мобільних пристроїв;
- описати програмну реалізації мобільного застосунку для підсистеми.

Об'єктом дослідження є процес надання доступу до web-застосунків та захист персональних даних.

Предметом дослідження є механізми та засоби, реалізації методів захисту web-застосунків.

Методи дослідження:

- аналіз відкритих джерел;
- порівняння методів автентифікації;
- моделювання підсистеми авторизації.

Практична цінність роботи полягає в наступному:

- розробці архітектури підсистеми авторизації до web-застосунків з використанням мобільних пристроїв на основі алгоритму асиметричного шифрування RSA;
- реалізації програмного модуля найважливіших механізмів функціонування мобільного застосунку для розроблюваної підсистеми на мові програмування Kotlin для пристроїв на базі ОС Android.

РОЗДІЛ 1
ОПИС АРХІТЕКТУРИ WEB-ЗАСТОСУНКІВ ТА МЕХАНІЗМІВ ВЗАЄМОДІЇ

1.1 Нормативно-правова база для функціонування WEB-застосунків

Розглядаючи питання інформаційної безпеки, основоположним нормативно-правовим документом є Закон України «Про інформацію», який регулює відносини щодо створення, збирання, одержання, зберігання, використання, поширення, охорони та захисту інформації [5]. Прямо чи опосередковано, будь-яка web-система постійно взаємодіє з інформацією, у більшості випадків її основною задачею являється обробка цієї самої інформації. Цей закон дуже чітко визначає рамки інформаційних відносин, позначаючи суб'єктами відносин: фізичних осіб, юридичних осіб, об'єднання громадян та суб'єкти владних повноважень; об'єктом інформаційних відносин – саму інформацію. В контексті функціонування web-застосунку суб'єктами інформаційних відносин є: користувачі застосунку, організація-власник застосунку, провайдери, що надають послуги підтримки та, при необхідності, державні органи, що регулюють відносини. Об'єктом відносин є інформація, що збирається, обробляється та циркулює у web-застосунку.

Закон класифікує інформацію з обмеженим доступом на конфіденційну, таємну та службову. Конфіденційною є інформація про фізичну особу; інформація, доступ до якої обмежено фізичною або юридичною особою, крім державних органів. Конфіденційна інформація може поширюватися за бажанням (згодою) відповідної особи у визначеному нею порядку. В контексті web-застосунку конфіденційна інформація – це будь-які особисті дані користувачів системи, наприклад паспортні дані, адреси місця реєстрації, водійські посвідчення, ідентифікаційні номери платників податків, а в призмі пандемії COVID-19, ще й дані сертифікатів про вакцинацію.

Постанова Кабінету Міністрів України «Про затвердження Правил забезпечення захисту інформації в інформаційних, телекомунікаційних та

інформаційно-телекомунікаційних системах» окреслює вимоги до захисту та обробки інформації [6]. Постанова визначає та підпорядковує захисту у системі такі види інформації: відкрита інформація, яка належить до державних інформаційних ресурсів, а також відкрита інформація про діяльність суб'єктів владних повноважень, військових формувань, яка оприлюднюється в Інтернеті, інших глобальних інформаційних мережах і системах або передається телекомунікаційними мережами; конфіденційна; службова та інформація, яка становить державну таємницю [6]. Таким чином, відкрита інформація фактично є публічною, оскільки усі користувачі повинні забезпечуватися доступом для ознайомлення. Детальніше про публічну інформацію у Законі України «Про доступ до публічної інформації», який визначає порядок здійснення та забезпечення права кожного на доступ до інформації [7]. Проте, навіть для відкритої інформації під час обробки у інформаційній системі має зберігатися цілісність аби уникнути умисної модифікації або ж знищення. Для службової або таємної інформації Постанова встановлює набагато жорсткіші умови надання доступу та обробки. Так доступ до такої інформації може надаватися тільки ідентифікованим та автентифікованим користувачам, при цьому у системі обов’язково має здійснюватися запис/логування усіх результатів ідентифікації та автентифікації користувачів для можливості здійснення подальшого моніторингу доступів. Для такої інформації обов’язково має здійснюватися захист від несанкціонованого доступу, тобто компрометації.

Закон України «Про захист інформації в інформаційно-комунікаційних системах» встановлює та більш детально пояснює методи та способи захисту інформації, що циркулює в системі, умови її обробки [8]. Також захист розповсюджується на програмне забезпечення, яке призначено для обробки цієї інформації. Варто зауважити, що для службової та таємної інформації має забезпечуватися технічний та криптографічний захист, засоби криптографічного захисту мають відповідати вимогам засобів, призначених для захисту службової та таємної інформації.

Закон України «Про захист персональних даних» регулює правові відносини, пов’язані із захистом і обробкою персональних даних і спрямований на захист права

людини на невтручання у особисте життя [9]. Закон визначає персональні дані як відомості чи сукупність відомостей про фізичну особу, яка ідентифікована або може бути конкретно ідентифікована. Також Закон регламентує права суб'єкта обробки персональних даних, використання персональних даних, підстави для обробки персональних даних, дії щодо збирання, накопичення та зберігання, поширення, видалення чи знищення персональних даних, порядок доступу до персональних даних. Документ окреслює суб'єктів відносин: «суб'єкт персональних даних» – фізична особа, особисті дані якої обробляються; володілець та розпорядник персональних даних – підприємства, установи і організації усіх форм власності, органи державної влади чи органи місцевого самоврядування, фізичні особи - підприємці, які обробляють персональні дані відповідно до закону. В розрізі web-застосунку суб'єктом персональних даних є будь-який користувач системи, який надає особисті дані. Розпорядником персональних даних є людина чи компанія-власник web-застосунку, що являється провайдером послуг для користувачів та збирає особисті дані, обов'язково за їхньої згоди. Наразі в багатьох сайтах та застосунках при заповнені форми з особистими даними, перед тим як надіслати її, користувач має зазначити, що він дає свою згоду на обробку персональних даних. Також на території Європейського союзу діє «Загальний регламент про захист даних», який вимагає, щоб сайти та web-застосунки в обов'язковому порядку запитували дозвіл користувачів на використання cookies [10].

1.2 Монолітна архітектура

Моноліт - це стародавнє слово, що означає величезний кам'яний блок. Хоча цей термін широко використовується сьогодні, уявлення залишається однаковим у всіх галузях. У програмній інженерії монолітна модель відноситься до єдиної неподільної одиниці. Монолітна архітектура є традиційним підходом для проектування web-застосунків та програмного забезпечення. Додаток в монолітній архітектурі розробляється як єдине ціле – моноліт/пакет. Розробка нескладного

web-застосунку починається з модульної багатошарової архітектури, яка складається з різних типів шарів наступним чином [12]:

- User Interface (Презентаційний шар або Інтерфейс користувача): для користувача найбільш важливий рівень, оскільки відповідає за обробку запитів протоколу передачі HyperText (HTTP) за допомогою HTML або XML / JSON (для API web-сервісів).
- Business logic (Шар бізнес логіки): бізнес-логіка web-застосунку, тобто фактично це структурно оформлена «проблема - ідея», яку вирішує застосунок.
- Database access (Шар доступу до бази даних): цей шар відповідає за доступ до всіх баз даних, які використовує web-застосунок у своїй роботі.
- Application integration (Інтеграційний шар додатків): на цьому рівні відбувається вся інтеграція web-застосунку, з іншими сервісами інтеграція з іншими сервісами (наприклад, через обмін повідомленнями або REST API).

Стандартна схема монолітної архітектури web-застосунку представлена на рисунку 1.1.

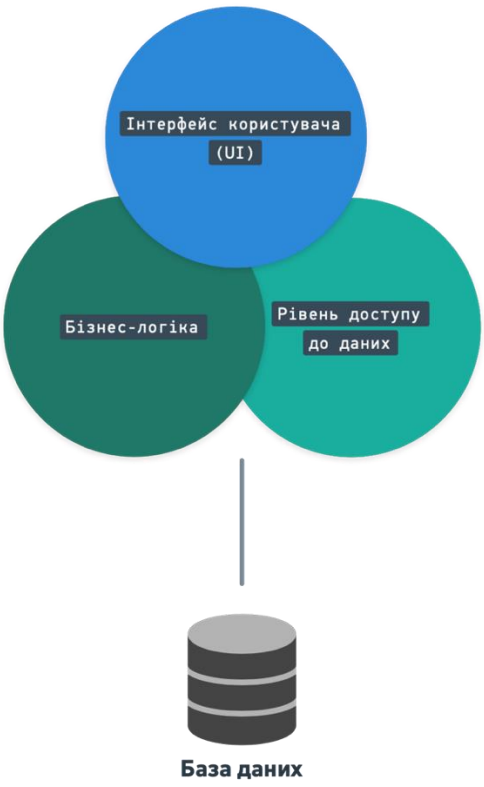


Рисунок 1.1 – Схема роботи web-застосунку з монолітною архітектурою

Навіть попри те, що монолітна архітектура має багатошарову архітектуру, остаточний web-застосунок або будь який продукт розробки будуть зібрані в єдиний моноліт/пакет і потім у середовищі використання будуть розгорнуті таким самим чином. Монолітні застосунки не мають модульності як такої та працюють в рамках єдиної кодової бази. Також варто зауважити, що кожен компонент архітектури і пов'язані з ним компоненти повинні бути присутніми для компіляції кінцевого коду аби створити функціонуючу збірку web-застосунку.

Переваги використання монолітної архітектури [12]:

- Web-застосунки прості у розробці;
- Тестування застосунків значно спрощується, можна реалізовувати наскрізне тестування;
- Для використання створеного застосунку достатньо просто перенести пакет застосунку на сервер;
- Легке масштабування по горизонталі шляхом простого копіювання інстансів та розміщення їх за балансувальником.

Недоліки використання монолітної архітектури:

- Важко вносити зміни. Додаток зібраний у єдиний моноліт, і містить велику кількість коду, що ускладнює розуміння роботи і швидке внесення виправлень;
- Моноліт доволі великий, що прямопропорційно впливає на збільшення часу запуску додатку;
- Компоненти web-застосунки сильно залежні один від одного, що ускладнює впровадження нових технологій, оскільки потрібно буде переписувати весь код, а не якусь частину;
- Складне масштабування. Компоненти застосунку можуть мати суперечливості стосовно використовуваних ресурсів;
- Надійність. Помилка в будь-якому модулі (наприклад, витік пам'яті) потенційно може зруйнувати весь процес. Крім того, оскільки всі екземпляри коду ідентичні, ця помилка вплине на доступність всього застосунку.

На ранніх стадіях розробки монолітна архітектура добре справляється з поставленими задачами, і в основному більшість великих і успішних додатків, які

існують сьогодні, були запущені як моноліт. Проте, монолітні застосунки мають перешкоди щодо впровадження нових технологій, оперативного внесення змін і масштабованості.

1.3 Мікросервісна архітектура

Термін “Мікросервісна архітектура”, також часто використовують як мікросервіси, почали використовувати в середині 2010-х, щоб описати особливий архітектурний стиль розробки web-застосунків та будь яких програмних засобів. Цей стиль побудови та розробки застосунків отримав велику популярність та розповсюдженого використання на фоні розвитку принципу гнучкої розробки та DevOps (Development & Operations). На сьогодні тема мікросервісної архітектури освітлюється у: статтях, блогах, дискусіях в соціальних мережах і презентаціях на конференціях. Дедалі більше компаній у всьому світі використовують цей стиль архітектури при створенні своїх програмних продуктів, а компанії з існуючими програмними рішеннями починають переводи їх на мікросервіси. Коли постає завдання впровадження гнучкої розробки і доставки складних корпоративних додатків — такий спосіб архітектури і розробки вважається одним з найдоречніших.

Мікросервісний архітектурний стиль — це підхід до розробки програмного забезпечення, який характеризується використанням самодостатніх, незалежних , невеликих та слабко пов’язаних між собою сервісів, які для комунікації використовують нескладні механізми, наприклад: HTTP (HyperText Transfer Protocol), gRPC (Google Remote Procedure Call), AMQP (Advanced Message Queuing Protocol) та інші [12]. Ідея полягає в тому, щоб розділити програмний код на набір менших взаємопов’язаних служб-сервісів замість того, щоб створювати єдиний монолітний застосунок. Схематичний приклад роботи мікросервісної архітектури наведено на рисунку 1.2:

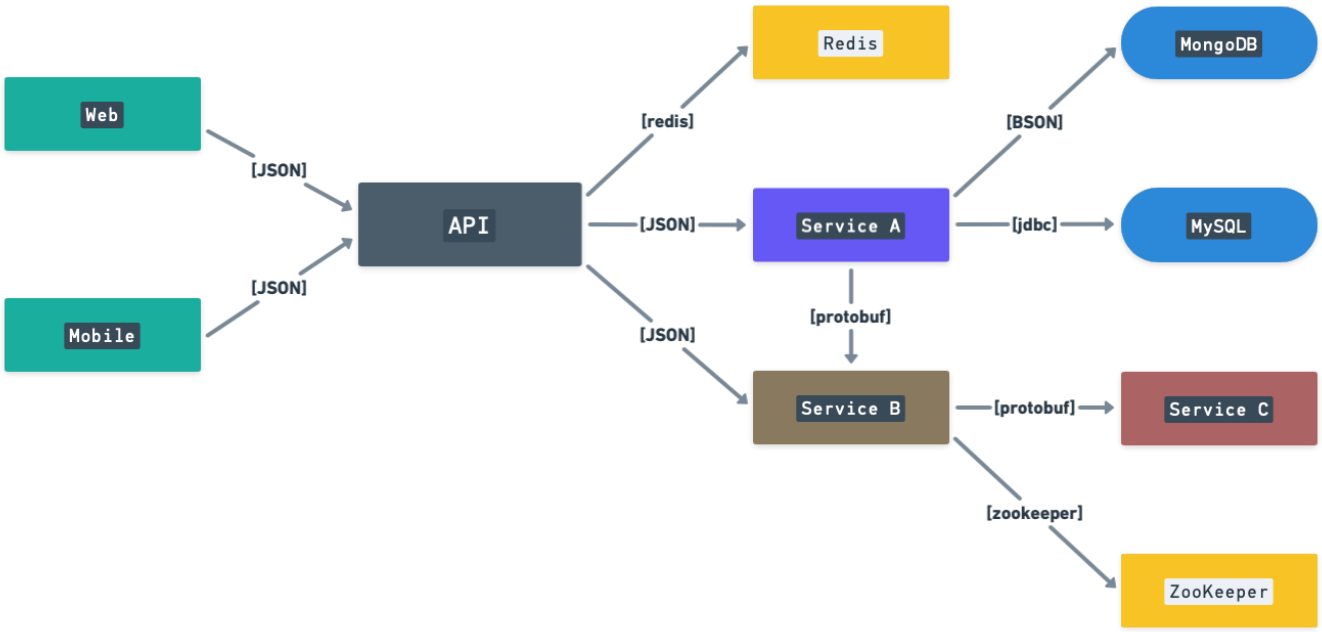


Рисунок 1.2 – Схематичний приклад роботи мікросервісної архітектури

Кожен мікросервіс є невеликим застосунком, який має власну архітектуру, що складається з бізнес-логіки разом із різними адаптерами. Деякі мікросервіси надають REST, RPC або API на основі повідомлень, а більшість сервісів використовують API, надані іншими сервісами. Також набір мікросервісів використовують для реалізації UI (user interface) для web-застосунків. Сервіси будуються відносно бізнес-потреб компанії/продукту, при цьому кожен має свою сферу відповідальності, та розгортаються у повністю автоматизованому середовищі незалежно один від одного. Сервіси можуть мати мінімальне централізоване управління при певних обставинах, можуть бути написані на різних мовах і використовувати різні технології зберігання даних [14].

Фактично, мікросервіси це набір маленьких контейнерів, які мають бути пов’язані між собою, щоб створити повний працюючий додаток. Для того, аби поєднувати мікросервіси між собою розробники використовують API.

Для кращого розуміння монолітної та мікросервісної архітектури нижче наведено порівняльну таблицю 1.1:

Таблиця 1.1

Порівняння монолітної та мікросервісної архітектури

	Монолітна	Мікросервісна
Мова програмування	Додаток повністю розроблений однією мовою програмування.	Кожен сервіс розробляється незалежно і може бути реалізований іншою мовою програмування.
Кодова база	Одна кодова база	Додаток має кілька баз коду. Кожен сервіс має окрему кодову базу.
Зрозумілість	Якщо додаток стає великим, то розуміння ускладнюється і зростає вірогідність заплутатися.	За рахунок дроблення на сервіси додаток стає доволі структурованим і нескладним у розумінні
Масштабування застосунків	Масштабувати застосунок вкрай складно, оскільки потрібно реплікувати весь програмний код, тобто весь застосунок.	Масштабувати застосунок дуже просто, оскільки за рахунок автономності, кожен сервіс може масштабуватися окремо, не зачіпаючи при цьому весь код додатку.
Розробка та розгортання	Оскільки додаток це єдиний пакет-моноліт, то розробка та інтеграція нового функціоналу призведе до змін усього додатку вцілому. З розгортанням аналогічно.	Сервіси дозволяють легко та гнучко постійно вносити зміни у додаток без зайвих труднощів, оскільки додаток збирається як конструктор з блоків.

Продовження таблиці 1.1

Запуск сервісу	Монолітні додатки об’ємні та важкі, що збільшує час запуску послуги	Мікросервісна архітектура значно пришвидшує запуск послуг додатку
Модель даних	Централізована модель даних	Об'єднана модель даних, що дозволяє кожній службі приймати власну модель даних.
Послідовність та доступність	Менш послідовний і доступний, оскільки будь-яке оновлення вимагатиме процесу розробки з нуля.	Легко доступні та висока послідовність.

Варто зауважити, що однією з ключових причин використання мікросервісів є те, що компанії хочуть мати змогу в край короткі терміни швидко вносити зміни. Світ розвивається доволі динамічно і компанії повинні бути гнучкими, щоб швидше реагувати на зміни вимог бізнесу, мати перевагу серед конкурентів аби отримати та утримати увагу і гроші користувачів. Мікросервіси допомагають впроваджувати зміни швидше, безпечніше і з більш високою якістю, тобто не сповільнювати швидкість розвитку продукту, навіть коли він розростається до вкрай великих розмірів. Це все досягається завдяки автономності сервісів, що дає можливість швидше та частіше виправляти помилки або впроваджувати новий функціонал дозовано. Також великою перевагою є те, що якщо в роботі застосунку стається збій, наприклад при оновленні, доволі швидко можна зрозуміти, які саме мікросервіси працюють некоректно, ізолювати їх та відновити роботу системи на попередній версії мікросервісу. В цей час буде можливість виправляти існуючі помилки.

При всьому цьому не потрібно забувати що такий підхід додає додаткову складність проекту в цілому. Як мінімум необхідні додаткові спеціалісти – DevOps’и для моніторингу та управління, при цьому між ними і розробниками

повинні бути тісні відносини і хороша взаємодія. Програмне забезпечення для розгортання мікросервісів, таке як Docker або Kubernetes, потребує великої сконцентрованості та уваги аби правильно розгорнути та налаштувати взаємодію між сервісами, саме тому цю роботу перенесено на окремих спеціалістів.

Багато всесвітньо відомих компаній вирішили проблеми моноліту змінивши архітектуру свої застосунків на мікросервісну. Серед таких варто виділити Walmart, Netflix, SoundCloud, Spotify, Medium, eBay, Stripe, PayPal, Uber, Twitter, та Amazon. Досвід Netflix щодо впровадження був настільки успішним, що компанія розробила та опублікувала велику кількість програмних засобів та підходів, за якими розроблялася їх архітектура мікросервісів. Сьогодні першопрохідцем розвитку мікросервісів по праву можна вважати Netflix, чий підхід став об’єктом дослідження для багатьох інших компаній у всьому світі.

1.4 Взаємодія WEB-застосунків з мобільними застосунками

Взаємодія web-застосунків та мобільних застосунків відбувається переважно через мережу Інтернет. Найпопулярнішим та найпростішим у реалізації варіантом є клієнт-серверна архітектура, в якій мобільний застосунок виступає клієнтською стороною та взаємодіє з сервером через API. API (Application Programming Interface) – це посередник, набір протоколів та правил взаємодії між сервісами, який визначає способи їх комунікації між собою. На рисунку 1.3 наведено просту схеми використання API.

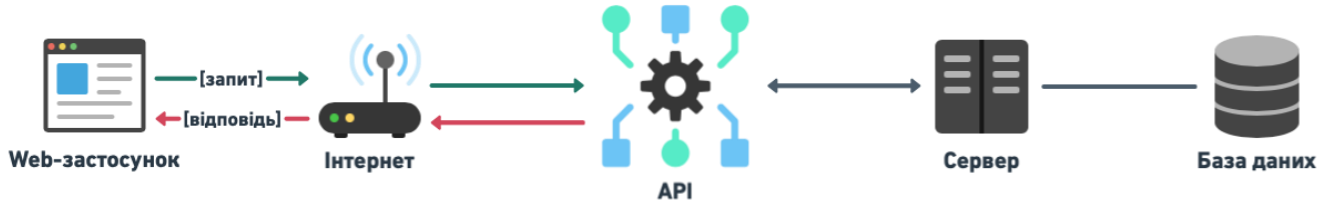


Рисунок 1.3 – Схема використання та роботи API

Виділяють три найпопулярніші способи реалізації API: RPC, REST та SOAP. Фактично, їх можна назвати форматами взаємодії, кожен з яких має свої певні характеристики та призначений для різних задач та досягнення різних цілей [18].

RPC (Remote Procedure Call) – це простий формат для надсилання кількох параметрів та отримання результатів. Використовуючи RPC клієнт викликає дії чи процеси на сервері, тоді як в інших реалізаціях клієнт і сервер переважно обмінюються даними чи ресурсами. Клієнти передають серверу назву метода та аргументи і отримують назад відповідь у вигляді JSON або XML [19]. JSON підтримує при обміні дані лише текстового формату, при цьому XML обробляє не тільки текст, а і зображення, графіки та діаграми. Таким чином, XML пропонує більше можливостей обробки документів, ніж JSON. Формат RPC почали використовувати приблизно в 80-х роках, але це не робить його автоматично застарілим. Великі компанії, такі як Google, Facebook (Apache Thrift) і Twitch (Twirp), використовують його для внутрішньої роботи API, щоб виконувати надзвичайно високопродуктивний обмін повідомленнями з низькими витратами [20]. RPC не зовсім підходить для приватних API через вкрай обмежену підтримку типів даних. Проте API на основі RPC доречно використовувати для внутрішніх частин більш об’ємних та складних API.

SOAP (Simple Object Access Protocol) – використовує лише мову XML. Формат даних XML дуже строгий і веде за собою ряд формальностей [18]. У поєднанні з масивною структурою повідомлень це робить SOAP найбільш складним у використанні API. SOAP доречно використовувати, коли компанії потрібна підвищена безпека та строго визначені правила для обміну даними між сервісами. Розробники часто використовують SOAP для внутрішніх або партнерських API. Логіка SOAP API написана мовою опису веб-служб (WSDL). Ця мова опису API визначає кінцеві точки та описує всі процеси, які можна виконати. Це дозволяє різним мовам програмування та IDE швидко налаштовувати зв’язок [20]. SOAP підтримує обмін повідомленнями із збереженням стану, тобто це значить, що сервер може зберігати отриману інформацію, особливо це виправдано для операцій зі складними транзакціями – банківські системи.

REST (Representational State Transfer) – більш проста альтернатива формату SOAP, яка була описана Роєм Філдінгом у 2000-му році у своїй докторській дисертації [20]. Формат REST робить доступними дані на стороні сервера, представляючи їх у простих для зчитування форматах, часто JSON та XML. Кожна одиниця інформації для REST – це унікальна URL-адреса, яку можна запросити у сервера і таким чином отримати дані [18]. В контексті мікросервісів це чудово працює для обміну даними між ними та об'єднанням в єдиний застосунок. Наразі REST являється найбільш використовуваними серед розробників, особливо для написання публічних API, які в свою чергу використовуються для архітектури мобільних та web-застосунків. Для більшого розуміння наведена таблиця 1.2 з порівняння форматів API, які були описані вище:

Таблиця 1.2

Порівняння форматів API: RPC, SOAP та REST

	RPC	SOAP	REST
Принцип роботи	Локальний виклик процедури	Передача повідомлень у строгому форматі	Передача повідомлень у вигляді ресурсу
Формат	XML, JSON, Protobuf, Thrift	Тільки XML	XML, JSON, HTML, простий текст
Складність використання	Легко	Важко	Легко
Частота Використання	Велика	Маленька	Велика
Застосування	Складні API або внутрішні мікросервіси	Платіжні системи, ідентифікація у фінансових системах	Публічні API

Висновки за розділом 1

У розділі 1 було описано нормативно-правову базу для функціонування web-застосунків, що дозволило визначити типи даних, які теоретично можуть оброблятися в застосунках. Це прямо впливає на побудову архітектури будь-якого застосунку та на побудову контуру безпеки, оскільки для службової та таємної інформації вимоги до обробки та забезпечення безпеки значно вищі.

Виконано аналіз двох найбільш популярних архітектур побудови web-застосунки: монолітної та мікросервісної.

Монолітна архітектура розглядає побудову застосунку як неподільної одиниці. Будь-які зміни, навіть самі невеликі, потребують перезбірки та повторного розгортання всього додатку. По мірі розвитку та розростання продукту стає дедалі складніше зберігати якісну модульну структуру, оскільки зміна логіки одного модуля може прямо впливати на зміни коду інших модулів. Монолітне програмне забезпечення також може бути важке у масштабуванні, оскільки різні модулі можуть мати конфлікти щодо розподілення ресурсів. Монолітна архітектура являється гарним рішенням для розробки невеликих та простих застосунків. Оскільки ця архітектура розглядається як традиційний спосіб розробки додатків, завжди краще мати хорошу фундаментальну базу знань.

На противагу монолітній, мікросервісна архітектура значно більш фрагментована, а кожен мікросервіс розгортається як окремий незалежний застосунок. Таким чином внесення змін до одного з мікросервісів можна виконувати не чіпаючи інших мікросервісів, вони можуть продовжувати працювати. Мікросервісна архітектура краще підходить для розробки складних додатків, де необхідне постійне масштабування, внесення змін у програмний код, реліз оновлень.

Також було розглянуто взаємодію веб-застосунків з мобільними застосунками шляхом використання API. З найбільш популярних та уживаних виділяють три підходи: RPC, SOAP та REST, кожен з яких має своє призначення та використовується для досягнення різних цілей. Проте в еру колосального зростання

використання мобільних пристроїв, найбільш популярним підходом до побудови API є REST, оскільки цей формат дозволяє будувати прості публічні API, які постійно використовуються при розробці мобільних застосунків.

Мобільні застосунки можуть взаємодіяти з web-застосунками, які побудовані на будь якій архітектурі. Але, якщо розглядати найбільш оптимальний та раціональний варіант взаємодії, варто сказати, що з web-застосунками на мікросервісній архітектурі набагато простіше побудувати механізми взаємодії та інтегрувати все в єдину систему. Оскільки мікросервіси розгортаються як окремі автономні контейнери, які обмінюються даними, то додати в цей процес мобільний застосунок буде набагато простіше – просто додати декілька сервісів та налаштувати комунікацію. У випадку з монолітною архітектурою варто було б переробляти логіку та більшу частину додатку. В контексті дипломної роботи для виконання практичної частини буде обрано web-застосунок з мікросервісною архітектурою.

РОЗДІЛ 2
ПРОЦЕС НАДАННЯ ДОСТУПУ ДО WEB-ЗАСТОСУНКІВ. ВРАЗЛИВОСТІ ТА МЕТОДИ ЗАХИСТУ

2.1 Огляд процесу надання доступу до WEB-застосунків

Процес надання доступу – це явище, з яким насправді людина у XXI столітті стикається кожного дня. Кожного дня ми отримуємо доступ до інформації у різного типу ресурсів, платформ та соціальних мереж навіть не здогадуючись, наскільки це складний та кропіткий процес з програмної точки зору та з точки зору безпеки даних. Несанкціонований доступ до даних і ресурсів є одним із найбільш небезпечних ризиків у кіберпросторі. Фондація OWASP у своєму звіті за 2017 рік визначила «Зламану автентифікацію» на друге місце серед 10 найбільших ризиків безпеки додатків, а «Зламаний контроль доступу» на п’яте [21]. При цьому, у порівнянні з 2017 роком, у 2021 році OWASP поставили «Зламаний контроль доступу» на перше місце, а «Зламану автентифікацію» на сьоме місце. Для більш наглядного розуміння у таблиці 2.1 наведено порівняння рейтингу 10 найбільших ризиків безпеки додатків у 2017 та 2021 роках:

Таблиця 2.1

Порівняння 10 найбільших ризиків безпеки додатків у 2017 та 2021 роках

Місце	**2017 рік**	**2021 рік**
1	Ін’єкції (наприклад SQL)	Зламаний контроль доступу
2	Зламана автентифікація	Криптографічні збої, Доступ до конфіденційних даних
3	Криптографічні збої, Доступ до конфіденційних даних	Ін’єкції (наприклад SQL)
4	Проблеми зовнішної сумісності ПЗ	Не безпечна архітектура ПЗ
5	Зламаний контроль доступу	Неправильна конфігурація безпеки

Продовження таблиці 2.1

6	Неправильна конфігурація безпеки	Використання застарілих компонентів або з наперед відомими вразливостями
7	Міжсайтовий скриптинг (XSS)	Зламана автентифікація та індентифікація
8	Збій цілісності програмного забезпечення та даних	Збій цілісності програмного забезпечення та даних
9	Використання застарілих компонентів або з наперед відомими вразливостями	Помилки логування та моніторингу безпеки
10	Помилки логування та моніторингу безпеки	Підробка запитів на стороні сервера

З таблиці видно, що в цьому проміжку в 4 роки, від деяких ризиків вдалося позбутися, наприклад: проблеми зовнішної сумісності ПЗ та міжсайтовий скриптин. Але також з’явилися нові ризики, які не фігурували у звіті за 2017 рік, а саме: не безпечна архітектура ПЗ (при цьому посіл топ-5 ризиків) та підробка запитів на стороні сервера.

Останнім часом було багато інцидентів щодо порушення доступу до даних (несанкціонований доступ). Найбільші у подібні ситуація потрапляла соціальна мережа Facebook, яка зараз належить компанії Meta, з мільйонами користувачів по усьому світу, а значить і з мільйонами терабайтів даних цих користувачів. Один з інцидентів свідчив про витік записів 540 мільйонів користувачів, після чого компанію запідозрили у навмисному продажу даних користувачів.

Створюваний web-застосунок, завжди повинен контролювати, хто чи що має доступ до його ресурсів та даних. Основна відповідальність — створювати належно високий рівень безпеки даних, включаючи потужні механізми контролю доступу.

Процес надання доступу включає в себе декілька етапів. Для опису контролю доступу в кібербезпеці використовують три наступні поняття: ідентифікація,

автентифікація та авторизація. Не дивлячись на те, що ці терміни тісно пов’язані, вони все ж таки мають певні специфічні відмінності, які необхідно уточнити та описати, щоб зрозуміти правильну термінологію щодо процесу надання доступу до систем у кіберпросторі.

Перед поясненням, термінів ідентифікації, автентифікації та авторизації, варто зазначити два інших фундаментальних поняття у контексті контролю доступу, а саме суб’єкт і об’єкт [23]. Ці поняття вже були розглянуті у розділі про нормативні документи, але в контексті доступу до різних видів інформації.

Суб'єкт – це сутність, яка звертається до об'єкта та виконує активну роль. В контексті web-застосунку, користувача, який здійснює доступ до застосунку, є суб’єктом. Але варто зауважити, що суб’єктом також може бути програма, сервіс чи процес, що звертаються до об’єкта.

Об’єкт — це пасивний компонент, до якого звертається суб’єкт. В контексті web-застосунку, він і є об’єктом у механізмі контролю доступу.

Ідентифікація є першим кроком контролю доступу і означає, що користувач заявляє про свою особу, тобто надає певний ідентифікатор. Ідентифікатором може виступати ім’я користувача, адреса email(найбільш розповсюджений наразі ідентифікатор для web-застосунків), номер телефону, придуманий нікнейм, ідентифікатор процесу, смарт-карта або що небудь інше, що може однозначно ідентифікувати суб’єкт [22]. Біометричні дані на сьогодні вважаються одним з самих надійних способів ідентифікувати особу. В якості біометрії можуть виступати:

- Відбитки пальців. Всі сучасні смартфони та більшість ноутбуків/клавіатур мають сканери відбитків пальців. Сучасні технології дозволяють розміщувати сканери відбитків під екрани у смартфонах;
- Сканер сітківки ока;
- Система розпізнавання обличчя. Компанія Apple розробила у своїх смартфонах функцію FaceID, яка за допомогою інфрачервоних променів робить 3D модель обличчя та зберігає на пристрої. Кожен раз при спробі розблокувати смартфон камера сканує обличчя особи та порівнює зі збереженою 3D моделю.
- Система розпізнавання голосу.

Автентифікація є другим кроком контролю доступу і означає, процес підтвердження суб'єкта/особи, тобто суб'єкт має довести, що він справді є тим, за кого себе видає. Це підтвердження ідентичності досягається шляхом надання облікових даних механізму контролю доступу. Якщо система при автентифікації вимагає як мінімум два облікових типи даних, то процес називається багатофакторною автентифікацією. Механізми контролю доступу потім перевіряють дійсність наданих облікових даних перед схваленням запиту на автентифікацію. Облікові дані, які використовуються для автентифікації, можна розділити на чотири різні групи [24]:

- Щось, що відоме суб'єкту (фактор автентифікації типу 1) –пароль, особистий ідентифікаційний номер (PIN);
- Щось, що є у суб'єкта (фактор автентифікації типу 2) – смарт-карта, карта пам'яті або маркер;
- Біометрія (фактор автентифікації типу 3) – відбиток пальця, топологія долоні, сканування сітківки ока, система розпізнавання обличчя тощо;
- Поведінка (фактор автентифікації типу 3) –динаміка натискання клавіші, шаблон підпису або зразок голосу.

В контексті web-застосунку або будь якої інформаційної системи, ідентифікатором є логін(email, ім'я, обліковий номер тощо), а автентифікаторами може виступати:

- Паролі;
- PIN-коди;
- Електронно цифрові підписи, в Україні – кваліфіковані цифрові підписи;
- біометричні дані тощо.

Автентифікація суб'єктів системи надає кілька переваг [24]:

- Запобігання крадіжок: Основна мета системи контролю доступу — обмежити доступ, щоб захистити особисті дані користувачів від крадіжки або зміни. Багато веб-сайтів, які потребують особистої інформації для своїх послуг, особливо ті, для яких потрібна інформація про кредитну картку або номер соціального

страхування особи, згідно із законодавством або нормативними актами повинні мати механізм контролю доступу.

- Рівні безпеки: Сучасні системи керування розвивалися разом із технологічним прогресом. Людина, яка хоче зберегти інформацію в безпеці, має більше можливостей, ніж просто чотиризначний PIN-код і пароль. Наприклад, замки з біометричним скануванням тепер можна встановлювати в домашні та офісні точки входу.

Авторизація є третім кроком контролю доступу і визначає рівень доступу суб’єкта до об’єкта, тобто вона визначає, які привілеї має суб’єкт [23]. Варто зауважити, що авторизація неможлива без ідентифікації та автентифікації.

Наприклад, у багаторівневій системі безпеки, той факт, що суб’єкт був автентифікований, не обов’язково означає, що суб’єкт може мати повний доступ до усієї системи (або повні привілеї над об'єктом.) У такій системі суб'єктам надається доступ відповідно до рівня їх дозволу. Іншим прикладом є права читання, запису та виконання, призначені суб’єктам управління файловою системою в операційних системах.

В контексті авторизації варто звернути увагу на моделі контролю доступу:

- Мандатна модель доступу (Mandatory access control). Це модель, в якій порядок доступу суб’єктів до об’єктів визначається на основі призначених міток конфіденційності для інформації, що міститься в об'єктах, і видачу офіційних дозволів суб'єктів на звернення до інформації такого рівня конфіденційності [25].

- Дискреційна модель доступу (Discretionary access control). Це модель, у якій власник або адміністратор захищеної системи, даних або ресурсу встановлюють політики, які визначають, права доступу суб’єкта до об’єкта [26].

- Рольова модель доступу (Role-based access control). Це модель, в якій права доступу суб’єктів до об’єктів групуються за специфікою використання та утворюють ролі. В контексті web-застосунку можна виділити наступі ролі доступу з різними правами: користувач системи, адміністратор системи, співробітник служби підтримки, власник системи тощо. У всіх цих ролей будуть відрізнятися рівні доступу до даних, які обробляються в системі.

2.2 Класифікація методів автентифікації

Етап автентифікації завжди виконується на початку роботи користувача з web-застосунком, до того, як відбудуться перевірки дозволу та регулювання, якщо йому необхідно отримати доступ до обмеженої інформації. Сам процес можна розділити у дві окремі фази - ідентифікацію та фактичну автентифікацію, які були описані у попередньому розділі. Залежно від ступеня довірчих стосунків, структури, особливостей мережі і віддаленості об'єкта перевірка облікових даних може бути односторонньою або взаємною. Вид облікових даних, їх кількість при запиті та специфіка використання визначають класи методів автентифікації.

2.2.1 Базова автентифікація

Базова автентифікація – є найпростішим методом автентифікації в процесі контролю доступу до web-застосунків та різного роду web-ресурсів. Першочергово вона була реалізована Арі Луотоненом в CERN у 1993 році і визначена у специфікації HTTP 1.0 у 1996 році [28]. Реалізація базової автентифікації є найпростішим методом серед усього переліку, оскільки для її роботи не потрібні файли cookie, ідентифікатори сеансу або сторінки входу – вона використовує стандартні поля в заголовку HTTP.

Під час роботи особи з web-застосунком, механізм автентифікації включається в той момент, коли застосунок запрошує у сервера доступ до захищеної інформації, і не надає при цьому в запиті даних для ідентифікації. Сервер на такий запит посилає відповідь-заголовок «401 Unauthorized» і відправляє до застосунку запит на ідентифікацію [27]. Після чого формується вікно, в якому користувач має ввести свій ідентифікатор і пароль. Після введення облікових даних, браузер відправляє новий HTTP запит на сервер, в якому міститься поле заголовку виду «Authorization: Basic <credentials>», де credentials – це облікові дані користувача, тобто логін та пароль. На рисунку 2.1 зображена діаграма послідовності, яка ілюструє механізм роботи базової автентифікації.

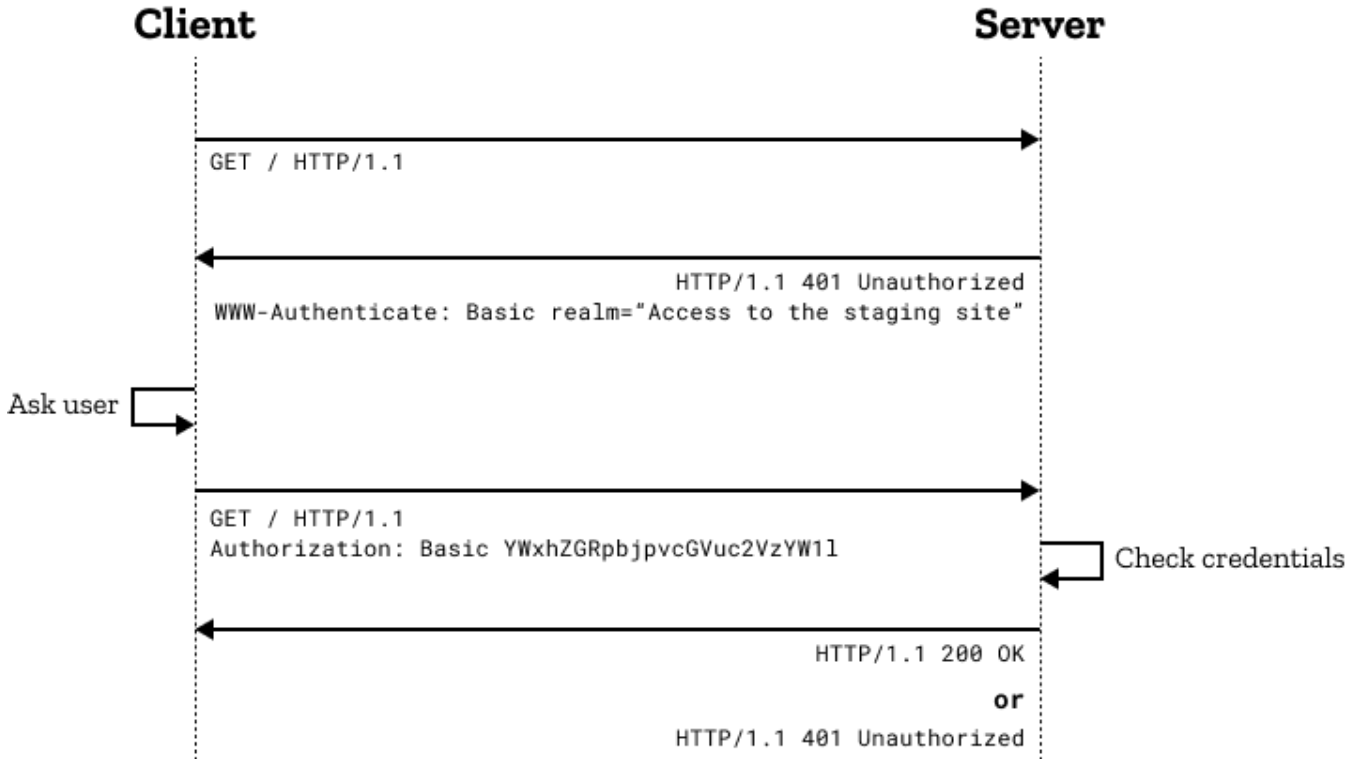


Рисунок 2.1 – Діаграма послідовності механізму роботи БА

Не дивлячись на відносну простоту, у механізмі БА існує цілий ряд особливостей. Однією з таких є – кешування даних браузером. Оскільки поле «Authorization: Basic<credentials>» має бути передано в заголовку кожного запиту HTTP, браузер має кешувати облікова дані, щоб користувачу не було необхідності постійно вводити ідентифікатор та пароль. Робиться це виключно для «полегшення» роботи користувача з web-застосунком, але з іншого боку, це може призвести до перехоплення цих персональних даних [27].

Персональні дані при БА не передаються прямо в явному вигляді, вони кодуються за технологією Base64. Не можна однозначно стверджувати, що Base64 забезпечує конфіденційність переданих облікових даних, оскільки вони не шифруються та не хешуються, а для розкодування такого рядка, достатньо скористатися однією функцією, яка включена у багатьох мовах програмування. Тому БА зазвичай використовується разом із HTTPS для того, аби забезпечети конфіденційність даних [28].

2.2.2 Дайджест автентифікація

Дайджест автентифікація фактично є покращеною, з точки зору безпеки, базовою автентифікацією. Це один із стандартних методів, який використовується веб-сервером для автентифікації облікових даних користувача до web-застосунку. Він не пересилає облікові дані користувача, тобто логін і пароль, у відкритому вигляді, а спочатку використовує хеш-функцію для хешування паролю. Технічно, дайджест автентифікація представляє собою застосування криптографічної хеш-функції MD5 до секрету(паролю) користувача з використанням випадкових значень для ускладнення криптоаналізу і запобігання replay-атак. Працює на рівні протоколу HTTP[29]. Тобто принциповою відмінністю від БА – є покращена конфіденційність облікових даних при передачі, що робить дайджест автентифікацію більш вживаною серед звичайних користувачів та систем. Але варто зауважити, що в останні роки MD5 не можна вважати однозначно безпечним методом хешування. Він був і залишається доволі популярним і для його взламу були придумані різні способи, оскільки у нього було відкрито багато вразливостей.

Дайджест автентифікація вперше з'явилася в RFC 2069 стандарті (розширення для HTTP: дайджест автентифікація). RFC 2069 описує класичну модель дайджест автентифікації, де захист забезпечується згенерованим сервером випадкових значень. Відповідь клієнта на запит автентифікації серверу формується наступним чином (де HA1, HA2, A1, A2 – назви змінних) [29, 30]:

$$HA_1 = MD5(A_1) = MD5(username{:}realm{:}password), \quad (2.1)$$

$$HA_2 = MD5(A_2) = MD5(method{:}URI), \quad (2.2)$$

$$Response = MD5(HA_1{:}nonce{:}HA_2), \quad (2.2)$$

де username – ідентифікатор користувача;

realm – стрічка, яка ідентифікує поточну зону дії автентифікації (сторінки або сторінки сайту);

password – пароль користувача;

URI – уніфікований ідентифікатор ресурсів;

Nonce – випадково згенероване число, яке сервер одноразово надає клієнту.

2.2.3 Автентифікація із застосуванням цифрового сертифікату

Автентифікація із застосуванням цифрових сертифікатів, як правило, використовує протокол із запитом і відповіддю. Технологія цифрових сертифікатів була розроблена для безпечного використання інфраструктури відкритих ключів (Public Key Infrastructure), принцип роботи якої засновано на асиметричному шифруванні [31].

Термін «PKI або Internet PKI» був введений IETF (Internet Engineering Task Force) для використання його в Інтернеті для сертифікатів X.509, де він зосереджується на тому, як браузери перевіряють і використовують сертифікати [31]. Тобто, PKI - це перелік правил та процедур, необхідних для створення, обробки, управління, зберігання та використання сертифікатів.

X.509 – це криптографічний стандарт для інфраструктури відкритого ключа, який стандартизує формати представлення та кодування [32]:

- сертифікатів відкритого ключа (public key certificates);
- атрибутів сертифікатів;
- алгоритм перевірки методу сертифікації;
- списку відкликаних сертифікатів.

Випуск сертифікатів регулює чітко визначена ієрархічна структура відповідальних органів (certificate authorities — CAs), при цьому сам сертифікат має також чітку структуру і складається з наступних пунктів:

- Сертифікат (Certificate):
  - версія
  - серійний номер
  - ідентифікатор алгоритму (Algorithm ID)
  - видавець (Issuer)
  - період дії (Validity)
  - суб’єкт сертифікату (Subject)
  - інформація про публічний ключ суб’єкту (алгоритм, публічний ключ)

- унікальний ідентифікатор видавця (опціонально)
- унікальний ідентифікатор суб’єкту (опціонально)
- розширення (опціонально)

- Алгоритм підпису сертифікату;
- Підпис сертифікату.

Стандарт X.509 визначає два види сертифікатів: сертифікат відкритого ключа та сертифікат атрибутів[32]. Сертифікат відкритого ключа надає підтвердження, що відкритий ключ дійсно належить конкретній особі, оскільки містить відповідну інформацію, яка була описана вище. Формат сертифіката відкритого ключа стандарту X.509 v3 наведено в НД RFC 5280. Також такі сертифікати можуть ідентифікувати особу та визначати операції, які особа може здійснювати, використовуючи закритий ключ, що відповідає відкритому ключу сертифікату. Сертифікат атрибутів — підтверджує атрибути та їх значення, які містять інформацію про особу, приналежність до групи, роль, повноваження тощо. Формат сертифіката атрибутів наведено в НД RFC 5755 [32].

Цифрові сертифікати зберігають довіру між користувачем та web-застосунком, які автентифікуються. Вони зв’язують інформації про публічний ключ та його власника, тим самим убезпечуючи передачу даних між сторонами. Автентифікацію за допомогою сертифікатів забезпечують кілька поширених протоколів, зокрема, найвідоміший та найпоширеніший протокол Secure Socket Layer (SSL), який застосовується практично у кожному web-браузері. Крім нього, застосовуються протоколи Transport Layer Security (TLS), Internet Key Exchange (IKE) тощо [33].

У багатьох протоколах передбачається, що клієнт надсилає запит серверу для того, щоб ініціювати автентифікацію. Якщо сервер підтримує метод автентифікації, запитуваний користувачем А, то починається обмін повідомленнями. Повідомлення Token ID повідомляє про те, що буде виконана взаємна автентифікація, а також може містити номер версії протоколу та ідентифікатор протоколу, але це не обов’язково. Користувач А очікує повідомлення Token BA1 від сервера В. Ідентифікатор протоколу в Token ID дозволяє користувачеві А переконатися, що

сервер відправляє очікуване повідомлення. Token BA1 складається тільки з випадкового числа N, це - свого роду запит, коректною відповіддю має бути цифровий підпис числа N. Користувач А підписує відповідь і надсилає свій сертифікат ключа підпису, для того щоб сервер за допомогою відкритого ключа міг виконати валідацію підпису. Отримавши відповідь Token AB від користувача А, сервер перевіряє, чи збігається значення N з відповідним значенням у повідомленні Token BA1, і встановлює, чи дійсно користувач А хоче виконати автентифікацію сервера В. Якщо на якомусь з етапів перевірок отримується негативне значення(результат), то й автентифікація завершується невдало. В іншому випадку, сервер перевіряє справжність сертифіката користувача А та його цифровий підпис, якщо сертифікат і підпис валідні, то автентифікація користувача А сервером В пройшла успішно [33].

2.2.4 Автентифікація із застосуванням смарт-карток та USB-ключів

Не дивлячись, що цифрові сертифікати X.509 можуть забезпечувати строгу автентифікацію користувача, закритий ключ сам по собі являється незахищеним. Закритий ключ, що найчастіше зберігається на комп'ютері власника, уразливий до прямих і мережевих атак. Функціональним аналогом цифрового ключа можуть виступати смарт-картки — пластикові картки з вбудованою мікросхемою, схожі на банківські картки. Для використання смарт-карток в інформаційних системах необхідні пристрої зчитування смарт-карток, які можуть підключатися до комп'ютера за допомогою USB порту. Існують більш сучасні варіанти, коли пристрій читання смарт-карток вже вбудований в клавіатуру. Найчастіше користувачу необхідно ввести пароль аби отримати доступ до захищеної інформації, що зберігається в пам'яті смарт-картки [27].

USB-ключі можна назвати ще більш функціональними в порівнянні зі смарт-картками, оскільки всі сучасні комп’ютери мають USB порти. До того ж, він став стандартним портом для підключення периферійних пристроїв і якщо мова йде про постійну автентифікацію в межах організації, то компанії не потрібно закуповувати

спеціальні зчитувачі. Автентифікацію на основі смарт-карток і USB-ключів доволі складно обійти програмними методами, оскільки в процесі бере участь фізичний об'єкт, який повинна мати особа, щоб автентифікуватися у систему. У разі викрадення картки або USB-ключа власник швидко може докласти зусиль аби заблокувати їх використання у системі, тому зловмисник не зможе авторизуватися. І смарт-картка, і USB-ключ можуть виступати другим фактором автентифікації, таким чином підвищуючи безпеку усього процесу контролю та надання доступу до системи.

2.2.5 Багатофакторна автентифікація

Багатофакторна автентифікація – це розширений тип автентифікації, коли в процесі отримання доступу до системи користувачу необхідно надати більше одного типу облікових даних. В попередньому розділі були згадані дані, які можуть виступати додатковим фактором, а саме:

- Щось, що відоме суб’єкту (фактор автентифікації типу 1) –пароль, особистий ідентифікаційний номер (PIN);
- Щось, що є у суб’єкта (фактор автентифікації типу 2) – смарт-карта, карта пам’яті або маркер;
- Біометрія (фактор автентифікації типу 3) – відбиток пальця, топологія долоні, сканування сітківки ока, система розпізнавання обличчя тощо;
- Поведінка (фактор автентифікації типу 3) –динаміка натискання клавіші, шаблон підпису або зразок голосу.

Багатофакторна автентифікація значно підвищує безпеку процесу надання та контролю доступу до системи, оскільки зменшує імовірність викрадення даних за допомогою використання другого фактору – зловмиснику не достатньо знати пароль від облікового запису, необхідно володіти додатковою інформацією.

Найпоширенішим на сьогодні типом багатофакторної автентифікації є двофакторна автентифікація. Механізм її роботи такий, що користувач надає

серверу логін, потім пароль(перший фактор) і має надати другий фактор у вигляді OTP-коду(One-time password). Наразі є декілька варіантів його отримання:

- Система може надсилати його SMS на ваш номер телефону;
- OTP може бути надісланий на ваш email;
- Мобільні застосунки. Найпопулярнішим мобільним застосунком-автентифікатором є Google Authenticator, який необхідно спочатку підключити до web-застосунку/системи, після чого на смартфоні генерується шестизначний код, який необхідно ввести у web-застосунок. Код генерується кожні 30 секунд, тобто відповідно термін його дії також 30 секунд, що значно підвищує рівень безпеки.

Перевагою двофакторної автентифікації з використанням мобільного пристрою є те, що немає необхідності використовувати додаткові токени: USB-ключі, смарт-картки – телефон завжди під рукою. Також загалом можна стверджувати, що багатофакторна автентифікація наразі є найбезпечнішим методом автентифікації, бо вона може поєднати у собі всі ті класи, які були описані вище. Проектуючи систему, можна визначити у вигляді додаткових факторів абсолютно різноманітні типи облікових даних. Варто зазначити, що в багатьох системах є можливість під час двофакторної автентифікації поставити відмітку «Довіряти пристрою» і користувачу не потрібно буде кожен раз при автентифікації вказувати другий фактор.

2.3 Поширені загрози процесу контролю-надання доступу та методи захисту

Контроль доступу являється одним з ключових елементів контуру безпеки інформаційної системи, в тому числі і web-застосунку, оскільки захищає системні ресурси та дані від несанкціонованого доступу, надаючи відповідний рівень авторизації для кожного користувача. За допомогою автентифікації та авторизації механізм контролю доступу гарантує, що користувачі є тими, за кого себе видають, і що вони мають відповідний доступ до даних компанії. Контроль доступу – це доволі широке поняття, яке використовується не тільки в контексті інформаційної безпеки,

його можна застосувати для обмеження фізичного доступу до будівель, кімнат і центрів обробки даних [34].

Атаки зазвичай обходять методи контролю доступу і націлені на крадіжки інформації або облікових даних користувача. Це дає зловмисникам великі переваги, оскільки вони можуть використати вкрадені дані для авторизації у систему і отримати ще більш розширений доступ до ресурсу.

Атака агрегації доступу здійснюється шляхом збору різної неконфіденційної, яка пов’язана з системою, для того аби отримати конфіденційну інформацію. Зловмисники знайдуть кілька фактів про цільову систему, після чого детально дослідивши їх, зможуть провести атаку. Наприклад розвідувальна атака, при якій хакери за допомогою різноманітних інструментів визначають елементи системи: IP-адресу сервера, операційну систему, відкриті порти, можливість завантажувати файли тощо [36]. Аби убезпечити систему від даного типу атаки, необхідно виконувати її постійне тестування та моніторинг, тобто аналізувати чи є у вільному доступі чутлива інформація про систему, наприклад – наявність відкритих портів, до яких можуть під’єднатися зловмисники.

Паролі – найбільш вразлива частина автентифікації. Атаки на паролі мають на меті викрити пароль користувача, адміністратора або root-а системи. В залежності від того, пароль якого типу користувача було викрито, зловмисник може отримати доступ до різного типу даних: на рівні звичайного користувача може отримати доступ до індивідуального облікового запису; на рівні адміністратора або root може отримати доступ до будь-якого облікового запису та авторизованих ресурсів. Другий випадок для хакерів є найбільш привабливим, оскільки від root-а можна створити бекдори, які можна буде використовувати пізніше. Паролі можуть бути скомпрометовані декількома способами [35]:

- Атака з перебором за словником (Dictionary attack). Атака заснована на тому, що зловмисники зламують паролі користувачів, аналізуючи звичайні слова на основі словників. Спеціальний інструмент сканує файл словника, щоб виявити «збіг» із введеним паролем. Фактично інструмент хешує звичайні слова(або їх комбінації) з файлу словника, після чого порівнює результати із зашифрованим

паролем, і декодує пароль у випадку, якщо знайдено відповідність між хешованим словом та зашифрованим паролем. Саме тому, якщо пароль складається переважно з звичайних слів зі словника, його можна доволі легко зламати. Нажаль, всі типи автентифікації, де використовуються паролі чи будь-які секретні значення, уразливі до атаки за словником. У 2009 році був інцидент, коли з використанням атаки за словником були зламані десятки акаунтів знаменитостей у соціальній мережі Твіттер [36].

- Атака грубої сили (Brute-Force attack). Ця атака передбачає перебір всіх можливих комбінацій букв, цифр і символів для виявлення пароля. Хакери використовують для реалізації атаки різноманітні складні інструменти, які автоматично створюють комбінації. В першочерговому її вигляді, суть атаки полягала в переборі символи до поки не знайде збіг за значенням з паролем. На сьогодні зловмисникам не потрібно шукати «чистий пароль», їм достатньо підібрати таку комбінацію, хеш значення якої буде збігатися з хеш-значенням, що зберігається в базі даних облікового запису. Наприклад, в базі даних для паролю «INFOSEC1» збережене хеш-значення 68FAC1. Інструмент грубої сили перебере можливі комбінації символів та здійснить порівняльний аналіз, щоб визначити, чи є відповідність хеш-значення. Якщо збіг буде знайдено, то пароль буде зламано [36].

- Сніффер атака (Sniffer attack). Сніфферна атака, також відома як атака підслуховування, відбувається коли хакер використовує спеціальне програмне забезпечення – сніфер, він же аналізатор протоколів або аналізатор пакетів, для захоплення маршрутизації трафіку мережі. Інструмент може захоплювати та читати будь-яку інформацію у відкритому вигляді, наприклад паролі, які передаються через мережу. Прикладом подібного інструменту є Wireshark, з яким неодноразово доводилося працювати під час процесу навчання; він може збирати будь-яку інформацію, яка відкрито передається через мережу. У попередньому розділі був розглянутий базовий тип автентифікації, при якому логін-пароль користувача передаються на сервер майже у відкритому вигляді, в даному випадку зловмисник міг би з легкістю перехопити його за допомогою Wireshark. Шифрування даних є найефективнішим заходом безпеки, оскільки сніфери не можуть аналізувати

зашифровані дані. Тобто починаючи з дайджест автентифікації, сніфер вже не зможе отримати пароль користувача, оскільки той передається на сервер у зашифованому вигляді. Системи виявлення вторгнень також можуть допомогти у виявленні сніффер атак, оскільки можуть надсилати сповіщення про роботу відповідних інструментів сніффінгу, якщо вони звісно правильно налаштовані [36].

Загалом варто зауважити, що надійні паролі – є ключем до захисту системи від перелічених атак. Використання довгих комбінацій з різних типів символів (малі літери, великі літери, спеціальні символи та цифри), бажано які не утворюють звичайні слова, значно ускладнять реалізацію атак перебору грубої сили та атак з використанням словника. Оскільки чим різноманітніші символи у паролі, тим більше часу інструментам необхідно, аби підібрати правильне значення, що значно підвищує зловмисникам ризики бути виявленими.

Атака Man-In-The-Middle (MITM) або «людина посередині» є однією з найдавніших та найнебезпечніших атак. Ця атака передбачає те, що зловмисник перехоплює трафік між користувачами та може переглянути або змінити його вміст, але при цьому, трафік надходить кінцевому користувачу. На рисунку 2.2 зображена спрощена схема реалізації атаки:

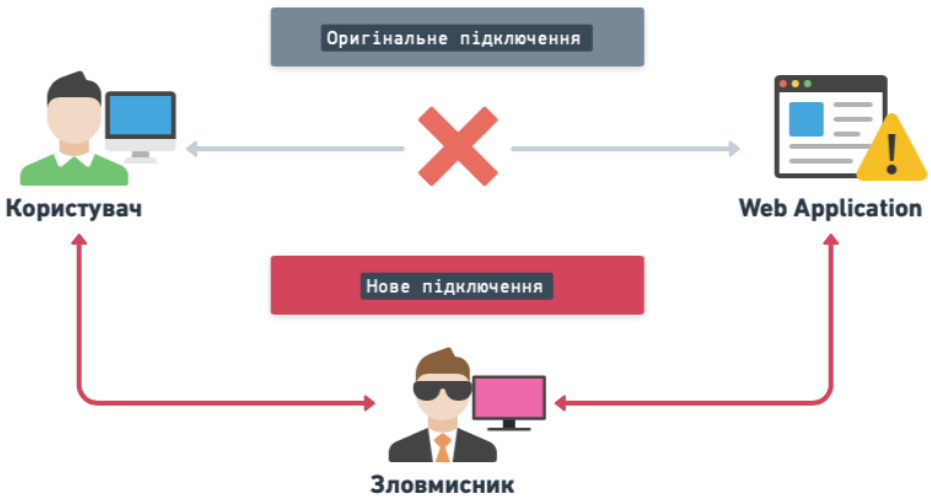


Рисунок 2.2 – Атака Man-In-The-Middle

У сучасному світі, зловмисники використовуються атаки MITM щоб підмінювати криптовалютні гаманці для викрадення коштів, перенаправлення

користувачів на шкідливі веб-сайти, які можуть здійснювати пасивний збір інформації про користувача, з метою її подальшого злочинного використання[38]. Загальнодоступні мережі Wi-Fi є прекрасним полігоном для реалізації MITM, оскільки ні маршрутизатор, ні підключений комп'ютер не перевіряють її ідентичність. Зловмисник повинен бути під'єднаний до тієї самої мережі, або ж просто мати комп'ютер в мережі, здатний перехоплювати трафік – і атака MITM вже працює. При атаці MITM перебування зловмисника поруч із жертвою не є обов'язковою умовою, оскільки існує багато зловмисних програмних інструментів, які здатні перехоплювати трафік та підміняти інформацію в будь-якому місці.

Захист від атак MITM полягає в тому, аби використовувати посилену автентифікацію кінцевої точки, наприклад TLS або SSL-сертифікати, які гарантують захищене з'єднання між клієнтом та web-застосунком, що в ідеалі не може бути підробленим. Двофакторна автентифікація також може виступати способом захисту від атак MITM. Як було вже зазначено вище, паролі являються одними з найуразливіших способів захисту облікових записів та систем, тому додавання другого фактора автентифікації, значно ускладнює перехоплення трафіку або злам шифру для зловмисника. Всі ці методи автентифікації детально було розглянуто у попередньому підрозділі. Але попри все не можна однозначно стверджувати, що шифровані дані неможливо зламати – зловмисникам інколи вдається підробити SSL-сертифікат і підмінювати офіційні банківські сайти на фальшиві веб-ресурси крадіжки інформації[38].

Висновки за розділом 2

У розділі 2 було детально розгляну процес контролю-надання доступу, що дозволило визначити усі етапи процесу та механізми їх взаємодії. Це прямо впливає на виконання практичної частини диплому, оскільки побудова підсистеми авторизації торкається усіх етапів, а саме:

- *Ідентифікація,* коли користувач заявляє про свою особу, тобто надає певний ідентифікатор.

- *Автентифікація* тобто, процес підтвердження суб'єкта, тобто суб'єкт має довести, що він справді є тим, за кого себе видає. Це підтвердження ідентичності досягається шляхом надання облікових даних механізму контролю доступу.

- *Авторизація,* яка визначає рівень доступу суб'єкта до об'єкта, тобто вона визначає, які привілеї має суб'єкт. Варто зауважити, що авторизація неможлива без ідентифікації та автентифікації.

Було виконано детальний аналіз та порівняння різновидів автентифікації, щоб обрати конкретний тип, який буде використано при проектуванні підсистеми авторизації у наступному розділі.

Базова автентифікація – є найпростішим методом автентифікації в процесі контролю доступу до web-застосунків та різного роду web-ресурсів. Під час роботи особи з web-застосунком, механізм автентифікації включається в той момент, коли застосунок запрошує у сервера доступ до захищеної інформації, і не надає при цьому в запиті даних для ідентифікації. Облікові дані не передаються прямо в явному вигляді, вони кодуються за технологією Base64, але це не забезпечує конфіденційність даних, оскільки вони не шифруються та не хешуються.

Дайджест автентифікація є покращеною, базовою автентифікацією. Він не пересилає облікові дані користувача, тобто логін і пароль, у відкритому вигляді, а спочатку хешує пароль за допомогою хеш-функції MD5.

Автентифікація із застосуванням цифрових сертифікатів, як правило, використовує протокол із запитом і відповіддю SSL, який застосовується практично у кожному web-браузері. Крім нього, застосовуються протоколи Transport Layer Security (TLS), Internet Key Exchange (IKE) тощо. Технологія цифрових сертифікатів була розроблена для безпечного використання інфраструктури відкритих ключів, які в свою чергу необхідні для обробки та використання сертифікатів X.509.

X.509 – це криптографічний стандарт для інфраструктури відкритого ключа, який стандартизує формати представлення та кодування сертифікатів відкритого ключа. Цифрові сертифікати зберігають довіру між користувачем та web-застосунком, які автентифікуються. Вони зв'язують інформації про публічний ключ та його власника.

Функціональним аналогом цифрового сертифікату можуть виступати смарт-картки — пластикові картки з вбудованою мікросхемою, схожі на банківські картки або ж USB-ключі. Для їх використання в системах необхідні пристрої зчитування смарт-карток, які можуть підключатися до комп'ютера за допомогою USB порту, і власне USB-порт для використання USB-ключа.

Багатофакторна автентифікація – це найбезпечніший тип автентифікації, оскільки в процесі отримання доступу до системи користувач повинен надати більше одного типу облікових даних, тобто додаткові фактори. Додатковими факторами найчастіше виступають: PIN-коди, OTP-коди, смарт-картки, біометричні дані (відбитки пальців, скан сітківки ока, скан топології долоні, 3D-скан обличчя) та поведінка (динаміка натискання клавіші, шаблон підпису або зразок голосу).

РОЗДІЛ 3
ПОБУДОВА ПІДСИСТЕМИ АВТОРИЗАЦІЇ ДЛЯ WEB-ЗАСТОСУНКІВ З ВИКОРИСТАННЯМ МОБІЛЬНИХ ПРИСТРОЇВ

3.1 Опис запропонованого рішення

У попередньому розділі детально було розглянуто процес надання-контролю доступу до web-застосунку, що в свою чергу дозволило зробити важливий висновок – даний процес являється однією з найуразливіших ланок в контексті функціонування web-застосунку. Зловмисники постійно намагаються отримати доступ до тих чи інших ресурсів, що прямо впливає на компрометацію даних користувачів. В даній роботі буде запропоноване рішення для захисту процесу авторизації у web-застосунок на основі посилення автентифікації додатковим фактором.

Як вже було згадано раніше, багатофакторна автентифікація на сьогодні є найбезпечнішим методом автентифікації, а також вона апріорі посилює захист усього процесу надання доступу до застосунку. Розроблювана підсистема авторизації базується на використанні додаткового фактора автентифікації – приватного ключа, який зберігається в захищеній області пам’яті мобільного пристрою. На рисунку 3.1 наведена схема взаємодії всіх складових підсистеми.

В підсистемі фігурує web-застосунок, доступ до якого буде отримуватися, а точніше його Front-end та Back-end частини, та мобільний застосунок з його back-end частиною. Для початку користувачу необхідно «під’єднати» мобільний застосунок до web-застосунку, щоб він міг надалі фігурувати у процесі авторизації. Як тільки користувач розпочинає процес авторизації до web-застосунку, на етапі автентифікації необхідне додаткове підтвердження від користувача через підключений мобільний застосунок.

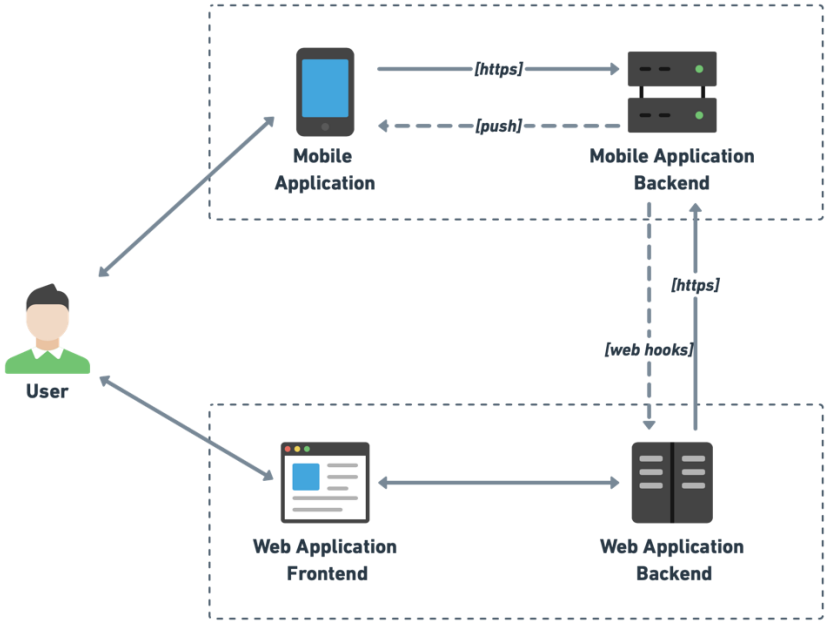


Рисунок 3.1 – Схема взаємодії складових підсистеми

Безпека рішення пов’язана з використанням асиметричного шифрування. Асиметричне шифрування базується на використанні двох ключів: публічного(відкритого) для шифрування та приватного(закритого) для розшифрування. Підґрунтям для асиметричної криптографії стали ідеї В. Діффі та М. Хеллмана про шифрування з двома ключами, що стали відомими у 1976 році [39]. Алгоритм, розроблений Р. Рівестом (Rivest), А. Шаміром (Shamir) і Л. Адлеманом (Adleman) у 1978 році, являється першим алгоритмом асиметричного шифрування, який мав практичне значення, і отримав назву RSA (перши літери прізвищ авторів) [39]. На рисунку 3.2 зображено схему використання асиметричного шифрування на публічному ключі.

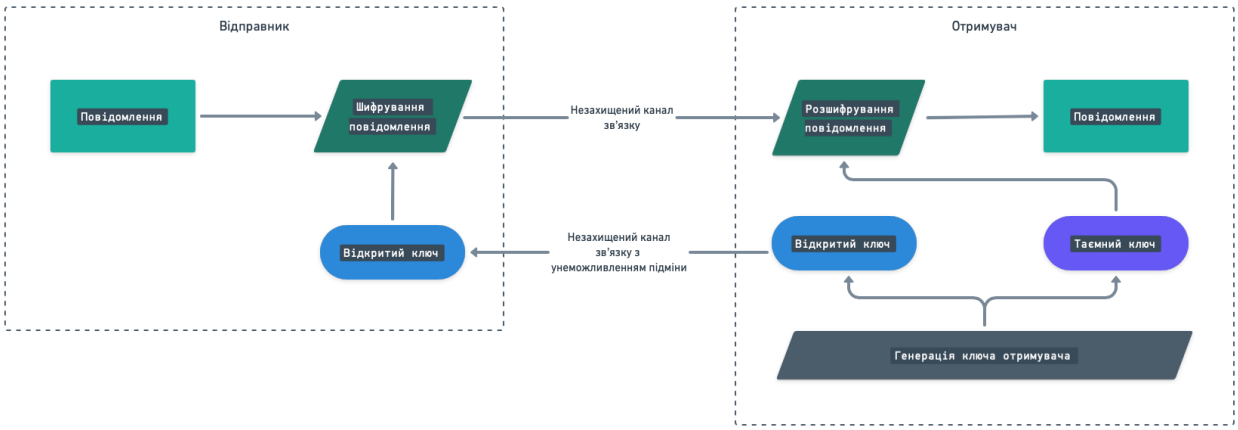


Рисунок 3.2 – Схема використання асиметричного шифрування

У теорії складності обчислень існує поняття, яке характеризує рівень складності обчислень (кількість операцій) в залежності від розміру вхідних даних. Одними з таких є поліноміальний і експоненційний характери залежності складності обчислень від кількості вхідних даних. В асиметричному шифруванні за Діффі-Хеллманом, зашифроване повідомлення при наявності приватного ключа розшифровується за поліноміальний час роботи обчислювальної системи, а у разі відсутності — за експоненційний час[39].

3.2 Архітектура підсистеми

Дослідження архітектури web-застосунків та процесу взаємодії їх з мобільними застосунками дозволило сформувати діаграму послідовностей для підсистеми авторизації у web-застосунок, а саме використання мобільного пристрою для автентифікації. Як вже було описано раніше, для початку користувачу необхідно під’єднати мобільний застосунок до web-застосунку. На рисунках 3.2-3.3 наведена діаграма послідовностей для їх підключення.

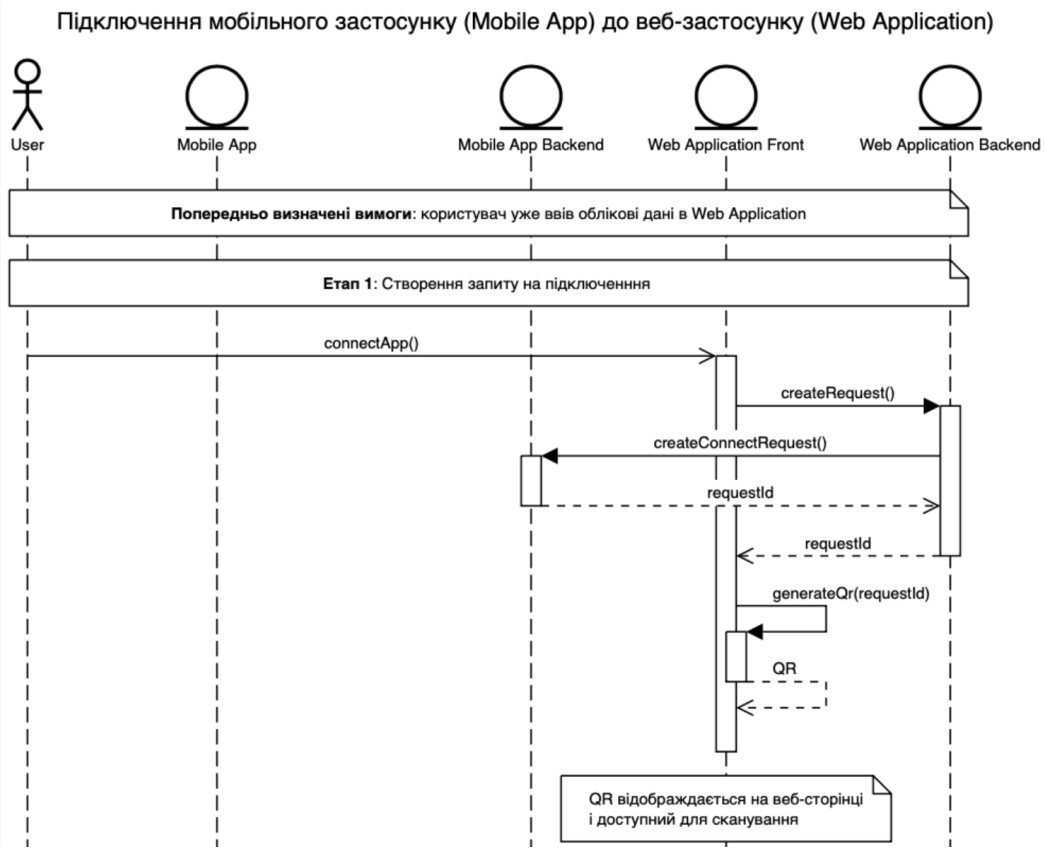


Рисунок 3.2 – Діаграма послідовностей для першого етапу підключення

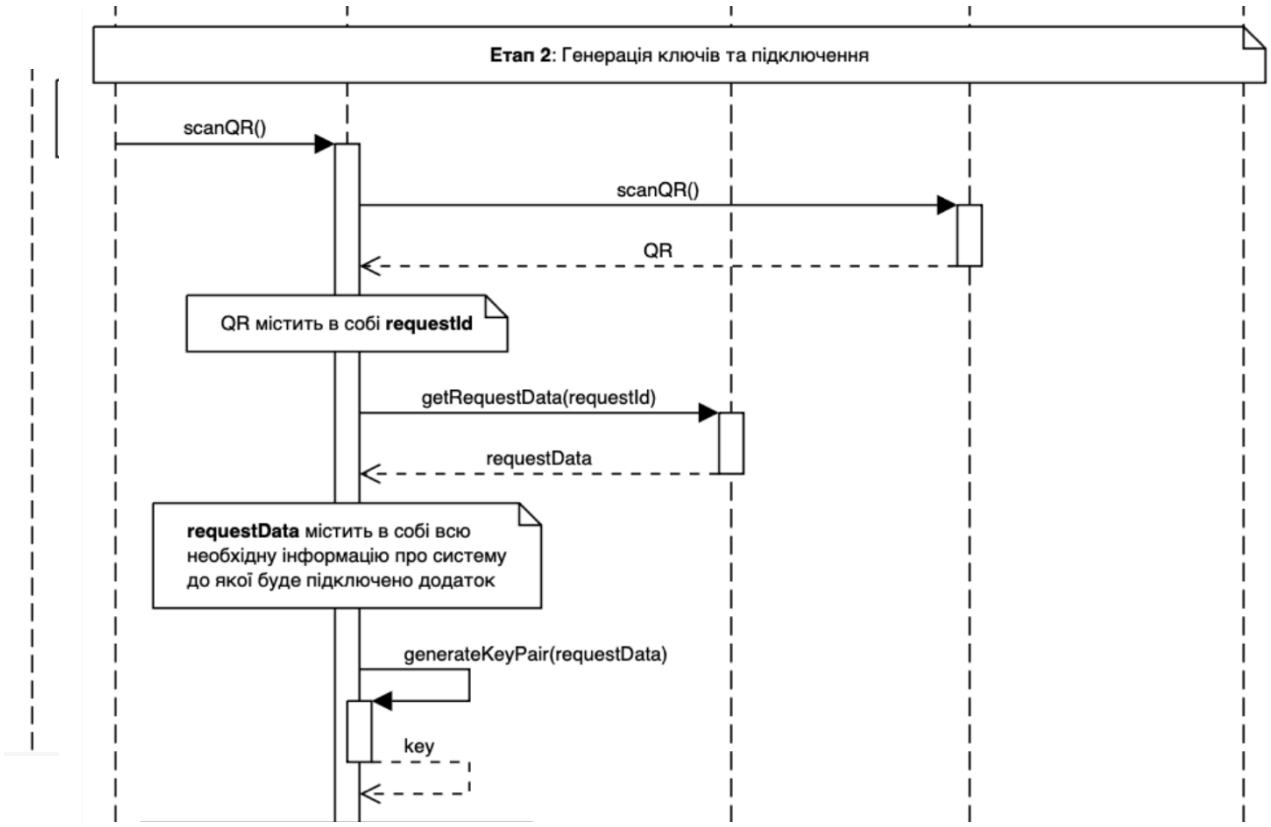


Рисунок 3.3 – Діаграма послідовностей для першого етапу підключення

Попередньою умовою для підключення мобільного пристрою є те, що користувач вже пройшов ідентифікацію, тобто ввів логін та пароль у web-застосунок. На першому етапі користувачем створюється запит на підключення мобільного додатку до web. Тобто в інтерфейсі web-застосунку(на діаграмі Web Application Front) користувач обирає підключення двофакторної автентифікації за допомогою згаданого мобільного застосунку, який умовно буде називатися KeyTool (на діаграмі Mobile App). Очевидно, що web-застосунок має дозволяти використання KeyTool як другого фактору автентифікації, тобто застосунки мають бути заздалегідь інтегровані. Web Application Front ініціює запит до Web App Backend на підключення KeyTool, після чого Web Backend відправляє запит до Mobile App Backend на створення підключення. Mobile App Backend повертає Web App Backend у відповідь на запит requestID – ідентифікатор підключення, який в свою чергу Web Application Backend повертає у відповідь Web App Front. Web App Front на основі

requestID генерує QR-код та відображає його в інтерфейсі, що робить його доступним для сканування користувачем.

На другому етапі відбувається генерація ключів та безпосередньо саме підключення. В даному випадку обов’язковою передумовою є встановлення застосунку KeyTool на мобільний пристрій. Користувач відкриває KeyTool на мобільному пристрої, де вже вбудовано QR-сканер та сканує згенерований на Web AppFront QR-код. На рисунку 3.4 наведено інтерфейс додатку KeyTool для описаного функціоналу.

Рисунок 3.4 – Інтерфейс мобільного застосунку KeyTool для сканування QR-коду

Після цього Mobile App KeyTool відправляє запит getRequestData(requestID) на Mobile App Backend, де requestID – це ідентифікатор підключення KeyTool до web-застосунку, який KeyTool отримав під час сканування QR. У відповідь мобільний додаток KeyTool отримує requestData, яка містить в собі всю необхідну інформацію про web-застосунок, до якого буде підключено KeyTool. Користувач має підтвердити підключення в інтерфейсі KeyTool як показано на рисунку 3.5.

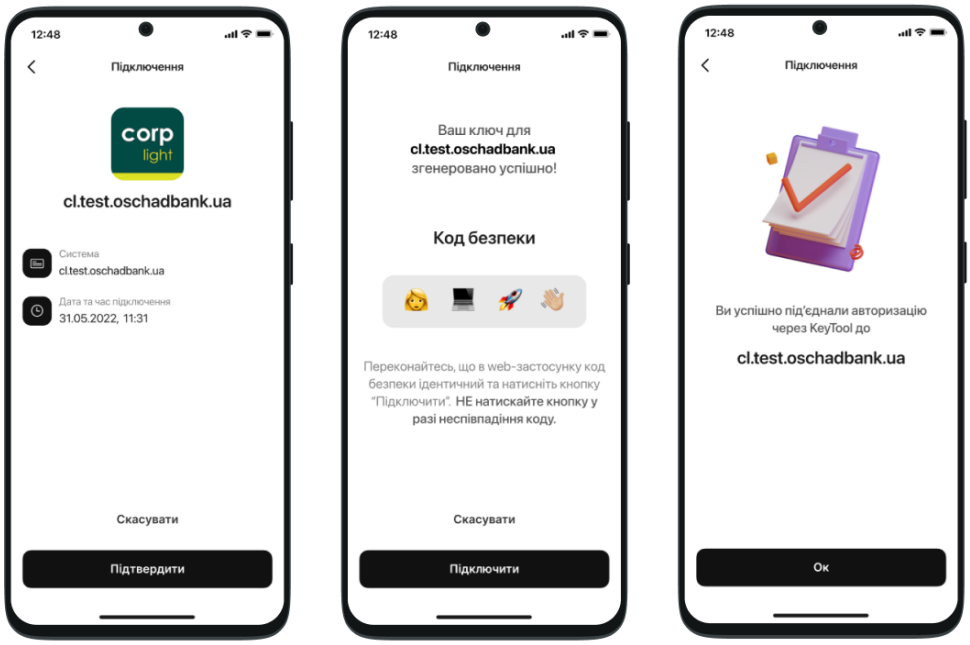


Рисунок 3.5 – Підтвердження підключення KeyTool до web-застосунку

У випадку, якщо користувач підтвердив підключення, Mobile App KeyTool генерує ключ на основі requestData за допомогою функції generateKeyPair(requestData). Згенерований ключ складається з двох частин: публічного ключа publicKey та приватного ключа privateKey. Після цього KeyTool передає на Mobile App Backend зв'язку requestID та publicKey за допомогою функції connect(requestID, publicKey), що означає формування зв'язку між мобільним додатком та web-застосунком. Mobile App Backend в свою чергу передає отриманий публічний ключ publicKey на Web Application Backend, на цьому процес підключення мобільного застосунку KeyTool до web-застосунку завершується. Мобільний застосунок зберігає зв'язку requestID та key(повний ключ) в локальній базі даних. Після завершення цього процесу користувач буде бачити в інтерфейсі мобільного застосунку KeyTool наявність ключа до web-застосунку, тобто інформацію про підключену систему.

Наступним кроком є безпосередньо сам процес авторизації користувача у web-застосунок. На рисунку 3.6 наведено діаграму послідовностей першого етапу процесу авторизації, а саме створення запиту на авторизацію у web-застосунок.

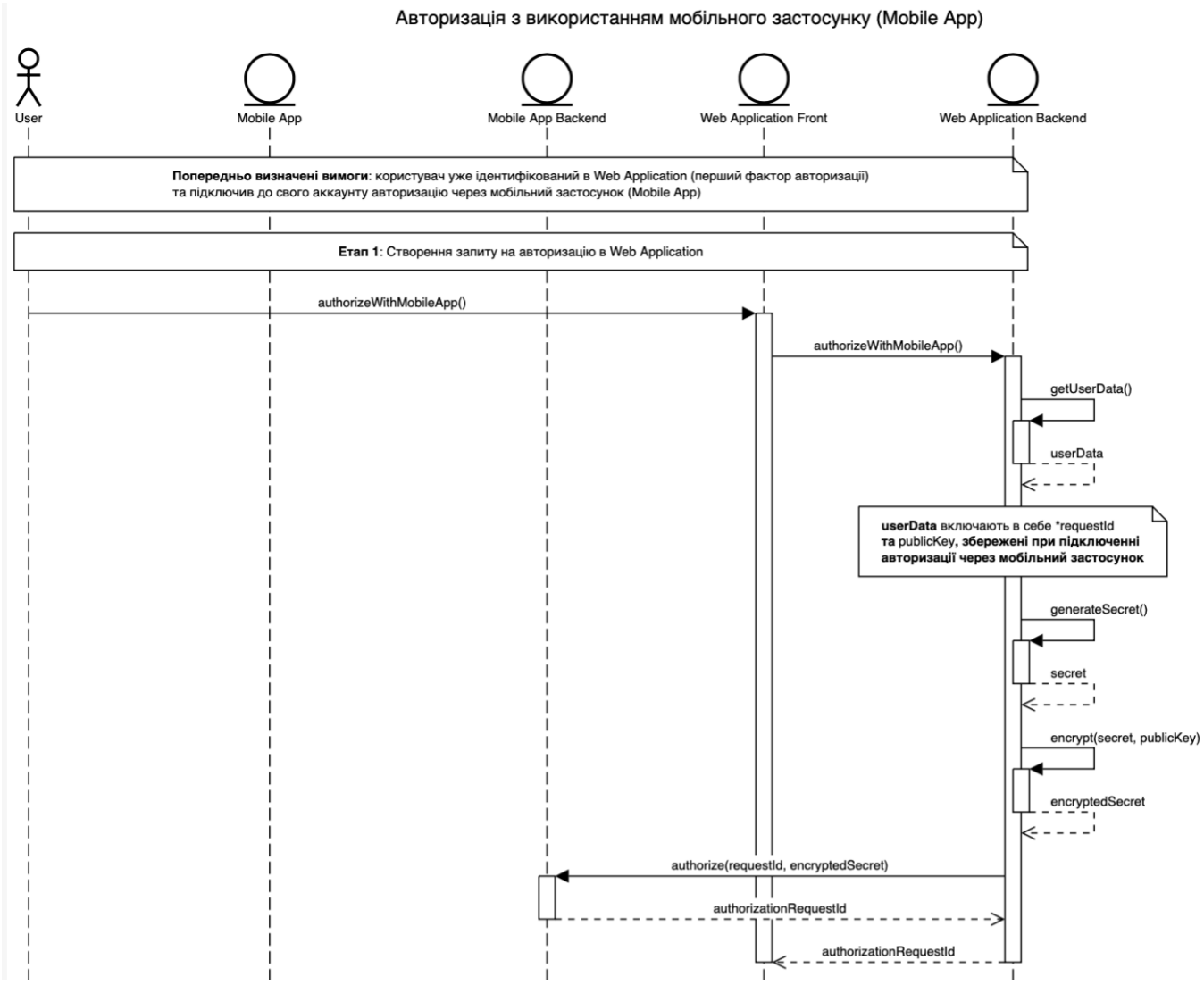


Рисунок 3.6 – Створення запиту на авторизацію у web-застосунок

Як тільки користувач пройшов етап ідентифікації, у web-застосунку з’являється повідомлення, що підключена двофакторна автентифікація з використанням KeyTool і користувачу необхідно підтвердити вхід. Web Application Front відправляє запит на Web Application Backend про намір користувача автентифікуватися. Web Application Backend обробляє запит, з підключеної бази даних збирає інформацію про користувача – userData, яка також включає в себе requestID (ідентифікатор підключення KeyTool до web-застосунку) та publicKey, які було збережено на етапі підключення мобільного застосунку KeyTool як другого фактора автентифікації. Далі Web Application Backend генерує секрет secret – це будь-який текст, яким web-застосунок ідентифікує запит на авторизацію та виконує його шифрування за допомогою publicKey. Після чого створює запит, в якому

надсилає Mobile App Backend requestID та зашифрований секрет encryptedSecret, тобто створює безпосередньо запит на авторизацію, Mobile App Backend у відповідь надсилає authorizationRequestID.

У разі успішного створення запиту на авторизацію, наступає етап підтвердження авторизації у web-застосунок за допомогою KeyTool, схема якого зображена на рисунку 3.7.

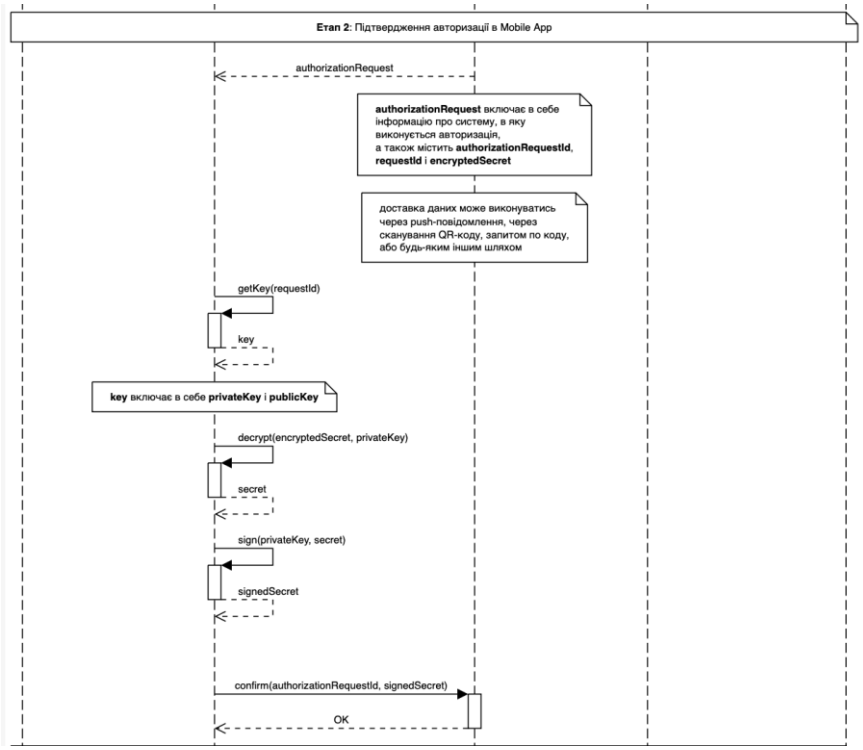


Рисунок 3.7 – Схема підтвердження авторизації у web-застосунок

Mobile App Backend відправляє push-повідомлення у додаток KeyTool, щоб сповістити користувача про необхідність підтвердження авторизації. У повідомленні передаються дані запиту на підключення authorizationRequest, які включають інформацію про систему в яку виконується авторизація, а також authorizationRequestID (ідентифікатор запиту на авторизацію), requestID та encryptedSecret. Можливі альтернативи push-повідомлення, тобто доставки даних: сканування QR-коду, запит по коду, або будь-яким іншим шляхом. Далі у мобільному застосунку користувач підтверджує авторизацію, використовуючи PIN-код або сканер відбитку пальця або FaceID, як показано на рисунку 3.8.

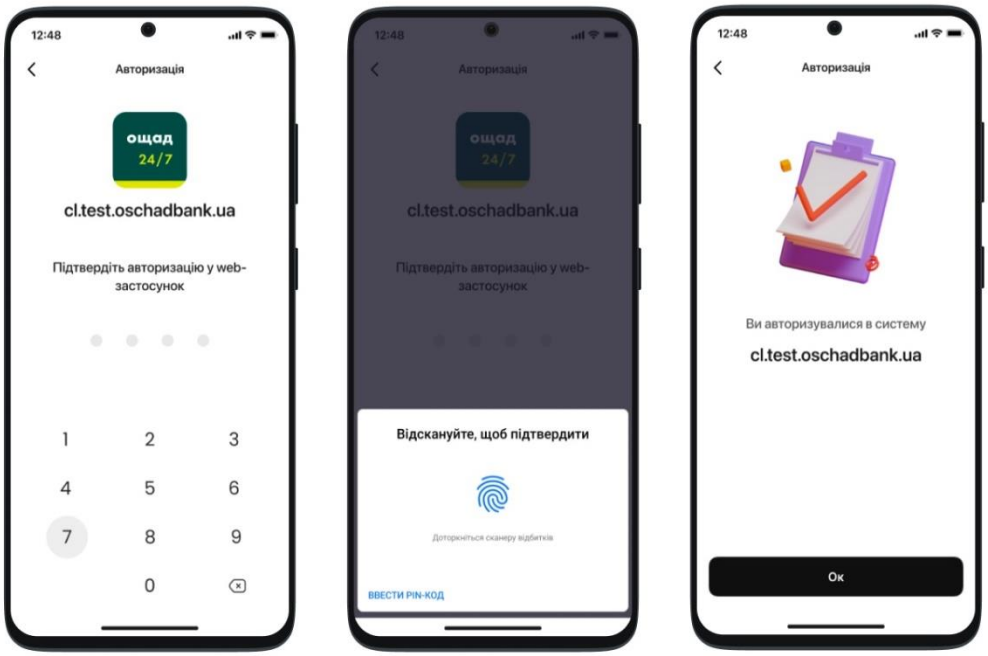


Рисунок 3.8 – Підтвердження авторизації до web-застосунку у KeyTool

Далі починаються внутрішні процеси, мобільний застосунок KeyTool. Розшифровує приватним ключем зашифрований секрет – decrypt(encryptedSecret, privateKey) та підписує «чистий» секрет приватним ключем – sign(privateKey, secret). Після чого мобільний застосунок створює запит на Mobile App Backend та відправляє підписаний секрет signedSecret у зв’язці з authorizationRequestID, що технічно являється підтвердження авторизації з боку користувача.

На третьому та останньому етапі перевіряється авторизація самим web-застосунком. На рисунку 3.9 наведено діаграму послідовностей для третього етапу.

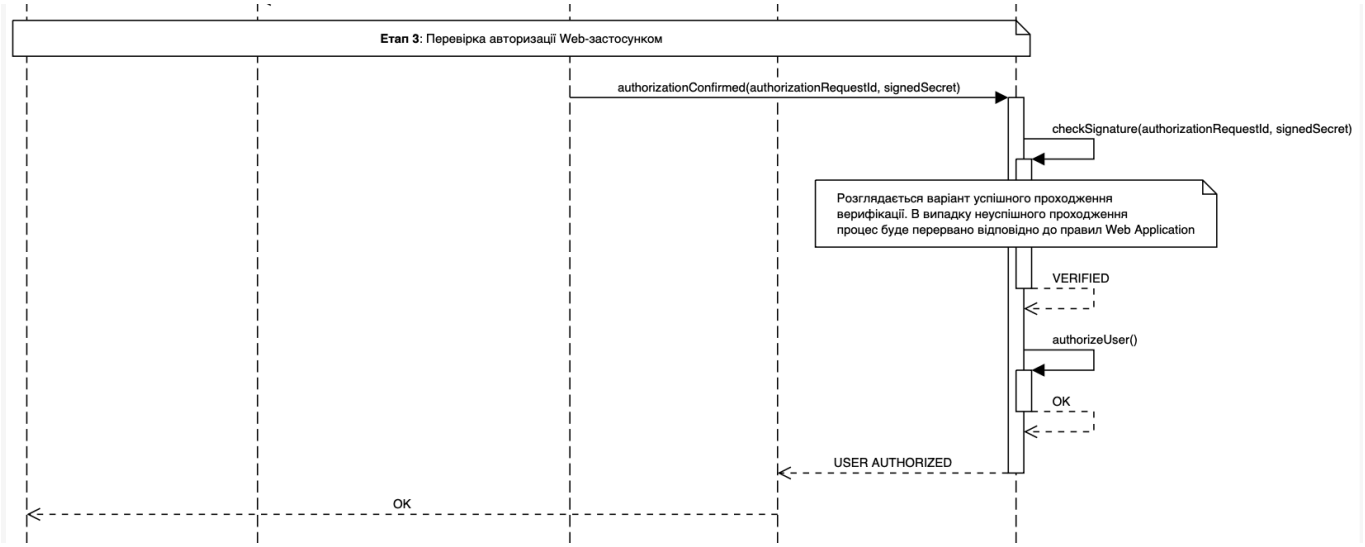


Рисунок 3.9 – Перевірка авторизації web-застосунком

Mobile App Backend відправляє запит на Web Application Backend і передає authorizationRequestID у зв'язці з підписаним секретом signedSecret, та фактично підтверджує авторизацію користувачем. Web Application Backend перевіряє валідність підписаного секрету(для перевірки валідності електронно цифрового підпису достатньо публічного ключа, який уже збережений в Web Application Backend). Після успішного проходження валідації секрету, виконується авторизація користувача у web-застосунок.

3.3 Інтеграція web-застосунку з мобільним застосунком

Для інтеграції web-застосунку з мобільним застосунком було розроблено відкрите API за специфікацією REST. API розгортається на стороні бекенду мобільного застосунку та містить в собі набір запитів, для створення і контролю статусів запитів на підключення користувача та авторизації.

Всі запити до API є авторизованими, механізм авторизації та дані для доступу web-застосунок отримує на етапі інтеграції з пропонованим сервісом. Механізм є стандартним (OAuth 2.0) та не є частиною даного дослідження, оскільки ніяк не впливає на його результати.

Специфікація POST запиту для створення запиту на підключення мобільного застосунку у якості другого фактору автентифікації до web-застосунку показана на рисунку 3.10.

`POST` ***/api/connect***

Body:

```
//empty
```

Response:

200 - OK

```
{
  "requestId": "string",
  "status": "PENDING"  // [PENDING, CONFIRMED, TIMEOUT, CANCELLED]
}
```

4xx - Error

```
{
  "message": "string"
}
```

Рисунок 3.10 – Специфікація запиту для створення запиту на підключення

Специфікація GET запиту для перевірки статусу запиту на підключення показана на рисунку 3.11. У випадку коли відповідь на запит GET прийшла 200 – ОК і при цьому значення статусу “CONFIRMED”, з'являється додатковий параметр “data”, який містить в собі параметр “ publicKey” типу string в форматі base64.

Рисунок 3.11 – Специфікація запиту для перевірки статусу запиту на підключення

Специфікація POST запиту для створення запиту на авторизацію у web-застосунок показана на рисунку 3.12.

Рисунок 3.12 – Специфікація запиту для створення запиту на авторизацію

Специфікація GET запиту для перевірки статусу запиту на авторизацію показана на рисунку 3.13. У випадку коли відповідь на запит GET прийшла 200 – ОК і при цьому значення статусу “SIGNED”, з'являється додатковий параметр “data”, який містить в собі параметр “ signedSecret” типу string в форматі base64.

Рисунок 3.13 – Специфікація запиту для перевірки статусу запиту на авторизацію

3.4 Опис програмної реалізації ключових елементів мобільного застосунку підсистеми

Програмна реалізація наведена для мобільних пристроїв на базі ОС Android.

Генерація ключа. Коли мобільний застосунок отримує запит на підключення нового web-застосунку він повинен згенерувати для такого запиту спеціальний ключ. Для практичної реалізації описаного рішення може бути використано

довільний криптографічний алгоритм з відкритим ключем. В даному рішенні використовується криптографічний алгоритм RSA.

В результаті генерації буде отримано ключ, який складається з двох частин: приватної та публічної. Ключ сформовано таким чином, що його приватна частина знаходиться в захищеній області пам’яті мобільного пристрою і не може бути звідти отримана ніяким чином. Всі операції з приватним ключем виконуються також в рамках захищеної внутрішньої операційної системи Android (Android Keystore System) без вивантаження приватного ключа за її межі. Жоден інший додаток не має можливості отримати доступ до такого ключа.

Код для генерації ключа для додатку операційної системи Android буде виглядати наступним чином (мова програмування Kotlin):

```
fun generateKey(keyId: String) {
    val keyPairGenerator = KeyPairGenerator.getInstance(
        KeyProperties.KEY_ALGORITHM_RSA,
        "AndroidKeyStore"
    )
    val parameterSpec: KeyGenParameterSpec = KeyGenParameterSpec.Builder(
        keyId,
        KeyProperties.PURPOSE_ENCRYPT
                or KeyProperties.PURPOSE_DECRYPT
                or KeyProperties.PURPOSE_SIGN
                or KeyProperties.PURPOSE_VERIFY
    )
   .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_RSA_OAEP)
   .setSignaturePaddings(KeyProperties.SIGNATURE_PADDING_RSA_PKCS1)
   .setDigests(KeyProperties.DIGEST_SHA256, KeyProperties.DIGEST_SHA512)
   .build()
    keyPairGenerator.initialize(parameterSpec)
    keyPairGenerator.generateKeyPair()
}
```

Генерація ключа відбувається в функції generateKey. Вхідні параметри:

- keyId - String - ідентифікатор ключа, за яким цей ключ буде доступний в майбутньому. В рамках рішення в якості цього параметру буде використовуватись значення requestId. Таким чином ідентифікатором ключа буде ідентифікатор підключення мобільного застосунку до web-застосунку.

Основні дії які виконуються:

1. Створення генератора ключів, з вказанням алгоритму для ключа (RSA) та місця його зберігання (AndroidKeyStore).

2. Ініціалізація параметрів ключа. Вказується його alias (keyId) та операції, для яких ключ може бути використаний (шифрування, дешифрування, підпис, верифікація).

3. Генератор ключів ініціалізується створеними параметрами.

4. Відбувається генерація ключа та збереження його в AndroidKeyStore.

Верифікація публічного ключа. В процесі підключення web-застосунку до мобільного застосунку виникає необхідність передати публічний ключ на сервер web-застосунку. Публічний ключ не є секретним, тому його перехоплення третьою стороною не є небезпечним. Але, існує можливість підміни такого ключа в процесі передачі. Для того, щоб уникнути такої підміни, необхідно щоб користувач самостійно перевірив чи є публічний ключ, який знаходиться в мобільному застосунку та публічний ключ, який отримав сервер, однаковими.

Публічний ключ - це довга послідовність бітів (довжина буде залежати від способу генерації ключів), яка може бути представлена в вигляді нечитабельного набору символів в форматі HEX або Base64. Порівняння таких наборів символів вручну є складно та займає багато часу. Для спрощення цієї процедури було розроблено алгоритм, який перетворює публічний ключ на послідовність unicode-символів, яка може бути інтерпретована як чотири emoji. Такий підхід значно спрощує звірку даних для користувача, оскільки перевірити ідентичність декількох emoji набагато простіше.

Код для генерації одного emoji на основі текстових даних наведено в додатку А (мова програмування Kotlin).

Наведене рішення містить в собі закінчену множину доступних emoji та функцію, emojiByIndex(), яка може повернути однозначний результат в вигляді одного emoji для будь-якого вхідного параметру в форматі цілого числа. Для того, щоб перетворити стрічку (string) з текстом до цілого числа використовується вбудована функція hashCode()

Для того, щоб перетворити публічний ключ в набір з чотирьох emoji буде використано наступний алгоритм:

1. Отримати публічний ключ в форматі HEX (текстове представлення бінарних даних).

2. Вирахувати значення SHA1 від публічного ключа в тестовому форматі.

3. Вирахувати перший emoji з використанням EmojiRepresentation, принцип роботи якого наведено нижче.

4. Вирахувати значення інших трьох emoji послідовно, в якості вхідного параметру використовуючи хеш SHA1 від значення отриманого на попередньому етапі.

5. Об’єднати всі чотири emoji в один рядок.

Код для такого рішення на мові програмування Kotlin наведено нижче:

```
fun getPublicKeyHex(keyId: String): String {
    val keyStore = KeyStore.getInstance("AndroidKeyStore").apply { load(null) }
    val certificate = keyStore.getCertificate(keyId)
    certificate.publicKey.encoded.joinToString(separator = "") { it.toHex() }
}

fun publicKeyToEmojis(publicKeyHex: String): String {
   val countOfEmojis = 4
   val emojis = StringBuilder()
   var sha1Hash: String = publicKeyHex
```

```
        repeat(countOfEmojis) {
            sha1Hash = sha1Hash.sha1()
            val emoji = EmojiRepresentation.represent(sha1Hash)
            emojis.append(emoji)
        }
        return emojis.toString()
    }
```

На рисунку 3.14 наведений інтерфейс мобільного застосунку для генерації emoji.

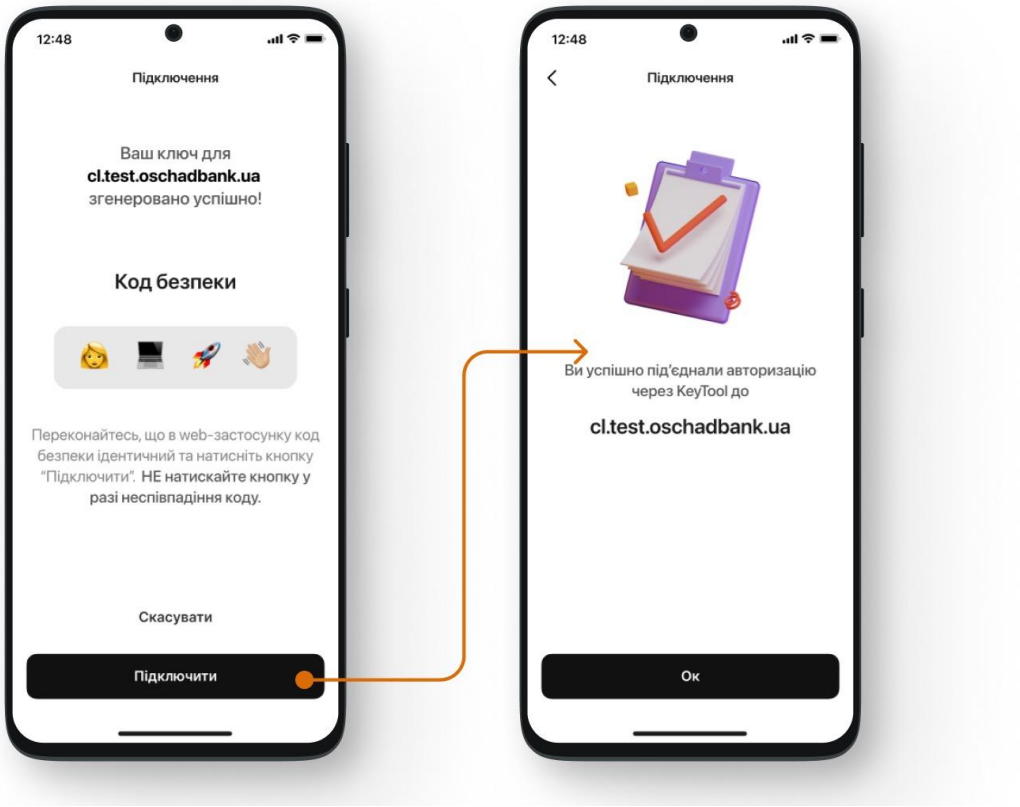


Рисунок 3.14 – Інтерфейс KeyTool зі згенерованими emojіна основі публічного ключа

Дешифрування секрету. “Секрет” - це будь-який текст, яким web-застосунок ідентифікує запит на авторизацію. На стороні web-застосунку “секрет” формується в результаті шифрування його публічним ключем отриманим від мобільного застосунку при підключені. Це є ще одним ступенем захисту, оскільки

розшифрувати такі дані можливо виключно з використанням приватного ключа, який знаходиться в додатку.

Код розшифрування “секрету” наведено нижче (мова програмування Kotlin).

```
fun decryptSecret(keyId: String, encryptedValue: String): String {
    val keyStore = KeyStore.getInstance("AndroidKeyStore").apply { load(null) }
    val cipher = Cipher.getInstance("RSA/ECB/OAEPWithSHA-256AndMGF1Padding")
    val key = keyStore.getKey(keyId, null) as PrivateKey
    val spec = OAEPParameterSpec(
        "SHA-256",
        "MGF1",
        MGF1ParameterSpec.SHA1,
        PSource.PSpecified.DEFAULT
    )
    cipher.init(Cipher.DECRYPT_MODE, key, spec)
    val bytes: ByteArray = Base64.decode(encryptedValue, Base64.NO_WRAP)
    return String(cipher.doFinal(bytes))
}
```

Дешифрування секрету відбувається в функції decryptSecret. Вхідні параметри:

- keyId - String - ідентифікатор ключа, яким необхідно виконати дешифрування;
- encryptedValue - String - зашифрований секрет.

Вихідний переметр:

- String - дешифрований секрет

Основні дії які виконуються:

1. Ініціалізації сховища ключів (AndroidKeyStore).

2. Ініціалізація об’єкта cipher алгоритмом дешифрування. Даний алгоритм обумовлений параметрами, які були вказані при генерації ключа.

3. Отримання доступу до ключа.

4. Ініціалізація параметрів розшифрування, ці парамтери також обумовлені алгоритмом ключа. ECB – режим шифрування, в даному випадку режим простої заміни.

5. Ініціалізація об’єка cipher для операції дешифрування.

6. Перетворення зашифрованого секрету з текстового формату в послідовність байт.

7. Розшифрування секрету.

Підпис секрету. Для того, щоб підтвердити авторизацію в мобільному застосунку необхідно накласти електронно-цифровий підпис (digital signature) на “секрет”, отриманий на попередніх етапах.

Електронно-цифровий підпис дозволяє підтвердити авторство електронного документа. В якості документа в рішенні, що пропонується, буде використовуватись “секрет”. Підпис пов'язаний з автором за допомогою криптографічних методів, і не може бути підроблений за допомогою звичайного копіювання.

Електронно-цифровий підпис накладається приватним ключем та може бути перевіреним будь-ким, хто володіє публічним ключем. При проходженні перевірки гарантується, що такий підпис міг накласти лише той, хто має доступ до приватного ключа. А, відповідно до описаних вище рішень та загальної архітектури мобільних операційних систем (Android та iOS), такий доступ є лише у конкретного мобільного застосунку на конкретному пристрої. Це робить пропоноване рішення безпечним до використання.

Код на мові програмування Kotlin наведено нижче.

```
fun signSecret(keyId: String, value: String): String {
    val keyStore = KeyStore.getInstance("AndroidKeyStore").apply { load(null) }
    val signature = Signature.getInstance("SHA256withRSA")
    val keyEntry = (keyStore.getEntry(keyId, null)) as KeyStore.PrivateKeyEntry
```

```
    signature.initSign(keyEntry.privateKey)
    signature.update(value.toByteArray())
    val signedBytes = signature.sign()
    return Base64.encodeToString(signedBytes, Base64.NO_WRAP)
}
```

Підпис секрету відбувається в функції signSecret.

Вхідні параметри:

- keyId - String - ідентифікатор ключа, яким необхідно виконати підпис
- value - String - секрет в текстовому форматі

Вихідний переметр:

- String - електронно-цифровий підпис в форматі Base64

Основні дії які виконуються:

1. Ініціалізації сховища ключів (AndroidKeyStore)
2. Ініціалізація об’єкта signature з вказанням параметрів хешування (обумовлено специфікацією ключа)
3. Отримання доступу до приватного ключа
4. Ініціалізація об’єкта signature приватним ключем
5. Встановлення даних, які необхідно підписати (секрету)
6. Підписання
7. Перетворення підпису з формату масиву байт в Base64-стрічку

У додатку Б представлені послідовність всіх можливих екранів мобільного застосунку для користувача.

Висновки за розділом 3

У розділі 3 було побудовано архітектуру підсистеми авторизації до web-застосунку з використанням мобільних пристроїв. Розроблена підсистема авторизації базується на використанні додаткового фактора автентифікації –

приватного ключа, який зберігається в захищеній області пам’яті мобільного пристрою.

В підсистемі фігурує web-застосунок, доступ до якого буде отримуватися, а точніше його Front-end та Back-end частини, та мобільний застосунок з його back-end частиною. Опис архітектури розділено на два великих етапи:

- Під’єднання мобільного застосунку до web-застосунку. Для початку користувачу необхідно «під’єднати» мобільний застосунок до web-застосунку, щоб він міг надалі фігурувати у процесі авторизації.
- Процес авторизації. Як тільки користувач розпочинає процес авторизації до web-застосунку, на етапі автентифікації необхідне додаткове підтвердження від користувача через підключений мобільний застосунок.

Безпека рішення пов’язана з використанням асиметричного шифрування. Асиметричне шифрування базується на використанні двох ключів: публічного та приватного. У рішенні використовується алгоритм RSA, який наразі вважається криптостійким.

Також було наведено програмну реалізації мобільного застосунку KeyTool, який фігурує у підсистемі, мовою програмування Kotlin. Описані найважливіші етапи функціонування застосунку, а саме:

- Генерація ключа (зв’язки публічного та приватного);
- Верифікація публічного ключа;
- Дешифрування «секрету»;
- Підпис «Секрету».

В розділі 3 та у додатку Б наведені можливі інтерфейси мобільного застосунку у випадку використання користувачем.

ВИСНОВКИ

У дипломній роботі було побудовано підсистему авторизації до web-застосунку з використанням мобільних пристроїв на основі асиметричного шифрування, а також представлено програмну реалізацію для мобільного застосунку, що фігурує у функціонуванні підсистеми.

У першій частині дипломної роботи було проведено аналіз функціонування web-застосунків. Аналіз складався з дослідження нормативно-правової бази в галузі інформаційної безпеки та діяльності web-застосунків (обробка інформації з різними рівнями доступності та методи захисту), огляд двох найпопулярніших архітектур для побудови web-застосунків, а саме монолітної та мікросервісної та дослідження механізмів взаємодії web-застосунків з мобільними застосунками на основі API - Application Programming Interface.

У другій частині дипломної роботи було розглянуто поетапно процес надання доступу до web-застосунків: ідентифікація, автентифікація та авторизація. Також проведено огляд найпоширеніших векторів атак на весь процес в цілому, та можливі методи запобігання. Найбільше уваги було приділено огляду методів автентифікації, оскільки практична частина роботи напряму пов’язана з процесом автентифікації, тож необхідно було розглянути всі можливі способи. Розглядалися наступні методи:

- Базова автентифікація
- Дайджест
- Автентифікація із застосуванням цифрового сертифікату
- Автентифікація із застосуванням смарт-карток та USB-ключів
- Багатофакторна автентифікація

У третій частині дипломної роботи на основі проаналізованих у першій та другій частині даних було розроблено архітектуру усієї підсистеми, в які фігурує користувач, мобільний застосунок та його Backend частина, web-застосунок та його Backend частина. Також представлений варіант реалізації мобільного застосунку для пристроїв на базі ОС Android, який виступає у якості другого фактора

автентифікації в підсистемі та функціонує з використанням алгоритму асиметричного шифрування RSA.

Виходячи із поставленої мети дипломної роботи були виконані наступні завдання:

- досліджено архітектури web-застосунків та механізми взаємодії з мобільними застосунками;
- проаналізовано процес надання доступу до інформації, методи автентифікації та їх вразливості;
- побудовано архітектуру підсистеми авторизації з використанням мобільних пристроїв;
- описано програмну реалізації мобільного застосунку для підсистеми.

Всі задачі було виконано в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мобільні телефони у світі [Електронний ресурс]. – Режим доступу: https://hromadske.ua/posts/mobilnimi-telefonami-u-sviti-koristuyetsya-bilshe-lyudej-nizh-tih-hto-maye-dostup-do-tualetiv-svitovij-bank

2. Скільки людей у світі використовують Інтернет [Електронний ресурс]. – Режим доступу: https://itsider.com.ua/stalo-vidomo-skilky-lyudej-u-sviti-vykorystovuyut-internet/

3. 27 Latest website statistics for 2022 [Електронний ресурс]. – Режим доступу: https://bloggingwizard.com/website-statistics/

4. Impressive password statistics to know in 2022 [Електронний ресурс]. – Режим доступу: https://webtribunal.net/blog/password-stats/#gref

5. Про інформацію [Електронний ресурс]: Закон України від 21.12.2019 № 2657-XII. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2657-12

6. Про затвердження Правил забезпечення захисту інформації в інформаційних, телекомунікаційних та інформаційно-телекомунікаційних системах [Електронний ресурс]: постанова КМУ від 29.03.2006 №373. – Режим доступу: https://zakon.rada.gov.ua/laws/show/373-2006-%EF#Text

7. Про доступ до публічної інформації [Електронний ресурс]: Закон України від 01.12.2019 № 2939-VI. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2939-17

8. Про захист інформації в інформаційно-комунікаційних системах [Електронний ресурс]: Закон України від 16.12.2020 № 1089-IX. – Режим доступу: https://zakon.rada.gov.ua/laws/show/80/94-вр#Text

9. Про захист персональних даних [Електронний ресурс]: Закон України від 16.12.2021 № 1971-IX. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2297-17#Text

10. Загальний регламент про захист даних [Електронний ресурс]: Регламент Європейського Союзу від 25.05.2018 № 679. – Режим доступу: https://uk.wikipedia.org/wiki/Загальний_регламент_про_захист_даних

11. Про Державну службу спеціального зв'язку та захисту інформації України [Електронний ресурс]: Закон України від 20.03.2020 № 3475-IV. – Режим доступу: https://zakon.rada.gov.ua/laws/show/3475-15

12. Monolithic vs. Microservices Architecture [Електронний ресурс]. – Режим доступу: https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59

13. Monolithic Architecture [Електронний ресурс]. – Режим доступу: https://www.techtarget.com/whatis/definition/monolithic-architecture

14. Мікросервісна архітектура [Електронний ресурс]. – Режим доступу: https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d

15. Плюси мікросервісної архітектури [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/261237/

16. Advanced Message Queuing Protocol [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/AMQP

17. Лучшая архитектура для MVP [Електронний ресурс]. – Режим доступу: https://habr.com/ru/company/otus/blog/476024/

18. Що таке API: простими словами про складне [Електронний ресурс]. – Режим доступу: https://hostiq.ua/blog/ukr/what-is-api/

19. Парадигми API [Електронний ресурс]. – Режим доступу: https://tproger.ru/articles/vybor-request-response-paradigmy-api-rest-rpc-ili-graphql/

20. Comparing API Architectural Styles: SOAP vs REST vs RPC [Електронний ресурс]. – Режим доступу: https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/

21. Top 10 Web Application security risks [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/

22. Access Control: Identification, Authentication and Authorization [Електронний ресурс]. – Режим доступу: https://www.thomasvitale.com/access-control-authentication-authorization/

23. What is Identification, Authentication and Authorization [Електронний ресурс]. – Режим доступу: https://cybersophia.net/articles/what-is/what-is-iaa/

24. Identification, Authentication, Authorization – What's the difference [Електронний ресурс]. – Режим доступу: https://imageware.io/identification-authentication-authorization-difference/

25. Мандатне керування доступом [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Мандатне_керування_доступом

26. Керування доступом на основі ролей [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Керування_доступом_на_основі_ролей

27. Автентифікація (веб) [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Автентифікація_(веб)

28. Basic Access Authentication [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Basic_access_authentication

29. Дайджест автентифікація [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Дайджест_автентифікація

30. What is digest authentication? [Електронний ресурс]. – Режим доступу: https://stackoverflow.com/questions/2384230/what-is-digest-authentication

31. Що таке і як працює PKI? [Електронний ресурс]. – Режим доступу: https://pki.com.ua/pki_help.html

32. X.509 [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/X.509

33. Автентифікація за допомогою сертифікатів [Електронний ресурс]. – Режим доступу: https://it.wikireading.ru/59915

34. What is access control? [Електронний ресурс]. – Режим доступу: https://www.citrix.com/solutions/secure-access/what-is-access-control.html

35. Common access control attacks and how to fend them off [Електронний ресурс]. – Режим доступу: https://houseofit.ph/blog/common-access-control-attacks-and-how-to-fend-them-off

36. Mitigating access control attacks [Електронний ресурс]. – Режим доступу: https://resources.infosecinstitute.com/certification/mitigating-access-control-attacks/

37. Three surprising ways your password could be hacked [Електронний ресурс]. – Режим доступу: https://resources.infosecinstitute.com/topic/3-surprising-ways-your-password-could-be-hacked/

38. Атака типу Man-In-The-Middle: що треба знати кожному [Електронний ресурс]. – Режим доступу: https://www.imena.ua/blog/man-in-the-middle/

39. Асиметричне шифрування [Електронний ресурс]. – Режим доступу: https://studfile.net/preview/9094212/page:19/

ДОДАТКИ

ДОДАТОК А

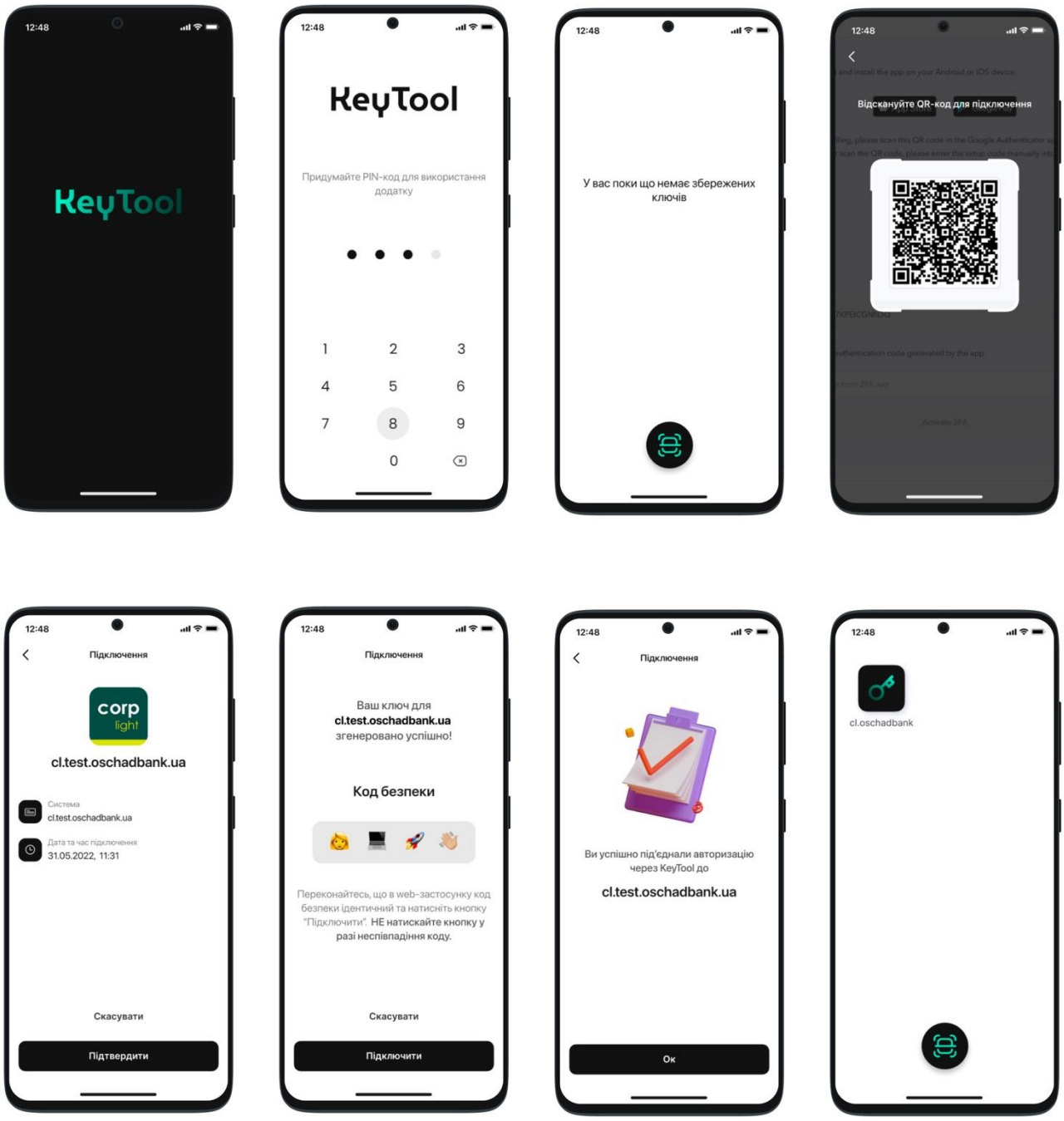
Програмний код генерації emoji

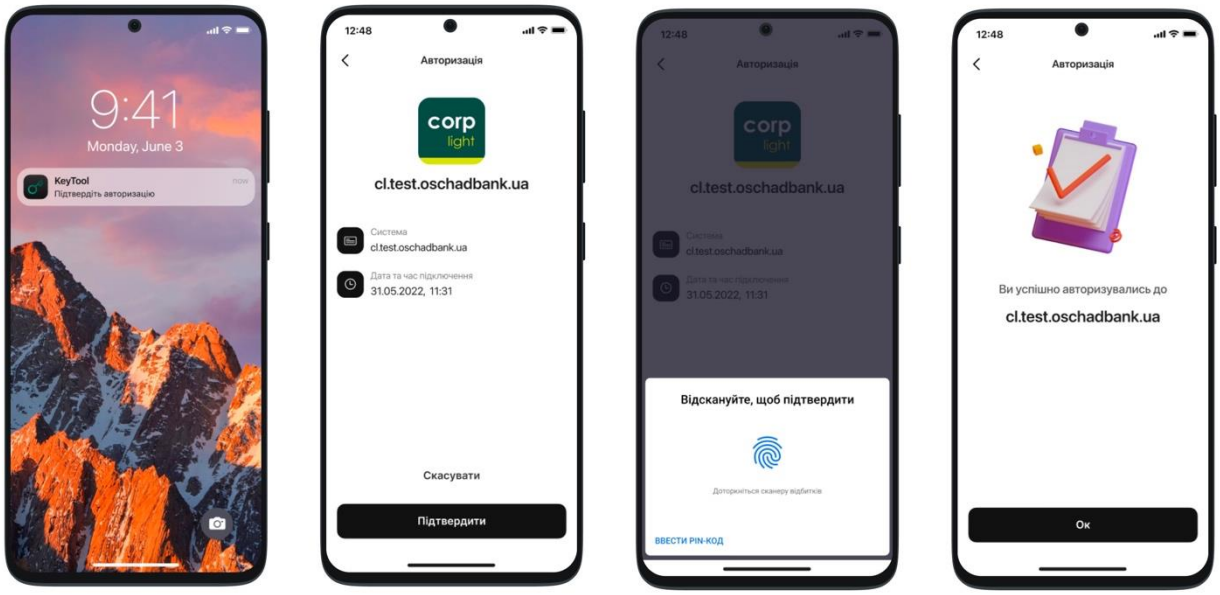
```
object EmojiRepresentation
    private val emojis: List<String> = listOf(
        "\uD83D\uDE00", "\uD83D\uDE03", "\uD83D\uDE04", "\uD83D\uDE01",
"\uD83D\uDE06", "\uD83D\uDE05",
        "\uD83D\uDE02", "\uD83E\uDD23", "\uD83E\uDD72", "\uD83D\uDE0A",
"\uD83D\uDE07", "\uD83D\uDE42",
        "\uD83D\uDE43", "\uD83D\uDE09", "\uD83D\uDE0C", "\uD83D\uDE0D",
"\uD83E\uDD70", "\uD83D\uDE18",
        "\uD83D\uDE17", "\uD83D\uDE19", "\uD83D\uDE1A", "\uD83D\uDE0B",
"\uD83D\uDE1B", "\uD83D\uDE1D",
        "\uD83D\uDE1C", "\uD83E\uDD2A", "\uD83E\uDD28", "\uD83E\uDDD0",
"\uD83E\uDD13", "\uD83D\uDE0E",
        "\uD83E\uDD78", "\uD83E\uDD29", "\uD83E\uDD73", "\uD83D\uDE0F",
"\uD83D\uDE12", "\uD83D\uDE1E",
        "\uD83D\uDE14", "\uD83D\uDE1F", "\uD83D\uDE15", "\uD83D\uDE41",
"\uD83D\uDE23", "\uD83D\uDE16",
        "\uD83D\uDE2B", "\uD83D\uDE29", "\uD83E\uDD7A", "\uD83D\uDE22",
"\uD83D\uDE2D", "\uD83D\uDE24",
        "\uD83D\uDE20", "\uD83D\uDE21", "\uD83E\uDD2C", "\uD83E\uDD2F",
"\uD83D\uDE33", "\uD83E\uDD75",
        "\uD83E\uDD76", "\uD83D\uDE31", "\uD83D\uDE28", "\uD83D\uDE30",
"\uD83D\uDE25", "\uD83D\uDE13",
        "\uD83E\uDD17", "\uD83E\uDD14", "\uD83E\uDD2D", "\uD83E\uDD2B",
"\uD83E\uDD25", "\uD83D\uDE36",
        "\uD83D\uDE10", "\uD83D\uDE11", "\uD83D\uDE2C", "\uD83D\uDE44",
"\uD83D\uDE2F", "\uD83D\uDE26",
        "\uD83D\uDE27", "\uD83D\uDE2E", "\uD83D\uDE32", "\uD83E\uDD71",
"\uD83D\uDE34", "\uD83E\uDD24",
```

```
        "\uD83D\uDE2A", "\uD83D\uDE35", "\uD83E\uDD10", "\uD83E\uDD74",
"\uD83E\uDD22", "\uD83E\uDD2E",
        "\uD83E\uDD27", "\uD83D\uDE37", "\uD83E\uDD12", "\uD83E\uDD15",
"\uD83E\uDD11", "\uD83E\uDD20",
        "\uD83D\uDE08", "\uD83D\uDC7F", "\uD83D\uDC79", "\uD83D\uDC7A",
"\uD83E\uDD21", "\uD83D\uDCA9",
        "\uD83D\uDC7B", "\uD83D\uDC80", "\uD83D\uDC7D", "\uD83D\uDC7E",
"\uD83E\uDD16", "\uD83C\uDF83",
        "\uD83D\uDE3A", "\uD83D\uDE38", "\uD83D\uDE39", "\uD83D\uDE3B",
"\uD83D\uDE3C", "\uD83D\uDE3D"
    )
    fun represent(value: String): String {
        val hash = value.hashCode()
        return emojiByIndex(index = hash)
    }

    private fun emojiByIndex(index: Int): String {
        val recountedIndex = normalizeIndex(index)
        return emojis[recountedIndex]
    }

    private fun normalizeIndex(index: Int): Int {
        return index.absoluteValue % emojis.size
    }
}
```

ДОДАТОК Б

Екрани мобільного застосунку




9:41
Monday, June 3
KeyTool
Підтвердіть авторизацію
12:48
Авторизація
corp light
cl.test.oschadbank.ua
Система
cl.test.oschadbank.ua
Дата та час підключення
31.05.2022, 11:31
Скасувати
Підтвердити
12:48
Авторизація
corp light
cl.test.oschadbank.ua
Система
cl.test.oschadbank.ua
Дата та час підключення
31.05.2022, 11:31
Відскануйте, щоб підтвердити
Доторкніться сканеру відбитків
ВВЕСТИ PIN-КОД
12:48
Авторизація
Ви успішно авторизувались до
cl.test.oschadbank.ua
Ок