Лабораторна робота №7

Створення веб застосунку та налаштування базової авторизації

Мета: Ознайомитись з основними методами створення React додатків та налаштування базової авторизації та маршрутизації сторінок.

7.1 Теоретичні відомості

Компонент у React — це базова одиниця побудови інтерфейсу, логічно завершений фрагмент UI, який поєднує в собі розмітку, поведінку та, за потреби, стан. З формальної точки зору сучасний React-компонент — це звичайна JavaScript/TypeScript-функція, яка приймає вхідні дані (props) і повертає опис інтерфейсу в вигляді JSX. Уся програма в React розглядається як дерево компонентів, де кореневий компонент (зазвичай App) містить інші компоненти, а ті, своєю чергою, можуть включати ще дрібніші. Таким чином утворюється ієрархічна структура, в якій кожен вузол відповідає за власну ділянку інтерфейсу та інкапсулює пов'язану з нею логіку.

```
function MyComponent(props) {
  return <div>...</div>;
}
```

JSX, який повертає компонент, є декларативним описом того, як має виглядати інтерфейс у конкретний момент часу. Це не HTML, а синтаксичний цукор над викликами React.createElement, який дозволяє описувати структуру UI засобами мови JavaScript/TypeScript. У JSX кожен компонент позначається тегом з назвою, що починається з великої літери. Це принципове правило: теги, які починаються з малої літери, трактуються як елементи DOM (наприклад, div, button), а теги з великої — як користувацькі компоненти (LoginForm, Dashboard). Кожен компонент у JSX повинен повертати один кореневий елемент (або фрагмент), тому вся структура, яку він описує, логічно згрупована в єдиний вузол. React будує внутрішнє представлення цього дерева (так званий Virtual DOM) і, при зміні стану чи вхідних даних, обчислює мінімальний набір змін, які потрібно застосувати до реального DOM у браузері.

```
const element = <h1>Hello, React</h1>;
```

Один компонент **повертає один кореневий елемент** (можна використовувати <>...</> – React Fragment); JSX працює **не з реальним DOM**, а з **віртуальним DOM** (Virtual DOM), який React синхронізує з браузером.

Вхідні дані компонента називаються props (properties). З теоретичної точки зору, props — це параметри функції-компонента. Вони задають зовнішній інтерфейс компонента: які дані він очікує отримати "ззовні", які значення вміє відображати, які callback-функції викликати у відповідь на події. Потік даних у React за замовчуванням односторонній: батьківський компонент передає props дочірньому, тобто дані рухаються згори вниз по дереву компонентів. Сам компонент не має права змінювати свої props, він лише їх читає і на їх основі формує JSX. Це забезпечує передбачуваність: якщо компонент розглядати абстрактно як математичну функцію UI = f(props, state), то props — це одна з вхідних змінних, контрольована оточенням, а не самим компонентом.

```
type ButtonProps = {
  label: string;
};

export const Button: React.FC<ButtonProps> = ({ label }) => {
  return <button>{label}</button>;
```

```
};

<Button label="Login" />
<Button label="Register" />
```

Окремим важливим аспектом є побічні ефекти та життєвий цикл компонента. Сам по собі акт рендерингу (обчислення JSX) має бути по можливості “чистим”, тобто без безпосередніх побічних дій: не варто всередині тіла компонента виконувати запити до мережі, запис в localStorage, прямі маніпуляції DOM тощо. Для цього в функціональних компонентах використовується хук useEffect, який дозволяє заявити ефект, що повинен бути виконаний після рендерингу або при зміні певних залежностей. У термінах життєвого циклу це те місце, де компонент “взаємодіє з зовнішнім світом”: завантажує дані з API, підписується на події, налаштовує таймери, а при демонтажі — скасовує підписки, очищує ресурси. Теоретично useEffect дає можливість чітко розділити обчислювальну частину компонента (перетворення props + state у JSX) і частину, пов’язану з побічними діями.

```
import { useEffect, useState } from "react";

export const HealthStatus: React.FC = () => {
  const [status, setStatus] = useState("checking...");

  useEffect(() => {
    fetch("/api/health_check")
      .then((res) => res.json())
      .then(() => setStatus("online"))
      .catch(() => setStatus("offline"));
  }, []); // [] означає: виконати ефект один раз після першого рендеру

  return <div>API status: {status}</div>;
};
```

Створення компонентів у React тісно пов’язане з принципом композиції. Ідея полягає в тому, що замість написання монолітних блоків інтерфейсу, де у одному фрагменті коду змішуються розмітка, логіка, стилі та взаємодія з сервером, застосунок ділиться на невеликі, добре визначені компоненти. Кожен з них відповідає за конкретний фрагмент UI (наприклад, форма авторизації, навігаційна панель, картка користувача, блок дашборду) і може бути перевикористаний у різних місцях. Це відповідає загальним принципам модульності та повторного використання в програмуванні: компонент виступає як модуль з чітким контрактом (описаний через тип props), а вся система — як композиція таких модулів.

```
export const Page: React.FC = () => {
  return (
    <>
      <Header />
      <Dashboard />
      <Footer />
    </>
  );
};
```

Ще один важливий теоретичний момент — це використання props.children для реалізації композиції “вмісту в контейнері”. Компонент може не лише отримувати набір явних параметрів (рядки, числа, callback-функції), але й приймати як вміст інші React-елементи.

Такий підхід дозволяє будувати абстракції на рівні макета: наприклад, створити універсальний компонент Card, який визначає рамки, фон, заголовок, але не знає наперед, що саме буде всередині. Цей внутрішній вміст передається йому як children. Теоретично це реалізація патерна “інверсії контролю” в UI: батьківський компонент задає структуру, а конкретний вміст підсовується йому ззовні, що робить систему максимально гнучкою.

fetch у сучасному JavaScript — це вбудований браузерний (і, частково, серверний у оточенні типу Node з polyfill’ами) інтерфейс для виконання HTTP-запитів. Теоретично його можна розглядати як абстракцію над мережевою взаємодією, яка реалізує модель “клієнт надсилає запит – сервер повертає відповідь”, а з боку JS надає промісоорієнтований інтерфейс. На відміну від старішого XMLHttpRequest, fetch має декларативніший синтаксис, працює виключно асинхронно, повертає Promise, а також оперує чітко визначеними сутностями: Request, Response, Headers, Body. Базовий виклик fetch(url) повертає Promise<Response>. Це означає, що результатом є не “готові дані”, а об’єкт Response, який описує відповідь сервера: статусний код, заголовки, тіло (body), методи для читання тіла в різних форматах (text(), json(), blob() тощо). Важливий момент: проміс, який повертає fetch, переходить у стан fulfilled навіть тоді, коли сервер відповів HTTP-помилкою (наприклад, 404 або 500). Для нього це “успішна доставка HTTP-відповіді”, а не логічна помилка бізнес-рівня. Відхилення промісу (reject) відбувається тільки при мережевих помилках (немає з’єднання, таймаут, зірваний TLS, CORS-блокування тощо). Це принципова відмінність і важлива теоретична деталь: обробка HTTP-коду статусу є окремою логікою, яку розробник має реалізовувати самостійно (наприклад, перевіряючи response.ok або response.status).

Об’єкт Response, що повертається fetch, інкапсулює інформацію про статус відповіді (status, statusText), заголовки (headers), а також тіло відповіді. Поле ok є зручним логічним прапором, який дорівнює true, якщо статус коду знаходиться в діапазоні 200–299. Тіло відповіді є потоковим (streaming) і реалізує інтерфейс Body. Теоретично тіло можна прочитати тільки один раз: методи response.text(), response.json(), response.blob() повертають проміс, який при першому виклику читає потік до кінця, і надалі тіло вважається спожитим. Це означає, що повторно викликати інший метод для того самого Response не можна, якщо ви не клонували його наперед (response.clone()). Така модель відповідає концепції одноразового читання потоку й важлива з погляду ефективності та контролю над ресурсами. Другим ключовим об’єктом є Request. Хоча найчастіше fetch викликають у короткій формі fetch(url, options), теоретично під капотом можна створювати й явно передавати об’єкт Request, який описує повний HTTP-запит: URL, метод (GET, POST, PUT, DELETE тощо), заголовки (Headers), тіло (body), режим CORS, налаштування кешу, креденшіали (cookies, авторизація) тощо. У параметрі options ми задаємо ці характеристики: method, headers, body, mode, credentials, cache, redirect, referrerPolicy тощо. Headers при цьому є окремим об’єктом, який інкапсулює набір HTTP-заголовків. Він реалізує інтерфейс ітерації та методи get, set, append, has, що дозволяє працювати з заголовками як зі словником, але з урахуванням специфіки HTTP (некейссенситивність ключів тощо).

Важливим теоретичним аспектом використання fetch є інтеграція з моделлю асинхронного програмування в JS. Оскільки fetch повертає Promise, він природно поєднується з async/await. З точки зору "сухої" теорії, async/await — це синтаксичне спрощення роботи з промісами: ключове слово await дозволяє записати асинхронну операцію в стилі “послідовного коду”, але під капотом усе так само залишається неблокуючим. Тому типовий патерн обробки fetch виглядає як const response = await fetch(...); const data = await response.json();. Тут перший await чекає приходу HTTP-відповіді, другий — повного читання й парсингу тіла як JSON.

fetch тісно пов’язаний із політикою безпеки браузера, зокрема з **same-origin policy** та CORS (Cross-Origin Resource Sharing). За замовчуванням браузер блокує спроби читати

відповіді з іншого домену, схеми чи порту, якщо сервер явно не дозволив це через CORS-заголовки (Access-Control-Allow-Origin та інші). Параметр mode у fetch може бути cors, no-cors або same-origin. У звичайному веб-застосунку типове значення — cors, яке дозволяє робити кросдоменно запити, але доступ до відповіді буде дозволений тільки тоді, коли сервер правильно налаштований на CORS. Параметр credentials визначає, чи будуть надсилатися кукі та інші “облікові дані”: omit (не надсилати), same-origin (тільки для того ж походження) або include (завжди надсилати, якщо не заборонено CORS). Теоретично це важливо, тому що fetch за замовчуванням (на відміну від деяких старіших API) не завжди автоматично підставляє cookies при кросдоменних запитах; контроль над цим винесено в параметр credentials, щоб уникнути неочевидних витоків даних. Ще одна суттєва частина теорії — управління життєвим циклом запиту. Стандартний fetch не має вбудованого таймаута, але дозволяє переривати запит через AbortController. AbortController надає об’єкт signal, який передається в fetch; якщо викликати controller.abort(), запит буде перервано, і проміс fetch перейде в стан відхилення з помилкою типу DOMException з ім’ям "AbortError". Таким чином, у теоретичній моделі fetch підтримує концепцію скасування (cancellation), що важливо для складних SPA, де користувач може часто змінювати контекст (наприклад, швидко переходити між сторінками) і немає сенсу завершувати старі запити.

На рівні протоколу fetch орієнтований на HTTP(S), але надає досить узагальнений інтерфейс для роботи з запитами та відповідями. Наприклад, об’єкт Response можна створювати вручну (через конструктор new Response(...)), а не лише отримувати з мережі. Це робиться у сервіс-воркерах та інших низькорівневих сценаріях, де розробник сам хоче керувати тим, що повертається у відповідь на певні запити, кешувати результати, підміняти відповіді тощо. З теоретичної точки зору, API fetch є реалізацією так званої "Fetch Standard" — окремої специфікації, яка описує, як саме мають поводитись запити, відповіді, заголовки, статуси, перехоплення та перенаправлення, з урахуванням вимог безпеки й сумісності у веб-середовищі.

З точки зору обробки помилок, важливо розрізняти кілька рівнів. На найнижчому рівні знаходяться мережеві збої: втрата з’єднання, DNS-помилки, блокування запиту політикою безпеки — у таких випадках fetch відхиляє проміс, і розробник повинен використовувати try/catch або метод catch у ланцюжку промісів. На наступному рівні знаходяться помилки протоколу HTTP: статуси 4xx і 5xx. Вони не зумовлюють відхилення промісу, але семантично означають, що запит не був “успішним” з точки зору бізнес-логіки. Теоретично правильним є явно перевіряти response.ok або конкретний response.status і вже в залежності від цього формувати логіку: показувати повідомлення користувачу, повторювати запит, перенаправляти на сторінку логіну тощо. На ще вищому рівні можна розглядати помилки формату даних: наприклад, якщо очікувався JSON певної структури, але насправді сервер повернув інший формат або напівпорожнє тіло — це вже логічна помилка на рівні прикладного протоколу між клієнтом і сервером, яка також має бути відловлена окремо.Таким чином, з теоретичної точки зору, fetch — це уніфікований, промісоорієнтований інтерфейс доступу до HTTP-ресурсів, побудований на чітко визначених об’єктах (Request, Response, Headers, Body) і тісно інтегрований із моделлю безпеки веб-браузера (same-origin policy, CORS, робота з креденшіалами). Він не є “магією” для отримання JSON, а радше низькорівневою, але зручною абстракцією над мережею, яка вимагає від розробника явного опрацювання статусів, форматів даних, помилок і життєвого циклу запитів.

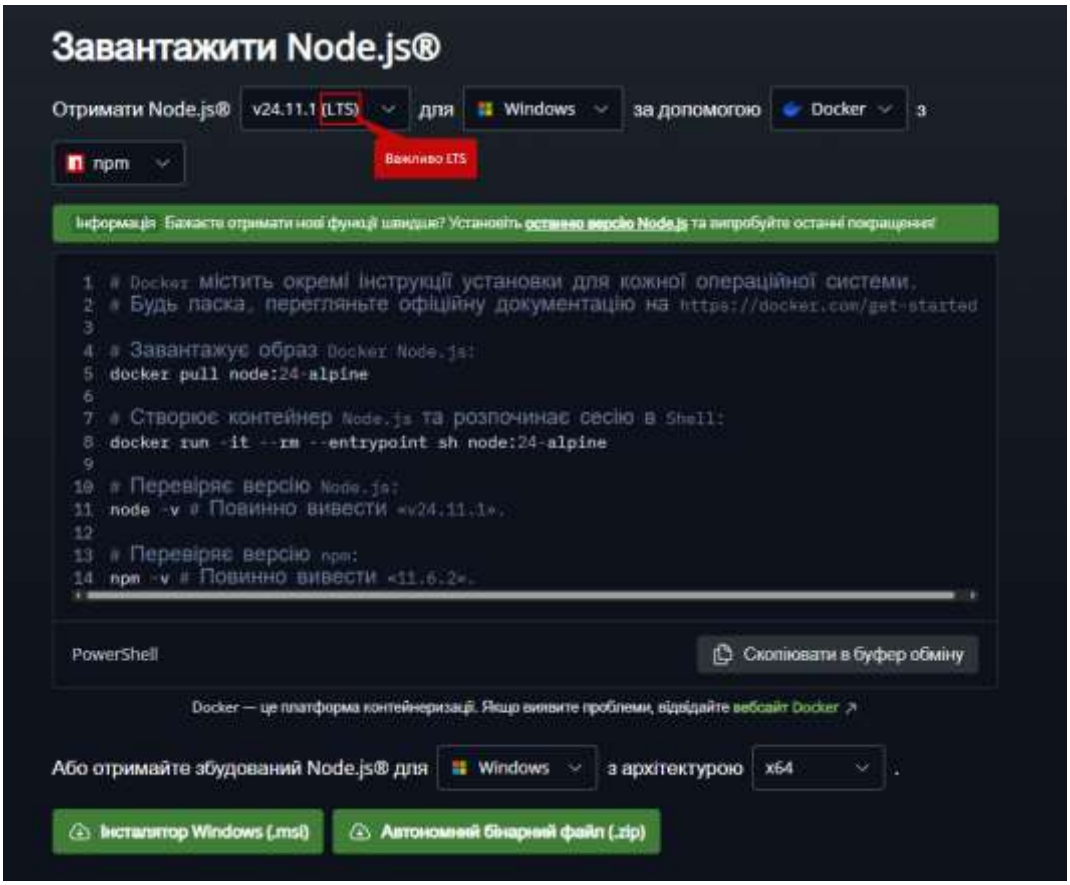
```
async function loadUsers() {
  const res = await fetch("https://api.example.com/users");


  if (!res.ok) {
    throw new Error(`Request failed with status ${res.status}`);
  }
```

```
  const data = await res.json(); // парсимо тіло як JSON
  console.log("Users:", data);
}
```

Більше інформації TS - https://www.typescriptlang.org/docs/, React - https://react.dev/learn

7.2 Підготовка до роботи

1. Встановити Node завантаживши інсталятор **nodejs.org**



2. Відкрити проект що був створений на лабораторній роботі 6 за домогою GitBash та перейти на гілку main (git checkout main) і стягнути зміни (git pull), зміни на гілці main відобразяться лише після того як буде схвалено попередній PR викладачем та гілку буде змерджено, після чого створити нову гілку для реалізації нового функціоналу відповідно назвавши її та перейти на неї. Після біля папки backend створити ще одну папку frontend.

```
s@DESKTOP-D7FK170 MINGW64 ~/Desktop/lab/project (feature/auth_back)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

s@DESKTOP-D7FK170 MINGW64 ~/Desktop/lab/project (main)
$ git pull
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (1/1), 905 bytes | 100.00 KiB/s, done.
From https://github.com/andrii910320/project
   89c6e11..a89aff4  main       -> origin/main
Updating 89c6e11..a89aff4
Fast-forward
 backend/app/__init__.py                          |  18 ++++++++++
 backend/app/__pycache__/__init__.cpython-311.pyc | Bin 0 -> 1197 bytes
 backend/app/__pycache__/config.cpython-311.pyc   | Bin 0 -> 926 bytes
 backend/app/api/__pycache__/auth.cpython-311.pyc | Bin 0 -> 2514 bytes
```

3. У створеній папці за допомогою npm create vite@latest . -- --template react-ts створити новий проект. (Використовуючи «.» проект буде створено в поточній папці) Обрати в запропонованих інструкціях установки:

```
$ npm create vite@latest . -- --template react-ts
Need to install the following packages:
create-vite@8.2.0
Ok to proceed? (y) y


> npx
> create-vite . --template react-ts

◇  Use rolldown-vite (Experimental)?:
   No

◇  Install with npm and start now?
   Yes

◇  Scaffolding project in C:\Users\Dr.Mays\Desktop\lab\project\frontend...

◇  Installing dependencies with npm...
```

Після завершення встановлення перейти на створений сайт та протестувати роботу:

```
◇  Starting dev server...

> frontend@0.0.0 dev
> vite


  VITE v7.3.0  ready in 1541 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

4. Встановити бібліотеку маршрутизації npm install react-router-dom;

5. Самостійно обрати дизайн та палітру кольорів для майбутнього веб-застосунку.

6. В створеному проекті знайти файл tsconfig.app.json в коді знайти

erasableSyntaxOnly: true та виправити на false

7.3 Порядок виконання роботи

1. В src/App.tsx реалізувати базову маршутизацію застосунку, а саме сторінку Login, Registration, DashboardPage та заглушку на випадок невідомих роутів NotFound(червоні підкреслення на import це нормально так як компоненти ще не реалізовані):

```
import type { FC } from "react";
import { Route, Routes } from "react-router-dom";
import { Registration } from "./pages/Registration";
import { Login } from "./pages/Auth";
import { DashboardPage } from "./pages/Dashboard";
import { NotFound } from "./pages/404";

export const App: FC = () => {
 return (
  <>
   <main>
    <Routes>
     <Route path="/login" element={<Login />} />
     <Route path="/registration" element={<Registration />} />
     <Route path="/dashboard" element={<DashboardPage />} />
     <Route path="*" element={<NotFound />} />
    </Routes>
   </main>
  </>
 );
};

export default App;
```

1.1 Для того щоб роутинг на сайті коректно працював потрібно обгорнути точку входу BrowserRouter, тому оновлений код для main.tsx виглядає так:

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.tsx'
import { BrowserRouter } from 'react-router-dom'

createRoot(document.getElementById('root')!).render(
 <StrictMode>
  <BrowserRouter>
   <App />
  </BrowserRouter>
 </StrictMode>,
)
```

1.2 Обнулити CSS стилі для усього проету в index.css:

```
* {
 margin: 0;
 padding: 0;
 box-sizing: border-box;
}
```

1.3 Створити файл .env в корні папки frontend та записати глобальні змінні проекту: VITE_API_URL та інші та додати файл .env у вийняткі .gitignore:

```
VITE_API_URL="http://127.0.0.1:5000"
```

2. Створити в src папки components, enums, interfaces, pages, api та реалізувати усі необхідні сторінки та компоненти:

src/pages/404.tsx

```
import { NotFoundPage } from "../components/NotFound";


export const NotFound = () => {
  return (
    <>
      <NotFoundPage />
    </>
  );
};
```

src/pages/Auth.tsx

```
import { LoginForm } from "../components/LoginForm";
import { NavBarAuth } from "../components/NavBarAuth";

export const Login = () => {
  return (
    <>
      <NavBarAuth></NavBarAuth>
      <LoginForm />
    </>
  );
};
```

src/pages/Dashboard.tsx

```
import { Dashboard } from "../components/Dashboard";

export const DashboardPage = () => {
  return (
    <div>
      <Dashboard />
    </div>
  );
};
```

src/pages/Registration.tsx

```
import { NavBarAuth } from "../components/NavBarAuth";
import { RegistrationForm } from "../components/RegistrationForm";

export const Registration = () => {
  return (
    <>
      <NavBarAuth></NavBarAuth>
      <RegistrationForm></RegistrationForm>
    </>
  );
};
```

3. Реалізувати потрібні компоненти для наявних сторінок в папці components. Кожна компонента знаходиться в окремій папці з назвою компоненти та містить файл index.tsx та style.module.css

components/ Dashboard / index.tsx – буде реалізовано а наступній роботі, зараз просто заглушка без стилів.

```
import React from "react";

export const Dashboard: React.FC = () => {
```

```
  return (
    <div >
      <h1>Dashboard</h1>

    </div>
  );
};
```

components/LoginForm/index.tsx

```
import React, { useState } from "react";
import s from "./style.module.css";
import type { IFormAuth } from "../../interfaces/forms";
import { login } from "../../api/auth/login";
import { Routes } from "../../enums/routes";
import { useNavigate } from "react-router-dom";
export const LoginForm = () => {
  const [userForm, setUserForm] = useState<IFormAuth>({
    username: "",
    password: "",
  });
  const [error, setError] = useState<string | null>(null);
  const navigation = useNavigate();

  const handleClick = (e: React.ChangeEvent<HTMLInputElement>) => {
    const { name, value } = e.target;
    setUserForm((prev) => ({ ...prev, [name]: value }));
  };

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    const result = await login(userForm);
    console.log(result);
    if ("error" in result) {
      setError("Incorrect password/login");
    } else {
      navigation(Routes.DASHBOARD);
    }
  };

  return (
    <div className={s.wrapper}>
      <div className={s.card}>
        <h2 className={s.title}>Authorization</h2>
        <p className={s.subtitle}>Log in or register to continue.</p>

        <div className={s.fields}>
          <input
            name="username"
            type="text"
            placeholder="Username"
            className={s.input}
            value={userForm.username}
            onChange={handleClick}
          />
          <input
            name="password"
            type="password"
            placeholder="Password"
            value={userForm.password}
            className={s.input}
            onChange={handleClick}
          />
        </div>

        <button className={s.button} onClick={handleSubmit}>
```

```
        Continue
       </button>

       <p className={s.hint}>
        Use a strong password that you don&apos;t reuse on other services.
       </p>
       {error && (
        <p className={s.hint} style={{ color: "#f97373" }}>
         {error}
        </p>
       )}
      </div>
    </div>
  );
};
```

components/LoginForm/style.module.css

```
.wrapper {
  min-height: calc(100vh - 64px);
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 2rem 1rem;
  background: radial-gradient(circle at top, #020617 0, #020617 40%, #000 100%);
}

.card {
  width: 100%;
  max-width: 420px;
  padding: 2rem 2.25rem;
  border-radius: 1.25rem;
  background: rgba(15, 23, 42, 0.85);
  border: 1px solid rgba(148, 163, 184, 0.45);
  box-shadow:
    0 24px 60px rgba(0, 0, 0, 0.9),
    0 0 0 1px rgba(15, 23, 42, 0.8);
  backdrop-filter: blur(18px);
  color: #e5e7eb;
  font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI",
    sans-serif;
}

.title {
  text-align: center;
  font-size: 1.5rem;
  font-weight: 650;
  margin-bottom: 0.25rem;
}

.subtitle {
  font-size: 0.9rem;
  text-align: center;
  color: #9ca3af;
  margin-bottom: 1.5rem;
}

.fields {
  display: flex;
  flex-direction: column;
  gap: 0.75rem;
  margin-bottom: 1.5rem;
}

.input {
  padding: 0.7rem 0.9rem;
```

```
  border-radius: 0.75rem;
  border: 1px solid rgba(148, 163, 184, 0.45);
  background: rgba(15, 23, 42, 0.8);
  color: #e5e7eb;
  font-size: 0.95rem;
  outline: none;
  transition:
    border-color 0.18s ease,
    box-shadow 0.18s ease,
    background 0.18s ease,
    transform 0.08s ease;
}

.input::placeholder {
  color: #6b7280;
}

.input:focus {
  border-color: rgba(59, 130, 246, 0.85);
  background: rgba(15, 23, 42, 0.95);
  box-shadow:
    0 0 0 1px rgba(59, 130, 246, 0.55),
    0 16px 45px rgba(15, 23, 42, 0.9);
  transform: translateY(-1px);
}

.button {
  width: 100%;
  padding: 0.75rem 1rem;
  margin-top: 0.25rem;
  border-radius: 999px;
  border: none;
  background: linear-gradient(135deg, #3b82f6, #6366f1);
  color: #f9fafb;
  font-size: 0.98rem;
  font-weight: 600;
  letter-spacing: 0.02em;
  cursor: pointer;
  box-shadow:
    0 10px 30px rgba(37, 99, 235, 0.65),
    0 0 0 1px rgba(15, 23, 42, 0.9);

  transition:
    transform 0.1s ease,
    box-shadow 0.18s ease,
    background 0.18s ease;
}

.button:hover {
  background: linear-gradient(135deg, #2563eb, #4f46e5);
  box-shadow:
    0 16px 40px rgba(37, 99, 235, 0.8),
    0 0 0 1px rgba(15, 23, 42, 0.95);
  transform: translateY(-1px);
}

.button:active {
  transform: translateY(0);
  box-shadow:
    0 8px 20px rgba(15, 23, 42, 0.95),
    0 0 0 1px rgba(15, 23, 42, 1);
}

.hint {
  text-align: center;
```

```
  margin-top: 0.75rem;
  font-size: 0.8rem;
  color: #6b7280;
}

@media (max-width: 480px) {
  .card {
    padding: 1.5rem 1.25rem;
    border-radius: 1rem;
  }

  .title {
    font-size: 1.35rem;
  }
}
```

components/NavBarAuth/index.tsx

```
import { Link } from "react-router-dom";
import s from "./style.module.css";

export const NavBarAuth = () => {
  return (
    <nav className={s.nav}>
      <Link to="/login" className={s.link}>
        Login
      </Link>
      <Link to="/registration" className={s.link}>
        Registration
      </Link>
    </nav>
  );
};
```

components/NavBarAuth/style.module.css

```
.nav {
  position: sticky;
  top: 0;
  z-index: 100;
  display: flex;
  justify-content: flex-end;
  align-items: center;
  gap: 1rem;

  padding: 0.75rem 2rem;

  background: radial-gradient(circle at top left, #1f2937, #020617);
  box-shadow: 0 10px 25px rgba(0, 0, 0, 0.4);

  font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI",
    sans-serif;
}

.link {
  position: relative;
  display: inline-flex;
  align-items: center;
  justify-content: center;
  padding: 0.45rem 1.1rem;
  border-radius: 999px;
  font-size: 0.95rem;
  font-weight: 500;
  letter-spacing: 0.02em;
  color: #e5e7eb;
  text-decoration: none;
  border: 1px solid rgba(148, 163, 184, 0.35);
```

```
  background: rgba(15, 23, 42, 0.6);
  backdrop-filter: blur(8px);
  transition:
    background 0.2s ease,
    border-color 0.2s ease,
    color 0.2s ease,
    transform 0.12s ease,
    box-shadow 0.2s ease;
}

.link:hover {
  background: rgba(59, 130, 246, 0.16);
  border-color: rgba(59, 130, 246, 0.7);
  box-shadow: 0 0 0 1px rgba(59, 130, 246, 0.25),
    0 8px 20px rgba(15, 23, 42, 0.9);
  transform: translateY(-1px);
}

.link:active {
  transform: translateY(0);
  box-shadow: 0 4px 10px rgba(15, 23, 42, 0.8);
}

.nav a:first-of-type {
  background: rgba(15, 23, 42, 0.3);
}

.nav a:last-of-type {
  background: linear-gradient(135deg, #3b82f6, #6366f1);
  border-color: transparent;
  color: #f9fafb;
}

.nav a:last-of-type:hover {
  background: linear-gradient(135deg, #2563eb, #4f46e5);
}
```

components/NotFound/index.tsx

```
import type { FC } from "react";
import { Link } from "react-router-dom";
import s from "./style.module.css";

export const NotFoundPage: FC = () => {
  return (
    <div className={s.wrapper}>
      <div className={s.card}>
        <div className={s.code}>404</div>
        <h1 className={s.title}>Page not found</h1>

        <div className={s.actions}>
          <Link to="/login" className={s.secondary}>
            Login
          </Link>
        </div>

        <div className={s.ghost} aria-hidden="true">
          <div className={s.ghostBody}></div>
          <div className={s.ghostEyes}>
            <span></span>
            <span></span>
          </div>
        </div>
      </div>
    </div>
  );
```

```
};
```

components/NotFound/style.module.css

```
.wrapper {
  min-height: calc(100vh - 64px);
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 2rem 1rem;
  background: radial-gradient(circle at top, #020617 0, #020617 40%, #000 100%);
  font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI",
    sans-serif;
}

.card {
  width: 100%;
  max-width: 480px;

  padding: 2.5rem 2.5rem 2rem;
  border-radius: 1.5rem;

  background: rgba(15, 23, 42, 0.9);
  border: 1px solid rgba(148, 163, 184, 0.45);
  box-shadow:
    0 24px 60px rgba(0, 0, 0, 0.9),
    0 0 0 1px rgba(15, 23, 42, 0.85);

  backdrop-filter: blur(20px);
  text-align: center;
  color: #e5e7eb;
}

.code {
  font-size: 3.5rem;
  font-weight: 800;
  letter-spacing: 0.12em;
  color: #1d4ed8;
  text-shadow:
    0 0 18px rgba(59, 130, 246, 0.7),
    0 0 40px rgba(59, 130, 246, 0.8);
  margin-bottom: 0.25rem;
}

.title {
  font-size: 1.5rem;
  font-weight: 650;
  margin-bottom: 0.5rem;
}

.actions {
  display: flex;
  justify-content: center;
  gap: 0.75rem;
  margin-bottom: 2rem;
}

.primary,
.secondary {
  display: inline-flex;
  align-items: center;
  justify-content: center;
```

```
  padding: 0.6rem 1.4rem;
  border-radius: 999px;
  font-size: 0.9rem;
  font-weight: 500;
  text-decoration: none;
  cursor: pointer;

  transition:
    transform 0.1s ease,
    box-shadow 0.18s ease,
    background 0.18s ease,
    border-color 0.18s ease,
    color 0.18s ease;
}

.primary {
  background: linear-gradient(135deg, #3b82f6, #6366f1);
  color: #f9fafb;
  box-shadow:
    0 10px 30px rgba(37, 99, 235, 0.7),
    0 0 0 1px rgba(15, 23, 42, 0.9);
  border: none;
}

.primary:hover {
  background: linear-gradient(135deg, #2563eb, #4f46e5);
  box-shadow:
    0 16px 40px rgba(37, 99, 235, 0.85),
    0 0 0 1px rgba(15, 23, 42, 0.95);
  transform: translateY(-1px);
}

.primary:active {
  transform: translateY(0);
  box-shadow:
    0 8px 24px rgba(15, 23, 42, 0.9),
    0 0 0 1px rgba(15, 23, 42, 1);
}

.secondary {
  border: 1px solid rgba(148, 163, 184, 0.5);
  background: rgba(15, 23, 42, 0.85);
  color: #e5e7eb;
}

.secondary:hover {
  border-color: rgba(59, 130, 246, 0.9);
  background: rgba(15, 23, 42, 0.95);
  box-shadow:
    0 0 0 1px rgba(59, 130, 246, 0.6),
    0 12px 28px rgba(15, 23, 42, 0.9);
  transform: translateY(-1px);
}

.secondary:active {
  transform: translateY(0);
  box-shadow:
    0 8px 20px rgba(15, 23, 42, 0.9),
    0 0 0 1px rgba(15, 23, 42, 1);
}
```

```
.ghost {
  position: relative;
  width: 80px;
  height: 80px;
  margin: 0 auto;
  opacity: 0.8;
  animation: float 3s ease-in-out infinite;
}

.ghostBody {
  width: 80px;
  height: 60px;
  background: radial-gradient(circle at 30% 20%, #1f2937, #020617);
  border-radius: 40px 40px 30px 30px;
}

.ghostEyes {
  position: absolute;
  top: 22px;
  left: 22px;
  display: flex;
  gap: 16px;
}

.ghostEyes span {
  width: 9px;
  height: 9px;
  border-radius: 999px;
  background: #e5e7eb;
  box-shadow: 0 0 10px rgba(148, 163, 184, 0.9);
}

@keyframes float {
  0% {
    transform: translateY(0);
  }
  50% {
    transform: translateY(-6px);
  }
  100% {
    transform: translateY(0);
  }
}

@media (max-width: 480px) {
  .card {
    padding: 2rem 1.5rem 1.75rem;
  }

  .code {
    font-size: 3rem;
  }

  .title {
    font-size: 1.35rem;
  }

  .actions {
    flex-direction: column;
  }
```

```
  .primary,
  .secondary {
    width: 100%;
  }
}
```

components/RegistrationForm/index.tsx

```
import { useState } from "react";
import s from "./style.module.css";
import { register } from "../../api/auth/registration";
import { useNavigate } from "react-router-dom";
import { Routes } from "../../enums/routes";

export const RegistrationForm: React.FC = () => {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [repeatPassword, setRepeatPassword] = useState("");
  const navigation = useNavigate();
  const [error, setError] = useState<string | null>(null);

  const handleSubmit = async (e: any) => {
    e.preventDefault();
    setError(null);

    if (!username || !password || !repeatPassword) {
      setError("Fill all inputs");
      return;
    }

    if (password !== repeatPassword) {
      setError("Password doesn't match");
      return;
    }

    const result = await register({ username, password });

    if ("error" in result) {
      if (result.status === 409) {
        setError("User already exist");
      } else {
        setError(result.error);
      }
    } else {
      alert("Успішно зареєстровано!");
      setTimeout(() => {
        navigation(Routes.LOGIN);
      }, 5000);
    }
  };

  return (
    <div className={s.wrapper}>
      <form className={s.card} onSubmit={handleSubmit}>
        <h2 className={s.title}>Registration</h2>
        <p className={s.subtitle}>Create new account.</p>

        <div className={s.fields}>
          <input
            name="username"
            type="text"
            placeholder="Username"
            className={s.input}
            value={username}
            onChange={(e) => setUsername(e.target.value)}
          />
          <input
```

```
            name="password"
            type="password"
            placeholder="Password"
            className={s.input}
            value={password}
            onChange={(e) => setPassword(e.target.value)}
          />
          <input
            name="repeatPassword"
            type="password"
            placeholder="Repeat password"
            className={s.input}
            value={repeatPassword}
            onChange={(e) => setRepeatPassword(e.target.value)}
          />
        </div>

        <button className={s.button} type="submit">
          Registration
        </button>
        {error && (
          <p className={s.hint} style={{ color: "#f97373" }}>
            {error}
          </p>
        )}
      </form>
    </div>
  );
};
```

components/RegistrationForm/index.tsx

```
.wrapper {
  min-height: calc(100vh - 64px);
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 2rem 1rem;
  background: radial-gradient(circle at top, #020617 0, #020617 40%, #000 100%);
}

.card {
  width: 100%;
  max-width: 420px;
  padding: 2rem 2.25rem;
  border-radius: 1.25rem;
  background: rgba(15, 23, 42, 0.85);
  border: 1px solid rgba(148, 163, 184, 0.45);
  box-shadow:
    0 24px 60px rgba(0, 0, 0, 0.9),
    0 0 0 1px rgba(15, 23, 42, 0.8);

  backdrop-filter: blur(18px);
  color: #e5e7eb;
  font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI",
    sans-serif;
}

.title {
  text-align: center;
  font-size: 1.5rem;
  font-weight: 650;
  margin-bottom: 0.25rem;
}

.subtitle {
  font-size: 0.9rem;
```

```
  text-align: center;
  color: #9ca3af;
  margin-bottom: 1.5rem;
}

.fields {
  display: flex;
  flex-direction: column;
  gap: 0.75rem;
  margin-bottom: 1.5rem;
}

.input {
  padding: 0.7rem 0.9rem;
  border-radius: 0.75rem;
  border: 1px solid rgba(148, 163, 184, 0.45);
  background: rgba(15, 23, 42, 0.8);
  color: #e5e7eb;
  font-size: 0.95rem;
  outline: none;

  transition:
    border-color 0.18s ease,
    box-shadow 0.18s ease,
    background 0.18s ease,
    transform 0.08s ease;
}

.input::placeholder {
  color: #6b7280;
}

.input:focus {
  border-color: rgba(59, 130, 246, 0.85);
  background: rgba(15, 23, 42, 0.95);
  box-shadow:
    0 0 0 1px rgba(59, 130, 246, 0.55),
    0 16px 45px rgba(15, 23, 42, 0.9);
  transform: translateY(-1px);
}

.button {
  width: 100%;
  padding: 0.75rem 1rem;
  margin-top: 0.25rem;
  border-radius: 999px;
  border: none;
  background: linear-gradient(135deg, #3b82f6, #6366f1);
  color: #f9fafb;
  font-size: 0.98rem;
  font-weight: 600;
  letter-spacing: 0.02em;
  cursor: pointer;
  box-shadow:
    0 10px 30px rgba(37, 99, 235, 0.65),
    0 0 0 1px rgba(15, 23, 42, 0.9);

  transition:
    transform 0.1s ease,
    box-shadow 0.18s ease,
    background 0.18s ease;
}

.button:hover {
  background: linear-gradient(135deg, #2563eb, #4f46e5);
```

```
  box-shadow:
    0 16px 40px rgba(37, 99, 235, 0.8),
    0 0 0 1px rgba(15, 23, 42, 0.95);
  transform: translateY(-1px);
}

.button:active {
  transform: translateY(0);
  box-shadow:
    0 8px 20px rgba(15, 23, 42, 0.95),
    0 0 0 1px rgba(15, 23, 42, 1);
}

.hint {
  text-align: center;
  margin-top: 0.75rem;
  font-size: 0.8rem;
  color: #6b7280;
}

@media (max-width: 480px) {
  .card {
    padding: 1.5rem 1.25rem;
    border-radius: 1rem;
  }

  .title {
    font-size: 1.35rem;
  }
}
```

4. Створити відповідні enum для роутів по проекту та intarfaces:

src/enums/routes.tsx

```
export enum Routes {
  HOME = "/",
  LOGIN = "/login",
  REGISTRATION = "/registration",
  DASHBOARD = "/dashboard",
}
```

5. Створити інтерфейси типізувавши відповіді та запити на сервер:

src/interfaces/forms.tsx

```
import type { IUserInfo } from "./users";

export interface IFormAuth {
  username: string;
  password: string;
}

export type LoginResponse =
  | { data?: IUserInfo }
  | { error: string; details?: number }
  | undefined
  | null;
export type LoginResult =
  | {
    status: number;
    data: IUserInfo;
  }
  | {
    status: number;
    error: string;
  };
```

```
export interface RegisterPayload {
  username: string;
  password: string;
}

export interface RegistrationSuccessResponse {
  message: string;
}

export type RegisterResult =
  | { data: RegistrationSuccessResponse }
  | { error: string; status?: number };
```

src/interfaces/users.tsx

```
export interface IUserInfo {
  id: number;
  username: string;
}
```

6. Реалізувати запити для логіну та реєстрації користувача, та перевірки серверу на health_check.

src/api/healthCheck.tsx

```
export const healthCheck = async (): Promise<string | boolean | undefined> => {
  const url = import.meta.env.VITE_API_URL;
  try {
    const res = await fetch(`${url}/api/health_check`, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
    });

    if (!res.ok) {
      const data = await res.json().catch(() => ({}));
      console.log(data);
      throw new Error((data as any).message || "Помилка запиту");
    }
  } catch (err) {
    return false;
  }
};
```

src/api/auth/login.tsx

```
import type {
  IFormAuth,
  LoginResponse,
  LoginResult,
} from "../../interfaces/forms";
import type { IUserInfo } from "../../interfaces/users";

export const login = async (payload: IFormAuth): Promise<LoginResult> => {
  const url = import.meta.env.VITE_API_URL;

  try {
    const res = await fetch(`${url}/api/auth/login`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(payload),
    });
    let body: { data?: IUserInfo } | null = null;

    try {
```

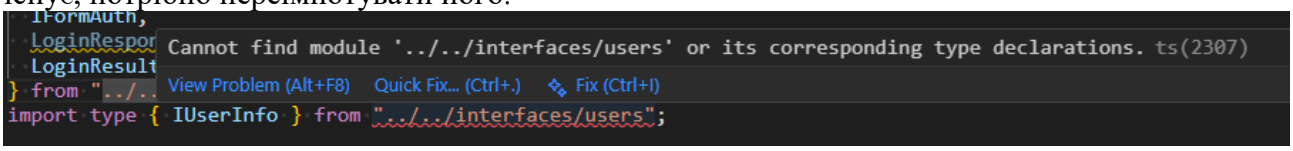
```
    body = (await res.json()) as { data?: IUserInfo };
   } catch {
    body = null;
   }

   if (res.status === 200 && body?.data) {
    return {
     status: res.status,
     data: body.data,
    };
   }

   return {
    status: res.status,
    error: "Something went wrong!",
   };
  } catch (e) {
   const message = e instanceof Error ? e.message : "Something went wrong!";

   return { error: message, status: 400 };
  }
};
```

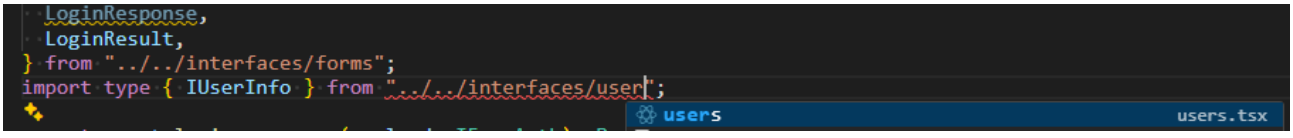
src/api/auth/registration.tsx

```
import type { RegisterPayload, RegisterResult, RegistrationSuccessResponse } from "../../interfaces/forms";

export const register = async (
  payload: RegisterPayload
): Promise<RegisterResult> => {
  const baseUrl = import.meta.env.VITE_API_URL;

  if (!baseUrl) {
   return {
    error: "VITE_API_URL не задано в .env",
   };
  }

  try {
   const res = await fetch(`${baseUrl}/api/auth/registration`, {
    method: "POST",
    headers: {
     "Content-Type": "application/json",
    },
    body: JSON.stringify(payload),
   });

   let json: unknown = null;
   try {
    json = await res.json();
   } catch {
    json = null;
   }

   if (res.status === 201 || res.status === 200) {
    const data = json as Partial<RegistrationSuccessResponse> | null;

    return {
     data: {
      message: data?.message || "Користувача успішно створено",
     },
    };
   }

   const errJson = json as { message?: string; detail?: string } | null;
   const backendMsg = errJson?.message || errJson?.detail;
```

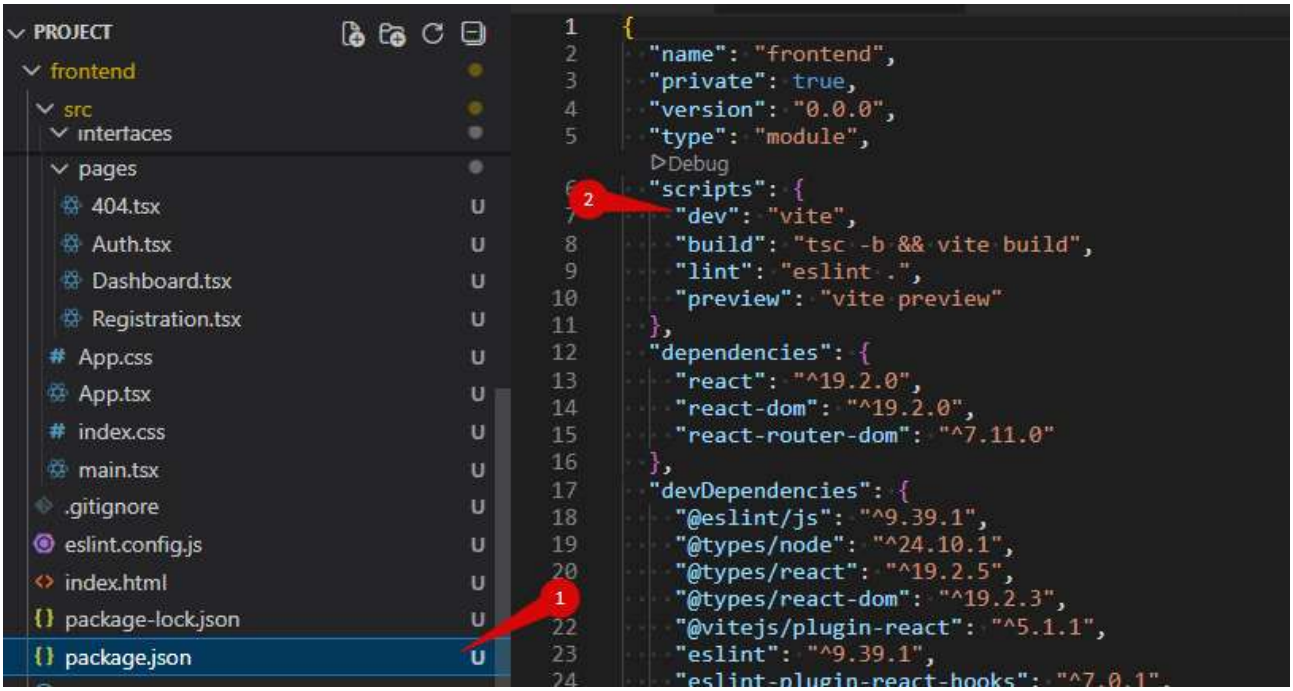
```
      if (res.status === 400) {
        return {
          status: res.status,
          error: backendMsg || "Невірні дані для реєстрації (400)",
        };
      }

      if (res.status === 409) {
        return {
          status: res.status,
          error: backendMsg || "Користувач з таким username вже існує (409)",
        };
      }

      if (res.status === 401 || res.status === 403) {
        return {
          status: res.status,
          error: backendMsg || "Немає доступу до операції реєстрації",
        };
      }

      if (res.status >= 500) {
        return {
          status: res.status,
          error: backendMsg || "Внутрішня помилка сервера при реєстрації",
        };
      }

      return {
        status: res.status,
        error:
          backendMsg ||
          `Неочікуваний статус від /api/auth/registration: HTTP ${res.status}`,
      };
    } catch (e) {
      const message =
        e instanceof Error ? e.message : "Невідома помилка при реєстрації";

      return {
        error: message,
      };
    }
  };
```

7. У випадку отримання помилки імпорту модуля «Не можна імпортувати модуль», але модуль існує, потрібно переімпотувати його:





8. Відкрити файл package.json та знайти команду запуска серверу в режимі розробки



Запустити застосунок за допомогою npm run dev.

9. Відкрити додатковий термінал та запустити серверну частину;

10. Протестувати зв'язок серверної та клієнтської частини(зареєструвати нового користувача, залогінитись, спробувати ввести невірні данні при логіні). При отримані такої помилки як на рисунку 7.1, увімкнути CORS на серверній частині додатку та перезавантажити сервер.

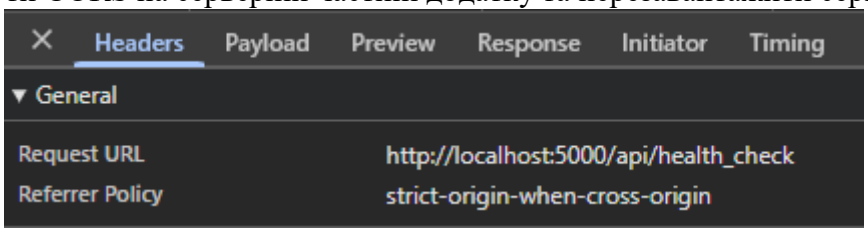


Рис.7.1

```python
def create_app() -> Flask:
    app = Flask(__name__)
    CORS(app)

    with app.app_context():

        init_db()

    app.register_blueprint(health_check, url_prefix="/api")
    app.register_blueprint(auth, url_prefix="/api/auth")

    return app
```

11. Після завершення роботи створити Pull Request на GitHub, зробити опис, що було виконано та дочекатись схвалення на об'єднання гілок. Надсилати на пошту звіт та посилання на PR.

7.4 Зміст звіту

1. Найменування і мета роботи;

2. Код по кожному пункту порядку виконання роботи;

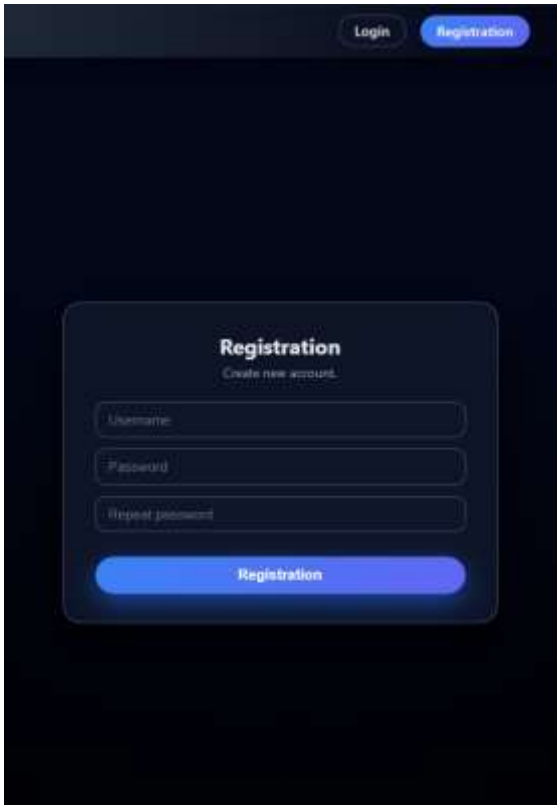
3. Результати роботи по кожному пункту виконання роботи;

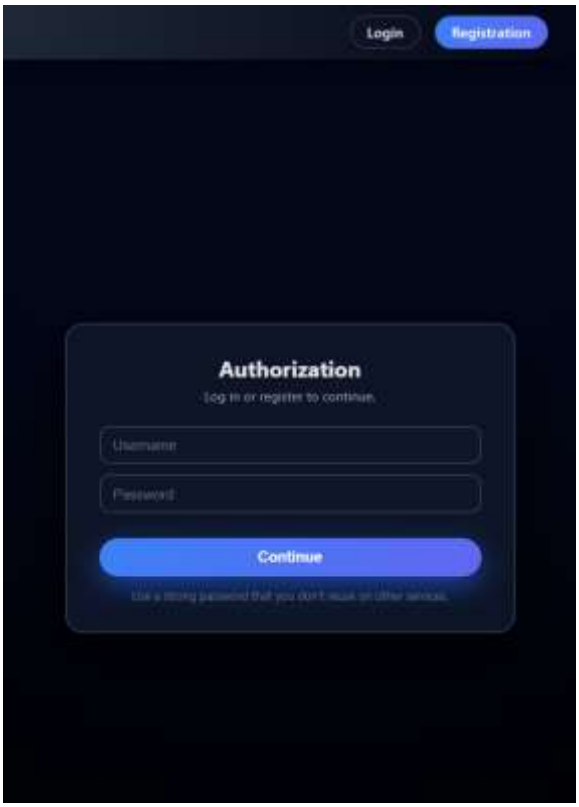
4. Висновки.

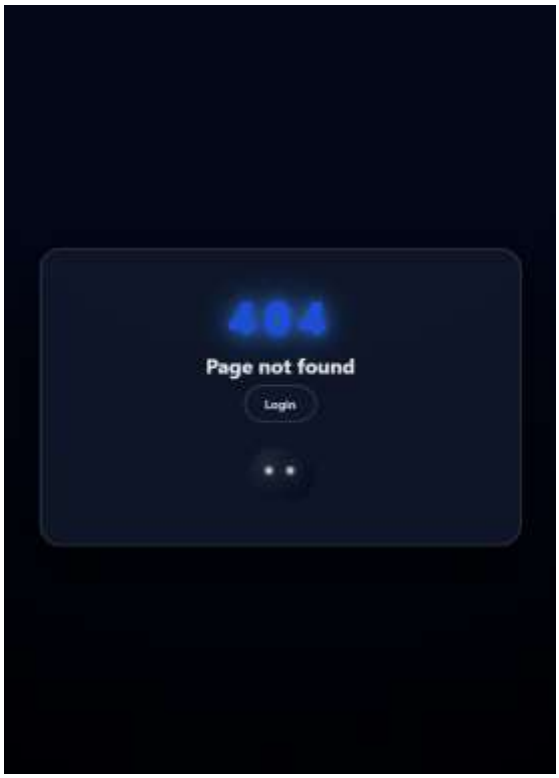
7.5 Контрольні запитання

1. Що таке fetch у JavaScript? Який тип значення він повертає?
2. Як за допомогою fetch реалізувати авторизацію, якщо сервер повертає токен доступу? Які кроки потрібно виконати на клієнті?
3. Чому небажано викликати fetch безпосередньо в тілі компонента (поза useEffect) у React?
4. Що таке CORS (Cross-Origin Resource Sharing) і як він впливає на роботу fetch у браузері?

Додаток 1






404
Page not found
Login


Dashboard
A quick overview of your account.
BETA
System status
Everything is running smoothly.
API online
Auth OK
Recent activity
Last login: just now
Profile updated: —
Active sessions: 1
What's next?
This area will later show real widgets, charts, statistics and latest events from your application.