

POLITECHNIKA ŚWIĘTOKRZYSKA

Wydział Elektrotechniki, Automatyki i Informatyki

Maksymilian Sowula, Paweł Marek

Numer grupy dziekańskiej: 1ID21A

Porównanie wybranych frameworków frontendowych – Next.js oraz Angular

Technologie Obiektowe - Projekt

1. Dokumentacja techniczna projektu

1.1. Część backendowa projektu

Backend tworzonych aplikacji został zrealizowany przy użyciu języka programowania Java oraz nowoczesnego frameworka Spring Boot, który znacząco ułatwia tworzenie skalowalnych i wydajnych aplikacji webowych. Spring Boot oferuje gotową strukturę projektu, wbudowane mechanizmy konfiguracyjne oraz liczne moduły ułatwiające integrację z różnymi usługami, takimi jak bazy danych, systemy bezpieczeństwa czy mechanizmy logowania.

W aplikacji wykorzystano również relacyjną bazę danych PostgreSQL, która służy do trwałego przechowywania danych użytkowników, takich jak dane logowania, profile, czy linki do przesłanych zdjęć. PostgreSQL to stabilny, wydajny system zarządzania bazą danych, wspierający zaawansowane operacje na danych oraz zapewniający zgodność z normami SQL.

Spring Boot został połączony z PostgreSQL za pomocą biblioteki Spring Data JPA, która upraszcza dostęp do danych dzięki wykorzystaniu mechanizmu mapowania obiektowo-relacyjnego (ORM). Pozwala to na operowanie na danych za pomocą klas Javy bez konieczności pisania zapytań SQL, co przyspiesza rozwój aplikacji i zwiększa jej czytelność.

Dodatkowo backend został wyposażony w warstwę REST API, dzięki której możliwa jest komunikacja z frontendem aplikacji – przysyłanie formularzy rejestracyjnych, logowanie użytkowników czy pobieranie danych o zdjęciach. Całość została zabezpieczona odpowiednimi mechanizmami autoryzacji i uwierzytelniania (przy użyciu JWT – JSON Web Tokens).

Aby uruchomić serwer backendowy należy wykonać następujące czynności:

- zainstalować oprogramowanie docker desktop ze strony: <https://www.docker.com/get-started/>,
- zainstalować oprogramowanie git ze strony: <https://git-scm.com/downloads>,

- sklonować projekt z repozytorium github za pomocą komendy: `git clone https://github.com/miluski/PhotoForum.git` <lokalizacja_sklonowania_repozytorium>,
- uruchomić konsolę, przejść do folderu, w którym znajduje się sklonowane repozytorium za pomocą komendy `cd`, a następnie uruchomić oprogramowanie docker desktop,
- w folderze z projektem utworzyć plik `.env` i wprowadzić do niego następującą zawartość:
 - `POSTGRES_DB=postgres`
 - `POSTGRES_PASSWORD=<twoje_hasło_do_bazy_danych>`
 - `POSTGRES_USER=<twoja_nazwa_użytkownika_do_bazy_danych>`
- przejść do lokalizacji `backend/src/main/resources` (w przypadku braku istnienia folderu `resources` utworzyć go), a następnie w nim utworzyć plik `application.properties` i wprowadzić do niego następującą zawartość:
 - `spring.application.name=backend`
 - `server.port=4443`
 - `server.ssl.key-store=/certs/keystore.p12`
 - `server.ssl.key-store-password=keystorePass`
 - `server.ssl.key-store-type=PKCS12`
 - `server.ssl.key-alias=tomcat`
 - `spring.servlet.multipart.max-file-size=50MB`
 - `spring.servlet.multipart.max-request-size=50MB`
 - `spring.datasource.url=jdbc:postgresql://photo-forum-postgres:5432/postgres`
 - `spring.datasource.username=<twoja_nazwa_użytkownika_do_bazy_danych>`
 - `spring.datasource.password=<twoje_hasło_do_bazy_danych>`
 - `spring.jpa.hibernate.ddl-auto=update`
 - `spring.jpa.show-sql=true`
 - `spring.jpa.properties.hibernate.format_sql=true`
 - `jwt.secret=<twój_wygenerowany_sekret_do_tokenów_jwt_w_formacie_sha256>`
 - `jwt.expiration=60000`

- `jwt.refresh.expiration=300000`
- `photo.dir=/media`
- w konsoli uruchomić komendę: `docker compose -f compose.backend.yaml --up --force-recreate`.

W aplikacji backendowej utworzono następujące punkty końcowe:

- `/api/v1/auth/is-authorized` – punkt końcowy zwracający kod 200, jeśli token autoryzujący użytkownika jest ważny, a 401 jeśli nie. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę GET,
- `/api/v1/auth/login` – punkt końcowy przyjmujący jako ciało żądania obiekt JSON zawierający login i hasło użytkownika, jako kod odpowiedzi zwraca kod 200 wraz z imieniem zalogowanego użytkownika w postaci napisu, jeśli dane są prawidłowe, a kod 403 jeśli dane uwierzytelniające są nieprawidłowe. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę POST,
- `/api/v1/auth/register` – punkt końcowy przyjmujący jako ciało żądania obiekt JSON zawierający login, hasło, imię, nazwisko i ścieżkę do avataru rejestrowanego użytkownika. Zwraca kod 200 jeśli żądanie przebiegnie pomyślnie, a kod 403 jeśli nie. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę POST,
- `/api/v1/auth/refresh-tokens` – punkt końcowy, który służy do wygenerowania nowej pary tokenów uwierzytelniających. Zwraca nowe tokeny wraz z kodem odpowiedzi 200, jeśli został przekazany `REFRESH_TOKEN` w ciasteczkach żądania i był ważny, a kod 403 w przypadku braku ważnego tokenu odświeżającego. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę POST,
- `/api/v1/auth/logout` – punkt końcowy służący do usunięcia z ciasteczek tokenów: `ACCESS_TOKEN` oraz `REFRESH_TOKEN` celem unieważnienia autoryzacji użytkownika i w efekcie jego wylogowania. Zwraca kod 200 wraz z wygaśniętymi tokenami w przypadku obecności ważnego `REFRESH_TOKEN` lub `ACCESS_TOKEN` w ciasteczkach żądania, a kod 403 w przypadku braku tych ciasteczek. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę POST,

- `/api/v1/photos/new` – punkt końcowy, który jako ciało żądania przyjmuje obiekt JSON z polem `path`. W przypadku poprawnego utworzenia zdjęcia w folderze `media` po stronie serwera zwraca kod 200, kod 404 jeśli nie znaleziono obiektu użytkownika, do którego można by przypisać autorstwo wstawianego zdjęcia oraz kod 500 w przypadku wystąpienia nieoczekiwanego błędu. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę POST,
- `/api/v1/photos/posts` – punkt końcowy dostępny dla wszystkich użytkowników zwracający wszystkie posty zdjęć znajdujące się w bazie danych w postaci tablicy zawierającej obiekty JSON z polami: `id`, `path`, `userDto`, `likesCount`, `commentDtos` z kodem 200 jeśli serwer jest dostępny i w bazie znajduje się chociaż 1 rekord, kod 404 w przypadku braku rekordów w bazie lub kod 500 w przypadku wystąpienia nieoczekiwanego błędu. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę GET,
- `/api/v1/photos/public/<nazwa_zdjęcia.rozszerzenie_zdjęcia>` - punkt końcowy dla wszystkich użytkowników zwracający zrenderowane po stronie serwera zdjęcie o podanej nazwie i rozszerzeniu jeśli istnieje z kodem odpowiedzi 200. Jeśli zdjęcie nie istnieje zostanie zwrócony kod 404, a w przypadku niespodziewanego błędu zostanie zwrócony kod 500. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę GET,
- `/api/v1/photos/<id_zdjęcia>/add-comment` – punkt końcowy, który służy do dodania komentarza do zdjęcia wskazanego w punkcie. Jako ciało żądania przyjmuje napis, który jest komentarzem do zdjęcia. Zwraca kod 200 w przypadku poprawnego dodania komentarza (wymogiem jest poprawny komentarz oraz ważny `ACCESS_TOKEN`), kod 404 w przypadku braku znalezienia wskazanego zdjęcia i posiadanej autoryzacji, kod 401 w przypadku braku autoryzacji do dodania komentarza oraz kod 500 w przypadku wystąpienia nieoczekiwanego błędu. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę POST,
- `/api/v1/photos/<id_zdjęcia>/add-to-favourites` – punkt końcowy, który służy do dodania do ulubionych zdjęcia wskazanego w punkcie. Zwraca kod 200 w przypadku ważnego `ACCESS_TOKEN` oraz istnienia zdjęcia, kod 404 w przypadku braku znalezienia wskazanego zdjęcia i posiadanej autoryzacji, kod 401 w przypadku braku autoryzacji oraz kod 500 w przypadku wystąpienia nieoczekiwanego błędu. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę POST,

- `/api/v1/photos/<id_zdjęcia>/remove-from-favourites` – punkt końcowy, który służy do usunięcia z ulubionych zdjęcia wskazanego w punkcie. Zwraca kod 200 w przypadku ważnego ACCESS_TOKEN oraz istnienia zdjęcia, kod 404 w przypadku braku znalezienia wskazanego zdjęcia i posiadanej autoryzacji, kod 401 w przypadku braku autoryzacji oraz kod 500 w przypadku wystąpienia nieoczekiwanego błędu. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę DELETE,
- `/api/v1/photos/public/id/<id_zdjęcia>` - punkt końcowy dla wszystkich użytkowników zwracający obiekt postu zdjęcia o podanym id jeśli istnieje z kodem odpowiedzi 200. Jeśli zdjęcie nie istnieje zostanie zwrócony kod 404, a w przypadku niespodziewanego błędu zostanie zwrócony kod 500. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę GET,
- `/api/v1/users/edit` – punkt końcowy służący do edycji wybranej, pojedynczej danej lub wszystkich danych użytkownika. Jako ciało żądania przyjmuje obiekt, który musi zawierać przynajmniej jedną ze wskazanych danych: login, password, name, surname, avatarPath. Zwraca kod 200 jeśli edycja przebiegnie pomyślnie, odświeżone tokeny jeśli użytkownik zmieni swój login, kod 404 jeśli użytkownik do edycji nie zostanie znaleziony lub kod 500 jeśli nastąpi nieoczekiwany błąd. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę PATCH,
- `/api/v1/users/get-favourite-photos` - punkt końcowy służący do pobrania ulubionych zdjęć zalogowanego użytkownika. Jako odpowiedź zwraca wszystkie polubione posty zdjęć przez użytkownika znajdujące się w bazie danych w postaci tablicy zawierającej obiekty JSON z polami: id, path, userDto, likesCount, commentDtos z kodem 200 jeśli serwer jest dostępny, kod 404 w przypadku braku znalezienia ulubionych zdjęć użytkownika lub kod 500 w przypadku wystąpienia nieoczekiwanego błędu. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę GET,
- `/api/v1/users/user-details` – punkt końcowy służący do pobrania obiektu JSON szczegółów zalogowanego użytkownika. Zwraca kod 200 z obiektem zawierającym: name, surname, login, password oraz avatarPath w przypadku, gdy użytkownik jest zautoryzowany, kod 401 jeśli nie jest zautoryzowany oraz kod 500 w przypadku wystąpieniu błędu serwera. Aby skomunikować się z tym punktem, należy w nagłówku żądania wystosować metodę GET.

1.2. Część frontendowa projektu – Next.js

Aby uruchomić część frontendową projektu we frameworku Next.js należy wykonać następujące czynności:

- pobrać node.js z oficjalnej strony: <https://nodejs.org/en/download>,
- otworzyć terminal, przejść do katalogu zawierającego frontend oparty na Next.js za pomocą komendy `cd 'frontend – next.js'`,
- zainstalować niezbędne zależności projektu określone w package.json za pomocą komendy: `npm install`,
- uruchomić lokalny serwer deweloperski Next.js, dzięki któremu projekt będzie dostępny w przeglądarce (domyślnie pod adresem `http://localhost:3000`) za pomocą komendy: `npm run dev`.

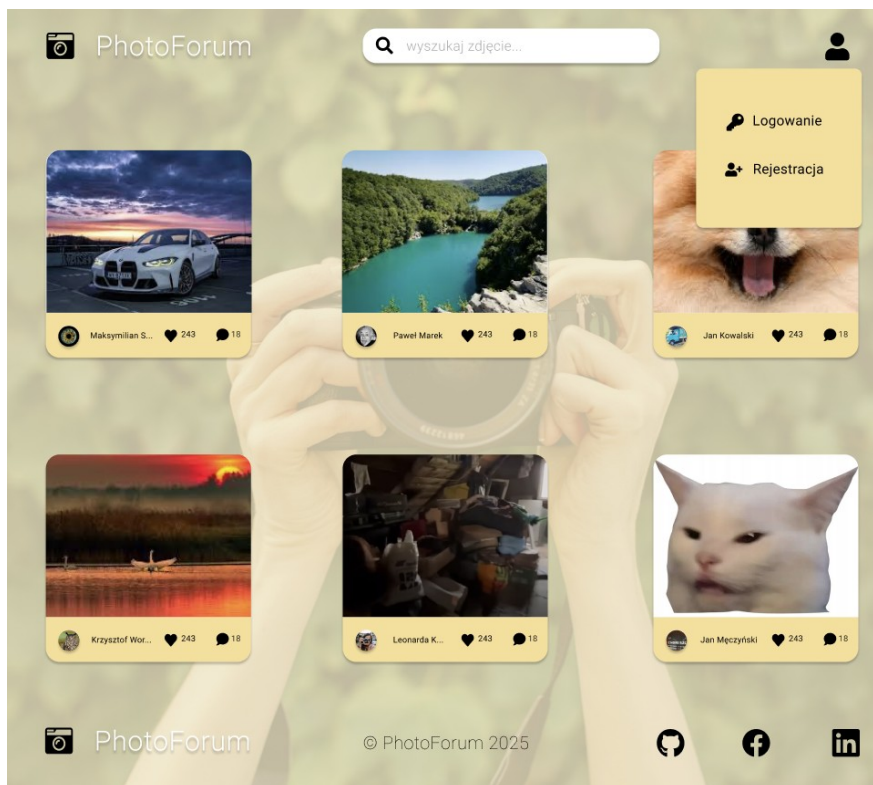
1.3. Część frontendowa projektu – Angular

Aby uruchomić część frontendową projektu we frameworku Angular należy wykonać następujące czynności:

- pobrać node.js z oficjalnej strony: <https://nodejs.org/en/download>,
- otworzyć terminal, przejść do katalogu zawierającego frontend oparty na Angular za pomocą komendy `cd 'frontend – angular'`,
- zainstalować angular command line interface za pomocą komendy: `npm install -g @angular/cli`,
- zainstalować niezbędne zależności projektu określone w package.json za pomocą komendy: `npm install`,
- uruchomić lokalny serwer deweloperski Angular, dzięki któremu projekt będzie dostępny w przeglądarce (domyślnie pod adresem `https://localhost:4200`) za pomocą komendy: `ng serve`.

2. Projektowanie szablonów graficznych aplikacji webowej

2.1. Strona główna – dla niezalogowanego użytkownika



Rysunek 1. Widok strony głównej aplikacji webowej.

Powyższy rysunek przedstawia stronę główną platformy służącej do dzielenia się zdjęciami. Na środku ekranu znajduje się galeria zdjęć użytkowników, ułożona w układzie siatki z sześcioma kafelkami. Każdy kafelek zawiera zdjęcie, nazwę użytkownika (częściowo skróconą, np. z inicjałem nazwiska), a także ikony oznaczające liczbę polubień (symbol serca) oraz liczbę komentarzy (dymek).

W prawym górnym rogu znajduje się ikona profilu, po kliknięciu której rozwija się menu z dwiema opcjami: „Logowanie” oraz „Rejestracja”, w przypadku, gdy użytkownik nie jest zalogowany. Pośrodku górnej części interfejsu widnieje pasek wyszukiwania z ikoną lupy i tekstem pomocniczym „wyszukaj zdjęcie...”, który pozwala użytkownikowi na filtrowanie zdjęć.

Całość utrzymana jest w estetyce fotograficznej – tło stanowi rozmyte zdjęcie rąk trzymających aparat fotograficzny, co wzmacnia charakter platformy jako miejsca skupionego wokół pasji do zdjęć.

2.2. Formularz rejestracji



The screenshot shows a mobile application interface for 'PhotoForum'. At the top, there is a camera icon and the text 'PhotoForum' on the left, and a user profile icon on the right. The background is a blurred image of a person's hands holding a camera. In the center, a white rounded rectangle contains the title 'Rejestracja' in bold black text. Below the title are four text input fields, each with a label above it: 'Imię', 'Nazwisko', 'Login', and 'Hasło'. At the bottom of this rectangle is a yellow button with the text 'Zarejestruj się'. At the bottom of the screen, there is a footer with a camera icon and 'PhotoForum' on the left, '© PhotoForum 2025' in the center, and three social media icons (Twitter, Facebook, LinkedIn) on the right.

Rysunek 2. Widok podstrony zawierającej formularz rejestracji.

Widok aplikacji przedstawia formularz rejestracji użytkownika na platformie. W centralnej części znajduje się okno z napisem "Rejestracja" u góry. Poniżej umieszczono cztery pola tekstowe z etykietami: "Imię", "Nazwisko", "Login" oraz "Hasło", umożliwiające wprowadzenie danych przez użytkownika. Na dole formularza znajduje się przycisk z napisem "Zarejestruj się", który służy do zatwierdzenia rejestracji. Układ jest prosty i przejrzysty, co ułatwia proces zakładania konta.

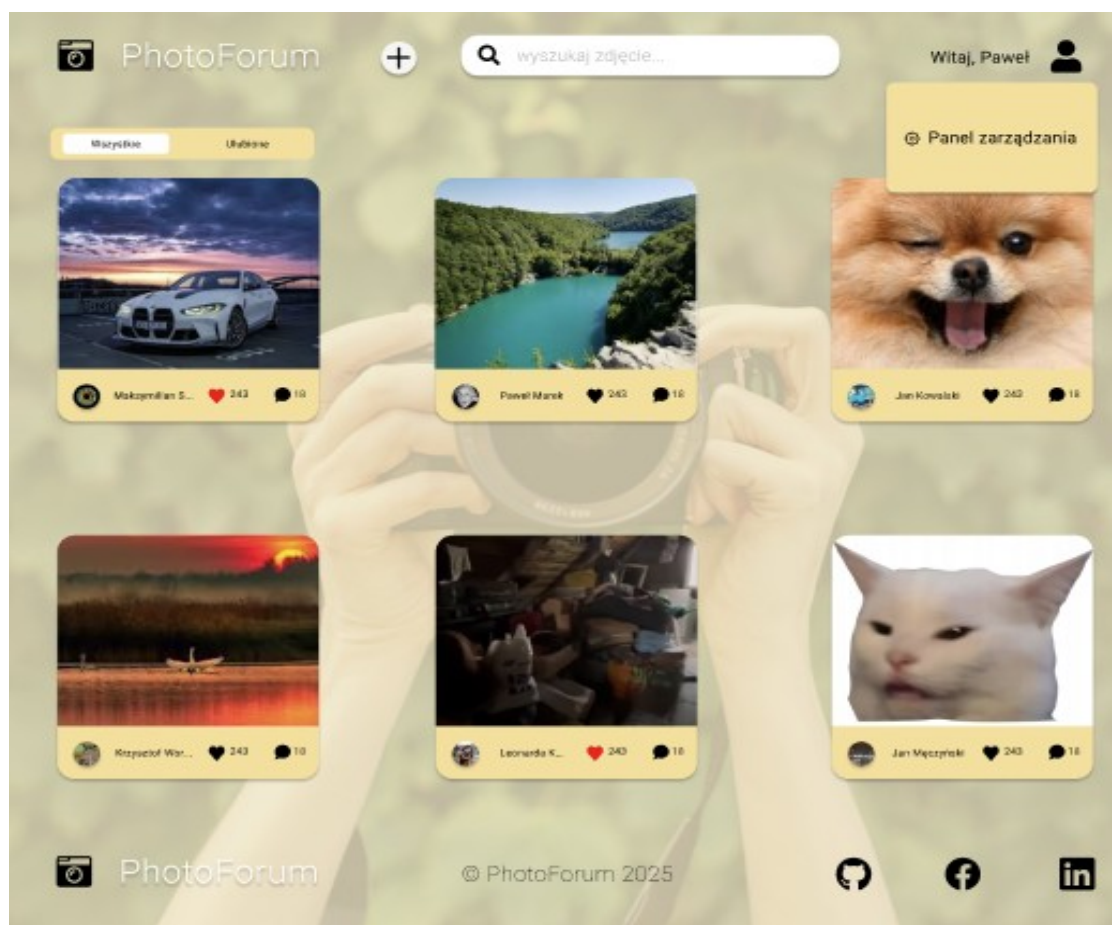
2.3. Formularz logowania

The image shows a mobile application interface for 'PhotoForum'. At the top, there is a camera icon and the text 'PhotoForum' on the left, and a user profile icon on the right. The background is a blurred image of hands holding a camera. In the center, a white rounded rectangle contains the title 'Logowanie' in bold black text. Below the title are two text input fields: the first is labeled 'Login' and the second is labeled 'Hasło'. Below these fields is a yellow button with the text 'Zaloguj się'. At the bottom of the screen, there is a footer with a camera icon and 'PhotoForum' on the left, '© PhotoForum 2025' in the center, and three social media icons (Twitter, Facebook, LinkedIn) on the right.

Rysunek 3. Widok podstrony zawierającej formularz logowania.

Powyższy rysunek przedstawia formularz logowania użytkownika. W centralnej części znajduje się okno z napisem "Logowanie" u góry. Poniżej umieszczono dwa pola tekstowe z etykietami: "Login" oraz "Hasło", które umożliwiają użytkownikowi wprowadzenie danych logowania. Na dole formularza znajduje się przycisk z napisem "Zaloguj się", służący do zatwierdzenia procesu logowania.

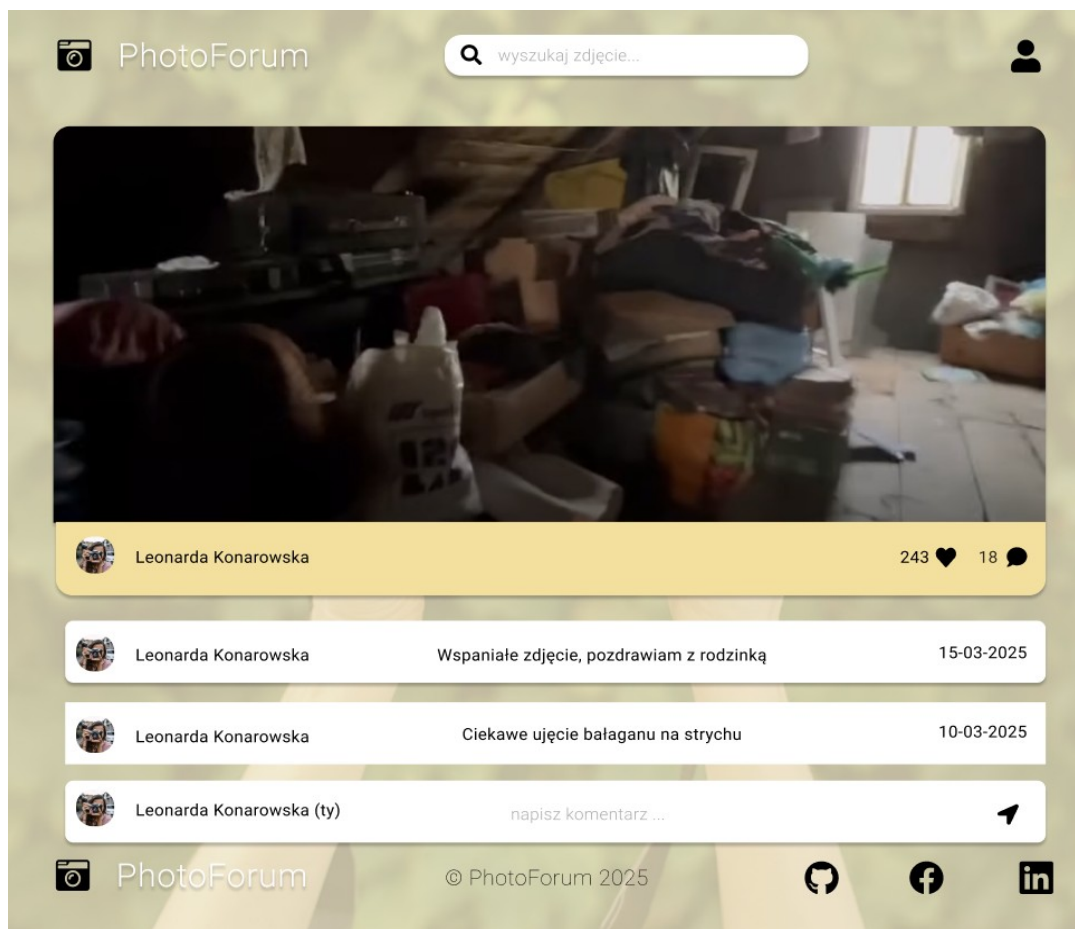
2.4. Strona główna – dla zalogowanego użytkownika



Rysunek 4. Widok podstrony zawierającej formularz logowania.

Po zalogowaniu, w prawym górnym rogu znajduje się ikona "Panel zarządzania", która pozwala na szybkie przejście do sekcji administracyjnej. W tym panelu użytkownik może edytować swoje dane. Dodatkowo, w lewej części ekranu dostępna jest zakładka "Ulubione", umożliwiająca przełączenie się na listę ulubionych zdjęć.

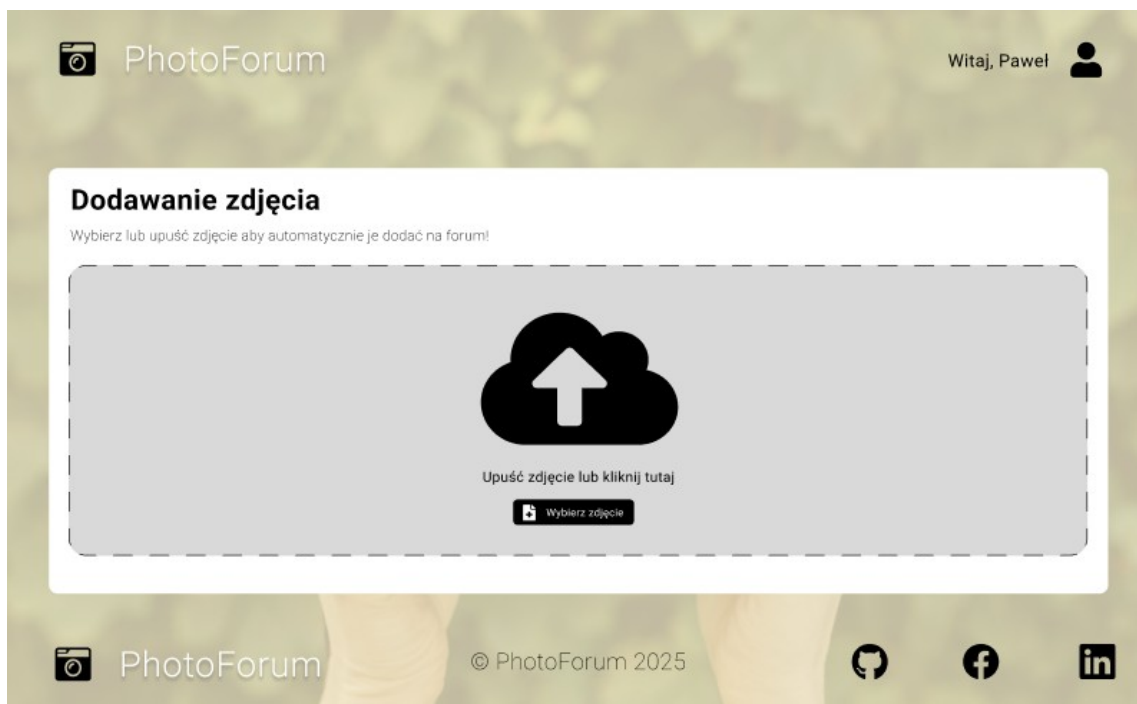
2.5. Widok szczegółów zdjęcia



Rysunek 5. Widok podstrony zawierającej szczegóły zdjęcia.

Powyższy rysunek przedstawia widok pojedynczego posta na platformie ze zdjęciami. W centralnej części podstrony znajduje się główne zdjęcie opublikowane przez użytkownika. Pod zdjęciem umieszczono informacje o autorze posta oraz statystyki interakcji, takie jak liczba polubień i komentarzy. Poniżej zdjęcia znajduje się sekcja komentarzy, wyświetlana w formie listy z datami publikacji, wraz z możliwością dodawania nowego komentarza.

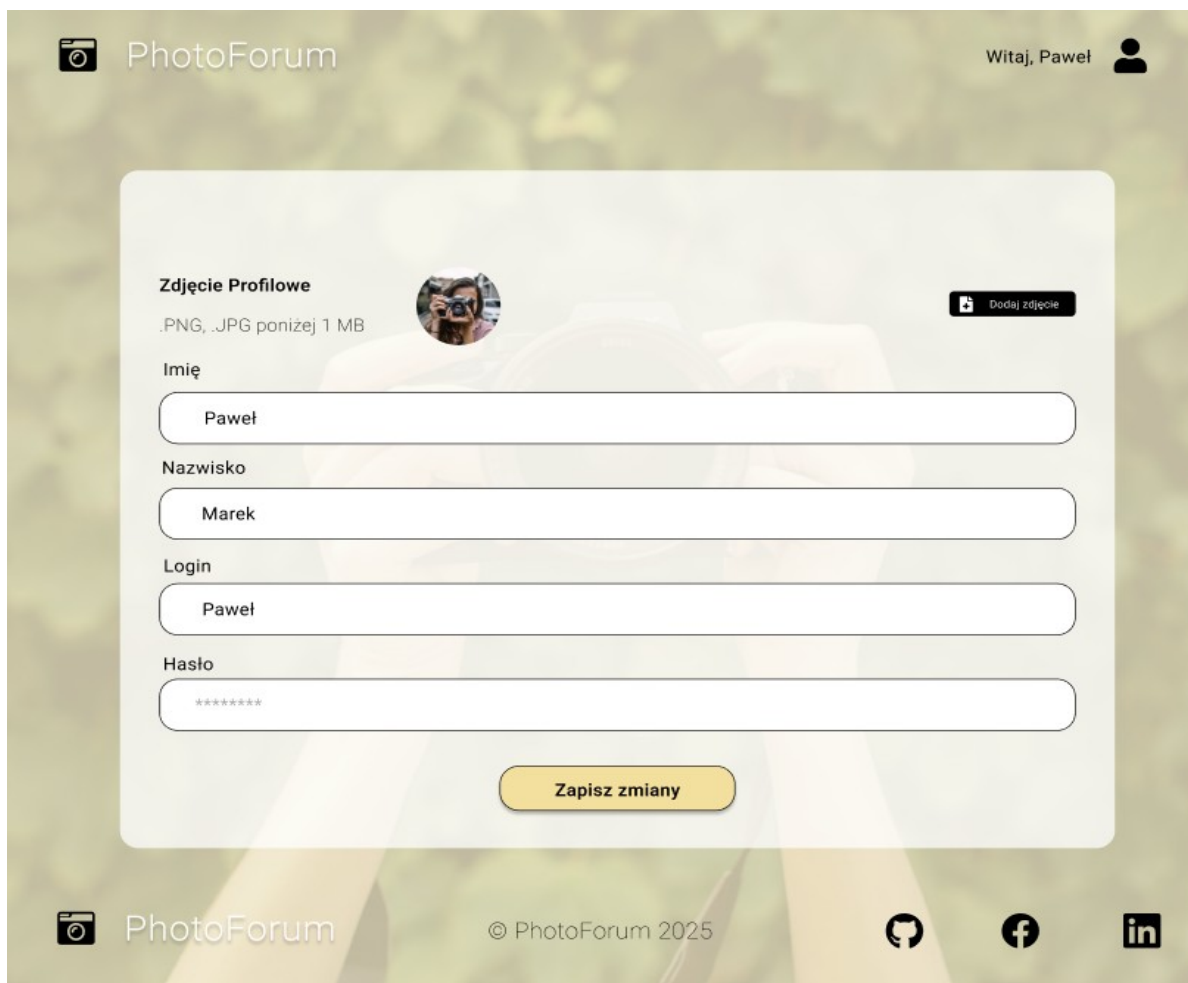
2.6 Widok podstrony dodawania nowego zdjęcia



Rysunek 6. Widok podstrony przeznaczonej do dodawania nowego zdjęcia.

Powyższy rysunek przedstawia podstronę dodawania zdjęcia na platformie. W centralnej części znajduje się okno z tytułem oraz opisem u góry. Poniżej widoczny jest prostokątny obszar z ikoną chmury i strzałką w górę, służący do przesyłania zdjęcia. Na dole tego obszaru znajduje się przycisk umożliwiający wybranie pliku z urządzenia.

2.7 Widok podstrony edycji szczegółów dotyczących konta użytkownika



The screenshot shows the 'PhotoForum' user profile editing interface. At the top left is the 'PhotoForum' logo. At the top right, it says 'Witaj, Paweł' next to a user icon. The main content area is a light gray card with a background image of a person holding a camera. Inside the card, there's a section for the profile picture labeled 'Zdjęcie Profilowe' with a circular preview and a 'Dodaj zdjęcie' button. Below this are input fields for 'Imię' (Paweł), 'Nazwisko' (Marek), 'Login' (Paweł), and 'Hasło' (masked with asterisks). A yellow 'Zapisz zmiany' button is at the bottom of the card. The footer contains the 'PhotoForum' logo, copyright '© PhotoForum 2025', and social media icons for GitHub, Facebook, and LinkedIn.

PhotoForum

Witaj, Paweł

Zdjęcie Profilowe

.PNG, .JPG poniżej 1 MB

Dodaj zdjęcie

Imię

Paweł

Nazwisko

Marek

Login

Paweł

Hasło

Zapisz zmiany

PhotoForum

© PhotoForum 2025

GitHub Facebook LinkedIn

Rysunek 6. Widok podstrony przeznaczonej do dodawania nowego zdjęcia.

Kolejnym widokiem aplikacji jest podstrona zawierająca formularz umożliwiający edycję szczegółów konta, w tym zmianę imienia, nazwiska, loginu oraz hasła. Ponadto istnieje możliwość zaktualizowania zdjęcia profilowego.

3. Porównanie tworzenia zestawu ścieżek aplikacji

3.1. Framework Next.js

Routing w aplikacji Next.js został zaprojektowany w oparciu o strukturę plików w katalogu app. Next.js automatycznie generuje ścieżki na podstawie hierarchii plików i folderów w tym katalogu, co pozwala na intuicyjne i szybkie tworzenie routingu bez potrzeby ręcznego definiowania tras.

Poniżej przedstawiono strukturę katalogu app oraz odpowiadające jej ścieżki w aplikacji:

- app/page.tsx

Odpowiada za stronę główną aplikacji, dostępna pod ścieżką /.

- app/account-settings/page.tsx

Odpowiada za stronę ustawień konta użytkownika, dostępna pod ścieżką /account-settings.

- app/add-photo/page.tsx

Odpowiada za stronę dodawania zdjęcia, dostępna pod ścieżką /add-photo.

- app/login/page.tsx

Odpowiada za stronę logowania, dostępna pod ścieżką /login.

- app/register/page.tsx

Odpowiada za stronę rejestracji, dostępna pod ścieżką /register.

- app/photo/[photoId]/page.tsx

Odpowiada za dynamiczną stronę szczegółów zdjęcia, dostępna pod ścieżką /photo/:photoId, gdzie :photoId jest dynamicznym parametrem.

- app/unauthorized.tsx

Odpowiada za stronę błędu autoryzacji, dostępna pod ścieżką /unauthorized.

- app/not-found.tsx

Odpowiada za stronę błędu 404, wyświetlaną w przypadku braku dopasowania ścieżki.

Next.js wykorzystuje strukturę plików do automatycznego generowania tras:

- Pliki `page.tsx` w katalogach odpowiadają statycznym ścieżkom.
- Foldery z nazwami w nawiasach kwadratowych, np. `[photoId]`, definiują dynamiczne segmenty ścieżek.
- Pliki takie jak `not-found.tsx` czy `unauthorized.tsx` są używane do obsługi błędów i specjalnych przypadków

Routing oparty na strukturze plików pozwala na szybkie prototypowanie i rozwój aplikacji, jednocześnie zapewniając czytelność i łatwość utrzymania kodu.

3.2. Framework Angular

Framework Angular przy tworzeniu nowego projektu automatycznie tworzy plik `app.routes.ts`, który zawiera podstawowo pustą tablicę o typie `Routes`, który definiuje, iż tablica jest typu `Route`, który narzuca na obiekty JSON konieczność posiadania pól `path` – ścieżka, która definiuje odwołanie do głównego linku strony oraz `component`, które definiuje komponent aplikacji renderowany przy wejściu na podaną ścieżkę przez użytkownika. W aplikacji zdefiniowano ścieżki: `login`, `register`, `dashboard`, `add-photo`, `single-photo`, `unauthorized` oraz specjalne ścieżki: `**` oznaczającą wszystkie niezdefiniowane ścieżki oraz `' '` oznaczającą podstawową ścieżkę aplikacji. Na poniższym listingu przedstawiono wygląd pliku `app.routes.ts`.


```

import { Routes } from '@angular/router';
import { AddPhotoComponent } from './add-photo/add-photo.component';
import { AuthGuard } from './auth.guard';
import { DashboardComponent } from './dashboard/dashboard.component';
import { HomeComponent } from './home/home.component';
import { LoginComponent } from './login/login.component';
import { NotFoundComponent } from './not-found/not-found.component';
import { RegisterComponent } from './register/register.component';
import { SinglePhotoComponent } from './single-photo/single-photo.component';
import { UnauthorizedComponent } from './unauthorized/unauthorized.component';

export const routes: Routes = [
  { path: '', component: HomeComponent, },
  { path: 'login', component: LoginComponent, },
  { path: 'register', component: RegisterComponent, },
  { path: 'dashboard', component: AddPhotoComponent, canActivate: [AuthGuard], },
  { path: 'add-photo', component: AddPhotoComponent, canActivate: [AuthGuard], },
  { path: 'single-photo', component: SinglePhotoComponent, },
  { path: 'unauthorized', component: UnauthorizedComponent, },
  { path: '**', component: NotFoundComponent, },
];

```

Listing 1: Plik app.routes.ts.

W aplikacjach Angular plik app.component.html pełni rolę głównego szablonu, od którego zaczyna się renderowanie całej aplikacji. To właśnie w tym pliku umieszczany jest znacznik `<router-outlet>`, który działa jako dynamiczne miejsce osadzania komponentów. Angular automatycznie wstrzykuje do niego odpowiednie komponenty na podstawie skonfigurowanych ścieżek (routing). Dzięki temu użytkownik, poruszając się po aplikacji, widzi zmieniające się widoki bez konieczności przeładowywania strony, co jest kluczowe dla działania aplikacji typu SPA (Single Page Application).

4. Porównanie skonfigurowania obsługi żądań sieciowych

4.1. Framework Next.js

Axios jest używany jako główna biblioteka do obsługi żądań HTTP. W projekcie Next.js skonfigurowano instancję Axios z podstawowym adresem URL oraz obsługą ciasteczek.

```
export const axiosInstance = axios.create({
  baseURL: "https://78.88.231.247:4443/api/v1",
  withCredentials: true,
  timeout: 10000,
});
```

Listing 2: Utworzenie instancji axios.

React Query jest używany do zarządzania stanem danych pochodzących z API. W projekcie skonfigurowano hooki do obsługi zapytań, takich jak pobieranie listy zdjęć.

```
export const useGetAllPhotosQuery = () => {
  const getUserDetails = async (): Promise<Photo[]> => {
    try {
      const response = await axiosInstance.get("/photos/posts");

      return response.data;
    } catch (error) {
      throw error;
    }
  };

  return useQuery({
    queryKey: ["all-photos"],
    queryFn: getUserDetails,
  });
};
```

Listing 3: Utworzenie custom hooka zawierającego logikę związaną z zapytaniem sieciowym, z wykorzystaniem react-query.

Utworzone hooki można następnie integrować w poszczególnych komponentach, gdzie potrzebujemy określonych danych pochodzących z API.

```
export default function Home() {  
  const { data: allPhotos } = useGetAllPhotosQuery();
```

Listing 4: Wykorzystanie utworzonego custom hooka w wybranym komponencie.

Dzięki tej konfiguracji aplikacja efektywnie obsługuje żądania sieciowe i zarządza stanem danych.

4.2. Framework Angular

Angular do obsługi żądań sieciowych zapewnia pakiet `@angular/common/http`, który zapewnia klasę `HttpClient` do obsługi żądań sieciowych. Aby móc ją wstrzykiwać do własnych klas czy serwisów należy w konfiguracji aplikacji dodać linijkę dotyczącą dostarczania tej klasy do konfiguracji aplikacji za pomocą providera. Przykładowe użycie klasy `httpClient` przedstawione jest na listingu 2.

```
public handleIsAuthorizedRequest(): Observable<never> {  
  const request: Observable<HttpResponse<Object>> = this.httpClient.get(  
    `${environment.apiUrl}/auth/is-authorized`,  
    { observe: 'response' }  
  );  
  request.subscribe({  
    next: (response: HttpResponse<Object>) =>  
      this._isAuthorizedUserSubject.next(response.status === 200),  
    error: () => this._isAuthorizedUserSubject.next(false),  
  });  
  return NEVER;  
}
```

Listing 5: Przykład użycia klasy `httpClient` do obsługi żądania sieciowego.

5. Porównanie zapewnienia możliwości stylowania i responsywności

5.1. Framework Next.js

W trakcie realizacji etapu dotyczącego stylowania i responsywności, zdecydowano się na wykorzystanie biblioteki Tailwind CSS jako głównego narzędzia do projektowania interfejsu użytkownika. Już na etapie tworzenia projektu Next.js za pomocą CLI wybrano wstępną konfigurację Tailwind CSS, co pozwoliło na szybkie rozpoczęcie pracy nad stylowaniem aplikacji. Tailwind CSS, dzięki swojej funkcjonalności opierającej się na klasach narzędziowych (utility-first), umożliwił efektywne i elastyczne projektowanie komponentów oraz zapewnienie spójności wizualnej w całej aplikacji.

Pierwszym krokiem było zdefiniowanie globalnych stylów, takich jak kolory, typografia oraz układ. W pliku konfiguracyjnym `tailwind.config.js` zdefiniowano niestandardową paletę kolorów, która odzwierciedlała branding aplikacji, oraz dostosowano ustawienia typografii, aby zapewnić czytelność i estetykę na różnych urządzeniach. Dzięki temu możliwe było łatwe stosowanie spójnych stylów w całym projekcie bez konieczności powtarzania kodu.

Następnie przystąpiono do projektowania i stylowania kluczowych komponentów aplikacji, takich jak nawigacja, formularze, dashboard oraz widoki zdjęć. Każdy z komponentów został zaprojektowany z myślą o responsywności, co osiągnięto dzięki wykorzystaniu predefiniowanych klas Tailwind CSS, takich jak `sm:`, `md:`, `lg:`, które umożliwiają stosowanie różnych stylów w zależności od rozdzielczości ekranu. Na przykład, w komponencie Header zastosowano klasy takie jak `hidden md:flex`, aby ukrywać lub wyświetlać elementy w zależności od rozmiaru ekranu.

W przypadku layoutu strony głównej, zastosowano siatkę opartą na klasach `grid` i `gap-4`, co pozwoliło na dynamiczne rozmieszczanie zdjęć w zależności od dostępnej przestrzeni. Dodatkowo, komponenty takie jak karty zdjęć zostały ostylowane z użyciem klas `rounded-lg`, `shadow-md`, czy `hover:shadow-lg`, co zapewniło nowoczesny i estetyczny wygląd.

Responsywność aplikacji została zapewniona poprzez testowanie na różnych urządzeniach, takich jak komputery stacjonarne, tablety i telefony komórkowe. Wykorzystano narzędzia deweloperskie przeglądarek, aby symulować różne rozdzielczości ekranu i upewnić się, że interfejs

użytkownika działa poprawnie w każdej sytuacji. W razie potrzeby wprowadzano drobne poprawki, takie jak dostosowanie rozmiarów czcionek czy marginesów, aby zapewnić optymalne doświadczenie użytkownika.

Podsumowując, etap stylowania i responsywności został zrealizowany z wykorzystaniem Tailwind CSS, co pozwoliło na szybkie i efektywne projektowanie interfejsu użytkownika. Dzięki zastosowaniu klas narzędziowych oraz predefiniowanych stylów, aplikacja jest estetyczna, spójna wizualnie i dostosowana do różnych urządzeń.

5.2. Framework Angular

Przy tworzeniu nowego projektu Angular do możliwości użytkownika udostępnia on następujące opcje stylowania:

- CSS – klasyczna opcja stylowania za pomocą kaskadowych arkuszy stylów,
- SCSS – preprocesor CSS z dodatkowymi funkcjami (zmienne, zagnieżdżanie, mixiny),
- SASS – alternatywna składnia do SCSS tylko bez klamereki średników,
- LESS – preprocesor CSS zapewniający takie same dodatkowe funkcje jak CSS, lecz z inną składnią,
- Stylus – preprocesor CSS z minimalistyczną składnią.

Aby ułatwić sobie stylowanie framework Angular nie zapewnia wbudowanej opcji wyboru frameworka z gotowymi klasami css umożliwiającymi ładną stylizację i responsywność aplikacji takich jak bootstrap czy tailwind. Aby ich użyć należy je zainstalować według instrukcji zawartej na oficjalnej stronie frameworka.

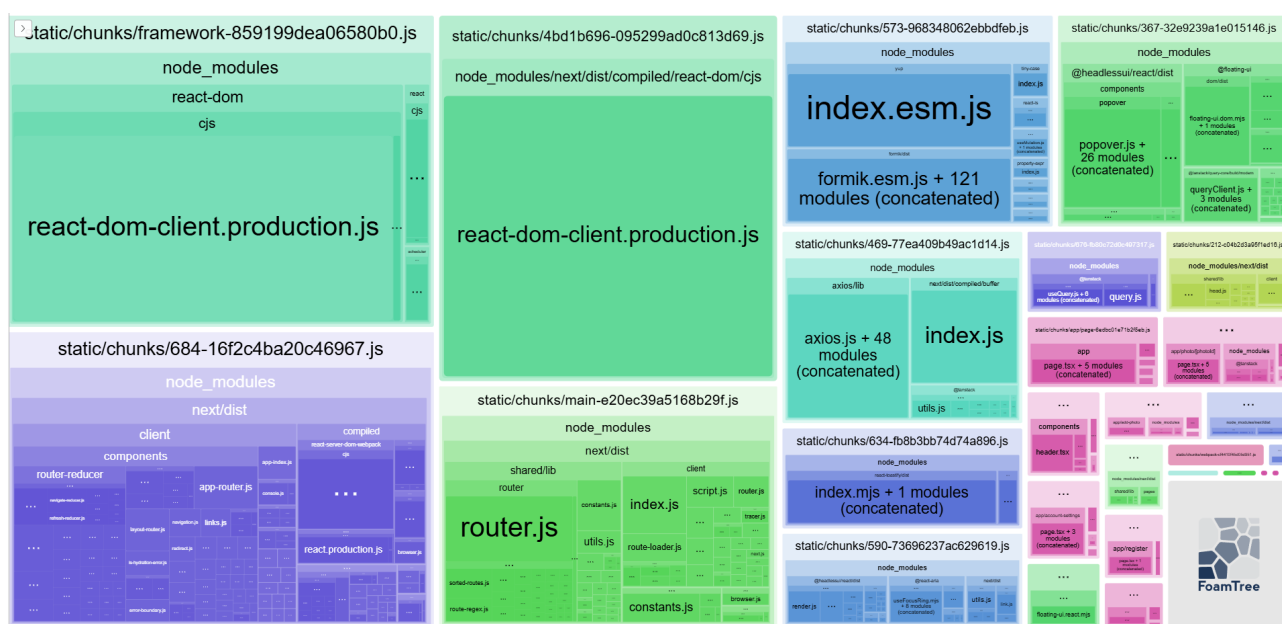
6. Porównanie optymalizacji aplikacji

6.1. Framework Next.js

W trakcie realizacji tego etapu skupiono się na optymalizacji aplikacji pod kątem wydajności, obejmującej zarówno żądania sieciowe, jak i renderowanie komponentów.

W zakresie żądań sieciowych wykorzystano bibliotekę React Query, która zapewnia wbudowane mechanizmy cachowania odpowiedzi API. Przykładem jest hook `useGetAllPhotosQuery`, który automatycznie przechowuje dane w pamięci podręcznej, co pozwala na ograniczenie liczby żądań do serwera. Dodatkowo, w miejscach wymagających dynamicznego wyszukiwania, takich jak komponent wyszukiwania zdjęć, zastosowano technikę debouncingu, aby ograniczyć liczbę wysyłanych żądań podczas wpisywania tekstu przez użytkownika. Pamięć podręczną zapytań można następnie aktualizować, za pomocą przypisanych wcześniej kluczy (każde wywołanie `useQuery` posiada własny unikalny klucz zapytania).

Analiza rozmiaru bundle'a została przeprowadzona za pomocą narzędzia `@next/bundle-analyzer`, które zostało skonfigurowane w pliku `next.config.ts`. Analiza wykazała, że największe zależności to `react-dom` oraz `formik`. Na podstawie wyników podjęto działania optymalizacyjne, takie jak usunięcie nieużywanych bibliotek oraz zastąpienie cięższych zależności lżejszymi odpowiednikami tam, gdzie było to możliwe.



Rysunek 7: Wynik analizy rozmiaru bundle'a aplikacji Next.js.

Wyniki analizy bundle'a przedstawiono w formie wizualizacji, która wskazała kluczowe obszary do optymalizacji. Na przykład, zmniejszono rozmiar wspólnych chunków, poprzez eliminację zbędnych importów i optymalizację kodu.

Na koniec przeprowadzono testy wydajnościowe, które potwierdziły poprawę szybkości ładowania aplikacji. Zmniejszono rozmiar początkowego JavaScriptu ładowanego na stronie głównej do 145 kB, co znacząco wpłynęło na czas renderowania. Dzięki zastosowanym optymalizacjom aplikacja działa szybciej i bardziej responsywnie, co przekłada się na lepsze doświadczenie użytkownika.

Ostateczny rozmiar utworzonej paczki ze wszystkimi niezbędnymi plikami wynosił 3.6 MB, ponieważ wliczono w to pliki HTML, CSS oraz utworzone pliki .js.

6.2. Framework Angular

Framework angular zapewnia pakiet `@angular/pwa` umożliwiający automatyczną konfigurację systemowego workera odpowiadającego za cacheowanie treści renderowanej na stronie. Dzięki temu unikane jest ponowne pobieranie tych samych obrazków, arkuszy stylów lub skryptów przez klienta dzięki czemu poprawiane jest jego odczucie płynności i stabilności podczas korzystania ze strony. Poniższy listing przedstawia konfigurację zapewniającą cacheowanie obrazków, stylów oraz skryptów.

```
{
  "$schema": "./node_modules/@angular/service-worker/config/schema.json",
  "index": "/index.html",
  "assetGroups": [ { "name": "app", "installMode": "prefetch", "resources": {
    "files": [ "/favicon.ico", "/index.csr.html", "/index.html", "/manifest.webmanifest",
"/*.css", "/*.js" ] } },
    { "name": "assets", "installMode": "lazy", "updateMode": "prefetch", "resources": {
      "files": [ "/*/*.*.(svg|cur|jpg|jpeg|png|apng|webp|avif|gif|otf|ttf|woff|woff2)" ]
    } }, { "name": "server-images", "installMode": "lazy", "updateMode": "prefetch",
      "resources": { "urls": [ "/api/v1/photos/**", "/api/v1/photos/public/**" ] }
    }
  ]
}
```

Listing 6: Konfiguracja zapewniająca cache'owanie obrazków, stylów oraz skryptów we frameworku angular.

Następnie wygenerowano analizę bundle aplikacji. Na poniższym rysunku widoczna jest mapa zasobów, przedstawiająca iż cała paczka po zbudowaniu do wersji produkcyjnej waży 460 kilobajtów, przedstawia iż aplikacja podzielona jest na kilka głównych plików:

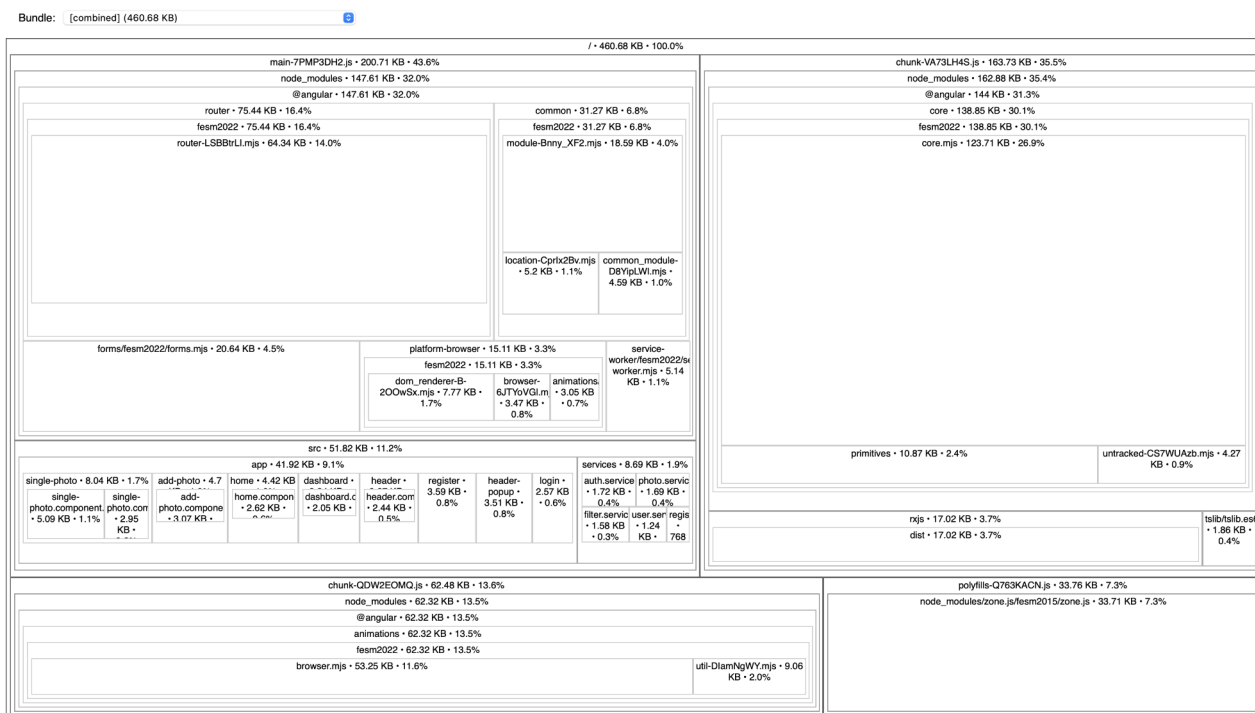
- main-7PMP3DH2.js: 200 KB – napisany kod + pakiet routingu,
- chunk-VA73LH4S.js: 163 KB – pakiet angular core i moduły,
- chunk-QDW2EOMQ.js: 62 KB – pakiet angular animations i typescript library,
- polyfills-....: 33 KB – strefy i pollyfils.

Dodatkowo, na widocznym obrazku widoczne jest, iż największe utworzone komponenty do obsługi ścieżek aplikacji ważą następująco:

- single-photo: 8 KB,
- add-photo: 4.7 KB,
- home: 4.4 KB,
- dashboard: 2 KB,
- header: ~2.4 KB.

Kolejno folder services waży łącznie 8.69 KB, a największe klasy serwisów ważą:

- photo.service: ~1.7 KB,
- auth.service: ~1.7 KB,
- filter.service: ~1.5 KB.



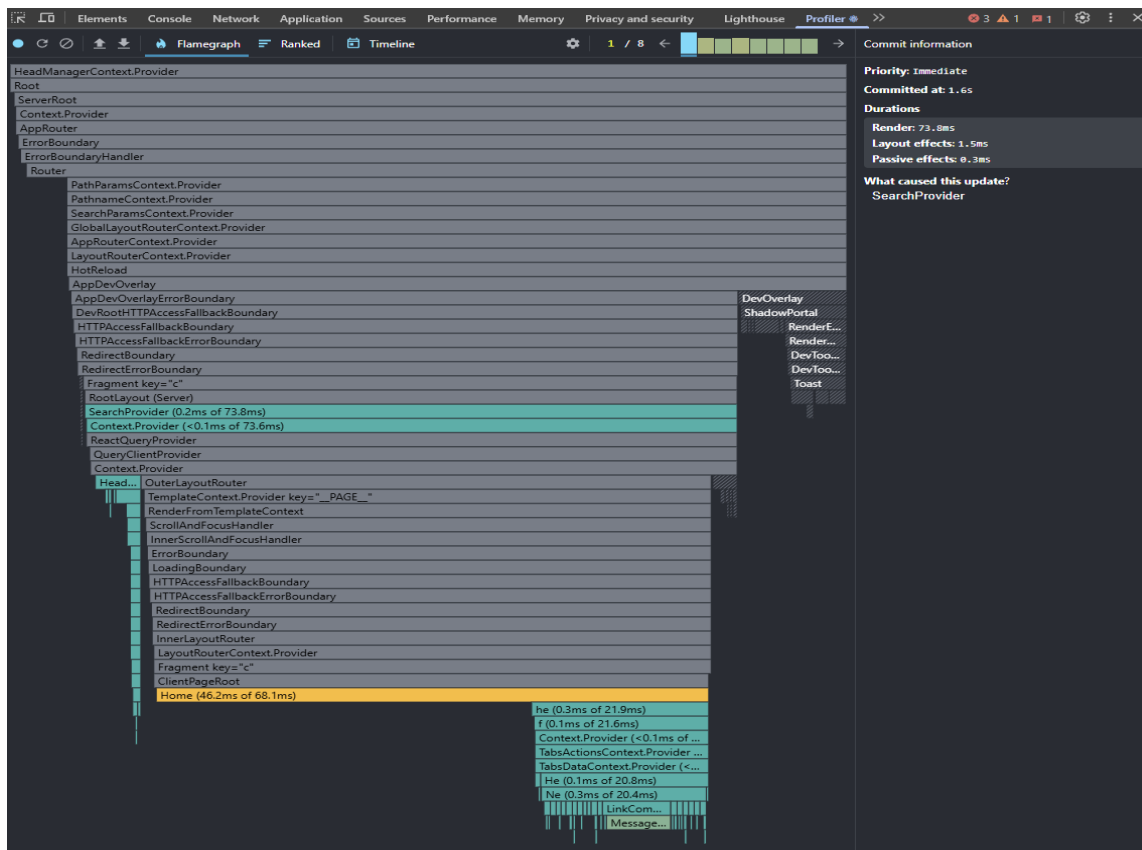
Rysunek 8: Wygenerowana mapa skryptów aplikacji napisanej we frameworku angular wraz z rozmiarami.

Ostateczny rozmiar utworzonej paczki ze wszystkimi niezbędnymi plikami wynosił 8.6 MB, ponieważ wliczono w to pliki HTML, CSS oraz utworzone pliki .js.

7. Porównanie szybkości podmiiany dużej ilości elementów w DOM'ie.

7.1. Framework Next.js

W trakcie realizacji etapu 4.6 przeprowadzono analizę szybkości podmiiany dużej ilości elementów w DOM, koncentrując się na płynności operacji oraz wpływie dynamicznych zmian na wydajność przeglądarki. W tym celu wykorzystano narzędzie React Profiler w przeglądarce Chrome, które pozwoliło na szczegółowe zbadanie czasu renderowania komponentów oraz liczby aktualizacji widoku.



Rysunek 9: Wyniki uzyskane z narzędzia React Profiler podczas filtrowania treści na stronie za pomocą funkcjonalności search.

Na załączonym obrazie widoczny jest wynik działania React Profilera podczas stosowania filtrów w pasku wyszukiwania aplikacji. Zaznaczone na zielono elementy wskazują komponenty, które zostały prerenderowane w wyniku zmiany stanu aplikacji. W szczególności, komponent Home został wyróżniony jako główny punkt renderowania, co wskazuje, że zmiana wprowadziła aktualizacje w jego podkomponentach.

Profilowanie wykazało, że renderowanie komponentu Home zajęło 46.2 ms z całkowitego czasu 68.1 ms. W tym czasie zaktualizowane zostały również inne komponenty, takie jak SearchProvider, który odpowiada za zarządzanie stanem wyszukiwania. Warto zauważyć, że czas renderowania SearchProvider wyniósł 2.0 ms, co wskazuje na jego efektywność w obsłudze zmian stanu.

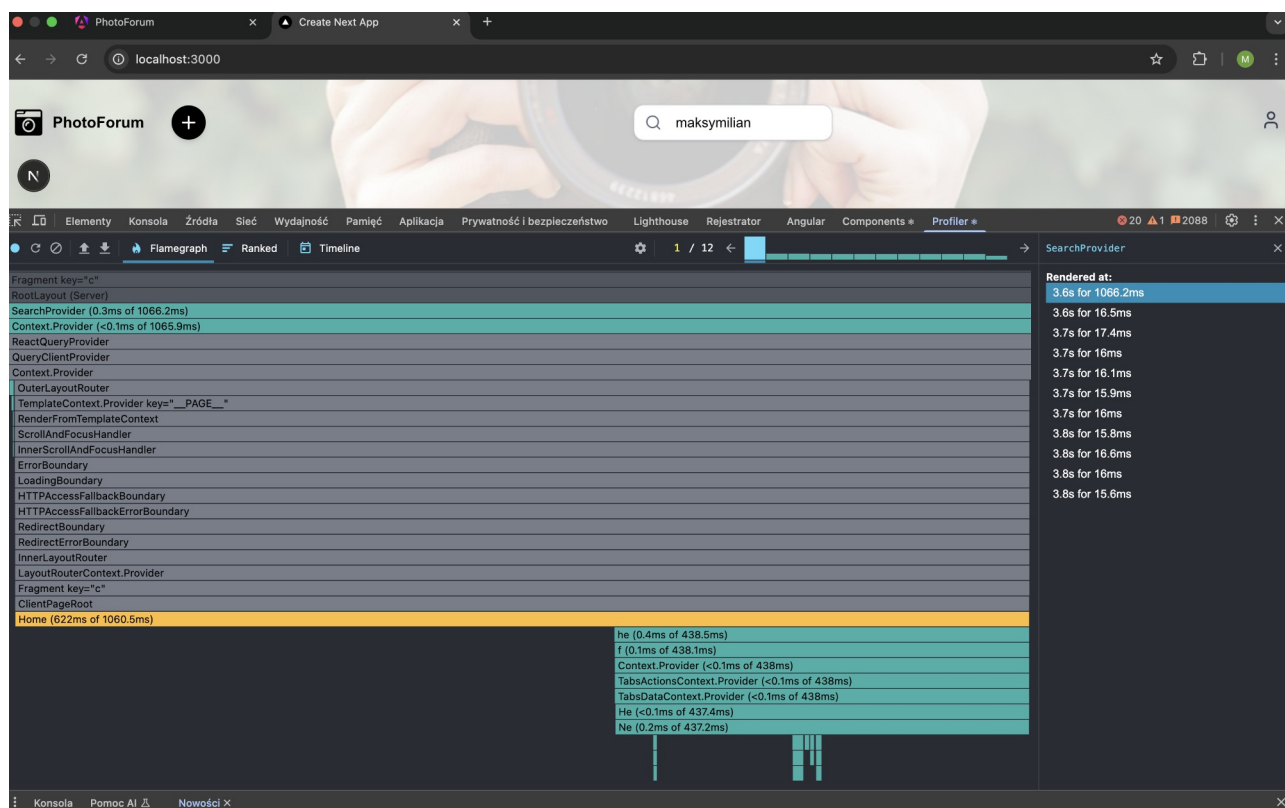
Analiza liczby re-renderów wykazała, że zmiana w pasku wyszukiwania spowodowała aktualizację widoku dla wszystkich widocznych elementów listy zdjęć. Każdy element listy został prerenderowany, co było konieczne w celu odzwierciedlenia wyników filtrowania. W tym

przypadku, mechanizm React Query oraz zarządzanie stanem w SearchProvider odegrały kluczową rolę w ograniczeniu liczby niepotrzebnych aktualizacji.

Podczas testów dynamicznego filtrowania zdjęć w pasku wyszukiwania aplikacja wykazała dobrą płynność działania. Pomimo dużej liczby elementów w liście, czas renderowania pozostał na akceptowalnym poziomie. W celu dalszej optymalizacji można rozważyć zastosowanie wirtualizacji listy za pomocą biblioteki react-window, co pozwoliłoby na renderowanie jedynie widocznych elementów.

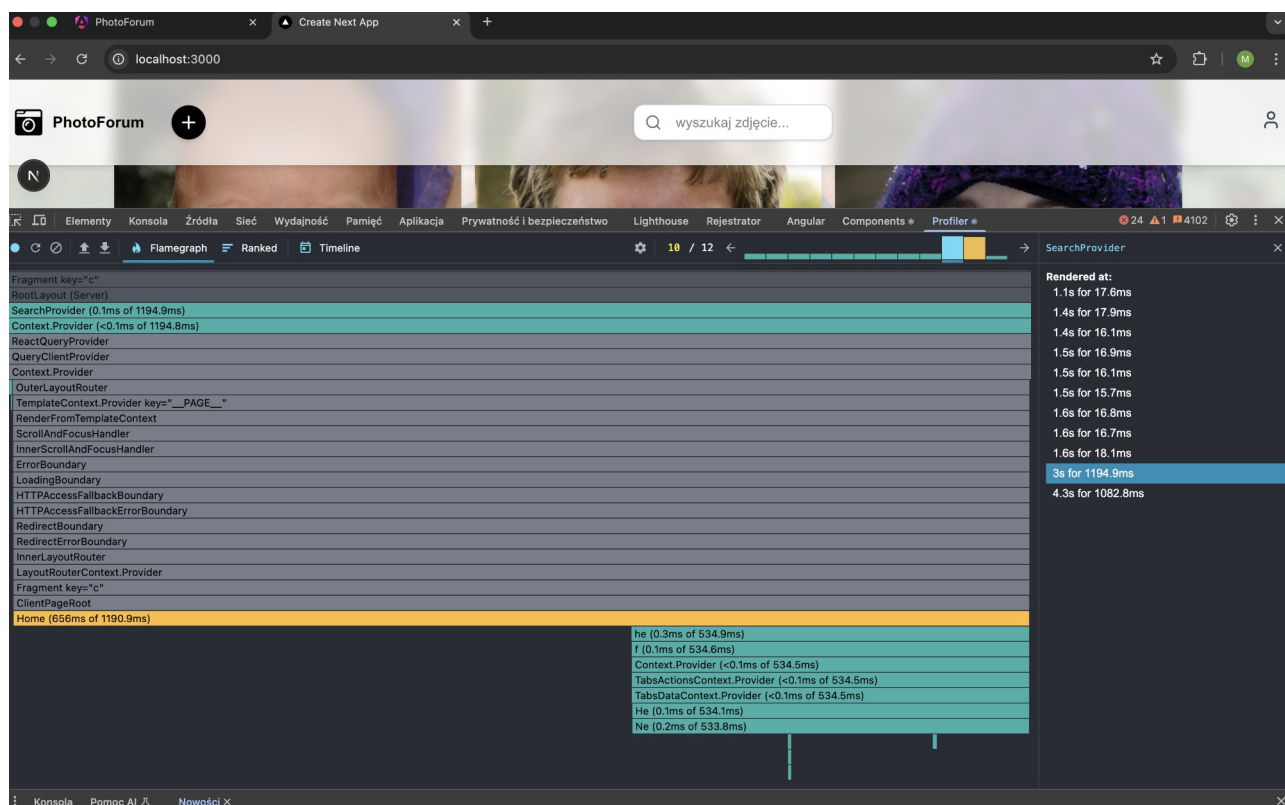
Analiza wykazała, że aplikacja radzi sobie dobrze z dynamiczną podmianą dużych zbiorów danych w DOM. Mechanizmy zarządzania stanem oraz React Query skutecznie ograniczają liczbę niepotrzebnych aktualizacji widoku. Dalsze optymalizacje, takie jak wirtualizacja listy, mogą jeszcze bardziej poprawić wydajność w przypadku bardzo dużych zbiorów danych.

Kolejno, analizę powtórzono dla większej liczby zdjęć wysokiej jakości z nieoptymalizowanym rozmiarem oraz formatem (.png). Analizę wykonano usuwając ze struktury DOM'u 1000 elementów oraz dodając je z powrotem za pomocą funkcji wyszukiwania. Poniższy rysunek przedstawia szybkość wyrenderowania strony domowej bez zdjęć po odfiltrowaniu zdjęć.



Rysunek 10: Wyniki uzyskane z narzędzia React Profiler podczas odfiltrowania treści na stronie za pomocą funkcjonalności search.

Poniższy rysunek prezentuje wyniki przywrócenia odfiltrowanych wcześniej elementów do struktury DOM'u (czas wyrenderowania szkieletów).



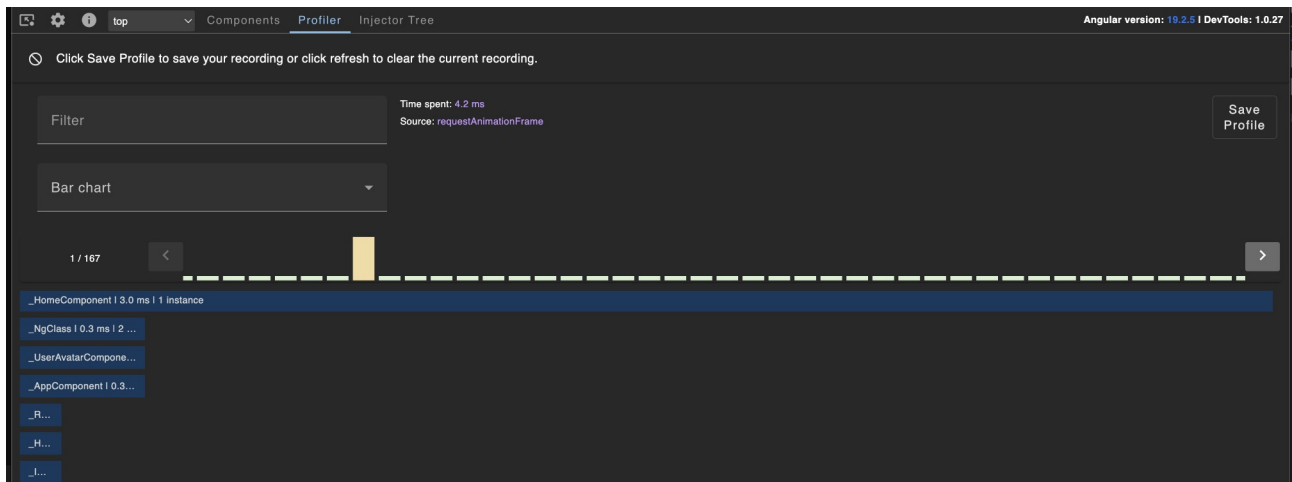
Rysunek 11: Wyniki uzyskane z narzędzia React Profiler podczas przywrócenia odfiltrowanych treści na stronie za pomocą funkcjonalności search.

Wyniki wykazują, że aplikacja utworzona we frameworku next.js poradziła sobie z zadaniem odfiltrowania treści i wyrenderowania początkowego widoku graficznego dla użytkownika w czasie 1,066 sekundy. Przywrócenie 1000 szkieletów do struktury zajęło 1,119 sekundy.

7.2. Framework Angular

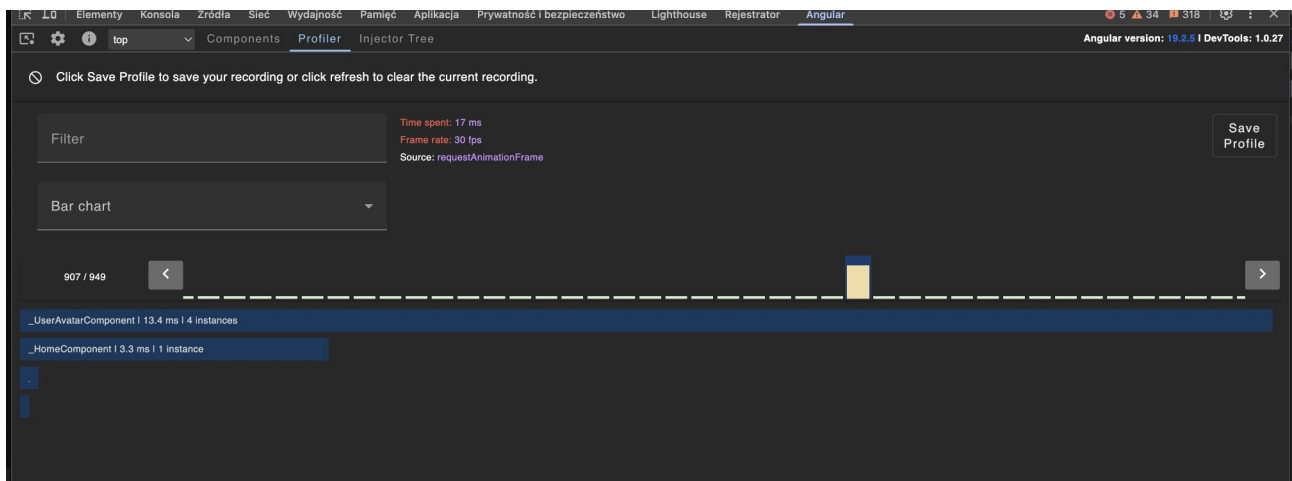
Aby przetestować podmianę treści zawierającej duże obiekty tj. obrazki oraz napisy zaimplementowano mechanizm filtrowania obrazków użytkownika odfiltrowujący jego ulubione obrazki oraz wszystkie obrazki, a także wyszukujący obrazki po danej frazie. Poniższy rysunek przedstawia szybkość usunięcia 15 obrazków ze struktury DOM'u strony i komponenty, których

zmiana struktury drzewa dotyczyła. W celu eksperymentu zmieniono filter ze wszystkich obrazków zalogowanego użytkownika na jego ulubione obrazki.



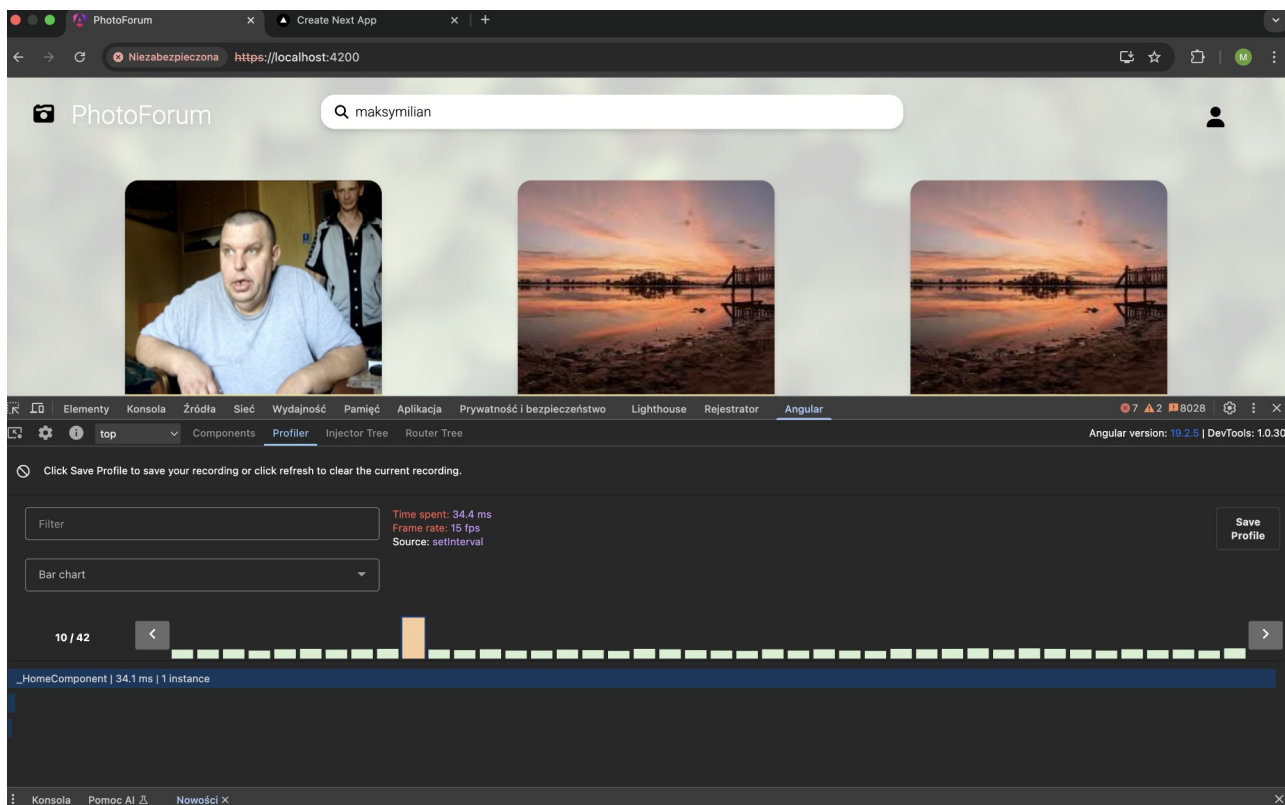
Rysunek 12: Szybkość usunięcia 15 obrazków ze struktury DOM.

Poniższy zrzut ekranu przedstawia szybkość dodania do struktury DOM'u 15 obrazków na nowo zmieniając filtr 'Ulubione' na 'Wszystkie'.



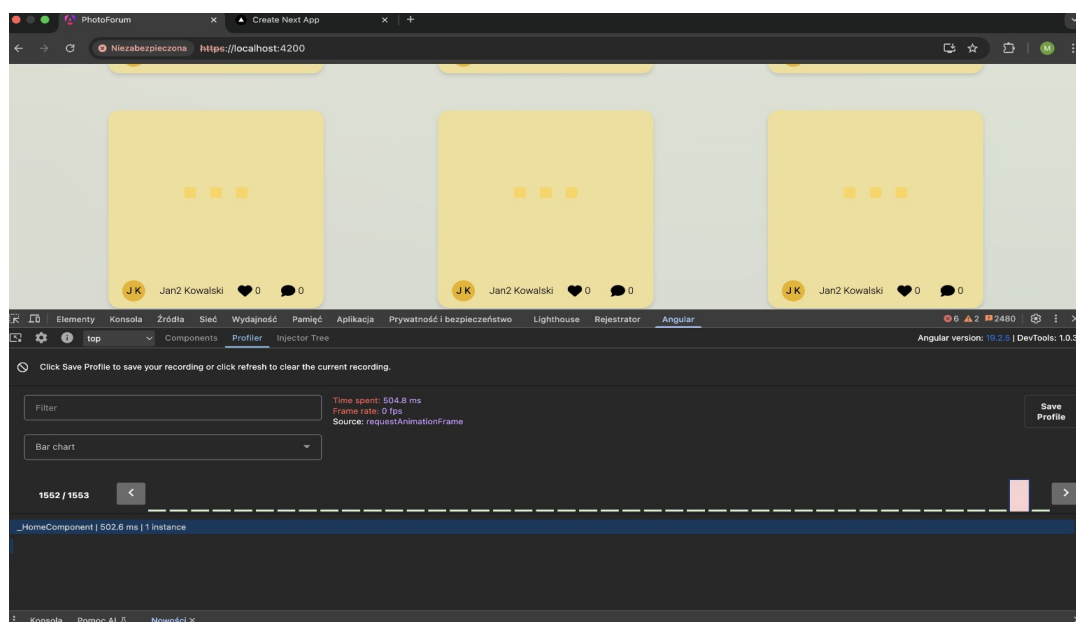
Rysunek 13: Szybkość dodania 15 obrazków do struktury DOM.

Eksperyment powtórzono jednakże tym razem aby wykonać miarodajne porównanie z frameworkiem next.js odfiltrowując 1000 elementów ze struktury i dodając je z powrotem, aby zobaczyć jak utworzona aplikacja poradzi sobie z wyrenderowaniem szkieletu graficznej strony dla użytkownika. Poniższy rysunek przedstawia szybkość odfiltrowania elementów ze struktury DOM.



Rysunek 14: Szybkość odfiltrowania 1000 elementów ze struktury DOM oraz wyrenderowania szkieletów zdjęć.

Następnie sprawdzono czas przywrócenia 1000 elementów do struktury. Poniższy rysunek przedstawia uzyskany wynik.



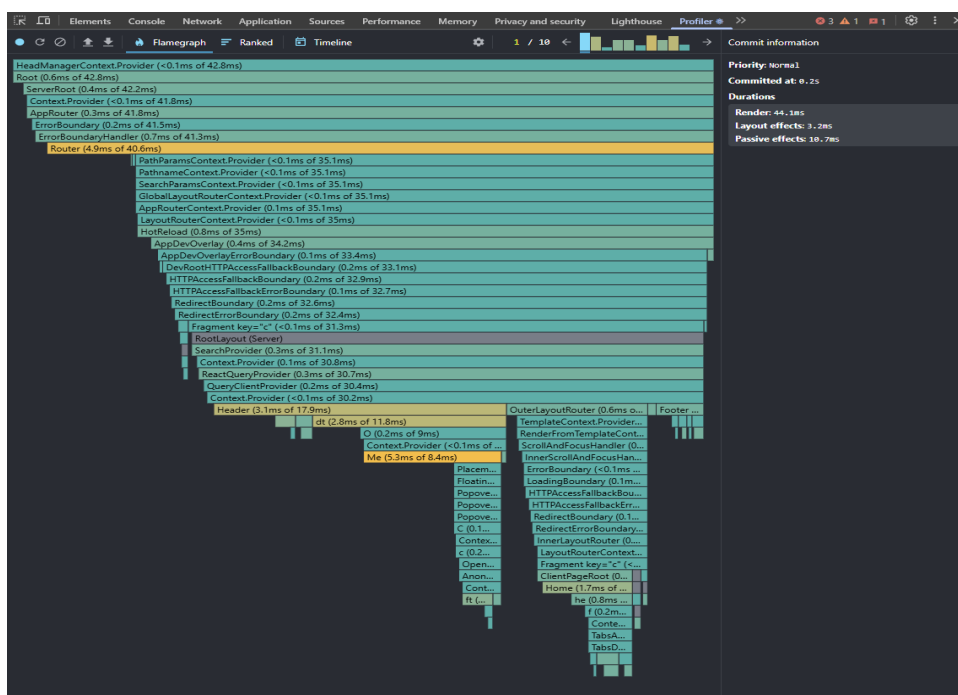
Rysunek 15: Szybkość dodania 1000 elementów do struktury DOM oraz wyrenderowania szkieletów zdjęć.

Wyniki wykazują, że aplikacja utworzona we frameworku angular poradziła sobie z zadaniem odfiltrowania treści i wyrenderowania początkowego widoku graficznego dla użytkownika w czasie 0,034 sekundy. Przywrócenie 1000 szkieletów do struktury zajęło 0,540 sekundy.

8. Porównanie wydajności aplikacji

8.1. Next.js

W trakcie realizacji etapu 4.7 przeprowadzono analizę wydajności aplikacji, koncentrując się na kluczowych wskaźnikach, takich jak czas ładowania, szybkość interakcji oraz efektywność przetwarzania danych. Wykorzystano narzędzia takie jak React Profiler oraz wbudowane mechanizmy przeglądarki Chrome, aby szczegółowo zbadać wydajność aplikacji w różnych scenariuszach użytkowania.



Rysunek 16: Przedstawienie czasu załadowania oraz rerenderu komponentów wraz z kluczowymi wskaźnikami uzyskane z narzędzia React Profiler.

Na załączonym obrazie widoczny jest wynik działania React Profiler podczas renderowania całej struktury DOM strony Next.js. Zaznaczone na zielono elementy wskazują komponenty, które zostały przerenderowane w wyniku zmian stanu aplikacji.

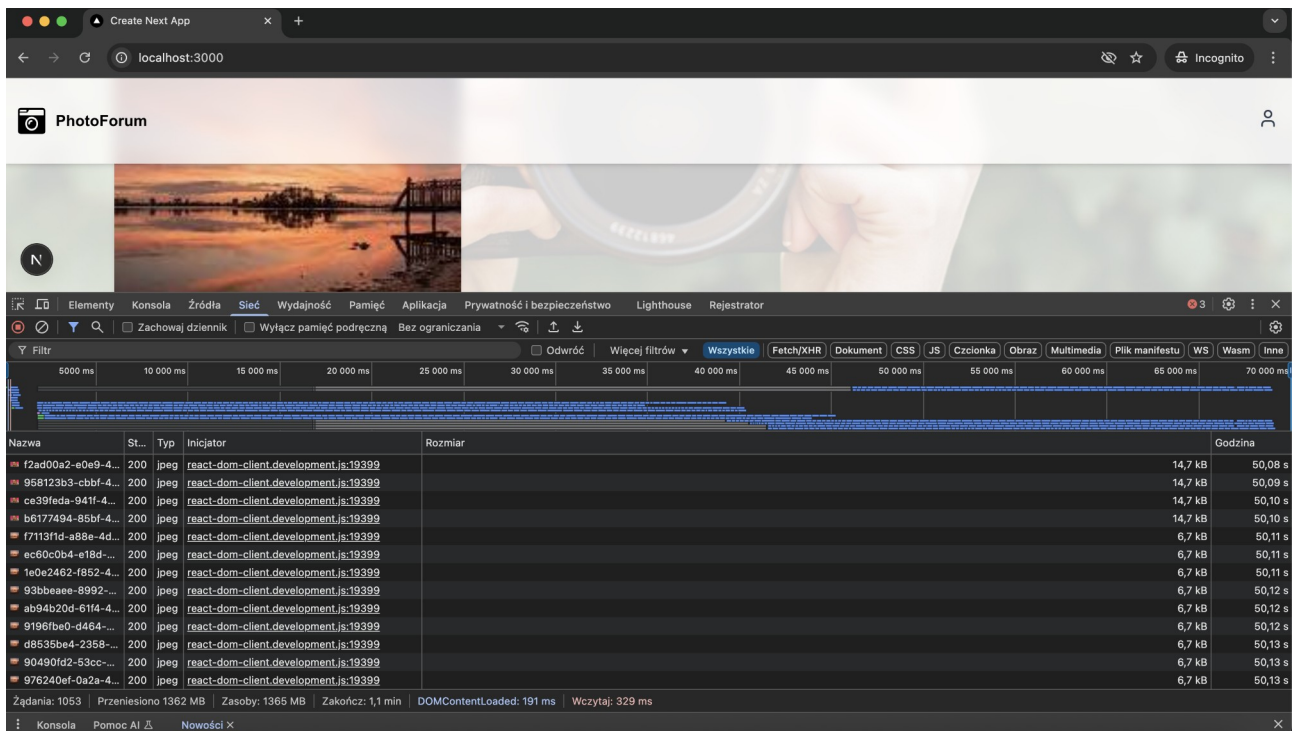
Profilowanie przeprowadzone podczas początkowego ładowania (initial loading) aplikacji Next.js wykazało, że całkowity czas renderowania wyniósł 42.8 ms. W tym czasie różne komponenty aplikacji były aktualizowane, co obejmowało zarówno zarządzanie stanem, jak i renderowanie interfejsu użytkownika. Dodatkowo, inne operacje, takie jak ładowanie danych i konfiguracja providerów, miały istotny wpływ na całkowity czas ładowania aplikacji.

Analiza wpływu zapytań sieciowych na wydajność aplikacji wykazała, że mechanizm React Query skutecznie zarządza pamięcią podręczną, co pozwala na ograniczenie liczby żądań do serwera. Na przykład, dane dotyczące autoryzacji użytkownika są przechowywane w pamięci podręcznej i odświeżane tylko w razie potrzeby, co znacząco zmniejsza obciążenie sieciowe.

Podczas testów dynamicznej aktualizacji danych, takich jak filtrowanie zdjęć w pasku wyszukiwania, aplikacja wykazała dobrą płynność działania. Czas renderowania pozostał na akceptowalnym poziomie, a mechanizmy zarządzania stanem skutecznie ograniczyły liczbę niepotrzebnych aktualizacji widoku. W przypadku paginacji i przełączania widoków, czas podmiany dużej ilości zasobów, takich jak obrazy, był również zadowalający dzięki zastosowaniu optymalizacji ładowania obrazów w Next.js.

Podsumowując, analiza wykazała, że aplikacja działa wydajnie zarówno podczas pierwszego ładowania, jak i w trakcie dynamicznych interakcji użytkownika. Mechanizmy zarządzania stanem oraz optymalizacje wprowadzone w kodzie skutecznie poprawiają płynność działania aplikacji. Dalsze optymalizacje mogą jeszcze bardziej zwiększyć wydajność w przypadku bardzo dużych zbiorów danych.

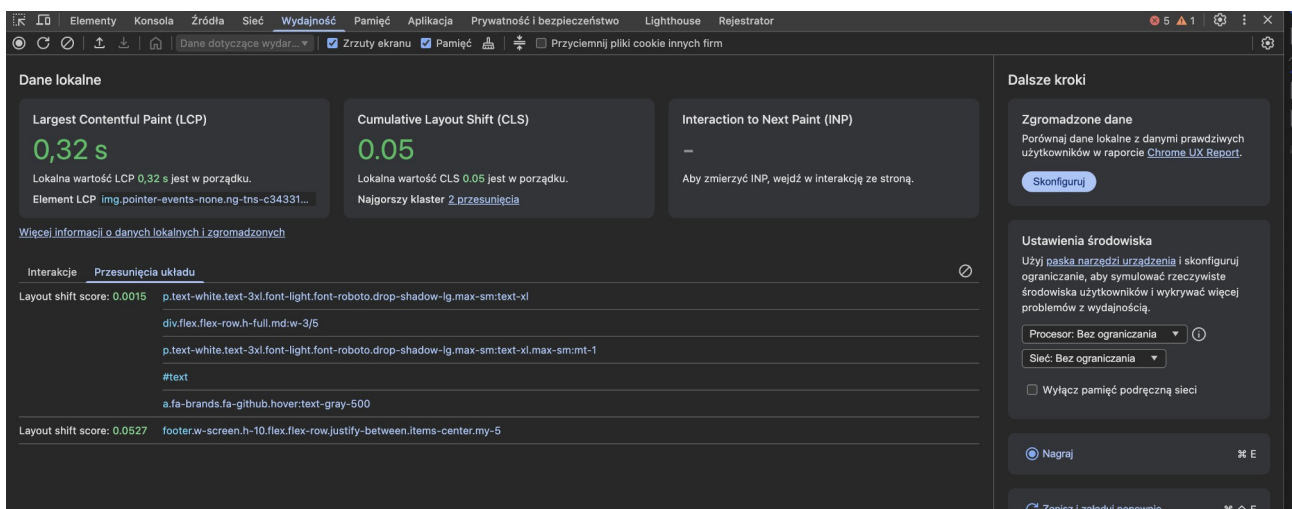
Po zrealizowaniu tego eksperymentu dla małej ilości zdjęć do bazy dodano 1000 nowych zdjęć i sprawdzono czas załadowania całej struktury strony wraz z zaczytaniem skryptów oraz obrazków. Poniższy rysunek przedstawia uzyskany wynik. Czas załadowania strony ze wszystkimi zdjęciami to 1,1 minuty.



Rysunek 17: Szybkość załadowania całej struktury strony we frameworku Next.js.

8.2. Angular

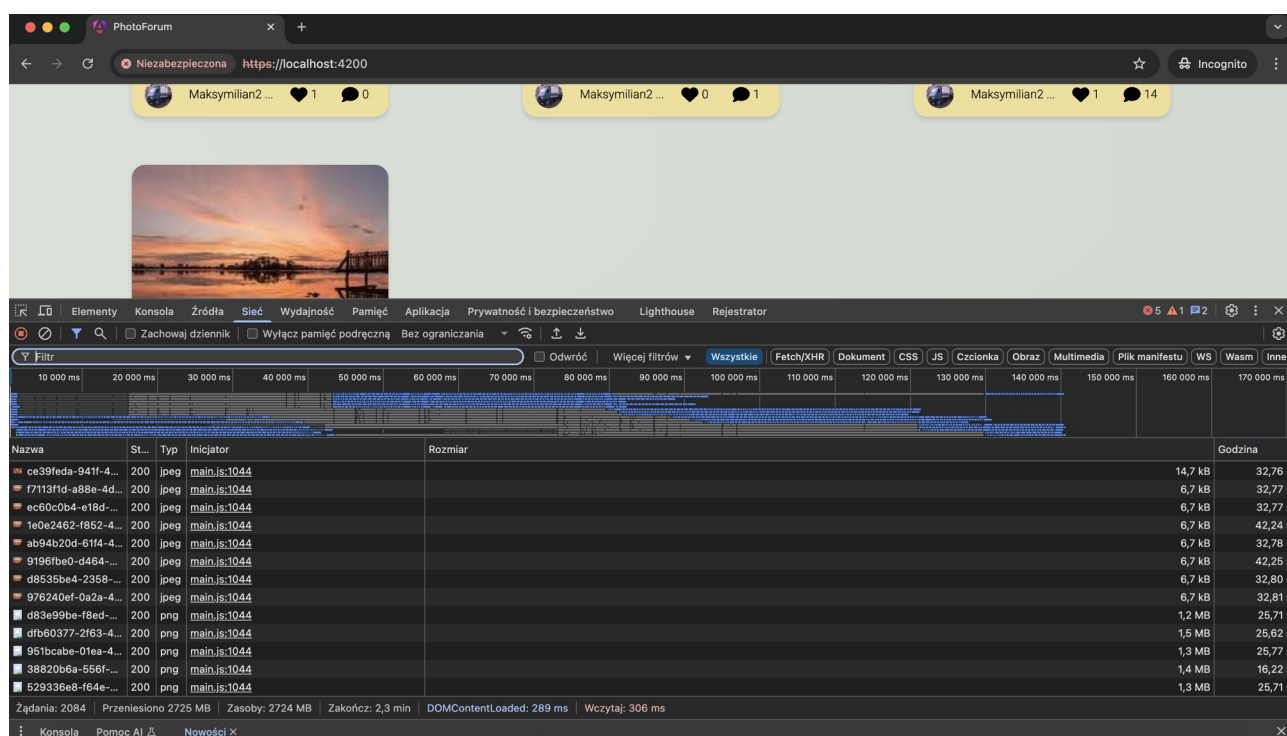
W celu zbadania wydajności aplikacji uruchomiono konsolę wraz z zakładką Wydajność oraz przeglądarkę Chrome w trybie Incognito. Następnie wprowadzono adres url aplikacji i sprawdzono uzyskane wyniki.



Rysunek 18: Uzyskane wyniki szybkości działania strony przy pierwszym jej załadowaniu.

Z powyższego zrzutu ekranu można odczytać, iż czas wyrenderowania największego elementu strony wynosi 0,32 sekundy. Kolejno, wartość CLS wynosi 0.05 co oznacza, że układ strony prawie się nie przesuwają. Z uzyskanych wyników można wywnioskować iż strona ładuje się dobrze i nie powoduje bezsensownej frustracji z powodu długiego oczekiwania.

Kolejno, do rzetelniejszej analizy przeprowadzono eksperyment analogiczny do frameworka next.js z testem szybkości załadowania całej struktury strony z 1000 zdjęciami. Poniższy rysunek przedstawia uzyskany wynik. Czas załadowania strony ze wszystkimi zdjęciami to 2,3 minuty.

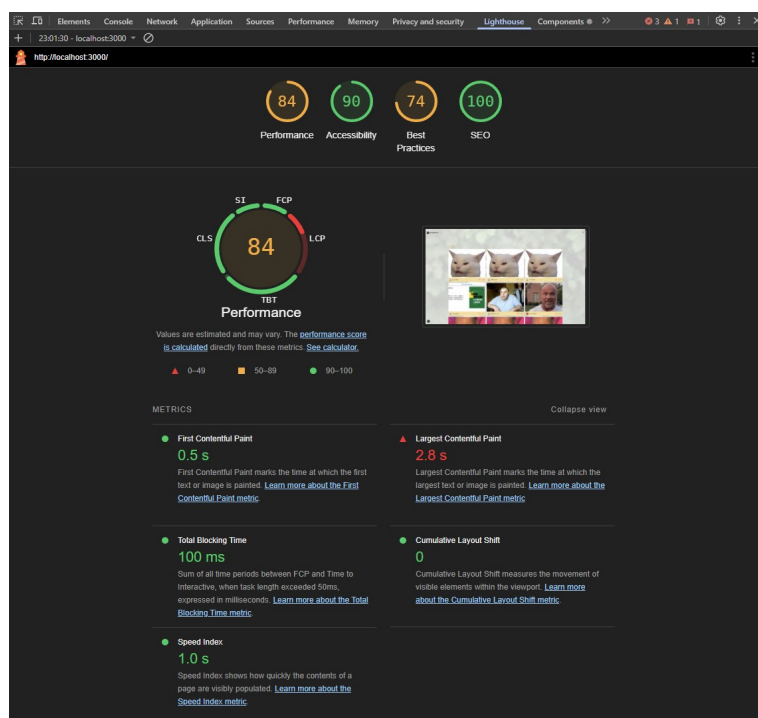


Rysunek 19: Szybkość załadowania całej struktury strony we frameworku Angular.

9. Porównanie wsparcia SEO

9.1. Next.js

W trakcie realizacji etapu 4.8 przeprowadzono analizę wsparcia SEO aplikacji, koncentrując się na kluczowych aspektach, takich jak poprawność metadanych, struktura adresów URL, indeksowalność treści oraz szybkość ładowania strony. Wykorzystano narzędzie Lighthouse w przeglądarce Chrome, które dostarczyło szczegółowych informacji na temat optymalizacji pod kątem wyszukiwarek.



Rysunek 20: Uzyskane wyniki SEO dla frameworka Next.js.

Weryfikacja poprawności generowania meta tagów, tytułów i opisów stron. Analiza wykazała, że aplikacja generuje podstawowe meta tagi, takie jak title i description, które są kluczowe dla SEO. Warto jednak upewnić się, że każda podstrona posiada unikalne i odpowiednio zoptymalizowane meta tagi, aby zwiększyć widoczność w wynikach wyszukiwania. Dodatkowo, należy zweryfikować, czy tagi Open Graph i Twitter Cards są poprawnie skonfigurowane, co może wpłynąć na lepsze udostępnianie treści w mediach społecznościowych.

Aplikacja korzysta z renderowania po stronie serwera (SSR), co zapewnia, że treści są dostępne dla botów wyszukiwarek. Dzięki temu dynamiczne elementy, takie jak listy zdjęć, są widoczne w kodzie HTML generowanym na serwerze.

Struktura adresów URL w aplikacji jest czytelna i przyjazna dla użytkowników oraz wyszukiwarek. Przykładowo, adresy takie jak /photo/[photoId] czy /account-settings są intuicyjne i odzwierciedlają zawartość stron.

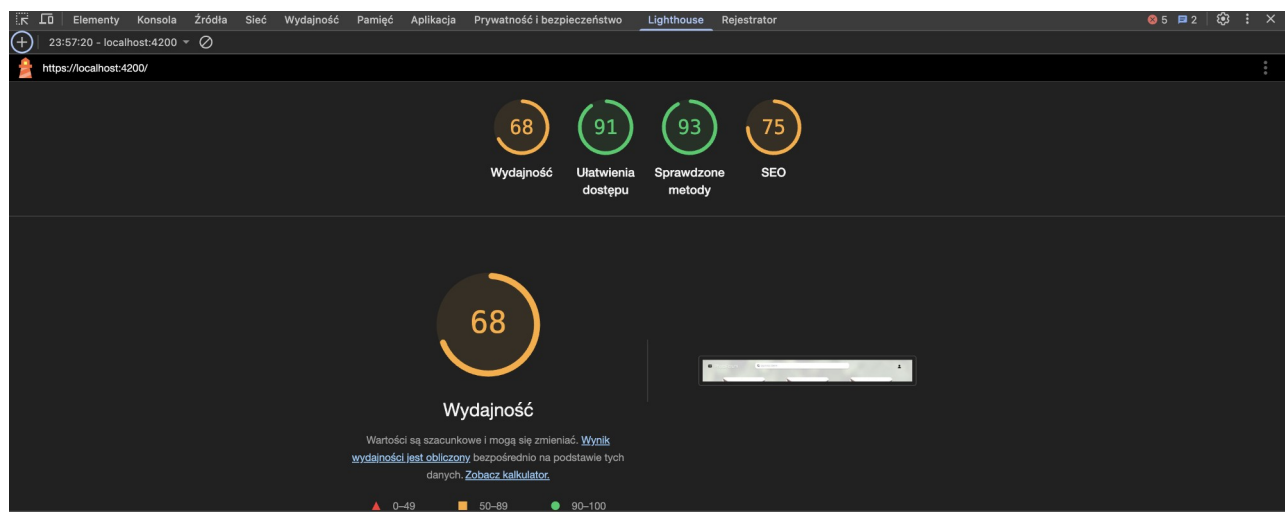
Na załączonym obrazie widoczny jest wynik testu Lighthouse, który wskazuje na 84 punkty w kategorii wydajności. Kluczowe metryki, takie jak First Contentful Paint (0.5 s) i Speed Index (1.0 s), są na bardzo dobrym poziomie.

Dzięki zastosowaniu SSR oraz React Query, dynamiczne treści, takie jak zdjęcia i dane użytkowników, są dostępne dla botów wyszukiwarek. Mechanizmy te zapewniają, że dane są renderowane na serwerze i dostarczane w pełni w kodzie HTML, co zwiększa ich indeksowalność.

Aplikacja jest dobrze zoptymalizowana pod kątem SEO, jednak istnieją obszary do dalszej poprawy, takie jak optymalizacja Largest Contentful Paint oraz weryfikacja konfiguracji plików robots.txt i sitemap.xml. Dzięki zastosowaniu SSR i czytelnej strukturze URL, aplikacja jest przyjazna zarówno dla użytkowników, jak i botów wyszukiwarek.

9.2. Angular

Framework Angular nie zapewnia wbudowanego wsparcia SEO strony co zrzuca odpowiedzialność za poprawną konfigurację i optymalizację treści strony na programistę. Poniższy zrzut ekranu przedstawia uzyskane wyniki SEO dla frameworka Angular.



Rysunek 21: Uzyskane wyniki SEO dla frameworka Angular.

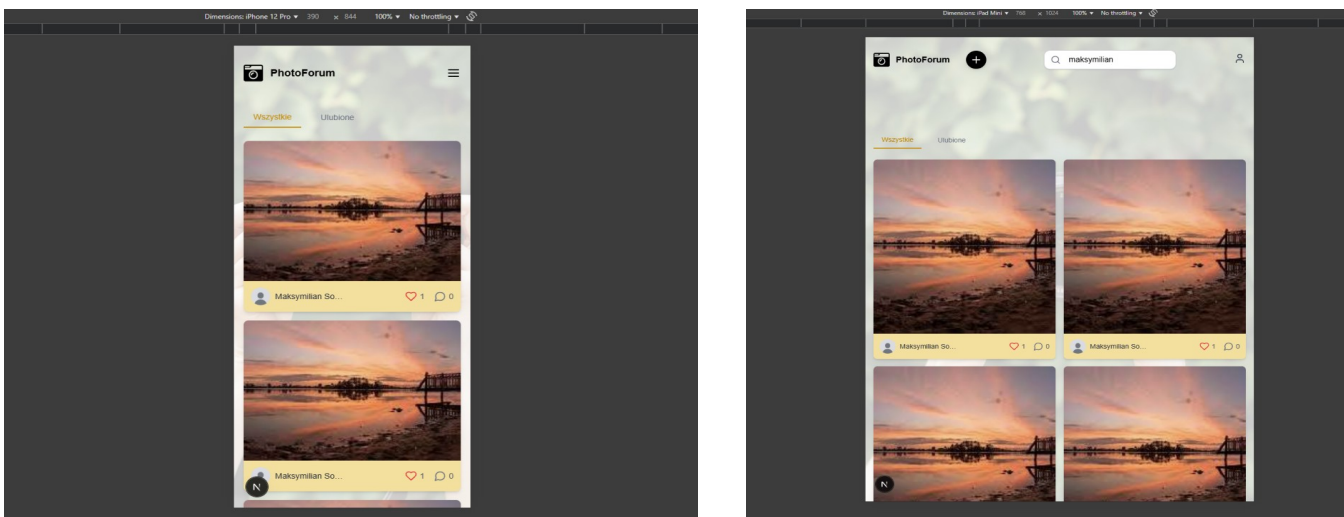
Z powyższego obrazka, można odczytać iż wydajność aplikacji jest przeciętna. Może być spowodowana tym, iż aktualnie zastosowane techniki optymalizacji nie są wystarczające oraz tym, iż testowana aplikacja była uruchamiana w trybie developerskim. Kolejno widoczne jest, iż dostępność aplikacji jest na poziomie 91 co oznacza, że aplikacja jest dobrze dostosowana dla osób z różnymi dysfunkcjami. Następnie wynik najlepszych praktyk wynosi 93 co oznacza, że aplikacja nie ma wielkich luk bezpieczeństwa czy nieprawidłowych skryptów. Ostatnim, najważniejszym w tej

analizie parametrem jest SEO score, który plasuje się na poziomie 75 co jest dobrym rezultatem, lecz mógłby być lepszym. Spowodowane to może być brakiem skonfigurowania meta tagów, webmanifestu czy też poprawności użytych tagów semantycznych na stronie.

10. Porównanie responsywności aplikacji.

10.1. Next.js

W trakcie realizacji etapu 4.9 przeprowadzono analizę wsparcia responsywności aplikacji, koncentrując się na dostosowaniu interfejsu użytkownika do różnych rozdzielczości ekranów i typów urządzeń. Testy obejmowały zarówno urządzenia mobilne, jak i tablety oraz komputery stacjonarne, w celu zapewnienia spójności i użyteczności interfejsu.



Rysunek 22: Widok Home w wersji webowej oraz mobilnej we frameworku Next.js.

Aplikacja została przetestowana na szerokim zakresie rozdzielczości, od małych ekranów mobilnych (320px) po duże monitory desktopowe (1920px i więcej). Układ strony, w tym siatka zdjęć w widoku głównym, dynamicznie dostosowuje się do dostępnej przestrzeni dzięki zastosowaniu klas Tailwind CSS, takich jak `grid-cols-1`, `sm:grid-cols-2` i `lg:grid-cols-3`. Testy wykazały, że układ pozostaje czytelny i estetyczny na wszystkich testowanych rozdzielczościach.

Elementy interfejsu, takie jak karty zdjęć, przyciski i pola wyszukiwania, poprawnie skalują się w zależności od rozmiaru ekranu. Na przykład, obrazy w siatce zdjęć są renderowane z klasą `object-cover`, co zapewnia ich odpowiednie dopasowanie bez zniekształceń. Dodatkowo,

zastosowanie klas takich jak `w-full` i `h-64 md:h-96` pozwala na dynamiczne dostosowanie wysokości obrazów w zależności od rozdzielczości.

Interakcje, takie jak nawigacja między zakładkami "Wszystkie" i "Ulubione", zostały przetestowane na urządzeniach dotykowych. Dzięki zastosowaniu komponentów z biblioteki `@headlessui/react`, takich jak `TabGroup` i `TabPanel`, przejścia między widokami są płynne i intuicyjne. Dodatkowo, elementy interaktywne, takie jak przyciski i linki, mają odpowiednie obszary dotykowe, co zwiększa ich użyteczność na ekranach dotykowych.

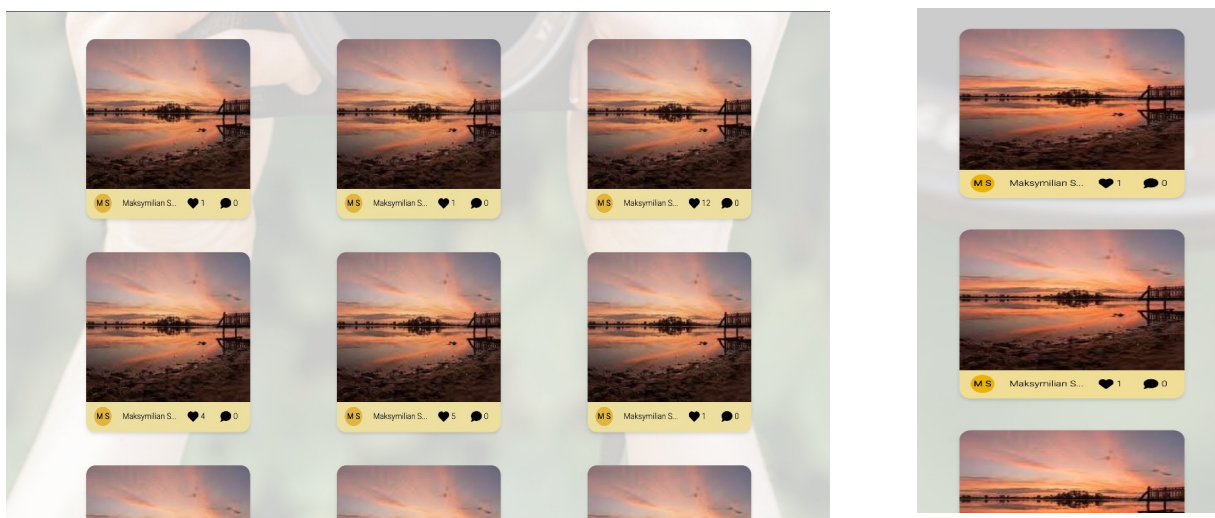
Elementy interaktywne, takie jak przyciski nawigacyjne i pola wyszukiwania, zostały przetestowane pod kątem dostępności na urządzeniach mobilnych. Na przykład, pole wyszukiwania w komponencie `Header` jest łatwo dostępne i responsywne, a jego stylizacja z klasami `focus-within:ring-2` i `focus-within:ring-yellow-500` zapewnia wizualne wskazanie aktywności.

Aplikacja została zaprojektowana zgodnie z zasadami `Mobile-First Design`, co oznacza, że stylizacja dla urządzeń mobilnych jest domyślna, a dodatkowe reguły dla większych ekranów są dodawane za pomocą klas takich jak `sm:`, `md:` i `lg:`. Dzięki temu aplikacja działa płynnie na urządzeniach o różnych rozmiarach ekranów, a jej interfejs pozostaje spójny i użyteczny.

Analiza wykazała, że aplikacja jest w pełni responsywna i dostosowana do różnych urządzeń. Układ strony, skalowanie elementów oraz interakcje zostały zoptymalizowane pod kątem użyteczności i dostępności. Dzięki zastosowaniu podejścia `Mobile-First Design` oraz narzędzi takich jak `Tailwind CSS` i `@headlessui/react`, aplikacja zapewnia wysoką jakość doświadczenia użytkownika na wszystkich typach urządzeń.

10.2. Angular

Podobnie jak w przypadku frameworka Next.js aplikacja utworzona za pomocą frameworka Angular zapewnia pełną responsywność zarówno na komputerach stacjonarnych, przenośnych oraz mobilnych. Responsywność ta została osiągnięta za pomocą podejścia Mobile First Design oraz frameworka TailwindCSS. Poniższy rysunek przedstawia wygląd widoku home w wersji webowej oraz mobilnej.



Rysunek 23: Widok Home w wersji webowej oraz mobilnej we frameworku Angular.

Obrazy w aplikacji zostały wyskalowane za pomocą usgawienia stałej szerokości i wysokości a także klasy object-cover powodującej dopasowanie się rozmiaru obrazka do rozmiaru kontenera rodzica.

Responsywność aplikacji wytestowano od rozdzielczości 344 x 882 kończąc na rozdzielczości 1920x1080. Testy nie wykazały żadnych nieprawidłowości mogących powodować frustrację czy niezadowolenie użytkownika.

Interakcje z użytkownikiem wykonywane są za pomocą własnych komponentów oraz zapewnienia płynnych przejść oraz animacji z pakietu `@angular/cdk`. Podsumowując, cała aplikacja utworzona we frameworku angular jest responsywna i przejrzysta w użytkowaniu.

11. Porównanie statystyk używalności obu frameworków

11.1. Next.js

Według ankiety przeprowadzonej przez platformę stackoverflow w roku 2024 framework był użytkowany przez następujące grupy ankietujących:

- 17.9% wszystkich ankietowanych,
- 18.6% profesjonalnych deweloperów,
- 17.9% osób uczących się programowania,
- 12.7% innych osób, które programują.

Również zgodnie z danymi odczytanymi dnia 17.05.2025 framework next.js odnotował na platformie npm 10 264 611 tygodniowych pobrań. Poniższy rysunek ukazuje wskazaną statystykę.

Repository

📁 github.com/vercel/next.js

Homepage

🔗 nextjs.org

⬇ Weekly Downloads

10 264 611



Rysunek 24: Ilość pobrań frameworka next.js.

Kolejno, z platformy github odczytano ilość gwiazdek przyznanych przez użytkowników jak i ilość problemów zgłaszanych przez użytkowników. Statystyka z dnia 17.05.2025 roku wynosiła:

- 132 000 gwiazdek,

- 2400 zgłoszonych problemów.

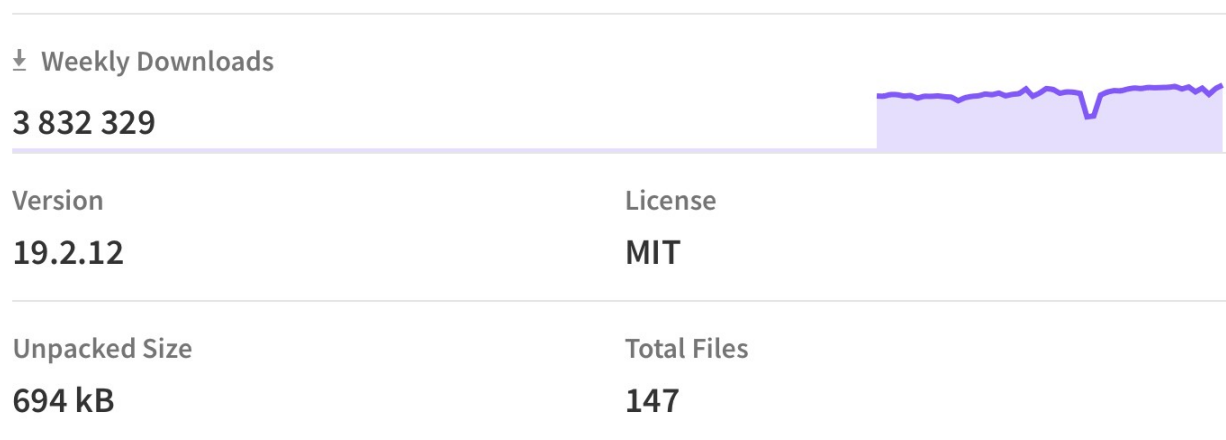
Wyniki te świadczą o tym, iż framework next.js jest bardzo popularnym narzędziem wykorzystywanym w podobnym stopniu przez wszystkie przebadane grupy programistów.

11.2. Angular

Według ankiety przeprowadzonej przez platformę stackoverflow w roku 2024 framework był użytkowany przez następujące grupy ankietujących:

- 17.1% wszystkich ankietowanych,
- 19.4% profesjonalnych deweloperów,
- 7.7% osób uczących się programowania,
- 10% innych osób, które programują.

Również zgodnie z danymi odczytanymi dnia 17.05.2025 framework angular odnotował na platformie npm 3 832 329 tygodniowych pobrań. Poniższy rysunek ukazuje wskazaną statystykę.



Rysunek 25: Ilość pobrań frameworka angular.

Kolejno, z platformy github odczytano ilość gwiazdek przyznanych przez użytkowników jak i ilość problemów zgłaszanych przez użytkowników. Statystyka z dnia 17.05.2025 roku wynosiła:

- 26900 gwiazdek,
- 268 zgłoszonych problemów.

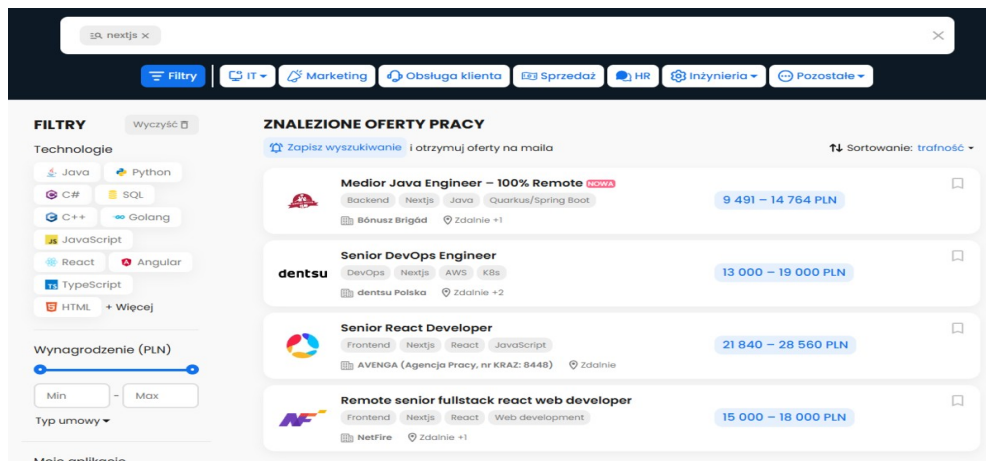
Wyniki te świadczą o tym, iż framework angular jest częściej używany przez profesjonalnych programistów oraz nie jest bardzo popularnym frameworkiem. Dodatkowo z uzyskanej ankiety można wywnioskować, iż framework ten nie jest wybierany jako pierwszy framework do nauki lub do programowania hobbistycznego.

12. Porównanie ofert pracy

Porównanie zostało przeprowadzone wykorzystując dane pochodzące z największego polskiego jobboarda technologicznego – nofluffjobs, oraz bazując na oficjalnych statystykach dostępnych w internecie.

12.1. Next.js

W przypadku frameworka Next.js liczba dostępnych ofert pracy jest niewielka i wynosi zaledwie 4. Dane pochodzą z dnia 24.05.2025 i skierowane są wyłącznie dla doświadczonych programistów (exp 3+). Zarobki w przypadku dostępnych ofert seniorskim plasują się w zakresie 13-28 tys. Wszystkie oferty wymagają jednak znajomości zestawu innych bibliotek, takich jak React.

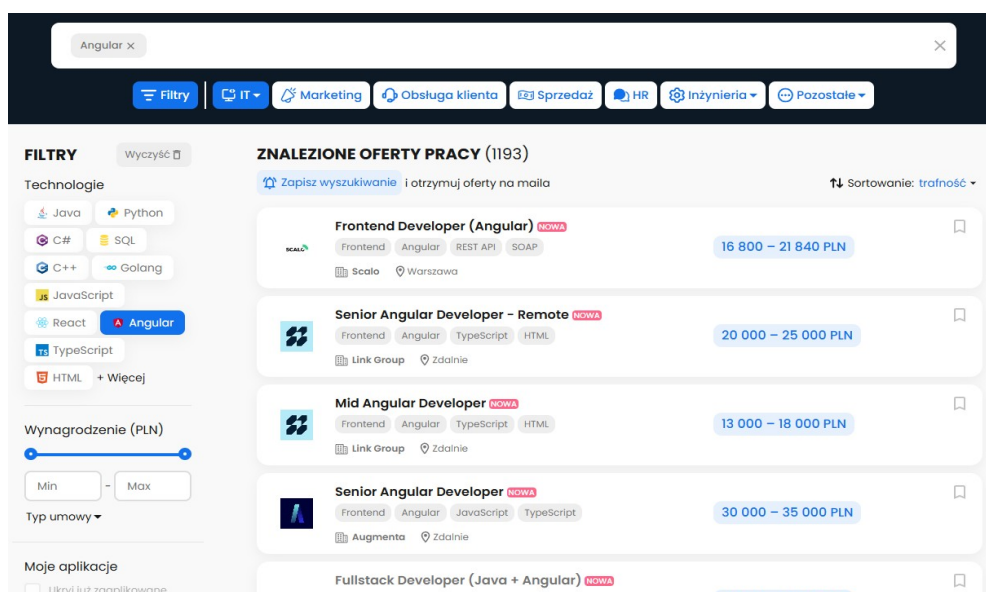


Rysunek 26: Dostępne oferty pracy w frameworku Next.js.

12.2. Angular

W przypadku Angulara sytuacja wygląda zupełnie odmiennie, gdzie dostępnych jest aż 1193 ofert pracy skierowanych dla programistów o różnym stopniu doświadczenia. Widelki płacowe wyglądają następująco:

- Junior (5 – 12 tyś. zł.)
- Mid (13 – 20 tyś. zł.)
- Senior (15 – 32 tyś. zł.)



Rysunek 27: Dostępne oferty pracy w frameworku Angular

Podsumowując, można zauważyć tendencję, że Next.js rzadko stanowi główny element oferty pracy – częściej jest on dodatkiem do innych technologii, które kandydat powinien znać. W przypadku stanowisk frontendowych najczęściej wymaga się znajomości Reacta lub Angulara, co tłumaczy dużą liczbę ofert związanych z tymi frameworkami, zwłaszcza Angulara.

13. Analiza struktury projektu

13.1. Next.js

Analiza struktury projektu w Next.js została przeprowadzona w celu oceny organizacji kodu, jego modułowości oraz łatwości w utrzymaniu i skalowaniu. W ramach tej analizy uwzględniono podział kodu na moduły i komponenty, czytelność struktury katalogów, zarządzanie stanem aplikacji, sposób komunikacji między komponentami oraz zgodność z najlepszymi praktykami frameworka.

Podział kodu na moduły i komponenty w projekcie jest dobrze zorganizowany. Aplikacja została podzielona na logiczne moduły, takie jak komponenty, hooki, interfejsy oraz inne zasoby, co pozwala na łatwe zarządzanie kodem. Komponenty są odpowiednio modularne i wielokrotnego użytku, co wspiera zasadę DRY (Don't Repeat Yourself). Na przykład, komponenty takie jak TabGroup i TabPanel są używane do zarządzania zakładkami w interfejsie użytkownika, a logika renderowania zdjęć została wydzielona do funkcji renderPhotos, co zwiększa czytelność i ułatwia testowanie.

Struktura katalogów i organizacja plików są przejrzyste i zgodne z konwencjami frameworka Next.js. Projekt zawiera dedykowane katalogi dla komponentów, hooków, interfejsów oraz innych zasobów, co ułatwia nawigację i odnajdywanie potrzebnych elementów. Na przykład, hooki odpowiedzialne za pobieranie danych są umieszczone w jednym miejscu, co pozwala na ich łatwe zarządzanie i rozbudowę. Taka organizacja wspiera pracę zespołową i minimalizuje ryzyko konfliktów w kodzie.

Zarządzanie stanem aplikacji zostało zrealizowane za pomocą dedykowanych hooków, takich jak useGetAllPhotosQuery czy useIsAuthorizedQuery. Hooki te umożliwiają efektywne pobieranie danych i zarządzanie stanem w sposób deklaratywny. Komunikacja między

komponentami odbywa się głównie za pomocą propsów oraz kontekstów, co zapewnia spójność i minimalizuje ryzyko błędów. Dodatkowo, zastosowanie hooka `useSearch` pozwala na centralne zarządzanie zapytaniem wyszukiwania, co ułatwia jego integrację z różnymi częściami aplikacji.

Projekt jest łatwy do rozwijania i dostosowania do nowych wymagań dzięki modularnej strukturze i zastosowaniu wzorców projektowych. Dodanie nowych funkcjonalności, takich jak kolejne zakładki czy nowe typy danych, wymaga jedynie rozszerzenia istniejących modułów lub dodania nowych komponentów. Na przykład, dodanie nowej zakładki w interfejsie użytkownika wymaga jedynie modyfikacji komponentu `TabGroup` oraz odpowiedniego hooka do zarządzania danymi. Dzięki temu projekt jest skalowalny i przygotowany na przyszłe zmiany.

Kod jest zgodny z najlepszymi praktykami frameworka Next.js. Wykorzystano funkcje takie jak dynamiczne routowanie, hooki React oraz komponenty klienta. Zastosowanie tych technik pozwala na optymalizację wydajności oraz poprawę doświadczenia użytkownika. Dodatkowo, kod jest czytelny, dobrze sformatowany i zgodny z zasadami czystego kodu, co ułatwia jego zrozumienie i utrzymanie.

Podsumowując, struktura projektu Next.js została zaprojektowana w sposób przemyślany i zgodny z dobrymi praktykami. Modularność, przejrzystość oraz zgodność z konwencjami frameworka sprawiają, że projekt jest łatwy w utrzymaniu, rozwijaniu i skalowaniu.

13.2. Angular

Struktura projektu Angular została oparta o konwencje narzucane przez Angular CLI, co zapewnia jednolitość oraz ułatwia nawigację po projekcie. Projekt został podzielony na katalogi `components`, `services`, `models`, `modules`, `assets` oraz `environments`, co pozwala na łatwe odnajdywanie odpowiednich zasobów. Komponenty zostały zorganizowane w sposób hierarchiczny, zgodny z logiką aplikacji, co wspiera zasadę separacji odpowiedzialności i ułatwia ponowne ich wykorzystanie.

Każdy komponent Angular składa się z pliku TypeScript (.ts), szablonu HTML (.html) oraz stylów (.scss lub .css), co zapewnia przejrzystość oraz separację logiki od warstwy prezentacji.

Zastosowanie dekoratorów, takich jak `@Component` i `@Injectable`, umożliwia jasne określenie funkcji poszczególnych elementów aplikacji. Moduły (`@NgModule`) grupują komponenty oraz serwisy w logiczne całości, co wspiera organizację kodu oraz jego skalowalność.

Zarządzanie stanem aplikacji oraz komunikacja między komponentami odbywa się głównie za pomocą serwisów (services). Serwisy korzystają z wstrzykiwania zależności (Dependency Injection), co pozwala na łatwe dzielenie danych i logiki biznesowej pomiędzy komponentami. W przypadku bardziej złożonych struktur danych wykorzystano `BehaviorSubject`, co umożliwia reaktywne zarządzanie stanem aplikacji.

Angular wspiera stosowanie wzorców projektowych, takich jak MVVM (Model-View-ViewModel), co pozwala na utrzymanie czystej architektury oraz zapewnia łatwość rozbudowy. Dodanie nowych funkcjonalności, np. dodatkowych komponentów lub widoków, odbywa się poprzez generowanie ich za pomocą CLI, co skraca czas implementacji oraz minimalizuje ryzyko błędów.

Projekt został stworzony zgodnie z najlepszymi praktykami frameworka Angular. Wykorzystano mechanizmy routingu (`RouterModule`), silne typowanie dzięki TypeScript oraz modularność aplikacji. Dzięki temu struktura projektu jest przejrzysta, skalowalna i umożliwia dalszy rozwój przy zachowaniu wysokiej jakości kodu.

Podsumowując, struktura projektu w Angularze cechuje się wysokim poziomem organizacji oraz zgodnością z konwencjami frameworka. Dzięki modularności, wykorzystaniu usług i narzędzi oferowanych przez Angular, projekt jest łatwy w utrzymaniu, rozwijaniu oraz przygotowany na dalszą skalowalność narzędzi testowych.

14. Analiza narzędzi testowych

14.1. Next.js

Analiza narzędzi testowych w projekcie opartym na Next.js obejmuje ocenę wsparcia dla testowania aplikacji na różnych poziomach. W projekcie zastosowano bibliotekę `@testing-library/react` do testów jednostkowych i integracyjnych oraz `jest` jako framework testowy.

Środowisko testowe zostało skonfigurowane w sposób prosty i intuicyjny. Wykorzystano `jest` jako główny framework testowy, który jest łatwy do integracji z Next.js. Dodatkowo, `@testing-library/react` umożliwia testowanie komponentów React w sposób zbliżony do rzeczywistego użytkowania aplikacji. W projekcie zastosowano również mockowanie zapytań HTTP za pomocą `jest.mock`, co pozwala na izolację testów od zewnętrznych zależności, takich jak API.

Dzięki wykorzystaniu `@testing-library/react` testy mogą być uruchamiane w różnych środowiskach, takich jak przeglądarki symulowane przez `jsdom`. Testy integracyjne, takie jak renderowanie komponentów z różnymi dostawcami (`QueryClientProvider` i `SearchProvider`), zapewniają, że aplikacja działa poprawnie w różnych konfiguracjach. Chociaż testy end-to-end nie zostały uwzględnione w tym fragmencie, framework Next.js wspiera narzędzia takie jak Cypress lub Playwright, które mogą być użyte do testowania na rzeczywistych urządzeniach.

Next.js nie posiada wbudowanego frameworka testowego, ale rekomenduje użycie `jest` i `@testing-library/react`. W projekcie zastosowano te narzędzia, co pozwala na przeprowadzanie testów jednostkowych i integracyjnych. Dodatkowo, mockowanie zapytań HTTP za pomocą `jest.mock` umożliwia testowanie logiki aplikacji bez konieczności wykonywania rzeczywistych zapytań sieciowych.

Next.js jest kompatybilny z wieloma popularnymi frameworkami testowymi. W projekcie zintegrowano `jest` i `@testing-library/react`, co umożliwia kompleksowe testowanie aplikacji. Dodatkowo, zastosowanie `QueryClientProvider` i `SearchProvider` w testach integracyjnych pokazuje, że framework wspiera testowanie złożonych scenariuszy, takich jak zarządzanie stanem i kontekstem. Możliwość mockowania zapytań HTTP za pomocą `jest` dodatkowo upraszcza testowanie logiki biznesowej.

Podsumowując, projekt wykorzystuje solidne narzędzia testowe, które zapewniają łatwość konfiguracji, możliwość testowania w różnych środowiskach oraz integrację z popularnymi frameworkami. Dzięki temu aplikacja jest dobrze przygotowana do utrzymania wysokiej jakości kodu i automatyzacji testów.

14.2. Angular

Analiza narzędzi testowych w projekcie Angular koncentruje się na ocenie wbudowanego wsparcia dla testowania jednostkowego, integracyjnego oraz możliwości testowania end-to-end. Angular domyślnie integruje narzędzia takie jak Jasmine i Karma, co pozwala na szybkie rozpoczęcie testowania bez potrzeby dodatkowej konfiguracji.

Środowisko testowe zostało zautomatyzowane i przygotowane przez Angular CLI. Każdy komponent, serwis czy pipe generowany przez CLI zawiera domyślny plik testowy (*.spec.ts), co wspiera konsekwentne pokrycie kodu testami. Jasmine służy jako framework do pisania testów, oferując czytelną składnię oraz bogaty zestaw matcherów, natomiast Karma odpowiada za uruchamianie testów w przeglądarce.

Testy jednostkowe w projekcie pozwalają na sprawdzanie poprawności działania komponentów, metod serwisów oraz reakcji na zmiany danych wejściowych. Dzięki zastosowaniu narzędzi takich jak TestBed, testy mogą być izolowane i kontrolowane, a zależności mockowane z użyciem HttpTestingModule lub własnych klas serwisowych. W testach uwzględniono także scenariusze z wykorzystaniem ReactiveFormsModule, co umożliwia testowanie walidacji i interakcji użytkownika z formularzem.

Dla testów end-to-end Angular oferuje integrację z narzędziami takimi jak Cypress lub wcześniej z Protractor (który jest obecnie przestarzały). Choć w analizowanym projekcie testy E2E nie zostały zaimplementowane, framework dostarcza pełne wsparcie do ich wdrożenia.

Zaletą Angulara jest silne typowanie TypeScript, które dodatkowo ułatwia pisanie testów poprzez podpowiedzi typów i redukcję błędów na etapie kompilacji. Dzięki integracji CLI z narzędziami testowymi, testy mogą być łatwo uruchamiane z poziomu terminala (ng test) i mogą być monitorowane pod kątem pokrycia kodu (code coverage).

Podsumowując, Angular zapewnia kompletne środowisko do testowania aplikacji od samego początku projektu. Dzięki domyślnym narzędziom i konwencjom testowym, testy są łatwe do napisania, integracji oraz uruchamiania. Projekt wykorzystuje potencjał testowy Angulara w zakresie testów jednostkowych i integracyjnych, co wspiera wysoką jakość kodu oraz stabilność aplikacji.

15. Analiza obsługi formularzy

15.1. Next.js

Analiza obsługi formularzy w projekcie opartym na Next.js wykazała, że framework wspiera tworzenie i zarządzanie formularzami w sposób intuicyjny i efektywny. W projekcie wykorzystano bibliotekę Formik do zarządzania formularzami oraz Yup do walidacji danych, co pozwala na łatwe definiowanie formularzy, obsługę błędów i integrację z danymi wejściowymi użytkownika.

Definiowanie formularzy w projekcie jest proste dzięki wykorzystaniu Formik. Formularze są tworzone jako komponenty React, a ich pola są powiązane z danymi za pomocą metody getFieldProps. Na przykład, w formularzu logowania i rejestracji, pola takie jak login i password są łatwo powiązane z obiektami initialValues i zarządzane przez Formik. Dzięki temu dane wejściowe użytkownika są automatycznie synchronizowane z modelem danych, co upraszcza proces implementacji.

Walidacja danych w projekcie jest realizowana za pomocą biblioteki Yup, która jest rekomendowana do współpracy z Formik. Yup umożliwia definiowanie schematów walidacji w sposób deklaratywny, co zwiększa czytelność kodu. Na przykład, w formularzu rejestracji, walidacja pola password obejmuje sprawdzenie minimalnej długości, obecności wielkich i małych liter, cyfr oraz znaków specjalnych. Mechanizm walidacji jest zintegrowany z Formik, co pozwala na automatyczne wyświetlanie komunikatów o błędach w interfejsie użytkownika.

Formik wspiera dynamiczne zmiany w formularzach, co pozwala na łatwe dostosowanie formularzy do zmieniających się wymagań. Na przykład, w formularzu ustawień konta, wartości pól są dynamicznie aktualizowane na podstawie danych użytkownika pobranych z API. Obsługa błędów użytkownika jest intuicyjna – Formik automatycznie oznacza pola z błędami i wyświetla odpowiednie komunikaty. Dodatkowo, błędy są obsługiwane w sposób asynchroniczny, co pozwala na walidację po stronie serwera i informowanie użytkownika o problemach, takich jak nieprawidłowe dane logowania.

Podsumowując, obsługa formularzy w projekcie jest dobrze zorganizowana dzięki wykorzystaniu Formik i Yup. Proces definiowania formularzy, walidacji danych oraz obsługi błędów jest intuicyjny i efektywny, co pozwala na szybkie tworzenie i zarządzanie formularzami w aplikacji.

15.2. Angular

W projekcie opartym na Angularze obsługa formularzy została zrealizowana w sposób niestandardowy – bez wykorzystania Reactive Forms czy Template-driven Forms w klasycznym rozumieniu (np. `FormGroup`, `Validators.required`, itp.). Zamiast tego, formularze i logika walidacji zostały w pełni zaimplementowane ręcznie, przy użyciu własnych serwisów oraz prostych powiązań danych (`[(ngModel)]`) w komponentach.

Dane wprowadzane przez użytkownika są wiązane bezpośrednio z polami obiektu przechowywanego w serwisie walidacyjnym (np. `loginValidationService.user.login`). Logika sprawdzająca poprawność danych (np. poprawny login lub hasło) również znajduje się w dedykowanym serwisie, który przechowuje flagi (`isLoginValid`, `isPasswordValid`) kontrolujące komunikaty walidacyjne wyświetlane użytkownikowi. Dzięki temu walidacja jest w pełni kontrolowana i dostosowana do potrzeb projektu, bez konieczności używania gotowych mechanizmów Angulara.

Takie podejście pozwala na dużą elastyczność – logika walidacyjna może być łatwo modyfikowana i rozszerzana, a serwisy mogą być ponownie używane w innych komponentach. Komunikaty o błędach są wyświetlane w zależności od stanu pól kontrolnych, co umożliwia

precyzyjne zarządzanie interfejsem użytkownika. Przykładowo, jeżeli `isPasswordValid === false`, odpowiedni komunikat zostaje pokazany pod polem hasła.

Zastosowane rozwiązanie dobrze wpisuje się w architekturę Angulara opartą o Dependency Injection, co ułatwia zarządzanie stanem i oddziela logikę biznesową od komponentu widoku. Mimo że podejście to odbiega od standardowej implementacji formularzy w Angularze, sprawdza się dobrze w przypadku prostych lub średnio złożonych scenariuszy, gdzie potrzeba pełnej kontroli nad przebiegiem walidacji i strukturą danych.

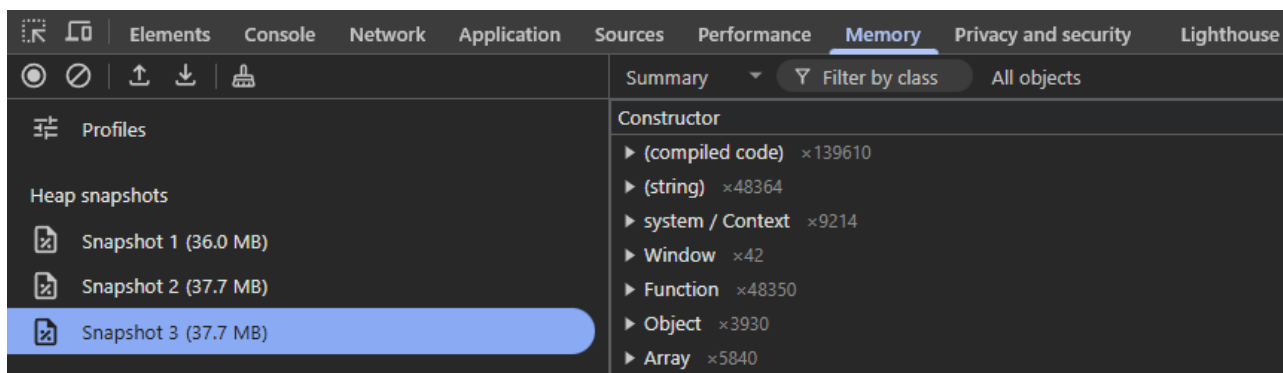
Podsumowując, w analizowanym projekcie Angular wykorzystano własnoręcznie stworzoną logikę walidacyjną i obsługę formularzy, opartą o `ngModel` i dedykowane serwisy. Takie podejście zapewnia pełną kontrolę nad interakcją z użytkownikiem, umożliwia łatwą modyfikację walidacji i dobrze wspiera rozdzielenie logiki od warstwy prezentacji.

16. Analiza wykorzystania CPU, GPU oraz zasobów dysku.

16.1. Next.js

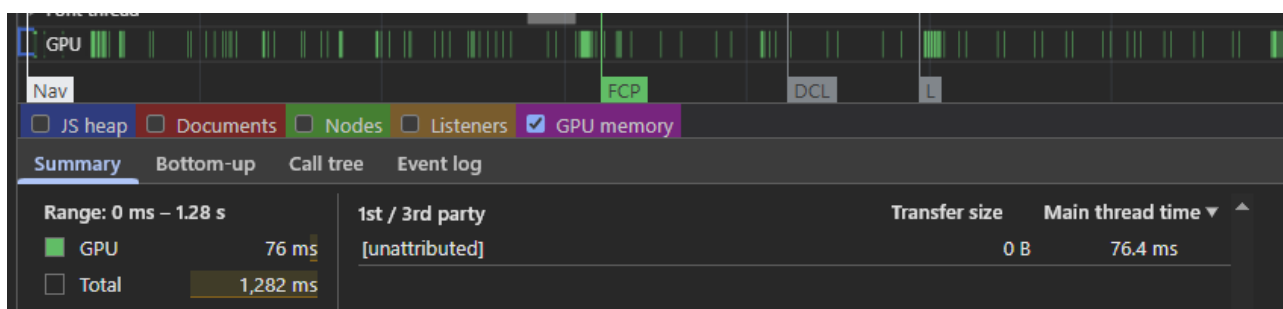
16.2. Angular

W celu zmierzenia wydajności strony pod kątem wykorzystania dysku wykorzystano zakładkę Memory w przeglądarce Chrome. Wykonano 3 pomiary, z których wynika, że stos JavaScript zajmuje średnio 37 MB. Niewielkie rozmiary danych w pamięci pokazują, że aplikacja Angular ma minimalne zapotrzebowanie na operacje dyskowe, działając głównie na danych już załadowanych do pamięci podręcznej przeglądarki.



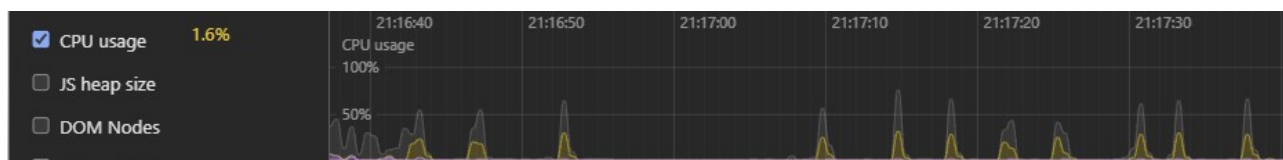
Rysunek 31: Wykorzystanie pamięci RAM przez aplikację utworzoną we frameworku Angular.

Poniższe zdjęcie z zakładki Performance w konsoli Chrome ujawnia, że aplikacja Angular wykorzystuje GPU przez 76 ms w analizowanym okresie, co sugeruje minimalne zaangażowanie procesora graficznego w operacje aplikacji.



Rysunek 32: Wykorzystanie GPU przez aplikację utworzoną we frameworku Angular.

Zdjęcie z monitora CPU usage pokazuje, że wykorzystanie CPU podczas przewijania galerii zdjęć wynosi 1,6%, z krótkimi skokami widocznymi na wykresie określonym przedziale czasowym. Niskie obciążenie procesora wskazuje, że aplikacja Angular nie wykonuje intensywnych obliczeń, co jest typowe dla aplikacji webowych zoptymalizowanych pod kątem interfejsu użytkownika.



Rysunek 33: Wykorzystanie CPU przez aplikację utworzoną we frameworku Angular.