НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри
_____ Сергій Стіренко
(підпис)
“__” _____2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

на тему: Сервіс багатофакторної автентифікації для веб-застосунків

Виконав (-ла): студент (-ка) 4 курсу, групи ІВ-71
(шифр групи)

Повх Михайло Іванович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник к.т.н., доц. Роковий О.П. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) проф. д.т.н. Сімоненко В.П. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Повха Михайла Івановича

1. Тема проєкту *Сервіс багатофакторної автентифікації для веб-застосунків*
керівник проєкту Роковий Олександр Петрович, к.т.н., доц.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 11.05 2021 року № 1139-с
2. Термін здачі студентом закінченого проєкту *29 травня 2021 р.*
3. Вихідні дані до проєкту *технічна документація, прикладний програмний інтерфейс.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Опис предметної області, аналіз наявних програмних рішень, розробка

архітектури сервісу багатофакторної автентифікації для веб-застосунків, тестування сервісу

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) діаграма класів, діаграма розгортання застосунку, блок-схеми алгоритмів роботи програми

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормконтроль	Сімоненко В. П.		

7. Дата видачі завдання 15.12.2020

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2020-15.12.2020*	
2.	*Вивчення та аналіз завдання*	*15.12.2021-15.03.2021*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2021-25.03.2021*	
4.	*Розробка структур окремих підсистем*	*25.03.2021-5.04.2021*	
5.	*Програмна реалізація системи*	*5.04.2021-15.04.2021*	
6.	*Оформлення пояснювальної записки*	*15.04.2021-20.05.2021*	
7.	*Захист програмного продукту*	*25.04.2021*	
8.	*Передзахист*	*23.05.2021*	
9.	*Захист*	*14.06.2021*	

Студент-дипломник _____
(підпис)

Керівник проєкту _____
(підпис)

Анотація

У даній роботі розглянуто процес автентифікації, його особливості та застосування, а також виконаний аналіз багатофакторної автентифікації, який є одним з найактуальніших та найпоширеніших видів автентифікації. Досліджено сучасні технології і удосконалені механізми автентифікації для вирішення задач, необхідних для коректної роботи веб-застоснунків.

Особлива увага приділена розробці та моделюванню власного сервісу багатофакторної автентифікації, який забезпечить надійну автентифікацію та авторизацію безпоседньо користувачам при доступі до веб-застосунків. Проведено аналіз ефективності розробленої моделі, показано її переваги у порівнянні з існуючими рішеннями.

Annotation

The authentication process, its nature, and utilization are studied in the given thesis. The analysis of multi-factor authentication, one of the most relevant and widespread types of authentication, is presented. Modern technologies and updated authentication mechanisms for problem-solving, that are required for the proper work of web applications, are explored.

Special attention is paid to the development and modeling of our multi-factor authentication service, with the help of which users, entering the web application, will be ensured by reliable authentication and authorization. The analysis of the developed model’s efficiency and its advantages in comparison to current models are presented in the given thesis.

ОПИС АЛЬБОМУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Додаток
	А4		Завдання на дипломний проєкт	2	
	А4	ІАЛЦ.467200.001 ВП	Відомість проєкту	2	
	А4	ІАЛЦ.467200.002 ТЗ	Технічне завдання	4	
	А4	ІАЛЦ.467200.003 ПЗ	Пояснювальна записка	61	
	А4	ІАЛЦ.467200.006 Д3	Функціональна схема	1	
	А4	ІАЛЦ.467200.005 Д2	Структурна схема	1	
	А4	ІАЛЦ.467200.004 Д1	Принципова схема	1	
	А4	ІАЛЦ.467200.007 Д4	Лістинг програми	10	

Зм.	Арк.	№ докум.	Підпис	Дата	*ІАЛЦ.467200.001 ОА*		
Розроб.		Повх М. І.			*Сервіс багатофакторної автентифікації для веб-застосунків* Опис альбому	Лит.	Арк 1 / Аркушів 1
Перев.		Роковий О. П.					
Реценз.						НТУУ «КПІ», ФІОТ, ІВ-71	
Н. Контр.		Сімоненко В. П.					
Затверд.		Стіренко С. Г.					

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи
освітньо-кваліфікаційного рівня <u>бакалавр</u>

на тему: «Сервіс багатофакторної автентифікації для веб-застосунків»

Київ – 2021

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку програмного забезпечення для потреб розробників веб-застосунків. Область застосування: веб-застосунки, котрим потрібно захистити частину свого функціоналу.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка сервісу багатофакторної автентифікації для веб-застосунків, його аналіз та тестування роботи сервісу.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики розробки вебсервісів, проєктування програмного забезпечення, наукові статті, публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до продукту, що розробляється

Система, що розробляється, повинна:

1. Реалізовувати процес багатофакторної автентифікації.
2. Бути відмовостійкою.

3. Мати можливість до подальшого розширення та доповнення.

5.2. Вимоги до програмного забезпечення

1. Установлене середовище виконання .NET Core 3.1

5.3. Вимоги до апаратної частини

1. Комп’ютер на базі процесору Intel Core i3 і вище
2. Оперативної пам’яті не менше 2 Гбайт
3. Вільного місця на жорсткому диску не менше 1 Гбайт
4. Стабільне підключення до Інтернету

6. ЕТАПИ РОЗРОБКИ

	Дата
Затвердження теми роботи	15.12.2020
Аналіз завдання та огляд існуючих рішень	15.03.2021-5.04.2021
Розробка архітектури та загальної структури системи	5.04.2021-10.04.2021
Програмна реалізація системи	10.04.2021-1.05.2021
Доопрацювання і налагодження системи	1.05.2021-6.05.2021
Оформлення пояснювальної записки	2.05.2021-20.05.2021
Передзахист та проходження нормативного контролю	29.05.2021
Захист дипломного проєкту	14.06.2021

Пояснювальна записка

до дипломного проєкту

на тему: «Сервіс багатофакторної автентифікації для веб-застосунків»

Київ – 2021

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

SFA	однофакторна автентифікація (англ. Single-factor authentication)
MFA	багатофакторна автентифікація (англ. Multi-factor authentication)
TFA(2FA)	двофакторна автентифікація (англ. Two-factor authentication)
PIN	персональний ідентифікаційний номер (англ. Personal identification number)
OTP	одноразовий пароль (англ. A one-time password)
HMAC	хеш-код автентифікації повідомлень (англ. Hash-based message authentication code)
HOTP	одноразовий пароль з використанням HMAC (англ. HMAC-based one-time password)
TOTP	тимчасовий одноразовий пароль (англ. Time-based One-Time Password)
ID	ідентифікатор (англ. Identifier)
API	прикладний програмний інтерфейс (англ. Application Programming Interface)
JWT	токен доступу на основі JSON (англ. JSON Web Token)
ORM	об’єктно-реляційне відображення (англ. Object-relational mapping)

ВСТУП

На сьогодні у зв’язку зі значним зростанням та розповсюдженням веб-застосунків чи веб-додатків у всіх сферах людської діяльності одним з найважливіших завдань таких систем є забезпечення конфіденційності, надійності та ефективності роботи в ході їх експлуатації. В сучасному світі інформація вважається найбільш цінним ресурсом, тому будь-яка приватна особа чи компанія потребує захисту своїх даних. Перевірка автентичності при доступі до електронних ресурсів чи веб-застосунків зараз вважається одним з основних та найпоширеніших заходів щодо забезпечення надійної автентифікації користувача.

Через безперервний розвиток обчислювальних мереж, а особливо мережі Інтернет, відбувається стрімке збільшення кількості веб-застосунків, що використовуються людиною для необхідних завдань. Безперечно, дані системи мають велику кількість переваг, але так чи інакше існують і недоліки. Веб-додакти залишатимуться завжди вразливими до витоку особистих даних, адже активний розвиток обчислювальної техніки дозволяє з часом обходити алгоритми автентифікації. Разом з тим веб-сервіс має бути зосередженим і на ефективності профільного функціоналу, щоб виграти конкуренцію на ринку, яка сьогодні ведеться постійно. Саме тому, одним з актуальних рішень для забезпечення надійної та безпечної автентифікації для веб-застосунків різного призначення є використання сучасних додаткових сервісів автентифікації, які унеможливлюють несанкційований вхід до інформаційної системи та заощаджують як матеріальні блага компанії, так і ресурси, необхідні для найефективнішої роботи застосунку.

Тривалий час сервіси автентифікації використовували логін і пароль як один з основних факторів авторизації. Проте завжди потрібно враховувати людський фактор. Зазвичай користувач послуговується одним паролем для всіх веб-додатків, не змінює його часу від часу. Цей фактор і те, що пароль потрібно зберігати локально, в хмарі чи інших носіях інформації робить

вразливішою систему до зовнішніх загроз, несанкційованого доступу до конфіденційної інформації.

Останнім часом все частіше застосовується багатофакторна автентифікація. Для її реалізації спільно використовуються два або більше факторів автентифікації, що сильно підвищує захищеність системи.

Для підвищення ефективності сервісів багатофакторної автентифікації постає питання дослідження існуючих та розробка нових систем, які забезпечують надійну автентифікацію, оскільки через високу складність структур сучасних мереж та сервісів завдання автентифікації користувача часто вирішується не в повному обсязі або навпаки має занадто високу складність.

РОЗДІЛ 1
ОГЛЯД НАЯВНИХ СЕРВІСІВ БАГАТОФАКТОРНОЇ АВТЕНТИФІКАЦІЇ

1. Опис предметної області

Автентифікація - це процедура встановлення належності користувачеві інформації в системі його ідентифікатора [1]. Це підтвердження того, що особа є тим, за кого вона себе видає, тобто це процес перевірки особистості користувача. В аналоговій формі це виглядає як письмовий підпис, номер соціального страхування чи паспорт.

Основним завданням автентифікації є перевірка ідентифікації, котра забезпечує контроль доступу для систем, щоб перевірити, чи облікові дані користувача збігаються з обліковим записом в базі даних вже авторизованих користувачів або на сервері автентифікації даних.

Зазвичай, користувачі ідентифікуються з ідентифікатором, а автентифікація виконується, під час надання користувачем облікових даних, наприклад паролю, котрий відповідає цьому ідентифікатору користувача. Переважна більшість послуговувачів веб-затосунків найбільш знайомі саме з автентифікацією за паролем (чи пін-кодом), що, як частина інформації, повинна бути відома тільки користувачеві. Таку автентифікацію ще часто називають слабкою або однофакторною (SFA). Та існують й інші методи та фактори, за котрими можне автентифікувати людину.

Ще до появи комп'ютерів використовувалися різні відмінні риси суб'єкта, його характеристики. Зараз використання тієї чи іншої характеристики в системі залежить від необхідної надійності, захищеності та вартості впровадження. Виділяють 3 фактори автентифікації [2-3]:

- Фактор знання, щось, що знаємо тільки ми — це, зазвичай, пароль. Паролем може бути окреме мовне слово, комбінація для замку, текстове слово або особистий ідентифікаційний номер (PIN). Це таємна інформація, котрою має володіти тільки авторизований суб'єкт. Парольний механізм, зазвичай, може бути досить легко втілений і повинен мати низьку вартість. Але існують певні недоліки: часто пароль зберегти у таємниці буває досить складно, адже постійно вигадуються нові види злому, крадіжки чи підбору паролю. Це робить парольний механізм слабозахищеним. Багато секретних питань, таких як «Де ти народився?», «Дівоча прізвище матері» є поганими прикладами фактора знань, оскільки вони можуть бути відомі широкій групі людей, або їх можна дослідити.
- Фактор володіння, щось, що є лише в користувача — пристрій автентифікації. В даному випадку важлива сама обставина володіння суб'єктом якимось особливим предметом: ключем від замка, печаткою. Дані часто вбудовуються в особливий інструмент автентифікації - смарт чи SIM-картку. Для зловмисника роздобути такий пристрій стає складніше, а ніж дістати пароль, а в разі крадіжки пристрою суб'єкт може відразу про це повідомити. Цей метод автентифікації є більш захищеним в порівнянні з парольним механізм, але безперечно він є дорожчим.
- Фактор властивості, щось, що є частиною нас — біометричні дані. Характеристикою є фізична особливість суб'єкта. Це можуть бути риси нашого обличчя, відбиток пальця або долоні, голос або особливість сітчатки чи райдужки ока. Для користувача, дане рішення є найлегшим: не потрібно запам'ятовувати пароль і пристрій автентифікації не треба тримати поруч. Однак сенсори, котрі підтверджуть біометрію користувача мають бути достатньо високочутливими, для того щоб блокувати зловмисників, котрі

схожі з користувачем за біометричними параметрами. Разом з тим для користувача важлива швидкодія застосунку. Вартість такої системи досить велика. Та, незважаючи на ряд недоліків, біометрика залишається досить перспективним фактором.

Автентифікація, що відбувається з використанням двох чи більше факторів називається багатофактороною або посиленою (далі MFA).

Використання кількох факторів автентифікації для підтвердження своєї особи базується на передумові, що несанкційований користувач навряд чи зможе надати фактори, необхідні для доступу. Якщо під час спроби автентифікації хоча б один із компонентів відсутній або поданий неправильно, ідентифікація користувача не встановлюється з достатньою достовірністю, і доступ до об’єкта, котрий захищається багатофакторною автентифікацією, тоді залишається заблокованим.

Суворою автентифікацією називають процес автентифікації, під час котрого використовується інформація без розкриття цієї інформації. Як правило, реалізовується за допомогою асиметричних криптографічних алгоритмів.

Сервіс багатофакторної автентифікації - це ефективний механізм контролю доступу уповноважених осіб до цінних ресурсів, таких як програмне забезпечення, мережевий сервер або обчислювальний пристрій. У цьому механізмі користувач повинен представити більше одного облікового запису для автентифікації. Ці облікові дані або фактори автентифікації змінюються залежно від потреб програми в безпеці. Користувач повинен послідовно подавати до системи безпеки кілька факторів автентифікації. Ці фактори повинні бути незалежними, щоб уникнути порушення всієї системи безпеки у випадку компрометації одного з цих облікових даних [4].

1.1. Вразливості однофакторної автентифікації у веб-застосунках

Чому варто використовувати саме два або більше факторів для автентифікації при виконанні запитів, які потребують ще один рівень захисту у веб-застосунках? Далі ми розглянемо основні вразливості однофакорної автентифікації.

Автентифікацію за паролем вважають досить вразливим способом, оскільки пароль часто можливо підібрати, а користувачі схильні використовувати одні й ті самі паролі для різних веб-застосунків або зберігати їх на мобільних пристроях. З'ясувавши пароль, зловмисник має доступ до ваших особистих даних, а ви про це навіть не дізнаєтесь. Крім того, розробники веб-додатків можуть припуститися концептуальних помилок, що спрощують злам облікових записів.

Пропоную список найбільш поширених вразливостей однофакторної автентифікації за паролем:

- Фішинг: надсилання електронного листа з проханням користувача виконати дію, наприклад, змінити пароль. Сайт, на який жертва вводить пароль, зазвичай знаходиться під контролем зловмисника.
- Шкідливе програмне забезпечення: програмне забезпечення, яке виконує недоброзичливі дії щодо пристрою, на якому він встановлений. Часто використовується як реєстратор ключів для отримання облікових даних, введених жертвою в ідентифікатори на веб-сайті.
- «Груба сила»: зловмисники намагаються вгадати пароль жертви, перевіряючи найчастіше використовувані паролі і навіть генеруючи комбінації символів, котрі часто мають криптографічно захищені паролі, які у багатьох випадках можуть бути «зламані», щоб виявити дійсний пароль користувача. Потім ці паролі використовуватимуться для спроб входу в облікові записи жертви.

- Фізичні: паролі, записані на нотатках або спостерігаються, коли жертва вводить їх у свою систему.

Ці явища явно зменшують переваги безпеки, які мають пропонувати паролі. Беручи це до уваги, неважко повірити, що навіть великі компанії можуть бути зламані.

Також часто зустрічаються вразливості веб-застосунків пов’язані саме з розробкою веб-додатків, котрі забезпечують автентифікацію за паролем. Перечислимо найбільш поширені:

- Веб-застосунок не дає послуговувачам можливість змінити пароль або не сповіщає користувачів про зміну пароля.
- Веб-застосунок не використовує надійні хеш-функції для зберігання паролів у базі даних.
- Веб-застосунок не видаляє сесії користувача після недовготривалого періоду неактивності або не дає функцію виходу з сесії [5].
- Веб-застосунок допускає передачу паролів по незахищеному HTTP з'єднанню, або в рядку URL.
- Веб-застосунок дозволяє користувачам створювати прості паролі.
- Веб-застосунок користується ненадійною функцією відновлення пароля, котру можна застосувати для отримання несанкціойованого доступу до чужих облікових записів.
- Веб-застосунок не захищений від можливості перебору паролів (bruteforce attacks).
- Веб-застосунок не вимагає від користувача змінити пароль після першої авторизації, коли він сам генерує і поширює паролі користувачам (тобто поточний пароль десь записаний).
- Веб-застосунок вразливий для session fixation-атак.
- Веб-застосунок не вимагає повторної автентифікації користувача для важливих дій: зміна пароля чи адреси доставки товарів ...

1.2. Огляд наявних програмних рішень

1.2.1. Duo Mobile

Додаток Duo Mobile розроблений як для операційної системи iOS, так і для Android. Його відмінною рисою є те, що даний сервіс може працювати повністю в режимі офлайн, і тому він застосовується навіть там, де немає стільникового зв'язку або доступу до мережі Інтернет.

За наявності інтернет-з'єднання Duo Mobile пропонує також автентифікацію за допомогою push-повідомлень. При кожному заході на сервер користувач негайно отримує запит, який він може або схвалити, або заборонити одним натисканням. Push-повідомлення були додані в прагненні зробити процедуру багатофакторної автентифікації максимально простою: замість того, щоб відкривати додаток для генерації одноразових паролів, шукати код до потрібного сервісу, а потім вписувати 6 цифр у вікно введення, користувачу потрібно тільки розблокувати смартфон і натиснути на кнопку «Approve». Крім того, додаток Duo Mobile дозволяє отримувати push-повідомлення і на смарт-годинник. Таке рішення є дуже актульним в наш час.

Duo Security пропонує спеціальний тарифний план «Personal». Він абсолютно безкоштовний, але кількість користувачів не повинна перевищувати 10-ти. Підтримуються додавання необмеженого числа інтеграцій та всі доступні методи автентифікації [6].

Переваги:

- Робота в режимі офлайн
- Push-повідомлення

Недоліки:

- Малий акцент на безпеці самого додатку
- Відсутність локалізації мовами, відмінними від англійської.
- Відсутня локалізація
- Висока вартість даного технологічного рішення.

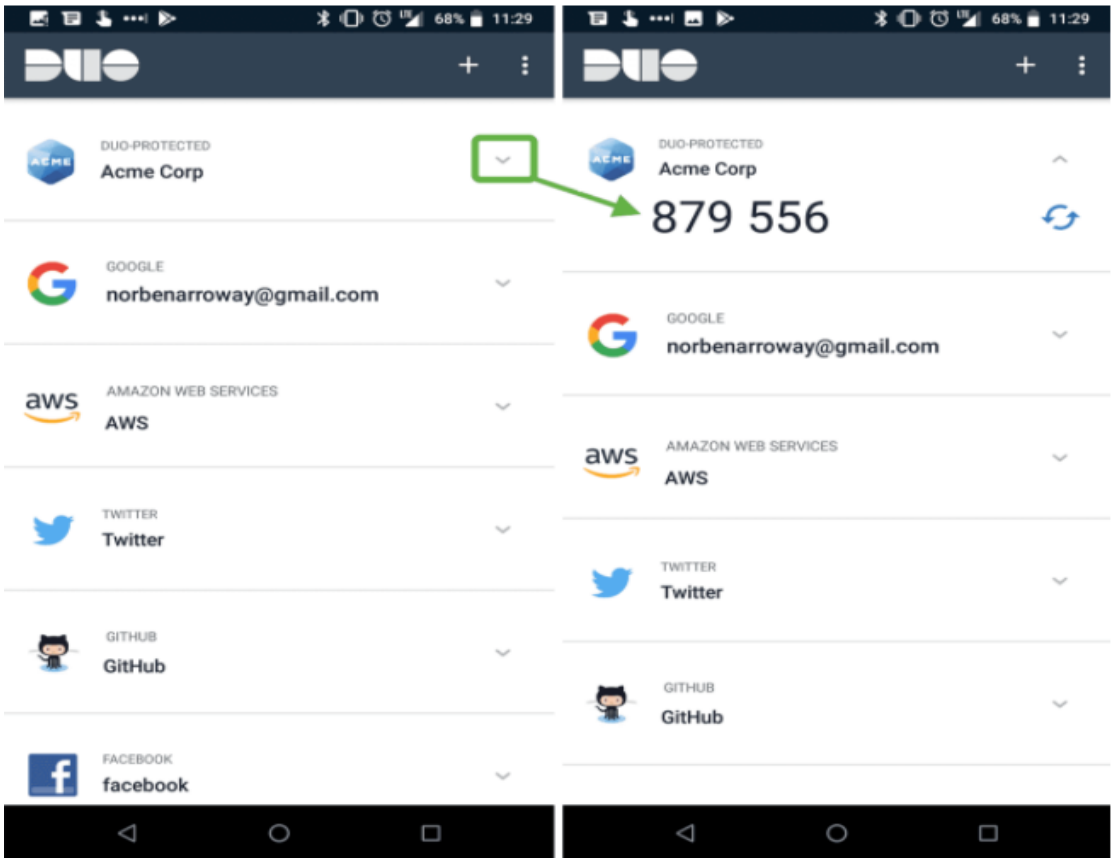


Рис. 1.1. Інтерфейс Duo Mobile.

1.2.2. ESET Secure Authentication

ESET Secure Authentication (далі ESA) є повністю самостійним програмним продуктом, призначеним для корпоративного використання, і представляє собою набір окремих модулів, котрі побудовані на клієнт-серверній архітектурі. За допомогою API двофакторна автентифікація через ESA впроваджується в будь-які інформаційні системи, такі як веб-портали, CRM, різні бухгалтерські програмні комплекси і т. д. А SDK дозволяє вбудовувати 2FA в власне програмне забезпечення.

Клієнтська частина ESET Secure Authentication встановлюється на мобільні пристрої під управлінням популярних операційних систем, приймаючи push-повідомлення та генеруючи одноразові паролі доступу [7].

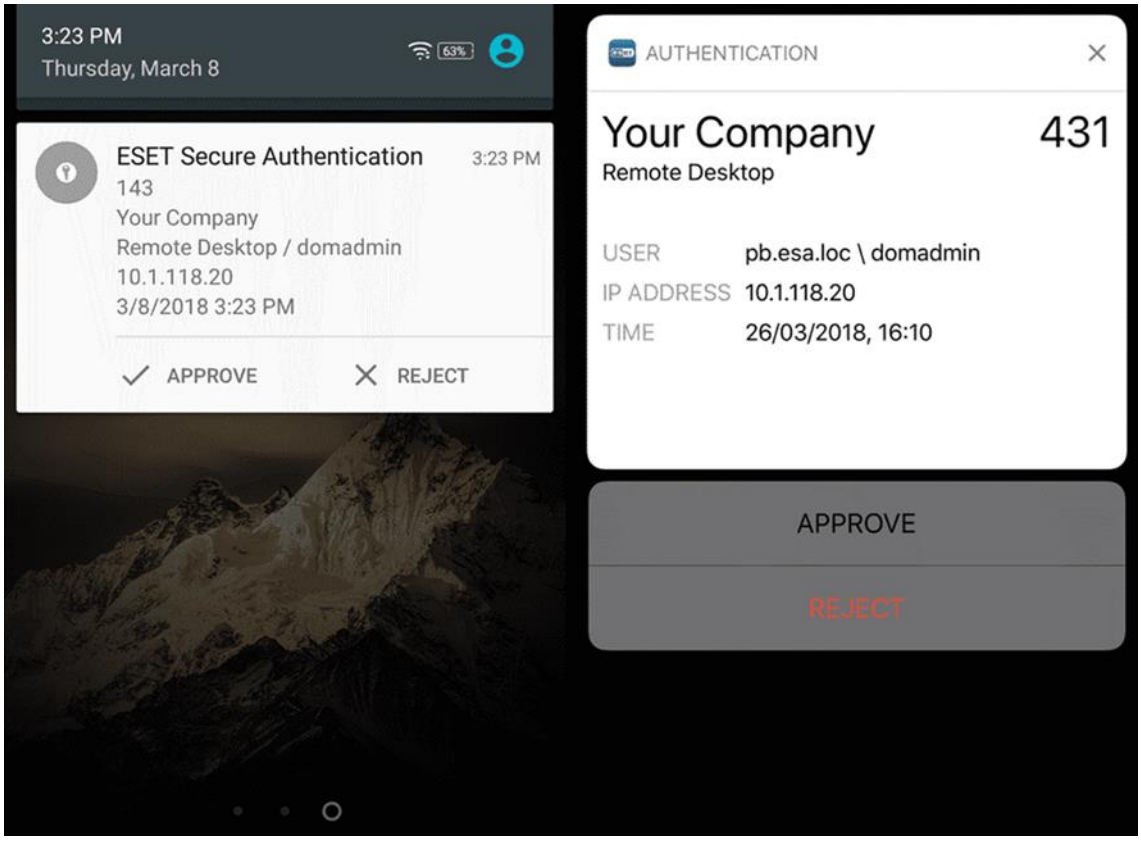


Рис. 1.2. Приклад push-повідомлень від ESET.

Переваги:

- Push-повідомлення
- Генерація одноразових паролів
- Висока безпека додатку

Недоліки:

- Відсутність шифрування
- Досить громіздке та корпоративне рішення.
- Важко інтегрувати у власну систему через відсутність SDK для більшості мов програмування
- Відсутність локалізації
- Досить дорогі пакети надання послуг, порівняно з існуючими сервісами на ринку.

1.2.3. Multifactor

Додаток створено для роботи з системою багатофакторної автентифікації multifactor.ru. Доступний для Android і iOS. Розроблялося в першу чергу з наголосом на зручність застосування для кінцевих користувачів. Вигідною перевагою є можливість роботи безпосередньо через додаток Telegram, в якому формуються push-повідомлення. При цьому немає необхідності завантажувати окремий додаток для отримання другого фактора.

Процес автентифікації при використанні програми Multifactor виглядає наступним чином: користувач при вході на підключений ресурс отримує push-повідомлення на свій пристрій з двома командами - «Підтвердити» і «Відхилити»; якщо немає підключення до інтернету, то додаток генерує одноразовий код [8].

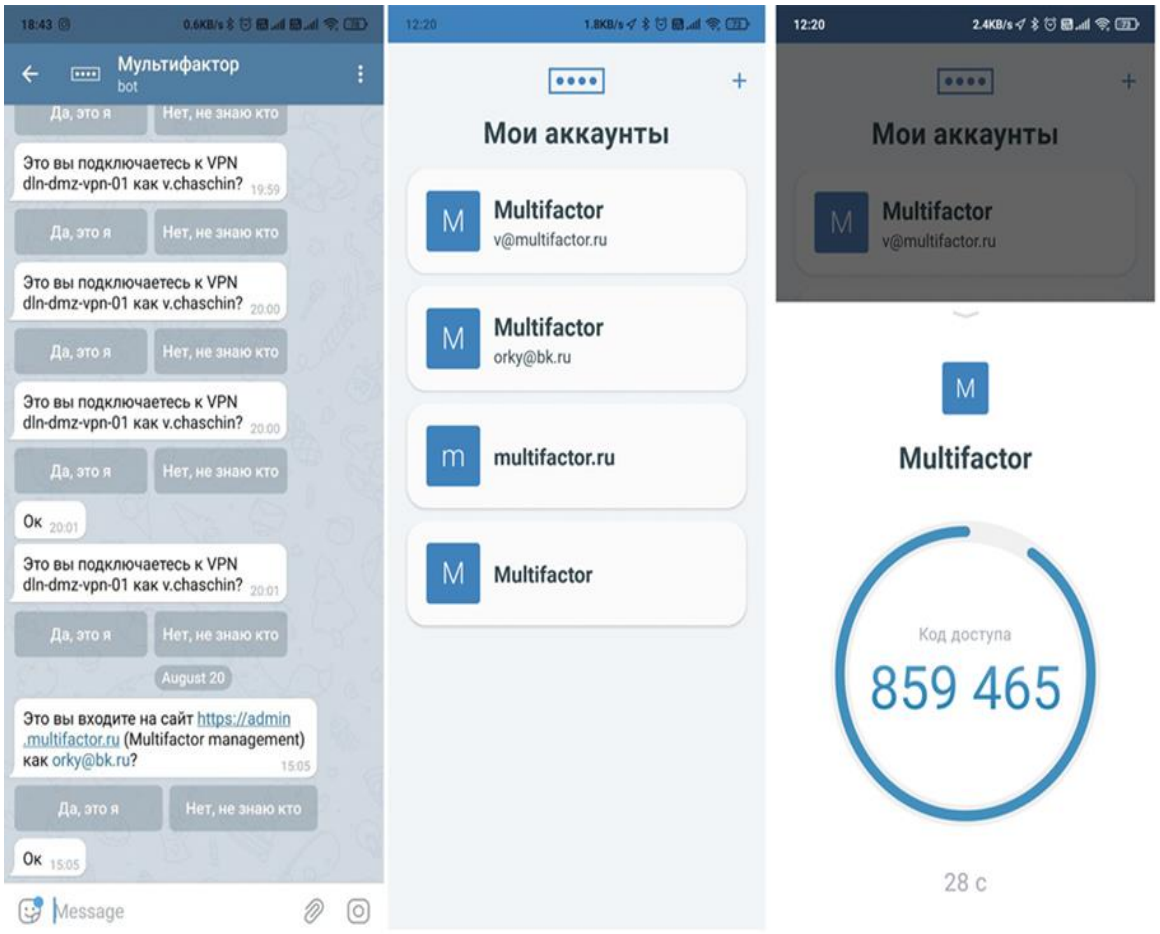


Рис. 1.3 Інтерфейс застосунку Multifactor.

1.3. Вимоги до проєктованого продукту

Беручи до уваги предметну область та аналіз існуючих програмних рішень, можна зазначити необхідні мінімальні вимоги, якими має відповідати сервіс багатофакторної автентифікації для веб-застосунків різного призначення.

По-перше, реалізована система повинна працювати швидко і мати простий і зрозумілий інтерфейс, який має бути інтуїтивним та таким, що до нього швидко звикаєш.

По-друге, сервіс багатофакторної автентифікації має забезпечувати перевірку користувача, щонайменше за двома факторами. Цими факторами будуть фактор знань та фактор власності. Для цього добре підійде мобільний пристрій користувача. Фактор біометрії, звичайно, також має багато переваг, але він значно важчий та дорожчий в реалізації. Тому такі фактори, як розпізнавання голосу, рис обличчя чи відбитків пальців не підходять для проєктованого сервісу. Також це можна пояснити тим, що не в кожного користувача на мобільному пристрої присутні достатньочутливі сенсори (або взагалі відсутні), котрі б забезпечували швидку та ефективну автентифікацію користувача.

Сервіс розроблюється переважно для простих веб-додатків, котрим потрібно захистити лише частину функціоналу. Для таких додатків існуючі рішення будуть занадто громіздкими та дорогими, адже в не в кожному з них присутній інструментарій «на вимогу».

В якості першого фактора автентифікації сервіс буде використовувати звичайний пароль, який буде зберігатись в базі даних в захешованому вигляді. Цей фактор веб-застосунки будуть використовувати для доступу до публічних даних з використання JWT-токену.

Аналізуючи схожі існуючі програмні рішення, я звернув увагу на те, що технологія одноразових паролів в якості другого фактора є одним з

найдешевших та найшвидших методів автентифікації. Саме тому дана технологія буде використана в розробленій моделі.

Для того, щоб створити економічно ефективне рішення на кожному етапі якого не потрібно додавати нові витрати, як, наприклад, пересилка SMS-повідомлень, було вибрано рішення генерації тимчасових одноразових паролів безпосередньо на мобільному пристрої. Для того, щоб уникнути зайвих витрат ми використаємо безоплатний мобільний додаток для генерування тимчасових одноразових паролів. Для надсилання push-повідомлень нам потрібно створювати власний додаток, що є досить часозатратним та дорогим рішенням. Так як, SMS відправляються по мережі GMS в текстовому форматі, їх можливо перехватити, що також є суттєвим недоліком такого методу.

Також в даній роботі ми створимо демонстративний веб-застосунок, як приклад - систему переказу коштів, котра буде використовувати двофакторну автентифікацію лише для деяких чутливих запитів API, іншими словами, для вразливих функцій, як виконання транзакцій чи доступ до персональних даних.

ВИСНОВКИ ДО РОЗДІЛУ 1

Насьогодні автентифікація є необхідною процедурою для забезпечення цілісності, конфіденційності та доступності інформації користувачів будь-яких систем чи ресурсів. Вона використовується часто і всюди для здійснення та обмеження контролю доступу, як фізичного до об'єкту, так і до інформації в веб-застосунку.

Розглянуто основні поняття автентифікації, її типи, сфери застосування. Показано проблеми та вразливості однофакторної автентифікації для веб-застосунків. Виокремлено переваги багатофакторної автентифікації.

Також в першому розділі було проаналізовано існуючі популярні рішення і визначено мінімальні необхідні складові, які повинен включати відповідний сервіс.

При огляді існуючих вже рішень було з'ясовано, що більшість із них є складними та дороговартісними для користувача системами, орієнтованими на профільовані ринки. Майже всі вони доволі комплексні, бо вирішують не тільки проблему автентифікації, а й загалом надають рішення з безпеки на кожному етапі використання веб-застосунку. Останній функціонал для веб-додатку не є зайвим, проте можливо сильно ускладнити співпрацю з сервісом та використовувати його не повною мірою – виникає ситуація, в котрій невелика компанія деколи переплачує за непотрібний їй функціонал.

Підсумовуючи вищесказане, спробую розробити сервіс, який матиме основні функції аналогів та не матиме надлишкового функціоналу.

Визначені мінімальні вимоги до сервісу, що дозволить краще поставити технічне завдання для його проєктування.

РОЗДІЛ 2
АНАЛІЗ ЗАСОБІВ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

2.1. Архітектура сервісу

Найважливішим і найважчим етапом в роботі над розробкою сервісу є його проєктування, тому велика увага приділяється вибору архітектури додатку.

Архітектура програмного забезпечення — спосіб структурування програмної або обчислювальної системи [9], абстракція елементів системи на певній фазі її роботи. Система може складатись з кількох рівнів абстракції і мати багато фаз роботи, кожна з яких може мати окрему архітектуру [10].

Варто виділити основні, на мій погляд, критерії хорошої архітектури: ефективність, гнучкість, масштабованість.

Гнучкість системи. З часом будь-який додаток доводиться оновлювати. Чим швидше і зручніше можна внести зміни в існуючий функціонал, тим менше проблем і помилок це викличе. Зміна одного фрагмента системи не повинна впливати на її інші компоненти. Такий застосунок є більш гнучким і конкурентоздатним. По можливості, архітектурні рішення не повинні зупиняти роботу системи, щоб додати до неї новий функціонал.

Ефективність системи. Оскільки програма повинна вирішувати поставлені завдання і добре функціонувати в різних умовах. Сюди можна віднести такі характеристики, як безпеку, продуктивність, надійність, здатність витримувати збільшені навантаження...

Можливість розширення системи - здатність системи до розширення новим функціоналом чи сутностями, не порушуючи при цьому власну структуру. Одним словом необхідно керуватися пнинципом YAGNI (you is not gonna need it). Розпочинаючи роботу над архітектурою нової системи, закладайте в систему лише найнеобхідніший функціонал, але залишайте

простір для нарощування додаткових можливостей у майбутньому. Бажано зробити це так, щоб внесення змін вимагало найменших зусиль.

Вимога, до гнучкості та розширюваності архітектури системи є настільки важливою, що навіть сформульовано окремий принцип дизайну архітектури - «Принципу відкритості/закритості» (Open-Closed Principle – один з принципів SOLID) : програмні сутності (класи, модулі, функції) повинні бути відкритими для розширення, але закритими для модифікації [11].

Підсумовуючи вище сказане, має бути можливість розширити або змінити поведінку системи без зміни вже існуючих частин самої системи. Це означає, що додаток слід проектувати так, щоб зміна його поведінки і додавання нового функціоналу були б досягнуті за допомогою написання нового коду (розширення), і при цьому не доводилося б змінювати вже існуючий код. Тоді поява нових вимог не спричинить за собою модифікацію існуючої логіки, а зможе бути реалізована перш за все за рахунок її розширення.

При проектуванні даного сервісу було вирішено використати класичну тришарову архітектуру. Це одна із найбільш розповсюджених видів архітектури. Основні компоненти даного рішення можна побачити на рис.2.1.

Тришарова архітектура складається із наступних рівнів:

– рівень представлення - це той рівень, з яким безпосередньо взаємодіє користувач. Цей рівень включає компоненти користувацького інтерфейсу, механізми отримання вхідних даних від користувача. В даній розробленій системі на даному рівні знаходитиметься демонстративний односторінковий веб-застосунок побудований з допомогою одного з фреймворків JavaScript.

– рівень бізнес-логіки використовує ряд компонентів, які відповідають за обробку даних, отриманих від рівня представлення, реалізує всю необхідну логіку додатку чи потрібні обчислення, генерацію одноразових паролів. Також рівень бізнес-логіки призначений для роботи з базою даних та передачі їх вже на рівень представлення

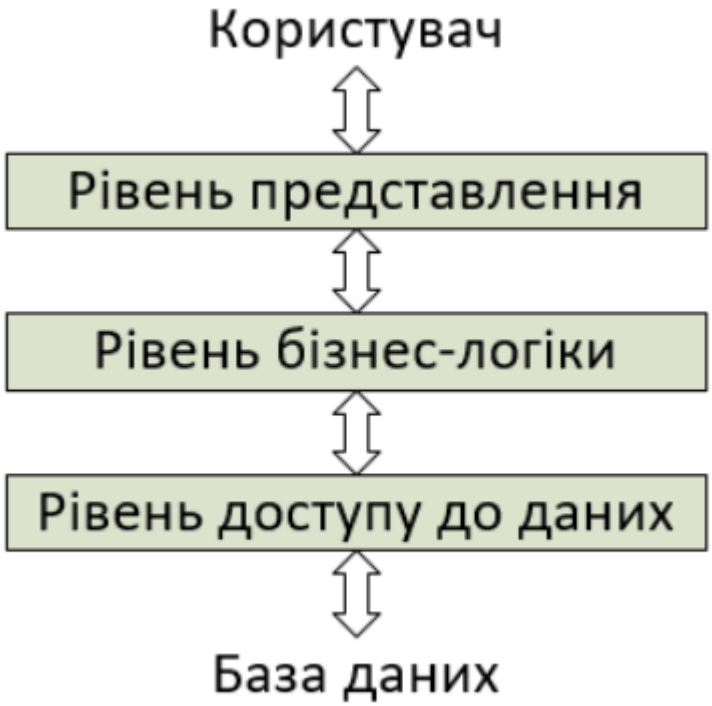


Рисунок 2.1 — Класична тришарова архітектура

– рівень доступу до даних зберігає моделі, що описують сутності, які використовуються. Зазвичай на даному рівні розміщуються специфічні класи для роботи з технологіями доступу до даних, наприклад, клас контексту даних. Тут також можуть бути репозиторії, через які, рівень бізнес-логіки взаємодіє з БД [12].

Слід відмітити, що рівень представлення та рівень доступу до даних не можуть взаємодіяти між собою, тобто рівень представлення не може безпосередньо звертатися до бази даних і навіть до рівня доступу до даних. Щоб це зробити, йому потрібно спочатку звернутись до рівня бізнес логіки, котрий в свою чергу вже звернеться до рівня доступу до даних.

2.2. Технології для реалізації серверної частини

2.2.1. Python

Python – це популярна високорівнева мова програмування, котра призначена для створення різного типу додатків. Це і веб-додатки, і настільні програми, ігри та робота з базами даних.

Доволі велику популярність Python набув у галузі досліджень штучного інтелекту на машинного навчання. Це всесвітньо відома кросплатформенна мультипарадигменна інтерпретована мова програмування з динамічною типізацією. ЇЇ винайшов в 1990 році Гвідо ван Россум. З того часу і до сьогодні вона активно розвивається.

Python проста у використанні, повноцінна мова програмування, що надає набагато більше можливостей у порівнянні з shell для структурування і підтримки великих програм. Але, з іншого боку, вона краще за C обробляє помилки і має вбудовані типи даних високого рівня, будучи мовою дуже високого рівня, такі як гнучкі масиви і словники, ефективна реалізація яких на C потребує чималих витрат часу.

Python - дуже проста мова програмування, що має лаконічний і в той же час досить простий і зрозумілий синтаксис. ЇЇ легко вивчити, оскільки вона підходить для розробки додатків будь-якого виду, і для веб-розробки в тому чилсі.

До мінусів можна віднести технологію GIL (глобальне блокування інтерпретатора), у зв'язку з котрою сильно знижується популярність використання мови при розробці багатопотокових програм. Присутня низька швидкодія, що притаманна для більшості інтерпретованих мов. Хоча мова й підтримує анотації типів, вони не є обов'язковими, що в тривалій перспективі може призвести до зниження швидкості розробки.

Flask — мінімалістичний фреймворк для мови Python для розробки веб-застосунків з REST API, котрий представляє рівень контролера в шаблоні MVC. Присутні розширення для роботи з базою даних, об'єктно-реляційного відображення, перевірки різноманітних видів автентифікації, форм. Шляхи у REST API визначаються за допомогою анотацій. Є підтримка серверного рендерингу веб-сторінок за допомогою HTML-шаблонів [13].

2.2.2. Java

Java - це мова програмування з відкритим кодом, в якої програми компілюються у байткод, який інтерпретується віртуальною машиною для конкретної платформи. Унаслідок цього мова є платформо-незалежною, тобто вона може бути використана практично в будь-якій ситуації завдяки своїй універсальності. Мова об’єктно-орієнтована, як і більшість мов веб-програмування. Існує велика кількість веб-фреймворків та бібліотек, які, як правило, мають досить вичерпну документацію, шо спрощує створення навіть дуже складних проєктів. Програми, написані на Java, легко розширювати, їх можна масштабувати за необхідністю і їх легко підтримувати, за умови, що програміст, який їх писав, знав свою справу. Оскільки Java порівняно важка для вивчення, це призводить до зменшення кількості фахівців з цієї мови програмування, через що, підприємствам доводиться більше витрачатись, ніж при використанні інших рішень. Переважно використовується для веб-розробки, зокрема для складних сервісів з довгим циклом підтримки та розробки, вимогою до яких є надійність, якісна архітектура та платформо-незалежність. Водночас недоліками можна назвати деяку архаїчність мови та повільність застосування там нововведень, об’єктно-орієнтована парадигма програмування, котра не здатна повною мірою застосовувати зручні концепції з мов функціонального програмування, погану швидкодію, порівняно з мовами з автомтичним чи ручним управлінням пам’яттю. Збирання сміття збільшує використання оперативної пам’яті.

Фреймворком Spring можна вважати найпопулярнішим для мови. Він реалізує принцип інверсії управління (англ. Inversion of Control або IoC). Щоб покращити модульність та розширюваність програми, управління залежностями окремих частин даної системи передається фреймворку, Найменшими компонентами є Spring-біни — Java-класи, управління якими передається до Spring Core — модулю для реалізації інверсії контролю, Spring

Boot, котрий відповідає за зв’язування окремих модулів між собою та Spring Security — управління доступами є ключовими модулями.

Java має велике число й інших популярних інструментів для розробки веб-застосунку: Spring Cloud, JDBC, Spring MVC, Hibernate і т.д. Серед переваг можна виокремити популярність, якість підтримки фреймворку, детальність документації. Недоліком є також складність фреймворку, через що значно знижується швидкість розробки. [14].

2.2.3. C#

C# є об’єктно-орієнтованою мовою програмування, котра підтримує поліморфізм, успадкування, перегрузку операторів, статичну типізацію. Це дозволяє вирішувати задачі з побудови великих, але в той же час гнучких, масштабних та легкорозширюваних додатків.

ЇЇ підтримка та вдосконалення ведеться до сих пір приватною корпорацією Microsoft, котра не дає доступ до вихідного коду мови, на відміну від мов з відкритим кодом, котрі підтримуються ентузіазмами, що забезпечує більшу надійність додатків та застосунків створених з її допомогою. ASP .NET Core є найпопулярнішим фреймворком для веб-робробки застосувань мовою C# (.NET). Він кросплатформенний. Даному фреймоворку притаманна неперервна компіляція, що довзоляє розробнику не відволікатись на перезавантаження змін при розробці. Він добре оптимізований для розгортання у хмарному сховищі, наприклад Azure.

Підтримує асинхронність, використовує NuGet пакеми для залежностей, а також чудово інтегрований із середовищами розробки (IDE) Visual Studio та Visual Studio Code. Серед недоліків можна відзначити те, що C# дуже легко розбирається. Це означає, що з великою часткою ймовірності код буде отриманий і вивчений конкурентами. Звичайно ж, є спеціальні інструменти, які можуть ускладнити цей процес, але на 100% захиститися від цього практично неможливо.

2.2.4. JavaScript

JavaScript – це мова програмування, яка уможливлює зробити Web - сторінку інтерактивною. Це означає, що вона реагує на дії користувача.

Спочатку використовувалась тільки для того, щоб надавати вебсторінкам інтерактивності, але після появи фреймовку Node.js став застосовуватись і для побудови масштабованих серверних додаків. Дана технологія швидко набирає популярністі в нових проєктах.

JavaScript є реалізацією стандарту ECMAScript, котрий використовується для стандартизації поведінки вебсторінок між різними браузерами. Мова активно розробляється з 1995 року. Основні архітектурні риси: прототипне програмування, динамічна типізація, функції як об'єкти першого класу, автоматичне керування пам'яттю. Середовищу також притаманна кросплатформеність, можливість багатопотоковості, гнучка система пакетів у NPM (Node Package Manager), нативна підтримка асинхронності (неблокуючий ввод-вивід).

Код дуже просто конфігурувати під свої стандарти розробки, використовуючи для цього, наприклад, prettier та TypeScript — мови, яка компілюється у JavaScript і яка, полегшує процес розробки завдяки таким можливостям, як узагальненому програмуванню чи статичній типізації

Недоліками є низька швидкодія, як і для інших інтерпретованих мов, погана політика з керуванням залежностями, котрі, наприклад, в змозі вирішувати велику кількість різних задач, але слабко контролюються і тому можуть бути малоефективними. [16].

2.2.5. Вибір мови програмування та фреймворку

Після огляду можливих технологій для розробки було вирішено використати мову C# та фреймворк ASP.NET Core для серверної частини у зв’язку з тим, що середовище виконання .NET надає багато корисних класів та послуг, які дозволяють розробникам писати захищений код,

використовувати криптографію. Існує великий вибір інструментів та бібліотек, котрі можна застосувати для програмування необхідного нам рішення. Мова C# є універсальною, вона широко застосовується при розробці будь-яких типів застосунків, у тому числі веб-сервісів. Обрана мова є сучасною та популярною, її застосування є актуальним та корисним для покращення навичок програмування. Проста в опануванні для початківців.

Також вибір був зумовлений більшим досвідом роботи із даною технологією. Мови Java, JavaScript та відповідні їм технології були відкинуті через гірший рівень знайомства з ними, а Python через не комфортну реалізацію підтримки асинхронності.

2.3. Підсистема зберігання даних

2.3.1. Вибір системи керування базою даних

Система керування базою даних (СКБД) — програмне забезпечення, що надає доступ користувачу до бази даних, забезпечуючи управління нею та надаючи інструменти для роботи з даними: операцій CRUD — створення (англ. create), читання (read), оновлення (update) і видалення (delete) [17].

У 1970 році Ф.Кодд запропонував реляційну модель бази даних, яка згодом набула популярності. За його визначенням: «Реляційна БД — це цифрова база даних, заснована на реляційній моделі даних. Дані зберігаються в таблицях, що вміщують рядки — окремі записи, та стовпці — атрибути різного типу даних, які представляють записаний об’єкт» [18]. Для того, щоб покращити зв’язність даних та уникнути повторення, використовуєься нормалізація. Для роботи з даними системами застосовується мова SQL (англ. Structured Query Language). В об’єктних БД інформація представляється у формі об’єктів. Вони були розроблені, щоби можна було легко взаємодіяти з об’єктно-орієнтованими мовами програмування. До переваг реляційних СКБД можна віднести легкость виконання складних запитів та маніпуляцій над даними. При цьому забезпечується узгодженість, ізольованість та атомарність

транзакцій — набору дій, які формують цілісну операцію. Серед недоліків можна виокремити погану масштабованість.

Прикладами найпопулярніших реляційних СКБД є Oracle DB, PostgreSQL, MySQL, MariaDB, Microsoft SQL Server, SQLite. PostgreSQL і Oracle DB є зокрема і об'єктними. Переваги Microsoft SQL Server полягають у вільності використання та кращій взаємодії з іншими продуктами Microsoft, котрі ми будемо використовувати для розробки проєктованої системи.

На початку цього століття почали набирати популярність non SQL (NoSQL) бази даних. Їх використання зростає через кращу масштабованість, швидкість розробки, гнучкість для зберігання слабко структурованих даних. Існує широкий ряд NoSQL баз даних, найбільш поширеними з яких є [19]:

1. Бази даних "ключ-значення" — за кожним унікальним ключем зберігається окреме значення. Застосовуються для даних, операції над якими не потребують складних запитів. Інколи для хешування. Хорошим прикладом цих СКБД є DynamoDB чи Redis.

2. Документні — зберігають дані в документних об'єктах, схожих на XML(Extensible Markup Language) або JSON (JavaScript Object Notation). За ключами зберігаються значення різних типів, зокрема об'єкти більш низької ієрархії. Прикладом таких СКБД є MongoDB.

3. Стовпцеві — зберігають дані у таблицях, динамічних стовпцях чи рядках. Стовпці є набором атрибутів, які містить об'єкт, що зберігається. За допомогою цього такі бази даних є гнучкішими за реляційні. HBase чи Cassandra є наглядним прикладом таких баз даних.

4. Графові — дані зберігаються у вершинах графа, а ребра показують відношення між ними. Зазвичай поширені у машинному навчанні або для відображення відношень між графами відношень тощо.

Порівнявши найпопулярніші системи контролю доступу до даних можна прийти до висновоку, що нереляційні СКБД гірше підходять для реалізації сервісу, через те що для виконання комплексних запитів до розроблюваного

сервісу необхідна чітка зв’язність даних та структурованість даних, котрі зберігатимуться. У випадку використання підходу сесій для підсистеми автентифікації є зміст використовувати базу даних “ключ-значення” [20].

З аналізу підходів вирішено використовувати Microsoft SQL Server у зв’язку з тим, що реляційні СКБД краще підходять для реалізації розроблюваного сервісу, вільністю використання, наявності значної кількості функціоналу та більшим досвідом роботи з даною системою.

2.3.2. Системи об’єктно-реляційного відображення

ORM (Object-Relational Mapping) — це технологія програмування, що поєднує базу даних та концепції об’єктно-орієнтованих мов програмування. Таким чином створюється “віртуальна об’єктна база даних”, яка взаємодіє із сутностями, збереженими в базі даних, як із об’єктами, і які розробник може використовувати на рівні бізнес-логіки [21].

Найпопулярнішим сучасним рішенням об’єктно-релаційного відображення для фреймворку ASP.NET є Entity Framework (EF).

Entity Framework (EF) Core є об'єктно-орієнтованою технологію від компанії Microsoft для доступу до даних. Тобто EF дозволяє працювати з базами даних, але прицьому є більш високим рівнем абстракції: EF дозволяє абстрагуватися від самої бази даних і її таблиць і працювати з даними незважаючи на тип сховища. Якщо на фізичному рівні ми оперуємо індексами, таблицями, зовнішніми і первинними ключами, то на концептуальному рівні, який нам пропонує Entity Framework Core, ми працюємо вже з об'єктами. Відмінною рисою Entity Framework Core є використання технології запитів LINQ для вибірки даних з бази даних. Цей інструмент я активно використовував при розробці для вибірки потрібних мені об'єктів, в тому числі пов'язаних різними асоціативними зв'язками.

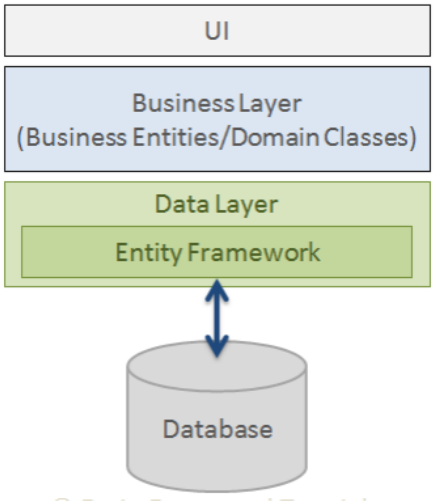


Рис. 2.2. Entity Framework в структурі розробленої моделі.

2.4. Технологія для реалізації клієнтської частини

AngularJS - це набір інструментів для побудови фреймворка, котрий добре підходить для створення веб-додатків. Він лего розширюємий і відмінно взаємодіє з іншими бібліотеками. Будь-яка особливість може бути змінена або замінена відповідно до унікальних потреб застосування.

Для розробки клієнтської частини використовувався веб-фреймворк AngularJS. Він був розроблений у 2009 році Міско Хевері та Адамом Абронсом. В основному підтримується Google та спільнотою приватних осіб та корпорацій для вирішення багатьох проблем, що виникають при розробці односторінкових додатків.

AngularJS - це структурний фреймворк для динамічних веб-додатків. AngularJS дозволяє розширити синтаксис HTML. В результаті код виходить виразним, читабельним, і легко підтримується. Його прив’язка даних та впровадження залежностей усувають значну частину коду, що спрощує написання коду для розробників. І все це відбувається в браузері, що робить його ідеальним партнером для будь-якої серверної технології. [22]

Враховуючи це, було вирішено вибрати AngularJS як основну технологію для розробки клієнтської частини. І так як я маю більший досвідом роботи із даною технологією, альтернативи як React.js та Vue.js були відкинуті через гірший рівень знайомства з ними.

2.5. Генерація OTP

2.5.1. OTP та алгоритми його генерації.

Одноразовий пароль (англ. one time password, OTP) — це пароль, який є дійсним тільки для одного сеансу автентифікації. Його дія також може бути обмежена певним проміжком часу. Перевага такого паролю порівняно статичним полягає в тому, що його неможливо використовувати повторно. Таким чином, зловмисник, що перехопив дані з успішної сесії автентифікації, не може використовувати скопійований пароль для отримання доступу до захищеної інформаційної системи. [23]

Запатентовано велику кількість технологій OTP, що робить стандартизацію в цій області важчою, оскільки кожна корпорація чи компанія хоче проштовхнути свою власну технологію, котра би була найбільш сприятливим для її кінчевого споживача. Разом з тим, це великий плюс, так як, постійно виникають нові алгоритми – технологія розвивається. Стандарти, однак, існують, наприклад, TOTP і HOTP.

HOTP заснований на HMAC (SHA-1), механізмі перевірки цілісності інформації, що передається або зберігається в ненадійному середовищі. Кодом автентичності повідомлення (MAC) називаються такі механізми, котрі перевіряють цілісність об'єкта на основі секретного ключа. Як правило, MAC використовується між двома сторонами, які поділяють секретний ключ для перевірки автентичності інформації, переданої між ними. Цей стандарт визначає MAC. Звідти і назва, HMAC це механім, який використовує секретний ключ в поєднанні з хеш-функцією.

В якості параметра сам факт генерації відповідає за динаміку генерації паролів. Кожного разу при створенні нового пароля лічильник подій(фактів генерації) збільшує на одиницю своє значення, і саме це монотонне зростаюче значення використовується як основний параметр алгоритму. Симетричний ключ, який повинен бути унікальним для кожного користувача, є другим параметром для розрахунку одноразових паролів. Цей ключ має бути закритим для всіх, крім сервера і самого генератора.

Основною відмінністю TOTP алгоритму від HOTP і TOTP є генерація пароля на основі мітки часу, яку TOTP алгоритм використовує в якості параметра. При цьому використовується не точне значення часу, а зазделігідь встановлений інтервал, наприклад 15 чи 30 секунд.

Незалежно від часу лічильника, HOTP генерує ключ на основі розподіленого секрету (Pre-Shared Key).

Завдяки цьому основа для лічильника в HOTP алгоритмі, на відміну від інших алгоритмів, що використовують таймер, захищена від розсинхронізації передавальних пристроїв. Це дозволяє HOTP-паролі залишатися дійсними протягом довготривалого часу. TOTP-паролі через вказаний інтервал стають не дійсними.

Отже, хоча TOTP використовують ту ж хеш-фукнцію що і HOTP, показана відмінність у роботі алгоритму показує, що TOTP є кращим та більш безпечним рішенням для одноразових паролів [24-25].

2.5.2. Google Authenticator

Google Authenticator - це програмний автентифікатор від Google, який надає можливість генерування одноразових паролів для перевірки автентичності, використовуючи алгоритм одноразового пароля на основі часу (TOTP) та алгоритм одноразового пароля на основі HMAC (HOTP)

Мобільний додаток генерує 80-бітний секретний ключ, хоча стандарт RFC 4226 рекомендує використовувати 160 бітів. Для генерації паролю потрібно передати в додаток закодований ключ в форматі base32 із 16, 26 або 32 символів. Є можливість сканування цього ключа у виглялі QR-коду. Застосунок обчислює хеш-значення HMAC-SHA1 за допомогою цього секретного ключа. Частина HMAC зчитується та відображається користувачеві у вигляді шестизначного коду.

Мобільний додаток не має можливості ініціалізації на багатьох пристроях одразу. Реалізований одразу на декількох популярних платформах.

Корпорація Google повністю володіє останніми версіями застосунку, хоча раніше дану технологію можна була легкго дістати в одному з GitHub репозиторіїв [26-27].

Таблиця 2.1. Основні технічні характеристики Google Authenticator

Алгоритми генерації одноразових паролів	TOTP та HOTP
Доступний функціонал	Генерація 6-ти або 8-ми значного одноразового паролю в якості другого фактора
Безкоштовний	Так
Розробник	Google LLC
Операційні системи	Android, BlackBerry, iOS, J2ME
Шифровані резервні копії	Ні
Використання на декількох пристроях	Ні
Розмір (в App Store)	20,4 Мб
Оцінка в App Store	3
Завантажень в Google Play	50,000,000+
Локалізація	Так
Захист від скріншотів	Ні

2.5.3. Microsoft Authenticator

Microsoft Authenticator – це один з найпопулярніших мобільних застосунків, котрі дозволяють генерувати одноразові пароль для зовнішніх сервісів багатофакторної автентифікації. На ринку з червня 2016 р. Працює додаток, як і більшість його аналогів. Доступний на різних платформах.

Microsoft Authenticator генерує шестизначний пароль, який відображається для кожного доданого облікового запису. Пароль дійсний протягом 30 секунд, що запобігає використанню коду кілька разів. Щоб ініціалізувати обліковий запис можна скористатись можливістю сканування QR-коду або ввести коду вручну [28].

Таблиця 2.2. Основні технічні характеристики Microsoft Authenticator

Алгоритми генерації одноразових паролів	TOTP
Доступний функціонал	Розпізнавання рис обличчя, сканер відбитку пальця або введення PIN, замість паролю; генерація 2 - 6-ти або 8-ми значного одноразового паролю
Безкоштовний	Так
Розробник	Microsoft
Операційні системи	Android і iOS.
Шифровані резервні копії	Так
Використання на декількох пристроях	Ні
Розмір (в App Store)	126.6 Мб
Оцінка в App Store	4.9
Кількість завантажень в Google Play	10,000,000+
Локалізація	Так
Захист від скріншотів	Ні

2.5.4. Twillio Authy

Twillio Authy – мобільний застосунок, котрий забезпечує генерацію одноразових паролів на основі алгоритму TOTP. Додаток має змогу працювати офлайн. Twillio Authy генерує шестизначний пароль, котрий дійсний протягом 30 секунд. Ініціалізація облікового запису проходить шляхом сканування QR-коду або введення коду вручну.

Авторизувавшись на сервері Twillio Authy за номером телефону, додаток дає змогу використовувати застосунок одночасно на декількох пристроях, котрі будуть синхронізовані. В разі втрати пристрою, як фактора власності, можна деавторизувати його з будь-якого іншого авторизованого пристрою.

Authy розроблений та доступний для мобільних пристроїв, які працюють під системою IOS, Android чи Windows, та для розумних годинників.

Таблиця 2.3. Основні технічні характеристики Twillio Authy 2FA

Алгоритми генерації одноразових паролів	TOTP
Доступний функціонал	Розпізнавання рис облич, сканер відбитку або введення PIN, замість паролю; генерація 6-ти значного одноразового паролю.
Безкоштовний	Так
Розробник	Authy Inc.
Операційні системи	iOS, Android, MacOS, Windows
Шифровані резервні копії	Так
Використання на декількох пристроях	Так
Розмір (в App Store)	50.6 Мб
Оцінка в App Store	4.9
Кількість завантажень в Google Play	10,000,000+
Локалізація	Так

2.5.5. Вибір мобільного застосунку для генерації OTP

Отже, приведемо аналіз застосунків для генерації OTP. Як ми бачимо усі сервіси мають можливість використання застосунку в якості генераторів одноразових паролів, що побудовані на основі таких криптографічних протоколів як HOTP та TOTP, що є необхідним для побудови проєктованого сервісу багатофакторної автентифікації. В кожному додатку протоколи автентифікації HOTP та TOTP побудовані згідно стандартів. На мою думку найдоцільнішим було б використання застосунку Google Authenticator адже в ньому наявний лише потрібний нам функціонал без зайвих, не потрібних нам функцій.

ВИСНОВКИ ДО РОЗДІЛУ 2

Після аналізу поширених шаблонів проєктування сервісів для веб-застосунків та огляду наявних технологій для розробки веб-додатків було вирішено використовувати багатошарову архітектуру, реалізуючи шаблон проєктування MVC. На рівні представлення використовується AngularJS з використанням бібліотеки Bootstrap. На рівні бізнес-логіки використовується MVC-фреймворк APS.NET Core, який дозволяє створювати API з допомогою шаблону ASP.NET Core Web API. Прийнято рішення використовувати об’єктно-реляційне відображення на рівні доступу до даних, а саме модуль Entity Framework, Microsoft SQL SERVER у якості СКБД.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1. Основні сценарії взаємодії користувача із сервісом

Діаграми послідовності (англ. Sequence Diagram) – це один з різновидів UML-діаграм для відображення певного процесу із задіяними в ньому елементами та сценарію їх роботи. Розгляньмо накреслені схеми, щоб показати основні сценараї взаємодії.

Отже, детально розглянемо перший важливий сценарій. Під час реєстрації користувача (рис. 3.1) початковими вхідними даними є логін та пароль. Потім відбувається валідація введеної інформації на клієнті. Перевіряється чи введений пароль дійсно відповідає формату: довжина та наявність в нім цифр чи великої літери.

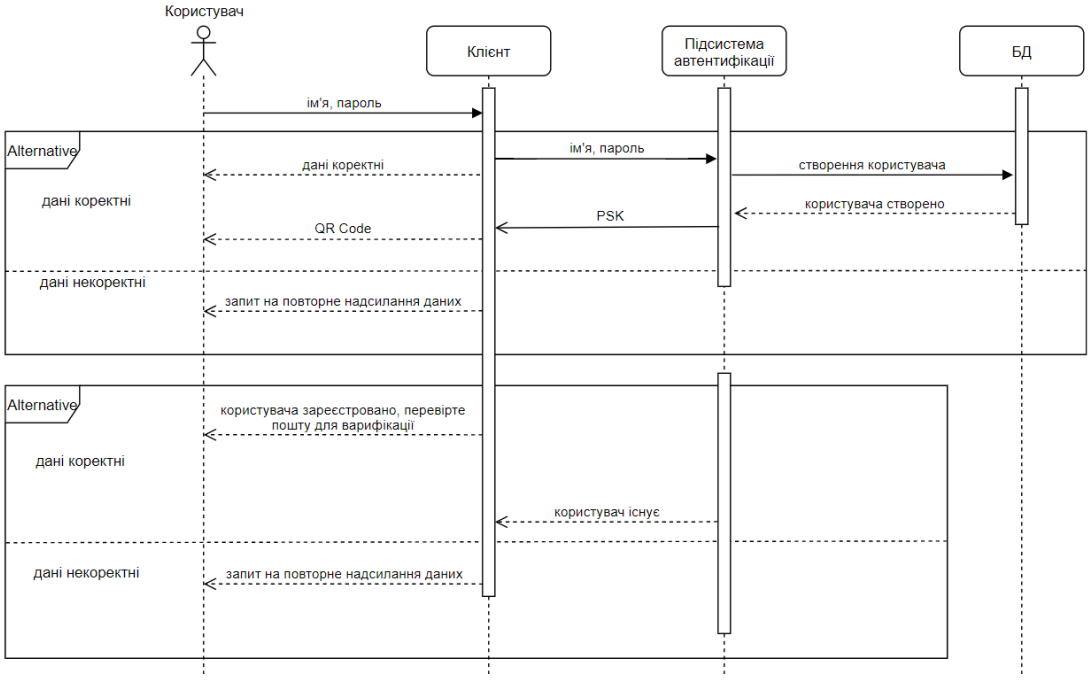


Рис. 3.1. Послідовність процесу реєстрації користувача.

Після того, як клієнт звернеться до сервісу автентифікації, відбувається перевірка наявності користувача у базі даних. Після того, як клієнт звернеться до сервісу автентифікації, відбувається перевірка наявності користувача у базі даних. Якщо введений логін вже існує, то сервіс відправляє запит на повторне

надсилання даних. При його відсутності створюється новий користувач. Після реєстрації користувач може розпочати свою автентифікацію.

При вході в систему (рис. 3.2) користувач надсилає свій логін та пароль. На рівні бізнес-локіки підсистема взаємодії з БД шукає користувача за логіном. Отриманий з клієнта пароль хешується та перевіряється зі збереженим у базі. При невідповідності на клієнт надсилається повідомлення з помилкою. Інакше у підсистемі авторизації для користувача створюється підписаний токен JWT та надсилається у заголовку Authorization з HTTP-відповіддю на клієнт.

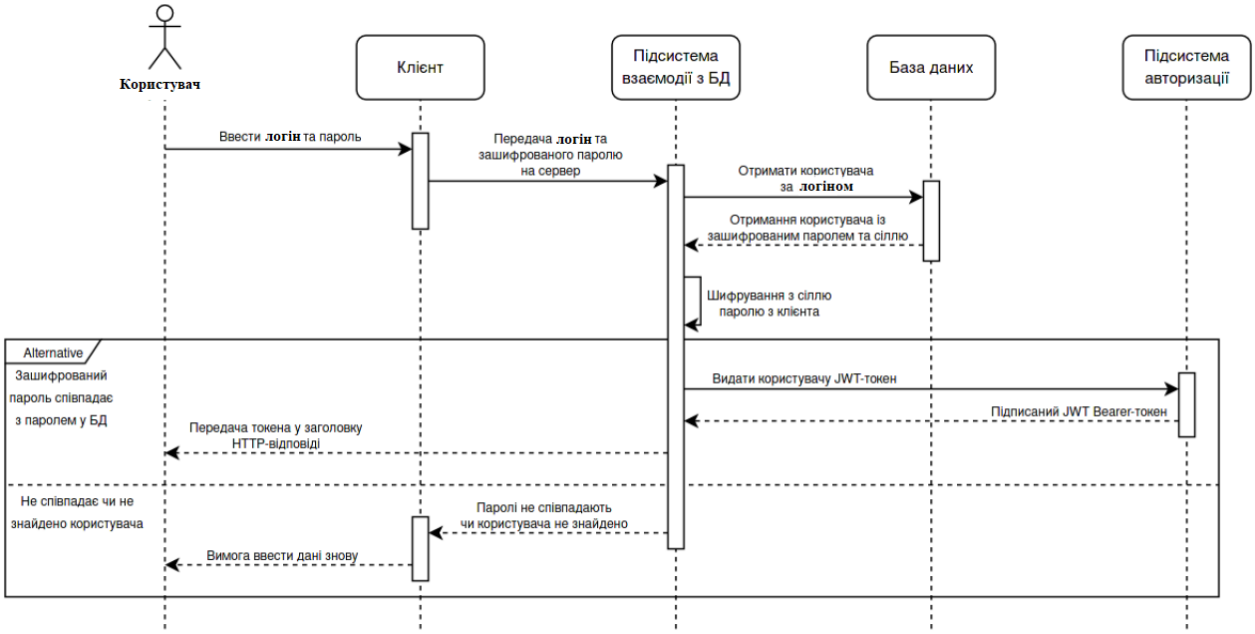


Рис. 3.2. Послідовність процесу автентифікації користувача в систему.

Тепер, коли користувач увійде в нашу систему, надавши своє ім’я користувача та пароль, він отримає токен JWT, який дозволить йому отримати доступ до наших захищених ресурсів API.

Поки він намагається отримати доступ до методів, котрі не захищені багатофакторною автентифікацією, нашому створеному сервісу API буде достатньо лише одного фактора – згенерованого токена (фактор знання).

При доступі до захищених методів, система вимагатиме другий фактор - TOTP, котрий згенерований на іншому пристрої користувача. Мобільний телефон в даному випадку буде фактором власності.

Продемонструємо сценарій, за якого користувач намагатиметься отримати доступ до методів, котрі захищені двома факторами.

Для того, щоб користувач виконав транзакцію (дію, яка захищена двома факторами автентифікації), він має бути вже автентифікованим за першим фактором, тобто увійти в систему за допомогою пароля. Щоб отримати доступ до захищених методів користувач має ввести в поле для вводу одноразовий пароль згенерований на мобільному пристрої. Після чого сервер порівняє цей пароль з тим, котрий він згенерував з PSK цього ж користувача, котрий можна дістати з бази даних за логіном запрошувача. Якщо паролі співпадуть, то користувачу буде надано доступ до виконання транзакцій, якщо ж ні – запит буде відхилено. Сервер поверне помилку HTTP 401, оскільки дана операція захищена і потребує другий фактор власності для успішної автентифікації.

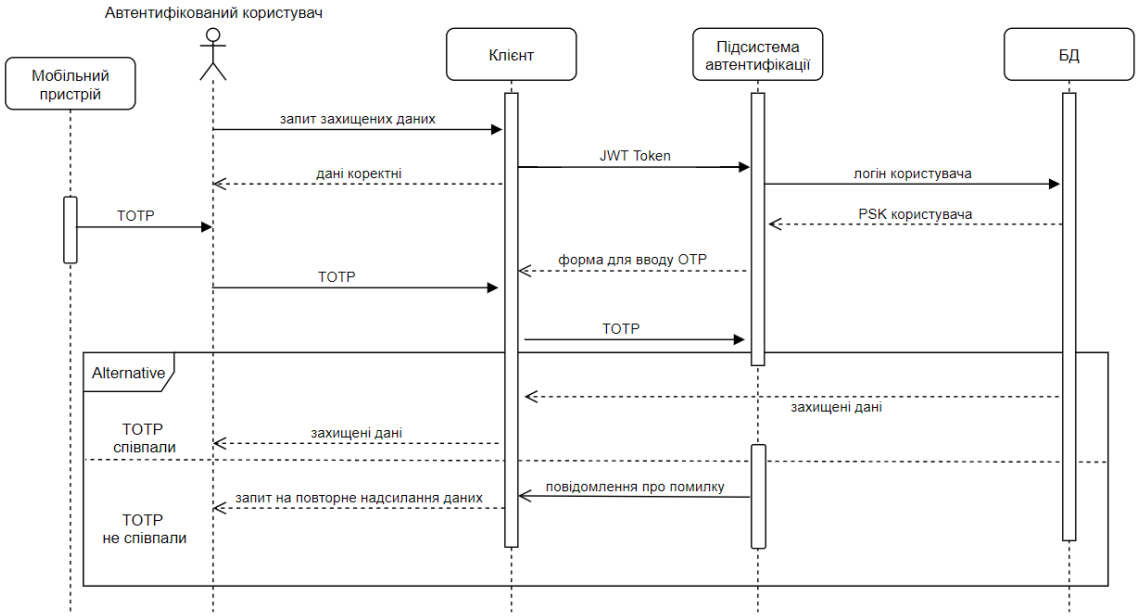


Рис. 3.3. Послідовність процесу запиту до даних котрі захищені двома факторами автентифікації.

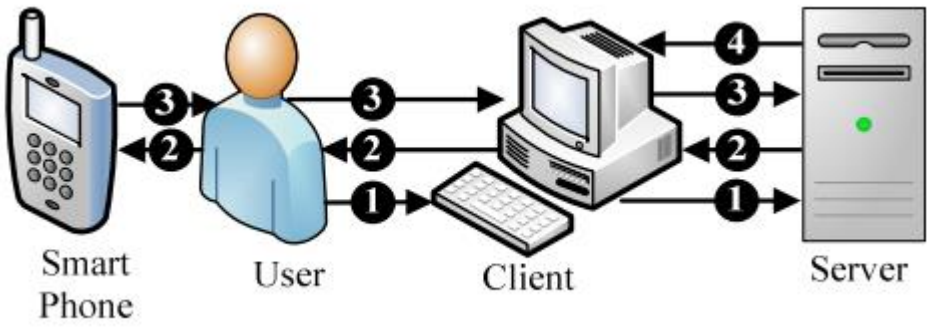


Рис.3.4. Процес надання другого фактору автентифікації.

3.2. Опис бази даних

У SQL Server базі даних було створено таблиці з використанням такої технології як Entity Framework. Цей інструмент суттєво спрощує створення потрібної нам бази даних.

Центральною концепцією Entity Framework є поняття сутності або entity. Сутність визначає набір даних, які пов'язані з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх колекціями.

Для роботи сервісу багатофакторної автентифікації нам достантньо побудувати модель даних, яка складається тільки з одної сутності User, проте для демонстрації роботоспрожності розробленої системи, необхідно також додати сутність Transaction.

Таблиця 3.1. Основні сутності моделі даних

Тип сутності	Опис
User	Представляє користувача.
Transaction	Представляє транзакцію.

В базі даних були створені таблиці, в яких зберігаються необхідні дані, котрі описують дані сутності та допомагають ефективно з ними працювати.

Таблиця Users – зберігає основні дані користувачів.

Таблиця 3.2. Таблиця Users

Ім’я властивості	Опис
Id	Ідентифікатор користувача.
PSK	Preshared key (попередньо-поширений ключ).
PasswordHash	Захешований пароль користувача – в модулі автентифікації використовується алгоритм sha256.
PasswordSalt	“Сіль” для паролю — випадкова рядкова послідовність, яка використовується при хешуванні паролю.
TwoFactorEnabled	Представляє чи користувач застосовує MFA.
AccessFaildCount	Кількість невдалих спроб автентифікації.
UserName	Унікальне ім’я користувача.
Transactions	Транзакції користувача

Таблиця Transactions зберігає історію грошових переказів користувача.

Таблиця 3.3. Таблиця Transactions

Ім’я властивості	Опис
Id	Ідентифікатор транзакції.
UserId	Ідентифікатор користувача.
Amount	Сума грошового переводу.
CreatedDate	Дата та час грошового переводу.

Як видно з опису таблиць, між нижи є єдиний зв'язок: один до багатьох між користувачем та його транзакціями.

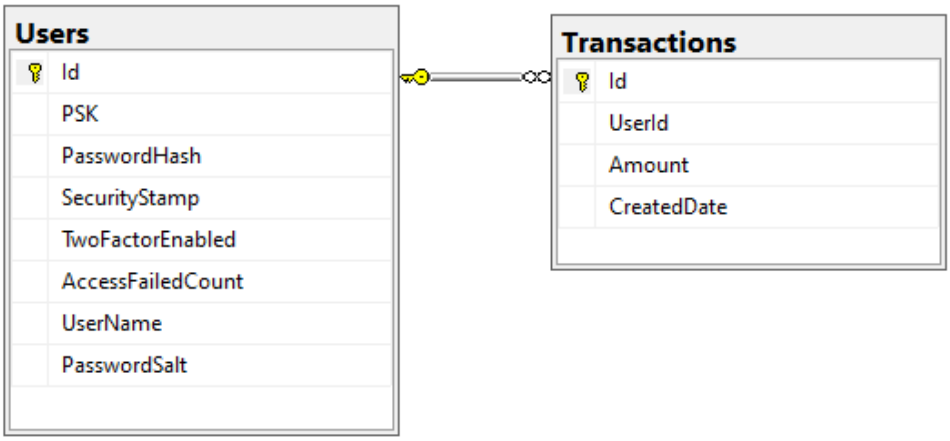


Рис. 3.5. Діаграма бази даних.

3.3. Розробка бізнез-логіки та доступу до даних

На рівні бізнез-логіки використовується ASP.NET фреймворк ASP.NET Core Web API, а на рівні доступу до даних – Entity Framework, котрий потрібно інтегрувати до проєкту. В файлі *Sturtup.cs* описується логіка ініціалізації додатку при запуску. Повний список конфігурації залежностей показиний у файлі *package.config*. Основним файлом конфігурації додатку є *Web.config*, в якому вказані певні чутливі дані, які використовуються в роботі застосунку (такі як адреса підключеної бази даних, логін та пароль доступу до неї чи секретний ключ для підпису JWT-токенів). В теці *Controllers* знаходяться контролери, які приймають ввід користувача і передають зміни до моделі. Сутності бази даних розміщені в теці *Models*. А основна логіка застусунку, як наприклад, генеруваня OTP паролів та опублікованих ключів для шифрування (Pre-shared Keys) описується у файлах, котрі знаходяться в теках *Core*, *Helpers* та *Attributes*.

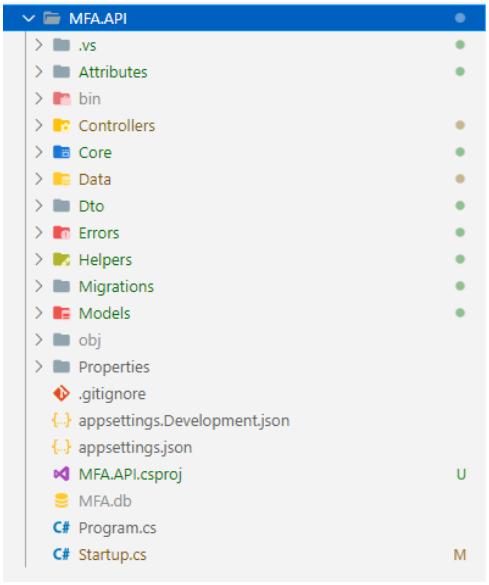


Рис. 3.6. Структура серверної частини проєкту в середовищі Visual Studio Code.

В контролерах (рис. 3.7) обробляється HTTP-запити. Бізнес-логіка в контролері мінімальна. Для захисту шляхів у REST API використовуються захисні атрибути *[Authorize]*, в яких перевіряється, чи є користувач автентифікованим, або навпаки — *[AllowAnonymous]* , котрі навпаки дають доступ навіть неавторизованим користувачам до певного шляху.

```
[HttpPost("login")]
0 references
public async Task<IActionResult> Login(UserForLoginDto userForLoginDto)
{
    var userFromRepo = await _repo.Login(userForLoginDto.Username
        .ToLower(), userForLoginDto.Password);

    if (userFromRepo == null)
        return Unauthorized();

    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config.GetSection("AppSettings:Token").Value));
    string tokenString = TokenService.GetToken(userFromRepo, key);
    return Ok(new
    {
        token = tokenString
    });
}
```

Рис. 3.7. Структура контролера логування.

Клас *DataContext.cs* відповідальний за комунікацію з базою даних. В ньому створюється контекст даних, котрий потім активно використовують для її вибірки. Він представляє таблиці у вигляді колекцій об’єктів.

Хорошим прикладом моделі є клас *User*, котрий представляє об`єкт за допомогою властивостей. Для взаємодії з рівенем представлення використовувались DTO (Data transfer object), котрі описна в теці *Dto* Атрибути анотації даних допомагають валідувати вхідні дані, котрі описують модель запиту реєстрації.

```
using System.Collections.Generic;

namespace MFA.API.Models
{
    10 references
    public class User
    {
        1 reference
        public int Id { get; set; }
        4 references
        public string Username { get; set; }
        2 references
        public byte[] PasswordHash { get; set; }
        2 references
        public byte[] PasswordSalt { get; set; }
        0 references
        public int AccessFailedCount { get; set; }
        1 reference
        public bool TwoFactorEnabled { get; set; }
        3 references
        public string PSK { get; set; }
        0 references
        public virtual ICollection<Transaction> Transactions { get; set; }
    }
}
```

Рис. 3.8. Структура класу моделі.

Детальніше розглянемо методи, котрі описані в класі *OtpGenerationService:*

Статичний метод *GetPSK* відповідає за генурування опоблікованих ключів для шифрування (Pre-Shared Key) з 16-ти символів. В основному це масив з 80-ти бітів, які кодуються в форматі base32. Цей формат використовує 26 великих латинських літер від A до Z та шість цілих цифр від 2 до 7. Цей обмежений набір символів є зручним для використання кінцевим споживачами.

Чому використовується кодування ключа саме у форматі base32? Це пояснюється тим, що мобільний додаток Google Authenticator, котрий використовується для реалізації та демонстрації розробленої системи, теж застосовує цей алгоритм. Цифри, котрі схожі на букви (тобто 0, 1, 8 та 9) пропускаються, адже вони можуть заплутати користувачів. При програмному рішенні застосувалась бібліотека *System.Security.Cryptography*, котра інтегрована в стандартний інструментарій фрейморку ASP.NET Core.

```
public static string GetPSK()
{
    byte[] key = new byte[10];
    using (var rngProvider = new RNGCryptoServiceProvider())
    {
        rngProvider.GetBytes(key);
    }
    return key.ToBase32String();
}
```

Рис. 3.9. Генерація Pre-Shared Key.

Реалізацію кодування base32 можна знайти в методі розширення з назвою *ToBase32String* в допоміжному класі *StringHelper*.

```
private static string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567";

1 reference
public static string ToBase32String(this byte[] secret)
{
    var bits = secret.Select(b => Convert.ToString(b, 2).PadLeft(8, '0')).Aggregate((a, b) => a + b);

    return Enumerable.Range(0, bits.Length / 5).Select(i => alphabet.Substring(Convert.ToInt32(bits.Substring(i * 5, 5), 2), 1)).Aggregate((a, b) => a + b);
}
```

Рис. 3.10. Реалізація кодування base32.

Статичний метод *GetHOTP* приймає закодований Preshared-Key в форматі base32 (16 символів) та лічильник, відповідальний за генерацію одноразових паролів (OTP), а повертає згенерований OTP.

```
private static string GetHOTP(string base32EncodedSecret, long counter)
{
    byte[] message = BitConverter.GetBytes(counter).Reverse().ToArray();
    byte[] secret = base32EncodedSecret.ToByteArray();
    HMACSHA1 hmac = new HMACSHA1(secret, true);

    byte[] hash = hmac.ComputeHash(message);
    int offset = hash[hash.Length - 1] & 0xf;
    int truncatedHash = ((hash[offset] & 0x7f) << 24) |
    ((hash[offset + 1] & 0xff) << 16) |
    ((hash[offset + 2] & 0xff) << 8) |
    (hash[offset + 3] & 0xff);

    int hotp = truncatedHash % 1000000;
    return hotp.ToString().PadLeft(6, '0');
}
```

Рис. 3.11. Метод генерації OTP.

Коли ми розглядали види та відмінності одноразових паролів, було відмічено що реалізація TOTP майже не відрізняється від HOTP. Єдина велика відмінність полягає в тому, що в якості значення лічильника підставляється величина, що залежить від часу.

. Тому в методі *GetOTPs* ми генеруємо список одразу з трьома одноразовими паролями: з попереднім, теперішнім та майбутнім. Причиною цього є можлива різниця в часі між сервером та годинником на мобільному пристрої, де пароль генерується за допомогою Google Authenticator. Цим ми полегшуємо ввід однарозового пароля для користувача з смартфону у веб-застосунок.

Для того, щоб контролювати для котрих методів (endpoint) потрібно застосовувати багатофакторну автентифікацію, було створено власний атрибут анотації даних.

```
namespace MFA.API.Attributes
{
    [AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = true, Inherited = true)]
    1 reference
    public class OtpAuthorizeAttribute : AuthorizeAttribute, IAsyncAuthorizationFilter
    {
        0 references
        public Task OnAuthorizationAsync(AuthorizationFilterContext context)
        {
            var preSharedKey = context.HttpContext.User.FindFirst("PSK").Value;
            bool hasValidTotp = OtpHelper.IsValidTotp(context.HttpContext.Request, preSharedKey);

            if (hasValidTotp)
            {
                return Task.FromResult<object>(null);
            }
            else
            {
                context.Result = new UnauthorizedResult();
                return Task.FromResult<object>(null);
            }
        }
    }
}
```

Рис. 3.12. Власний атрибут анотації даних, створений для того, щоб захистити потрібні шляхи (endpoints).

Цей побудований атрибут авторизації успадковує атрибут *AuthorizeAttribute*, реалізовуючи інтерфейс *IasyncAuthorizationFilter*, та пегегружає його метод *OnAuthorizationAsync*.

Ми можемо бути впевненими, що виконання коду цього метода не розпочнеться, поки користувач не буде автентифікований за першим фактором, тобто поки HTTP запит не буде містити в заголовку *Authorization* валідний токен доступу.

При реалізації даного атрибуту використовувався допоміжний метод *IsValidTotp*, головним завданням якого є валідація одноразоваго паролю, котрий надсилається в заголовку запиту.

. Якщо метод повертає значення *true*, це означає, що тимчасовий пароль співпав з тим паролем, котрий був згенерований на сервері. Тим самим, користувач підтвердить другий фактор автентифікації. Якщо метод повертає *false* – сервер поверне HTTP статус з кодом 401.

```
1 reference
private const string _otp_header = "X-OTP";
1 reference
public static bool IsValidTotp(this HttpRequest request, string key)
{
    StringValues otp;
    if (request.Headers.TryGetValue(_otp_header, out otp))
    {
        if (!string.IsNullOrWhiteSpace(otp.ToString()))
        {
            if (OtpGenerationService.GetOTPs(key).Any(t => t.Equals(otp)))
            {
                return true;
            }
        }
    }
    return false;
}
```

Рис. 3.13. Метод валідації одноразових тимчасових паролів.

3.4. Розробка API

У даній таблиці описано API (англ. Application Program Interface) — інтерфейс для влаємодії клієнтської та серверної частини сервісу на рівні бізнес-логіки.

Таблиця 3.4. API

Шлях	Потрібний вид автентифікації	Короткий опис
POST api/auth/register	Не потрібна	Реєстрація
POST api/auth/login	Не потрібна	Вхід в систему
GET api/transactions/history	Однофакторна	Демонстративний метод котрий повертає список транзакцій користувача
POST api/transactions/transfer	Багатофакторна	Виконання транзаційї

3.5. Розробка клієнтської частини застосунку

Клієнтська частина розробленої програми призначена лише для демонстрації роботи сервісу багатофакторної автентифікації, тому в ній не реалізовані поширені функції для повноцінного використання веб-застосунку (локалізація, адаптивність, сучасний дизайн тощо).

Дана часина системи складається з односторінкового застосунку на AngularJS. У файловій структурі рівня сирцевого коду, як і в серверній частині, знаходиться файл зі змінними оточення, *package.json* і *Web.config*. Основний файлом клієнтської частини є *app.js,* в якому відбувається ініціалізація та основного компонента в котрий будуть вставлятись інші.

Тека *content* буде містити використовувані стилі, *scripts* - модулі. Скрипт бібліотеки *angular.min.js* та всі інші потрібні скрипти підключається безпосередньо в файлі *index.html.*

Для взаємодії з віддаленим HTTP-сервером AnuglarJS представляє ключовий сервіс *$http*. При розробці використано бібліотеку Bootstap.

Таблиця 3.5. Шляхи навігації у клієнтській частині застосунку

Шлях	Опис
/home	Головна сторінка
/login	Увійти
/signup	Зареєструватись
/transhistory	Перегляд історій транзакцій
/transfer	Виконання транзакції

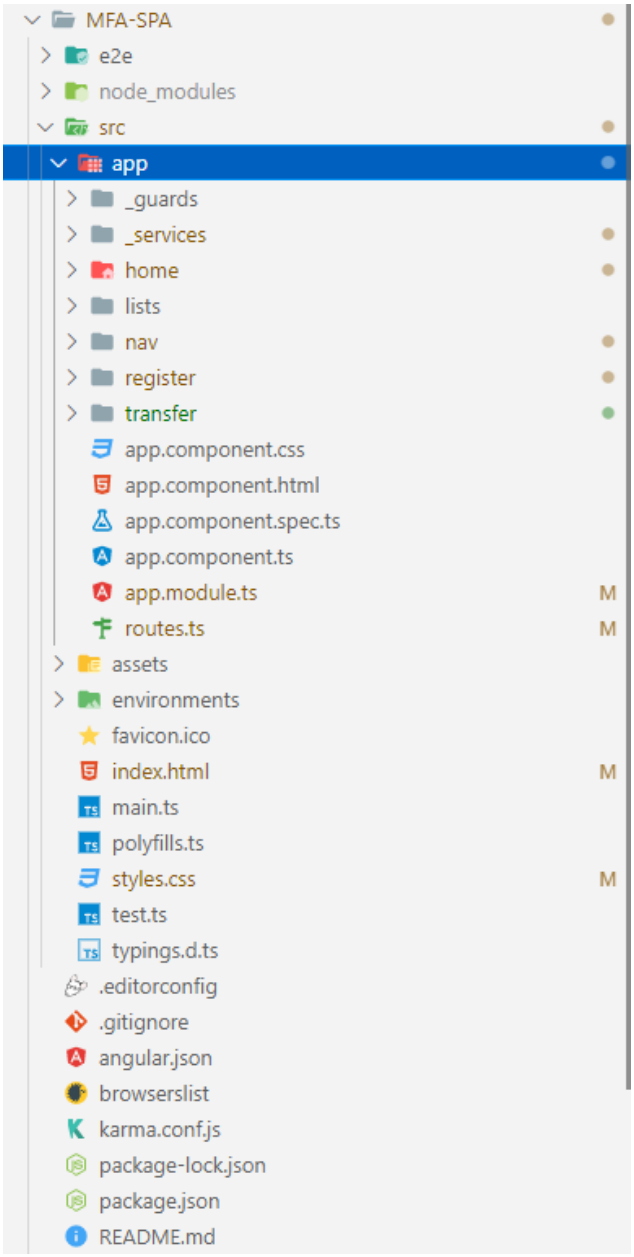


Рис. 3.13. Структура сирцевого коду клієнтської частини.

ВИСНОВКИ ДО РОЗДІЛУ 3

В розділі була описана розробка компонентів сервісу, а саме бази даних, серверної та клієнтської частини сервісу. Вибрані технології, описані в другому розділі дипломної роботи, були описані детальніше та показано, як їх використано при розробці сервісу.

В клієнтській частині сервісу реалізований SPA-застосунок з мінімалістичним веб-дизайном, котрий в змозі продемонструвати основні функції та цілі сервісу багатофакторної автентифікації. Було реалізовано генерування QR-коду для зручності використання розробленого сервісу з допомогою мобільного пристрою. Був розроблений захищений API на основі MVC-фреймворку ASP.NET Core Web API та спроєктована база даних для роботи сервісу.

РОЗДІЛ 4
ТЕСТУВАННЯ РОЗРОБЛЕНОГО СЕРВІСУ

В даному розділі описаний функціонал сервісу з точки зору користувача, оглянянуто його базовий функціонал та можливості. Було вирішено, що для тестування розробленого сервісу є недоцільним розгортання веб-застосунка в мережі. Тестування веб-додатку буде вводитись на локальній машині з домопогою таких технологій як Postman, середовище розробки та збірки веб-застосунків Visual Studio. Для тестування клієнтської частини застосунку було обрано браузер Google Chrome. Необхідним є мобільний пристрій, на котрому встановлено додаток Google Authenticator.

Спочатку перевіримо справність роботи API за допомогою технології Postman. Postman це свого роду швейцарський ніж, який дозволяє створювати і виконувати запити, документувати й моніторити сервіси в одному місці [31]. Виконаємо HTTP запит регестрації користувача. Для цього в тілі запиту потрібно ввести вхідні дані у форматі JSON. Як бачимо на рис. 4.1 результатом виконання запиту є HTTP статут 200, котрий говорить нам про успішне виконання. В тілі ми отримуємо Pre-Shared Key зареєстрованого користувача.

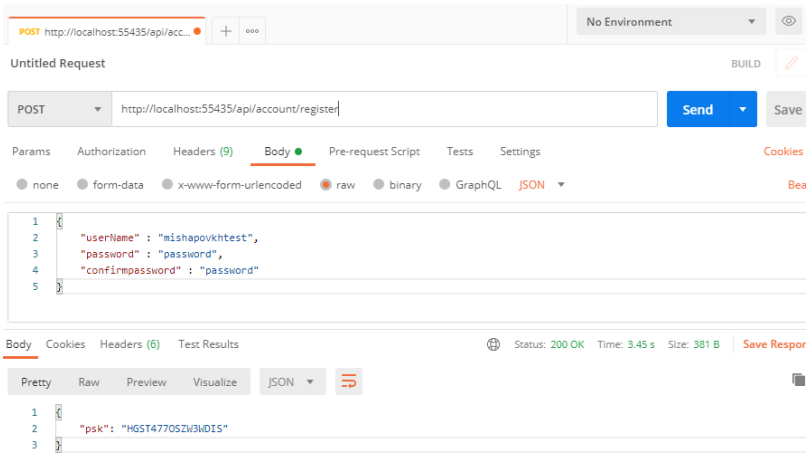


Рис. 4.1. Результат виконання незахищеного запиту реєстрації.

Тепер нам потрібно отримати токен доступу, щоб дозволити нам запитувати захищені шляхи. Це є першим фактором автентифікації, оскільки

користувач надасть своє ім’я користувача та пароль. Надішлемо запит HTTP POST до кінцевої точки /api/auth/login.

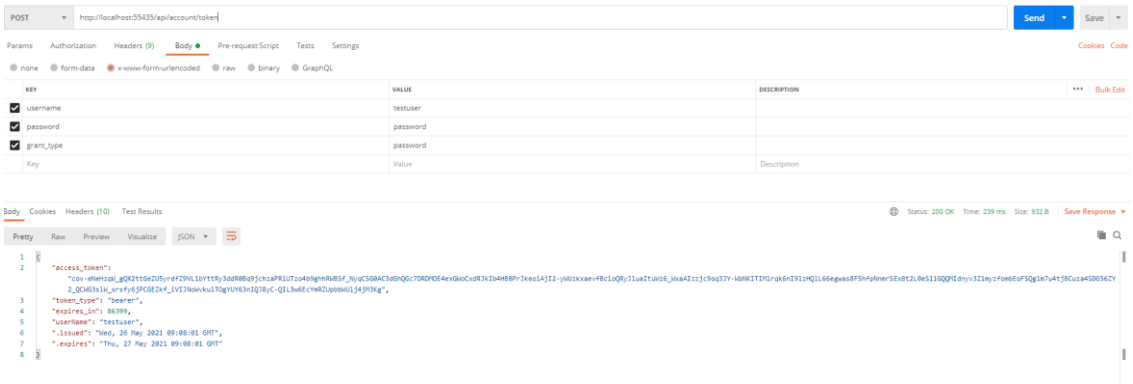


Рис. 4.2. Результат виконання запиту логування.

Далі на прикладі кінцевої точки /api/transactions/transfer покажемо, що успішне виконання запиту неможливе без надання друго фактору автентифікації (OTP).

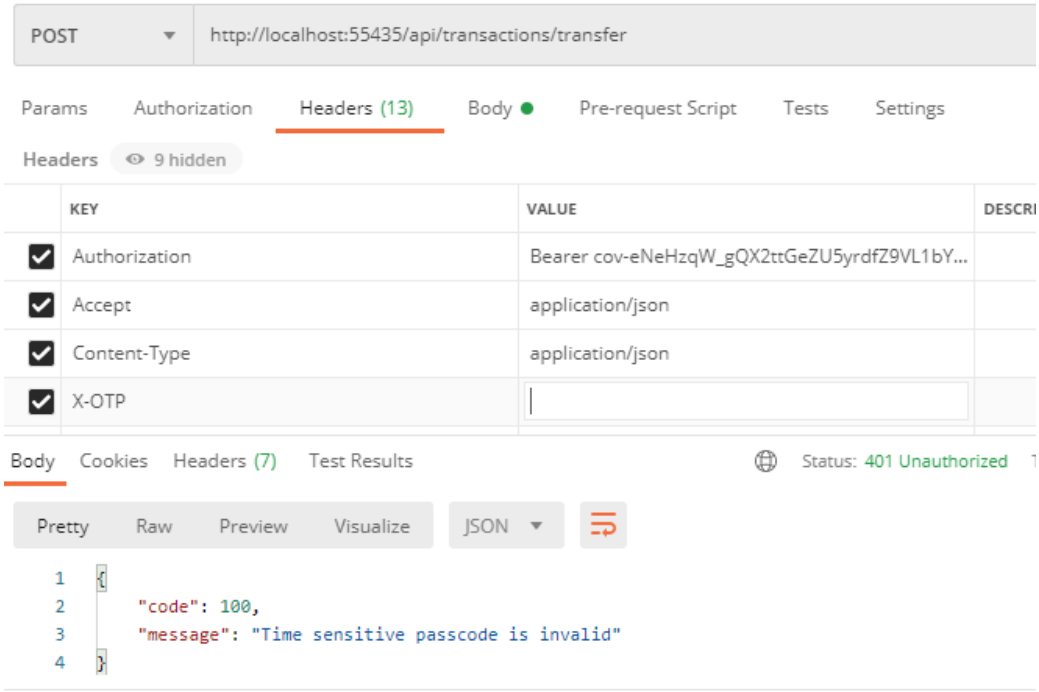


Рис. 4.3. Виконання транзакція без надання другого фактору.

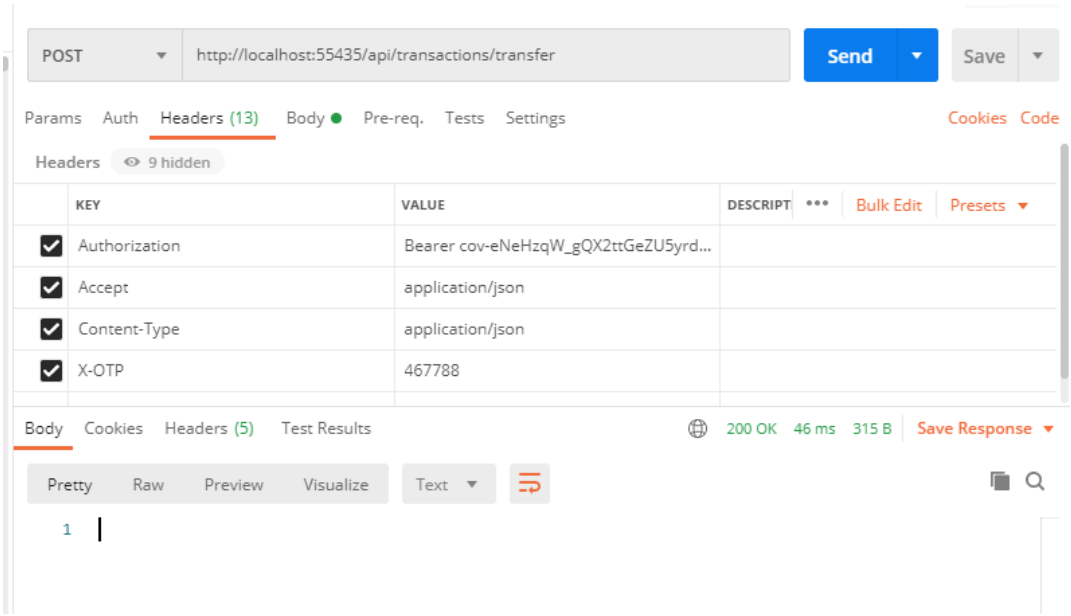


Рис. 4.3. Виконання транзакції з наданим валідним тимчасовим одноразовим паролем.

Тепер давайте протестуємо веб-застосунок разом з інтерфейсом. Отже, після запуску веб-розгортання розробленого сервісу на локальній машині у вікні браузера показується головна сторінка з можливістю увійти чи зареєструватись. На рис. 4.4 показано головну сторінку з реалізованим мінімалістичним дизайном. Сторінка не відрізняється для користувачів, що увійшли чи не увійшли в систему, і використовується для загального опису застосунку.

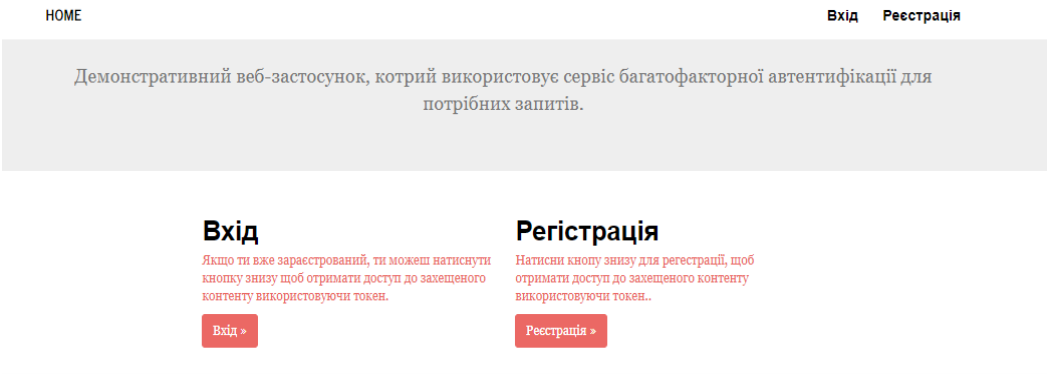


Рис. 4.4. Головна сторінка на локальному комп’ютері.

Користувач може увійти в систему за допомогою ім’я користувача та правильно введеного пароля. Це вважається першим етапом двофакторної автентифікації. В потрібних полях для вводу є валідація вхідних даних, що показано на рис. 4.5.

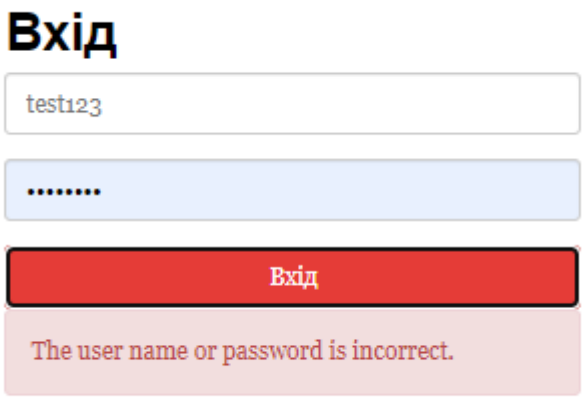


Рис. 4.5. Валідація вхідних даних у формі входу.

Детальніше розглянемо процес реєстрації користувача в систему. Для того, щоб бути зареєсрованим, послуговувачу даної системи потрібно надати унікальне ім'я користувача, пароль і його підтвердження у відповідні поля вводу. Після успішної валідації відбувається реєстрація. В API створюється унікальний Pre-Shared Key, який повертається в користувальцький інтерфейс у формі QR-коду та простого тексту (наприклад «MSXFPXNXKZ2MFD2T»). Для того, щоб пройти двофакторну автентифікацію для майбутніх захищених запитів, користувач має відкрити додаток Google Authenticator, встановлений на його смартфоні, і просканувати цей ключ або ввести його вручну, якщо на його мобільному пристрої відсутній сканер QR-коду. Інтерфейс користувача виглядатиме як на малюнку нижче:

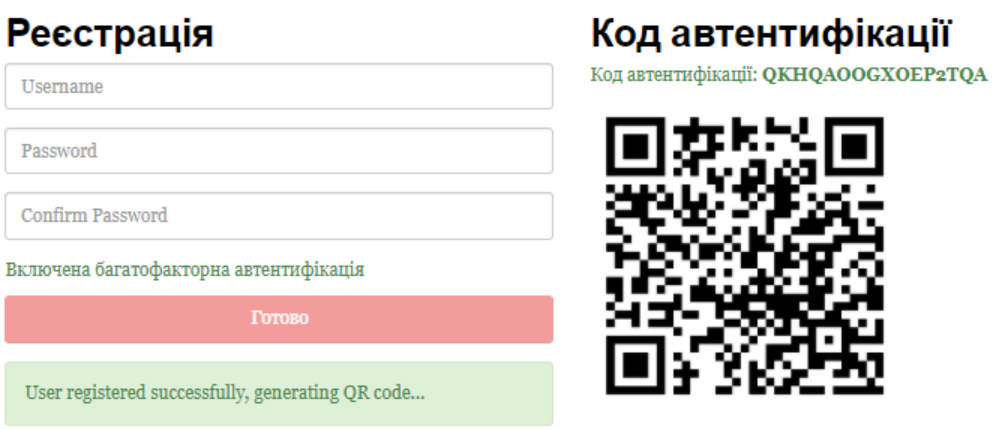


Рис. 4.6. Реєстрація користувача.

Після того, як користувач увійшов у систему, йому надається доступ до методів, котрі захищені однофакторною автентифікацією. Як показано на рис. 4.7, зареєстрований користувач, котрий надав правильні дані при вході в систему, матиме доступ до перегляду історії транзакцій.

Рис. 4.7. Доступ до шляху, котрий захищений однофакторною автентифікацією.

Для того, що користувач виконав запит з підвищеним рівнем захисту, як, наприклад, виконанання грошового платежу, сервіс багатофакторної автентифікації запросить надати другий фактор автентифікації (фактор власності). Користувацький інтерфейс надасть поле для вводу тимчасового коду доступу, котрий постійно генерується в мобільному пристрої.

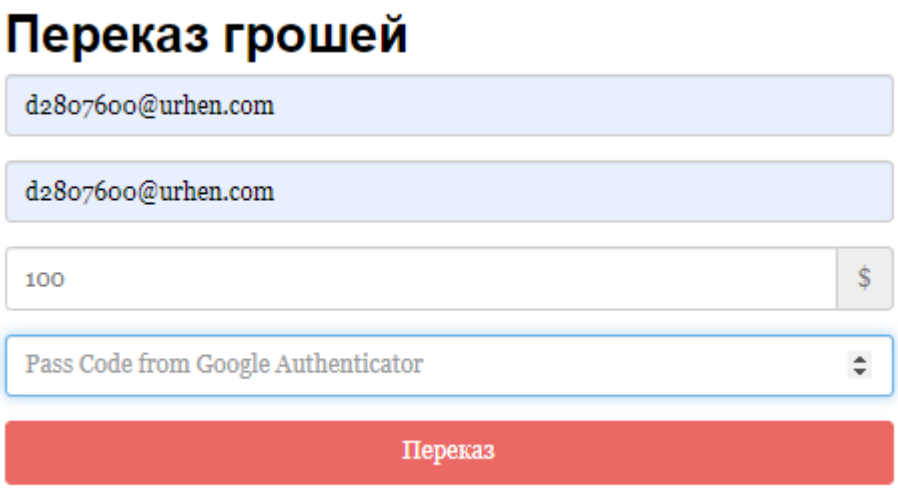


Рис. 4.8. Інтерфейс для виконання транзакції.

На рис 4.9 показано інтерфейс мобільного додатку Google Authonticator. Тут генеруються тимчасовий пароль кожні 30 секунд.

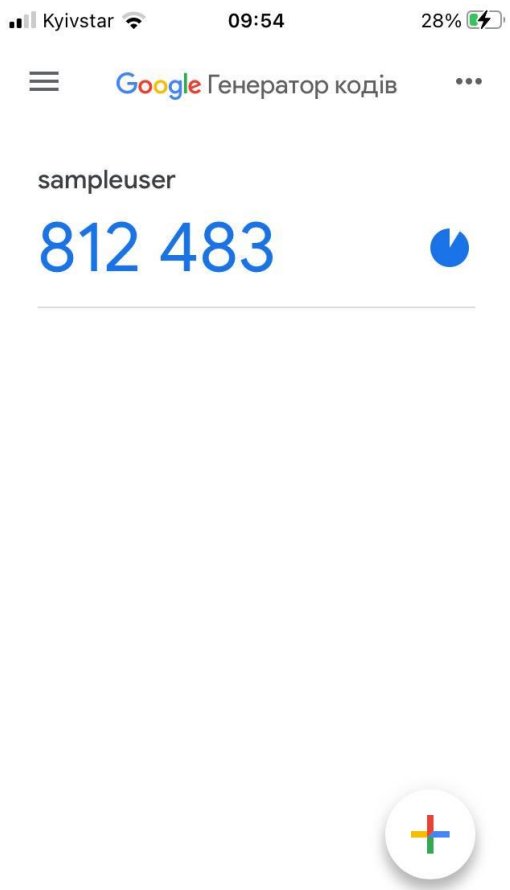


Рис. 4.9. Скріншот мобільного додатку.

Після того, як API отримає пароль, згенерований на мобільному пристрої, відбудеться процес валідації. Пройшовши перевірку другого фактора автентифікації, користувач здійснить переказ грошей, тобто запит з підвищеним захистом виконається.

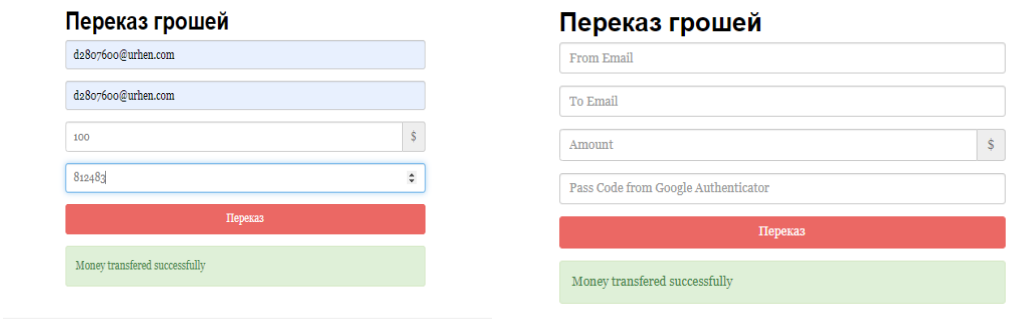


Рис. 4.10. Приклад проходження перевірки другого фактора автентифікації.

ВИСНОВКИ ДО РОЗДІЛУ 4

В розділі було протестовано роботу розробленого демонстраційного веб-застосунку та API сервіс багатофакторної автентифікації. Продемонстровано роботу базового функціоналу, а саме реєстрації користувача, його автентифікацію за допомогою одного та двох факторів.

Тестування сервісу було проведено з допомогою програми Postman, яка показує, що реалізований сервіс повністю відповідає вимогам системи, котру потрібно було розробити.

.

ВИСНОВКИ

В дипломній роботі було розглянута тема важливості автентифікації в сучасному світі. Детально було описано недоліки однофакторної автентифікації та переваги використання багатофакторної для веб-застосунків, котрі цього потребують для виконання чутливих до безпеки завдань. Розглянуті існуючі програмні рішення для реалізації даного виду автентифікації. Було виявлено, що вони не завжди можуть відповідати вимогам користувача. Більшість з них є досить нагромадженими зайвими методами чи факторами, котрі більшість людей ніколи не використовує через часозатратність чи відсутність сучасних сканерів, котрі можуть автентифікувати людину за допомогою її біометричних характеристик.

Запропоноване рішення є застосунком, що повністю відповідає сучасним тенденціям розробки подібних сервісів. Тобто розроблений сервіс реалізовує один з найнадійніших та найдешевших методів автентифікації користувача за фактором власності.

Окреслений мінімальний функціонал може використовуватись для надання додаткового рівня захисту потрібним процесам веб-застосунків, а використані для розробки технології дозволяють підтримувати і доповнювати сервіс у довгостроковій перспективі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Автентифікація (інформаційні технології) // Велика українська енциклопедія : у 30 т. / проф. А. М. Киридон (відп. ред.) та ін. — 2016. — Т. 1 : А — Акц. — 592 с. — ISBN 978-617-7238-39-2.
2. Harini, N.; Padmanabhan, T.R. 2CAuth: A new two factor authentication scheme using QR-code. Int. J.Eng. Technol. 2013, 5, 1087–1094.
3. Multi Factor Authentication A Survey — [Електронний ресурс] Режим доступу https://www.researchgate.net/publication/322288752_Multi-Factor_Authentication_A_Survey
4. Multifactor Authentication For Software Protection — [Електронний ресурс] Режим доступу https://www.researchgate.net/publication/315734643_MULTIFACTOR_AUTHENTICATION_FOR_SOFTWARE_PROTECTION
5. С.Н. Ардатский, О.С. Бартунов Управление доступом в сложных информационных системах. – Образовательные порталы России. Выпуск 1. - 2005.-187 с..
6. Multi-Factor Authentication from Duo — [Електронний ресурс] Режим доступу https://duo.com/product/multi-factor-authentication-mfa
7. ESET Secure Authentication Product Overview — [Електронний ресурс] Режим доступу https://cdn1.esetstatic.com/ESET/INT/Products/Business/Endpoint/Authentication/ESA/v03/ESET_Secure_Authentication_Product_Overview.pdf
8. Многофакторная аутентификация — [Електронний ресурс] Режим доступу https://multifactor.ru
9. Е. М. Пройдаков, Л. А. Теплицький. Англо-Український тлумачний словник з обчислювальної техніки, Інтернету і програмування. — СофтПрес. — С. 552. — ISBN 966-530-070-9.

10. Fielding Roy. Architectural Styles and the Design of Network-based Software Architectures. — Каліфорнійський університет в Ірвайні, 2000. — 24 May. Архівовано з джерела 7 січня 2017
11. Принципы для разработки: KISS, DRY, YAGNI, BDUF, SOLID, APO и бритва Оккама — [Електронний ресурс] Режим доступу https://habr.com/ru/company/itelma/blog/546372/.
12. Кратко о типах архитектур программного обеспечения, и какую из них выбрали мы для IaaS-провайдера — [Електронний ресурс] Режим доступу https://habr.com/ru/company/1cloud/blog/424911/.
13. What is Python? Executive Summary — [Електронний ресурс] Режим доступу https://www.python.org/doc/essays/blurb/.
14. Java — [Електронний ресурс] Режим доступу https://techterms.com/definition/java.
15. C# 6.0 draft specification — [Електронний ресурс] Режим доступу https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction.
16. What is JavaScript? — [Електронний ресурс] Режим доступу https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.
17. Silberschatz, Abraham; Sudarshan, S. (2011). Database system concepts (вид. 6). New York: McGraw-Hill. ISBN 9780073523323. OCLC 436031093. Codd. E. A relational model of data for large shared data banks. // Communications of the ACM. — 1970.
18. What is NoSQL? NoSQL Datadases Explained. Документація MongoDB. — [Електронний ресурс] Режим доступу https://www.mongodb.com/nosqlexplained
19. Chamith M. Session Management in Nodejs Using Redis as Session Store. // Medium. — 2020. — [Електронний ресурс] Режим доступу

https://medium.com/swlh/session-management-in-nodejs-using-redis-assessionstore-64186112aa9.

20. "What is Object/Relational Mapping?". Hibernate Overview. JBOSS Hibernate. Retrieved 19 April 2011.
21. What Is AngularJS? [Електронний ресурс] – Режим доступу https://docs.angularjs.org/guide/introduction.
22. "One-Time Passcodes for 2FA: A Fast or Slow Death? | IDology Blog". IDology. August 2, 2017.
23. HOTP: An HMAC-Based One-Time Password Algorithm [Електронний ресурс] // Network Working Group. – 2005. – Режим доступу https://tools.ietf.org/html/rfc4226.
24. TOTP: Time-Based One-Time Password Algorithm [Електронний ресурс] // Internet Engineering Task Force. – 2008. – Режим доступу https://tools.ietf.org/html/rfc6238.
25. Getting Started [Електронний ресурс] – Режим доступу: https://developers.google.com/api-client-library/javascript/start/start-js.
26. Authentication using the Google APIs Client [Електронний ресурс] – Режим доступу https://developers.google.com/api-client-library/javascript/start/start-js.
27. What is the Microsoft Authenticator app? [Електронний ресурс] – Режим доступу https://docs.microsoft.com/en-us/azure/active-directory/user-help/user-help-auth-app-overview.
28. Features [Електронний ресурс] // authy. – 2019. – Режим доступу https://authy.com/features/.
29. TouchID, PIN, Password, Encryption [Електронний ресурс] // Authy. – 2019. – Режим доступу https://authy.com/features/secure/.
30. Знайомтесь, Postman – для тестувальника must have [Електронний ресурс] – Режим доступу https://www.quality-assurance-group.com/znajomtes-postman-dlya-testuvalnyka-must-have/.

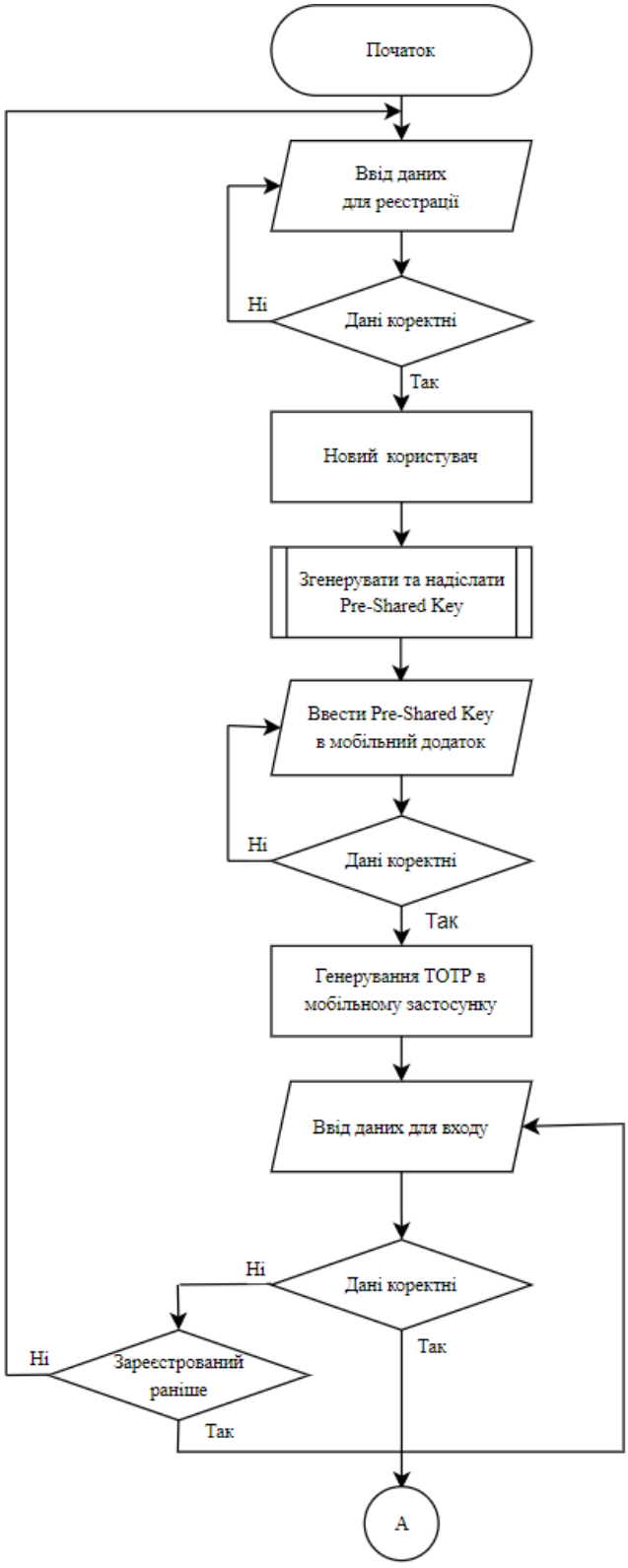
ДОДАТОК 1

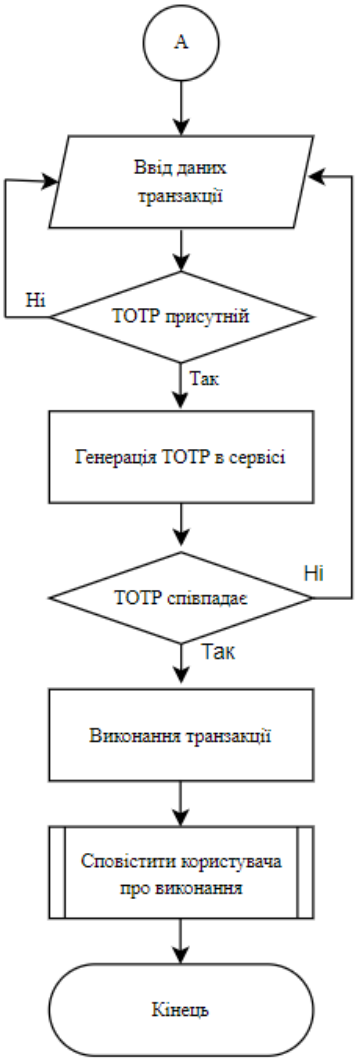
Сервіс багатофакторної автентифікації для веб-застосунків

Принципова схема – алгоритм роботи програми
ІАЛЦ.467200.004 Д1

Аркушів 1

Київ – 2021





					ІАЛЦ.467200.004 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Повх М. І.			*Принципова схема. Алгоритм роботи програми*	Лит. / Арк 1 / Аркушів 1
Перев.		Роковий О. П.				
Реценз.						НТУУ «КПІ», ФІОТ, ІВ-71
Н. Контр.		Сімоненко В. П.				
Затверд.		Стіренко С. Г.				

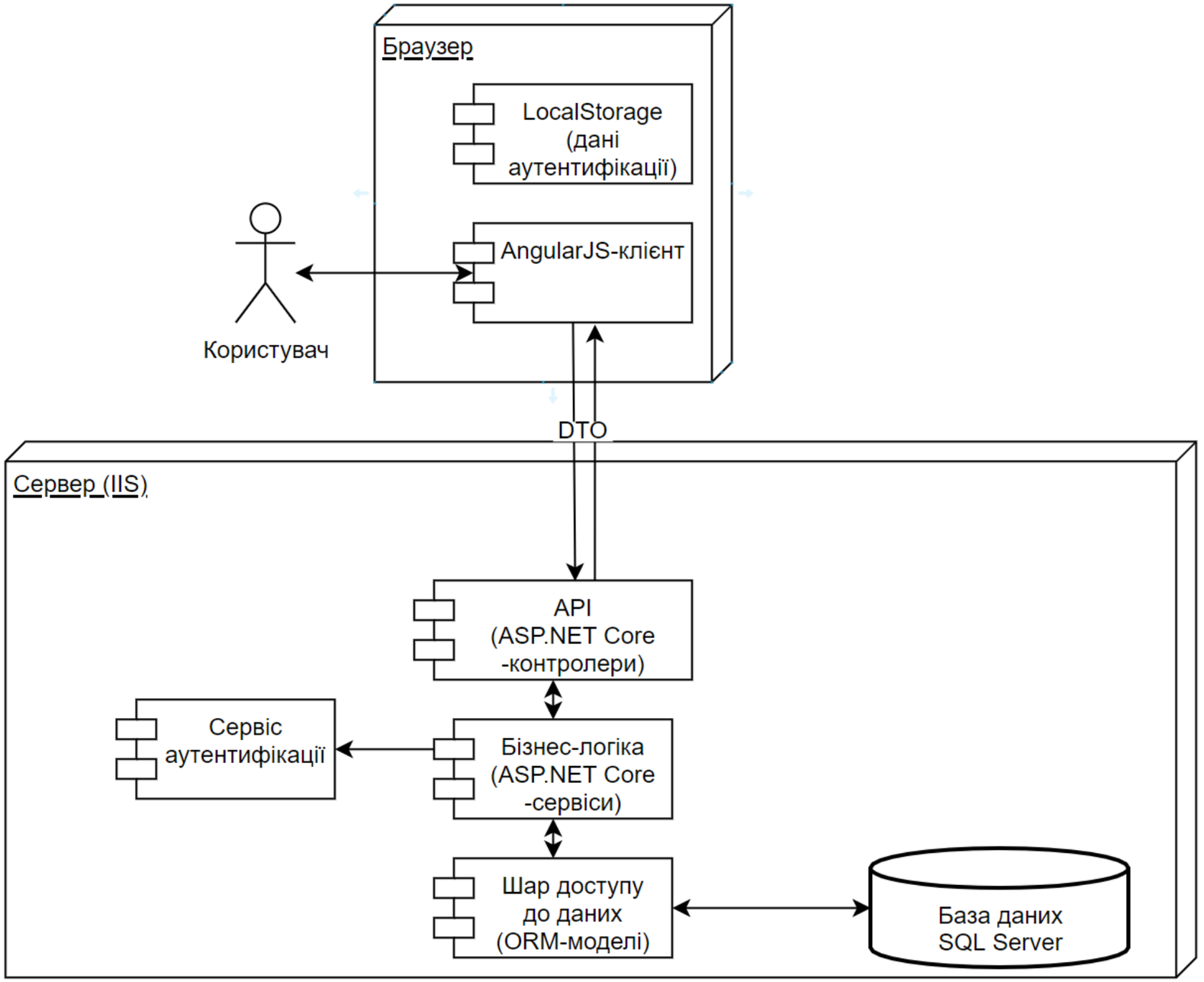
ДОДАТОК 2

Сервіс багатофакторної автентифікації для веб-застосунків

Структурна схема – діаграма розгортання застосунку
ІАЛЦ.467200.005 Д2

Аркушів 1

Київ – 2021



					ІАЛЦ.467200.005 Д2	
Змн	*Арк.*	*№ докум.*	*Підпис*	*Дат*	*Структурна схема. Діаграма розгортанння застосунку*	*Літ.* / *Маса* / *Масш.*
Розроб.		Повх М. І.				
Перевір.		Роковий О. П.				*Аркуш* / *Аркушів*
Т. Контр.						
Н. Контр.		Сімоненко В. П.			*Дипломна робота*	НТУУ «КПІ», ФІОТ, ІВ-71
Затверд.		Стіренко С. Г.				

ДОДАТОК З

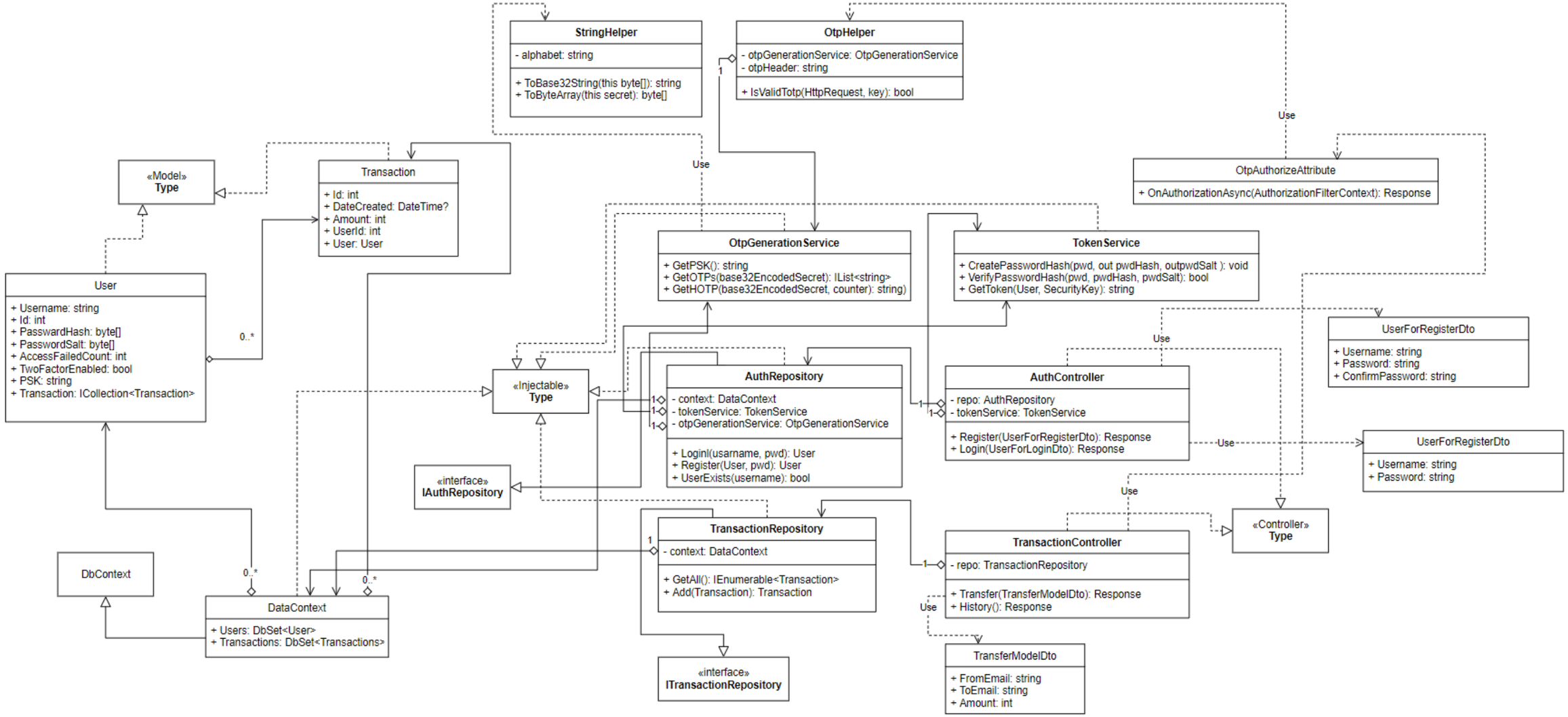
Сервіс багатофакторної автентифікації для веб-застосунків

Функціональна схема – діаграма класів

ІАЛЦ.467200.006 ДЗ

Аркушів 1

Київ – 2021



					ІАЛЦ.467200.006 ДЗ
Змн	Арк.	№ докум.	Підпис	Дат	
Розроб.		Повх М.І.			Функціональна схема. Діаграма класів
Перевір.		Роковий О. П.			Літ. / Маса / Масш.
Т. Контр.					Аркуш / Аркушів
Н. Контр.		Сімоненко В. П.			Дипломна робота
Затверд.		Стіренко С. Г.			НТУУ «КПІ», ФІОТ, ІВ-71

ДОДАТОК 4

Сервіс багатофакторної автентифікації для веб-застосунків

Лістинг програми
ІАЛЦ.467200.007 Д4

Аркушів 10

Київ – 2021

Клас User

```
using System.Collections.Generic;

namespace MFA.API.Models
{
    public class User
    {
        public int Id { get; set; }
        public string Username { get; set; }
        public byte[] PasswordHash { get; set; }
        public byte[] PasswordSalt { get; set; }
        public int AccessFailedCount { get; set; }
        public bool TwoFactorEnabled { get; set; }
        public string PSK { get; set; }
        public virtual ICollection<Transaction> Transactions { get; set; }
    }
}
```

Клас Transaction

```
using System;

namespace MFA.API.Models
{
    public class Transaction
    {
        public int Id { get; set; }
        public virtual User User { get; set; }
        public int UserId { get; set; }
        public DateTime? DateCreated { get; set; }
        public int Amount { get; set; }
    }
}
```

Клас DataContext

```
using MFA.API.Models;
using Microsoft.EntityFrameworkCore;

namespace MFA.API.Data
{
    public class DataContext : DbContext
    {
        public DataContext(DbContextOptions<DataContext> options) : base(options){}
        public DbSet<User> Users { get; set; }
        public DbSet<Transaction> Transactions { get; set; }
    }
}
```

Інтерфейс IAuthRepository

```
using System.Threading.Tasks;
using MFA.API.Models;

namespace MFA.API.Data
{
```

```
    public interface IAuthRepository
    {
         Task<User> Register(User user, string password);
         Task<User> Login(string username, string password);
         Task<bool> UserExists(string username);
    }
}
```

Інтерфейс ItransactionRepository

```
using System.Collections.Generic;
using System.Threading.Tasks;
using MFA.API.Models;

namespace MFA.API.Data
{
    public interface ITransactionRepository
    {
        Task<Transaction> Add(Transaction transaction);
        Task<IEnumerable<Transaction>> GetAll();

    }
}
```

Клас AuthRepository

```
using System;
using System.Threading.Tasks;
using MFA.API.Core;
using MFA.API.Models;
using Microsoft.EntityFrameworkCore;

namespace MFA.API.Data
{
    public class AuthRepository : IAuthRepository
    {
        private readonly DataContext _context;
        public AuthRepository(DataContext context)
        {
            _context = context;
        }

        public async Task<User> Login(string username, string password)
        {
            var user = await _context.Users.FirstOrDefaultAsync(x => x.Username == us
ername);

            if (user == null)
                return null;

            if (!TokenService.VerifyPasswordHash(password, user.PasswordHash, user.Pa
sswordSalt))
                return null;

            return user;
        }

        public async Task<User> Register(User user, string password)
```

```
        {
            byte[] passwordHash, passwordSalt;
            TokenService.CreatePasswordHash(password, out passwordHash, out passwordS
alt);

            user.PasswordHash = passwordHash;
            user.PasswordSalt = passwordSalt;
            user.PSK = OtpGenerationService.GetPSK();
            user.TwoFactorEnabled = true;

            await _context.Users.AddAsync(user);
            await _context.SaveChangesAsync();

            return user;
        }

        public async Task<bool> UserExists(string username)
        {
            if (await _context.Users.AnyAsync(x => x.Username == username))
                return true;

            return false;
        }
    }
}
```

Клас TransactionRepository

```
using System.Collections.Generic;
using System.Threading.Tasks;
using MFA.API.Models;
using Microsoft.EntityFrameworkCore;

namespace MFA.API.Data
{
    public class TransactionRepository : ITransactionRepository
    {
        private readonly DataContext _context;
        public TransactionRepository(DataContext context)
        {
            _context = context;

        }

        public async Task<IEnumerable<Transaction>> GetAll()
        {
            var transactions = await _context.Transactions.Include(t => t.User).ToLis
tAsync();
            return transactions;
        }

        public async Task<bool> SaveAll()
        {
            return await _context.SaveChangesAsync() > 0;
        }

        public async Task<Transaction> Add(Transaction transaction)
        {
            await _context.Transactions.AddAsync(transaction);
```

```
            await _context.SaveChangesAsync();
            return transaction;
        }
    }
}
```

Клас TokenService

```
using System;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using MFA.API.Models;
using Microsoft.IdentityModel.Tokens;

namespace MFA.API.Core
{
    public static class TokenService
    {
        public static void CreatePasswordHash(string password, out byte[] passwordHas
h, out byte[] passwordSalt)
        {
            using (var hmac = new System.Security.Cryptography.HMACSHA512())
            {
                passwordSalt = hmac.Key;
                passwordHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(pa
ssword));
            }
        }

        public static bool VerifyPasswordHash(string password, byte[] passwordHash, b
yte[] passwordSalt)
        {
            using (var hmac = new System.Security.Cryptography.HMACSHA512(passwordSal
t))
            {
                var computedHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetByte
s(password));
                for (int i = 0; i < computedHash.Length; i++)
                {
                    if (computedHash[i] != passwordHash[i]) return false;
                }
            }
            return true;
        }
        public static string GetToken(User user, SecurityKey key)
        {
            var claims = new[]
            {
                new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
                new Claim(ClaimTypes.Name, user.Username),
                new Claim("PSK", user.PSK)
            };

            var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Sign
ature);

            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(claims),
```

```
                Expires = DateTime.Now.AddDays(1),
                SigningCredentials = creds
            };

            var tokenHandler = new JwtSecurityTokenHandler();
            var token = tokenHandler.CreateToken(tokenDescriptor);
            return tokenHandler.WriteToken(token);
        }
    }
}
```

Клас OtpGenerationService

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography;
using MFA.API.Helpers;

namespace MFA.API.Core
{
    public static class OtpGenerationService
    {
        public static string GetPSK()
        {
            byte[] key = new byte[10];
            using (var rngProvider = new RNGCryptoServiceProvider())
            {
                rngProvider.GetBytes(key);
            }
            return key.ToBase32String();
        }

        public static IList<string> GetOTPs(string base32EncodedSecret)
        {
            DateTime epochStart = new DateTime(1970, 01, 01, 0, 0, 0, 0, DateTimeKind
.Utc);

            long counter = (long)Math.Floor((DateTime.UtcNow - epochStart).TotalSecon
ds / 30);
            var otps = new List<string>();

            otps.Add(GetHOTP(base32EncodedSecret, counter - 1));
            otps.Add(GetHOTP(base32EncodedSecret, counter));
            otps.Add(GetHOTP(base32EncodedSecret, counter + 1));

            return otps;
        }

        private static string GetHOTP(string base32EncodedSecret, long counter)
        {
            byte[] message = BitConverter.GetBytes(counter).Reverse().ToArray();
            byte[] secret = base32EncodedSecret.ToByteArray();
            HMACSHA1 hmac = new HMACSHA1(secret, true);

            byte[] hash = hmac.ComputeHash(message);
            int offset = hash[hash.Length - 1] & 0xf;
            int truncatedHash = ((hash[offset] & 0x7f) << 24) |
            ((hash[offset + 1] & 0xff) << 16) |
```

```
            ((hash[offset + 2] & 0xff) << 8) |
            (hash[offset + 3] & 0xff);

            int hotp = truncatedHash % 1000000;
            return hotp.ToString().PadLeft(6, '0');
        }
    }
}
```

Клас OtpHelper

```
using System.Linq;
using MFA.API.Core;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Primitives;

namespace MFA.API.Helpers
{
    public static class OtpHelper
    {
        private const string _otp_header = "X-OTP";
        public static bool IsValidTotp(this HttpRequest request, string key)
        {
            StringValues otp;
            if (request.Headers.TryGetValue(_otp_header, out otp))
            {
                if (!string.IsNullOrWhiteSpace(otp.ToString()))
                {
                    if (OtpGenerationService.GetOTPs(key).Any(t => t.Equals(otp)))
                    {
                        return true;
                    }
                }
            }
            return false;
        }
    }
}
```

Клас StringHelper

```
using System;
using System.Linq;

namespace MFA.API.Helpers
{
    public static class StringHelper
    {
        private static string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567";
        public static string ToBase32String(this byte[] secret)
        {
            var bits = secret.Select(b => Convert.ToString(b, 2).PadLeft(8, '0')).Agg
regate((a, b) => a + b);
            return Enumerable.Range(0, bits.Length / 5).Select(i => alphabet.Substrin
g(Convert.ToInt32(bits.Substring(i * 5, 5), 2), 1)).Aggregate((a, b) => a + b);
        }

        public static byte[] ToByteArray(this string secret)
```

```
        {
            var bits = secret.ToUpper().ToCharArray().Select(c => Convert.ToString(al
phabet.IndexOf(c), 2).PadLeft(5, '0')).Aggregate((a, b) => a + b);
            return Enumerable.Range(0, bits.Length / 8).Select(i => Convert.ToByte(bi
ts.Substring(i * 8, 8), 2)).ToArray();
        }
    }
}
```

Клас OtpAuthorizeAttribute

```
using System.Threading.Tasks;
using MFA.API.Helpers;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc.Filters;
using System;
using Microsoft.AspNetCore.Mvc;

namespace MFA.API.Attributes
{
    [AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple =
 true, Inherited = true)]
    public class OtpAuthorizeAttribute : AuthorizeAttribute, IAsyncAuthorizationFilte
r
    {
        public Task OnAuthorizationAsync(AuthorizationFilterContext context)
        {
            var preSharedKey = context.HttpContext.User.FindFirst("PSK").Value;
            bool hasValidTotp = OtpHelper.IsValidTotp(context.HttpContext.Request, pr
eSharedKey);

            if (hasValidTotp)
            {
                return Task.FromResult<object>(null);
            }
            else
            {
                context.Result = new UnauthorizedResult();
                return Task.FromResult<object>(null);
            }
        }
    }
}
```

Клас TransferModelDto

```
namespace MFA.API.Dto
{
    public class TransferModelDto
    {
        public string FromEmail { get; set; }
        public string ToEmail { get; set; }
        public double Amount { get; set; }
    }
}
```

Клас UserForLoginDto

```
namespace MFA.API.Dto
{
    public class UserForLoginDto
    {
        public string Username { get; set; }
        public string Password { get; set; }
    }
}
```

Клас UserForRegisterDto

```
using System.ComponentModel.DataAnnotations;

namespace MFA.API.Dto
{
    public class UserForRegisterDto
    {
        [Required]
        public string Username { get; set; }

        [Required]
        [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters lo
ng.", MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm password")]
        [Compare("Password", ErrorMessage = "The password and confirmation password d
o not match.")]
        public string ConfirmPassword { get; set; }
    }
}
```

Клас AuthController

```
using System.Text;
using System.Threading.Tasks;
using MFA.API.Core;
using MFA.API.Data;
using MFA.API.Dto;
using MFA.API.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;

namespace MFA.API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private readonly IAuthRepository _repo;
        private readonly IConfiguration _config;
        public AuthController(IAuthRepository repo, IConfiguration config)
        {
```

```
            _config = config;
            _repo = repo;
        }

        [HttpPost("register")]
        public async Task<IActionResult> Register(UserForRegisterDto userForRegisterD
to)
        {
            userForRegisterDto.Username = userForRegisterDto.Username.ToLower();

            if (await _repo.UserExists(userForRegisterDto.Username))
                return BadRequest("Username already exists");

            var userToCreate = new User
            {
                Username = userForRegisterDto.Username
            };

            var createdUser = await _repo.Register(userToCreate, userForRegisterDto.P
assword);

            return Ok(new
            {
                PSK = createdUser.PSK
            });
        }

        [HttpPost("login")]
        public async Task<IActionResult> Login(UserForLoginDto userForLoginDto)
        {
            var userFromRepo = await _repo.Login(userForLoginDto.Username
                .ToLower(), userForLoginDto.Password);

            if (userFromRepo == null)
                return Unauthorized();

            var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config.GetSect
ion("AppSettings:Token").Value));
            string tokenString = TokenService.GetToken(userFromRepo, key);
            return Ok(new
            {
                token = tokenString
            });
        }
    }
}
```