МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

“До захисту допущений”
Зав. Кафедри комп’ютерних наук та прикладної математики
д. т. н., професор Турбал Ю.В.
«____»_____ 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА
ТЕМА:

«МЕТОДИ ТА ТЕХНОЛОГІЇ ЗАХИСТУ АВТОРИЗАЦІЇ ВЕБ-ДОДАТКУ»

Виконав: Рожанчук Богдан Олегович
Студент навчально-наукового інституту автоматики, кібернетики та обчислювальної техніки
групи КН-41.
_____
Підпис

Керівник: Зубик Ярослав Ярославович

_____
Підпис

Національний університет водного господарства та природокористування

Навчально науковий інститут автоматики, кібернетики та обчислювальної техніки

Кафедра **комп'ютерних наук та прикладної математики**

Освітньо-кваліфікаційний рівень **бакалавр**

Галузь знань 12 «*Інформаційні технології*»

Спеціальність 122 «*Комп'ютерні науки*»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

“____” _____20___року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Рожанчуку Богдану Олеговичу

1. Тема роботи *Методи багатофакторної автентифікації у веб-застосунках*

керівник роботи к.т.н., доцент Зубик Ярослав Ярославович, ст. викл.

затверджені наказом вищого навчального закладу від *19 квітня 2023 року С № 449*.

2. Термін подання роботи студентом *23 червня 2023 р.*

3. Вихідні дані: провести дослідження основних понять і методів, які забезпечують автентифікацію для запобігання несанкціонованому доступу до веб-застосунків та інформації, що вони містять. При цьому варто врахувати особливості веб-ресурсів. Після аналізу методів автентифікації, необхідно реалізувати їх та протестувати на практиці.

4. Зміст розрахунково-пояснювальної записки: Тестування обраних та реалізованих методів захисту від несанкціонованого доступу до інформації .

5.Консультанти розділів роботи(проекту)

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Розділ 1	Зубик Я.Я.		
Розділ 2	Зубик Я.Я.		
Розділ 3	Зубик Я.Я.		
Розділ 4	Зубик Я.Я.		

7. Дата видачі завдання : 30.09.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Огляд літератури за обраною тематикою		
2	Аналіз літературних джерел		
3	Обґрунтування рішення		
4	Аналіз веб-застосунків та особливостей їх архітектури		
5	Аналіз сучасних методів автентифікації		
6	Розробка та дослідження програмного комплексу веб-застосунку багатофакторної автентифікації		
7	Підготовка виступу і презентації		

Студент _____ Рожанчук Б.О.

(підпис) (прізвище та ініціали)

Керівник роботи _____ Зубик Я. Я.

(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

WWW – Всесвітня мережа (World Wide Web)

HTTP – Протокол передачі даних (Hyper Text Transfer Protocol)

PKI − Інфраструктура публічного ключа (Public Key Infrastructure)

SSL/TLS – Протоколи захищеної передачі даних (Secure Sockets Layer/Transport Layer Security)

2FA - Двофакторна автентифікація (Two-Factor Authentication)

OTP - Одноразовий пароль (One-Time Password)

JWT - Токен JSON (JSON Web Token)

ВСТУП

У початку третього тисячоліття спостерігається експоненціальний розвиток цифрових технологій, який суттєво впливає на людей та суспільство в цілому. Сучасний світ стає все більш цифровим, і кожен з нас постійно користується програмами та технологічними засобами не тільки для особистих, але й для робочих потреб. Перехід до цифрової економіки вже стає невід'ємною частиною розвитку сучасної держави та суспільства. Смартфони, 4G-покриття та публічні WI-FI мережі стали невід'ємною частиною нашої повсякденної рутини.

Важливим аспектом для нас всіх є безпека особистих даних, а також безпека держави та суспільства, яка є невід'ємною складовою національної безпеки. Для забезпечення цих аспектів ми використовуємо різні програмні та апаратні засоби, які не завжди є оптимальними з функціональної та економічної точки зору.

Безпека інформаційної системи є одним з найважливіших завдань під час її експлуатації, оскільки швидкість прийняття рішень, ефективність та надійність роботи значною мірою залежать від конфіденційності, цілісності та доступності інформаційних ресурсів. У сучасному світі, де кожна компанія або особа потребує захисту своїх даних, безпека та перевірка автентичності стають особливо важливими.

Традиційні механізми простої автентифікації вичерпують себе. Компанії, які продовжують використовувати традиційні методи автентифікації, такі як паролі, стикаються з ризиком злому та несанкціонованого доступу до інформації. Тому сучасні технології безпеки інформаційних систем орієнтовані на використання більш надійних і складніших методів автентифікації.

Одним з таких методів є біометрична автентифікація, яка використовує унікальні фізіологічні або поведінкові характеристики людини для ідентифікації. Наприклад, відбиток пальця, розпізнавання обличчя, сканування райдужної оболонки ока та інші біометричні дані можуть бути використані для підтвердження особи.

Ще одним інноваційним підходом є використання двофакторної або багатофакторної автентифікації. Цей метод передбачає комбінацію кількох незалежних факторів для підтвердження особи, наприклад, пароль разом з використанням фізичного пристрою (такого як смартфон) або відбитку пальця.

Також значний розвиток отримала автентифікація на основі публічного ключа (PKI). Цей метод використовує криптографічні ключі для підтвердження особи та забезпечення безпеки комунікації. Він базується на використанні публічного та приватного ключів, що забезпечують цілісність та конфіденційність даних.

Крім того, блокчейн-технології також можуть бути використані для автентифікації та забезпечення безпеки інформації. Блокчейн забезпечує розподілену та недоступну для змін базу даних, що дозволяє створити довірену мережу, яка може бути використана для підтвердження автентичності даних та осіб.

Враховуючи швидкий розвиток технологій, майбутнє автентифікації може бути пов'язане з використанням штучного інтелекту, машинного навчання та аналізу поведінки користувачів для виявлення аномальних дій та підозрілих активностей.

Важливо зазначити, що незалежно від використовуваних методів автентифікації, постійний розвиток та оновлення систем безпеки є необхідним для протидії новим загрозам і забезпечення надійності захисту інформації.

РОЗДІЛ 1
ОГЛЯД СУЧАСНОГО СТАНУ ВИКОРИСТАННЯ ВЕБ-ЗАСТОСУНКІВ ТА АНАЛІЗ ЇХ БЕЗПЕКОВИХ АСПЕКТІВ.

1.1 Архітектура World Wide Web

World Wide Web (WWW) є впливовою та широко використовуваною інтернет-технологією, що радикально змінила спосіб сприйняття та обміну інформацією. Всесвітня мережа базується на протоколі HTTP, забезпечує зручний та універсальний інтерфейс для звернення до різноманітних ресурсів в мережі Інтернет.

WWW був винайдений та розроблений Тімоті Бернерс-Лі в 1989 році в Європейському організації ядерних досліджень (CERN). Вперше він був представлений як система гіпертекстового документування, де текстові документи могли містити гіперпосилання, що дозволяли переходити з одного документу на інший. Згодом ця концепція розвинулася і перетворилася на веб-браузери, веб-сайти та веб-додатки, які ми використовуємо сьогодні.

Основою WWW є гіпертекстовий документ, який може містити текст, зображення, відео, звук та інші медіа-елементи. Головною характеристикою гіпертексту є наявність гіперпосилань, які дозволяють користувачеві переходити з одного документу на інший, незалежно від їх фізичного розташування.

У веб-браузері користувач може вводити URL-адреси веб-сторінок, що викликають запити до веб-серверів. Веб-сервери, у свою чергу, надсилають веб-сторінки назад до браузера, який їх відображає на екрані. Цей процес здійснюється за допомогою протоколу HTTP (Hypertext Transfer Protocol).

WWW стало платформою для безлічі інтернет-послуг та додатків, таких як соціальні мережі, електронна комерція, онлайн-медіа, онлайн-банкінг та інше.

Простота та доступність WWW дозволяють користувачам з усього світу отримувати інформацію, спілкуватися та проводити різноманітні дії в Інтернеті.

1.2 Класифікація Веб-застосунків

Веб-застосунки використовуються в різних сферах діяльності і надають зручний доступ до різноманітних послуг та ресурсів в Інтернеті. Сучасні технології та інновації сприяють швидкому розвитку веб-застосунків, що стають важливим компонентом для бізнесу. Компанії все частіше звертаються до послуг розробки веб-застосунків, оскільки вони можуть ефективно вирішувати великий спектр проблем та задовольняти потреби різних користувачів.

Веб-застосунок (веб-додаток) є програмним забезпеченням, розробленим для роботи в інтернет-середовищі і призначеним для взаємодії з користувачем через веб-браузер. Він надає можливість користувачам виконувати різноманітні функції, доступатися до інформації, спілкуватися, виконувати операції та взаємодіяти з іншими користувачами через мережу Інтернет.

Класифікація веб-застосунків важлива для систематизації їх особливостей, включаючи стек технологій, сферу застосування, архітектуру та навантаженість системи. Ці методи допомагають визначати характеристики додатків і підходи до їх розробки.

Важливим аспектом веб-застосунків є їх навантаженість, яка визначається кількістю користувачів, що можуть взаємодіяти з системою одночасно, та обсягом даних, які обробляються. Навантаженість веб-застосунків може бути високою, особливо для популярних та масштабованих систем, і вимагає ефективного управління ресурсами

1.3 Архітектура Веб-застосунків

Архітектура програмного забезпечення є структурою, на основі якої будується додаток і взаємодіють його модулі та компоненти. Веб-додатки мають свою власну архітектуру, яка визначає взаємодію програмних компонентів та спосіб доставки даних через HTTP. Існують різні архітектурні шаблони, які використовуються при проектуванні програмного забезпечення. Основними компонентами веб-додатків є веб-браузер, веб-сервер і сервер баз даних.

Компоненти веб-додатків. Зазвичай веб-додатки складаються з трьох основних компонентів:

- Веб-браузер, який взаємодіє з користувачем, отримує вхідні дані та контролює логіку відображення, забезпечуючи взаємодію з додатком. При необхідності проводиться перевірка вхідних даних користувача.
- Веб-сервер, який знаходиться на стороні сервера і відповідає за обробку бізнес-логіки та запитів користувачів, керуючи операціями програми. Він здатний обробляти запити від різних клієнтів.
- Сервер бази даних, який забезпечує необхідні дані для роботи програми та опрацьовує завдання, пов'язані з даними. У багаторівневій архітектурі сервери баз даних можуть виконувати бізнес-логіку за допомогою збережених процедур.

Побудова багаторівневої архітектури сучасного веб-додатка дозволяє чітко визначити роль кожного компонента програми та забезпечує легку модифікацію на відповідному рівні, не впливаючи на решту додатка. Це сприяє зручному написанню, налагодженню, керуванню та повторному використанню коду.

Веб-додаток має архітектуру, що дозволяє користувачам взаємодіяти з сервером та серверною службою через браузер. Код розміщений у браузері, де він отримує запити і надає користувачеві необхідну інформацію. Саме тут користувач

бачить дизайн UI/UX, інформаційні панелі, повідомлення, конфігураційні налаштування, макет та інтерактивні елементи. Ось кілька часто використовуваних технологій інтерфейсу:

- **HTML**: стандартна мова розмітки, яка дозволяє розробникам структурувати вміст веб-сторінок за допомогою набору елементів сторінки.
- **CSS**: популярна мова таблиць стилів, яка дозволяє розробникам відокремлювати вміст і макет веб-сайтів для сайтів, розроблених з використанням мов розмітки. За допомогою CSS можна задавати стиль елементів та повторно їх використовувати.
- **JavaScript**: популярна мова програмування на стороні клієнта, яка використовується на більшості веб-сайтів. Усі браузери мають вбудований движок JavaScript для виконання JavaScript-коду на пристроях. Використання JS на клієнтській стороні дозволяє зменшити навантаження на сервер.
- **React**: відкритий JS-фреймворк, який набув популярності в останні роки. React дозволяє розробникам легко створювати високоякісні динамічні веб-додатки з мінімальними зусиллями та обсягом коду. Він простий у використанні та має широку документацію та набір зручних інструментів, що дозволяють повторно використовувати код.
- **Vue**: JS-фреймворк, який дозволяє розробникам легко створювати користувацькі інтерфейси для вебу, настільних комп'ютерів та мобільних пристроїв. Vue.js постачається з набором зручних інструментів, що задовольняють базові потреби програмування, і легко інтегрується з існуючими проектами. Його документація лаконічна і добре структурована.
- **Angular**: відкритий JS-фреймворк для веб-додатків, який є одним з популярних інтерфейсів розробки середовищ на ринку. Він надає всі необхідні функції для розробки додатків, такі як компоненти, модулі, шаблони, директиви, сервіси та залежності, маршрутизація та інше.

Серверний компонент є важливою складовою архітектури веб-додатку. Він приймає запити користувачів, виконує бізнес-логіку та забезпечує доступ до зовнішніх систем. Існує декілька популярних технологій для реалізації серверного компонента:

- **Java**: об'єктно-орієнтована мова програмування, яка дозволяє розробникам писати код для будь-якої платформи, використовуючи віртуальну машину Java. Вона проста у вивченні та має потужні можливості.
- **Python**: високорівнева мова програмування з відкритим вихідним кодом, яка має широкий спектр застосувань. Python підходить для різних проектів, включаючи веб-додатки, мобільні додатки та ігри.
- **Ruby**: популярна мова програмування, яка разом з фреймворком Ruby on Rails дозволяє швидко розробляти та розгортати програми. Вона має багато корисних інструментів та забезпечує безпеку додатків.
- **Golang**: проста мова програмування, яка нагадує мову C. Код на Go компілюється швидко та створює компактні файли. Вона має вбудовану бібліотеку та підтримку тестування.
- **Laravel**: PHP-фреймворк з відкритим вихідним кодом, який надає розробникам зручні та потужні інструменти для швидкої та ефективної розробки високоякісних веб-додатків. Laravel також пропонує розширення за допомогою пакетів Composer, що дозволяє розширити функціональність фреймворку шляхом використання готових рішень, розроблених спільнотою розробників.
- **Node.js**: відкрите кросплатформове середовище виконання з відкритим вихідним кодом, яке має велику бібліотеку модулів, яка дозволяє розробникам швидко реалізовувати різноманітні функціональні можливості. Вона включає модулі для роботи з мережею, файловою системою, базами даних, шифруванням, роботи з HTTP і багато інших. Це дозволяє розробникам

ефективно будувати веб-сервери, REST API, чат-боти, мікросервіси та інші типи додатків.

- **.NET**: базується на моделі об'єктно-орієнтованого програмування, що дозволяє розробникам створювати складні програми, розділяючи їх на менші модулі або компоненти.

Архітектура веб-додатків може бути класифікована залежно від способу розробки та розгортання програмного забезпечення. Побудова архітектури веб-додатку передбачає комплекс заходів, спрямованих на чітке визначення структури системи. Варто зауважити, що немає універсального рішення для архітектури, оскільки все залежить від конкретного проекту, його мети і особливостей роботи веб-додатку.

Для визначення архітектури проекту необхідно мати:

- Технічне завдання.
- Архітектора або досвідченого розробника.
- Розуміння обмеження часу.
- Усвідомлення життєвого циклу проекту.
- Розуміння очікуваного навантаження.

Для визначення повної архітектури проекту необхідно аналізувати значну кількість інформації. Проте, для визначення типу архітектури може бути достатньо загальної концепції проекту.

Загалом, існують три основних типи архітектури веб-додатків:

- Монолітна архітектура.
- Мікросервісна архітектура.
- Безсерверна архітектура.

Монолітна архітектура, яка є традиційним підходом, передбачає об'єднання всього програмного коду та функціоналу в одному великому блоку - моноліті. Такий

додаток складається з презентаційного шару, бізнес-логіки та доступу до бази даних, які працюють в межах одного процесу. Монолітна архітектура має свої переваги, зокрема простоту розробки, легкість у розгортанні та управлінні. Однак, у неї також є недоліки, такі як складність масштабування, залежність компонентів та важкість впровадження нових технологій.

З іншого боку, мікросервісна архітектура - це підхід, в якому веб-додаток розбивається на набір невеликих автономних сервісів, які працюють окремо та взаємодіють між собою через мережу. Кожен сервіс відповідає за конкретну функціональність і може бути розроблений, масштабований та впроваджений незалежно від інших сервісів. Мікросервісна архітектура має переваги, такі як масштабованість, резильєнтність, гнучкість та можливість використовувати різні технології. Однак, вона також вимагає складнішого управління та розгортання.

Отже, аналізуючи отримані результати, ми прийшли до висновку, що вибір між монолітною та мікросервісною архітектурою залежить від конкретного проекту та його вимог. Монолітна архітектура підходить для невеликих та простих додатків, де простота розробки та управління мають переважаючу вагу. З іншого боку, мікросервісна архітектура рекомендується для складних додатків, які потребують масштабованості, гнучкості та стійкості.

1.4 HTTP/HTTPS – протоколи

Протокол передачі даних HTTP (Hypertext Transfer Protocol) використовується для комунікації між веб-клієнтом та веб-сервером. Він є основним протоколом для передачі гіпертекстових документів, таких як веб-сторінки, зображення, відео та інші ресурси через Інтернет.

HTTP базується на клієнт-серверній моделі, де клієнт (наприклад, веб-браузер) ініціює запит до сервера, а сервер надсилає відповідь з відповідними

даними. Кожен запит та відповідь HTTP має свою структуру та формат, які дозволяють передавати різноманітну інформацію.

HTTP використовує методи запитів, такі як GET, POST, PUT, DELETE та інші, для виконання різних дій над ресурсами на веб-сервері. Наприклад, GET-запит використовується для отримання ресурсу з сервера, а POST-запит для відправки даних до сервера для обробки.

Протокол HTTP також використовує заголовки, які містять додаткову інформацію про запит або відповідь. Заголовки можуть включати таку інформацію, як тип контенту, розмір файлу, кешування, аутентифікація та інші деталі, які допомагають забезпечити правильну обробку та передачу даних.

Протокол HTTP має безліч застосувань у сучасному Інтернеті, включаючи веб-переглядачі, веб-служби, API, мобільні додатки та інші веб-технології. Він є основою для взаємодії між користувачем та веб-сервером, дозволяючи отримувати веб-сторінки та передавати дані через мережу.

Важливо зазначити, що існує також розширена версія протоколу - HTTPS, який забезпечує шифрування та безпеку передачі даних. HTTPS використовує SSL або TLS протоколи для захисту конфіденційності та цілісності даних, що передаються між клієнтом та сервером. Це особливо важливо при передачі чутливої інформації, такої як паролі, особисті дані та фінансові відомості.

Узагальнюючи, HTTP - це протокол передачі даних, що забезпечує комунікацію між клієнтом та сервером у мережі Інтернет. Він використовується для отримання веб-ресурсів та передачі даних через мережу. HTTPS - розширена версія HTTP, яка забезпечує безпеку та шифрування передачі даних. Обидва протоколи використовуються широко і є основою для роботи багатьох веб-технологій та додатків.

Приклад взаємодії та принципова схема зображена на рисунку 1.1

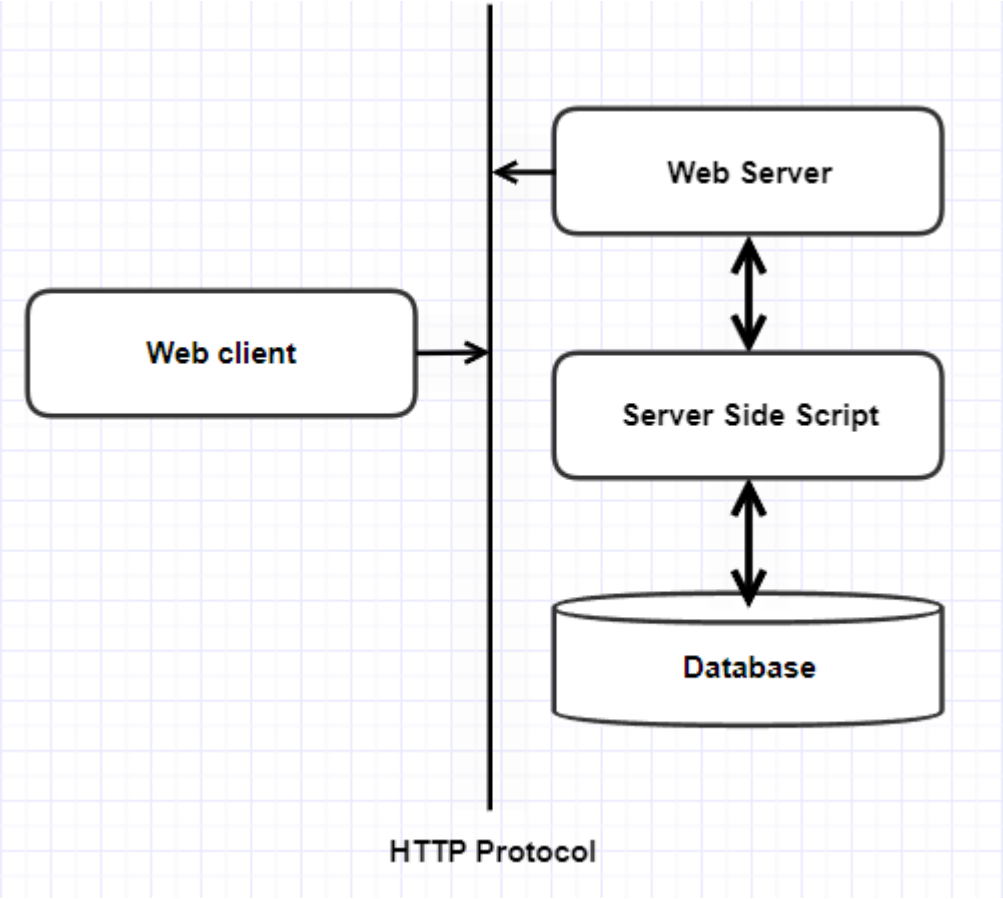

Рисунок 1.1 Архітектура HTTP

Захищений протокол передачі даних гіпертексту (HTTPS) є розширеною та безпечною версією протоколу HTTP. З метою забезпечення безпеки передачі даних, протокол HTTPS використовує шифрування. Це особливо важливо, коли користувачі передають конфіденційну інформацію, таку як дані для входу в банківський обліковий рахунок, електронну пошту або медичне страхування. HTTPS поєднує у собі HTTP і криптографічні протоколи SSL або TLS (Secure Sockets Layer або Transport Layer Security). Використання шифрування та інших механізмів захисту даних робить HTTPS надійним і безпечним засобом передачі інформації між клієнтом і сервером.

Одним з головних елементів безпеки HTTPS є шифрування даних. Коли користувач відправляє дані через HTTPS, вони перетинають мережу в зашифрованому вигляді, що унеможливлює просте перехоплення і зламання цих даних. Криптографічні протоколи SSL або TLS забезпечують встановлення захищеного каналу зв'язку між клієнтом і сервером, що гарантує цілісність, конфіденційність та автентичність переданих даних.

Крім шифрування, HTTPS використовує інші механізми безпеки, такі як сертифікати безпеки. Сертифікати використовуються для перевірки автентичності сервера, що дозволяє клієнтам впевнитися, що вони спілкуються саме зі справжнім сервером, а не зі зловмисником. Крім того, HTTPS може використовувати механізми контролю цілісності, які дозволяють перевірити, чи були дані під час передачі змінені чи пошкоджені.

У результаті використання HTTPS, передача конфіденційної інформації стає надійною та захищеною від несанкціонованого доступу. Це робить HTTPS незамінним інструментом для багатьох онлайн-сервісів, які працюють зі чутливими даними користувачів. Користувачі можуть бути впевнені в безпеці своїх особистих даних, а власники веб-сайтів мають засіб для забезпечення довіри та захисту своїх користувачів.

Загалом, HTTPS є незамінним інструментом для захисту конфіденційної інформації та забезпечення безпеки передачі даних у світі, де онлайн-загрози стають все складнішими. Використання HTTPS є стандартом безпеки для багатьох веб-сайтів, особливо тих, які обробляють фінансову, медичну чи особисту інформацію користувачів. Захищений протокол HTTPS забезпечує довіру, конфіденційність та захист даних у цифровій епохі.

Протокол TLS (Transport Layer Security) є криптографічним протоколом, який забезпечує безпеку з'єднання та захищену передачу даних через мережу. Він є наступником протоколу SSL (Secure Sockets Layer) і використовується для

шифрування комунікації між клієнтом і сервером, забезпечуючи конфіденційність, цілісність та автентичність даних. TLS забезпечує безпеку шляхом використання різних криптографічних протоколів та алгоритмів. Він включає механізми шифрування, які забезпечують конфіденційність даних шляхом перетворення їх у незрозумілу форму під час передачі. Крім того, TLS використовує механізми цифрових підписів для перевірки автентичності даних та забезпечення цілісності, що дозволяє виявляти будь-які зміни або втручання у передані дані. Протокол TLS вдосконалюється та оновлюється з метою запобігання спробам підриву безпеки з'єднання. Хоча TLS вважається дуже безпечним протоколом, все ж існують способи, які можуть стати на шляху до безпеки його використання.

Одним зі способів підриву безпеки TLS є атаки на приватні ключі. Приватний ключ використовується для розшифрування зашифрованих даних, тому його розкриття може дозволити зловмисникам отримати доступ до конфіденційної інформації. Зокрема, атаки типу "Man-in-the-Middle" можуть спотворювати спілкування між клієнтом і сервером, підміняючи їх приватні ключі та перехоплюючи передані дані.

Також існує ризик використання застарілих або незабезпечених версій протоколу TLS. Застарілі версії можуть містити вразливості, які можуть бути використані зловмисниками для отримання доступу до переданих даних. Тому важливо використовувати найновіші та безпечні версії TLS і забезпечувати їх постійне оновлення. Зрозуміло, що зловмисники намагаються знайти нові способи підриву безпеки TLS, тому постійне вдосконалення протоколу та застосування найкращих практик є необхідними для забезпечення безпеки комунікації.

Враховуючи всі ці фактори, TLS є надійним та ефективним засобом захисту передачі даних через мережу. Протокол TLS забезпечує безпеку та конфіденційність і є одним із основних компонентів безпечної комунікації в сучасному цифровому світі.

1.5 Загрози безпеки Веб-Застосунків.

Перш за все, загрози для інформаційної безпеки веб-застосунків є актуальною проблемою в сучасному цифровому світі. Вони виникають внаслідок різних умов і факторів, що створюють небезпеку для безпеки інформаційного простору. Ці загрози можуть мати різні ознаки і спрямованість, тому важливо класифікувати їх для більш ефективного аналізу та управління ризиками.

Однією з найважливіших загроз є загрози конфіденційності. Вони полягають у неправомірному доступі до інформації і можуть призвести до її витоку. Це особливо стосується інформації обмеженого доступу, такої як державна таємниця, комерційна таємниця або персональні дані. Загрози конфіденційності можуть виникати через людський фактор, такий як випадкове делегування привілеїв користувачам, а також через збої в програмному та апаратному забезпеченні. Розуміння цих загроз допоможе нам прийняти необхідні заходи для захисту конфіденційності веб-застосунків.

Іншим видом загроз є загрози порушення цілісності інформації. Ці загрози пов'язані з можливістю модифікації інформації, що зберігається в інформаційній системі. Зловмисники можуть змінювати дані, внесені користувачами або автоматизованими процесами, що може призвести до недостовірних або некоректних результатів. Захист цілісності є критичним аспектом для забезпечення надійності веб-застосунків та їх коректної роботи.

Загрози доступності також мають велике значення. Вони виникають, коли дії унеможливлюють або ускладнюють доступ до ресурсів інформаційної системи. Це може бути результатом хакерських атак, зламу мережевої інфраструктури або навмисного перевантаження системи. Загрози доступності можуть суттєво вплинути на функціональність веб-застосунків і спричинити втрату послуги для користувачів.

Важливо зазначити, що джерела загроз можуть бути розташовані як внутрішньо, так і зовні системи. Внутрішні загрози включають недбалість або зловживання співробітників, недостатньо ефективні політики безпеки та недоліки в системі управління доступом. Зовнішні загрози можуть бути представлені зловмисниками, хакерами, кіберзлочинцями або конкурентами, які намагаються зламати систему або завдати шкоди.

У своїй дипломній роботі я дослідив різні методи та засоби для виявлення, запобігання та реагування на ці загрози веб-застосунків. Я вивчив методи шифрування, контролю доступу, аудиту безпеки та моніторингу подій для забезпечення цілісності, конфіденційності та доступності інформації.

На основі результатів своєї роботи я рекомендую впроваджувати комплексні підходи до захисту веб-застосунків, включаючи технічні, організаційні та правові заходи. Політики безпеки та навчання персоналу є необхідними елементами для забезпечення ефективного управління загрозами. Також важливо систематично оновлювати програмне та апаратне забезпечення для усунення відомих вразливостей.

Усвідомлення загроз веб-застосунків та розуміння їхніх можливих наслідків є ключовим для побудови безпечного та надійного інформаційного простору. Запровадження відповідних заходів безпеки допоможе зменшити ризики порушення конфіденційності, цілісності та доступності веб-застосунків.

1.6 Дослідження видів кібератак

У сучасному цифровому світі, де інформаційні технології є невід'ємною частиною нашого життя, кібератаки стали серйозною загрозою для інформаційної безпеки. Кібератака може бути визначена як будь-який несанкціонований доступ до комп'ютерних систем, мереж або електронних пристроїв з метою отримання,

модифікації або руйнування даних, а також завдання шкоди системі або її користувачам.

Кібератака є складним процесом, в якому зловмисники використовують різноманітні техніки та інструменти з метою вторгнутися в інформаційну систему та виконати небажані дії. Ці дії можуть включати в себе крадіжку конфіденційної інформації, внесення змін у функціонал системи, відмову в обслуговуванні (DoS), розповсюдження шкідливого програмного забезпечення та багато іншого.

Серед видів атак Фішинг (Phishing) є одним з найпоширеніших видів кібератак, в якому зловмисники намагаються видурити конфіденційну інформацію (наприклад, паролі, номери кредитних карток) від користувачів, вдаючись до підробки відомих організацій або осіб. Зазвичай, зловмисники висилають електронні листи або повідомлення, що містять посилання на підроблені веб-сайти, де користувачі повинні ввести свої конфіденційні дані. Після отримання цих даних зловмисники можуть використовувати їх для крадіжки або маніпуляції.

Ще одним видом атак є Denial-of-Service або DoS (відмова в обслуговуванні). Цей тип кібератаки має на меті перенавантаження або перекриття ресурсів мережі або системи з метою заборонити доступ до них легітимним користувачам. Зловмисники можуть використовувати різні методи, такі як відправка великої кількості запитів, що перевантажують сервер, або експлуатація вразливостей мережевих протоколів. Результатом може бути відмова в обслуговуванні або значне зниження продуктивності системи. Іншим видом кібератаки є ін'єкція. Цей тип атаки полягає в тому, що зловмисник вставляє шкідливий код або вводить шкідливі дані в веб-застосунок або систему, з метою отримання несанкціонованого доступу або виконання небажаних операцій.

Прикладом є SQL-ін'єкція (SQL Injection) це вид кібератаки, коли зловмисник використовує недоліки веб-додатка для внесення шкідливого SQL-коду в запити до бази даних. Шкідливий код може призвести до розкриття конфіденційної інформації

або навіть отримання повного контролю над системою. Зловмисники можуть отримати доступ до цінних даних або внести зміни у базу даних, що може призвести до витоку інформації або порушення цілісності даних.

Аналізуючи види атак, варто згадати про соціальну інженерію. Соціальна інженерія передбачає маніпуляцію людьми з метою отримання неправомірного доступу до системи або конфіденційної інформації. Цей концепт наголошує, що найслабшою ланкою у системі безпеки є людина.

Розповсюдження шкідливого програмного забезпечення (Malware): Цей тип кібератаки включає різні види шкідливого програмного забезпечення, такі як віруси, черв'яки, троянські програми, шпигунське програмне забезпечення тощо. Шкідливе програмне забезпечення може бути розповсюджене через електронну пошту, веб-сайти, USB-пристрої або інші канали. Після інфікування комп'ютера воно може наносити шкоду шляхом крадіжки інформації, втрати даних, заморозки системи або здійснення інших небажаних дій.

Ще одним видом атаки є атака Man-in-the-Middle (MITM), яка представляє серйозну загрозу для безпеки комунікаційних систем. У цьому виді атаки зловмисник встановлює позицію "людини посередині" між двома комунікуючими сторонами, при цьому перехоплює та контролює всю передачу даних між ними.

Ботнет (Botnet) теж поширений вид атаки, представляють собою мережі комп'ютерів, які підконтрольовані зловмисниками за допомогою шкідливого програмного забезпечення. Комп'ютери, що належать до ботнету, називаються ботами. Зловмисники можуть використовувати ботнети для виконання різних кібератак, таких як DoS атаки або розповсюдження спаму. Керуючи ботнетом, зловмисники можуть використовувати потужність обчислювальних ресурсів ботів для своїх злочинних цілей.

У цьому розділі були розглянуті різні види кібератак, включаючи фішинг, віруси та черв'яки, DoS атаки, SQL-ін'єкцію, соціальний інжиніринг, ботнети та розповсюдження шкідливого програмного забезпечення. Кожен вид кібератаки має свої особливості та наслідки, і їх розуміння є важливим для розробки ефективних заходів забезпечення кібербезпеки. Застосування відповідних заходів захисту та свідоме користування інформаційними технологіями можуть допомогти уникнути чи пом'якшити наслідки кібератак.

Висновки за першим розділом

У першому розділі дипломної роботи було проведено огляд сучасного стану використання веб-застосунків та аналіз їх безпекових аспектів. В рамках огляду була розглянута архітектура World Wide Web, яка є основою для функціонування веб-застосунків. Було виявлено, що веб-застосунки можуть бути класифіковані за різними критеріями, такими як тип взаємодії користувача, функціональні можливості тощо.

Детально розглянуто архітектуру веб-застосунків, включаючи клієнт-серверну модель, використання протоколу HTTP/HTTPS для передачі даних між клієнтом та сервером. Було з'ясовано, що безпека веб-застосунків є надзвичайно важливим аспектом, оскільки вони мають доступ до конфіденційної інформації користувачів та можуть бути вразливими перед різними видами кібератак.

В рамках аналізу безпекових аспектів були розглянуті загрози безпеки веб-застосунків, серед яких можна виділити SQL-ін'єкції, DoS атаки, Man-in-the-Middle та інші. Важливо зазначити, що веб-застосунки повинні бути належним чином захищені від цих загроз, оскільки їхнє порушення може призвести до витоку конфіденційної інформації, порушення цілісності даних та інших негативних наслідків. Також було проведено дослідження видів кібератак, зокрема вивчено соціальну інженерію та атаку Man-in-the-Middle. Виявлено, що соціальна інженерія полягає у маніпуляції людьми з метою отримання неправомірного доступу до

системи або конфіденційної інформації, тоді як атака Man-in-the-Middle передбачає перехоплення та контроль комунікації між сторонами.

Загальною висновком є те, що безпека веб-застосунків є надзвичайно важливою та актуальною проблемою. Необхідно вживати заходів для захисту веб-застосунків від різноманітних загроз та атак, таких як використання безпечних протоколів, шифрування даних, перевірка автентичності користувачів та інші заходи безпеки. Далі у роботі буде проведений детальний аналіз та висвітлення способів захисту веб-застосунків від цих загроз.

РОЗДІЛ 2
АНАЛІЗ ЗАХИСТУ ВЕБ-ЗАСТОСУНКІВ З ВИКОРИСТАННЯМ БАГАТОФАКТОРНОЇ АВТЕНТИФІКАЦІЇ

2.1 Поняття автентифікації та керування доступу.

Автентифікація є важливим етапом процесу керування доступом, який передує авторизації. Вона забезпечує встановлення належності користувача до системи та підтверджує, що особа, яка намагається отримати доступ, є тим, за кого себе видає.

Керування доступом включає в себе сукупність заходів, спрямованих на визначення повноважень і прав доступу користувачів, а також контроль за їх дотриманням Правил розмежування доступу. Воно регулює, які ресурси системи можуть бути доступні для користувача, і забезпечує контроль за їх використанням.

Процес отримання доступу до інформації в системі може включати такі етапи:

- Ідентифікацію.
- Автентифікацію.
- Авторизацію.

Ідентифікація - це процедура розпізнавання користувача в системі шляхом використання ідентифікатора, такого як логін. Вона дозволяє суб'єкту системи (користувачу або процесу, що діє від імені користувача) повідомити своє ім'я за допомогою унікального параметра - ідентифікатора, який є відомим іншій стороні. Під час ідентифікації здійснюється порівняння введеного суб'єктом параметра з відомим іншій стороні. У разі успішної ідентифікації відбувається автентифікація.

Автентифікація може проводитись за допомогою логіна та пароля, де користувач вводить свої ідентифікаційні дані, які порівнюються зі значенням,

збереженим у захищеній базі даних. У разі успішної автентифікації користувачу надається доступ до системи.

Авторизація - це процес керування рівнями та засобами доступу до захищених ресурсів системи на основі ідентифікатора і пароля користувача або надання певних повноважень особі або програмі для виконання певних дій в системі обробки даних. Вона визначає, які дії користувач або програма можуть виконувати в системі.

Автентифікація та керування доступом мають важливе значення в різних сферах, включаючи:

- інтернет-сервіси;
- електронну пошту;
- соціальні мережі;
- банківські системи;
- корпоративні сайти;
- інтернет-магазини;

2.2 Багатофакторна автентифікація

Поняття багатофакторної автентифікації є важливим для забезпечення безпеки систем та захисту інформації. У своїй дипломній роботі ми пропонуємо проаналізувати цей термін та розглянути його в контексті забезпечення надійності контролю доступу до систем.

Багатофакторна автентифікація є розширеною формою автентифікації, де користувачу потрібно пред'явити більше одного "доказу механізму аутентифікації" для отримання доступу до інформації. Це забезпечує вищий рівень безпеки, оскільки зловмисникові стає набагато складніше отримати всі необхідні докази для автентифікації.

У системах багатофакторної автентифікації виділяються три головні фактори:

1.Фактор знання: Це щось, що користувач знає, наприклад, пароль або PIN-код. Цей фактор є одним з найпоширеніших, але він має свої недоліки, такі як можливість підбору пароля зловмисниками. Тому потрібно враховувати рівень складності паролів та регулярну зміну їх.

2.Фактор володіння: Це щось, що користувач має фізично, наприклад, пристрій аутентифікації або картка доступу. Цей фактор забезпечує вищий рівень безпеки, оскільки зловмисник повинен мати фізичний доступ до пристрою або картки.

3.Фактор властивості: Це щось, що є частиною користувача, наприклад, біометричні дані, такі як відбиток пальця, розпізнавання обличчя або голос. Біометрична автентифікація є одним з найбільш надійних методів, оскільки фізичні характеристики користувача унікальні для кожної особи.

Кожен з цих факторів має свої переваги та недоліки, і їх використання в системі залежить від вимог до безпеки та вартості впровадження.

Таким чином, у дипломній роботі можна детально розглянути поняття багатофакторної автентифікації, проаналізувати різні фактори та їх застосування в системах контролю доступу. Також варто розглянути питання безпеки та вартості впровадження різних методів автентифікації для вибору оптимального рішення для конкретної системи.

2.3 Двохфакторна автенифікація в веб-застосунках

Двофакторна аутентифікація (2FA) є ефективним методом забезпечення безпеки веб-застосунків. Вона додає додатковий рівень захисту до стандартного

процесу автентифікації, що базується на використанні лише логіну та паролю. Застосування 2FA відповідає сучасним вимогам безпеки, оскільки паролі можуть бути скомпрометовані через підбір або атаки віддаленого доступу.

Основна ідея 2FA полягає в тому, щоб користувач доводив свою ідентичність через два незалежні фактори аутентифікації. Зазвичай ці фактори поділяються на три категорії: щось, що ви знаєте (наприклад, пароль), щось, що ви маєте (токен або одноразовий код) та щось, що ви є (біометричні дані, такі як відбиток пальця або розпізнавання обличчя).

У процесі двофакторної автентифікації користувач, після введення свого логіну та паролю, повинен надати ще один фактор аутентифікації. Наприклад, це може бути одноразовий код, який надсилається на мобільний телефон користувача або генерується спеціальним додатком аутентифікації. Після успішної перевірки обох факторів користувач отримує доступ до веб-застосунку.

Використання двофакторної автентифікації суттєво знижує ризик несанкціонованого доступу до системи, оскільки зловмиснику потрібно мати не тільки правильний логін та пароль, але й фізичний доступ до другого фактора аутентифікації. Це значно ускладнює заволодіння аккаунтом навіть у випадку, коли пароль став відомим третій стороні.

Загалом, двофакторна автентифікація є важливим елементом безпеки веб-застосунків, оскільки вона забезпечує додатковий рівень захисту для користувачів. Реалізація 2FA вимагає розробки та впровадження відповідного програмного модуля, який би підтримував різні методи факторів аутентифікації та взаємодіював з користувачами під час процесу автентифікації.

2.4 Методи біометричної автентифікації

Оскільки біометрика залишається досить перспективним методом ідентифікації особи, варто розглянути його детальніше. Цей спосіб ідентифікації

особи є найстарішим, оскільки часто використовується у криміналістиці ще з XIX століття, але також є найновішим в галузі використання інформаційних технологій. Біометрія - це ідентифікація людини за унікальними, властивими тільки їй біологічними ознаками. Це може бути автоматизоване розпізнавання осіб на основі їх біологічних та поведінкових характеристик, що робить кожну людину унікальною.

Використання біометричної ідентифікації в галузі інформаційної безпеки є логічним рішенням. Біометрія є складним та ефективним способом ідентифікації особи, оскільки на основі біометричних характеристик дозволяє чітко ідентифікувати людину.

Головною перевагою біометричної ідентифікації є те, що біометричні ідентифікатори неможливо втратити чи забути, вони складно підробити, і кожна людина має унікальні біометричні ознаки. Точність біометричної ідентифікації наближена до ста відсотків.

Проте методи біометричної автентифікації також мають деякі недоліки. Реалізація системи біометричної ідентифікації вимагає значних вкладень, і деякі біометричні ідентифікатори можуть бути підроблені, наприклад, відбитки пальців, підпис або голос.

Незважаючи на ці недоліки, біометрична ідентифікація є найбільш надійним методом. Вона може бути використана у комплексній ідентифікації разом з іншими методами, такими як парольна ідентифікація. Також можна проводити ідентифікацію за декількома біометричними ідентифікаторами одночасно, що підвищує надійність ідентифікації. Біометричні ознаки поділяються на статичні (фізіологічні) та динамічні (психологічні).

Зазначені методи дозволяють ідентифікувати особу з високою ймовірністю. Крім того, існують інші біометричні ідентифікатори, такі як ідентифікація на основі

термограми обличчя, термограми долоні та геометрії вуха, які забезпечують точність ідентифікації особи близьку до 100 відсотків. Ідентифікація на основі ДНК також є можливою, але вимагає значного обсягу часу та складності.

2.5 Токени автентифікації

У комп'ютерних системах, токен автентифікації є засобом безпеки, що містить дані, необхідні для проходження сеансу входу та ідентифікації користувача. Такий токен дозволяє встановити ідентичність користувача, групи користувачів, привілеї та, у деяких випадках, певної програми.

Токен автентифікації можна розглядати як об'єкт, що включає в себе дані про захист процесу або потоку. Його використовують для прийняття рішень щодо безпеки та зберігання захищеної інформації, пов'язаної з системною сутністю. Хоча зазвичай токен використовується для представлення інформації про безпеку, він також може містити додаткові дані, які можуть бути приєднані під час створення токену. Токени можна дублювати без необхідності спеціальних привілеїв, наприклад, створити новий токен з меншим рівнем доступу для обмеження доступу запущеної програми. У Windows маркер доступу використовується, коли процес або потік намагаються взаємодіяти з об'єктами, що мають дескриптори безпеки (об'єкти, що захищаються). Він представлений системним об'єктом типу Token.

Маркер доступу генерується службою входу після того, як користувач успішно пройшов аутентифікацію на основі облікових даних, наданих користувачем. База даних аутентифікації містить необхідні дані для створення початкового токену сеансу входу, включаючи ідентифікатор користувача, ідентифікатор первинної групи, всі додаткові групи, до яких користувач належить, та іншу інформацію. Токен додається до початкового процесу, створеного для сеансу користувача, і успадковується наступними процесами, створеними в рамках цього сеансу. Кожного разу, коли такий процес намагається взаємодіяти з об'єктом, на якому використовується контроль доступу, Windows порівнює дані в дескрипторі

безпеки цього об'єкта з вмістом поточного ефективного токену доступу. На основі цього порівняння приймається рішення про дозвіл або заборону доступу, а також визначається, які операції (читання, запис/зміна тощо) можуть виконуватися програмою, яка намагається отримати доступ.

У контексті веб-технологій особливо популярними є JWT-токени. JSON Web Token (JWT) - це об'єкт JSON, який відповідає стандартам RFC 7519. Він є безпечним способом передачі інформації між двома сторонами. JWT складається з заголовка (header), який містить загальну інформацію про токен, з тілом (payload), яке містить корисну інформацію, таку як ідентифікатор користувача і його роль, і з підпису (signature). JWT можна легко застосовувати, оскільки він є самодостатнім і містить усю необхідну інформацію. Він часто використовується для автентифікації API, передаючи його через HTTP-заголовок або URL .

Цей підхід є універсальним і підтримується різними мовами програмування, такими як .NET, Python, Node.js, Java, PHP, Ruby, Go, JavaScript та Haskell. Завдяки своїй гнучкості та надійності JWT знаходить застосування в багатьох сценаріях .

Висновки за другим розділом

У другому розділі були проведені дослідження, що стосувалися методів захисту веб-застосунків, а також збереження цілісності, конфіденційності та доступності інформації за допомогою різних засобів і методів, таких як багатофакторна, двофакторна та біометрична автентифікація. Були визначені основні методи та засоби багатофакторної автентифікації, натхненні досвідом провідних компаній у різних галузях, таких як розробники програмного та апаратного забезпечення для веб- та мобільних додатків, державний сектор та банківська сфера, де вже застосовуються такі інструменти.

Вважаємо, що багатофакторна автентифікація може значно зменшити ймовірність крадіжки особистих даних в Інтернеті, оскільки для здійснення

шахрайства необхідно мати більше, ніж просто знання пароля потенційної жертви. При виборі факторів або методів автентифікації для системи, слід спиратися на рівень захищеності, вартість реалізації системи та забезпечення мобільності користувачів.

РОЗДІЛ 3
РОЗРОБКА ТА ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ З ВИКОРИСТАННЯ ДВОХФАКТОРНОЇ АВТЕНТИФІКАЦІЇ

3.1 Вимоги до системи та вибір технологій стеку.

У даній дипломній роботі розглядається модель веб-застосунка, яка використовує двофакторну автентифікацію з метою забезпечення конфіденційності інформації. В основу веб-застосунка була взята мікросервісна архітектура, що сприяє зменшенню зв'язності між компонентами системи, підвищенню стійкості до навантаження та полегшенню підтримки програмного комплексу.

Для реалізації описаних в роботі алгоритмів створимо веб-проект з використанням Python та Django. Це досить популярні технології, вони широко використовуються сьогодні, тому така реалізація буде дуже актуальною.

Окрім того, реалізуємо наш додаток у формі CTF – завдання. Ми використаємо методи двохфакторної авторизації для доступу до деякої секретної інформації – в CTF зазвичай такою інформацією є FLAG.

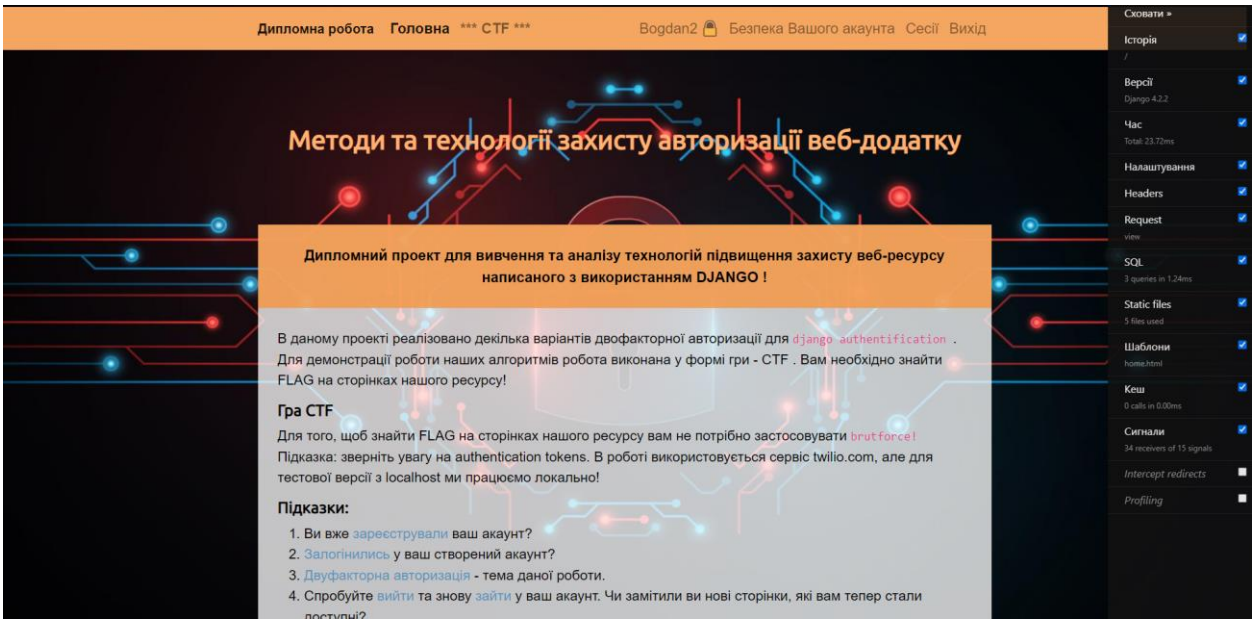

Рис. 3.0 Головна сторінка CTF - завдання

Модель програмного застосунка складається з 4 модулів та модуля автотестів. Основні модулі включають: сервер автентифікації, сховище тимчасових паролів, сервер бази даних та сервіс логування.

Перед вибором технологій для веб-застосунка потрібно визначитися з операційною системою для серверів. При виборі ОС важливими аспектами є безпека даних, стійкість системи, рівень відповідності до потреб технології та зручність інтеграції з іншими технологіями.

Варто розглянути дві сучасні та популярні ОС - Windows і Linux. Windows пропонує багато функцій, за які необхідно платити, тоді як Linux є відкритим кодом і безкоштовним, але вимагає більше уваги до налаштування з боку системного адміністратора.

Розглянуто сервер як програмне забезпечення для вирішення завдань апаратного забезпечення. Обладнання може бути різним - від одного хост-комп'ютера, підключеного до внутрішньої мережі, до високотехнологічного масиву зовнішніх апаратних служб у хмарі.

Вибір між Windows і Linux для живлення сервера залежить від потреб бізнесу, рівня IT-досвіду та програмного забезпечення, яке необхідно встановити. Це також може залежати від постачальника послуг, з яким ви плануєте співпрацювати.

Windows Server, розроблений Microsoft з комерційною метою, має свої переваги. Платіть за послуги і отримуйте кращу підтримку, ніж від Linux з відкритим кодом, який підтримується спільнотою. Підтримка клієнтів Windows, очікувано, здійснюється через Microsoft та їхніх партнерів. Програми Windows, такі як Outlook, Office і т.д., безпосередньо інтегруються з серверами Windows. Якщо ви користуєтесь програмним забезпеченням та послугами Windows, то має сенс запускати їх на рідній платформі.

Якщо ви використовуєте сервер бази даних Microsoft SQL, він не працюватиме на сервері Linux, якщо ви не встановите емулятор Windows. Для цього потрібно придбати копію Windows та окремо програмне забезпечення бази даних. Сервер Windows часто розглядається як готове рішення, яке швидко та легко налаштовується. Якщо ви бажаєте отримати віддалений доступ до робочого столу з інтуїтивно зрозумілим графічним інтерфейсом, Windows надає цю можливість без потреби в програмуванні командного рядка, який є необхідним для Linux.

Linux - це операційна система з відкритим кодом (ОС) та платформа IT-інфраструктури, яка дозволяє використовувати різні дистрибутиви, такі як Ubuntu, Fedora та CentOS. Вихідний код доступний для змін та оновлення функцій програмного забезпечення. Користувачі можуть переглядати та редагувати вихідний код для модифікації функціональності або виправлення помилок.

Linux, як відкрита система, є безкоштовним. Веб-хостинг-провайдерам потрібно платити лише за технічну підтримку для установки та обслуговування, якщо це необхідно. Провайдери бізнес-серверів не накладають додаткових витрат на замовників. З іншого боку, при використанні серверів Windows компанія зазвичай повинна платити за операційну систему та ліцензію на періодичне використання.

Пакет сервера Windows, що розроблений компанією Microsoft з метою комерційного отримання прибутку, має свої переваги. Заплачуйте за послуги даного пакету і отримуйте кращу підтримку, порівняно з відкритим джерелом Linux, який розробляється та підтримується спільнотою. Підтримка клієнтів Windows здійснюється через Microsoft та їхніх торгових посередників, що є очікуваним. Ваші програми Windows (наприклад, Outlook, Office) безпосередньо інтегруються з серверами Windows. Якщо ви використовуєте програмне забезпечення та послуги Windows, то має сенс запускати їх на рідній платформі.

У випадку використання сервера бази даних на основі Microsoft SQL, він не буде працювати на Linux-сервері, якщо ви не встановите емулятор Windows. Для

цього потрібно окремо придбати копію Windows і програмне забезпечення бази даних. Сервер Windows часто розглядається як повноцінне рішення, яке можна швидко і легко налаштувати. Якщо ви бажаєте отримати віддалений доступ до робочого столу з інтуїтивно зрозумілим графічним інтерфейсом, Windows пропонує це без необхідності програмування командного рядка, яке зазвичай вимагається в Linux.

Linux - це операційна система з відкритим кодом (ОС) та IT-інфраструктурна платформа, яка дозволяє розповсюджувати такі дистрибутиви, як Ubuntu, Fedora та CentOS. Вихідний код Linux доступний для редагування та оновлення функціональності програмного забезпечення розробниками. Користувачі можуть звертатися до вихідного коду для редагування функцій або виправлення помилок.

Linux, завдяки своєму відкритому коду, є безкоштовним. Веб-хостинговим компаніям потрібно платити лише за технічну підтримку для встановлення та обслуговування програмного забезпечення (якщо це необхідно). Постачальники бізнес-серверів не мають передавати додаткові витрати на своїх клієнтів. З іншого боку, за використання серверів Windows компанія зазвичай мусить платити за операційну систему та ліцензію на періодичне використання.

Linux є платформою, яка легко сумісна з іншими програмними продуктами з відкритим кодом і забезпечує швидкий та простий інтерфейс. Користувачі Linux можуть запускати програми Windows, але для цього потрібно придбати програмне забезпечення для інтерфейсу та оплатити ліцензію Windows. Це особливо корисно, якщо ви маєте застарілі програми, які вимагають емуляції Windows.

Сервери Linux та програми, що працюють на них, зазвичай вимагають менше комп'ютерних ресурсів, оскільки вони оптимізовані для ефективної роботи. Більш того, розробники можуть змінювати сервери та програмне забезпечення Linux без перезавантаження, що не є можливим у середовищі Windows. Сервери Microsoft Windows часто сповільнюються при роботі з багатьма базами даних і мають

більший ризик виникнення збоїв. З точки зору безпеки, Linux є більш надійним в порівнянні з Windows. Хоча жодна система не є повністю захищеною від атак та зловмисного програмного забезпечення, Linux зазвичай стає менш привабливою мішенню для хакерів. Оскільки Windows є популярною платформою для багатьох програм, хакери частіше спрямовують свої атаки на цю систему.

Варто зазначити, що Linux був розроблений з великою увагою до безпеки, як глобальна мережа Internet. Тому у питаннях безпеки Linux завжди відрізняється від багатьох сучасних операційних систем, включаючи комерційні версії Unix.

В сучасних ОС існують два основних підходи до забезпечення безпеки: перший полягає в додатковому захисті слабкого захищеного ядра операційної системи за допомогою фаєрволу та інших захисних механізмів, а другий - в інтеграції фаєрволу та інших засобів захисту на рівні ядра системи.

Перший підхід до захисту, який використовує компанія Microsoft, був проаналізований на основі останніх версій Windows NT. У той час, як розробники Linux вибрали другий підхід, вбудувавши код брандмауера безпосередньо в ядро Linux, починаючи з версії 2.0. Це призвело до створення потужної інтегрованої системи захисту.

При розробці системи захисту для будь-якої операційної системи, важливо розуміти, що створити повністю безпечну комп'ютерну систему на практиці неможливо. Можна лише створити додаткові бар'єри для зловмисників, які намагаються проникнути в систему. Обсяг та якість засобів захисту, які використовуються в Linux, залежать від конкретної галузі використання. Крім того, важливо знайти баланс між кількістю встановлених засобів захисту та зручністю використання для звичайних користувачів при управлінні Linux-системою.

Для оцінки ефективності системи захисту ОС Linux необхідно детально розібратися з можливими загрозами, які можуть бути реалізовані зловмисниками на

рівні операційної системи в різних ситуаціях. Один з підходів до контролю захисту полягає у використанні контрольних сум для основних системних і конфігураційних файлів, які потім порівнюються з еталонними значеннями, збереженими в спеціальній базі даних. Це дозволяє адміністратору контролювати будь-які зміни в системі.

Одним з варіантів є використання Tripwire, який можна розмістити на недоступному для запису гнучкому диску і запускати його щодня. Також важливо забезпечити конфіденційність даних шляхом збереження їх у зашифрованому вигляді на дисках. Для цього в Linux використовуються криптографічні файлові системи CFS (Cryptographic File System) і TCFS (Transparent Cryptographic File System), які забезпечують повне шифрування файлової системи. Мережевий захист є дуже важливим аспектом у сучасних мережевих технологіях. Часто саме мережеві атаки є найуспішнішими. Тому в Linux приділяється значна увага мережевій безпеці. У Linux використовуються різні ефективні засоби для забезпечення мережевої безпеки:

- захищена оболонка ssh для унеможливлення атак, що мають на меті отримання паролів шляхом аналізу протоколів;
- програми tcp_wrapper для обмеження доступу до різних служб вашого комп'ютера;
- мережеві сканери для виявлення вразливостей вашого комп'ютера;
- демон tcpd для виявлення спроб сканування портів з боку зловмисників (додатково рекомендується регулярно переглядати файли системного журналу);
- система шифрування PGP (Pretty Good Privacy);
- програма stelnet (захищена версія відомої програми telnet);
- програма qmail (захищена доставка електронної пошти);
- програма ipfwadm для налаштування міжмережевих екранів (firewall);

- режим перевірки паролів вхідних з'єднань для систем, що дозволяють підключення через зовнішні комутовані лінії зв'язку або локальну мережу.

В проекті передбачається використання веб-сервера Nginx як операційної системи. Nginx є проксі-сервером та веб-сервером з відкритим вихідним кодом. Цей сервер має різні версії для різних операційних систем, таких як FreeBSD, GNU/Linux, Solaris, Mac OS X і Microsoft Windows. Розробка Nginx почалася у 2002 році Ігорем Сисоєвим для компанії Rambler, і вона продовжується й до цього часу. Уосени 2004 року вийшов перший публічний реліз. З 2011 року розробкою програми займається компанія Nginx Inc., заснована самим Ігорем Сисоєвим, і вона розвиває як безкоштовні, так і комерційні версії продукту.

Головні функції цього веб-сервера включають можливість виступати як HTTP-сервер і проксі-сервер для протоколів IMAP/POP3. Він надає такі особливості як обробка статичних запитів, індексні файли, автоматичне створення списку файлів, кешування дескрипторів відкритих файлів, прискорене проксіювання з можливістю кешування, підтримка прискореного FastCGI та memcached серверів, простий розподіл навантаження та відмовостійкість, модульність, фільтри, gzip, byte-ranges (докачка), chunked відповіді, HTTP-аутентифікація, SSI-фільтр, паралельне виконання вкладених запитів на одній сторінці, підтримка SSL, експериментальна підтримка вбудованого Perl та HTTP/2. У режимі IMAP/POP3-проксі сервера веб-сервер може перенаправляти користувачів на відповідні бекенди IMAP/POP3 з використанням зовнішнього HTTP-сервера аутентифікації. Він також підтримує просту аутентифікацію (LOGIN, USER/PASS), SSL і StartTLS.

Для забезпечення збереження інформації користувачів, ми використовуємо SQL та систему управління базами даних MySQL, що дозволяє ефективно керувати даними. SQL (Structured Query Language — мова структурованих запитів) — це декларативна мова програмування, яка використовується для взаємодії користувача з базами даних. Вона дозволяє формувати запити, оновлювати та керувати

реляційними базами даних, створювати та модифікувати схему бази даних, а також контролювати доступ до неї. SQL сам по собі не є системою керування базами даних або окремим програмним продуктом. Відмінності між SQL і дійсними мовами програмування (наприклад, C або Pascal) полягають у тому, що SQL може формувати інтерактивні запити або використовуватись в прикладних програмах як інструкції для керування даними. Більше того, стандарт SQL включає функції для визначення змін, перевірки та захисту даних.

SQL - це мова програмування, яка використовується для виконання запитів і змін до бази даних, а також для керування базами даних. Багато баз даних підтримують SQL з розширеннями до стандартної мови. Для управління даними всередині системи потрібна Система Управління Базами Даних (СУБД). пошуку інформації в базах даних, а також контролю доступу до цих даних.

Найбільш поширеними системами управління базами даних (СУБД) для SQL є MySQL та PostgreSQL. Крім них, на ринку також присутній Microsoft SQL Server (MSSQL). Зокрема, MySQL є лідером серед цих систем та користується популярністю серед таких великих компаній, як Apple, Amazon, Cox Communications, WordPress, Nokia, NASA, Google та Digg.

MySQL вважається ефективним рішенням для малих і середніх застосувань. Код сервера MySQL може бути компільований на різних платформах. Найбільш повні можливості сервера виявляються в UNIX-системах, оскільки вони підтримують багатопоточність, що сприяє підвищенню продуктивності системи в цілому.

Після затвердження узгоджень з операційною системою, веб-серверами та сервером системи управління базами даних (СУБД), перейдемо до виконання вищих рівнів реалізації програмного комплексу. Серверна логіка та автоматизоване тестування системи будуть розташовані на сервері, відповідальному за автентифікацію користувачів. На сьогоднішній день існує широкий вибір серверних

мов програмування та платформ, таких як Node.js, PHP, Java, Python, Scala, Go. Однак, найбільш логічним та простим рішенням є PHP, який становить 75% усіх веб-проектів, розроблених у 2020 році. PHP - потужна мова програмування та інтерпретатор, що взаємодіє з веб-сервером як модуль або як незалежний бінарний додаток CGI. PHP може звертатися до файлів, виконувати різні команди на сервері і встановлювати мережеві з'єднання. Через це всі скрипти, що виконуються на сервері, потенційно можуть бути небезпечними. Основна мета розробки PHP полягала у створенні більш безпечної мови програмування (у порівнянні з Perl, C) для написання CGI-додатків. Завдяки ряду налаштувань під час компіляції та динамічних налаштувань програми, завжди можна знайти оптимальну комбінацію свободи дій і безпеки.

З огляду на те, що існує різноманітність використання PHP, наявні безліч опцій, що керують його поведінкою. Широкий спектр варіантів дозволяє використовувати PHP для різних цілей, однак це також означає, що певні комбінації опцій можуть призводити до небезпечних налаштувань сервера. Гнучкість конфігурації PHP може бути порівняна з гнучкістю самої мови. PHP може бути використаний для створення повноцінних серверних додатків, які використовують можливості операційної системи, доступні для вказаного користувача. Також він може використовуватися як простий файл, який підключається на сервері з мінімальним ризиком у контрольованому середовищі. Безпека та якість налаштування цього середовища в значній мірі залежить від кваліфікації PHP-розробника.

PHP є важливим компонентом для забезпечення безпеки сервера, оскільки PHP-скрипти можуть працювати з файлами і каталогами на диску. Це означає, що існують конфігураційні налаштування, які контролюють доступ до файлів і операції, що можуть бути виконані над ними. Важливо бути обережним, оскільки будь-який файл з правами на читання ("world-readable") може бути доступним для

будь-якого, хто має доступ до файлової системи. Початково PHP був розроблений з повноправним доступом до файлової системи, що дозволяло скриптам PHP читати системні файли, такі як /etc/passwd, керувати мережевими з'єднаннями або відправляти завдання на принтер. Тому важливо переконатись, що файли, з якими ми працюємо, є тими, які ми очікуємо.

PHP постійно оновлюється та вдосконалюється, як будь-яка інша велика система. Кожна нова версія містить численні зміни, які стосуються підвищення безпеки, розширення конфігураційних можливостей та забезпечення стабільності системи. Рекомендується регулярно оновлювати PHP і бути в курсі останніх змін у нових версіях.

Для збереження OTP (одноразового пароля) та швидкого доступу до нього можна використовувати Redis. Redis є відкритою системою зберігання даних в пам'яті, яка може використовуватись як база даних, кеш-пам'ять або брокер повідомлень. Він підтримує різноманітні структури даних, такі як рядки, хеші, списки, набори, відсортовані набори, растрові зображення, гіперлоги, геопросторові індекси та потоки. Redis має вбудовану реплікацію, підтримку сценаріїв Lua, винесену заміну найменш нещодавно використаних елементів (LRU), транзакції та різні рівні стійкості на диску. Він також забезпечує високу доступність через Redis Sentinel та автоматичне розділення з Redis Cluster.

Redis написаний мовою ANSI C і працює на більшості POSIX-сумісних систем, таких як Linux та BSD. Redis розроблений та протестований найбільше на Linux та OS X, тому рекомендується використовувати Linux для розгортання системи. Хоча Redis може працювати на інших системах, таких як Solaris або SmartOS, підтримка на них може бути обмеженою. Офіційної підтримки для версій Windows немає.

Для досягнення високої продуктивності Redis працює з данними в оперативній пам'яті. Залежно від випадку використання, дані можуть бути

збережені на диск шляхом періодичного збереження або додаватися до журналу для кожної команди. Ви можете вимкнути збереження, якщо вам потрібен лише кеш-пам'ять з багатофункціональним керуванням. Redis також підтримує асинхронну реплікацію master-slave з швидкою неблокуючою першою синхронізацією, автоматичне повторне підключення з частковою ресинхронізацією при відновленні з розщеплення.

Таким чином, сучасний веб-додаток та забезпечення безпеки потребують використання широкого спектру технологій, які взаємодіють між собою.

3.2 Деякі практичні аспекти функціонування двохфакторної автентифікації

Виконаємо реалізацію системи. Основним модулем, відомим як ядро системи, буде API сервера автентифікації, який надасть можливість пройти автентифікацію з використанням двох факторів. Перший фактор - це пароль користувача, а другим - верифікація через пошту.

Перший крок у впровадженні системи - створення таблиці користувачів у базі даних MySQL. Для реалізації міграції PHP коду в SQL ми скористаємося гнучкою системою міграцій, доступною у згаданому фреймворку.

Другим кроком буде створення функціоналу реєстрації користувача та збереження його даних у базі даних. З метою підвищення спостережності за системою, кожен крок буде логуватися в окремий файл на сервері.

Процес автентифікації буде складатися з двох етапів:

- Перевірка логіну та пароля користувача.
- Перевірка OTP-пароля з листа, який надісланий на пошту.

Перевірка логіну та пароля користувача є базовим етапом будь-якої системи безпеки. Інформація про користувача зберігається у захешованому вигляді, а хеш

пароля формується при реєстрації з використанням алгоритму хешування bcrypt. Bcrypt є адаптивною криптографічною функцією для безпечного зберігання паролів, розробленою Нільсом Провосом і Девідом Маз'єрсом. Ця функція базується на шифрі Blowfish, який був представлений на конференції USENIX у 1999 році.

У даному веб-застосунку верифікація паролю користувача відбувається наступним чином. Оскільки система побудована на архітектурі REST і сервер автентифікації взаємодіє з іншими модулями, як внутрішніми, так і зовнішніми, через протокол HTTP, а дані передаються у форматі JSON, для коректного процесу необхідно відправити POST-запит через HTTP на ендпоінт логіну. На рисунку 3.1 показано симуляцію цього процесу за допомогою утиліти POSTMAN."

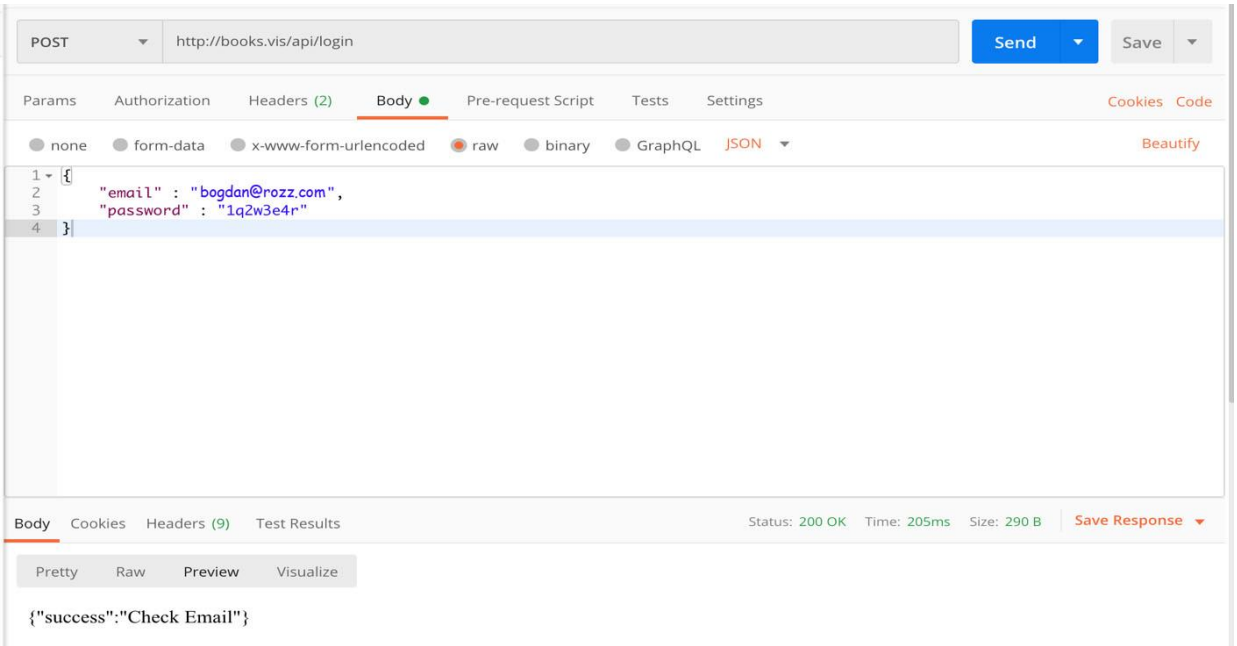

Рисунок 3.1 – Запит першого етапу

Після отримання запиту, дані проходять на сервер і можуть мати один з трьох можливих результатів. Перший варіант - дані не відповідають правилам валідації

запиту на сервері (наприклад, некоректна електронна адреса або пароль менше 8 символів). Базова валідація використовується для відсіювання фейкових або неправдивих даних ще до проведення криптографічної перевірки. У такому випадку сервер поверне відповідь з HTTP-кодом 422 Unprocessable Entity. Цей код означає, що сервер розуміє тип запиту, але не може обробити вміщені в нього інструкції, оскільки вони не відповідають вимогам синтаксису.

На рисунку 3.2 зображено приклад запиту з невалідною поштовою адресою.

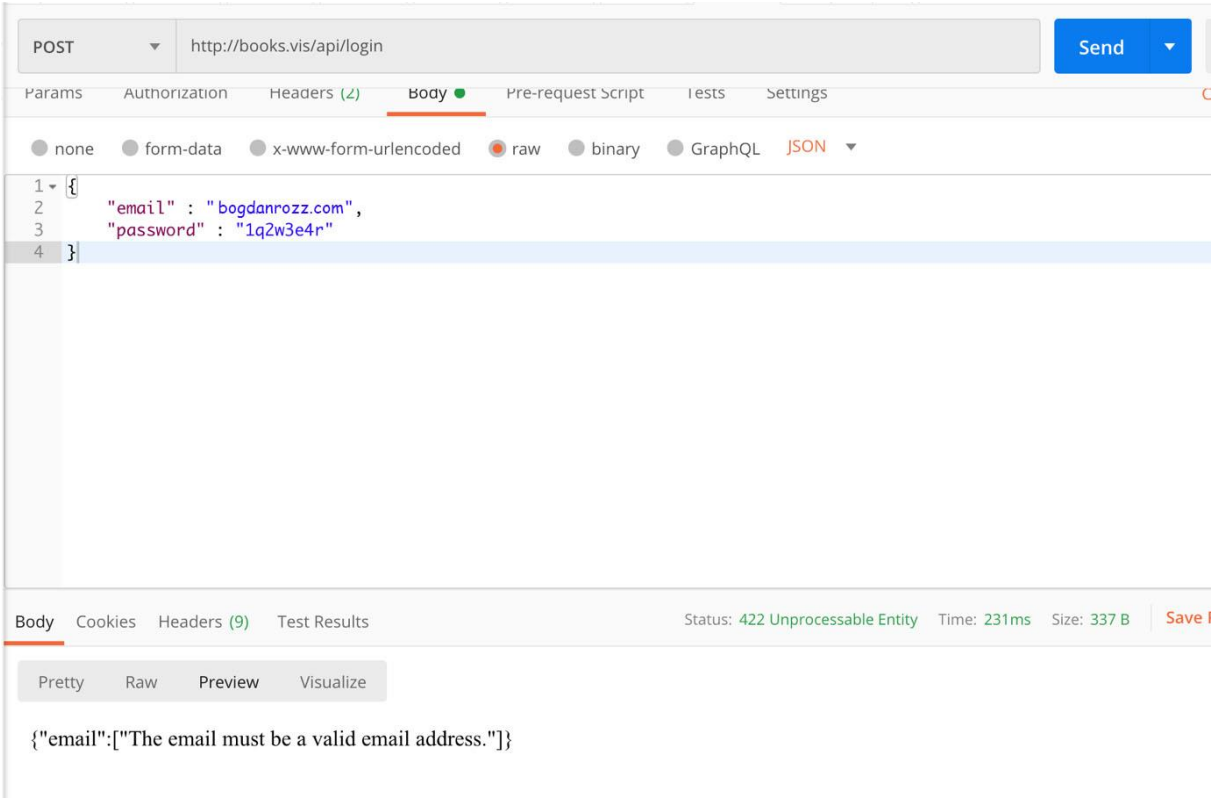

Рисунок 3.2 – Запит першого етапу з невалідними даними

Наступний варіант − це дані коректні, але під час перевірки пароля виявлено неспівпадіння між хешем введеного пароля та хешем пароля, що зберігається в базі даних. В результаті, сервер автентифікації повертає HTTP-код 401. Цей код вказує на те, що запит не був авторизований через недостатні облікові дані для доступу до цільового ресурсу.

Даний сценарій наглядно продемонстровано на рисунку 3.3.

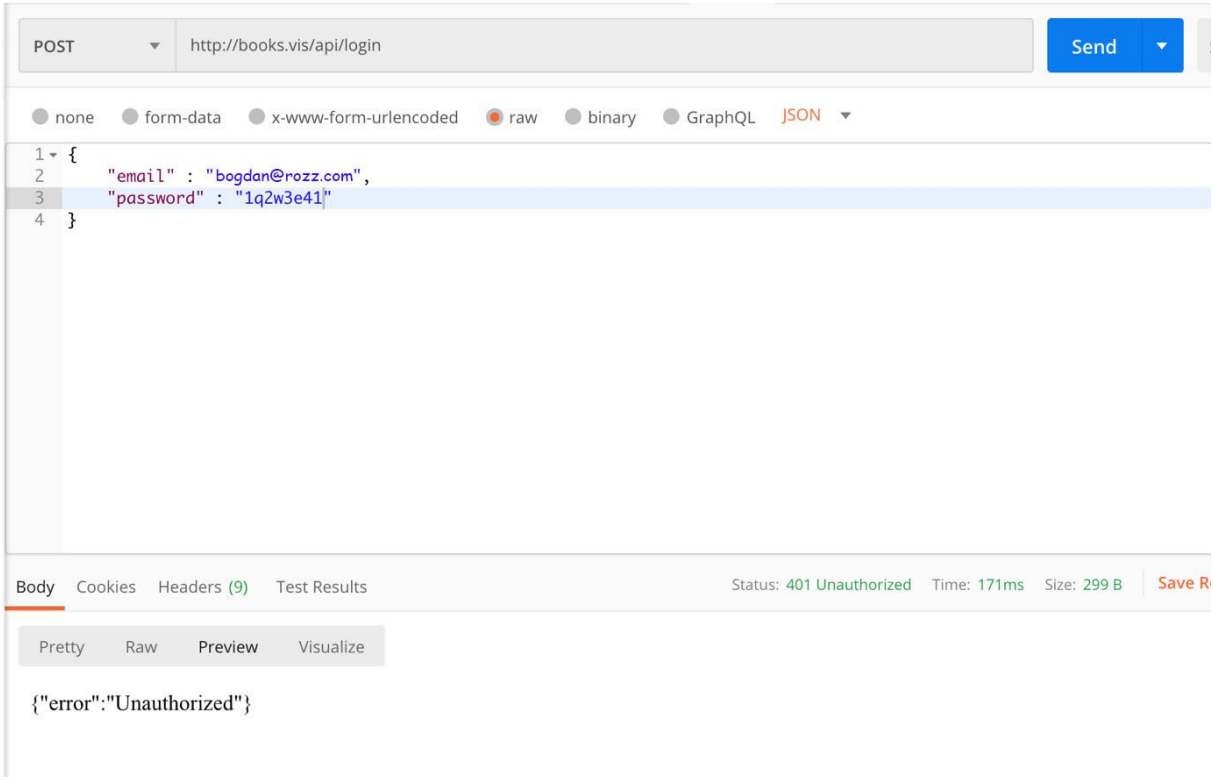

Рисунок 3.3 – Запит першого етапу з неправильним паролем

У третьому варіанті першого етапу автентифікації, перевіряється валідність даних та правильність пароля. Якщо дані є валідними, то генерується одноразовий пароль (OTP), який стане фактором для другого етапу автентифікації. Цей OTP надсилається на електронну пошту користувача. В додатку А показано алгоритм формування та збереження такого пароля. Для формування тимчасового пароля використовується алгоритм хешування md5. MD5 (Message Digest 5) є 128-бітним алгоритмом хешування, розробленим професором Рональдом Л. Рівестом у 1991 році. Він використовується для створення "відбитків" або "дайджестів" повідомлень будь-якої довжини. MD5 був створений як заміна недосконалого алгоритму MD4. Його опис наведений у RFC 1321. Згідно з RFC 6151, починаючи з 2011 року, цей алгоритм вважається недостатньо надійним, але він все ще добре підходить для створення тимчасових псевдовипадкових даних. У якості вхідного параметра використовується поточний Unix-час. Після формування OTP він

зберігається в Redis-сховищі на 900 секунд і надсилається на електронну пошту. Приклад листа показано на рисунку 3.4.

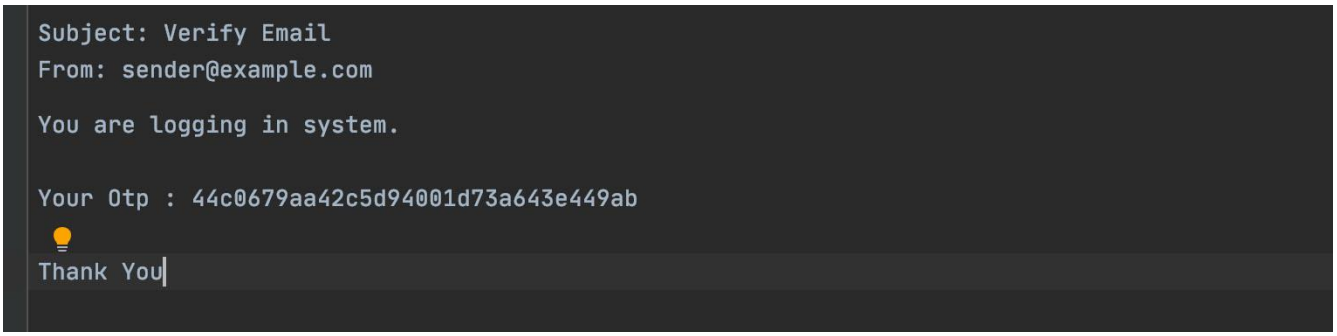

Рисунок 3.4 – Лист з OTP паролем

Другий крок в процесі автентифікації передбачає перевірку користувача за допомогою електронної пошти та видачу йому JWT-токена для подальшого доступу до системи. Користувач повинен надіслати OTP-пароль та електронну пошту протягом 900 секунд після попереднього етапу, як показано на малюнку 3.5. Аналогічно до першого кроку, запит має бути відправлений на HTTP-ендпоінт за допомогою методу POST, а тіло запиту повинно бути у форматі JSON.

При правильних даних, сервер надсилає відповідь зі статусом 200 та JWT-токеном, який дозволить ідентифікувати та авторизуватися в системі. У випадку проблем з валідністю або коректністю даних, отримуємо аналогічний до першого етапу код та повідомлення про помилку - 422 і 401. Код, який виконує надання та верифікацію другого етапу, можна знайти в Додатку Б.

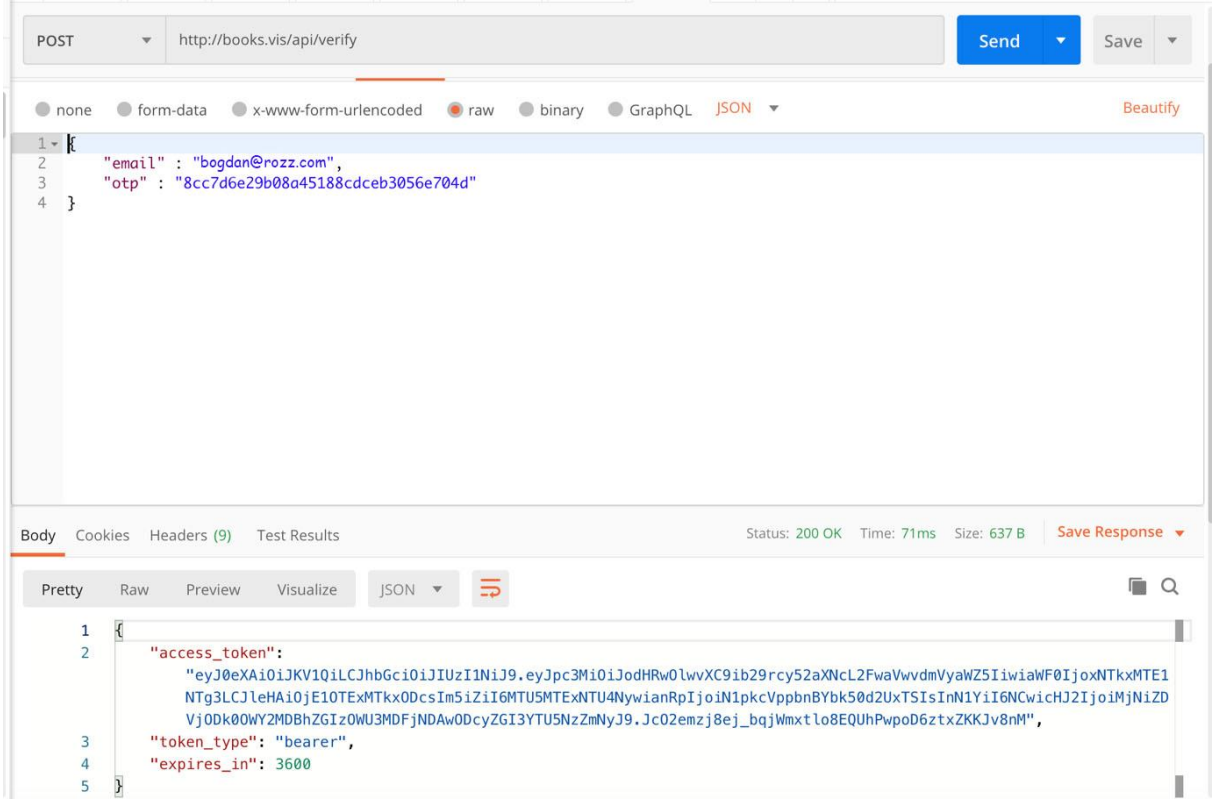

Рисунок 3.5 – Запит з отримання JWT-токену

Процес формування JWT- токена , як було зазначено в другому розділі складається з трьох частин:

- Формування заголовку.
- Формування корисного навантаження.
- Формування криптографічного підпису токену.

Формування заголовку - це простий процес, в якому ми вказуємо службову інформацію: токен "JWT" і алгоритм хешування, який ми використовуємо - HMAC SHA256.

HMAC (Hash-based message authentication code, код аутентифікації повідомлень на основі хеш-функцій) - це механізм перевірки цілісності інформації. Він дозволяє гарантувати, що дані не були змінені сторонніми особами під час передачі або зберігання в ненадійному середовищі. HMAC використовує MAC

(Message Authentication Code), який описаний в RFC 2104, і є стандартом, що описує спосіб обміну даними та перевірки цілісності за допомогою секретного ключа.

Формування корисного навантаження - це друга частина токену, в якій містяться інформація про користувача, час життя токену і емітент токену.

Остання і найважливіша частина JWT - токену - це підпис. Він представляє собою хеш-функцію, яка обчислюється на основі попередніх двох частин токену, закодованих у форматі base64, і секретного ключа, який зберігається на сервері автентифікації.

Отже, система автентифікації включає кілька компонентів та модулів, які можна розширювати для задоволення потреб інформаційної безпеки.

3.3 Реалізація двофакторної автентифікації у веб-додатку Django

3.3.1 Налаштування проекту Django

Для реалізації описаних в роботі алгоритмів створимо веб-проект з використанням Python та Django. Це досить популярні технології, вони широко використовуються сьогодні, тому така реалізація буде дуже актуальною.

Окрім того, реалізуємо наш додаток у формі CTF – завдання. Ми використаємо методи двохфакторної авторизації для доступу до деякої секретної інформації – в CTF зазвичай такою інформацією є FLAG.

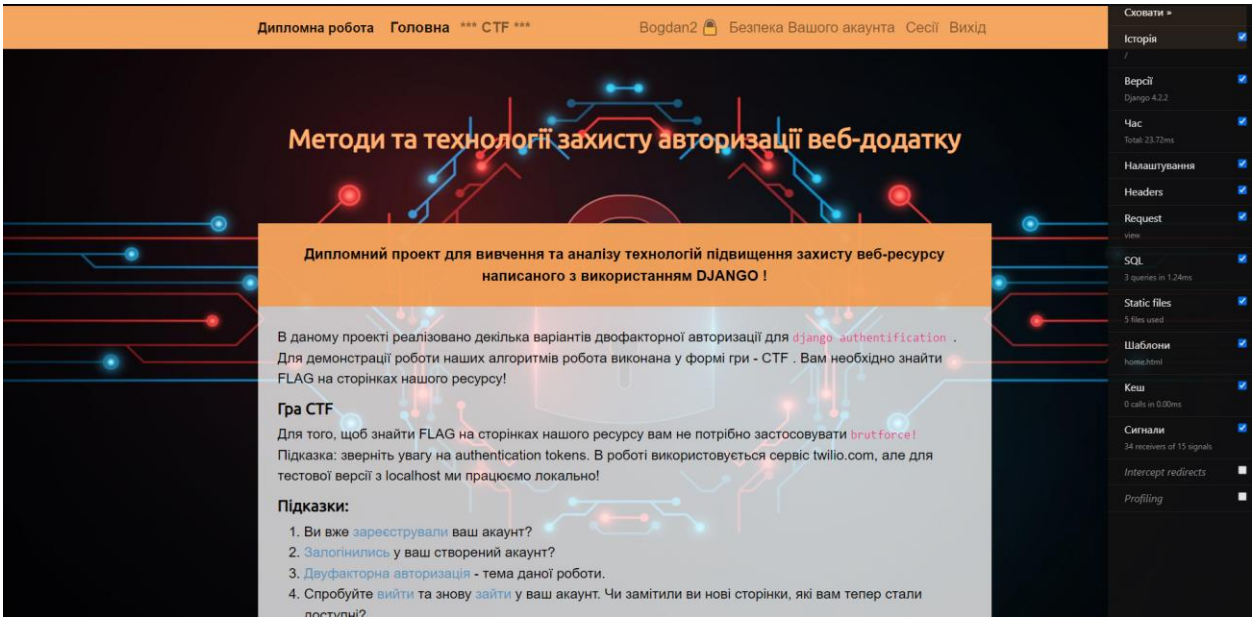

Рис. 3.6 Головна сторінка CTF - завдання

Для створення проекта спершу встановлюємо усі необхідні бібліотеки python, зокрема саму Django, pillow, requests. Всі залежності вказані в файлі requirements.txt.

Лістинг 3.1 Імпорти

```
from django.conf import settings
from django.contrib import admin
from django.contrib.auth.views import LogoutView
from django.urls import include, path

from two_factor.gateways.twilio.urls import urlpatterns as tf_twilio_urls
from two_factor.urls import urlpatterns as tf_urls

from .views import (
    ExampleSecretView, HomeView, RegistrationCompleteView, RegistrationView,
)
```

Наступне створюємо додаток. Всі операції здійснюємо в терміналі PyCharm:

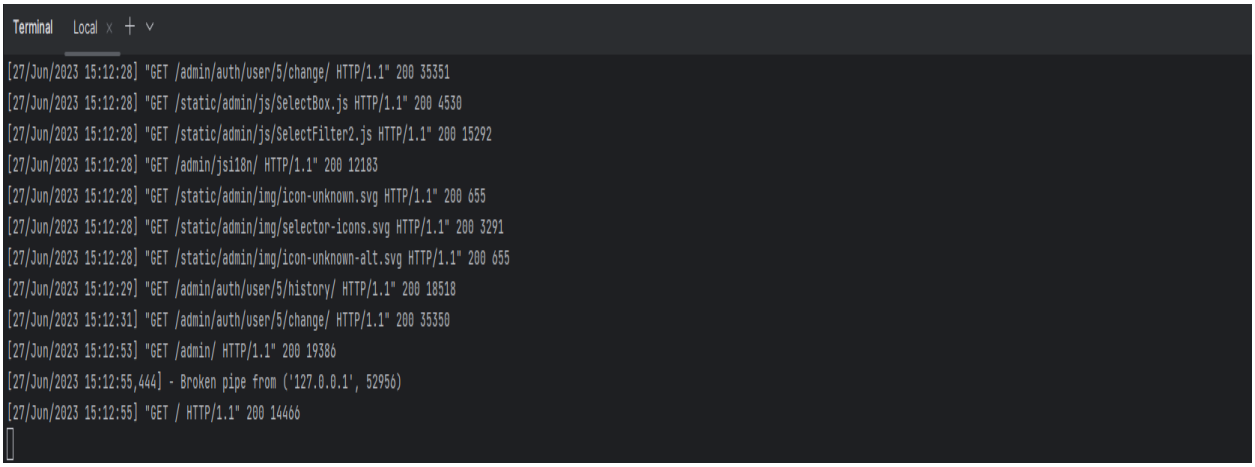

Рис. 3.7 Операції в терміналі

Виконуємо команду python3 manage.py startapp diplom

Ця команда створить новий каталог із файлами, зокрема будуть створені наприклад контролери (views), які повинні знаходиться во **views.py** , моделі в **models.py** , тести в **tests.py** , налаштування адміністративної частини в **admin.py** , реєстрація програми в **apps** .

Каталог *migrations* використовується, щоб зберігати "міграції" - файли, які дозволяють автоматично оновлювати базу даних у міру зміни моделей.

__init__.py - порожній файл для того, щоб Django і Python розпізнавали папку як модуль Python і це дозволяє нам використовувати його об'єкти всередині інших частин проекту.

Після створення додатка, нам потрібно зареєструвати його в проекті шляхом додаванням імені додатка до списку INSTALLED_APPS у налаштуваннях проекту (**settings.py**).

У нашому проекті INSTALLED_APPS будуть виглядати наступним чином:

Лістинг 3.2 INSTALLED_APPS

```
INSTALLED_APPS = [
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'user_sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.admin',
    'django_otp',
    'django_otp.plugins.otp_static',
    'django_otp.plugins.otp_totp',
    'django_otp.plugins.otp_email',
    'two_factor',
    'two_factor.plugins.phonenumber',
    'two_factor.plugins.email',
    'example',

    'debug_toolbar',
    'bootstrapform'
]
```

Увесь проект має наступну структуру:

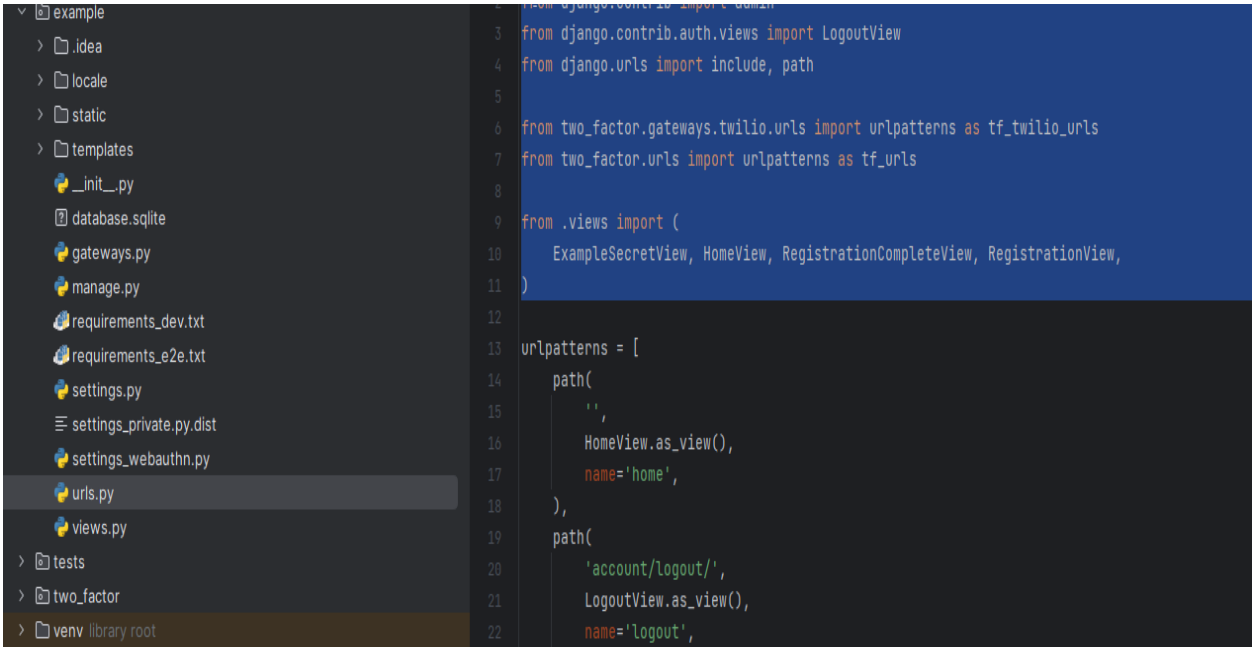

Рис.3.8 Структура проекта

3.3.2 SQLite

Ми будемо використовувати базу даних SQLite для цього проекту. Налаштування підключення до БД вказуємо в файлі **settings.py**

Лістинг 3.3 Налаштування доступу до БД

```
PROJECT_PATH = os.path.abspath(os.path.dirname(__file__))


DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(PROJECT_PATH, 'database.sqlite'),
    }
}
```

Django використовує ORM щоб співвідносити визначення моделей в Django додатку зі структурами даних , які використовуються базою даних . Коли ми

змінюємо наші моделі , Django відстежує зміни і може створити файли міграцій (у папці **/ locallibrary /catalog/migrations/**) .

При створенні сайту , Django автоматично додав кілька моделей , щоб ми могли їх використовувати в адмін-панелі. Щоб створити потрібні таблиці в базі даних які відповідають певним необхідно виконати наступні команди.

python manage.py makemigrations

python manage.py migrate

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, otp_email, otp_static, otp_totp, phonenumber, two_factor, user_sessions
Running migrations:
  No migrations to apply.
```

Рис.3.9 Migrate

3.3.3 Головна сторінка

Ми працюємо в тестовому режимі – розміщуємо проект на localhost. Головна сторінка - http://127.0.0.1:8000/ додатку має наступний вигляд.

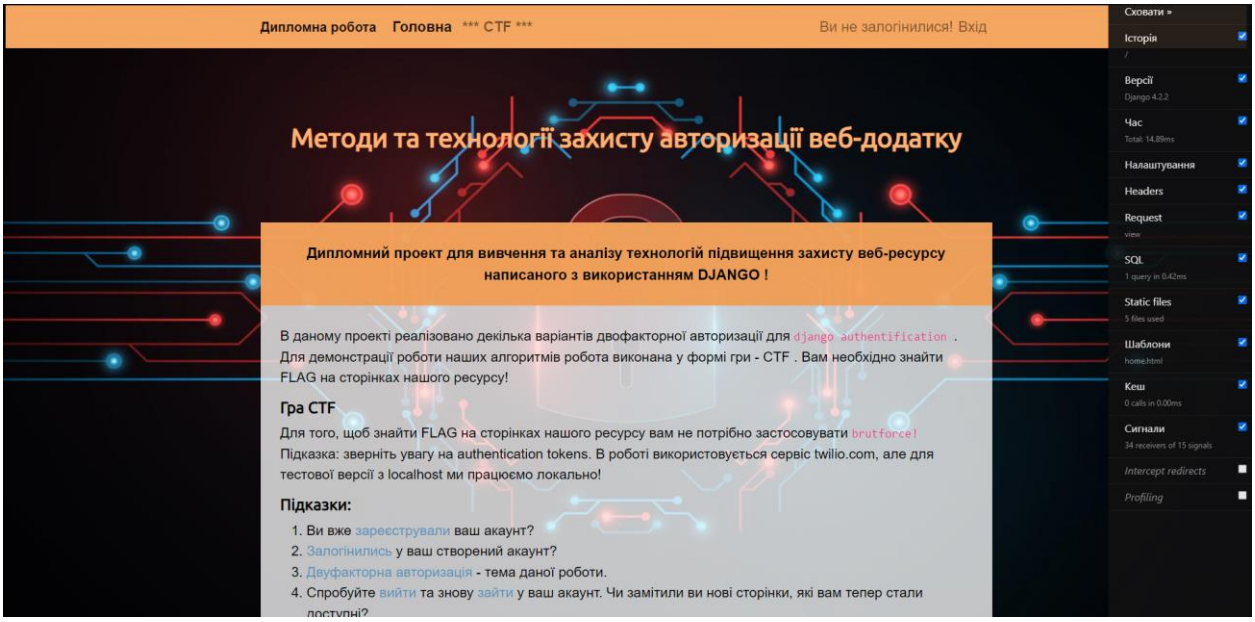

Рис. 3.10 Головна сторінка

В даному проекті реалізовано декілька варіантів двофакторної авторизації для django authentification . Для демонстрації роботи наших алгоритмів робота виконана у формі гри - CTF . Користувачам необхідно знайти FLAG на сторінках нашого ресурсу. Це можна зробити тільки пройшовши двофакторну авторизацію. Відразу на головній сторінці знаходяться підказки, як шукати FLAG – секретну інформацію.

Для задання зовнішнього вигляду використовуємо шаблон, який розширює _base.html:

Лістинг 3.4 Головна сторінка

```
{% extends "_base.html" %}
{% load i18n %}
{% load static %}

{% block title %}{% trans "Дипломна робота Двуфакторна авторизація в django "
%}{% endblock %}
{% block nav_home %}active{% endblock %}

{% block content %}
  <div style="
color: #ffb375;

";>
      <h1>{% blocktrans trimmed %} <br> <br> <b> <center>Методи та технології захисту авторизації веб-додатку</center></b>   <br> <br>{% endblocktrans %}</h1>
</div>
      <center> <b>Дипломний проект для вивчення та аналізу технологій підвищення захисту веб-ресурсу написаного з використанням DJANGO
!</b></center>
```

```
</div>
    <p>{% blocktrans trimmed %}В даному проекті реалізовано декілька варіантів двофакторної авторизації для  <code>django authentification </code> .
    Для демонстрації роботи наших алгоритмів робота виконана у формі гри - CTF . Вам необхідно знайти FLAG на сторінках нашого ресурсу! {% endblocktrans %}</p>

    <h4><b>{% trans "Гра CTF " %}</b></h4>
    <p>{% blocktrans trimmed%} Для того, щоб знайти FLAG на сторінках нашого ресурсу вам не потрібно застосовувати
        <code>brutforce! </code> Підказка: зверніть увагу на authentication tokens.
    В роботі використовується сервіс twilio.com, але для тестової версії з localhost ми працюємо локально! {% endblocktrans %}</p>

    {% url 'registration' as reg_url %}
    {% url 'two_factor:login' as login_url %}
    {% url 'two_factor:setup' as setup_url %}
    {% url 'logout' as logout_url %}
    {% url 'secret' as secret_url %}


 <h4><b>{% trans "Підказки:" %}</b></h4>
 <ol>
  <li>{% blocktrans trimmed %}Ви вже <a href="{{ reg_url }}">зареєстрували</a> ваш
    акаунт? {% endblocktrans %}</li>
  <li>{% blocktrans trimmed %}<a href="{{ login_url }}">Залогінились</a> у ваш
```

```
створений акаунт?
    {% endblocktrans %}</li>
  <li>{% blocktrans trimmed %}<a href="{{ setup_url }}">Двуфакторна авторизація
</a> - тема даної роботи. {% endblocktrans %}</li>
  <li>{% blocktrans trimmed %}Спробуйте <a href="{{ logout_url }}">вийти</a> та
знову
    <a href="{{ login_url }}">зайти</a> у ваш акаунт. Чи замітили ви нові сторінки,
які вам тепер стали доступні?{% endblocktrans %}</li>
  <li>{% blocktrans trimmed %} Все, у вас достатньо підказок! Але якщо Ви все-
таки не знаєте, де ж відповідь, то ось вам остання підказка:
    <a href="{{ secret_url }}"> секретна сторінка </a>! :-){% endblocktrans %}</li>
 </ol>
{% endblock %}
```

Код головної сторінки включає наступні елементи: імпорти, опис класа HomeView та RegistrationView, декоратор ExampleSecretView та ще деякі функції. Фактично із основної сторінки ми маємо посилання та імпорти багатьох інших, які описані в інших модулях.

Лістинг 3.5 Клас HomeView, RegistrationView, декоратор ExampleSecretView

```
from django.conf import settings
from django.contrib.auth.forms import UserCreationForm
from django.shortcuts import redirect, resolve_url
from django.views.decorators.cache import never_cache
from django.views.generic import FormView, TemplateView
```

```
from two_factor.views import OTPRequiredMixin
from two_factor.views.utils import class_view_decorator


class HomeView(TemplateView):
    template_name = 'home.html'


class RegistrationView(FormView):
    template_name = 'registration.html'
    form_class = UserCreationForm

    def form_valid(self, form):
        form.save()
        return redirect('registration_complete')


class RegistrationCompleteView(TemplateView):
    template_name = 'registration_complete.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['login_url'] = resolve_url(settings.LOGIN_URL)
        return context


@class_view_decorator(never_cache)
```

```
class ExampleSecretView(OTPRequiredMixin, TemplateView):
    template_name = 'secret.html'
```

3.3.4 Форма реєстрації

Спершу необхідно реалізувати реєстрацію користувача. Переходимо по посиланню, яке доступне як в верхній панелі, так і в підказках до гри - http://127.0.0.1:8000/account/register/. Бачимо форму реєстрації:

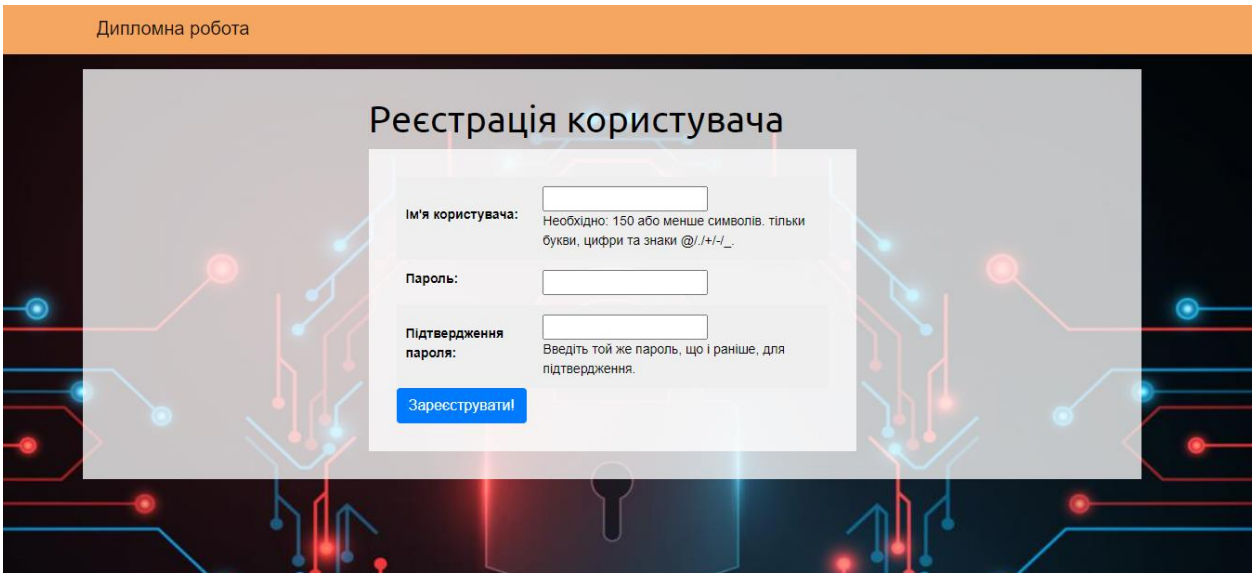

Рис. 3.11 Реєстрація користувача

Після натискання Зареєструвати переходимо на сторінку:

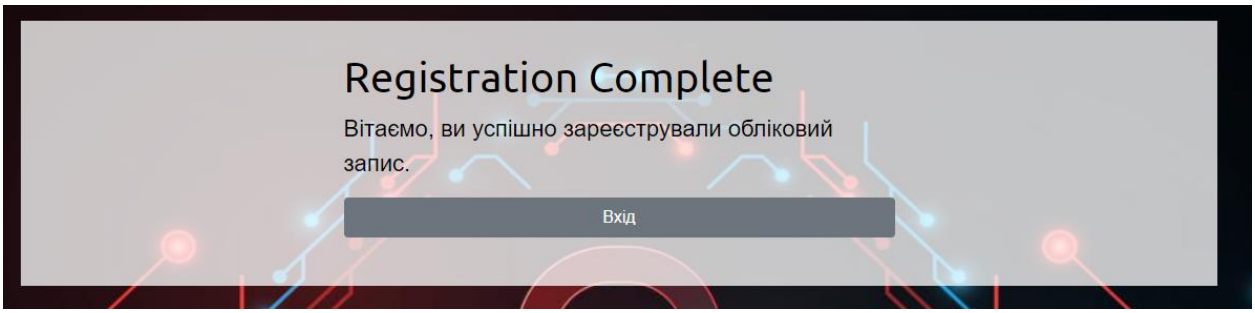

Рис. 3.12 Продовження виконання завдання

Нам пропонують відразу залогінитись. Авторизуємось:

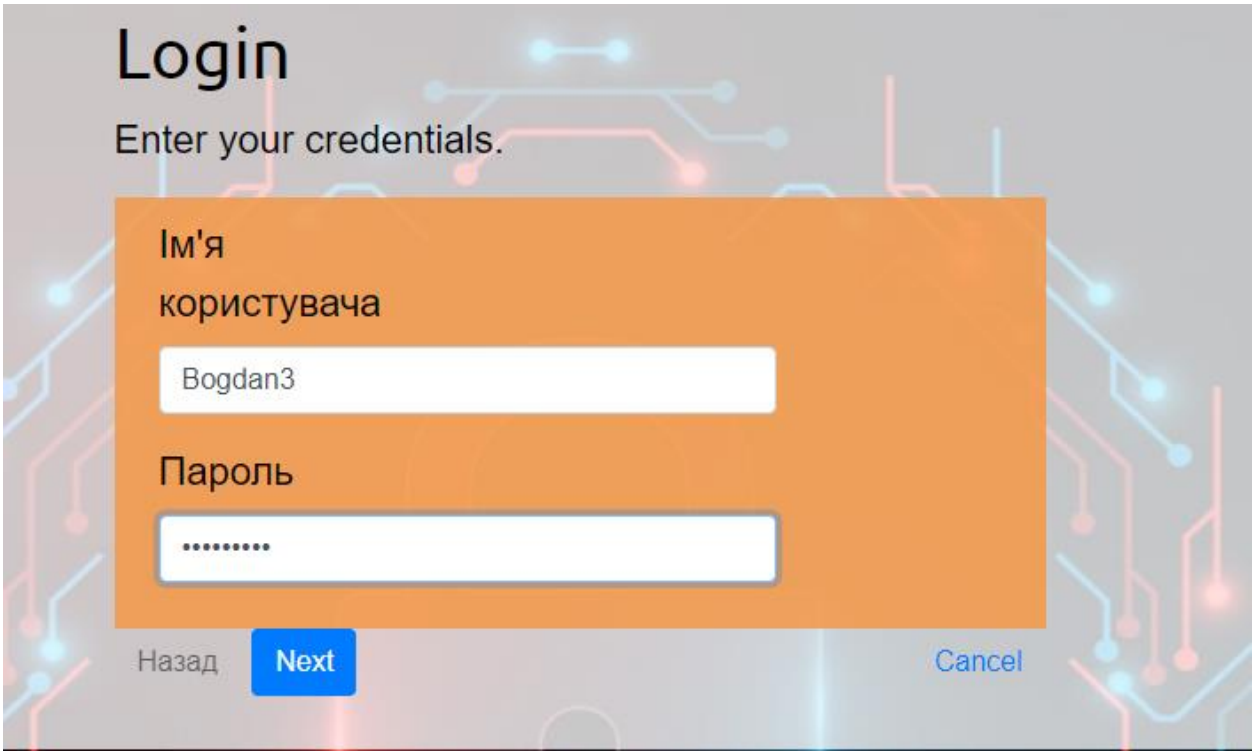

Рис.3.13 Вхід на сайт простого користувача

3.3.5 Двофакторна аутентифікація

Після реєстрації користувача, виникає повідомлення, що двофакторна аутентифікація відключена. Пропонується включити її. Зробимо це:

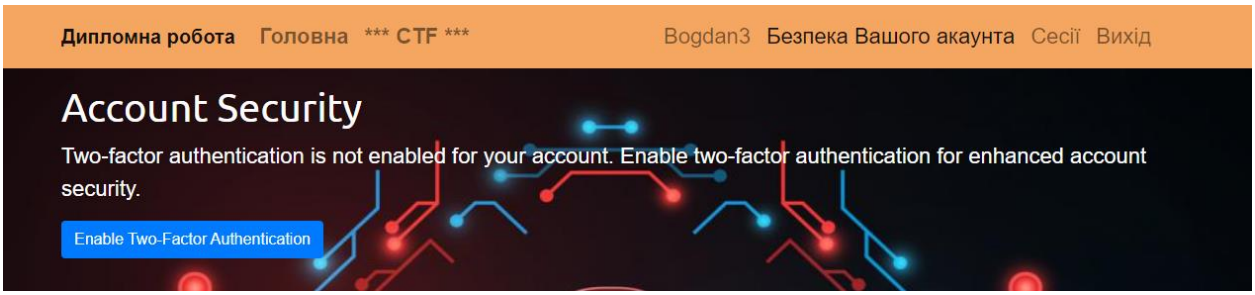

Рис. 3.14 Безпека акаунта

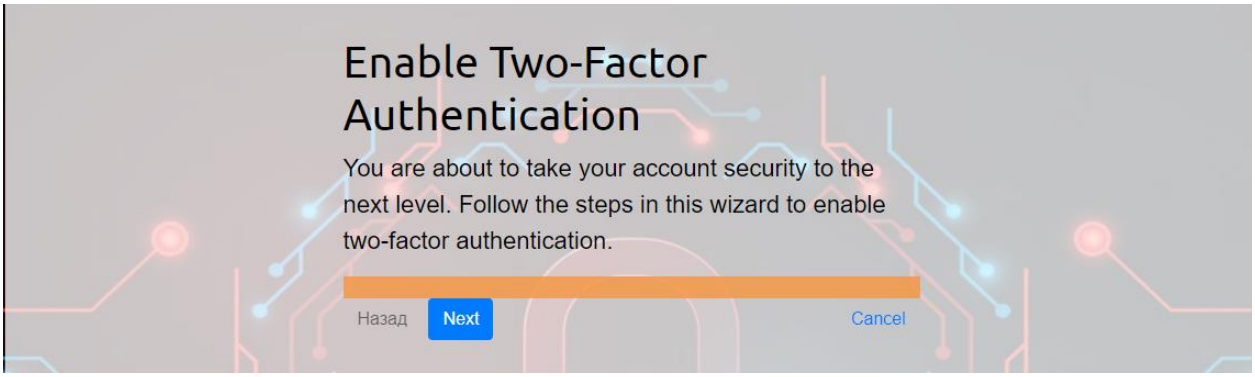

Рис. 3.15 Вмикаємо двофакторну авторизацію

В роботі реалізовано кілька методів аутентифікації, для кожного реалізовано відповідні функції. Зокрема, доступні token generator, phone call, text message, email. При цьому в роботі використовується сервіс twilio.com, через який можна налаштувати дзвінки-підтвердження, надсилання повідомлень. Управління виконується через адміністративну панель:

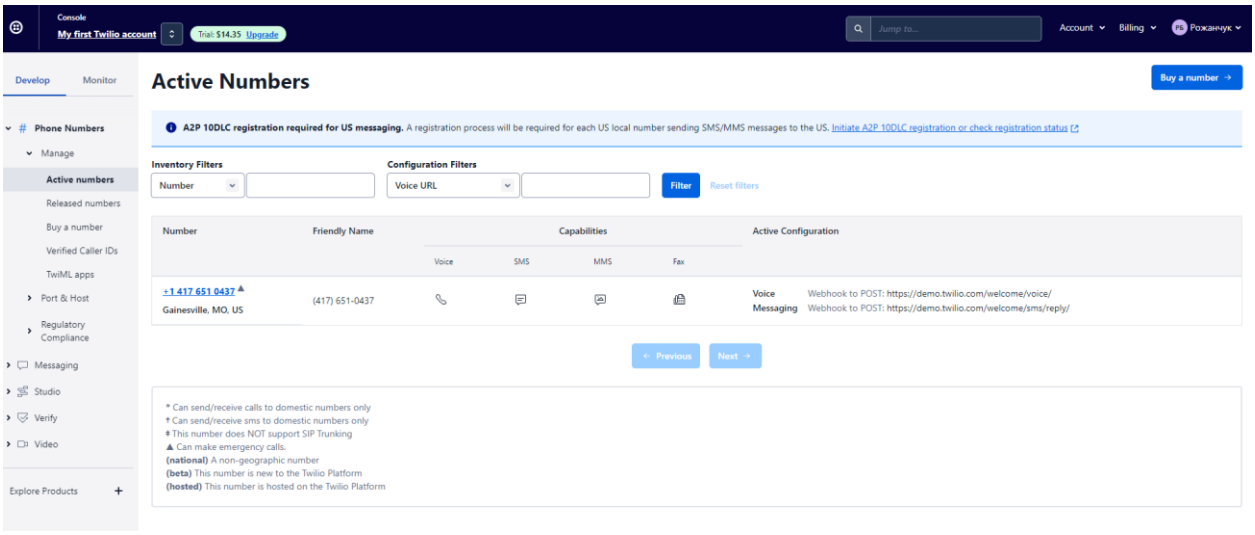

Рис.3.16 twilio.com

У коді Python нам потрібно вказати параметри API Key, які генеруються в акаунті twilio для того, щоб можна було автоматично генерувати запити.

Лістинг 3.6 TWILIO_ACCOUNT_SID та TWILIO_AUTH_TOKEN

TWO_FACTOR_SMS_GATEWAY = 'two_factor.gateways.twilio.gateway.Twilio'

TWO_FACTOR_CALL_GATEWAY = 'two_factor.gateways.twilio.gateway.Twilio'

TWILIO_ACCOUNT_SID = '**********'

TWILIO_AUTH_TOKEN = '**********'

TWILIO_CALLER_ID = '**********'

В проекті реалізовано наступні методи двофакторної аутентифікації:

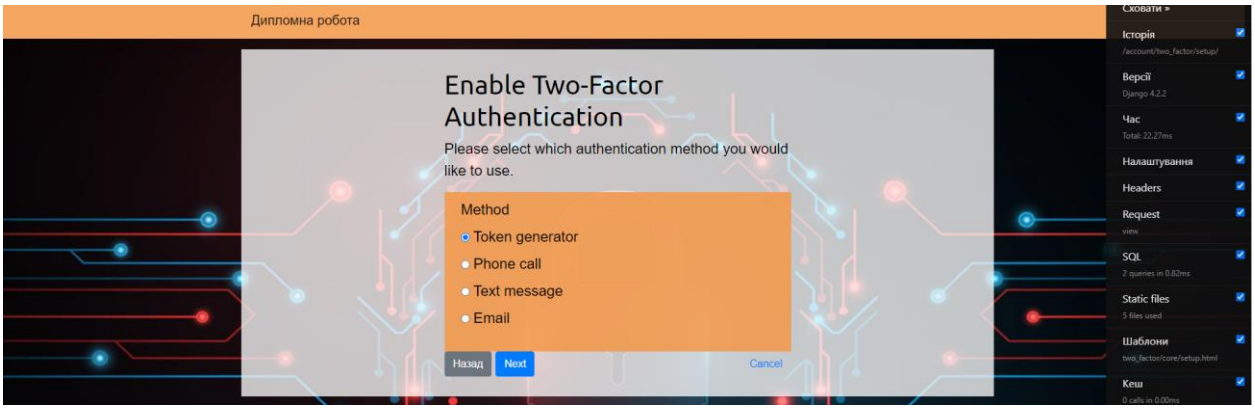

Рис. 3.17 Методи автентифікації

Ящо обираємо token generator отримуємо вікно:

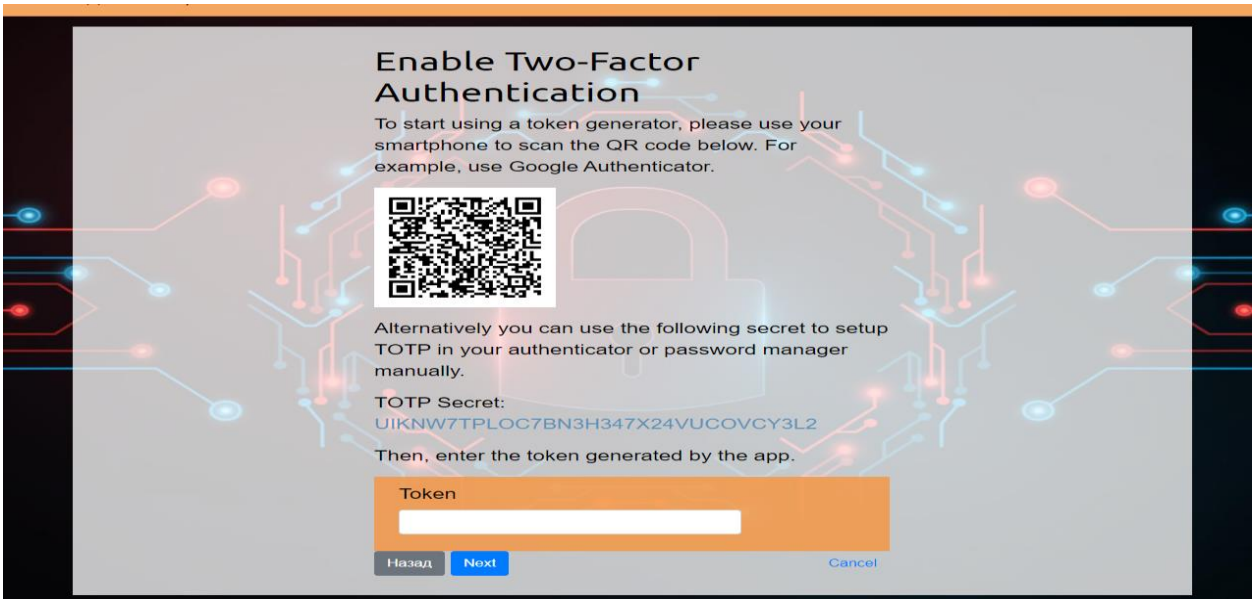

Рис. 3.18 token generator

В коді працювати з токенами можна наступним чином. Спочатку ми маємо на основі ACCOUNT_SECURITY_API_KEY ініціалізувати зєднання і отримати API через AuthyApiClient.Зареєструвавши користувача, окремо описуємо функцію для sms, для дзвінків та для класичних токенів.

Лістинг 3.7 Робота з AuthyApiClient(

```
authy_api = AuthyApiClient(settings.ACCOUNT_SECURITY_API_KEY)



def register(request):
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            authy_user = authy_api.users.create(
                form.cleaned_data['email'],
                form.cleaned_data['phone_number'],
                form.cleaned_data['country_code'],
            )
            if authy_user.ok():
                twofa_user = TwoFAUser.objects.create_user(
                    form.cleaned_data['username'],
                    form.cleaned_data['email'],
                    authy_user.id,
                    form.cleaned_data['password']
                )
                login(request, twofa_user)
                return redirect('2fa')
            else:
```

```
            for key, value in authy_user.errors().items():
                form.add_error(
                    None,
                    '{key}: {value}'.format(key=key, value=value)
                )
    else:
        form = RegistrationForm()
    return render(request, 'register.html', {'form': form})
@login_required
def token_sms(request):
    sms = authy_api.users.request_sms(request.user.authy_id, {'force': True})
    if sms.ok():
        return HttpResponse('SMS request successful', status=200)
    else:
        return HttpResponse('SMS request failed', status=503)


@login_required
def token_voice(request):
    call = authy_api.users.request_call(request.user.authy_id, {'force': True})
    if call.ok():
        return HttpResponse('Call request successfull', status=200)
    else:
        return HttpResponse('Call request failed', status=503)


@login_required
def token_onetouch(request):
```

```
    details = {
        'Authy ID': request.user.authy_id,
        'Username': request.user.username,
        'Reason': 'Demo by Account Security'
    }

    hidden_details = {
        'test': 'This is a'
    }

    response = authy_api.one_touch.send_request(
        int(request.user.authy_id),
        message='Login requested for Account Security account.',
        seconds_to_expire=120,
        details=details,
        hidden_details=hidden_details
    )
    if response.ok():
        request.session['onetouch_uuid'] = response.get_uuid()
        return HttpResponse('OneTouch request successfull', status=200)
    else:
        return HttpResponse('OneTouch request failed', status=503)
```

Якщо хочемо виконувати двофакторну автентифікацію через дзвінки, то маємо вказати номер:

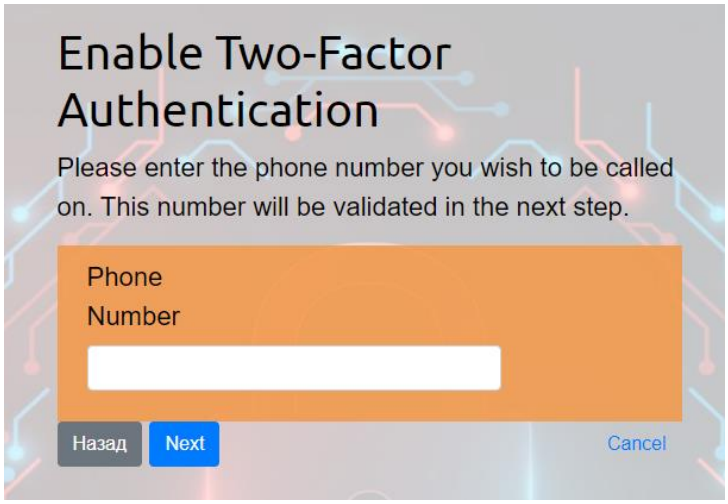

Рис. 3.19 Автентифікація по номеру телефона

Обмеження по часу, протягом якого дійсний токен, ми встановлюємо в коді самі, в даному випадку виставлено через параметр TWO_FACTOR_REMEMBER_COOKIE_AGE.

```
TWO_FACTOR_REMEMBER_COOKIE_AGE = 120
```

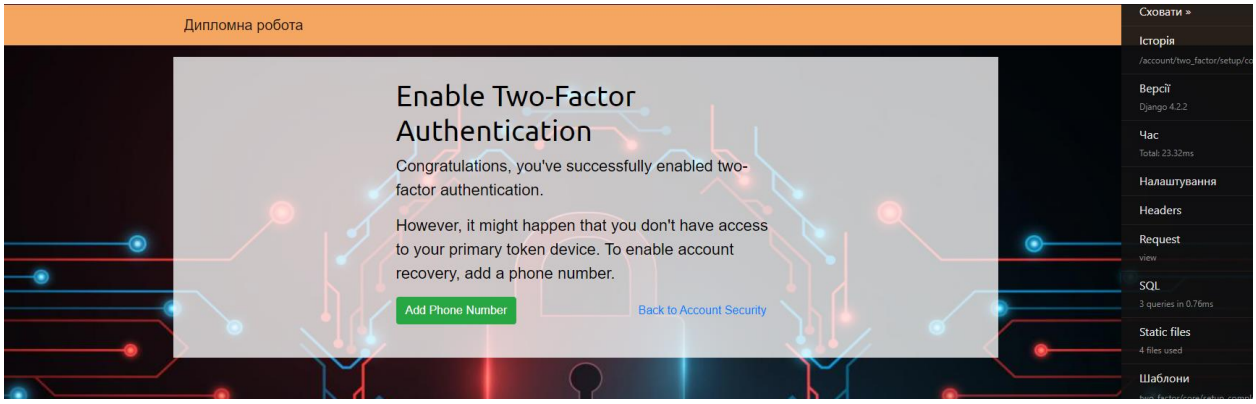

Рис. 3.20 Правильно введений токен

Для роботи з токенами створено клас:

Лістинг 3.8 Робота з токенами

```
class AuthenticationTokenForm(OTPAuthenticationFormMixin, forms.Form):
    otp_token = forms.RegexField(label=_("Token"),
```

```
                        regex=r'^[0-9]*$',
                        min_length=totp_digits(),
                        max_length=totp_digits())
    otp_token.widget.attrs.update({
        'autofocus': 'autofocus',
        'pattern': '[0-9]*',
        'autocomplete': 'one-time-code',
    })


    use_required_attribute = False
    idempotent = False

    def __init__(self, user, initial_device, **kwargs):
            super().__init__(**kwargs)
        self.user = user
        self.initial_device = initial_device

        # Add a field to remember this browser.
        if getattr(settings, 'TWO_FACTOR_REMEMBER_COOKIE_AGE', None):
            if settings.TWO_FACTOR_REMEMBER_COOKIE_AGE < 3600:
                minutes = int(settings.TWO_FACTOR_REMEMBER_COOKIE_AGE / 60)
                label = _("Don't ask again on this device for %(minutes)i minutes") %
{'minutes': minutes}
            elif settings.TWO_FACTOR_REMEMBER_COOKIE_AGE < 3600 * 24:
                hours = int(settings.TWO_FACTOR_REMEMBER_COOKIE_AGE / 3600)
                label = _("Don't ask again on this device for %(hours)i hours") % {'hours':
hours}
```

```
            else:
                days = int(settings.TWO_FACTOR_REMEMBER_COOKIE_AGE / 3600 /
24)
                label = _("Don't ask again on this device for %(days)i days") % {'days': days}

            self.fields['remember'] = forms.BooleanField(
                required=False,
                initial=True,
                label=label
            )

    def clean(self):
        self.clean_otp(self.user)
        return self.cleaned_data


class BackupTokenForm(AuthenticationTokenForm):
    otp_token = forms.CharField(label=_("Token"))
```

3.3.6 Налаштування безпеки

В налаштуваннях акаунта можна встановити пареметри безпеки, а асаме привязати номер телефона, використати bacup tokens.

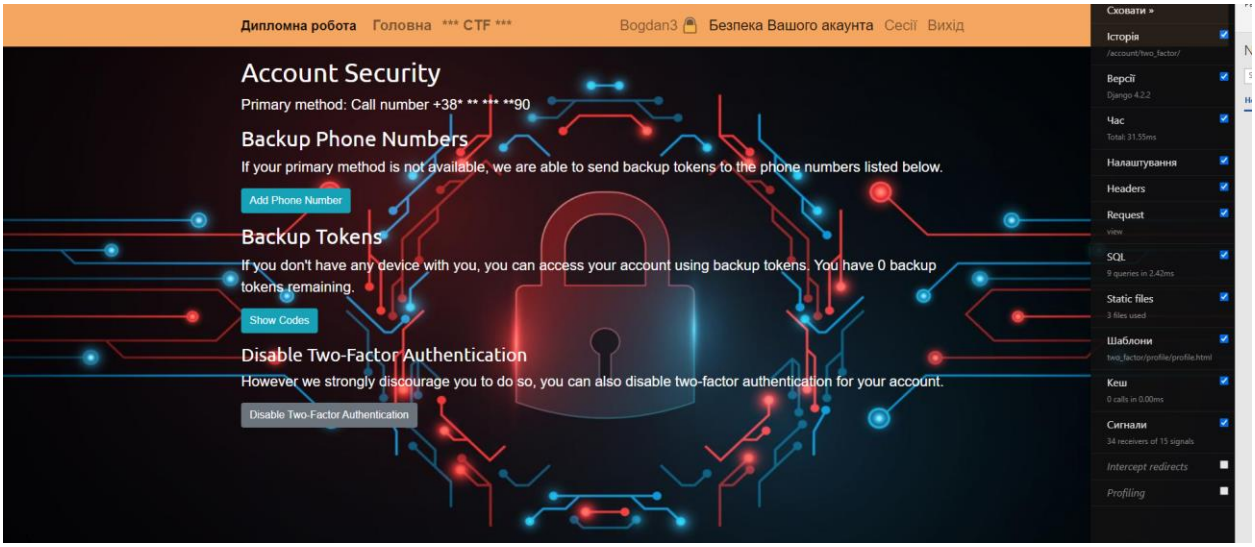

Рис. 3.21 Безпека акаунта

Також реалізовано перегляд активних сесій для підвищення рівня безпеки:

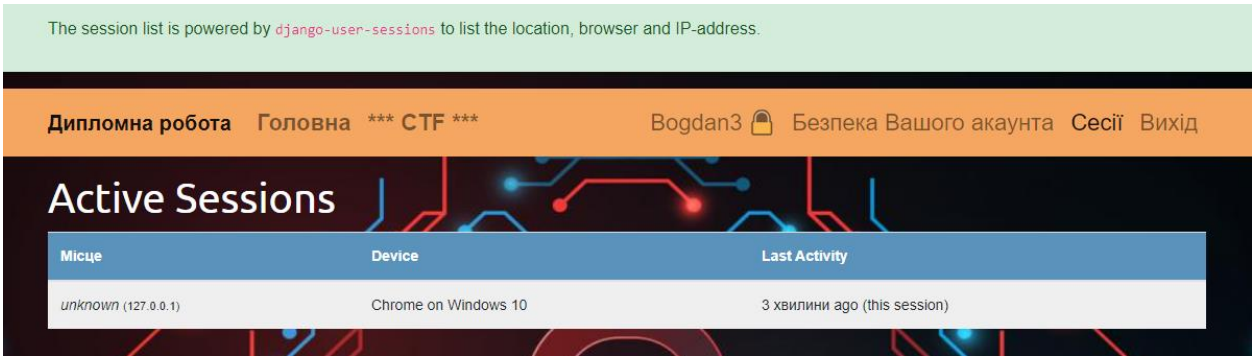

Рис. 3.22 Сесії

В боковій панелі керування можна знайти інформацію по всіх основних параметрах, а саме – історія запитів до сайту, версія ПЗ, зокрема Django, python, дані про основні встановлені налаштування, інформація про статичні файли, шаблони, кеш , запити до БД.

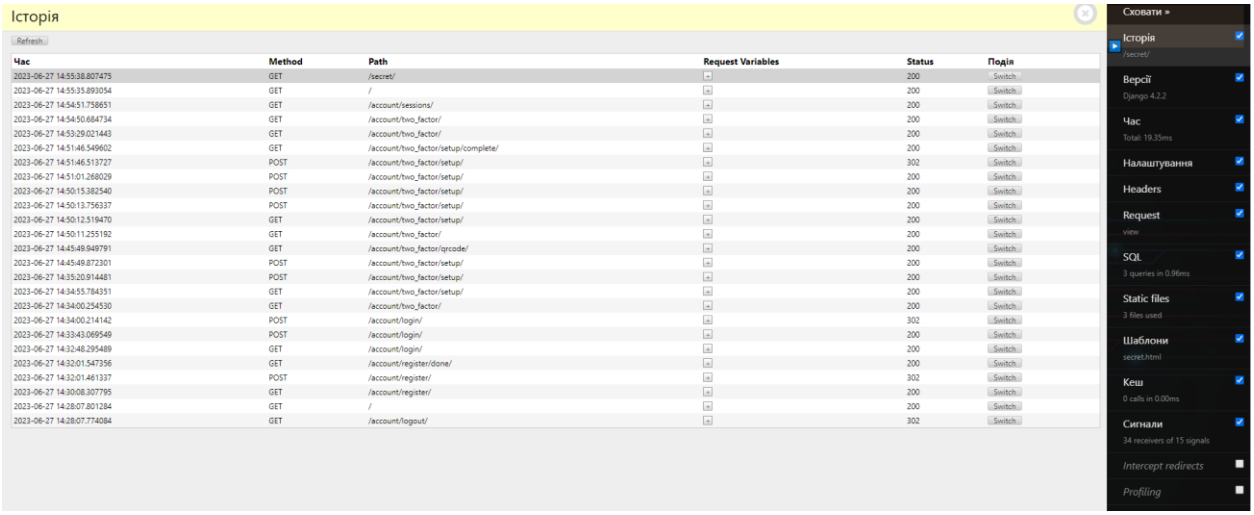

Рис. 3.23 Історія запитів до ресурсу

Settings from example.settings

Setting	Значення
ABSOLUTE_URL_OVERRIDES	{}
ADMINS	[]
ALLOWED_HOSTS	[]
APPEND_SLASH	True
AUTHENTICATION_BACKENDS	('django.contrib.auth.backends.ModelBackend',)
AUTH_PASSWORD_VALIDATORS	'********************'
AUTH_USER_MODEL	'auth.User'
CACHES	{'default': {'BACKEND': 'django.core.cache.backends.locmem.LocMemCache'}}
CACHE_MIDDLEWARE_ALIAS	'default'
CACHE_MIDDLEWARE_KEY_PREFIX	'********************'
CACHE_MIDDLEWARE_SECONDS	600
CMS_LANGUAGES	{1: [{'code': 'ru', 'hide_untranslated': False, 'name': 'Русский', 'public': True, 'redirect_on_fallback': True}, {'code': 'uk', 'hide_untranslated': False, 'name': 'Українська', 'public': True, 'redirect_on_fallback': True}, {'code': 'en', 'hide_untranslated': False, 'name': 'English', 'public': True, 'redirect_on_fallback': True}], 'default': {'hide_untranslated': False, 'public': True, 'redirect_on_fallback': True}}
CSRF_COOKIE_AGE	31449600
CSRF_COOKIE_DOMAIN	None
CSRF_COOKIE_HTTPONLY	False
CSRF_COOKIE_MASKED	False
CSRF_COOKIE_NAME	'csrftoken'
CSRF_COOKIE_PATH	'/'
CSRF_COOKIE_SAMESITE	'Lax'
CSRF_COOKIE_SECURE	False
CSRF_FAILURE_VIEW	'django.views.csrf.csrf_failure'
CSRF_HEADER_NAME	'HTTP_X_CSRFTOKEN'
CSRF_TRUSTED_ORIGINS	[]
CSRF_USE_SESSIONS	False
DATABASES	{'default': {'ATOMIC_REQUESTS': False

Рис. 3.24 Налаштування

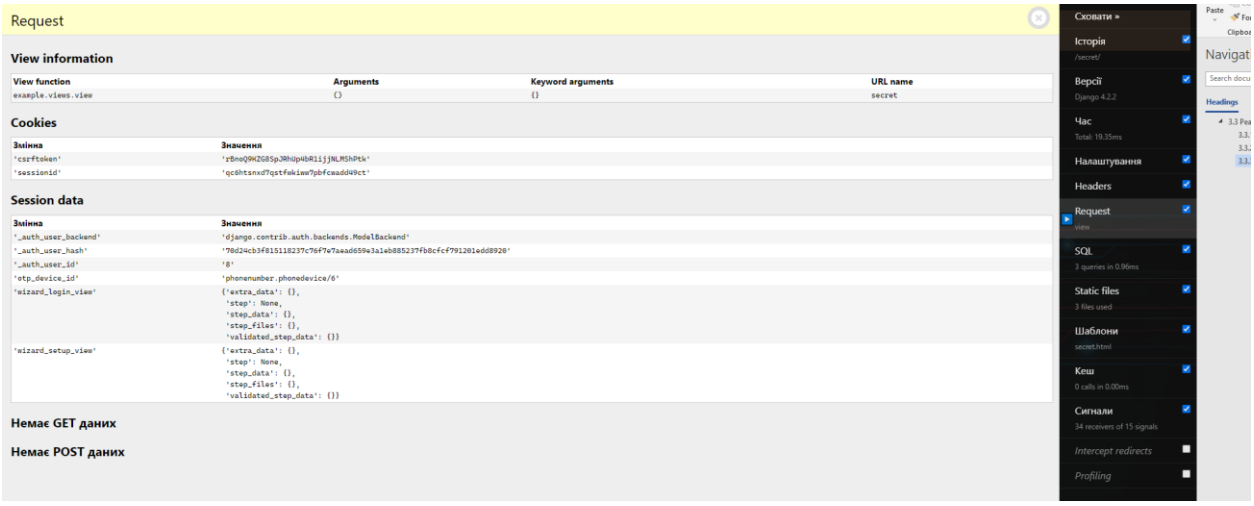

Рис. 3.25 Запити

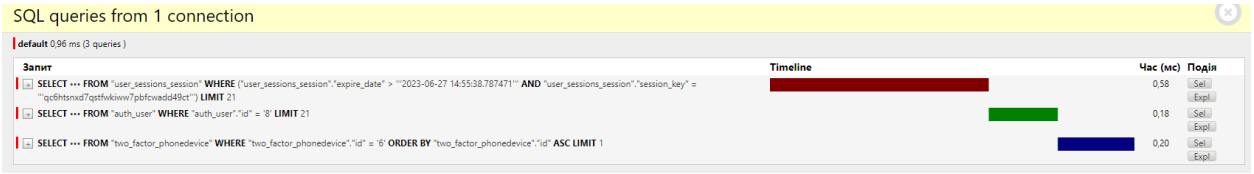

Рис. 3.26 SQL queries from 1 connection

3.3.7 FLAG

В результаті, після активації двофакторної аутентифікації нам стають доступні секретні дані – сторінка http://127.0.0.1:8000/secret/

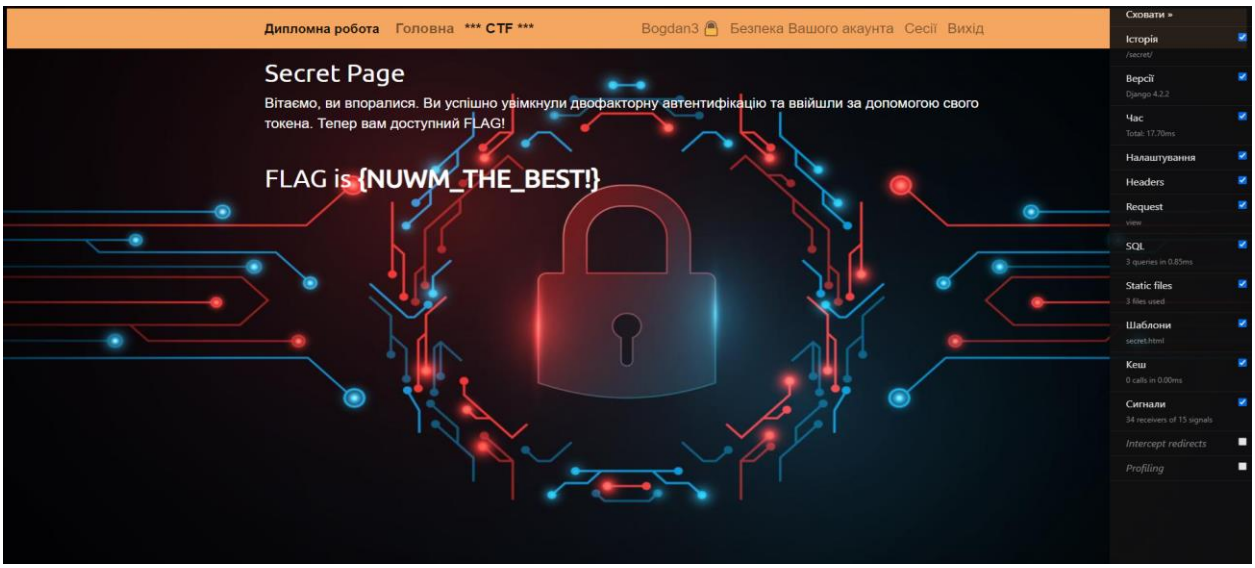

Рис. 3.27 FLAG

При цьому слід зауважити, що якщо спробувати отримати доступ без встановлення двофакторної аутентифікації, навіть із простою реєстрацією, ми отримуємо:

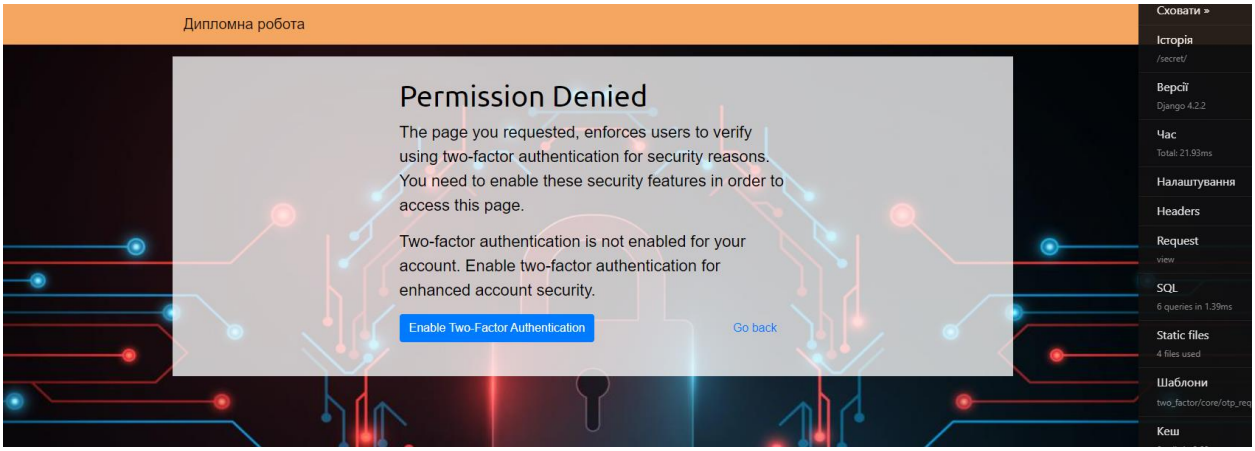

Рис. 3.28 http://127.0.0.1:8000/secret/

3.3.8 Адміністративна панель

Також створений адмін для сайта. Залогінитись під адміном можна з наступними даними marshall/12345(і токеном звичайно, адже для адміна встановлена двофакторна авторизація):

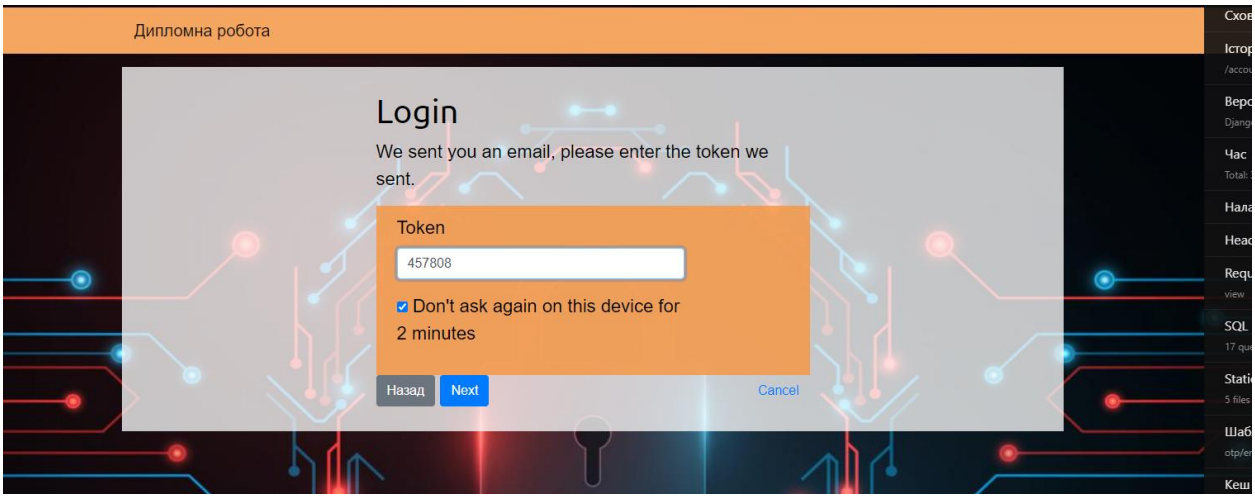

Рис. 3.29 Токен при авторизації

Адміністративна панель знаходиться за адресом http://127.0.0.1:8000/admin/ та виглядає наступним чином:

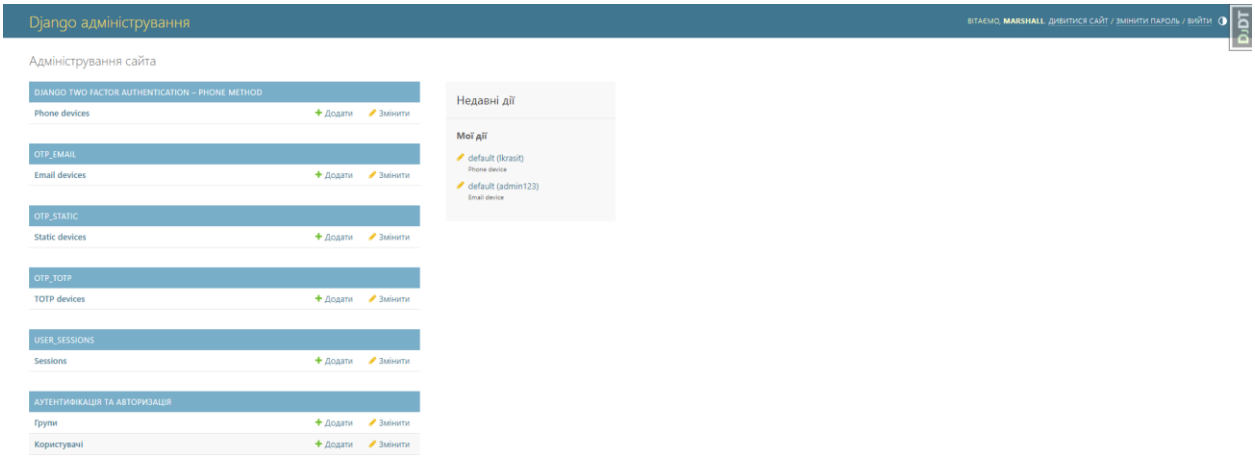

Рис. 3.30 Адміністративна панель

Додатково через панель можемо налаштовувати token для різних обєктів:

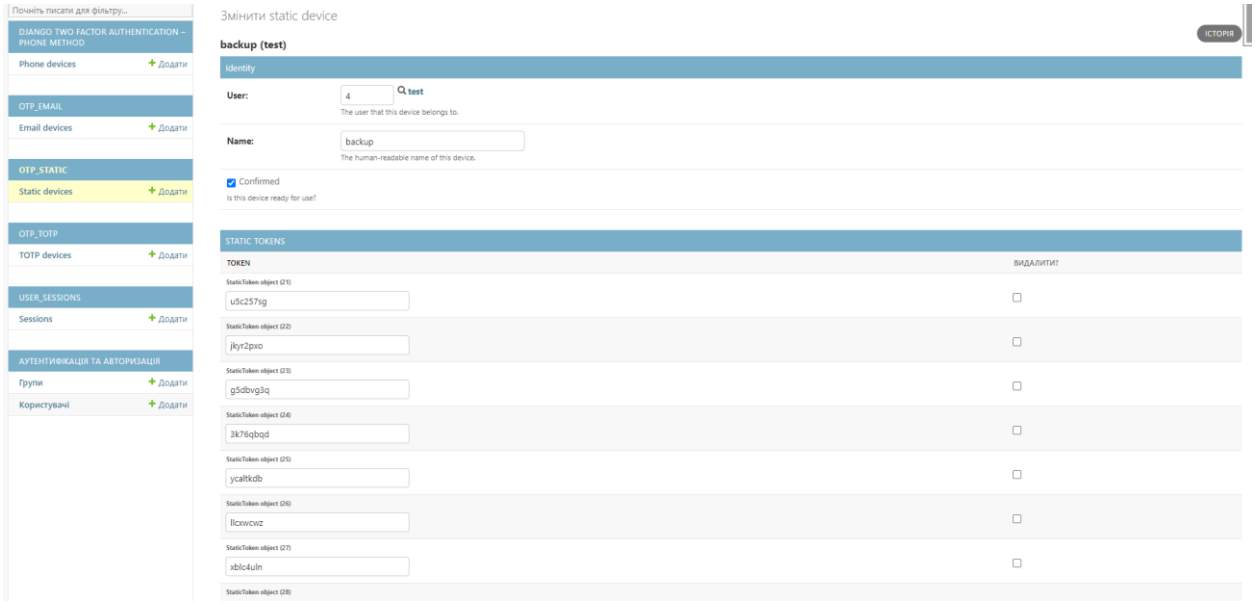

Рис. 3.31 Токени

Для роботи з правами адміністратора описано відповідні функції перевірки прав доступу. Якщо прав недостатньо, відбувається перенаправлення на форму входу.

Лістинг 3.9 Адміністрування ресурса

```
def patch_admin():
    @monkeypatch_method(AdminSite)
    def login(self, request, extra_context=None):

        redirect_to = request.POST.get(REDIRECT_FIELD_NAME,
request.GET.get(REDIRECT_FIELD_NAME))

        if not redirect_to or not url_has_allowed_host_and_scheme(url=redirect_to,
allowed_hosts=[request.get_host()]):
            redirect_to = resolve_url(settings.LOGIN_REDIRECT_URL)

        return redirect_to_login(redirect_to)
```

Лістинг 3.10 Права доступу

```
def has_permission(self, request):

    if not super().has_permission(request):
        return False
    return request.user.is_verified()

def login(self, request, extra_context=None):

    redirect_to = request.POST.get(REDIRECT_FIELD_NAME,
request.GET.get(REDIRECT_FIELD_NAME))

    if not redirect_to or not url_has_allowed_host_and_scheme(url=redirect_to,
allowed_hosts=[request.get_host()]):
        redirect_to = resolve_url(settings.LOGIN_REDIRECT_URL)
```

return redirect_to_login(redirect_to)

Адміністратор може також переглядати сесії, користувачів, модифікувати дані:

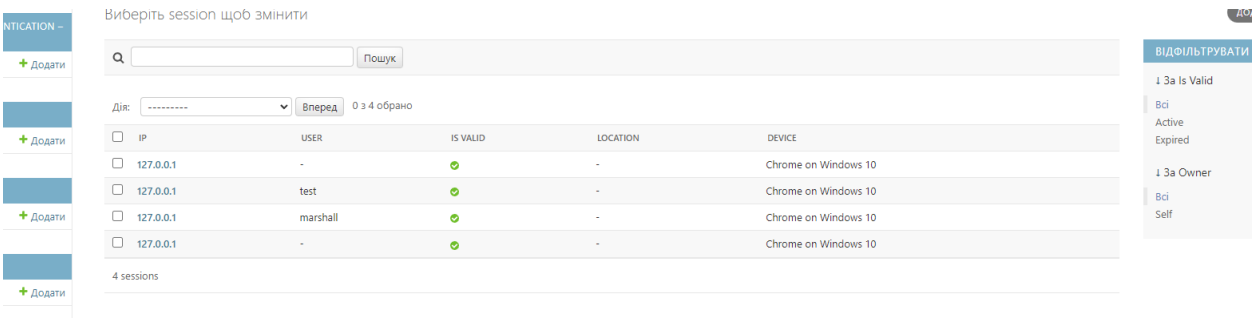

Рис.3.32 Активні сесії

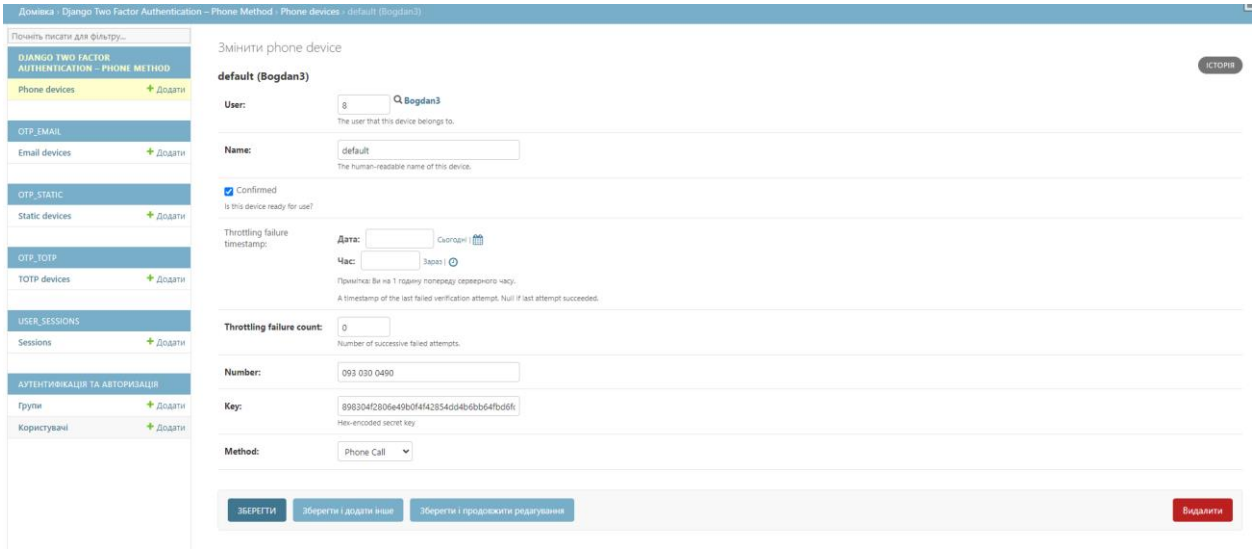

Рис. 3.33 Налаштування користувача

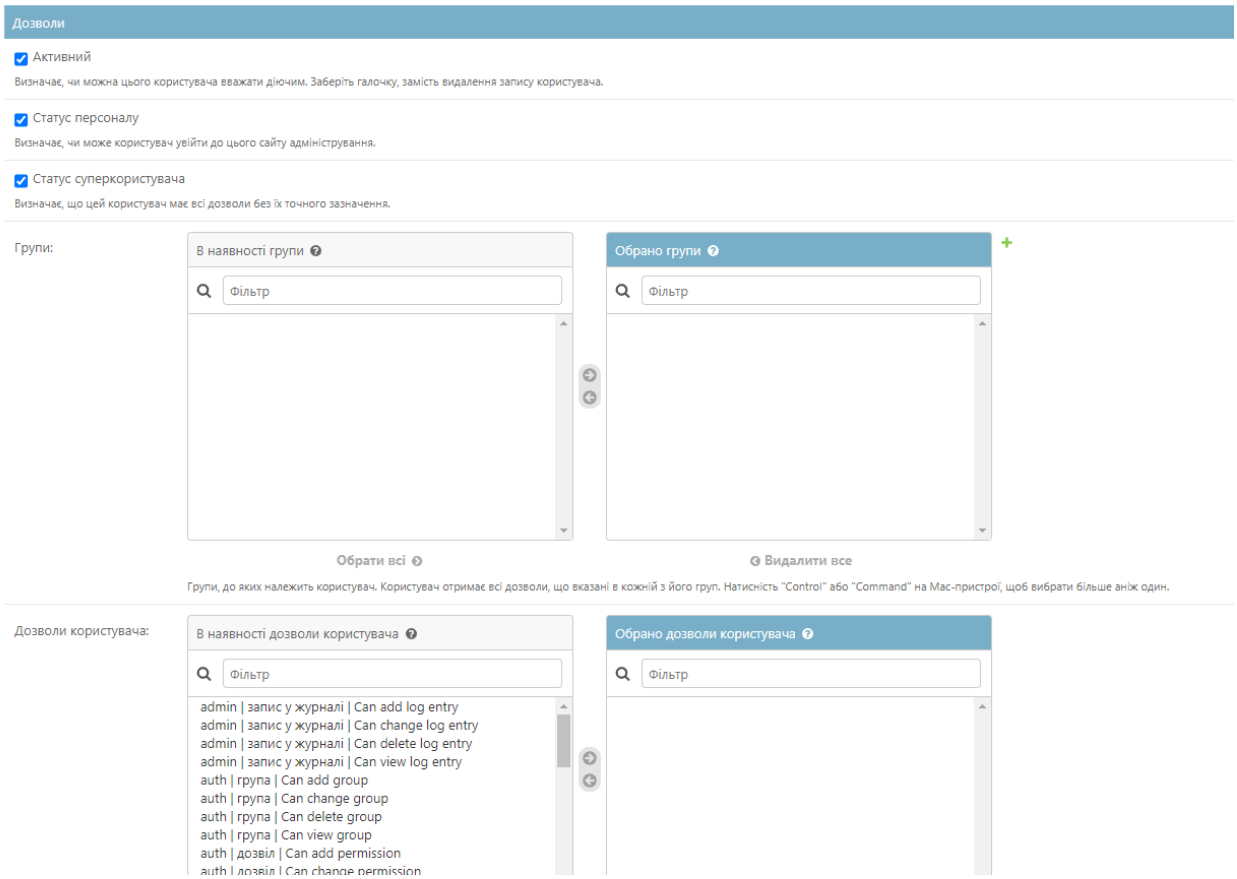

Рис. 3.34 Дозволи

3.4 Тестування системи

Модель веб-застосунка і реалізація двофакторної автентифікації містять багато компонентів, що ускладнює перевірку їх працездатності та стійкості системи. Це особливо важливо не лише з точки зору правильності виконання бізнес-логіки, але й з позиції інформаційної безпеки. У системі взаємодії з веб-застосунком використовується лише один канал - HTTP протокол та формат даних JSON. Для первинного тестування розробники, тестувальники та інженери інформаційної безпеки використовують утиліту POSTMAN. Протягом демонстрації роботи були надані знімки екрану цієї утиліти.

Postman - це комерційна настільна програма, яка доступна для Windows, Mac OS та Linux. Програма має безкоштовну версію, а також платні плани, які надають

можливості співпраці та документації. Ці функції більш актуальні для розробників, ніж для пен-тестерів. Postman дозволяє керувати колекціями HTTP-запитів для тестування різних викликів API, а також середовищами, що містять змінні. Він не замінює проксі-сервера (наприклад, Burp, ZAP, Mitmproxy), але дійсно виступає як відсутній рівень браузера та клієнтського додатка. Іншими альтернативами є інструменти з відкритим вихідним кодом, такі як Insomnia та Advanced REST Client, комерційний варіант SoapUI або власноруч розроблені інструменти, засновані на інтерфейсі Swagger / Swagger UI або згортанні.

На практиці ручне тестування не завжди виправдовує себе через швидке зростання функціоналу або зміни в роботі. Його виконання забирає багато часу, тому логічним кроком є використання автоматичних тестів. Наприклад, можна використати функціональні тести, які описані у Додатку В, для перевірки функціонування першого етапу.

Висновки за третім розділом

Отже, в останньому розділі було проведено вибір стеку технологій, надано обґрунтування для кожної з них з точки зору інформаційної безпеки. Було розглянуто основні компоненти двофакторної аутентифікації, такі як PHP, SQL, Linux та Redis. Також були виділені основні вимоги до системи в цілому, зокрема надійність, стресостійкість, масштабованість та забезпечення захисту від несанкціонованого доступу.

Враховуючи вищезгадані технології та фактори, була розроблена система двофакторної аутентифікації, яка складається з двох етапів. Перший етап полягає у перевірці пароля користувача за логіном та відправленні спеціального одноразового пароля (OTP) на електронну пошту. Другий етап включає верифікацію з використанням тимчасового пароля, отриманого з електронної пошти користувача.

Цей підхід забезпечує більш високий рівень безпеки, оскільки вимагає двох незалежних факторів для підтвердження ідентичності користувача. Сполучення пароля та OTP забезпечує відносно надійний механізм аутентифікації, оскільки одноразовий пароль генерується на підставі унікальних факторів та відправляється користувачу для підтвердження його ідентичності.

Таким чином, розроблена система автентифікації з двома факторами відповідає високим стандартам безпеки та забезпечує надійний контроль доступу до системи.

ВИСНОВКИ

У ході дослідження, проведеного в рамках дипломної роботи, були досягнуті основні цілі дослідження. Здійснено аналіз сучасних веб-застосунків та вивчено особливості їх створення, експлуатації та забезпечення інформаційної безпеки. Особлива увага була приділена методам автентифікації, які використовуються у веб-застосунках.

В рамках дослідження було визначено поняття веб-застосунка та його класифікація. Веб-застосунок розглядається як програмний комплекс, що працює через мережу Інтернет та має кілька параметрів для класифікації, зокрема, за архітектурою, навантаженістю та сферою застосування.

У процесі дослідження було проаналізовано різні методи автентифікації та визначено, що найбільш оптимальним є двохфакторна автентифікація, яка використовує тимчасовий пароль та спеціальний токен автентифікації після верифікації. Цей підхід забезпечує більш високий рівень безпеки в порівнянні з іншими методами.

Для реалізації системи автентифікації була розроблена модель, яка базується на популярних технологіях. Дані технології були проаналізовані з огляду на сучасні загрози інформаційній безпеці. Було встановлено, що використання тимчасового пароля та електронної пошти не ставить під загрозу загальний рівень інформаційної безпеки системи, але спрощує процес розробки та підтримки системи.

Одним із основних практичних результатів дипломної роботи є розробка та тестування сучасного веб-застосунку з використанням двохфакторної автентифікації. Завдяки цьому дослідженню були досягнуті поставлені цілі та виконані поставлені завдання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Всесвітня мережа. [Електронний ресурс] : Вікіпедія , Стаття “ Всесвітнє павутиння. ” − Режим доступу: World Wide Web . – URL: https://uk.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D1%81%D0%B2%D1%96%D1%82%D0%BD%D1%94_%D0%BF%D0%B0%D0%B2%D1%83%D1%82%D0%B8%D0%BD%D0%BD%D1%8F
2. HTTP – протокол. [Електронний ресурс] :Вікіпедія , Стаття “ HTTP ” – Режим доступу: World Wide Web. – URL: https://uk.wikipedia.org/wiki/HTTP
3. Transport Layer Security. [Електронний ресурс] :Вікіпедія , Стаття “ Transport Layer Security”. – Режим доступу: World Wide Web. – URL: https://en.wikipedia.org/wiki/Transport_Layer_Security
4. Nginx. [Електронний ресурс] : Вікіпедія , Стаття “ Nginx ” – Режим доступу: World Wide Web. – URL: https://uk.wikipedia.org/wiki/Nginx
5. Access token. [Електронний ресурс] : Вікіпедія , Стаття “ Access token. ” – Режим доступу: World Wide Web. – URL: https://en.wikipedia.org/wiki/Access_token
6. 422 Unprocessable Entity. [Електронний ресурс] : MDN Web DOCS , Специфікація “ 422 Unprocessable Entity ” – Режим доступу: World Wide Web. – URL: https://developer.mozilla.org/uk/docs/Web/HTTP/Status/422
7. 401 Unauthorized. [Електронний ресурс] : MDN Web DOCS , Специфікація “ 422 Unprocessable Entity ” – Режим доступу: World Wide Web. – URL: https://developer.mozilla.org/uk/docs/Web/HTTP/Status/401
8. Kupin A. I., Kumchenko Y. O. Usage of training methods to parameterization of multilayer neural computing structures for technological processes //Радіоелектронні і комп’ютерні системи. – 2014. – №. 5. – С. 100-104.
9. JSON Web Token (JWT) . [Електронний ресурс] : tools.ietf.org , Специфікація “ JSON Web Token (JWT) . ” – Режим доступу: World Wide Web. – URL: https://tools.ietf.org/html/rfc7519
10. Das M. L. Two-factor user authentication in wireless sensor networks //IEEE transactions on wireless communications. – 2009. – Т. 8. – №. 3. – С. 1086-1090.

11. Коротка історія мов програмування. URL:http://younglinux.info/python/programminglanguage.php(Дата звернення: 12.06.2018).
12. Wikipedia, Python. URL:https://ua.wikipedia.org/wiki/Python(Дата звернення: 12.06.2018).
13. A Brief Timeline of Python. URL:http://python–history.blogspot.ru/2009/01/brief–timeline–of–python.html(Дата звернення: 12.06.2018).
14. Wikipedia, Історія мови програмування Python. URL:https://ua.wikipedia.org/wiki/Історія_програмування_Python(Дата звернення: 12.06.2018).
15. Короткий огляд мови Python. URL:http://www.helloworld.ru/texts/comp/lang/python/python2/index.htm(Дата звернення: 13.06.2018).
16. Форсьє Дж., Django. Розробка веб-додатків на Python, 2009. -456 с.
17. Django. URL:https://djbook.ru/ch01s03.html(Дата звернення: 13.06.2018).
18. Python Django. URL:http://ep–z.ru/stroitelstvo/sayt/python/python–django(Дата звернення: 14.06.2018).
19. PostrgeSQL. URL:https://habr.com/post/282764/(Дата звернення: 14.06.2018).

ДОДАТОК А

Фрагмент коду

```
public function register(RegisterRequest $request)
    {
        $user = $this->userService->create($request->validated());
        $token = auth()->login($user);
        return $this->respondWithToken($token);
    }
class UserService
{
    /**
     * @var UserRepository
     */
    private $userRepository;
    public function __construct(UserRepository $userRepository)
    {
        $this->userRepository = $userRepository;
    }
    public function create(array $credentials): User
    {
       return  $this->userRepository->create($this->prepareCredentials($credentials));
    }
    protected function prepareCredentials($credentials): array
    {
         $credentials['password'] = bcrypt($credentials['password']);
         return $credentials;
    }
```

ДОДАТОК Б

Фрагмент коду

```
class AuthController extends Controller
{
    /**
     * @var UserService
     */
    private $userService;

    public function __construct(UserService $userService)
    {
        $this->userService = $userService;
    }
    public function login(LoginRequest $request)
    {
        $credentials = $request->validated();
        $otp = md5(time());
        Cache::put("email_verify:" . $credentials['email'] , $otp, 900);

        if (!$token = auth()->attempt($credentials)) {
            return response()->json(['error' => 'Unauthorized'], 401);
        }
        Mail::to($credentials['email'])->send(new
VerifyEmail($credentials['email'], $otp));

        return response()->json(['success' => 'Check Email'],200);
    }
```

```
    public function verify(VerifyRequest $request)
    {
        $credentials = $request->validated();
        $key = "email_verify:" . $credentials['email'];
        $user = $this->getUser($credentials['email']);

        if(Cache::get($key) === $credentials['otp'] && $user) {
            $token =  auth()->tokenById($user->id);
            return  $this->respondWithToken($token);
        };
        return response()->json(['error' => 'Unauthorized'], 401);
    }
    protected function respondWithToken($token)
    {
        return response()->json([
            'access_token' => $token,
            'token_type' => 'bearer',
            'expires_in' => auth()->factory()->getTTL() * 60
        ]);
    }
    protected function getUser($email): ?User
    {
        return User::where(['email' => $email])->first();
```

ДОДАТОК В

Фрагмент коду

```
class LoginTest extends TestCase
{
    /**
```

```
     *
     * @test
     * @return void
     */

    use WithFaker, RefreshDatabase;


    public function a_user_can_login()
    {
        $user = factory(User::class)->create();
        $payload = [
          'email'    => $user->email,
          'password' => 'password',
        ];

        $response = $this->postJson('/api/login', $payload);

        $response->assertStatus(200)
            ->assertJsonStructure(['access_token', 'token_type', 'expires_in']);

    }
}

class RegisterTest extends TestCase
{
    /**
     *
```

```
     * @test
     * @return void
     */

    use WithFaker, RefreshDatabase;

    public function a_user_can_register()
    {
        $this->withoutExceptionHandling();

        $payload = [
            'name'     => 'user',
            'email'    => 'user@user.com',
            'password' => 'password',
        ];

        $response = $this->json('POST', 'api/register', $payload);

        $response->assertStatus(200)
            ->assertJsonStructure(['access_token', 'token_type', 'expires_in']);

        $this->assertDatabaseHas('users', ['email' => $payload['email']]);
    }
```

ДОДАТОК С

Фрагмент коду

```
#!/usr/bin/env python
import os
```

```
import sys

# Set this directory's root on the path
sys.path.append(os.path.dirname(os.path.abspath(os.path.dirname(__file__))))

if __name__ == "__main__":
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "example.settings")

    from django.core.management import execute_from_command_line

    execute_from_command_line(sys.argv)
import os

DEBUG = True

PROJECT_PATH = os.path.abspath(os.path.dirname(__file__))

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(PROJECT_PATH, 'database.sqlite'),
    }
}

STATIC_URL = 'static/'

AUTHENTICATION_BACKENDS = (
    'django.contrib.auth.backends.ModelBackend',
```

```
)

TIME_ZONE = 'Europe/Amsterdam'

# Make this unique, and don't share it with anybody.
SECRET_KEY =
'KJBKBJHVYFUYGKHVGFUYGKHVJFGUYVCFXDFSZDGFVBKJNLKTYRDTRD'

MIDDLEWARE = (
    'django.middleware.common.CommonMiddleware',
    'user_sessions.middleware.SessionMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.middleware.locale.LocaleMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django_otp.middleware.OTPMiddleware',
    'debug_toolbar.middleware.DebugToolbarMiddleware',
    'two_factor.middleware.threadlocals.ThreadLocals',
)

ROOT_URLCONF = 'example.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(PROJECT_PATH, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
```

```
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

INSTALLED_APPS = [
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'user_sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.admin',
    'django_otp',
    'django_otp.plugins.otp_static',
    'django_otp.plugins.otp_totp',
    'django_otp.plugins.otp_email',
    'two_factor',
    'two_factor.plugins.phonenumber',
    'two_factor.plugins.email',
    'example',

    'debug_toolbar',
    'bootstrapform'
```

```
]


LOGOUT_REDIRECT_URL = 'home'
LOGIN_URL = 'two_factor:login'
LOGIN_REDIRECT_URL = 'two_factor:profile'

INTERNAL_IPS = ('127.0.0.1',)

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
        },
    },
    'loggers': {
        'two_factor': {
            'handlers': ['console'],
            'level': 'INFO',
        }
    }
}

TWO_FACTOR_CALL_GATEWAY = 'example.gateways.Messages'
TWO_FACTOR_SMS_GATEWAY = 'example.gateways.Messages'
```

```
PHONENUMBER_DEFAULT_REGION = 'UA'

TWO_FACTOR_REMEMBER_COOKIE_AGE = 120  # Set to 2 minute for testing

TWO_FACTOR_WEBAUTHN_RP_NAME = 'Диплом НУВГП'

SESSION_ENGINE = 'user_sessions.backends.db'

EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
DEFAULT_FROM_EMAIL = 'lkrasit@gmail.com'

SILENCED_SYSTEM_CHECKS = ['admin.E410']

try:
    from .settings_private import *  # noqa
except ImportError:
    pass

CMS_LANGUAGES = {
## Customize this
'default': {
    'public': True,
    'hide_untranslated': False,
    'redirect_on_fallback': True,
},
1: [
    {
```

```
        'name': 'Українська',
        'public': True,
        'hide_untranslated': False,
        'redirect_on_fallback': True,
        'code': 'uk',
    },
    {
        'name': 'English',
        'public': True,
        'hide_untranslated': False,
        'redirect_on_fallback': True,
        'code': 'en',
    },
],
}
from django.conf import settings
from django.contrib import admin
from django.contrib.auth.views import LogoutView
from django.urls import include, path

from two_factor.gateways.twilio.urls import urlpatterns as tf_twilio_urls
from two_factor.urls import urlpatterns as tf_urls

from .views import (
    ExampleSecretView, HomeView, RegistrationCompleteView, RegistrationView,
)

urlpatterns = [
```

```
path(
    '',
    HomeView.as_view(),
    name='home',
),
path(
    'account/logout/',
    LogoutView.as_view(),
    name='logout',
),
path(
    'secret/',
    ExampleSecretView.as_view(),
    name='secret',
),
path(
    'account/register/',
    RegistrationView.as_view(),
    name='registration',
),
path(
    'account/register/done/',
    RegistrationCompleteView.as_view(),
    name='registration_complete',
),
path('', include(tf_urls)),
path('', include(tf_twilio_urls)),
path('', include('user_sessions.urls', 'user_sessions')),
```

```
    path('admin/', admin.site.urls),
]

if settings.DEBUG:
    import debug_toolbar
    urlpatterns += [
        path('__debug__/', include(debug_toolbar.urls)),
    ]
from django.conf import settings
from django.contrib.auth.forms import UserCreationForm
from django.shortcuts import redirect, resolve_url
from django.views.decorators.cache import never_cache
from django.views.generic import FormView, TemplateView

from two_factor.views import OTPRequiredMixin
from two_factor.views.utils import class_view_decorator


class HomeView(TemplateView):
    template_name = 'home.html'


class RegistrationView(FormView):
    template_name = 'registration.html'
    form_class = UserCreationForm

    def form_valid(self, form):
        form.save()
```

```
        return redirect('registration_complete')


class RegistrationCompleteView(TemplateView):
    template_name = 'registration_complete.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['login_url'] = resolve_url(settings.LOGIN_URL)
        return context


@class_view_decorator(never_cache)
class ExampleSecretView(OTPRequiredMixin, TemplateView):
    template_name = 'secret.html'
from django.conf import settings
from django.contrib.admin import AdminSite
from django.contrib.auth import REDIRECT_FIELD_NAME
from django.contrib.auth.views import redirect_to_login
from django.shortcuts import resolve_url
from django.utils.http import url_has_allowed_host_and_scheme

from .utils import monkeypatch_method


class AdminSiteOTPRequiredMixin:
    """
    Mixin for enforcing OTP verified staff users.
```

```
        Custom admin views should either be wrapped using :meth:`admin_view` or
        use :meth:`has_permission` in order to secure those views.
        """

        def has_permission(self, request):
            """
            Returns True if the given HttpRequest has permission to view
            *at least one* page in the admin site.
            """
            if not super().has_permission(request):
                return False
            return request.user.is_verified()

        def login(self, request, extra_context=None):
            """
            Redirects to the site login page for the given HttpRequest.
            """
            redirect_to = request.POST.get(REDIRECT_FIELD_NAME,
request.GET.get(REDIRECT_FIELD_NAME))

            if not redirect_to or not url_has_allowed_host_and_scheme(url=redirect_to,
allowed_hosts=[request.get_host()]):
                redirect_to = resolve_url(settings.LOGIN_REDIRECT_URL)

            return redirect_to_login(redirect_to)


    class AdminSiteOTPRequired(AdminSiteOTPRequiredMixin, AdminSite):
```

```
        """
        AdminSite enforcing OTP verified staff users.
        """
        pass



    def patch_admin():
        @monkeypatch_method(AdminSite)
        def login(self, request, extra_context=None):
            """
            Redirects to the site login page for the given HttpRequest.
            """
            redirect_to = request.POST.get(REDIRECT_FIELD_NAME,
request.GET.get(REDIRECT_FIELD_NAME))


            if not redirect_to or not url_has_allowed_host_and_scheme(url=redirect_to,
allowed_hosts=[request.get_host()]):
                redirect_to = resolve_url(settings.LOGIN_REDIRECT_URL)


            return redirect_to_login(redirect_to)



    def unpatch_admin():
        setattr(AdminSite, 'login', original_login)



    original_login = AdminSite.login
    from django.apps import AppConfig
    from django.conf import settings
```

```
class TwoFactorConfig(AppConfig):
    name = 'two_factor'
    verbose_name = "Django Two Factor Authentication"

    def ready(self):
        if getattr(settings, 'TWO_FACTOR_PATCH_ADMIN', True):
            from .admin import patch_admin
            patch_admin()
from binascii import unhexlify
from time import time

from django import forms
from django.conf import settings
from django.utils.translation import gettext_lazy as _
from django_otp.forms import OTPAuthenticationFormMixin
from django_otp.oath import totp
from django_otp.plugins.otp_totp.models import TOTPDevice

from .plugins.registry import registry
from .utils import totp_digits


class MethodForm(forms.Form):
    method = forms.ChoiceField(label=_("Method"),
                               widget=forms.RadioSelect)
```

```
        def __init__(self, **kwargs):
            super().__init__(**kwargs)

            method = self.fields['method']
            method.choices = [
                (m.code, m.verbose_name) for m in registry.get_methods()
            ]
            method.initial = method.choices[0][0]



    class DeviceValidationForm(forms.Form):
        token = forms.IntegerField(label=_("Token"), min_value=1, max_value=int('9'
* totp_digits()))

        token.widget.attrs.update({'autofocus': 'autofocus',
                                   'inputmode': 'numeric',
                                   'autocomplete': 'one-time-code'})
        error_messages = {
            'invalid_token': _('Entered token is not valid.'),
        }

        def __init__(self, device, **kwargs):
            super().__init__(**kwargs)
            self.device = device

        def clean_token(self):
            token = self.cleaned_data['token']
            if not self.device.verify_token(token):
```

```
            raise forms.ValidationError(self.error_messages['invalid_token'])
        return token


    class TOTPDeviceForm(forms.Form):
        token = forms.IntegerField(label=_("Token"), min_value=0, max_value=int('9'
* totp_digits()))

        token.widget.attrs.update({'autofocus': 'autofocus',
                                   'inputmode': 'numeric',
                                   'autocomplete': 'one-time-code'})

        error_messages = {
            'invalid_token': _('Entered token is not valid.'),
        }

        def __init__(self, key, user, metadata=None, **kwargs):
            super().__init__(**kwargs)
            self.key = key
            self.tolerance = 1
            self.t0 = 0
            self.step = 30
            self.drift = 0
            self.digits = totp_digits()
            self.user = user
            self.metadata = metadata or {}

        @property
```

```
def bin_key(self):
    """
    The secret key as a binary string.
    """
    return unhexlify(self.key.encode())


def clean_token(self):
    token = self.cleaned_data.get('token')
    validated = False
    t0s = [self.t0]
    key = self.bin_key
    if 'valid_t0' in self.metadata:
        t0s.append(int(time()) - self.metadata['valid_t0'])
    for t0 in t0s:
        for offset in range(-self.tolerance, self.tolerance + 1):
            if totp(key, self.step, t0, self.digits, self.drift + offset) == token:
                self.drift = offset
                self.metadata['valid_t0'] = int(time()) - t0
                validated = True
    if not validated:
        raise forms.ValidationError(self.error_messages['invalid_token'])
    return token


def save(self):
    return TOTPDevice.objects.create(user=self.user, key=self.key,
                                     tolerance=self.tolerance, t0=self.t0,
                                     step=self.step, drift=self.drift,
                                     digits=self.digits,
```

```
                                name='default')


class DisableForm(forms.Form):
    understand = forms.BooleanField(label=_("Yes, I am sure"))


class AuthenticationTokenForm(OTPAuthenticationFormMixin, forms.Form):
    otp_token = forms.RegexField(label=_("Token"),
                                 regex=r'^[0-9]*$',
                                 min_length=totp_digits(),
                                 max_length=totp_digits())
    otp_token.widget.attrs.update({
        'autofocus': 'autofocus',
        'pattern': '[0-9]*',  # hint to show numeric keyboard for on-screen keyboards
        'autocomplete': 'one-time-code',
    })

    # Our authentication form has an additional submit button to go to the
    # backup token form. When the `required` attribute is set on an input
    # field, that button cannot be used on browsers that implement html5
    # validation. For now we'll use this workaround, but an even nicer
    # solution would be to move the button outside the `<form>` and into
    # its own `<form>`.
    use_required_attribute = False
    idempotent = False

    def __init__(self, user, initial_device, **kwargs):
```

```
            """
            `initial_device` is either the user's default device, or the backup
            device when the user chooses to enter a backup token. The token will
            be verified against all devices, it is not limited to the given
            device.
            """
            super().__init__(**kwargs)
            self.user = user
            self.initial_device = initial_device

            # Add a field to remember this browser.
            if getattr(settings, 'TWO_FACTOR_REMEMBER_COOKIE_AGE', None):
                if settings.TWO_FACTOR_REMEMBER_COOKIE_AGE < 3600:
                    minutes = int(settings.TWO_FACTOR_REMEMBER_COOKIE_AGE /
60)
                    label = _("Don't ask again on this device for %(minutes)i minutes") %
{'minutes': minutes}
                elif settings.TWO_FACTOR_REMEMBER_COOKIE_AGE < 3600 * 24:
                    hours = int(settings.TWO_FACTOR_REMEMBER_COOKIE_AGE /
3600)
                    label = _("Don't ask again on this device for %(hours)i hours") %
{'hours': hours}
                else:
                    days = int(settings.TWO_FACTOR_REMEMBER_COOKIE_AGE /
3600 / 24)
                    label = _("Don't ask again on this device for %(days)i days") % {'days':
days}

                self.fields['remember'] = forms.BooleanField(
                    required=False,
```

```
                initial=True,
                label=label
            )

    def clean(self):
        self.clean_otp(self.user)
        return self.cleaned_data


class BackupTokenForm(AuthenticationTokenForm):
    otp_token = forms.CharField(label=_("Token"))
from django.apps.registry import apps
from django.urls import include, path

from two_factor.views import (
    BackupTokensView, DisableView, LoginView, ProfileView, QRGeneratorView,
    SetupCompleteView, SetupView,
)

core = [
    path(
        'account/login/',
        LoginView.as_view(),
        name='login',
    ),
    path(
        'account/two_factor/setup/',
        SetupView.as_view(),
```

```
        name='setup',
    ),
    path(
        'account/two_factor/qrcode/',
        QRGeneratorView.as_view(),
        name='qr',
    ),
    path(
        'account/two_factor/setup/complete/',
        SetupCompleteView.as_view(),
        name='setup_complete',
    ),
    path(
        'account/two_factor/backup/tokens/',
        BackupTokensView.as_view(),
        name='backup_tokens',
    ),
]

profile = [
    path(
        'account/two_factor/',
        ProfileView.as_view(),
        name='profile',
    ),
    path(
        'account/two_factor/disable/',
        DisableView.as_view(),
```

```
        name='disable',
    ),
]
```

ДОДАТОК D

Фрагмент коду

```
plugin_urlpatterns = []
for app_config in apps.get_app_configs():
    if app_config.name.startswith('two_factor.plugins.'):
        # Phonenumber used to be include in the two_factor core. Because we
        # don't want to change the url names and break backwards compatibility
        # we keep the urls of the phonenumber plugin in the core two_factor
        # namespace.
        if app_config.name == 'two_factor.plugins.phonenumber':
            namespace = None
        else:
            namespace = app_config.label
        try:
            plugin_urlpatterns.append(
                path(
                    f'account/two_factor/{app_config.url_prefix}/',
                    include(f'{app_config.name}.urls', namespace)
                ),
            )
        except AttributeError:
            pass

urlpatterns = (core + profile + plugin_urlpatterns, 'two_factor')
```