МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНЕ НЕКОМЕРЦІЙНЕ ПІДПРИЄМСТВО
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КІБЕРБЕЗПЕКИ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри кібербезпеки

_____Анна ІЛЬЄНКО
"_____"_____ 20__ р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ "МАГІСТР"

Тема: Метод авторизації користувачів до web-ресурсів з використанням біометричних даних

Виконавець: Євгеній ГОРБАТЮК

Керівник: к.т.н., доцент Андрій ПЕТРЕНКО

Нормоконтролер: к.т.н., доцент Андрій ПЕТРЕНКО

Київ 2024

ДЕРЖАВНЕ НЕКОМЕРЦІЙНЕ ПІДПРИЄМСТВО
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Факультет комп'ютерних наук та технологій
Кафедра кібербезпеки
Освітній ступінь магістр
Спеціальність 125 «Кібербезпека та захист інформації»
Освітньо-професійна програма «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри кібербезпеки

_____Анна ІЛЬЄНКО
«30»___08_____2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Горбатюка Євгенія Олександровича

1. Тема кваліфікаційної роботи: Метод авторизації користувачів до web-ресурсів з використанням біометричних даних
затверджена наказом ректора від 30.08.2024 р. №1695/ст.
2. Термін виконання роботи: з 30.08.2024 по 15.12.2024
3. Вихідні дані до роботи: проаналізувати сучасну проблематику існуючих методів авторизації та автентифікації до web-ресурсів, їх надійність, а також можливість використання біометричних даних для їх заміни. Розробити веб-додаток для демонстрації можливості використання біометрії для доступу до

веб-ресурсів та продемонструвати його ефективність в якості складової інших систем, де потрібна авторизація.

4. Зміст пояснювальної записки: Аналіз предметної облісті, проблематика; Механізм біометричної автентифікації; реалізація методу біометричної автентифікації.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу: презентація.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Провести аналіз проблемної області	30.08.2024 – 10.09.2024	*Виконано*
2.	Визначити недоліки сучасної авторизації, запропонувати власне рішення	11.09.2024 – 19.09.2024	*Виконано*
3.	Спроектувати власний веб-додаток, знайти технічні засоби реалізації	20.09.2024 – 08.10.2024	*Виконано*
4.	Реалізувати власний веб-додаток	09.10.2024 – 27.10.2024	*Виконано*
5.	Протестувати веб-додаток	28.10.2024 – 05.11.2024	*Виконано*

7. Дата видачі завдання: «30» __08__ 2024 р.

Керівник кваліфікаційної роботи: _____ Андрій ПЕТРЕНКО

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання: _____ Євгеній ГОРБАТЮК

(підпис здобувача вищої освіти) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Метод авторизації користувачів до web-ресурсів з використанням біометричних даних»: 97 с., 16 рис., 4 табл., 40 джерел.

Об’єкт дослідження: системи авторизації користувачів на веб-ресурсах.

Предмет дослідження: методи біометричної авторизації та їх інтеграція у веб-ресурси.

Мета кваліфікаційної роботи: розробити та дослідити метод авторизації користувачів на веб-ресурсах із використанням біометричних даних для підвищення рівня безпеки та зручності.

Методи дослідження: теоретичний аналіз існуючих рішень, порівняльний аналіз методів біометричного доступу з апаратної та програмної сторін, реалізація методу авторизації з використанням WebAuthn та скануванням обличчя.

Практична цінність: розроблений метод авторизації може бути впроваджений у веб-ресурси для підвищення рівня безпеки, мінімізації ризику компрометації даних користувачів та покращення користувацького досвіду.

Наукова новизна: поєднання різних типів авторизації підвищує зручність користувача, а використання біометрії допомагає усунути недоліки традиційних методів авторизації за допомогою логіну та паролю.

ASP.NET, C#, FACEAPI, FIDO2, NEXT.JS, WEB, WEBAUTHN, АВТЕНТИФІКАЦІЯ, АВТОРИЗАЦІЯ, БІОМЕТРІЯ, БРАУЗЕР, ВХІД, САЙТ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

Акаунт	–	(Account) скорочена назва облікового запису користувача, розмовний термін
Креденшл	–	(Credential) облікові дані
ПК	–	Персональний Комп’ютер
ШІ	–	Штучний Інтелект
CORS	–	(Cross-Origin Resource Sharing) спільне використання ресурсів різними джерелами
DI	–	(Dependency Injection) впровадження залежностей, паттерн проектування, в якому залежності об’єкту передаються йому ззовні, замість створення їх всередині об’єкту
EF	–	Entity Framework
QR	–	(Quick Response) швидка відповідь, спеціальний двомірний штрих-код, який може бути швидко розшифрований камерою пристрою
SIM	–	(Subscriber Identity Module) модуль ідентифікації абонента
SMS	–	(Short Message Service) СМС, служба коротких повідомлень
SPAM	–	(Spam Promotion and Advertising Message) – СПАМ, небажані та непрошені повідомлення, які несуть масовий характер, та частіше за все використовуються для реклами чи обману
JWT	–	(JSON Web Token) – стандарт створення токенів доступу, що використовуються для авторизації та автентифікації
WebAuthn	–	(Web Authentication) стандарт автентифікації на web-ресурсах без використання паролів

ВСТУП

Актуальність. Дана розробка та тема є актуальними, так як ми живемо у цифрову епоху, де існує безліч web-сервісів та ресурсів, до більшості з яких потрібно мати зареєстрований обліковий запис. Наразі спостерігається тенденція, що облікові записи прив'язуються до електронної пошти чи номеру телефону, і для входу чи відновлення доступу використовуються вони: код підтвердження по СМС, чи посилання на електронну пошту. Це – потенційна вразливість, так як номер телефону може бути втрачено, чи скомпрометовано, як і електронну пошту. Ще один фактор – при втраті доступу до засобів відновлення паролю, при його забутті, система відновлення доступу через службу підтримки на більшості сервісів недосконала, вимагає документи, тощо, та займає багато часу на розгляд та вирішення.

Цей проект має на меті вирішити це питання та автоматизувати процес відновлення доступу до облікового запису, без участі служби підтримки та витрачення часу на це.

Метою кваліфікаційної роботи є аналіз існуючих методів реєстрації, авторизації та відновлення доступу до облікових записів користувачів, та розробка власного методу, який використовує біометричні дані користувача в якості підтвердження його особи.

Для досягнення цього, передбачається виконання **таких задач**:

– проаналізувати існуючи методи отримання доступу до облікових засобів, виявлення їх плюсів та мінусів;

– визначити, які біометричні дані є оптимальними для отримання доступу до облікового запису користувача;

– реалізувати власний web-застосунок, для демонстрації роботи запропонованого методу авторизації.

Галузь застосування. Даний метод авторизації може бути вбудований у будь-який web-ресурс, де є облікові записи користувачів, та бути як основним методом авторизації, так і додатковою системою безпеки та запобігання втраті

доступу.

Об’єктом дослідження є наявні методи авторизації, як типові, так і нетипові.

Предметом дослідження метод біометричної авторизації користувачів на web-ресурсах.

Методи дослідження полягають у виявленні вразливостей наявних наразі методів авторизації та автентифікації користувачів, та визначенні переваг у підході з використанням біометрії.

Новизна одержаних результатів. Даний метод авторизації є нетиповим, і йде у розріз із загальноприйнятими методами авторизації та відновлення доступу. Це дозволяє використовувати в якості ключа до облікового запису дані самої людини, проти використання номеру телефону чи електронної пошти, доступ до яких може бути втрачено, на відміну від власних якостей. Наразі авторизація за біометричними даними існує лише як додаткова система, для пришвидшення отримання доступу, у своїй більшості – на мобільних пристроях, але варто актуалізувати питання безпеки у мережі. Даний проект вирішує це питання також для більшості користувачів ПК чи ноутбуків.

Практичне значення цього методу в тому, що це:

– пришвидшить доступ до облікового запису;

– запобігає отриманню доступу іншими користувачами, крім власника облікового запису;

– запобігає втраті доступу до облікового запису;

– є гнучким методом, який можна використовувати як основний, чи додатковий метод отримання доступу.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ПРОБЛЕМАТИКА

1.1 Методи автентифікації до web-ресурсів та їх проблеми

Говорячи про облікові записи користувачів (також ще звані «акаунтами»), перш за все на думку спадає мережа інтернет та web-ресурси. Так, облікові записи існують не тільки там, а й в операційних системах пристроїв, таких як Windows [1], Android [2], тощо. Обліковий запис задумано, щоб різні люди, які користуються тим самим пристроєм, мали різні відокремлені простори для роботи, а також – різні права. Це ж стосується й web-технологій, де назва облікового запису «акаунтом» більш розповсюджена. Мається на увазі, що існує спільний web-ресурс, яким користується багато людей, і для їх ідентифікації, кожен з них має свій обліковий запис.

Облікові записи тісно пов’язані з питаннями кібербезпеки, так як саме їх потрібно захищати насамперед. Отримуючи доступ до облікового запису іншої людини, зловмисник отримує її так звану «чутливу» інформацію. Цього ніхто не хоче, а тому акаунти захищаються різними методами.

Подальший аналіз будемо проводити, спираючись на принципи функціонування та захисту соціальних мереж чи месенджерів [3], у якості web-ресурсів.

Перш за все, й найочевидніше – обліковий запис захищається паролем. Основні дані отримуються під час процесу реєстрації, коли людина реєструє новий обліковий запис та надає інформацію про себе, яка включає її ідентифікатор (Login/Username, «логін/ім’я користувача»), та пароль. Це – основні та найпримітивніші підходи, які лежать у основі сервісів й досі.

Для того, щоб дозволити користувачеві писати від імені певного облікового запису, людина повинна авторизуватися. Процес авторизації – це надання користувачеві певних прав доступу до ресурсу (у нашому прикладі – передання прав доступу до облікового запису, які включають у себе керування

ним, написання від його імені постів/коментарів, викладання записів на сторінці, фотографій, відео, тощо). Але процесу авторизації передує процес автентифікації – це ідентифікація користувача, як власника облікового запису [4].

Найочевидніше, що вводить користувач під час автентифікації – це ім'я користувача та свій пароль. Все просто: ім'я користувача шукається у базі даних ресурсу, потім порівнюються геші паролів [5], якщо все зійшлося – автентифікація успішна, користувач авторизується й отримує доступ до свого акаунту. Якщо ні, користувач отримує помилку, й йому пропонується пройти автентифікацію заново.

Проблематика, яка бачиться наразі у сучасній автентифікації – її неперсоналізованість. Так як персональними даними частіше за все виступають номер телефона користувача, або його електронна адреса. При такому підході, окрім паролю, користувач вводить свій номер телефону чи пошту, замість імені користувача (в залежності від реалізації), та автентифікується за допомогою них.

Подивимось глибше на номер телефону та електронну адресу, та зауважимо, що це – далеко не вибір, так як електронна пошта у своїй основі для реєстрації потребує номеру телефону [6]. Бачимо взаємозв'язок: для реєстрації/автентифікації на web-ресурсі необхідні електронна адреса чи номер телефону, проте вибір цей – ілюзорний, його немає, бо пошта реєструється на номер телефону. Сучасні мобільні пристрої мають по дві SIM-карти, тобто середньостатистичний користувач має два номери телефону, проте номери телефону можна використовувати для створення різних електронних адрес, використовуючи різних надавачів послуг обміну електронними листами (сервіси електронного листування з різних доменів). Основна думка тут в тому, що за основу персоналізації облікового запису відповідає номер телефону користувача, який може бути втрачено, змінено, тощо. Незважаючи на наявність електронної пошти, все одно все прив'язано насамперед до номеру телефону, й він є ключем до всього, що пов'язано з користувачем.

Ще одна закономірність – залежність від мобільних пристроїв. Маючи доступ лише до ПК, не маючи електронної адреси, користувачеві заблокована

можливість реєстрації, так як без номеру телефону й доступу до нього, руки зв’язані. Номер телефону, як основа персоналізації облікових записів у інтернеті, та як ключ до них – рішення не найкраще, але саме така тенденція зараз спостерігається.

1.2 Номер телефону як ключ до облікових записів

Окреме питання виникає, якщо розглянути саме поняття номеру телефону. Номер – це ідентифікатор у мобільній мережі деякого оператору стільникового зв’язку [7]. Окрім цього, номер може бути як персоніфікований, так і ні. Наразі, законодавство України не змушує користувачів мобільного зв’язку персоніфікувати свої номери телефонів, проте така можливість існує у мобільних операторів. Процес персоніфікації номеру мобільного телефону – це реєстрація та прив’язка номеру телефону до паспортних даних користувача, що запобігає незаконному та несанкціонованому перевиданню SIM-картки зловмисником без відома власника, так як для перевидання потрібно буде пред’явити паспорт чи інший документ, що посвідчує особу [8].

Розглянемо деякі неприємні ситуації, які пов’язані з номерами телефонів, які можуть бути причиною втрати облікових записів:

1. Дуже популярна зловмисна схема, яка працює й до сих пір полягає у тому, що оператор стільникового зв’язку надає можливість перевидання SIM-картки при її втраті чи пошкодженні, на той самий номер телефону. Доводиться це дуже просто: при бажанні випустити SIM з вже існуючим номером телефону, користувач звертається до оператору мобільного зв’язку, де отримує інформацію, що на нову SIM-картку може бути призначений цей номер, при умові доведення користувачем, що цей номер належить саме Вам. Доведенням для персоніфікованого номеру телефону виступають офіційні документи, що посвідчують особу, а для неперсоніфікованого номеру – частіше за все запитують останні номери телефонів, які взаємодіяли з цим номером (здійснювалися дзвінки). Цим і користуються зловмисники, навмисне

здійснюючи велику кількість дзвінків на номер жертви зі своїх номерів, а потім просто називаючи їх та час здійснення викликів [8-9]. При вдалій махінації, зловмисник отримує номер телефону, який належить жертві, а як наслідок, й доступ до всіх його облікових записів, як і до електронної пошти. Тут же маємо й доступ до банківських рахунків, так як мобільний банкінг використовує номер мобільного телефону як для входу, так і для скидання паролю.

2. Уявімо, що мобільний пристрій залишено без нагляду на деякий час. При цьому, зловмисник може без проблем, на знаючи паролів, просто скинути його за номером телефону чи по електронній адресі, ввести дані на своєму пристрої, зчитавши захисні коди чи посилання з СМС чи електронного листа, відправлених на пристрій власника облікового запису, а потім видалити ці дані з пристрою. Для цього достатньо не більше п’яти хвилин, якщо знати, що робити.

3. Телефон було загублено. Нехай він з паролем, але ніщо не заважає витягнути з нього SIM-картку, й відновити доступ до всіх облікових записів. Або ж, повертаючись до п.2 – поцупити SIM-картку з залишеного без нагляду пристрою.

4. Оператори перевидають SIM-картки з номерами, які вже були у використанні [10]. У номера телефону є термін дії, і зазвичай він визначається роком з моменту останнього поповнення рахунку. Таким чином визначається, користуються номером досі, чи ні, й звільняють номери, які вже не перебувають у використанні, втрачені, не приносять кошти. Приведемо приклад однієї знайомої, яка нещодавно втратила свій акаунт Telegram через перевидання SIM-картки. Telegram реєструється на номер телефону, що й зробила людина, але потім вона переїхала до Польщі, й більше року не користувалася своєю SIM. Як результат, SIM було перевидано, придбано іншою людиною, якою й відновлено доступ до акаунту Telegram, який їй не належить. Окрім цього, при спробі реєстрації електронної пошти Gmail, частіше за все матимемо попередження, що цей номер вже використовується у одному з облікових записів. Там є свої нюанси у відновленні та покращений захист, про що далі, у п.1.3 цього розділу.

Як бачимо, тенденція зі вразливістю облікових записів через номер

телефону з кожним роком зростає, так як все більше номерів й SIM йде на перевидання. Але з цим намагаються боротися не шляхом виключення SIM й номеру телефону як ідентифікатору, а всередині продуктів користування – шляхом деактивації акаунтів через деякий проміжок часу неперервної відсутності активності [11]. Цей шлях теж не зовсім доречний, так як людина може раптом згадати, що десь там давно було цікаве листування, чи важлива інформація, спробувати знайти її через рік, два, три, п’ять, й не мати змоги до цього, так як змінила за цей час кілька SIM, а акаунт давно деактивований.

Також, варто зазначити тут можливість створення так званих «фейкових» акаунтів, частіше за все створених для омани та маніпуляцій, розповсюдження СПАМу. Цьому сприяють чинники:

- Можливість реєстрації багатьох електронних адрес на один номер телефону, й використання їх на одному й тому ж сервісі для створення багатьох облікових записів;

- Відносно низька вартість SIM-карток й номерів телефону, які можна придбати у великій кількості, так як при купівлі SIM ніхто не вимагає документи.

Маємо лінійну залежність, у якій кількість можливих облікових записів для людини розраховується як кількість номерів телефону, якими вона володіє, помножена на кількість можливих для створення електронних адрес на один номер телефону. До таких акаунтів можуть підключатися боти, з них може автоматично розповсюджуватися різна інформація, тощо.

В додаток до вище сказаного, використання номеру телефона в якості ідентифікатору облікового запису має ще один неприємний нюанс. Користувачі можуть зіштовхнутися з тим, що хтось, вручну чи використовуючи комп’ютерну програму, почне СПАМити кнопкою «Забули пароль», використовуючи відомий номер телефону, й власник номеру буде завалений безліччю СМС повідомлень з відновлення доступу [12]. Це може продовжуватися нескінченно, і ніяк від цього не втекти, окрім блокування джерела надходження повідомлень, а повідомлення можуть бути від Kyivstar, месенджерів, соціальних мереж, тощо. Окрім цього, коди підтвердження, які надходять у СМС, іноді є дуже вразливими. Як

наприклад у того ж Kyivstar, який присилає чотиризначний числовий код, шанс «вгадати» який сягає 0.01%, що досить багато, а спроб вгадати його є кілька. Тому, використовуючи комп'ютерну програму для абсолютного перебору, можна як довести до сказу власника, так і, якщо пощастить, зламати обліковий запис оператору мобільного зв'язку.

1.3 Захист облікових записів від компрометування номеру телефону

Вразливість до компрометування номеру телефона не залишилася непоміченою, але помічено її не всюди, та якось дивно: про це знаюсь, й начебто щось з цим роблять, але цього явно недостатньо для гарантування конфіденційності персональної інформації та захисту. Наприклад, один з найрозповсюдженіших методів додаткового захисту акаунту від зламу – двофакторна автентифікація [13]. Але він має на увазі, що додаткове підтвердження йде через СМС чи лист на електронну адресу, що не працює від слова зовсім, якщо номер телефону скомпрометовано.

Тому, розробники web-ресурсів зазичай перевіряють при автентифікації чи відновленні доступу ще деякі дані, крім введених номеру/пошти та паролю. Серед таких даних у мережі інтернет частіше використовуються дані, які передаються у заголовку запиту за замовчуванням: ідентифікатор пристрою, геолокація заснована на IP, web-агент(User-Agent) – інформація про програму, з якої здійснюється запит [14]. Тому, при вході з нового пристрою, який раніше не використовувався, чи з іншої країни, тощо, застосовуються додаткові перевірки, які, тим не менш, все так само засновані на номері телефону, СМС, чи листі на електронну пошту.

Тут варто відмітити окремо захист, який реалізовано Google для їх Gmail-акаунтів [15]. Електронна пошта Gmail є основним обліковим записом для багатьох користувачів, через те, що Google володіє операційною системою Android, а тому, будь-який пристрій Android має бути прив'язаний до певного Gmail, але може бути прив'язаний й до кількох облікових записів одночасно. Так,

ця прив’язка все ще не обов’язкова, проте без неї користуватися телефоном буде набагато складніше: Google-акаунт забезпечує доступ до магазину додатків Google Play, дозволяє використовувати хмарні сховища Google Drive та Google Photos, синхронізувати контакти, тощо. Тому, мати Google-акаунт наразі є пунктом за замовчуванням, навіть якщо використовувати систему IOS, так як такий сервіс, як YouTube теж лежить у екосистемі Google, як і багато чого іншого.

Як же працює реєстрація та автентифікація Google? Насамперед, обліковий запис все ще реєструється на номер телефону [6], але може реєструватися й на іншу електронну пошту(наприклад, користувача створює реєструє пошту на домені «@aol.com», авторизується на YouTube та в процесі отримує Gmail-акаунт, прив’язаний до AOL-пошти). Можливість реєстрації пошти на іншу пошту – не панацея, тому що інша пошта все одно буде прив’язана до номеру телефону, проте це знижує вразливість: щоб зламати акаунт Google, потрібно знайти, на яку пошту він зареєстрований, а сервісів надання послуг електронного листування зараз дуже багато.

Разом з тим, Google використовує мобільний пристрій за основу та як ключ доступу до акаунту. Це реалізовано дуже цікаво, з думкою про користувачів. Так, наприклад, для входу у Google-акаунт з ПК, користувач вводить облікові дані, але не отримує доступ одразу. Спочатку його перевіряють через мобільний пристрій: пропонується ввести код, який приходить на пристрій у центр повідомлень, чи спливає посеред екрану. Або ж коли на ПК висвічується певне число, й потрібно натиснути на те ж саме число з кількох запропонованих на мобільному пристрої.

Тим не менш, навіть при такому підході вразливості залишаються. Згадуємо випадок залишеного без нагляду мобільного пристрою. Дізнатися з нього Gmail-адресу доволі просто, ввести собі у телефон чи ПК, та натиснути потрібні цифри на телефоні власника – доступ отримано, після чого видаляються всі повідомлення про вхід з листів Gmail та центру повідомлень. Займає не більше п’яти хвилин часу.

Також, ця система часто ставить самого власника пристрою у некомфортне положення, коли, наприклад, єдиний пристрій, до якого був прив'язаний акаунт – пошкоджено чи втрачено, людина є власником облікового запису, але не може увійти по правильній адресі/номеру та паролю, так як у неї знову й знову запитують код з неактуального пристрою, доступу до якого вже немає, або ж пропонують підтвердити по СМС, але доступу й до SIM-картки немає, телефон же втрачено.

При такому розвитку подій, також передбачено окремі перевірки й повернути доступ все ж можна, хоча й проблематично. Після довгих блукань по всім перевіркам, системи Google все ж «розуміють», що змоги підтвердити через вже авторизований пристрій у користувача немає, й починають запитувати інформацію по акаунту, яку може знати лише (чи не лише?) власник, такі як – ім'я та прізвище, які вказані при реєстрації, відповіді на секретні запитання, останній прив'язаний номер телефону. Омовка в тому, що цей підхід, як і ці засоби, можуть все ж таки не надати доступ навіть власнику: часто можна бачити повідомлення, що вказаних даних все ще недостатньо для ідентифікації користувача як власника облікового запису.

Якщо ж обліковим записом довгий час не користувалися, то виникає питання, чи дійсно початковий власник намагається увійти в нього. Це виникає, наприклад, при перевиданні SIM оператором, та купівлею її у звичайному магазині потенційним зловмисником. На прикладі того ж Google, це реалізовано прямим повідомленням, що цей обліковий запис довгий час був неактивним, тому здійснюється перевірка особи. Їй пропонується ввести ім'я та прізвище, які вказано для цього акаунту. Справа в тому, що це виглядає як надійний захист, так як ці дані з акаунту не отримати. Новий власник SIM не знає електронної адреси, до якої намагається отримати доступ, єдине що у нього є – це номер телефону, як наслідок – доступ до всіх підтверджень по СМС. Інформація Google-акаунтів не публічна, дізнатися ім'я та прізвище, дату народження просто зайшовши кудись й ввівши Gmail не можна. Але тут працює інша вразливість, яку люди самі й залишають, використовуючи свої справжні імена, прізвища, дати

народження. А знайти минулого власника номеру телефона можна багатьма способами – Instagram, Facebook, X, Telegram, тощо. Окремо існують web-ресурси, де зберігаються архіви сторінок VK, або інших, де можна побачити чий це був номер, та знайти всі потрібні дані. Після їх введення, Google вважає зловмисника власником, й надає доступ до облікового запису, хоча людина просто купила «нову» SIM та знайшла інформацію у відкритих джерелах.

Тому, захист від компрометування номеру телефона не ідеальний від слову «зовсім», й те, що номер досі використовується у сучасному світі як основний ідентифікатор виглядає як жарт.

1.4 Використання облікового запису для доступу до іншого облікового запису (OAuth)

Через величезну кількість web-ресурсів, де користувач повинен мати обліковий запис, з'явилися думки про те, що користувач легко у цьому заплутається й з часом просто буде забувати всі облікові дані, номери, імена користувача, паролі. А як відомо, гарною практикою безпеки облікових даних та доступу є використання окремих паролів для різних web-ресурсів. Але з такою кількістю, паролі доведеться записувати чи ще якось зберігати, їх не запам'ятати. Так, існують програми по типу 1Password [16], які це й роблять – запам'ятовують Ваші паролі, але чи довіряти таку чутливу інформацію якійсь програмі, хоч яку б безпеку вона не гарантувала? Так, існують менеджери паролів, як у самій системі Android, в якості паролів, що зберігаються у хмарі на Google-акаунті, так і на інших системах прями у браузерах. Це – також вихід, але не такий зручний, якщо подумати про перевстановлення системи (Windows), очищення даних (спеціальне чи випадкове, не завжди ініційоване власником пристрою).

Тому, з'явилася система, яка дозволяє авторизуватися за допомогою вже наявного акаунту. Найчастіше для цього використовується Google-акаунт, проте крім нього можна використовувати Microsoft-аккаунт, Apple ID, тобто найрозповсюдженіші типи облікових записів. Технологія має назву OAuth [17].

Функціонує це все через систему JWT-токенів. За цією системою, там де вона доступна, а доступна вона багато де, й все частіше це зустрічається, користувач повинен авторизуватися у Google через його обліковий запис Gmail, а для інших ресурсів та сервісів, користувач натискає кнопку «Увійти через Google» [18]. При цьому створюється запит на сервер Google, по якому формується JWT, що підтверджує користувача. Цей JWT зберігається на цільовому сервері ресурсу, у який користувач входить, та на пристрої користувача. При цьому, нових паролів та іншої інформації користувач не придумує та не надає, як і не надає інформації від Google, крім адреси своєї електронної пошти, імені акаунту, та фотографії профілю. Сервери спілкуються запитами окремо, перевіряють дані на валідність, що й забезпечує доступ. Такий спосіб входу ще зручний тим, що в Google можна переглянути всі сторонні сервіси, у яких користувач авторизувався за його обліковим записом Google, видалити непотрібні підключення, не пам’ятати зайві паролі. Таким чином, акаунт Google виступає як мастер-акаунт. Вразливість лише в тому, що при втраті цього мастер-акаунту чи його зламі, як і з номером телефону, користувач втрачає й всі сервіси, до яких він авторизувався через нього.

Розглянемо автентифікацію через JWT більш детально [19]. На рис. 1.1 зображено схему цієї автентифікації [20]. Як бачимо зі схеми, у ній фігурує браузер та сервер.

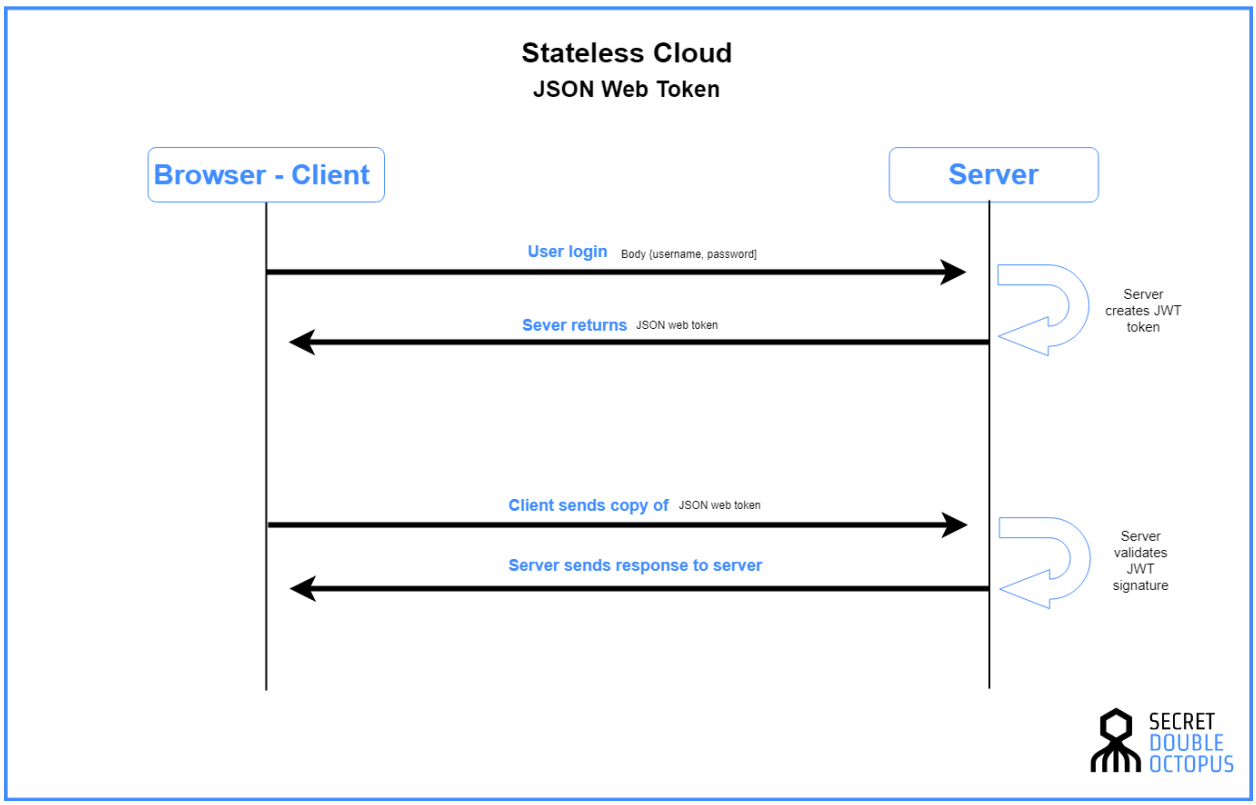


Рис. 1.1. Схема автентифікації за допомогою JWT

На прикладі входу на сторонній сервіс через Google-акаунт, маємо такий алгоритм OAuth, що працює всередині цієї реалізації:

1. Користувач хоче авторизуватися на сервісі (А) за допомогою свого Google-акаунту. При натисканні кнопки входу через Google, будучи авторизованим у Google, браузер користувача надсилає на сервер Google (Б) запит отримання JWT.

2. Сервер Google (Б) перевіряє валідність сеансу користувача, й формує JWT, який надсилається у браузер з відповіддю.

3. Отриманий JWT надсилається на сервер сервісу (А) у тілі запиту, чи через Cookie (в залежності від реалізації), де формується запис про цього користувача, якщо він входить вперше, чи знаходяться вже збережені облікові дані цього користувача.

Таким чином, доки користувач має доступ до свого Google-акаунту, та авторизований у ньому, він автоматично має доступ до всіх сервісів, у яких використовуються JWT від Google.

Ця схема – загальна, вона може використовуватися та надаватися будь-яким ресурсом, у якому використовується автентифікація традиційними методами, та якщо цільовий сервіс підтримує вхід через JWT цього ресурсу.

У висновку маємо гарний приклад розподілення обов'язків. Так, новим web-ресурсам та сервісам не потрібно перейматися про збір та зберігання облікових даних, особливо, якщо вони не гарантують їх надійне зберігання (інтернет-магазини, форуми, тощо), а користувачеві не потрібно запам'ятовувати нову зв'язку username/пароль для даного сайту. Задача перекладається на вже перевірені ресурси, яким можна довіряти, такі як Google чи Apple, де реалізовано всі перевірки автентифікації, відновлення доступу, тощо. Тому гіганти індустрії й дозволяють звертатися по JWT до них.

1.5 Методи відновлення доступу

Оскільки інтернет постійно розвивається та розширюється, щодня з'являється багато ресурсів та сервісів, які потребують реєстрації та наявності облікового запису. Це – популярні соціальні мережі, такі як Instagram, Facebook, X, месенджери по типу Telegram, Viber, WhatsApp, але й безліч інтернет-магазинів, тематичних форумів (наприклад, 4pda), хмарних сховищ (MEGA, Dropbox), і інших та інших.

По-перше, без функціонування OAuth-авторизації, користувачам було б дуже складно, так як виникають проблеми:

1. Різні сервіси мають різні вимоги до username/паролів, тому якщо користувач використовує звичну зв'язку паролю з шести цифр та трьох букв латинського алфавіту, а сервіс раптом вимагає додати сюди ще й символ, швидше за все при введенні паролю наступного разу через деякий час, користувач зіткнеться з труднощами.

2. За правилами власної кібербезпеки, до кожного ресурсу потрібен унікальний пароль [21], але цим часто нехтують. Дуже багато користувачів мають один чи два-три паролі на будь-які випадки життя, й користуються ними.

Уявімо, що деякий сервіс, де реєструється користувач, компроментує дані. Тоді решта сервісів, де використовується той самий пароль, під загрозою. Й навпаки, якщо користувач все ж додержується рекомендацій, то чим більше облікових записів він має, тим більше зв'язок usernames/паролів йому потрібно пам'ятати, зрештою – записувати чи користуватися хмарними менеджерами облікових даних.

3. Один сервіс пропонує реєстрацію лише на номер телефону, інший – лише на електронну пошту, третій – на вибір, інший – вимагає електронну адресу при реєстрації, але при вході просить ввести username, та тільки його.

Так, маємо проблему уніфікації даних. Це можна порівняти з телефонами 2000-х років та їх зарядними пристроями, коли різні виробники випускаючи різні моделі телефонів, щоразу придумували нові зарядні пристрої, які не підходили до інших мобільних пристроїв. А створення авторизації на основі JWT можна порівняти з введенням MicroUSB, а згодом й USB Type-C в якості стандарту. Як не дивно, уніфікації вимог до паролів та імен користувача досі немає, проте є JWT, який полегшує життя усім.

Очевидно й передбачено, що користувачі забувають свої облікові дані, особливо паролі. Так як випадки передбачені, то існують й рішення. Найчастіше, знову ж таки, вирішення реалізовано через інший сервіс. Якщо користувач не пам'ятає пароль, але у нього є доступ до номеру телефону чи електронної пошти (що очевидно, так як пошта й номер взагалі можуть існувати одні й ті ж самі для всіх облікових записів, й всі розуміють, що це - основа) – він може відновити доступ. Найчастіше, шляхом відправлення СМС з кодом підтвердження на номер телефону чи електронну пошту, або ж ще на електронну пошту можуть висилатися посилання для скидання паролю. Після цього, користувач повинен придумати й підтвердити новий пароль.

Ще одним розповсюдженим методом відновлення доступу є відповідь на секретне питання чи декілька таких питань [22]. Для використання цього методу, у користувачів запитують при реєстрації відповіді на певні питання, знати які потенційно може лише він. Розповсюджені варіанти питань: дівоче прізвище

матері, шкільне прізвисько, кличка домашнього улюбленця, рідне місто. Проте, деякі ресурси надають можливість не лише обрати питання з перелічених, але й придумати питання самому. Цей метод відновлення має місце, якщо доступ до мобільного телефону чи електронної адреси втрачений, але зараз використовується нечасто, або ж лише як етап додаткового захисту. Це не дивно, так як «дівоче прізвище матері» чи «кличку улюбленця» можна випитати у людини, чи вже знати, якщо ви з людиною близькі. А користувачі, які це передбачують, й вводять нереальні дані, ризикують забути й це.

З часом, метод відповіді на секретні питання дещо еволюціонував, й наразі використовується дещо інша система, яка називається «секретна/кодова фраза» [23]. Як приклад сервісів, де це реалізовано, можна привести proton.mail та крипто-гаманець у Telegram. Суть у тому, що при реєстрації, генерується випадковий набір слів, які не несуть спільного сенсу. При вході чи відновленні доступу, у користувача запитуються всю кодову фразу, або ж її окремі частини (наприклад, третє, п’яте та дев’яте слова). Якщо користувач вводить дані правильно – отримує доступ. Інакше – ні. Як і застерігають розробники, де реалізована ця система, фразу потрібно зберегти у окремий файл, який надійно зберігати, а втрата цього файлу може скінчитися неможливістю отримання доступу до облікового запису. Звідси й недоліки цього методу захисту:

- Слова не несуть спільного сенсу, тому запам’ятати набір навіть з десяти слів важко, а часто їх близько двадцяти.

- Потрібно зберігати кодову фразу у надійному місці, щоб її ніхто не міг поцупити та скористатися у власних цілях.

Від цього методу захисту у користувачів більше проблем, ніж користі. Але він існує та використовується, зокрема, як варіант відновлення доступу на proton.mail, а для крипто-гаманця Telegram – це основний та єдиний метод автентифікації.

Але якщо користувач взагалі не має доступу до жодного зі всіх методів відновлення? Вихід все ще є, і в такому разі користувачеві пропонується звернутися у службу підтримки використовуваного ресурсу [24]. При цьому

взаємодія людина-техніка змінюється взаємодією людина-людина. Деякі сервіси мають власні регламенти розгляду заявок на відновлення доступу до облікових записів, деякі – ні. Служба підтримки приймає рішення відносно наданої користувачем інформації. Спочатку, ідентифікується обліковий запис, до якого користувач намагається відновити доступ. Це може бути посилання, ідентифікатор, чи ім’я користувача, якщо він його пам’ятає. Потім, служба підтримки, маючи доступ до інформації у системі, запитує у користувача дані, які він пам’ятає. Це можуть бути попередні паролі, номери телефонів, електронні адреси, інформація профіля, перше використане ім’я на ресурсі, тощо. Інформацію запитують по максимуму, з кожною відповіддю наближаючи користувача до рішення надати йому доступ. Однак, найважливішим фактором абсолютної впевненості, що обліковий запис належить саме цій людині, є її документи. Зазвичай, вимагається щоб на обліковому записі були реальні фотографії користувача, а також справжнє ім’я, й ці дані повинні збігатися з офіційними документами особи. Іноді також використовується перевірка через друзів та пов’язані облікові записи, коли інші люди надають свої дані та підтверджують, що акаунт належить саме цій людині. Взаємодія зі службою підтримки не є оптимальним шляхом за кількома критеріями:

- Вона не дає абсолютної впевненості у тому, що обліковий запис належить цій людині й створений нею, навіть якщо ім’я та фотографії сходяться з документами.

- Відновлення займає багато часу.

- Рішення залежить від операторів підтримки. Це людський фактор, а тому результат може сильно варіюватися від працівника до працівника, залежати від його настрою, тощо, та чи вважає він надану кількість інформації достатньою.

- У наш час взаємодія людина-людина для відновлення облікового запису використовується вкрай рідно, й в основному офіційними структурами. Бачити такі методи відновлення на інших web-ресурсах вкрай дивно, тому що в наш час, з нашими технологіями, те, що може бути автоматизовано – має бути автоматизовано.

Навіть при наданні великої кількості інформації по обліковому запису, яка пам’ятається, може бути прийняте рішення не надати доступ, та навпаки – можуть надати доступ людині, яка взагалі не є власником. А така ситуативність в плані кібербезпеки недопустима.

1.6 Використання біометричних даних для авторизації

Використання біометричних даних наразі набуває всі більшої популярності. Спочатку ці методи ідентифікації використовувалися лише на об’єктах підвищеної секретності та у надзахищених системах [25]. Пов’язано це було з ціною датчиків для зчитування біометричних ознак.

Серед біометричних даних виділимо такі:

- Розпізнавання обличчя;
- Відбитки пальців;
- Сканування сітківки ока.

Біометричні дані – це особисті унікальні якості людини, які можуть бути використані для підтвердження особи. Особливого розповсюдження вони набули як метод захисту мобільних пристроїв, в якості швидшого та зручнішого способу блокування екрану.

Всі ми звикли до своїх смартфонів. Так як смартфон – портативний багатозадачний пристрій, він містить дуже особистих даних, в тому числі – додатки та браузер, у яких збережені доступи до різноманітних облікових записів. Окрім цього, у пристрою є пам’ять, а це конфіденційні фото та відео. Особиста інформація повинна захищатися, й основним шляхом захисту смартфону є блокування екрану. Другим шаром захисту виступає можливість окремого блокування додатків.

Блокування реалізовано кількома методами, серед яких пропонується обрати один [26]. Основні методи захисту:

- PIN-код (лише цифри);
- Пароль (будь-які символи);

- Графічний ключ (поле 3х3 точки, ключ формується довільним сполученням від 4 точок).

Серед цих методів бачимо новий, ще не описаний вище. Графічний ключ як метод блокування з'явився через особливості сенсорних екранів смартфонів, а зрештою – планшетів, мобільних ПК, ноутбуків з сенсорним екраном. Вище наведено основні методи блокування, проте є й альтернативні, менш розповсюджені, які використовують ті ж зручності сенсорного екрану. Так, наприклад, існує можливість розблокування екрану жестом – малювання пальцем символу на екрані, який порівнюється зі збереженим. Це чимось нагадує підпис, але простіше.

А деякі розробники реалізують взагалі унікальні ідеї блокування екрану чи додатків. Комусь було мало поля 3х3 графічного ключа, й він розширив поле введення до 10х10, та навіть більше. В іншому методі використані приховані точки, які потрібно правильно перетягнути, тощо.

Проте, не дивлячись на всі зручності та особливості сенсорного введення, еволюція прийшла з монтуванням в смартфони дактилоскопічних датчиків відбитку пальця. У наш час це дуже дешево та спрощує використання смартфону, це вже звичайність, яку навіть не помічаєш.

Система сканування відбитку пальця використовується як додатковий захист, основним все ще залишається PIN/графічний ключ/пароль, та його потрібно пам'ятати (про що й турбується більша частина смартфонів, запитуючи основний метод автентифікації хоча б раз на 72 години). Але використання відбитку дозволяє не вводити при розблокуванні екрану складний графічний ключ, 8-10-значний пароль/PIN. Дотик до датчику – й пристрій розблоковано. Водночас, це забезпечує приголомшливий захист, так як відбитки мають високу унікальність, а тому вірогідність того, що пристрій розблокують пальцем, який не вносили в пам'ять пристрою, прямує до нуля. Датчиків є кілька видів, в основному – від ємкісних, які сканують відбиток за допомогою електричного току (по принципу роботи самого екрану смартфону), до оптичних, які є чорно-білими камерами, які роблять знімок відбитку для його аналізу.

Серед переваг використання відбитку: цей метод отримання доступу використовує як ключ саму людину, та її власні особливості, які не потрібно запам’ятовувати, записувати, зберігати, тощо. Серед недоліків – людина може пошкодити палець, шкіру, й відбиток буде змінено на деякий час, але цей недолік нівелюється можливістю зберегти у смартфоні до десяти відбитків, тобто хоч по одному на кожний палець рук власника. Але, як вже сказано, завжди можна використати основний метод блокування, забути який сам пристрій користувачеві не дасть, й буде вимагати його введення кожні три доби. Звісно, ризики все ще є, наприклад – розблокування пристрою доторком датчику до пальця сплячого користувача, але це вже ближче до фантастики.

Інший розповсюджений метод розблокування з’явився дещо пізніше за розпізнавання відбитку пальця, й це – розпізнавання обличчя. Для цього зазвичай використовується або фронтальна камера пристрою, або додатковий датчик на передній панелі смартфону. Функціонує це по аналогії з датчиком відбитку, а саме – при взятті пристрою в руки, коли обличчя користувача потрапляє на камеру, блокування з пристрою знімається автоматично. Але сканування має свій ряд недоліків:

- Неможливість користуватися розпізнаванням у темряві (окрім розпізнавання не через фронтальну камеру, а через окремий датчик, який працює на інфрачервоному світлі, що дорожче у виробництві, або ж використання фронтального спалаху для камери, що у темряві не має сенсу, так як буде сліпити користувача).

- Спрощеність розблокування від сплячого чи утримуваного силоміць користувача.

Та ці дві системи без проблем працюють одночасно, додаючи зручності.

Зараз розблокування за допомогою відбитку та обличчя мають навіть ультра-бюджетні моделі смартфонів, що робить ці технології доступними взагалі всім бажаючим.

Що ж до автентифікації на web-ресурсах, та чому це досі не використовується в інтернеті? Ми бачимо тенденцію використовувати номер

телефону, електронну пошту, паролі, але не біометрію. Вона досі залишається мобільним ексклюзивом, хоча й потрібно обмовитися, що останнім часом все більше моделей ноутбуків виробляється з датчиком відбитку пальця.

Насправді, сказати, що біометрія взагалі не використовується при автентифікації в мережі – рівносильно брехні, так як така система все ж існує, але натрапити на неї просто користуючись мережею – важко. Ось в Україні є BankID [27], використовується лише у офіційних державних цілях для точної ідентифікації особи та надання їй спрощеного доступу.

Розглянемо приклад підтвердження особи через BankID. Це може виникати з різних причин: персоніфікація номеру телефону на сайті оператору, підписання петиції, реєстрація банківських рахунків, все що пов’язано з документами. Документи в електронному вигляді у громадян є у будь-якому банку, тому при запиті – банк може зі згоди користувача надавати доступ до цих документів. Сам обліковий запис для особистого банківського кабінету реєструється на номер телефону та пароль, але тут не про це. При запиті з сайту на підтвердження особи через мобільний пристрій, користувач, наприклад, обирає підтвердження через ПриватБанк [28], та отримує запит на відкриття додатку ПриватБанку. При відкритті, скоріше за все, користувач використовує спрощену систему входу в додаток, й щоб не вводити номер телефону та пароль у вже збережений обліковий запис, скористається відбитком пальця. У додатку вказується, що є запит на підтвердження особи за документами, й користувач натискає кнопку підтвердження, потім повертається на сайт, де дані вже отримано й дію виконано.

Ця система заслуговує особливої уваги, так як тут вже реалізовано біометричну автентифікацію, хоча вона й не ідеальна, й в цій системі взагалі біометрія не на першому плані. Хоча б тому, що якби користувач використовував не смартфон, а ПК, йому довелось би здійснювати вхід через номер/пароль, а ось з мобільним пристроєм все інакше.

Схему описаного вище входу наведено на рис. 1.2, її зручно зобразити засобами UML, а саме – діаграми послідовностей.

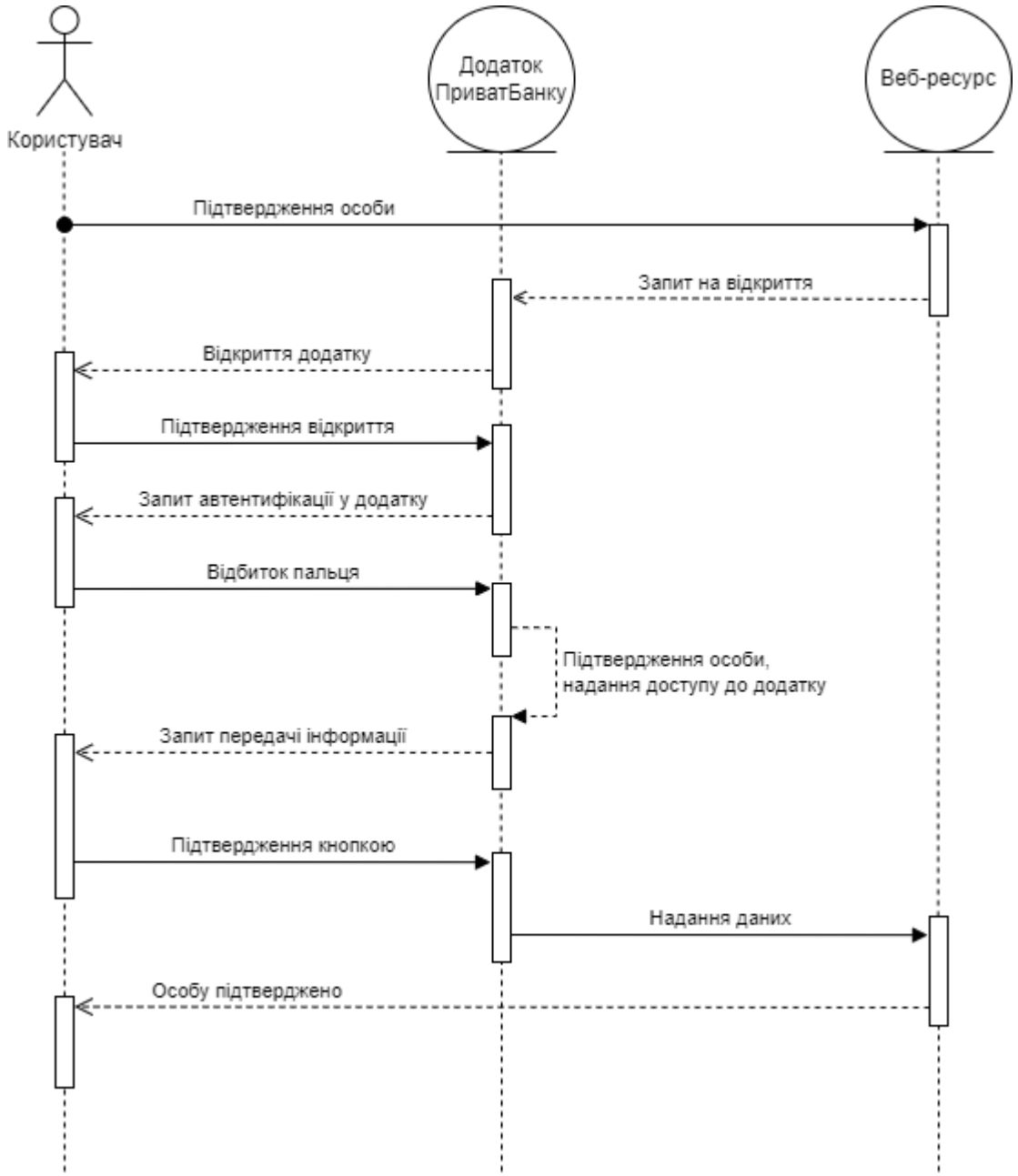


Рис. 1.2. Схема авторизації через BankID ПриватБанку

Зазначимо, що підтвердити відкриття додатку ПриватБанку з оновленнями стало можливо не тільки відбитком, але й за допомогою сканування обличчя.

За цим прикладом, якщо прибрати необхідність бути вже авторизованим у ПриватБанку й використовувати відбиток/обличчя як пришвидшений спосіб входу, а навпаки – взяти його за основу, маємо готову та робочу схему автентифікації на web-ресурсі за допомогою біометрії.

Як додаткові методи біометричної автентифікації, порівняємо такі:

розпізнавання обличчя, відбиток пальця, голос, сітківка (табл. 1.1).

Таблиця 1.1

Порівняння методів біометричної перевірки користувачів

Критерій	**Розпізнавання обличчя**	**Відбиток пальцю**	**Голос**	**Сітківка ока**
Надійність	Середня	Висока	Середня	Дуже висока
Зручність	Середня	Висока	Низька	Низька
Швидкість	Висока	Висока	Висока	Середня
Доступність для пристроїв	Висока	Середня	Висока	Низька
Вартість	Середня	Низька	Низька	Висока
Чутливість до зовнішніх умов	Висока (освітлення)	Незалежна	Висока (шум)	Незалежна

Виходячи з порівняння, сканування сітківки є максимально надійним методом, але найменш доступним. Голос використовувати незручно, й він залежить від шумів. Варто зупинитися на використанні відбитку пальцю (де він доступний), але й скористатися скануванням обличчя (для підтримки більшої кількості пристроїв, які можуть не мати сканеру відбитку пальцю).

1.7 WebAuthn в якості біометричної автентифікації

WebAuthn (Web Authentication) [29] – це сучасний стандарт для автентифікації на web-ресурсах, який дозволяє користувачам безпечно отримувати доступ до облікових записів без введення паролів до цих облікових записів. Звичайний користувач скоріше за все вже стикався з WebAuthn, але не звертав на це особливої уваги. Наприклад, на смартфоні цей стандарт використано як додатковий метод захисту при зверненні до менеджеру паролів Google, або при зміні налаштувань Google-акаунту, а на ПК – при вході в той

самий Google чи GitHub.

В основі стандарту лежить використання замість звичних паролів так званих «автентифікаторів». В якості автентифікатору можуть виступати фізичні ключи безпеки (FIDO2) [30], довірені пристрої та їх власні системи захисту, зокрема й біометрія. Розглянемо детальніше, як WebAuthn дозволяє уникнути використання паролю:

1. Фізичний ключ (FIDO2). FIDO2 – це протокол безпеки, а самі ключі – це фізичні пристрої, або ж програми. Виділяють платформонезалежні та платформозалежні ключі. Перший тип ключів частіше всього – окремий фізичний носій, по типу флешки, який підключається до цільового пристрою по USB, NFC, BLE. Дані передаються по окремому спеціальному протоколу CTAP. Платформозалежні ключі частіше всього є спеціальними програмами чи модулями на рівні операційної системи пристрою (наприклад, окремий чіп на материнській платі), доступ до якого надається тільки якщо вхід в систему успішний. Але у платформозалежних ключів є й мінус – кожен новий пристрій повинен реєструвати ключ окремо.

2. Довірений пристрій та метод доступу до нього. Випадок, коли в якості автентифікатору виступає сам пристрій, а для підтвердження дії використовується метод доступу до пристрою, а не до ресурсу. Зручно це тим, що пароль від ноутбуку чи смартфону користувач навряд забуде, так як ними він користується постійно, на відміну від введення паролів до web-ресурсів, де згадувати його доводиться найчастіше – кілька разів на місяць. Система ця працює наступним чином: якщо пристрій прив’язано до облікового запису, й він є довіреним, то замість запитувати пароль від облікового запису, web-ресурс запитує пароль від пристрою, а ось в якості паролю може бути як PIN, так і набір символів, графічний ключ, а ще й біометрія, яку взагалі не потрібно пам’ятати.

Цей тип автентифікації працює через відкритий та закритий ключі, тому це більш надійно, ніж пароль. Через мережу передається тільки відкритий ключ, сенсу й користі з якого для зловмисника небагато, в той час як приватні ключі надійно захищені на сервері чи на пристрої.

WebAuthn надає свій API, через який можна звернутися до пристрою користувача, й підтвердити пристрій, замість придумувати та пам’ятати паролі. Варто ще й зазначити, що якщо навіть й використовується біометрія, то відбитки пальців чи скани обличчя не зберігаються, що дозволяє не боятися «злиття» конфіденційних чи анонімних даних, так як біометрія використовується лише для формування ключів чи цифрових підписів, ніде не зберігаючись. Детальніше про це у розділі 2.

Таким чином бачимо, що стандарт WebAuthn має на меті полегшити доступ користувачів до облікових записів, та водночас – зміцнити традиційні методи підтримання захисту та безпеки. Взагалі, ця технологія не нова, з’явилася вона у 2016-му році, як частина роботи FIDO Alliance по створенню методів автентифікації, які не вимагають паролів. А от стандартом він став у березні 2019-го року. Технології не стоять на місці, зараз 2024-й рік, проте широкого розповсюдження стандарт не набув, й досі залишається на стадії переходу, коли його поступово впроваджують найбільші компанії та корпорації. На це є свої причини. Проте у контексті даної роботи, сама наявність та функціональність цього стандарту є дуже цінною, так як в тому числі, WebAuthn дозволяє автентифікуватися на web-ресурсах за допомогою біометрії. Єдине, що у ньому біометрія – далеко не основний метод автентифікації, а все ще другорядний.

1.8 Висновки до розділу 1

В підсумку, мається проблема автентифікації користувачів до web-ресурсів, яка криється у всім звичних та вже традиційних методах автентифікації.

Проблема криється у кількох чинниках:

- Символьний пароль, як основа безпеки користувача. Ці паролі, хоч і містять низку вимог, які гарантують їх надійність, не убезпечують їх повністю. Безпека користувача покладається на плечі самого користувача, а злами паролів відбуваються все частіше, особливо з появою та розвитком ШІ. Якщо раніше

можна було не боятися зламу методом грубої сили (BruteForce), просто створюючи пароль з використанням різних алфавітів та довжиною від 8-10 символів, то наразі ШІ здатні виявляти тенденції у паролях користувачів, через що злам методом перебору має набагато більше шансів на успіх. Окрім цього, символьні паролі забувають, записують та втрачають.

- Номер телефону. Так чи інакше, єдине, що надійно пов’язує користувача з його обліковими записами – це номер телефону. Обліковий запис реєструється на номер телефону чи електронну адресу, але електронна адреса обов’язково реєструється на номер телефону. Потенційні ризики очевидні, номери телефонів легко скомпрометувати, особливо методом SIM-swapping`у. Втрачаючи номер телефону, користувач ризикує всіма обліковими даними.

Якщо у символьного паролю й є переваги, серед яких – справжня надійність, якщо користувач тямить у безпеці, й тримає пароль лише у голові, не ведеться на фішингові посилання, не використовує у паролі імена чи дати, то у номеру телефону немає таких переваг, окрім простоти використання цієї системи.

Оператор не гарантує користувачу, що його номер не буде перевидано, чи його не отримає інша людина обманом, просто поцупивши SIM-картку, тощо. Оператор не веде аудит пристроїв, й не перевіряє коли й куди переставили SIM-картку, хто прочитав та видалив SMS, тощо.

Проект має на меті усунути цю зв’язку, надавши перевагу швидшому та безпечнішому методу автентифікації на web-ресурсах – використанню біометричних даних.

Під час дослідження виявлено, що доступ до облікових записів та інформації за допомогою біометрії вже подекуди реалізовано на деяких сервісах, і у всіх випадках елементом зручності виступає мобільний пристрій, так як він має дактилоскопічний сенсор відбитку пальця та камеру для зчитування обличчя. Зрештою, мобільний пристрій і так є основою при реєстрації чи втраті доступу, підтвердженні дій, тому його використання нічого не змінює, а повна відмова від залежності недоцільна, так як ноутбуки та ПК мають вбудовані

пристрої для зчитування біометричних ознак рідше, й це дорожче. Зараз камера та датчик відбитку пальця є навіть у найдешевших смартфонах, чого не сказати про комп’ютери.

Але, зараз отримувати доступ через біометрію можна лише як через другорядну та спрощену систему. Вона не береться за основу, вона лише економить користувачеві його час, при цьому залишаючи номер телефону, електронну пошту та пароль основними засобами автентифікації.

Проблема використання паролів та аналіз їх вразливостей не нові. Тому, зараз існує цікавий стандарт автентифікації, такий як WebAuthn, який поступово впроваджується, але дуже повільно – стандартом він став у 2019-му році [31], але досі широко не використовується. В якості автентифікаторів WebAuthn в тому числі використовуються й біометричні дані.

Проблематика використання номеру телефону очевидна, а серед факторів, чому від системи досі не відмовились:

- Традиційність. Користувачі й розробники просто звикли до використання зв’язки номерів телефонів, електронних адрес та паролів. Придумувати щось нове та впроваджувати його може бути як мінімум незрозумілим для кінцевого користувача.

- Номер в якості універсального ідентифікатора. Для багатьох компаній номер телефону пов’язується з людиною, так як він є у більшості людей, та за допомогою нього можна використовувати вже реалізовані підсистеми автентифікації – СМС, двофакторна автентифікація, скидання паролю. Це не змінюється, щоб не втрачати аудиторію та не створювати проблем з підтримкою.

- Маркетинг. При використанні номерів телефону, користувачів легко додати в бази для розсилок повідомлень рекламного характеру, чи продзвонів.

- Вкорінення системи. Реалізувати та підтримати номер телефону в якості ідентифікатору простіше й дешевше, ніж вигадувати біометричний вхід. В разі чого, компанія-розробник web-ресурсу не винна у втраті номеру телефону користувача, й до неї не буде претензій, на відміну від випадкового розповсюдження біометричних баз даних.

- Підвищення залежності від операторів мобільного зв’язку. Користувачі повинні підтримувати SIM-картку й номер активними, поповнювати рахунок, щоб номер не було перевидано, так чи інакше взаємодіяти з номером.

Щоб ці проблеми були закриті, необхідно, щоб метод авторизації з використанням біометрії був:

1. Безпечний, та без збереження чутливих біометричних даних користувача, для запобігання їх компрометування.

2. Швидкий та зручний, щоб не створювати зайвих проблем користувачам на фоні традиційних номеру телефону з символьним паролем.

Переваги використання біометрії у відмові від запам’ятовування паролів, використання номерів телефону, та підвищенні безпеки, так як біометричні дані складніше підробити чи спіймати на фішинг.

РОЗДІЛ 2.
МЕХАНІЗМИ БІОМЕТРИЧНОЇ АВТЕНТИФІКАЦІЇ

2.1 Фізичні пристрої біометричної автентифікації

Використання біометричних ознак для підтвердження користувача має низку переваг перед іншими методами. Серед них: швидкість, зручність, надійність. Проте, для того, щоб зчитувати біометричні ознаки людини, потрібні відповідні пристрої. Зосередимо увагу на найпоширеніших типах біометричної автентифікації – скануванні відбитку пальцю та обличчя [32].

Почнемо зі сканеру відбитку пальцю, він же – дактилоскопічний датчик. Цей пристрій наразі найпоширеніший, дуже дешевий у виробництві, й дуже важко знайти смартфон, у який би його не було вбудовано. Він компактний та зручний. Раніше його розташовували на задній панелі смартфонів, потім – почали суміщати з кнопками живлення збоку, чи апаратною кнопкою «Додому» на передній панелі, а потім – взагалі почали ховати сканер під екран. Так, сканер відбитку пальця – це те, до чого всі звикли й користуються ним у повсякденному житті, він спрощує доступ до пристрою та додатків, так як замість вводити паролі, користувач просто торкається пальцем у потрібному місці, й вже має доступ.

За типами, є кілька видів датчиків відбитку пальця, й вони мають власні особливості.

1. Ємкісні датчики. Працюють на основі вимірювання електричних зарядів, які з’являються при торканні пальця з поверхнею датчика, зчитують мікроскопічні особливості відбитку за рахунок різниць провідності на рельєфі структури шкіри. Вони швидкі, точні та безпечні, але мають недолік в тому, що зайва волога на шкірі заважає правильно розпізнавати відбиток, тому користуватися ними найкраще сухими руками. Саме ці датчики й вбудовують у більшість кнопок, вони й були раніше на задній панелі смартфону, та у кнопках «Додому».

2. Оптичні датчики. Вони є оптичними сенсорами, які можна порівняти зі специфічними камерами. Робиться знімок поверхні пальця за рахунок освітлення його невидимим для людини світла, наприклад – інфрачервоного. Саме ці датчики зазвичай встановлюються під екран пристрою. Але вони менш надійні, а також можуть гірше зчитувати інформацію, через шар скла дисплею.

3. Ультразвукові датчики. Використовують ультразвукові хвилі для зняття карти відбитку. Хвилі відбиваються від рельєфу шкіри, що дозволяє дуже точно передавати відбиток. Наразі це найбільш точна та безпечна технологія, яка працює навіть при забруднених чи вологих пальцях. Встановлюються у преміальні пристрої, наприклад Samsung Galaxy S-Series.

4. Теплові датчики. Отримують відбиток за рахунок різниць температур на випуклостях та впадинах шкіри, але використовуються рідко, так як значно поступаються точністю ємкісним та оптичним.

Як бачимо, дактилоскопічні датчики відрізняються технологіями, за якими вони функціонують, надійністю розпізнавання (супротиву зовнішнім чинникам), та рівнем безпеки. Так, можна виділити, що найкращими у плані безпеки є ємкісні та ультразвукові датчики, так як вони аналізуюсь саме рельєф поверхні, яка до них дотикається, а різниця в них у ціні – ємкісні дешевші й не мають супротиву до бруду й вологи, ультразвукові – дорожчі й незважають на ці фактори.

Далі розглянемо пристрої для сканування обличчя. Тут все набагато простіше, так як у будь-якому випадку сканується зображення, й все працює на відстані – через камеру.

Найпростіше та найдешевше, використовувати пристрій, який вже є. Це – фронтальна камера, яка за всі роки вже стала невід’ємною частиною будь-якого смартфону та ноутбуку, й наявна навіть на найбюджетніших пристроях. Але таке використання має дуже значний недолік: фронтальна камера початково не пристосована для такої задачі, й добре знімає зображення чи відео лише при денному світлі. І тоді, як у ноутбуках є яскравий дисплей й хоча б невеликий світлодіод для освітлення користувача у темний час доби чи приміщенні,

тенденція фронтального спалаху на смартфонах стає рідкістю. Її змінює функція екранного спалаху, коли екран смартфону на короткий проміжок часу стає чисто білим, а яскравість збільшується до максимальної. Говорити про зручність не доводиться – це постійно сліпило б користувача у темряві, й приносило б не зручність, а дискомфорт.

Цю проблему вирішує спеціальний сканер обличчя, який являє собою ту ж камеру, але чутливу до інфрачервоного світла, яка додатково оснащується й інфрачервоним спалахом. Як відомо, людське око не бачить інфрачервоного світла, тому сканування обличчя навіть у абсолютній темряві стає можливим без дискомфорту. Проблема тільки у тому, що спеціальні інфрачервоні сканери обличчя встановлюються у пристрої вище середнього сегменту, а частіше – у преміальні, тому про доступність тут мова не йде.

Ці два типи біометричної автентифікації беруться за основу, як найдоступніші й найзручніші. Наразі неможливо уявити пристрій сканування сітківки ока чи самої ДНК у мобільному пристрої, як і зручність його використання, на відміну від відбитку пальця чи обличчя. Ці методи дешеві, надійні, високого рівня безпеки, й дозволяють перевірити особу миттєво.

Зважуючи ці два методи, варто сказати й про їх доступність на різних типах пристроїв. Так, для ноутбуків та ПК поки що досі сканер відбитку пальця не є параметром «за замовчуванням», й з’являється на преміальних ноутбуках чи корпусах ПК, а ось камера вбудована у ноутбук (хоча й низької якості), й найчастіше є у власника ПК в якості периферії. Так, можна докупити сканер відбитку пальця як USB-пристрій, але це додаткові кошти й додатковий пристрій, тому розглядати це в якості доступності звичайному користувачу буде помилкою. А ось на смартфоні сканер відбитку є на більшості сучасних моделей, й взагалі моделей останніх років.

В контексті аутентифікації на web-ресурсах, можна розглядати як відбиток пальця, так і сканування обличчя, але варто враховувати як універсальність, так і безпеку. Так, виділимо такі пункти:

- ПК. Не має пристроїв зчитування біометрії «за замовчуванням», а отже

потребує додаткових витрат на периферію (USB, тощо).

- Ноутбук. Частіше за все має тільки вбудовану камеру поганої якості, рідше – має сканер відбитку пальця. Також може використовувати периферію, але суть ноутбуку – мобільність, а отже не має сенсу, так як не покращує життя користувача, а заважає йому.

- Смартфон. Більшість пристроїв мають сканер відбитку пальця, який вже є стандартом, безпекою та надійністю. Має фронтальну камеру, і якщо ПЗ підтримує розблокування по обличчю через камеру, або взагалі мається окремий сканер обличчя – може використовувати його.

Таким чином, бачимо різницю у пристроях, що варто враховувати. Пропонується відмовитися від використання ПК чи ноутбуків в якості зчитувачів біометричних даних, й взяти упор на використання саме смартфону. Цьому сприяють такі фактори:

- Доступність смартфонів. Смартфони у наш час використовуються всіма та усюди, тоді як ноутбуки чи ПК – більше для роботи та вдома. Смартфон придбати дешевше ніж комп’ютер, є люди, які мають смартфон й користуються мережею, та не мають комп’ютера, що ніяк не впливає на їх життя.

- Наявність пристроїв. Смартфони вже мають всі потрібні пристрої: обов’язковий сканер відбитку, фронтальна камера чи сканер для обличчя.

- Смартфон вже є автентифікатором. Повертаючись до розділу 1, було розглянуто, що всюди фігурує номер телефону чи/та електронна пошта, тому навіть реєстрація на web-ресурсі без смартфону просто неможлива. Окрім цього, популярні ресурси надають користувачеві доступ до облікових записів через смартфон: Google просить ввести коди, відправлені на телефон, або підтвердити через нього, Telegram/WhatsApp/Viber пропонують сканувати QR-код чи отримувати СМС.

2.2 Принцип роботи біометричного входу

При розгляді біометричної автентифікації слід зосередити особливу увагу

на тому, як збираються біометричні дані, й як вони використовуються, чому це безпечно, й у чому їх перевага.

Умовно, можна поділити процес на три етапи:

1. Збір даних через фізичний сканер;

2. Обробка зібраної інформації;

3. Збереження результатів обробки;

4. Використання збереженої біометрії.

Розглянемо ці етапи більш детально. Алгоритм дій що для відбитку пальця, що для знімку обличчя – однаковий, різняться лише методи, які притаманні самим типам біометрії.

2.2.1 Збір даних зі сканеру.

При зніманні відбитку пальцю, дані, та їх вид, залежать від самого типу датчику. Так, при прикладанні пальцю до сканеру, ємкісні датчики будують карту різниць потенціалів на поверхні датчику, отримуючи цифрову карту рельєфу поверхі пальця, де кожне значення вказує на наявність гребня чи впадини. Оптичні датчики захватують зображення відбитку, ультразвувові – надсилають високочастотні звукові хвилі для отримання 3D-карти рельєфу.

Що стосується сканування обличчя, тут простіше – у будь-якому випадку формується зображення обличчя, яке йде далі на обробку.

При отриманні даних, вони відцифровуються, перетворюючись з фізичних даних на електронні сигнали. Зібрана пристроями інформація відправляється на локальний процесор смартфону/ноутбуку (наприклад, окремий модуль безпеки, по типу Secure Enclave у iPhone, чи TrustZone для Android). На цьому етапі дані – «сирі», які просто описують структуру відбитку чи обличчя [33].

2.2.2 Обробка зібраної інформації

Відбувається у кілька основних етапів [34].

- Попередня обробка (Pre-processing). Отримані дані будь-якого типу повинні бути очищені та підігнані до подальшого опрацювання. Тут застосовується фільтрація від шумів, усунення дефектів зображення, збільшення контрастності для кращого виявлення структури, нормалізація для вирівнювання

значень та стандартизації, спрощення аналізу. Наприклад, зображення вирівнюється, нормалізується його розмір, яскравість. Також відбувається сегментація – виділення областей, які мають потенційно корисну інформацію, та відкидання потенційно не маючих сенсу частин.

- Вилучення характеристик (Feature extraction). Після того, як дані отримано й нормалізовано, з них вилучаються характерні особливості. З відбитку пальця за допомогою методу мінуцій вилучаються унікальні точки – закінчення гребнів, їх розгалуження точки біфуркації. Важливий не сам вигляд цих особливостей, а їх положення, кути нахилу, так як вони – унікальні для людини. Але також можуть фіксуватися й інші признаки, по типу відстані мож гребнями, їх орієнтація. Для обличчя ж дані, будь то 2D чи 3D скан, також вилучаються характеристики, по типу відстані між очима, відстані та кута від очей до носа, пропорції обличчя. На сучасному етапі технологій це робиться за допомогою глибоких нейронних мереж, таких як FaceNet, DeepFace.

За вилученими даними формується так званий шаблон. На основі координат мінуцій, їх орієнтації, відстаней, кутів, створюється математичний опис ключових характеристик. Таким чином самі відбитки та скани обличчя не зберігаються на пристрої, зберігаються лише вилучені характеристики. Зібрати з них назад відбиток чи обличчя – неможливо, тому дані користувача у безпеці. Шаблони використовуються при спробах ідентифікувати особу при біометричній автентифікації.

Окрім цього, шаблони шифруються різними криптографічними алгоритмами, які залежать від ОС та ПЗ користувача, виробника апаратного забезпечення, тощо.

2.2.3 Збереження результатів обробки. Утворений шаблон зберігається у захищеній пам’яті пристрою, як вже було згадано – Secure Enclave чи TrustZone. Це запобігає копіюванню чи передачі чутливих даних третім особам, хоча відновити вигляд відбитку чи обличчя з шаблону неможливо, їх характеристика все ще чутлива. Доступ до захищеної пам’яті, так як це зазвичай окремий чип на платі, неможливий навіть з операційної системи. Всі операції з біометрією

виконуються всередині цієї зони, в неї можна лише передати дані, а не отримати.

2.2.4 Використання збереженої біометрії. При спробі автентифікації користувача, виконуються ті ж самі дії, але спрощено – не потрібно сканувати палець та обличчя з різних боків, під кутами, тощо. Скан знімається одразу, одразу подається на ті ж самі алгоритми обробки, утворюючи новий шаблон. Далі, шаблон отриманий при спробі автентифікації, порівнюється зі збереженим. Він передається у ту ж саму захищену зону, там порівнюється, звідки просто повертається результат – особу прийнято, чи відхилено.

Кожен біометричний шаблон являє собою вектор ознак – координати ключових точок для відбитку, та набір відстаней між анатомічними точками для обличчя. Вимірюється відстань між векторами. Найбільш поширені методи порівняння:

- Метод Евклідової відстані – вимірювання по формулі Евкліда, аналогічно до вимірювання відстані між точками простору.

- Косинусна відстань – вимірювання кута між двома векторами ознак, чим менший кут – тим вища вірогідність співпадіння.

- Кореляційні методи – використовується кореляція між наборами даних для вимірювання ступеню співпадіння.

Варто уточнити, що при порівнянні шаблонів, використовується не абсолютне порівняння, а існує певний «поріг співпадіння», 100%-ве співпадіння не потрібне. Порівняння виконується як обрахунок оцінки співпадіння. Зазвичай поріг ставиться від 80% до 95%. Це обумовлено тим, що умови сканування можуть різнитися: відмінності в освітленні, куті, або навіть невеликі зміни у фізіологічних рисах можуть впливати на отриманий шаблон. Саме тому використовується гнучкий поріг співпадіння, який дозволяє допускати незначні відхилення у даних.

Також, при розглядання алгоритмів зіставлення відбитків та обличчя, по окремості, для відбитків використовуються:

- Алгоритм мінуцій – зіставлення ключових точок (мінуцій) на двох зображеннях. Якщо кількість співпадінь мінуцій перевищує порогове значення,

користувач приймається.

- Алгоритми на основі гістограми напрямків – аналізуються напрямки гребенів та борозен, на основі яких будується гістограма. Аналізується схожість гістограм.

Для обличчя:

- Алгоритм Feature Matching – порівнюються вилучені особливості, відстані між точками. При мінімальних відмінностях – обличчя вважається ідентичним.

- Методи на основі нейронних мереж – використання глибоких нейронних мереж як для створення, так і для порівняння шаблонів. Модель навчена розпізнавати обличчя, в процесі навчання вона генерує векторні представлення облич, які потім порівнюються за косинусною відстанню чи іншим метрикам.

Шаблон, який отримується являє собою криптографічний ключ, який може використовуватися як в самому пристрої, так і для інших автентифікацій, навіть у мережі інтернет. Це можна порівняти з електронним підписом, який утворюється з особистих якостей користувача.

Роботу у реальному часі забезпечує оптимізація алгоритмів обробки векторів ознак. Процес автентифікації проходить за кілька мілісекунд, так як обчислення відбуваються на окремому компоненті пристрою.

2.3 Засоби перевірки біометричної інформації для web-ресурсу

З появою стандартів, таких як WebAuthn (частина FIDO2), більше не потрібні окремі додатки для біометричної автентифікації на web-ресурсах. Це дозволяє забезпечувати доступ користувачів до облікових записів без необхідності пам’ятати та вводити паролі, а в додаток до швидкодії – робить автентифікацію ще безпечнішою.

WebAuthn пов’язаний не лише з біометрією, біометрія у ньому – лише складова частина. Він використовує публічні та приватні ключі, підтримує автентифікатори, сканери відбитків та обличчя.

Розглянемо алгоритм дій у деталях:

1. Реєстрація.

Користувач має бажання зареєструватися на web-ресурсі. Використовується пристрій з біометричним сканером. Web-ресурс відправляє запит на пристрій на генерацію пари ключів – публічного та приватного. Біометрія зчитується та обробляється на стороні клієнту (у пристрої користувача), більш того – у тій самій вже згаданій захищеній зоні (Secure Enclave, TrustZone). Приватний ключ зберігається тільки на пристрої, в той час як публічний передається на сервер (звичайно через захищений HTTPS), для подальших перевірок.

2. Автентифікація.

Під час входу, користувач знов використовує біометричний сканер. Створюється новий шаблон, який порівнюється зі збереженим на пристрої. У разі збігу, генерується підпис за допомогою приватного ключа, який передається на сервер для верифікації через збережений публічний ключ.

Переваги використання WebAuthn полягають у тому, що біометричні дані, хай це й не зображення відбитку чи обличчя, а лише зашифрований вектор ознак, ніколи не покидають пристрій користувача. Це робить неможливим їх крадіжку при передачі. Використовувати біометрію елементарно швидше та зручніше, ніж пам’ятати паролі. Також, це захищає від основного методу крадіжок акаунтів – фішингу. Так як автентифікація виконується через пару ключів, неможливо перехопити публічний ключ і просто використати його на іншому сайті, оскільки приватний ключ все ще знаходиться тільки на пристрої користувача.

Для використання WebAuthn потрібно виконати такі кроки:

1. Налаштувати підтримку WebAuthn на стороні серверу. Це робиться через WebAuthn API будь-якою зручною та підтримуваною мовою програмування (наприклад C# чи Node.js).

2. На стороні клієнту використати WebAuthn JavaScript API, який ініціює проходження користувачем біометричної реєстрації/перевірки.

3. Передати дані для верифікації. Отриманий у процесі публічний ключ

передається на сервер, де зберігається для наступних перевірок.

У цих діях, браузер лише ініціює процес проходження біометричної перевірки, але біометричний шаблон нікуди не передається. Всі розрахунки відбуваються в самому пристрої, відповіддю є лише публічний ключ.

Доступу до приватного ключа немає навіть у системи. Саме він використовується для створення цифрового підпису, який посвідчує користувача. В аутентифікації WebAuthn використовується механізм «виклик-відповідь» (challenge-response). Сервер відправляє унікальний виклик (challenge), який підписується з використанням приватного ключа, потім пристрій відправляє підписані дані назад (response). Цей механізм запобігає атакам відтворення, так як кожний виклик унікальний для кожного сеансу. Окрім цього, публічні ключи прив’язуються до домену, тому використання їх на інших web-ресурсах неможливе, навіть якщо зловмисник якось їх перехопить.

2.4 Висновки до розділу 2

У другому розділі розглянуто основні механізми біометричної автентифікації, зокрема фізичні пристрої, що використовуються для зчитування біометричних даних. Визначено, що дактилоскопічні датчики, зокрема ємкісні, оптичні, ультразвукові та теплові, мають свої особливості, переваги та недоліки, зокрема в плані безпеки та точності. Ультразвукові датчики виявляються найбільш надійними, проте їх вартість обмежує їх використання в масових пристроях.

Досліджено також пристрої для сканування обличчя, які, хоча й легкодоступні завдяки вбудованим фронтальним камерам, мають обмеження, пов’язані з якістю зображення в умовах недостатнього освітлення. Спеціалізовані інфрачервоні сканери пропонують покращену якість зчитування, однак їх доступність обмежена преміальними моделями.

Особливу увагу приділено універсальності та доступності обох методів: смартфони з вбудованими сканерами відбитків пальців та фронтальними

камерами стають основним засобом для біометричної автентифікації, в той час як ПК та ноутбуки вимагають додаткових витрат на периферійні пристрої.

Також описано принципи роботи біометричного входу, які складаються з чотирьох основних етапів: збору даних, їх обробки, збереження та використання. Це надає змогу зрозуміти, як біометричні дані обробляються та зберігаються, а також чому такі системи вважаються безпечними.

Загалом, розділ демонструє, що біометрична автентифікація є зручним, надійним та швидким способом підтвердження особи, з можливістю подальшого розвитку та вдосконалення технологій для забезпечення більшої безпеки в цифровому середовищі.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ МЕТОДУ БІОМЕТРИЧНОЇ АВТЕНТИФІКАЦІЇ

3.1 Переваги використання WebAuthn

Повернемось до питання засобів біометричної автентифікації та шляхів доступу до них. Серед основної та доступної біометрії було виділено відбиток пальця та сканування обличчя. Розглянемо доступність впровадження цих методів у WEB.

Перш за все, так як за біометрію відповідає окремий пристрій, прямого доступу до неї немає ні у користувача, ні у самої операційної системи. Тому, для використання відбитку пальцю, потрібно робити запит до операційної системи пристрою, а от камера є пристроєм загального користування. Для спрощення, розглянемо окремо (табл. 3.1).

Таблиця 3.1

Порівняння вбудованих пристроїв біометрії

	Шлях доступу	**Примітка**
Сканер відбитку	Захищені, через запити з ОС	Використовують вбудовані біометричні алгоритми та збереження у ізольованій пам’яті
Сканер обличчя (апаратний)		
Камера	Прямий, як до звичайного периферійного пристрою	Можна використати лише для сканування обличчя, потребує реалізації алгоритмів вилучення ознак

Виходячи з порівняння, звісно можна використати окремо камеру пристрою, так як вона наявна як у мобільному телефоні, так і вбудована у ноутбуки та часто є у власників ПК, але це також грає роль обмеження й складності реалізації. Необхідність відтворити алгоритми вилучення ознак з обличчя, їх порівняння, дотримання стандартів щодо незбереження зліпку обличчя та неможливості його початкового відтворення. Також це обмеження: використовувати тільки обличчя, через камеру, освітлювати його у темний час доби чи у темному приміщенні.

Як вже вказано, системи які початково призначені для біометрії – захищені, ізольовані, до них немає доступу навіть у ОС, але ОС може з ними взаємодіяти: направляти запити та отримувати відповіді. Саме на цій особливості й побудовано стандарт WebAuthn, який користується самою системою для використання біометрії пристрою. Окрім цього, будучи частиною FIDO2, цей стандарт у своєму API реалізує не тільки один тип доступу, а суміщає їх й надає користувачеві вибір.

Справа у тому, що основною задумкою WebAuthn є відмова від паролів та спрощення системи автентифікації для користувача, проте й її зміцнення в плані безпеки одночасно. Для доступу до web-ресурсу замість паролю використовується не сама біометрія (вона захищена та ізольована), а пристрій користувача. Повертаючись до розділу 1, мобільний пристрій чи ноутбук, як будь-який пристрій у якому є ізольований біометричний компонент, вже сам по собі є автентифікатором, який може бути використаний в якості ключа. Тому, мобільний телефон чи ноутбук використовують для доступу до web-ресурсу, а ось дія підтверджується вже зручним для користувача способом – в тому числі й біометрією.

Основні переваги:

1. Стандарт використовує вбудовані біометричні алгоритми в пристрої. Це звільняє розробника від необхідності глибокого розуміння машинного навчання та комп’ютерного зору.

2. WebAuthn вже інтегровано з безпечними рішеннями на рівні ОС. При

самостійній розробці алгоритму перетворення обличчя/відбитку пальця у ключ, це завжди буде менщ безпечно, так як дані можна буде зберегти лише у спільній пам’яті, а тут вони – ізольовані навіть від системи на апаратному рівні.

3. Так як WebAuthn є стандартом, це спрощує інтеграцію з різними системами та пристроями, в тому числі кроссплатформеність. Самостійний алгоритм міг би зіткнутися з відсутністю підтримки, сумісності на різних пристроях.

Окрім цього, WebAuthn має підтримку основних біометричних систем, але й турбується про їх відсутність чи пошкодження: при використання телефону, якщо користувач не може сканувати палець чи обличчя, він може просто ввести PIN-код від пристрою, таким чином його підтверджуючи. Стандарт не потребує окремих додатків чи доповнень, та спокійно працює на рівні браузеру, якщо браузер підтримує цю технологію. Найпоширеніші браузери Google Chrome та Opera, що працюють на рушії Chromium, мають підтримку, й проблем з використанням WebAuthn у них немає, як на Desktop-просторі так і у мобільному.

Таким чином, розробка методу автентифікації виконано з використанням стандарту WebAuthn через всю сукупність його переваг та можливість підтвердження користувача довіреного пристрою через відбиток пальцю. Як додатковий метод автентифікації використано сканування обличчя користувача.

3.2 Використані програмні засоби

Для дотримання сучасного підходу до виконання проекту, обирається відповідна мова програмування під виконувану задачу. Даний проект є лише частиною, яка відповідає за автентифікацію та авторизацію, але як і будь-який web-ресурс він поділяється на front-end та back-end частини.

Виконання front- та back- частин по окремості вже стало нормою, задля розподілення логіки та відповідальності, а також використання більш оптимальних засобів для їх реалізації. Тому, клієнтська частина (front-end)

виконується на JavaScript, а саме – Next.js(React). А щодо серверної частини (back-end), можна використати будь-яку зручну та достатню для виконання задачі мову, серед яких – Node.js, C#, Java, Python. Принципової різниці у цьому немає, для кожної з цих мов є своя FIDO2 бібліотека, яка забезпечує підтримку WebAuthn [35]. Але варто відмітити:

1. Node.js – дозволить повністю написати проект на JavaScript, як клієнтську так і серверну частини, та не розділяти його логіку. Проте, для серверної частини він має недоліки в плані продуктивності та оптимізації. Він підходить для асинхронних задач, але при виконанні складних обчислень чи роботи з великою кількістю одночасних запитів, можуть виникати проблеми через модель подій, засновану на одному потоці. Це вимагає додаткових зусиль з оптимізації.

2. Python – має бібліотеки FIDO2, та й взагалі живе за рахунок бібліотек, які написані іншими мовами. Являє собою інтерпретатор, а отже – повільніший. Дуже поширена мова через низький поріг входження та зрозумілість синтаксису, але найкраще підходить для невеликих web-проектів, дата-аналітики та задач штучного інтелекту.

3. Java/C# – мови дуже близькі одна до одної, повністю об’єктно-орієнтовані, але кожна зі своїми нюансами. Використовують механізм компіляції JIT (Just-In-Time), що забезпечує гарну продуктивність. Проте, в той час як C# повністю є компілятором, Java використовує свою віртуальну машину й все ж інтерпретує код, в той час як система JITC (Just In Time Compilation) дозволяє визначати багаторазове використання коду, й компілювати програму частково лише для повторного використання.

Ще раз варто наголосити, що принципової різниці у виборі мови немає, так як бібліотеки FIDO2 існують для всіх цих мов (рис. 3.1), дозволяючи вбудувати систему авторизації у будь-який проект. Тому, для серверної частини обрано C# ASP.NET, враховуючи його Entity Framework, суворість типізації й гарну інтеграцію у Visual Studio, а також швидкодію, так як це – повноцінний компілятор.

Список використаних програмних засобів для проекту:

– Front-end:

Next.js (React.js), IDE Visual Studio Code.

– Back-end:

C# ASP.NET Web-API, Entity Framework Core (MS SQL Server), Fido2NetLib, IDE Visual Studio 2022.

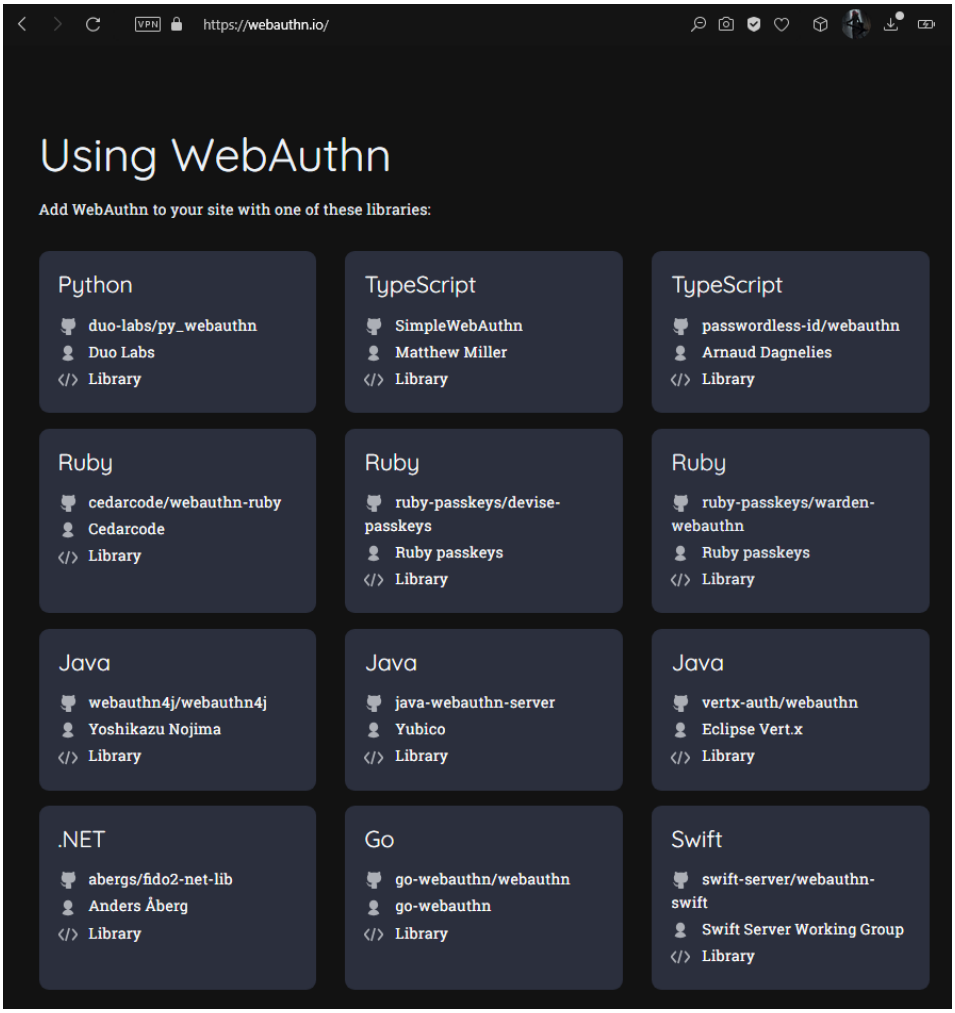


Рис. 3.1. Підтримка WebAuthn мовами програмування

Окрім цього, використовується:

- OpenSSL, mkcert – утиліти для створення та конвертації HTTPS сертифікатів [36-37].

- NGROK [38] – утиліта для тунелювання localhost у мережу Інтернет.

3.3 Створення серверної частини web-додатку (ASP.NET)

Серверна частина web-додатку створюється за шаблоном ASP.NET Web-API. Вона слугує для реакції на запити клієнту, та обробки й збереження даних. Діаграма класів ASP.NET проекту наведена на рис. 3.2.

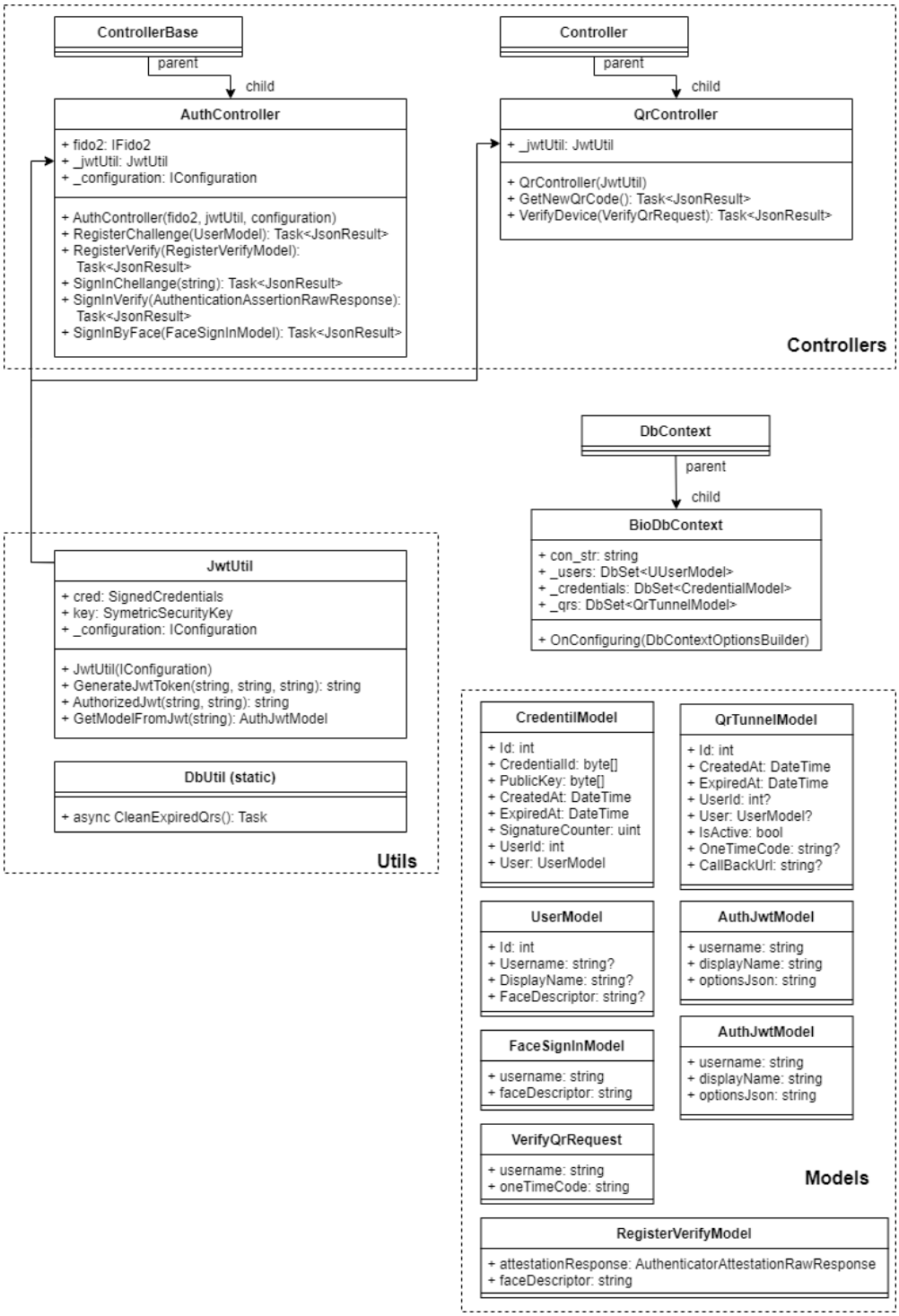


Рис. 3.2. Діаграма класів back-end`у

3.3.1 База даних серверу

Перш за все, для управління даними та доступу до них, потрібна база даних. У якості такої використовується Microsoft SQL Server, у вигляді локального файлу (local-db) – зменшена версія ядра БД, яка може оперуватися відразу у середовищі розробки, без додаткових інструментів. В якості серверу для неї використовується персональний ноутбук, на якому ведеться розробка та налагодження. Ця СУБД обрана через її повну інтеграцію з платформою .NET та IDE Visual Studio, а також легкість взаємодії через Entity Framework [39].

Після створення порожньої БД, використовується підхід Code-First, при якому таблиці описуються спочатку кодом класів моделей даних, а потім з них через процес міграції утворюються таблиці у самій БД засобами EF. Тому, важливі сутності (Entities), створюються у програмі у вигляді класів у окремій директорії (/Models). Схему створених класів зображено на рис. 3.3.

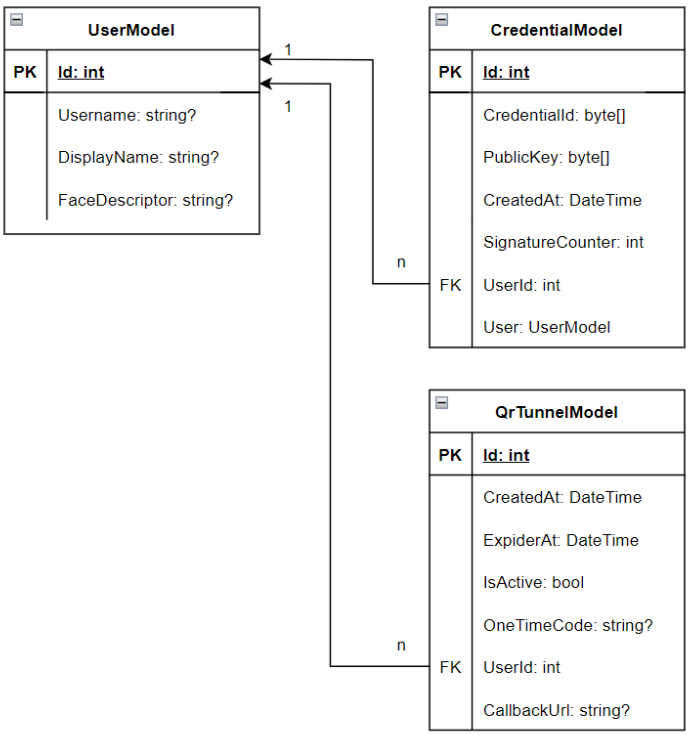


Рис. 3.3. ER-діаграма моделей даних

Клас UserModel має власний ідентифікатор, дві строки імені користувача та строку дескриптору обличчя.

Username використовується для ідентифікації самого користувача при його вході на web-ресурс, Це поле одночасно являє собою альтернативу паролю, так як воно не обов'язково має збігатися з DisplayName, та навіть більше – має відрізнятися від нього, щоб забезпечувати додатковий рівень захисту, адже не знаючи Username, навіть маючи зареєстрований для входу пристрій, увійти не вийде.

DisplayName – це ім'я користувача, яке використовується для відображення всередині web-ресурсу. ім'я профілю, акаунту, тощо, при написанні коментарів, постингу, листування, та тому подібних дій.

FaceDescriptor – це строка з параметрами обличчя користувача, яка зберігає вектор ознак, для можливості входу через сканування обличчя. Це окреме поле, а не таблиця, так як обличчя користувача може існувати лише в одному екземплярі, а можливості додавання кількох облич не передбачено.

Другий клас, одночасно й таблиця даних, це CredentialModel. Він описує Credentials (з англ. «облікові дані», далі – креденшл). Креденшл має такі унікальні поля, як CredentialId та PublicKey. CredentialId – це поле для особистого унікального ідентифікатору креденшл, на відміну від звичайного Id, який є ідентифікатором запису у базі даних. PublicKey – це публічний ключ, який використовується у процесі верифікації користувача й відкрито передається через мережу без ризиків, він прив'язаний свій до кожного креденшелу. Також, сутність містить посилання на модель користувача, до якої вона прив'язана.

Також записано QrTunnelModel – клас, який відповідає за зберігання одноразових токенів доступу до облікового запису шляхом сканування QR-коду. Ці дані перетворюються на QR-код на front-end частині. Має дату створення та дату вичерпання, статус активності, сам одноразовий код, прив'язку до користувача, та інформацію, куди має бути надіслана інформація при виклику-відповіді.

Описаний на діаграмі зв'язок «багато до одного» говорить, що один

користувач може мати більш ніж один креденшл, та на нього може генеруватися більш ніж один QR-код для входу, що пояснюється тим, що для одного й того самого акаунту може існувати кілька пристроїв, який надано доступ, й кожен такий пристрій-автентифікатор буде мати всій власний креденшл в якості запису у БД, тому й QR-кодів може бути кілька, так як вони генеруються та оновлюються на сторінках. Без таблиці з QR-кодами, потрібно було б кожного разу редагувати поле у записі користувача, що не є оптимальним.

Після створення класів, налаштовано BioDBContext для EF, у якому записано строку підключення до файлу LocalDB та записані датасети DbSet<UserModel> _users, DbSet<CredentilModel> _credentials, DbSet<QrTunnelModel> _qrs. Після цього за допомогою терміналу NuGet виконано команди міграції:

Add-Migration InitialCreate;

Update-Database;

Це створює у папці проекту директорую Migrations з автоматично згенерованими класами, що описують процес створення таблиць у БД, а самі таблиці додано у базу.

3.3.2 Моделі даних для API

Для правильної роботи клієнт-серверної логіки програми, кожен API endpoint повинен приймати свою чітко визначену структуру даних. Це важливо, так як не отримавши точної структури, front-end звернеться з запитом до серверу, й не отримає відповіді, так як метод контролеру для переданої структури буде не визначено. Спеціальні моделі вводяться для специфічних та складних типів даних, які не є вже вбудованими.

Створені моделі даних для API наведено в табл. 3.2.

Таблиця 3.2

Моделі даних для API

Клас	**Дані**	**Призначення**
AuthJwtModel	username : string displayName : string optionsJson : string	Для верифікації користувача за допомогою WebAuthn
FaceSignInModel	username : string faceDescriptor : string smileDescriptor : string gloomyDescriptor : string	Для входу користувача за обличчям
RegisterVerifyModel	attestationResponse : AuthenticatioAttestationRawResponse faceDescriptor : string smileDescriptor : string gloomyDescriptor : string	Для верифікації реєстрації користувача
VerifyQrRequest	username : string oneTimeCode : string	Для верифікації доступу користувача через одноразовий QR-код

Дані моделі використовуються як структури даних, які приймаються методами контролерів при POST-запитах до back-end`у. Більшість полів структур даних представлена типом string, що пояснюється тим, що основна мова запитів та відповідей – текстовий формат JSON, який дозволяє передавати текстом складні структури даних із багатьох рівнів вкладень, масиви, тощо.

Докладніше про використання цих класів – у п. 3.3.4.

3.3.3 Допоміжні класи (Utils)

Для даного проекту, створено два допоміжних класи, які полегшують роботу та зменшують кількість дублювання коду. Ці класи – DbUtil та JwtUtil, які використовуються для роботи з базою даних та JWT, відповідно.

На даний момент, DbUtil є статичним класом з єдиним статичним методом async Task CleanExpiredQrs(), який. як видно з назви, видаляє з БД QR-коди, які вичерпали свій термін життя. Це автоматизує роботу з БД, достатньо викликати цей метод у оптимальних місцях, й зайві дані з БД будуть видалені.

Клас JwtUtil не є статичним, але його екземпляр існує у програмі як Singleton, й використовує DI для впровадження. Так як клас використовується для створення та використання JWT-токенів, однією з його властивостей є ключ шифрування токену, який зберігається у конфігурації серверної частини web-додатку, що й забезпечує захист JWT. Методи цього класу:

– GenerateJwtToken(username, displayName, optionsJson) – повертає JWT-токен, в якому зберігаються імена користувача й параметри (використовується в процесі реєстрації для запам’ятовування імен користувача та WebAuthn challenge);

– AuthorizedJwt(username, displayName) – повертає JWT з іменами користувача, характеризує авторизованого користувача;

– GetModelFromJwt(token) – приймає готовий JWT. розшифровує з нього username, displayName, optionsJson та повертає результат у вигляді екземпляру AuthJwtModel. Використовується під час верифікації при реєстрації та вході за допомогою WebAuthn.

3.3.4 Створення API контролерів

Контролери API поділяються на сфери діяльності та відповідальності, групуючи методи, які належать одній задачі чи меті. Вони обробляють запити та слугують endpoint`ами back-end`у. Так як проект зосереджено на авторизації та

автентифікації користувача, створено основний AuthController, який відповідає за процеси реєстрації та авторизації користувачів, та додатковий QrController, який займається виключно QR-кодами.

Структури endpoint`ів та запитів до back-end можна переглянути на рис. 3.4.

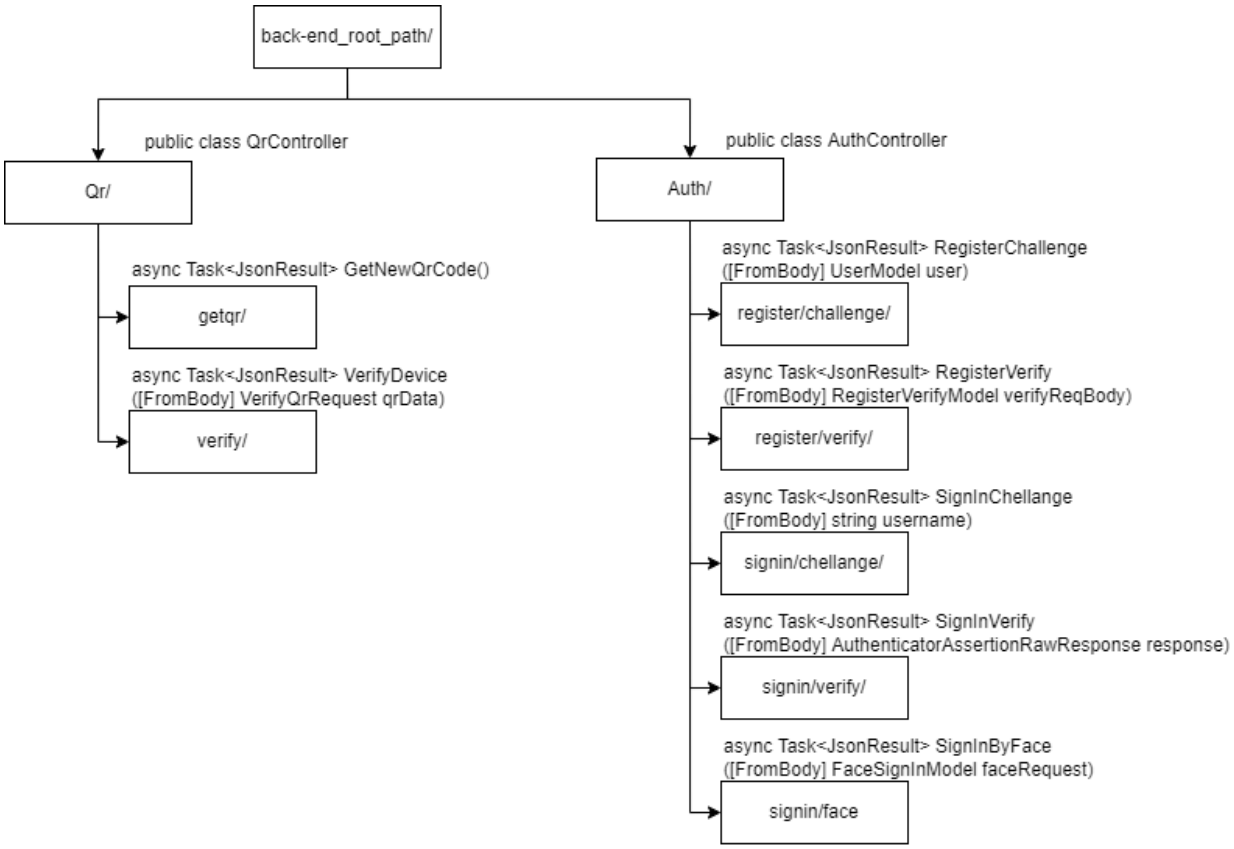


Рис. 3.4. Мапа back-end API endpoints

Розглянемо детальніше всі ці методи.

– Qr/getqr

Генерує новий QR-код. При отриманні запиту на генерацію нового QR-коду, викликається функція очищення застарілих QR, після чого генерується унікальний одноразовий код. Далі, він використовується для створення об’єкту QrTunnelModel, який додається до БД у таблицю _qrs та повертається з відповіддю на запит.

– Qr/verify

Отримує username та одноразовий код у запиті, знаходить запис про нього у БД. Перевіряє, чи не вичерпано термін дії. Якщо код активний, видаляє його запис з БД, формує JWT-токен авторизованого користувача й повертає його у відповіді.

– Auth/register/challenge

Відповідає за генерацію WebAuthn виклику (challenge) та повертає його на клієнт. Формує об’єкт Fido2User, який використовується для створення унікального виклику при запиті на створення нового креденшела. Додатково, формує JWT-токен, який містить згенерований виклик, та обидва username користувача (для верифікації).

– Auth/register/verify

Виконує верифікацію користувача при реєстрації. Викликається після проходження виклику (підтвердження пристрою) на клієнті. Приймає RegisterVerifyModel, в якому міститься відповідь на виклик та три дескриптори обличчя користувача (вектор ключових точок) – для звичайного обличчя, усмішки та похмурості (підвищення надійності та захист від підробки). Окрім цього, з заголовку запиту вилучається JWT, згенерований на етапі Auth/register/challenge, з якого розшифровується username, displayName, optionsJson з початковим викликом. Перевіряється, чи результат проходження виклику дійсно виконано на той виклик, який було згенеровано. Якщо все правильно, то перевіряється наявність обраних username у БД, якщо імена не зайняті – формуються записи про користувача, запис нового креденшела використаного пристрою, а користувачеві повертається авторизований JWT, який дозволяє доступ до нового облікового запису.

Даний алгоритм верифікації один з найдовших у коді, тому його варто показати схемою алгоритму (рис. 3.5).

– Auth/signin/chellange

Генерує виклик для входу користувача в наявний обліковий запис. Працює по аналогії з реєстрацією, але приймає тільки username. Знаходить користувача за ним у БД, а також його збережені креденшели. На їх основі формує новий

унікальний виклик, який повертається на клієнт. Так само зберігає username`и та створений виклик у JWT.

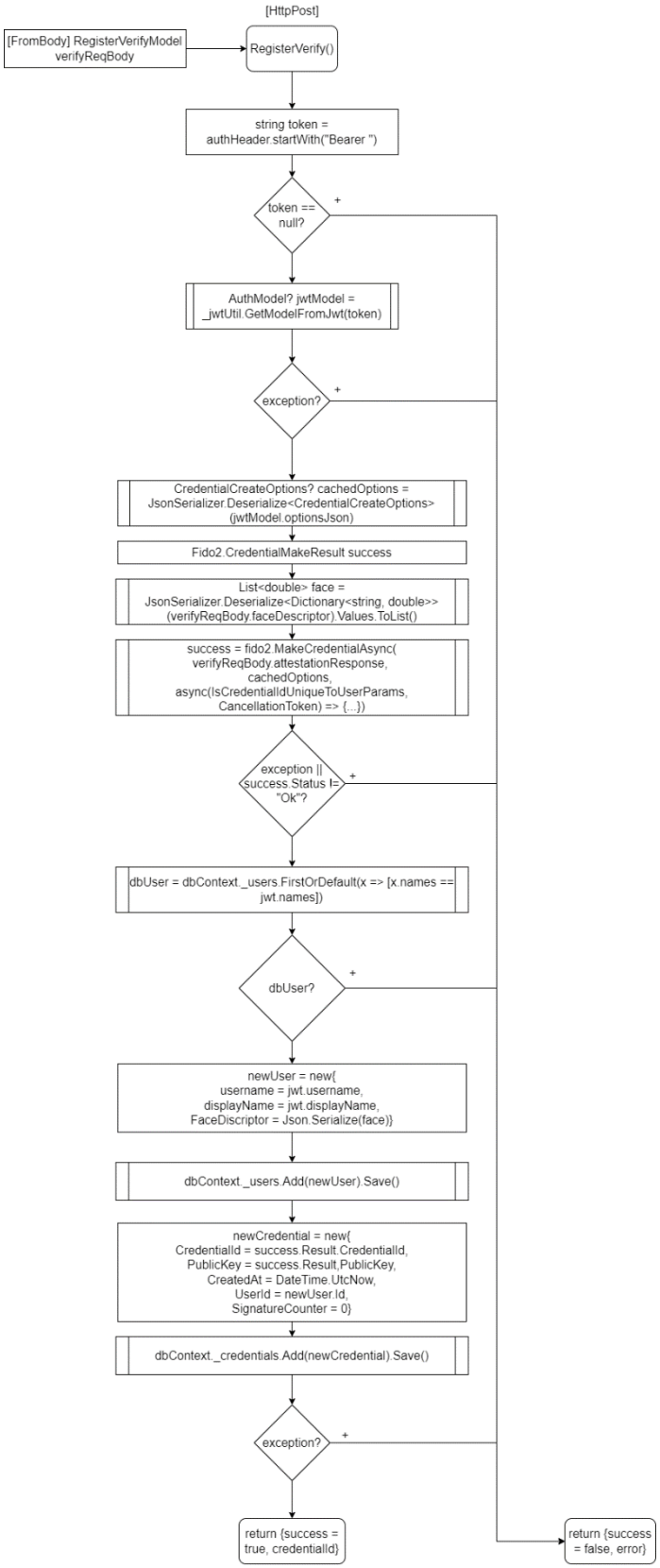


Рис. 3.5. Схема алгоритму RegisterVerify()

– Auth/signin/verify

Перевіряє відповідь клієнта на виклик авторизації. Працює за аналогією з верифікацією реєстрації, відмінність у тому, що при успіху – інкрементується лічильник креденшелу SignatureCounter. Це дозволяє уникати атак методом відтворення, та генерувати кожного разу унікальні виклики на основі наявних креденшелів. При успіху, генерує JWT авторизованого користувача, який дозволяє доступ.

– Auth/signin/face

Приймає FaceSignInModel, в якому міститься username та три дескриптора обличчя. Знаходить у БД запис користувача та збережені дескриптори, після чого рахує відстань між векторами ознак методом Евклідової відстані. Якщо показник Евклідової відстані менший за 0.3 для кожного дескриптору, обличчя приймається. Граничне значення відстані, при якому обличчя вважається прийнятим, знайдено експериментально, та є таким, яке потребує від користувача відтворення умов, при яких проводилася реєстрація. При успіху, генерується JWT з дозволом доступу до акаунту.

Таким чином, маємо набір методів API, які дозволяють проходити автентифікацію без участі паролю, номеру телефону чи електронної адреси, та орієнтовані на використання біометричних ознак обличчя користувача, та використання довіреного пристрою, який перевіряє відбиток пальцю чи обличчя за допомогою WebAuthn, чи PIN, якщо доступу до сканерів немає. Для того, щоб передавати доступ від пристрою на пристрій, використовуються одноразові QR-коди, які повинні скануватися авторизованим пристроєм.

Підхід з використанням WebAuthn дозволяє використовувати вбудовані в пристрій криптографічні алгоритми, й користуватися біометрією пристрою для його підтвердження. Окрім цього, це – можливість користуватися спеціальними апаратними ключами FIDO2 з вбудованими сканерами відбитку пальця, що є універсальним та безпечним. Зрештою, довіряти безпеку облікового запису криптографічним алгоритмам пристрою та збереженню даних у окремому

захищеному чипі, чи у апаратному ключі з біометричним підтвердженням – надійніше, ніж користуватися номером телефону. Інші пристрої та ключи можна підтвердити за допомогою QR-кодів, а на випадок втрати всього – користувач може просто відсканувати обличчя.

3.3.5 Генерація та застосування HTTPS сертифікатів

Безпечний та захищений web-ресурс неможливо уявити без використання HTTPS-протоколу (HTTP-Secured). На відміну від звичайного HTTP, на якому він заснований, він має основну перевагу у вигляді шифрування даних. Наразі, звичайний HTTP майже не використовується, й весь сучасний інтернет побудовано на HTTPS.

Зазвичай, розробка програмних web-продуктів ведеться на HTTP, що дуже її спрощує, а сам HTTPS додають вже на релізі. Проте в випадку цього проекту, сам процес розробки вимагає використання HTTPS, так як по звичайному каналу HTTP не працює WebAuthn (виводиться помилка, що запит здійснено без дійсного сертифікату безпеки).

Сертифікати отримуються від так званих ЦС (Центи Сертифікації), але це досить проблематично для програмного продукту на етапі розробки поза компанією, та вимагає зайвого часу. Тому, щоб все ж таки забезпечити себе сертифікатом, розробники використовують спеціальні утиліти, які можуть генерувати самопідписані (самописні) сертифікати.

Сама IDE Visual Studio для проекту ASP.NET вже генерує власний самописний сертифікат, але з ним також є проблема. Так як у проекті використовуються крос-доменні запити, це викликає конфлікт. Тому, для реалізації HTTPS потрібно скористатися окремим зовнішнім інструментом.

Серед таких інструментів обрано OpenSSL та mkcert. За допомогою mkcert згенеровано сертифікати, за допомогою bash-команди:

mkcert localhost 192.168.0.100 192.168.0.101 192.168.0.102 192.168.0.103 192.168.0.104 192.168.0.105 192.168.0.106 192.168.0.107 192.168.0.108

192.168.0.109 192.168.0.110

Результатом є HTTPS сертифікат для localhost, дійсний протягом 365 днів. Він справний та довірений для вказаного набору IP-адрес, так як адреси ПК в процесі розробки можуть змінюватися у локальній мережі.

OpenSSL використовуються для конвертування сертифікату у формат RSA, підтримуваний ASP.NET, так як Next.js підтримує сертифікат у початковому форматі, а ASP.NET – ні.

Для конвертування використовується спеціальна команда:

openssl rsa -in localhost-key.pem -out localhost-key-rsa.pem

Після чого, отримані сертифікати дозволяють запускати та використовувати HTTPS як на back-end, так і на front-end частинах проекту.

3.3.6 Налаштування CORS

При створенні крос-доменних web-ресурсів, та розділення web-додатку на окремий front-end-проект та back-end-проект, варто приділяти увагу самому питанню доступу до частин такого ресурсу. Щоб гарантувати, що потенційні зловмисники не зможуть маніпулювати з API напряму, використовується механізм CORS, який це обмежує [40].

CORS забороняє вільний доступ до API, та блокує всі запити, крім списку дозволених. Дозвіл надається на джерело/джерела (Origins). На початку програми, у Program.cs налаштовуються ці політики. Політики CORS налаштовуються окремо для back-end та для FIDO2.

Список відомих джерел задається у програмі:

```
string localhost_domen = "https://localhost:3000";
string localhost_back = "https://localhost:5001";
string ngrok_domen = "https://xxxx-xxx-x-xx-xxx.ngrok-free.app";
```

Для заборони доступу до API з невідомих джерел, задається:

```
builder.Services.AddCors(options =>
        {
            options.AddPolicy("AllowFrontend",
```

```
        builder =>
        {

            builder.WithOrigins(localhost_domen, ngrok_domen, localhost_back)
                .AllowAnyHeader()
                .AllowAnyMethod()
                .AllowCredentials();
        });
});
```

Задано, що від даних відомих джерел, дозволяється використання будь-якого заголовку запиту, методу, та дозволено використання креденшел (в тому числі, JWT та Cookies).

Для FIDO2 задається:

```
builder.Services.AddFido2(options =>
    {
        options.ServerDomain = ngrok_domen.Replace("https://", "");
        options.ServerName = "Passwordless ASP.NET API";
        options.Origins = new(){ localhost_domen, ngrok_domen };
        options.TimestampDriftTolerance = 300000;
    });
```

Задано основний домен, з якого здійснюються запити для використання WebAuthn, та дозволені джерела.

Всі ці дані повинні бути змінені при використанні в реальному проекті, та ті, які відповідають реальним доменам та URL на реальних серверах.

Таким чином, серверна частина web-ресурсу захищається від несанкціонованих запитів, які здійснюють не з дозволеного front-end`у.

3.4 Клієнтська частина web-додатку (Next.js)

Фактично, front-end відповідає за те, що бачить користувач при взаємодії з web-ресурсом. Але окрім цього, front-end також надсилає запити до сервера, а ще може виконувати обчислення, обробку даних, тощо, щоб розвантажити сервер, та розподілити навантаження між сервером, та клієнтом.

Під час розподілення логіки, та покладаючи окремі задачі на клієнт, в контексті кібербезпеки потрібно турбуватися про можливість використання цього як вразливостей. Так як весь код, що виконується на клієнті, відкритий та

може бути переглянутий через панель інструменті розробника прямо в браузері, дані можуть бути підмінені, змінені алгоритми, надіслані неправильні запити, та ще багато чого іншого.

В даному проекті, клієнт займається лише збором даних, відправкою запитів та прийомом відповідей. Шифрування та подібні процеси на клієнт не виносяться, а підробка результатів проходження challenge чи зміна дескриптора обличчя нічого не дає. Тому, можна вважати, що клієнт написаний правильно з точки зору кібербезпеки.

Ще, варто зазначити, що даний клієнт – скоріше демонстрація роботи створеного ASP.NET API, ніж повноцінний проект. Проект задумано як складову частину, а не самостійну повноцінну одиницю.

Окрім цього, на клієнті реалізовано ще один механізм web-захисту, так як запити йдуть не напряму на back-end, а проксуються (детальніше про це у п.3.4.2).

3.4.1 Web-сторінки

Для клієнту розроблено web-сторінки, які дозволяють у повній мірі продемонструвати використання всіх методів автентифікації, в тому числі й основний – біометрію.

Для обраного Next.js директорії та файли зі скриптами також являють собою мережеві шляхи (URL) на самому web-ресурсі. Схему всіх розроблених сторінок можна бачити на рис. 3.6. Для спрощення тестування та розуміння всіх процесів, на всі сторінки додано поле для логування дій, в якому покроково відображається, що й як відбувається, виводяться помилки, та повідомлення про успіх.

1. Основною самостійною сторінкою є «register/». На цій сторінці відбувається повний процес реєстрації користувача. Інформація, яка потрібна щоб зареєструватися: username, displayName, FIDO2 credential, та дескриптор обличчя. Таким чином усувається недолік традиційних методів автентифікації –

користувачеві, замість запам’ятовувати зв’язку з імені та паролю, необхідно вказати два імені – те, що використовується для входу (username), та те, яке відображуватиметься в додатку як його ім’я (displayName). При цьому, запам’ятати потрібно лише username, а displayName можна не тримати в голові – його просто буде збережено у БД й використано у майбутньому.

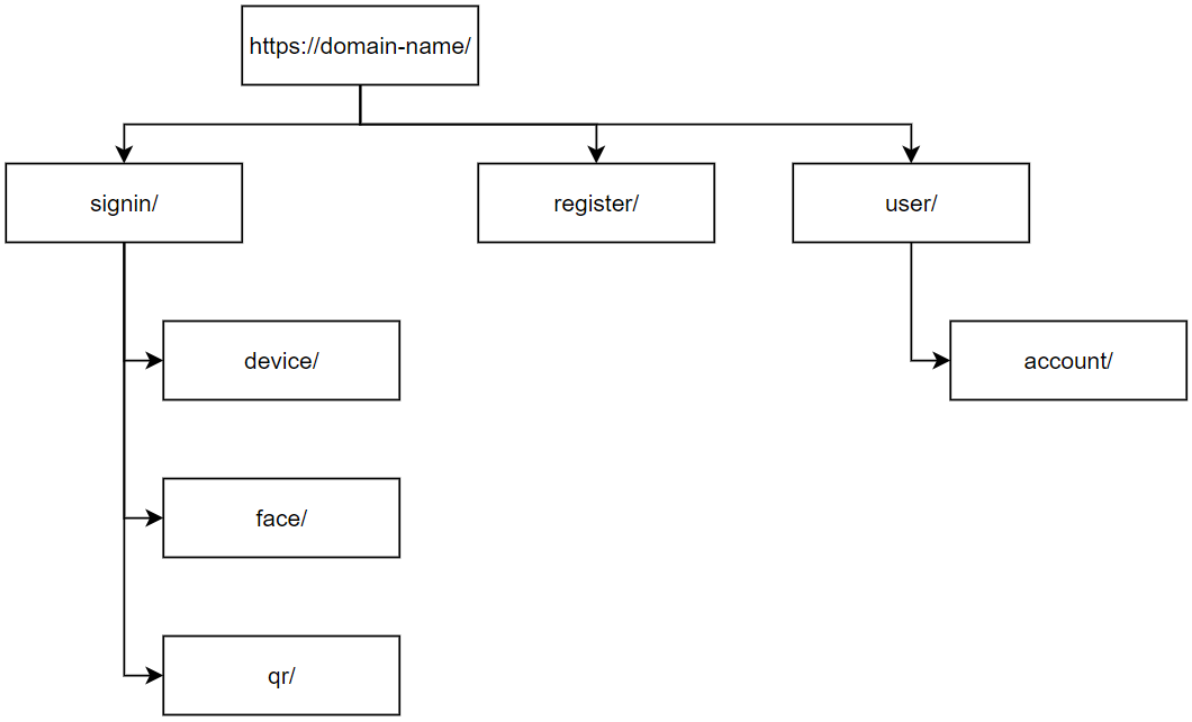


Рис. 3.6. Схема URL web-ресурсу

Решта даних, що потрібні для реєстрації – FIDO2 credential та дескриптор обличчя, теж не потрібно пам’ятати. Маючи доступ до пристрою, з яким проводилась реєстрація – користувач має доступ до облікового запису. Це справедливо, доки існує хочу б один авторизований пристрій. Запобіжним заходом проти втрати доступу, та для відновлення, виступає обличчя користувача, яке також не може бути втрачене (пластичні операції, які змінюють ключові точки на обличчі, до уваги не беруться).

На цій сторінці, з боку клієнту, перевіряється порожність полів, а кнопка реєстрації недоступна до виконання послідовного сканування трьох станів обличчя користувача. Спрощено, алгоритм всіх дій при реєстрації подано на рис.

3.7.

Опишемо це більш детально. Наразі, не всі браузери підтримують WebAuthn, але це не є проблемою, так як основні та найрозповсюдженіші браузери масового користування мають цю підтримку. Справа в тому, що виклик на підтвердження пристрою у систему надсилає саме браузер, тому це й є важливим – щоб ця підтримка була. У Google Chrome та Opera є підтримка WebAuthn, тому перейматися не варто – реєстрація може бути пройдена у них.

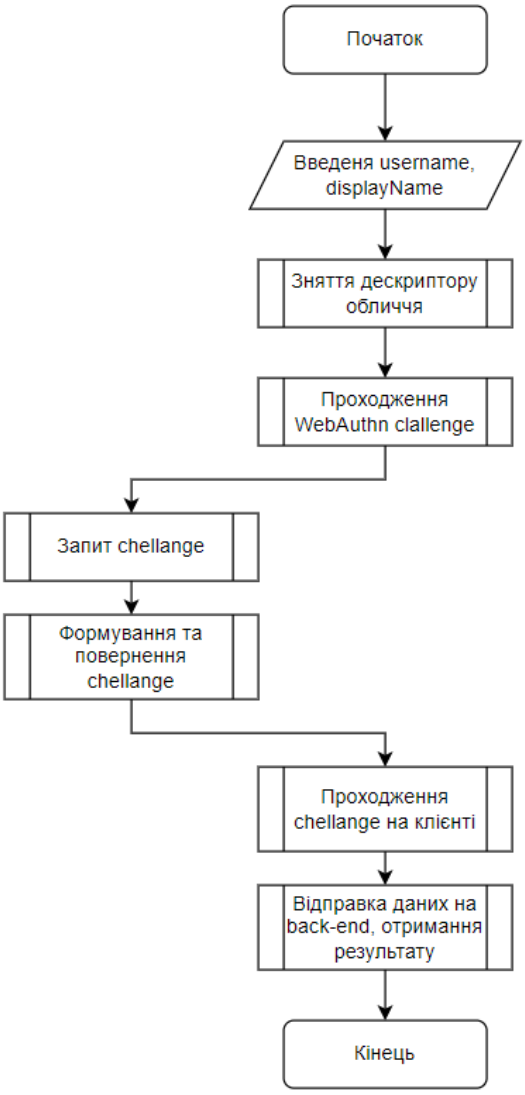


Рис. 3.7. Схема алгоритму реєстрації

На всякий випадок, на клієнті поставлено перевірку підтримки – тестовий виклик *window.PublicKeyCredential*. Далі, виконується запит до серверу на

отримання нового виклику для пристою користувача. Разом з викликом, клієнт отримує JWT-токен, в якому містяться вказані ним імена, а також початково утворений виклик для процесу верифікації. Цей JWT зберігається у пам’яті браузеру, використовуючи *localStorage*.

Далі, ініціюється проходження виклику клієнтом – за це вже відповідає пристрій. Після проходження, результат надсилається в тілі запиту на сервер, разом з JWT у заголовку, для валідації. Сервер перевіряє всі дані, й в разі успіху – зберігає у БД запис користувача: імена, дескриптор обличчя та креденшл використаного пристою.

Для зняття дескриптору обличчя використана бібліотека *faceapi*, яка працює за допомогою претренованих моделей штучного інтелекту (ШІ). В даній бібліотеці існують моделі для різних задач, проте для задачі зняття дескриптору використані моделі:

– *ssd_mobilenetv1_model* – для знаходження обличчя на зображенні.

– *face_landmark_68_model* – для знаходження ключових точок на знайденому обличчі.

– *face_recognition_model* – для вилучення вектору ознак (дескриптора) зі знайдених ключових точок.

Моделі завантажені за інструкцією від розробника *faceapi*, й навантаження від їх використання також припадає на клієнт. Це зроблено з кількох причин. Основна причина звісно ж у безпеці, а потім – у швидкодії, тому що при вилученні ознак на стороні серверу, на сервер потрібно було б передати зображення обличчя користувача, для подальших маніпуляцій, а його може бути перехоплено й використано не за призначенням. Якщо ж вилученням дескриптору займається клієнт, мінус тільки в тому що клієнт має завантажити та запустити моделі на своєму пристрої, проте на сервер вже передається готовий дескриптор – «сирий» (raw-) масив даних. З цього масиву, все ще неможливо отримати початкове обличчя, не те що початкову фотографію. А завантаження моделей відбувається лише при першому відвідуванні сторінки, надалі вони зберігаються к кеші браузеру й використовуються повторно. Окрім цього, моделі

доволі легкі: загальний обсяг файлів використаних моделей – 11.8 Мб пам’яті, що можна порівняти зі звичайною web-сторінкою, наповненою медіа-контентом.

2. Основною сторінкою входу є «signin/device» – саме вона використовує FIDO2 WebAuthn для автентифікації користувача шляхом перевірки пристрою з використанням біометрії. Дана сторінка запитує від користувача лише його username, а далі проходить звичайна процедура виклику, та його проходження, з перевіркою результату. Схема входу за WebAuthn наведена на рис. 3.8.

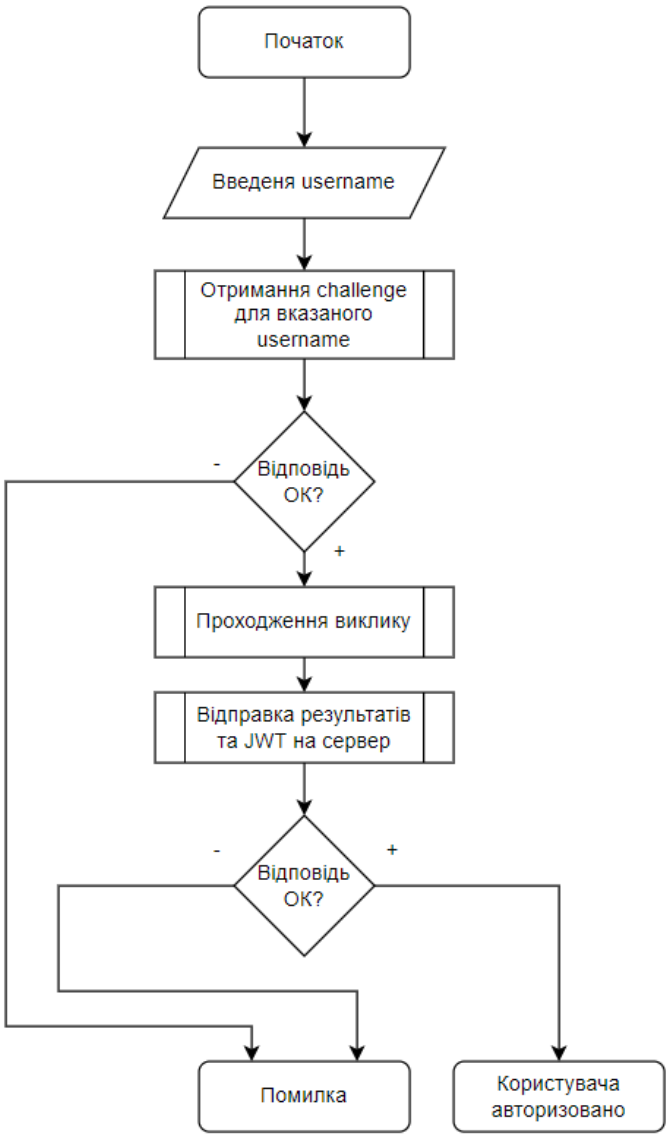


Рис. 3.8. Схема входу користувача за WebAuthn

Все по аналогії з реєстрацією, але простіше. Проте, є й нюанси. По-перше,

захист від атак відтворення – при зверненні до серверу, при коректному та наявному у БД username, лічильник збережених креденшел інкрементується, що гарантує створення унікального виклику (challenge) WebAuthn для клієнту кожного разу при зверненні. Тому, перехоплення самого виклику чи результатів його проходження абсолютно нічого не дає для зловмисника.

Окрім цього, варто також відмітити особливість роботи WebAuthn, яка грає на руку безпеці, але може створювати незручності для користувача та розробника. Справа в тому, що при реєстрації WebAuthn ініціює створення та збереження приватного ключа на пристрої користувача. Пошук ключа здійснюється за username та доменним іменем самого ресурсу, для якого він використовується. Тому, при невеличкій зміні адреси web-ресурсу, ключ вже не буде знайдено – користувач отримає повідомлення про те, що для такого домену взагалі немає збережених ключів, хоча взагалі на ньому є ключ, який би підійшов. Власне, для уникнення такої втрати доступу, при масштабуванні чи перенесенні сайту чи збоях у роботі пристрою користувача, й було введено автентифікацію по обличчю як один з варіантів доступу.

3. Сторінка входу за обличчям «signin/face» не потребує детального пояснення. Вона використовується для автентифікації по обличчю, має поле для введення username, та область для зняття дескриптору обличчя (за аналогією з реєстрацією, через *faceapi* та ті ж самі моделі ШІ). Далі, виконується запит на сервер, де шукається користувач за його іменем, та знаходиться відстань між векторами ознак обличь методом Евклідової відстані. Відносно отриманого значення відстані, та порогового значення, заданого на сервері, формується відповідь – надати доступ, чи ні. Окрім цього, окремим фактором безпеки виступає те, як працює *faceapi*, тому це може вимагати від користувача часткового відтворення умов зняття дескриптора, як при реєстарації – положення обличчя, кут нахилу, тощо. Тому, використати просту підміну реального обличчя статичною фотографією не вийде, а поріг прийому користувача задано так, що потрібне максимальне співпадіння.

4. Сторінка входу за допомогою QR-коду «signin/qr» займається

відображенням QR для цільового пристрою. При відвідання цієї сторінки, клієнт ставить запит до серверу на отримання нового QR з даними у форматі JSON. Дані, отримані дані за допомогою бібліотеки react-qr-code перетворюються на звичайне звичне квадратне зображення QR, який може бути прочитаним камерою іншого пристрою. Принцип роботи заключається в тому, що сканування ведеться зі сторінки авторизованого облікового запису. Сам згенерований QR не містить ні імені користувача, ні іншої важливої інформації, крім одноразового коду. Детальніше про це – далі.

5. Сторінка «user/account» у даному проекті слугує просто ідентифікатором того, що авторизація користувача виконана успішно. Окрім цього, на цій сторінці й міститься сканерQR-коду, так як тут вже користувач має повний доступ. Використовуючи бібліотеку react-qr-scanner, клієнт за допомогою камери пристрою вилучає дані з QR-коду у формат JSON, та надсилає підтвердження одноразового коду на сервер. Сервер, в свою чергу, зв’язує даний одноразовий код з користувачем, який його просканував, та надає доступ до облікового запису цього користувача тому пристрою, на якому в цей момент відкрито QR з цим одноразовим кодом. У контексті реалізації даного проекту, такий доступ ніде не зберігається, а отримання доступу таким шляхом передбачає лише отримання авторизованого JWT-токену, проте не збереження нового пристрою до таблиці _credentials у БД, так як для цього потрібно реєструватися через WebAuthn. Цей метод доступу швидше доданий у проект для його більшої повноти, так як він не є біометричним, проте має місце бути, в якості демонстрації одного зі шляхів відмови від факту символьного паролю, який потрібно пам’ятати.

3.4.2 Проксі-сервер та сертифікати

Використання Next.js в якості інструменту для створення front-end має ще одну безумовну перевагу. Він поєднує в собі можливості створення користувацьких інтерфейсів за допомогою засобів React.js, а також серверної реалізації Node.js. Таким чином, front-end також виступає в якості окремого

серверу, а це дозволяє додатково захистити web-ресурс.

Пам’ятаємо, що програмний код клієнтської частини, та того, що відображено у браузері – може бути там же й переглянуто через інструменти розробника. Через це, при прямих запитах зі сторінок на ASP.NET сервер, зловмисник міг би бачити:

- URL по якому йде запит;

- Структуру запиту.

Це створює певну небезпеку, так як знаючи серверний URL ендпоінта, можна робити запити взагалі без участі сторінок браузеру, та корегувати структуру запиту як завгодно, перебираючи варіанти. Цей вид атак належить типу ін’єкцій (injection), та є атакою на основі модифікації запиту. Крім цього, можна було б спробувати знайти інші endpoints, які, можливо, недоступні зі сторінок.

Для уникнення цього, використовується проксування. Суть в тому, що замість того, щоб відправити запит напряму на back-end сервер зі сторінки, запит відправляється на внутрішній сервер Next.js (проксі), а вже звідти на back-end. При цьому, користувач не має доступу до внутрішнього серверу, й не може бачити, куди цей запит далі надсилається (та чи взагалі надсилається). На рівні проксі, перед відправкою на кінцевий сервер, запит може бути видозмінено, виконано перетворення даних, тощо, про які користувач також не зможе дізнатися.

Проксування використовується для всіх запитів у проекті. Для цього створено спеціальну директорію «api» у папці «pages», яка сигналізує Next.js про те, що розробка ведеться не тільки для сторінок браузеру, а й на серверну частину.

Проксі-сервер дублює структуру реального back-end серверу, проте приховує його адресу у мережі, дозволяючи таким чином захищатися.

3.4.3 Ngrok

Ngrok – це інструмент для шлюзування локальних web-додатків у глобальну мережу інтернет. Для даного проекту питання тестування стояло гострим ребром, тому що варто було перевірити проект як з комп'ютера, так і з мобільних пристроїв.

Так як в процесі реалізації було виконано жорсткі обмеження запитів, політика CORS, самописні сертифікати HTTPS, то з'явилася проблема поширення доступу до web-ресурсу для мобільного пристрою. В якості рішення й знайдено Ngrok, так як він фактично виступає уніфікатором клієнтського домену для різних пристроїв, забезпечуючи стійку та єдину адресу доступу до клієнту.

Після реєстрації на офіційному сайті та встановленні консольного додатку на комп'ютері, задається токен доступу до інструменту. Потім, засобами Ngrok, створюється глобальний шлюз локального серверу у мережу. В якості такого серверу виступає localhost:3000, на якому розміщено Next.js front-end. При цьому, самописні сертифікати не викликають проблем, тунель відкривається на повноцінному HTTPS, єдине, що потрібно після запуску додати новий Ngrok-домен у політику CORS, як на front-end, так і на back-end.

Ще раз наголошується, що це – інструмент розробки, який дозволяє тимчасово перевірити web-ресурс в реальних умовах, ніби він розміщений на реальному сервері. При відкритті тунелю генерується унікальний URL, вигляду https://xxxx-xxx-x-xx-xxx.ngrok-free.app, де «x» – це цифри та латинські літери.

3.5 Тестування та демонстрація роботи

Для тестування розробленого web-ресурсу, було використано:

- Власний ноутбук, який використовувався для розробки, який не має сканеру відбитку пальця, системи WindowHello, та камеру поганої якості;
- Два мобільні пристрої на операційній системі Android, які мають

вбудовані ємкісні дактилоскопічні датчики, нормальні фронтальні камери, а на одному з них пристутній модуль NFC (Near Field Communication).

Для доступу використовувався уніфікований URL, отриманий за допомогою Ngrok. База даних була розгорнута локально, через MS SQL Server LocalDb.

3.5.1 Тестування за допомогою ноутбуку

Для початку, надається дозвіл на використання камери на даній сторінці. Після цього, потрібно зняти дескриптори обличчя (рис. 3.9).

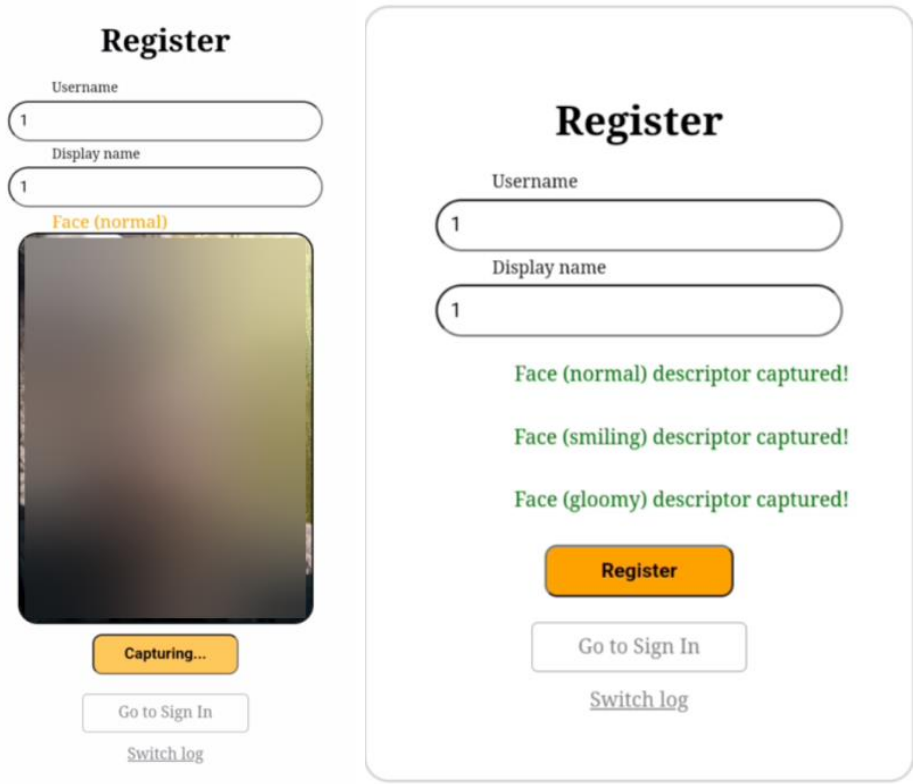
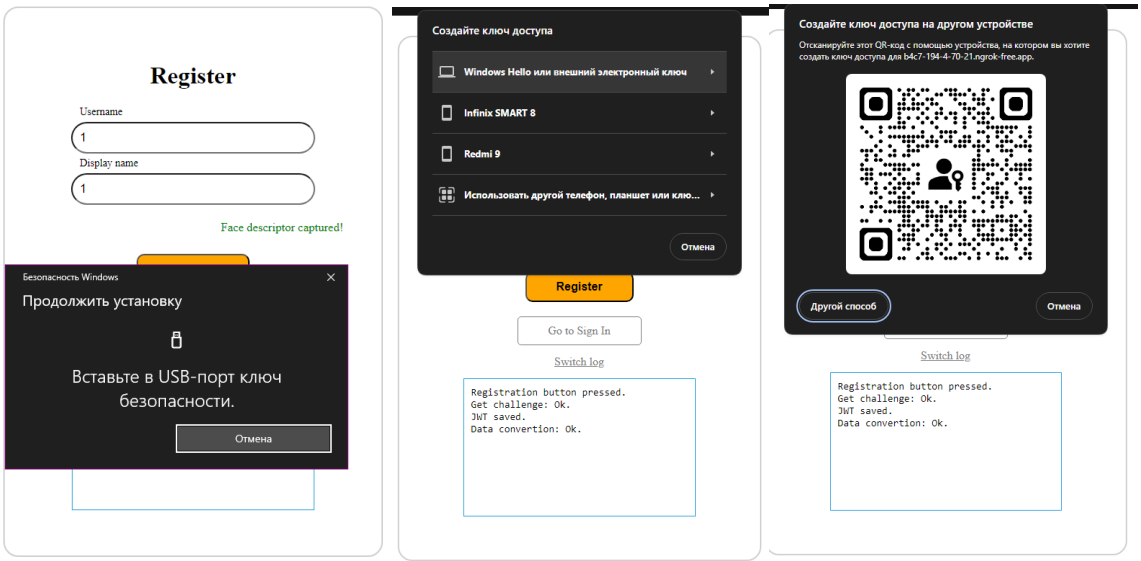


Рис. 3.9. Зняття дескрипторів обличчя

При спробі реєстрації облікового запису користувача, здійснюється спроба проходження виклику на пристрої, який ініціював виклик. Тут починає працювати WebAuthn. Перш за все, пропонується використати фізичний платформонезалежний USB FIDO2 ключ (рис. 3.10, а). При відхиленні запиту,

пропонується використати альтернативні методи підтвердження: WindowsHello, збережені пристрої (тут вони є, так як додаток вже тестувався), чи додати пристрій – телефон, планшет, тощо (рис. 3.10, б). Додавання пристрою здійснюється шляхом сканування одноразового FIDO2 QR-коду (рис. 3.10, в).



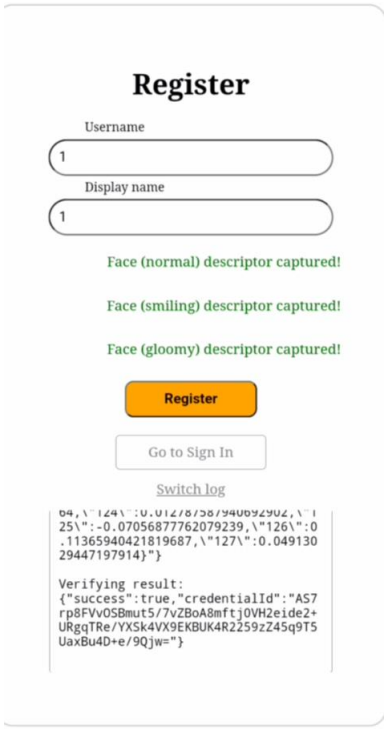
(а) (б) (в)

Рис. 3.10. Робота WebAuthn на ноутбуці

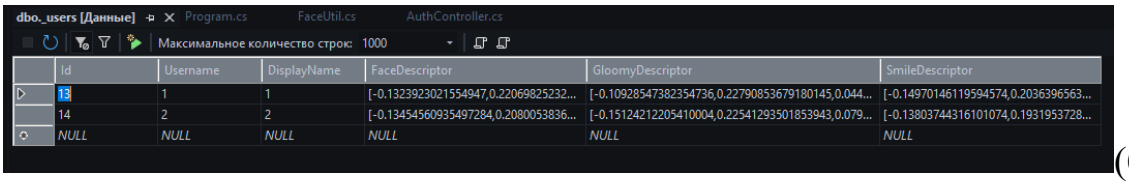
Проблеми виникають при спробі використання телефону в якості автентифікатору, так як заданий формат QR-коду не розпізнається звичайними сканерами у браузерах, тому що він має свій власний формат даних – FIDO2. Єдиний додаток, який було знайдено для успішного сканування цього QR – Google Authenticator. Хоча цей додаток й задумано для зберігання токенів доступу у Google та синхронізація їх з хмарним сховищем користувача, його використання не суперечить принципу відмови від залежності від інших сервісів – при використанні телефону з Google Authenticator, токен доступу FIDO2 не з’явився у списку збережених ключів, хоча й був збережений у TrustZone пристрою, що підтверджується наступними успішними автентифікаціями за його допомогою. Тому, у контексті цього проекту, Google Authenticator виступає не більш як сканер QR-коду.

Скріншоти цього процесу на мобільному пристрої зняти неможливо, так як

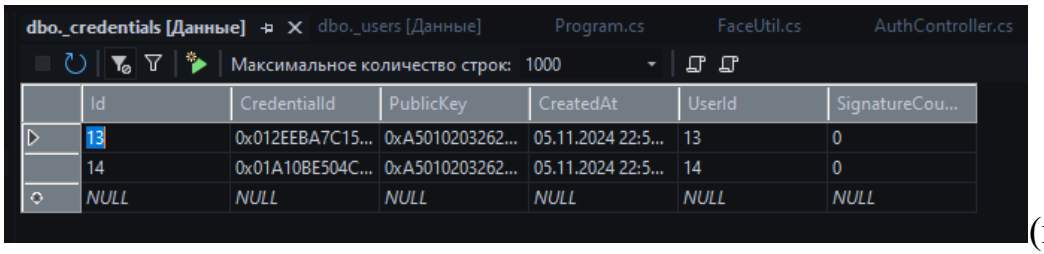
додаток блокує зняття знімку екрану. Після проходження реєстрації, можна бачити повідомлення про успіх у логах сторінки (рис. 3.11, а), та те, що у відповідних таблицях з’явились записи про користувача (рис. 3.11, б) та пристрій (рис. 3.11, в).



(а)



(б)



(в)

Рис. 3.11. Успіх реєстрації: (а) на сторінці, (б) таблиця _users, (в) таблиця _credentials

Спроба авторизації за обличчям вимагає чіткого позиціонування обличчя в кадрі та якості зображення, для того, щоб якомога точніше зняти дескриптор. Тому, варто омовитися, що якщо дескриптор при реєстрації знімався з поганою

якістю, то й верифікацію потрібно буде проходити в тих самих умовах. Так і навпаки, якщо дескриптор знімався з телефона з гарною камерою, в ясний сонячний день, де на фото чітко видно всі риси обличчя користувача, автентифікуватися за допомогою фото з камери ноутбуку роздільною здатністю 0.3 МП не вийде. Обраний поріг відстаней векторів дозволяє убезпечити це, так как граничне значення виставлено на 0.3, а це доволі точний показник. Під час тестів авторизації з різних відстаней на камеру ноутбуку, де з відстанню дуже сильно страждає якість, відстань між векторами часто сягала: ≈0.31, ≈0.35, ≈0.4, й це вже не дозволяє авторизуватися.

Сторінку авторизованого користувача показано на рис. 3.12.

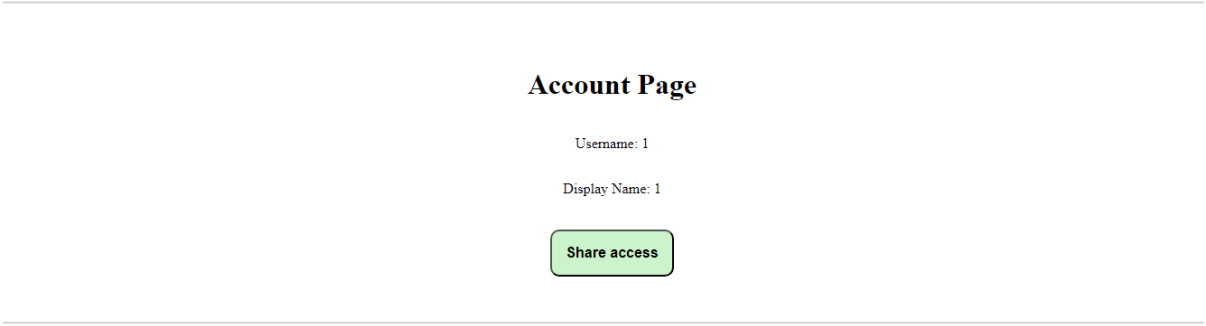


Рис. 3.12 – Сторінка авторизованого користувача

Для прикладу, виведено імена користувача, а функціональне значення має кнопка «Share Access», яка відкриває камеру для сканування QR-коду доступу зі сторінки «signin/qr».

Повертаючись до методів авторизації, якщо для доступу до акаунту використати WebAuthn, процедура дуже схожа на реєстрацію, проте не потребує фото обличчя, та просить лише один username – основний. Як і на рис. 3.10, пропонується обрати пристрій для доступу, та пройти на ньому виклик. При цьому, якщо пристрій вже збережено, не потрібно сканувати QR-код за допомогою Google Authenticator. Та навіть більше: навіть при відсутності додатку Google Authenticator на смартфоні, виклик на ньому буде ініційовано, що ще раз доводить, що облікові дані зберігаються не в додатку, а додаток потрібен

лише для сканування QR-коду FIDO2 при першому доступі, доки пристрої ще не пов’язані.

Розглядати QR-автентифікацію web-ресурсу не будемо, так як там нічого особливого не має, та вона не несе нічого спільного з використанням біометрії, існуючи більше як елемент зручності та периферії.

3.5.2 Тестування на мобільному пристрої

Перш за все варто відмітити, що все ще не всі браузери реалізують підтримку WebAuthn, тому з деяких із них не вийде виконати реєстрацію, а авторизуватися можна лише по обличчю. Приклад наведено на рис. 3.13, де у логах відображено, що браузер не підтримує технологію. Використовувався Soul Browser v.1.4.32.

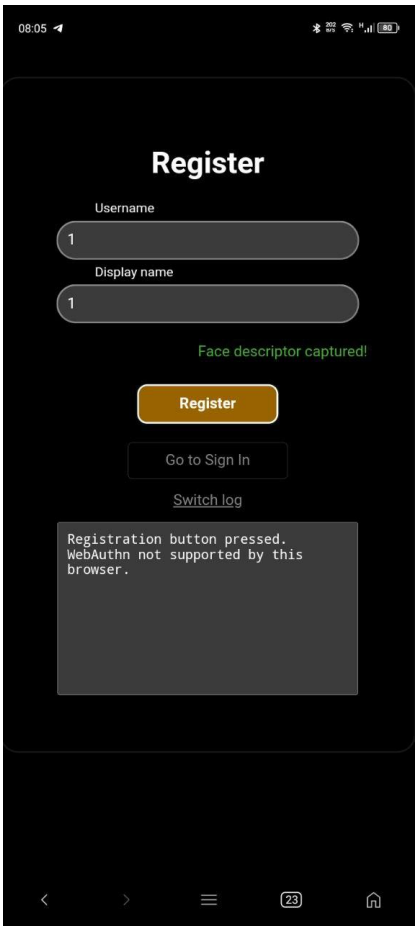


Рис. 3.13 – Відсутня підтримка WebAuthn браузером

Проте, це не означає що сам пристрій його не підтримує, так як у Google Chrome все чудово працює. Проблема підтримки WebAuthn браузерами – це проблема розробників браузерів, так як дана технологія визнана як стандарт, й має бути підтримувана у сучасних розробках.

В подальшому ж, все аналогічно до ПК, окрім пункту автентифікації за WebAuthn. З Google Chrome також бачимо варіанти отримання доступу та проходження виклику, але на пристрої з вбудованою біометрією також є кнопка «Цей пристрій» (рис. 3.14).

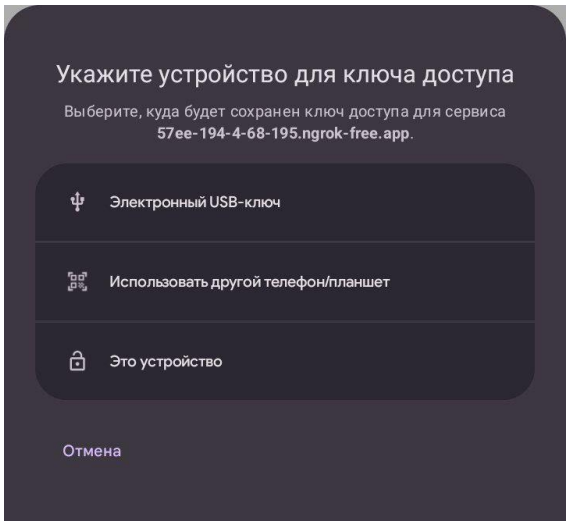


Рис. 3.14 – Варіанти проходження виклику на мобільному пристрої

На смартфоні з апаратною підтримкою NFC, існує ще один варіант: пройти виклик через цю технологію. Для використання WebAuthn на пристрої з підтримкою біометрії, не потрібно сканувати будь-які QR-коди, виклик ініціюється самим браузером прямо у систему, яка й звертається до апаратних засобів підтвердження пристрою.

В підсумку тестування було проведено 100 спроб входу до створеного облікового запису за допомогою обличчя, та розраховано помилки 1-го та 2-го роду. В контексті біометричного входу, помилка першого роду (FRR) означає частину відхилення доступу для користувача, який зареєстрований в системі, а

помилка другого роду (FAR) – частина схвалення входу для користувачів, які не були зареєстровані у системі (помилкові схвалення). Результати тестування наведено у табл. 3.3.

Таблиця 3.3

Помилки входу за допомогою обличчя

Кількість спроб	**Помилкових відхилень**	**Помилкових схвалень**	**FRR**	**FAR**
100	5	0	5%	0%

В результаті тестування, бачимо, що система має високу надійність, яка зумовлена зняттям трьох дескрипторів обличчя, але через це зростає показник FRR (помилкові відхилення зареєстрованого користувача), якщо відрізняються умови зйомки та вирази обличчя.

3.6 Висновки до розділу 3

У даному розділі було продемонстровано процес тестування та роботу розробленого web-ресурсу з використанням біометричної аутентифікації на основі WebAuthn. Було проведено тестування на різних пристроях: ноутбуці без вбудованих біометричних датчиків та мобільних телефонах з підтримкою дактилоскопічних сенсорів і камер різної якості.

У процесі тестування виявлено кілька важливих особливостей:

1. Використання ноутбука з низькою якістю камери ускладнює розпізнавання обличчя та може вплинути на точність верифікації. Це підтвердило, що якість зображення під час реєстрації і авторизації повинна бути співставною. Водночас, це й виступає додатковим захистом.

2. Формат QR-коду, що генерується для WebAuthn, не підтримується стандартними сканерами у браузерах. Однак Google Authenticator успішно

використовується як інструмент для сканування, не порушуючи принцип відмови від сторонніх сервісів.

3. На мобільних пристроях із вбудованою біометрією було доведено можливість ініціювати запити без сканування QR-кодів, що значно спрощує процес доступу. Підтримка WebAuthn залежить від браузера: не всі браузери, мають повну інтеграцію технології.

Загалом результати тестування підтверджують працездатність і надійність розробленої системи аутентифікації. Незважаючи на деякі обмеження, пов'язані з обладнанням та підтримкою WebAuthn у різних браузерах, реалізований підхід демонструє перспективність використання біометрії для створення зручної та безпечної авторизації без використання паролів чи прив’язки до електронної пошти або телефону.

ВИСНОВКИ

У межах дипломної роботи на тему «Метод авторизації користувачів на web-ресурсах з використанням біометрії» було розроблено web-ресурс із підтримкою біометричної аутентифікації на основі технології WebAuthn. Основна мета дослідження полягала у створенні системи авторизації, яка б забезпечувала безпечний доступ без використання паролів, електронної пошти або прив’язки до номера телефону.

У процесі виконання роботи було вирішено декілька завдань:

1. Теоретичне обґрунтування важливості використання біометричних даних для сучасної автентифікації.

2. Аналіз технології WebAuthn, її принципів роботи та можливостей інтеграції.

3. Реалізація прототипу системи з використанням ASP.NET Core на бекенді та Next.js на фронтенді.

4. Імплементація біометрії за допомогою face-api.js для обробки зображень на клієнтській стороні та Fido2 для взаємодії з браузером і ключами автентифікації.

5. Тестування та аналіз працездатності системи на різних пристроях і платформах, щоб оцінити ефективність підходу.

Загалом у роботі було підтверджено, що використання біометричних даних є перспективним напрямом для підвищення зручності та безпеки авторизації, хоча й виникають певні виклики щодо якості обладнання та підтримки браузерами.

У першому розділі було проаналізовано сучасні методи авторизації, зокрема паролі, двофакторну автентифікацію та біометричні технології. Було розглянуто їхні переваги та недоліки. Особлива увага приділялася питанням безпеки та зручності для кінцевих користувачів. На основі цього аналізу зроблено висновок, що традиційні методи, такі як паролі та SMS-коди, поступаються біометричним системам, які забезпечують одночасно високу

безпеку та простоту використання.

Також було визначено актуальність використання WebAuthn як сучасного стандарту для безпарольної аутентифікації, що не вимагає прив’язки до конкретного пристрою або додатку.

Другий розділ був присвячений огляду фізичних пристроїв для вилучення біометричних ознак користувачів, підіймалося питання їх доступності, й визначено, що сканер відбитку пальця та камера зараз присутні навіть у найдешевших смартфонах. Розглянуто, яким чином працює біометрія, та як початкові дані перетворюються у ключі.

У третьому розділі було детально розглянуто процес реалізації та тестування розробленої системи. Було прийнято низку важливих рішень:

– ASP.NET Core обрано для реалізації бекенд-сервісу завдяки його продуктивності та зручності в роботі з аутентифікацією.

– На фронтенді використано Next.js для забезпечення швидкого рендерингу та зручної взаємодії з API.

– Fido2 обрано для реалізації протоколу WebAuthn, що дозволяє здійснювати аутентифікацію за допомогою криптографічних ключів.

– face-api.js використано для обробки біометричних даних (розпізнавання обличчя) на клієнтському боці, що забезпечує високу швидкість роботи та захист персональних даних.

Тестування проводилося на ноутбуках і мобільних телефонах із різними характеристиками камер та біометричних сенсорів. Отримані результати показали, що:

– Якість обладнання впливає на точність розпізнавання обличчя: низька роздільна здатність камер на ноутбуках може ускладнити авторизацію.

– Використання Google Authenticator як сканера QR-кодів підтвердило зручність, хоча технологія WebAuthn не завжди підтримується браузерами.

– На мобільних пристроях із вбудованими біометричними сенсорами авторизація працює швидше та без сканування QR-кодів, що робить процес доступу ще зручнішим.

Загальні результати підтвердили, що розроблена система є працездатною, проте для її ефективної роботи потрібні сучасні пристрої та браузери з підтримкою WebAuthn.

Реалізована у межах дипломної роботи система біометричної аутентифікації продемонструвала свою ефективність і перспективність для використання на web-ресурсах. Вона відповідає вимогам сучасних тенденцій у галузі кібербезпеки, відмовляючись від використання паролів, електронної пошти та телефонів.

Використання біометрії підвищує рівень безпеки та забезпечує більш зручний досвід для користувачів. Водночас, важливу роль відіграє якість обладнання: розпізнавання обличчя може бути ускладнене на пристроях із низькою роздільною здатністю камер. Також важливо враховувати обмеження щодо сумісності браузерів із WebAuthn, що потребує використання новітніх версій популярних браузерів.

Розроблена система відкриває нові можливості для авторизації на web-ресурсах, дозволяючи:

1. Забезпечити швидкий та безпечний доступ до ресурсів без необхідності запам’ятовувати паролі.

2. Використовувати біометричні дані або криптографічні ключі для безпечної автентифікації.

3. Відмовитися від прив’язки до сторонніх сервісів, зберігаючи дані локально та захищаючи приватність користувачів.

Загалом, реалізація даного проєкту підтвердила, що біометрична аутентифікація має потенціал для широкого впровадження на різних платформах, підвищуючи як зручність користування, так і безпеку доступу.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Windows: Local Users. URL: https://learn.microsoft.com/en-us/windows/security/identity-protection/access-control/local-accounts (дата звернення 01.09.2024).

2. Android: Delete, switch, or add users. URL: https://support.google.com/android/answer/2865483?hl=en#zippy=%2Cswitch-user%2Cdelete-user (дата звернення 01.09.2024).

3. 7 Best Practices for Social Media Security. URL: https://cybersierra.co/blog/best-practices-for-social-media-usage/ (дата звернення 02.09.2024).

4. What Is Identification, Authentication and Authorization? URL: https://www.proof.com/blog/what-is-identification-authentication-and-authorization#:~:text=Identification%20and%20authentication%20validate%20a,identifying%2C%20authenticating%20and%20authorizing%20them. (дата звернення 02.09.2024).

5. How do passwords work? URL: https://delinea.com/blog/how-do-passwords-work#:~:text=Password%20hashing%20turns%20your%20password,getting%20access%20to%20your%20passwords. (дата звернення 02.10.2024).

6. How to set up a Gmail account in under five minutes. URL: https://www.hubspot.com/email-signature-generator/set-up-gmail-account (дата звернення 03.09.2024).

7. Telephone number. URL: https://en.wikipedia.org/wiki/Telephone_number (дата звернення 03.09.2024).

8. Реєстрація SIM-карт за паспортами — наразі фейк. URL: https://ratusha.lviv.ua/reyestracziya-sim-kart-za-pasportamy-narazi-fejk/ (дата звернення 04.09.2024).

9. SIM swapping attacks. URL: https://www.ownyouronline.govt.nz/personal/know-the-risks/common-risks-and-threats/sim-swapping-attacks/ (дата звернення 04.09.2024).

10. Мобільні оператори “Київстар”, Vodafone та Lifecell продають старі номера абонентів. URL: https://tehnofan.com.ua/2021/09/21/mobil′ni-operatory-kyyivstar-vodafone-ta-lifecell-prodayut′-stari-nomera-abonentiv/ (дата звернення 05.09.2024).

11. How To Delete Your Telegram Account: A Step-by-Step Guide. URL: https://www.comparitech.com/blog/vpn-privacy/delete-telegram-account/#:~:text=what%20to%20do%3A-,Open%20the%20Telegram%20mobile%20app,six%20months%2C%20or%20a%20year. (дата звернення 05.09.2024).

12. Що робити, якщо прийшло багато смс з кодами підтвердження з різних сайтів. URL: https://gsminfo.com.ua/41880-shho-robyty-yakshho-pryjshlo-bagato-sms-z-kodamy-pidtverdzhennya-z-riznyh-sajtiv.html (дата звернення 06.09.2024).

13. Двофакторна аутентифікація. URL: https://uk.wikipedia.org/wiki/Довідка:Двофакторна_аутентифікація#:~:text=Двофакторна%20автентифікація%20(ДФА%2C%202FA),мобільному%20пристрої%20чи%20комп%27ютері. (дата звернення 06.09.2024).

14. HTTP-запит (HTTP request). URL: https://qalight.ua/baza-znaniy/http-zapyt-http-request/ (дата звернення 07.09.2024).

15. Центр безпеки Google. URL: https://safety.google/intl/uk/ (дата звернення 10.09.2024).

16. 1Password. URL: https://1password.com/ (дата звернення 11.09.2024).

17. OAuth 2.0. URL: https://oauth.net/2/ (дата звернення 12.09.2024).

18. Вхід через Google. URL: https://support.google.com/accounts/answer/12849458?hl=uk#zippy=%2Cкнопка-увійти-через-google (дата звернення 12.09.2024).

19. Introduction to JSON Web Tokens. URL: https://jwt.io/introduction (дата звернення: 13.09.2024).

20. Stateless Authentication. URL: https://doubleoctopus.com/security-wiki/network-architecture/stateless-authentication/ (дата звернення: 13.09.2024).

21. Створення та використання надійних паролів. URL:

https://support.microsoft.com/uk-ua/windows/створення-та-використання-надійних-паролів-c5cebb49-8c53-4f5e-2bc4-fe357ca048eb (дата звернення: 14.09.2024).

22. Як відновити свій обліковий запис Google? URL: https://support.google.com/accounts/answer/7299973?hl=uk (дата звернення: 15.09.2024).

23. Set account recovery methods in case you forget your Proton password. URL: https://proton.me/support/set-account-recovery-methods (дата звернення: 15.09.2024).

24. Відновлення втраченого або забутого акаунта Steam. URL: https://help.steampowered.com/uk/faqs/view/3944-4D89-1B3E-27DE (дата звернення: 15.09.2024).

25. Біометрія. URL: https://uk.wikipedia.org/wiki/Біометрія (дата звернення: 16.09.2024).

26. Методи блокування телефона: як надійно захистити смартфон на базі Android. URL: https://www.eset.com/ua/about/newsroom/blog/smartphone-security/metody-blokirovki-telefona-kak-nadezhno-zashchitit-smartfon-na-baze-android/ (дата звернення: 17.09.2024).

27. Що таке BankID? URL: https://www.kmu.gov.ua/usi-pitannya-po-e-poslugam/sho-take-bankid (дата звернення: 18.09.2024).

28. Сервіс “BankID”. URL: https://conditions-and-rules.privatbank.ua/main/view-content-489/?lang=uk (дата звернення: 18.09.2024).

29. WebAuthn. A better alternative for securing our sensitive information online. URL: https://webauthn.guide/ (дата звернення: 19.09.2024).

30. What Is FIDO2 and How Does It Work? URL: https://hideez.com/en-ua/blogs/news/fido2-explained (дата звернення: 20.09.2024).

31. WebAuthn. URL: https://uk.wikipedia.org/wiki/WebAuthn (дата звернення: 20.09.2024).

32. Що безпечніше: сканер відбитків пальця чи розпізнавання обличчя? URL: https://www.imena.ua/blog/fingerprint-scanner-vs-facial-recognition/ (дата

звернення: 21.09.2024).

33. Understand fingerprint security URL: https://support.google.com/pixelphone/answer/6300638?hl=en#zippy=%2Cfingerprint-hardware-security-requirements (дата звернення: 22.09.2024).

34. Procedia Computer Science. URL: https://www.sciencedirect.com/science/article/pii/S1877050910003479 (дата звернення: 23.09.2024).

35. Using WebAuthn. URL: https://webauthn.io/ (дата звернення: 27.09.2024).

36. OpenSSL. URL: https://www.openssl.org/ (дата звернення: 28.09.2024).

37. mkcert. URL: https://github.com/FiloSottile/mkcert (дата звернення: 28.09.2024).

38. ngrok. URL: https://ngrok.com/ (дата звернення 28.09.2024).

39. Microsoft SQL Server EF Core Database Provider. URL: https://learn.microsoft.com/uk-ua/ef/core/providers/sql-server/?tabs=dotnet-core-cli (дата звернення 29.09.2024).

40. Enable Cross-Origin Requests (CORS) in ASP.NET Core. URL: https://learn.microsoft.com/uk-ua/aspnet/core/security/cors?view=aspnetcore-8.0 (дата звернення 30.09.2024).

ДОДАТКИ

Додаток А

Код контролеру AuthController.cs

```
using Microsoft.AspNetCore.Mvc;
using Fido2NetLib;
using Fido2NetLib.Objects;
using System.Text;

using JsonSerializer = System.Text.Json.JsonSerializer;

using WebAuthnServerApp.Models;
using WebAuthnServerApp.DB;
using Microsoft.EntityFrameworkCore;
using System.Diagnostics;
using WebAuthnServerApp.Utils;
using Microsoft.IdentityModel.Tokens;

using WebAuthnServerApp.Models.DbEntities;


namespace WebAuthnServerApp.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class AuthController : ControllerBase
    {
        private readonly IFido2 fido2;
        private readonly JwtUtil _jwtUtil;
        private readonly IConfiguration _configuration;

        public   AuthController(IFido2   fido2,   JwtUtil   jwtUtil,
IConfiguration configuration)
        {   //  DI FIDO2
            this.fido2 = fido2;
            _jwtUtil = jwtUtil;
            _configuration = configuration;
        }

        // Generate challenge for client, gets User data
        [HttpPost("register/challenge")]
        public  async  Task<JsonResult>  RegisterChallenge([FromBody]
UserModel user)
        {
            // User params
            var fido2User = new Fido2User
            {
                DisplayName = user.DisplayName,
```

```
                    Name = user.Username,
                    Id = Encoding.UTF8.GetBytes(user.Username)
                };

                //  Get new credentials
                var options = fido2.RequestNewCredential(fido2User, new
List<PublicKeyCredentialDescriptor>(),
                    AuthenticatorSelection.Default,
AttestationConveyancePreference.None);

                // Convert options to JSON string
                string optionsJson = JsonSerializer.Serialize(options);

                //  Save user data to JWT
                string jwt = _jwtUtil.GenerateJwtToken(user.Username,
user.DisplayName, optionsJson);

                // Returns challenge for WebAuthn API on client
                return new JsonResult(new { options, jwt });
            }



            //  Generate chellange for SignIn process
            [HttpPost("signin/chellange")]
            public async Task<JsonResult> SignInChellange([FromBody]
string username)
            {
                // Get user from DB
                using (BioDBContext dBContext = new())
                {
                    var dbUser = await
dBContext._users.FirstOrDefaultAsync(u => u.Username == username);

                    if (dbUser == null)
                        return new JsonResult(new { success = false,
error = "User not found" });

                    // Create Fido2User from user data
                    var fido2User = new Fido2User
                    {
                        DisplayName = dbUser.DisplayName,
                        Name = dbUser.Username,
                        Id = Encoding.UTF8.GetBytes(dbUser.Username)
                    };

                    // Get all credentials for this user
                    var credentials = await dBContext._credentials
                        .Where(c => c.UserId == dbUser.Id)
                        .Select(c => new
PublicKeyCredentialDescriptor(c.CredentialId))
```

```
                        .ToListAsync();

                    // Generate challenge for WebAuthn
                    var options = fido2.GetAssertionOptions(credentials,
UserVerificationRequirement.Required);

                    // Convert options in JSON
                    string                optionsJson                =
JsonSerializer.Serialize(options);

                    // Generate JWT
                    string                    jwt                    =
_jwtUtil.GenerateJwtToken(dbUser.Username,           dbUser.DisplayName,
optionsJson);


                    return new JsonResult(new { options, jwt, challenge
= options.Challenge });
                }
            }




            //  Verifuing users challenge response
            [HttpPost("signin/verify")]
            public   async   Task<JsonResult>   SignInVerify([FromBody]
AuthenticatorAssertionRawResponse response)
            {
                var                    authHeader                    =
HttpContext.Request.Headers.Authorization.ToString();
                string  token  =  authHeader.StartsWith("Bearer  ")  ?
authHeader.Substring("Bearer ".Length).Trim() : null;

                if (string.IsNullOrEmpty(token))
                    return new JsonResult(new { success = false, error =
"Backend: JWT missing" });

                // check JWT
                AuthJwtModel? jwtModel = null;
                try { jwtModel = _jwtUtil.GetModelFromJwt(token); }
                catch  (Exception  ex)  {  return  new  JsonResult(new  {
success = false, error = $"Backend: Invalid JWT -> {ex.Message}" }); }

                if (string.IsNullOrEmpty(jwtModel.optionsJson))
                    return new JsonResult(new { success = false, error =
"Backend: Missing assertion options in JWT" });

                // deserialize saved options
                AssertionOptions            cachedOptions            =
```

```
JsonSerializer.Deserialize<AssertionOptions>(jwtModel.optionsJson);

                using (BioDBContext dBContext = new())
                {
                    // find user
                    var            dbUser            =            await
dBContext._users.FirstOrDefaultAsync(u       =>       u.Username       ==
jwtModel.username);
                    if (dbUser == null)
                        return  new  JsonResult(new  {  success  =  false,
error = "User not found" });

                    // get all user`s credentials
                    var credentials = await dBContext._credentials
                        .Where(c => c.UserId == dbUser.Id)
                        .ToListAsync();

                    if (!credentials.Any())
                        return  new  JsonResult(new  {  success  =  false,
error = "No credentials found for user" });


                    AssertionVerificationResult result;


                    // Go throw all credentials for this user
                    foreach (var credential in credentials)
                    {
                        byte[] storedPublicKey = credential.PublicKey;
                        uint          storedSignatureCounter          =
credential.SignatureCounter;

                        try
                        {
                            // 1. Get clientDataJSON from JSON
                            var             clientDataJSON            =
response.Response.ClientDataJson;

                            // 2. Deserialize clientDataJSON
                            var               clientData               =
JsonSerializer.Deserialize<Dictionary<string, object>>(clientDataJSON);

                            // 3. Get chellange from clientData
                            if (!clientData.TryGetValue("challenge", out
var challengeObj))
                                return  new  JsonResult(new  {  success  =
false, error = "Challenge not found in clientData" });

                            var           receivedChallenge           =
challengeObj.ToString();
```

```
                            // 4. Convert chellange
                            var           originalChallenge           =
Base64UrlEncoder.Encode(cachedOptions.Challenge);                       //
cachedOptions.Challenge - byte[]

                            if (receivedChallenge != originalChallenge)
                                return new JsonResult(new { success =
false, error = "Challenge not equal to original challenge" });

                            // Library check
                            result = await fido2.MakeAssertionAsync(
                                response,
                                cachedOptions,
                                storedPublicKey,
                                storedSignatureCounter,
                                async (args, cancellationToken) => {
return await Task.FromResult(true); }
                            );
                        }
                        catch (Exception ex) { return new JsonResult(new
{ success = false, error = ex.Message }); }

                        if (result.Status == "ok")
                        {
                            // Update counter in DB
                            credential.SignatureCounter                 =
result.Counter;
                            await dBContext.SaveChangesAsync();

                            var                   jwt                   =
_jwtUtil.AuthorizedJwt(jwtModel.username, jwtModel.displayName);

                            return new JsonResult(new { success = true,
jwt });
                        }
                    }

                    return new JsonResult(new { success = false, error =
"No valid credentials in DB" });
                }
            }




            [HttpPost("signin/face")]
            public   async   Task<JsonResult>   SignInByFace([FromBody]
FaceSignInModel faceRequest)
            {
                try
                {
```

```
                    List<double>                face                =
JsonSerializer.Deserialize<Dictionary<string,
double>>(faceRequest.faceDescriptor).Values.ToList();
                    List<double>             faceSmiling            =
JsonSerializer.Deserialize<Dictionary<string,
double>>(faceRequest.smileDescriptor).Values.ToList();
                    List<double>             faceGloomy             =
JsonSerializer.Deserialize<Dictionary<string,
double>>(faceRequest.gloomyDescriptor).Values.ToList();


                    using (BioDBContext context = new())
                    {
                        UserModel?         dbUser         =         await
context._users.FirstOrDefaultAsync(x        =>        x.Username        ==
faceRequest.username);
                        if     (dbUser     ==     null)     throw     new
ArgumentException("No such user");

                        List<double>              dbFace             =
JsonSerializer.Deserialize<List<double>>(dbUser.FaceDescriptor);
                        List<double>            dbFaceSmile          =
JsonSerializer.Deserialize<List<double>>(dbUser.SmileDescriptor);
                        List<double>            dbFaceGloomy         =
JsonSerializer.Deserialize<List<double>>(dbUser.GloomyDescriptor);

                        //  Evclid`s distance

                        double                faceMatch              =
FaceUtil.computeFaceDifference(face, dbFace);
                        double              faceSmileMatch           =
FaceUtil.computeFaceDifference(faceSmiling, dbFaceSmile);
                        double             faceGloomyMatch           =
FaceUtil.computeFaceDifference(faceGloomy, dbFaceGloomy);

                        if(faceMatch >= 0.3 || faceSmileMatch >= 0.3 ||
faceGloomyMatch >= 0.3)
                            throw    new    Exception($"Face    mismatch,
({(faceMatch + faceSmileMatch + faceGloomyMatch) / 3.0})");



                        string                  jwt                  =
_jwtUtil.AuthorizedJwt(dbUser.Username, dbUser.DisplayName);
                        return new JsonResult(new { success = true, jwt
});
                    }
                }
                catch  (Exception  ex)  {  return  new  JsonResult(new  {
success = false, error = $"Error in FaceSignIn: {ex.Message}" }); }
            }
```

```
            // Verifying user after challenge
            [HttpPost("register/verify")]
            public async Task<JsonResult> RegisterVerify([FromBody]
RegisterVerifyModel verifyReqBody)
            {

                var authHeader =
HttpContext.Request.Headers.Authorization.ToString();
                string token = authHeader.StartsWith("Bearer ") ?
authHeader.Substring("Bearer ".Length).Trim() : null;

                if (string.IsNullOrEmpty(token))
                    return new JsonResult(new { success = false, error =
"Backend: JWT missing" });


                // Check JWT
                AuthJwtModel? jwtModel = null;
                try { jwtModel = _jwtUtil.GetModelFromJwt(token); }
                catch (Exception ex) { return new JsonResult(new {
success = false, error = $"Backend: Invalid JWT -> {ex.Message}" }); }


                //      Deserialize      challenge      options      to
CredentialCreateOptions obj
                CredentialCreateOptions?        cachedOptions        =
JsonSerializer.Deserialize<CredentialCreateOptions>(jwtModel.optionsJson)
;
                Fido2.CredentialMakeResult success;

                Debug.WriteLine(verifyReqBody.faceDescriptor);
                List<double>                  face                  =
JsonSerializer.Deserialize<Dictionary<string,
double>>(verifyReqBody.faceDescriptor).Values.ToList();
                List<double>               faceSmiling               =
JsonSerializer.Deserialize<Dictionary<string,
double>>(verifyReqBody.smileDescriptor).Values.ToList();
                List<double>               faceGloomy                =
JsonSerializer.Deserialize<Dictionary<string,
double>>(verifyReqBody.gloomyDescriptor).Values.ToList();


                try
                {
                    // Generate credentials
                    success = await fido2.MakeNewCredentialAsync(
                        verifyReqBody.attestationResponse,
                        cachedOptions,
                        async (IsCredentialIdUniqueToUserParams args,
```

```
CancellationToken cancellationToken) =>
                        {
                            bool isUnique = false;

                            using (BioDBContext dBContext = new())
                            {
                                var      dbCredential      =      await
dBContext._credentials.FirstOrDefaultAsync(x    =>    x.CredentialId    ==
args.CredentialId);
                                if  (dbCredential  ==  null)  isUnique  =
true;
                            }

                            return await Task.FromResult(isUnique);
                        }
                    );
                }
                catch  (Exception  ex)  {  return  new  JsonResult(new  {
success = false, error = $"Backend: NewCredentil error -> {ex.Message}"
}); }


                if (success.Status != "ok")
                    return new JsonResult(new { success = false, error =
success.ErrorMessage });

                // Save user and credential to DB
                try
                {
                    using (BioDBContext dBContext = new())
                    {
                        //  Check usernames in DB
                        UserModel?        dbUser        =        await
dBContext._users.FirstOrDefaultAsync(x =>
                            x.Username == jwtModel.username ||
                            x.DisplayName == jwtModel.displayName);

                        if (dbUser != null)
                            return new JsonResult(new { success = false,
error = "User with such Username or DisplayName already exist." });

                        //  Create User obj
                        UserModel newUser = new UserModel
                        {
                            Username = jwtModel.username,
                            DisplayName = jwtModel.displayName,
                            FaceDescriptor                                =
JsonSerializer.Serialize(face).ToString(),
                            SmileDescriptor                               =
JsonSerializer.Serialize(faceSmiling).ToString(),
                            GloomyDescriptor                              =
```

```
JsonSerializer.Serialize(faceGloomy).ToString()
                            };


                            //  Add and save user in DB
                            await dBContext._users.AddAsync(newUser);
                            await dBContext.SaveChangesAsync();

                            // Create CredentialModel obj
                            var newCredential = new CredentialModel
                            {
                                CredentialId = success.Result.CredentialId,
                                PublicKey = success.Result.PublicKey,
                                CreatedAt = DateTime.UtcNow,
                                UserId = newUser.Id,
                                SignatureCounter = 0
                            };

                            //  Add and save Credential in DB
                            await
dBContext._credentials.AddAsync(newCredential);
                            await dBContext.SaveChangesAsync();
                        }

                        return new JsonResult(new { success = true,
credentialId = success.Result.CredentialId });
                    }
                    catch (Exception ex) { return new JsonResult(new {
success = false, error = ex.Message }); }
                }
            }
        }
```