Streetview

Dokumentacja projektowa PZSP2

Wersja 0.0

24.04.23

Semestr 23L

Zespół nr 8 w składzie:

Franek Kozioł

Andrii Demydenko

Minh Nguyen Con

Mentor zespołu: Krystian Radlak

Spis treści

[1 Wprowadzenie 1](#_Toc126861746)

[1.1 Cel projektu 1](#_Toc126861747)

[1.2 Wstępna wizja projektu 1](#_Toc126861748)

[2 Analiza wymagań 1](#_Toc126861749)

[2.1 Wymagania użytkownika i biznesowe 1](#_Toc126861750)

[2.2 Wymagania funkcjonalne i niefunkcjonalne 1](#_Toc126861751)

[2.3 Przypadki użycia 1](#_Toc126861752)

[3 Definicja architektury 1](#_Toc126861753)

[4 Dane trwałe 2](#_Toc126861754)

[4.1 Model logiczny danych 2](#_Toc126861755)

[4.2 Przetwarzanie i przechowywanie danych 2](#_Toc126861756)

[5 Specyfikacja analityczna i projektowa 2](#_Toc126861757)

[6 Projekt standardu interfejsu użytkownika 2](#_Toc126861758)

[7 Specyfikacja testów 2](#_Toc126861759)

[*8* *Wirtualizacja/konteneryzacja* 2](#_Toc126861760)

[9 Bezpieczeństwo 2](#_Toc126861761)

[10 Podręcznik użytkownika 2](#_Toc126861762)

[11 Podręcznik administratora 2](#_Toc126861763)

[12 Podsumowanie 3](#_Toc126861764)

[13 Bibliografia 3](#_Toc126861765)

# Wprowadzenie

## Cel projektu

Celem projektu jest wytworzenie wolnego i otwartego oprogramowania umożliwiającego przeglądanie zbioru zdjęć sferycznych w przeglądarce internetowej z uwzględnieniem urządzeń o niskiej mocy obliczeniowej, urządzeń mobilnych, urządzeń z ograniczonym dostępem do internetu i przeglądarek bez wsparcia dla JavaScript.

## Wstępna wizja projektu

Jako, że projekt ma nie korzystać z javascript, do „obracania się”, będą wykorzystywane przyciski: góra, dół, prawo, lewo. Kliknięcie przycisku spowoduje wyrenderowanie obrazka lekko obróconego w pożądaną stronę.

# Analiza wymagań

## Wymagania użytkownika i biznesowe

Wymagania biznesowe:

* Aplikacja ma być open source

Wymagania użytkownika:

* Aplikacja ma być szybka
* Aplikacja ma być niezawodna

Wymagania systemowe:

* Aplikacja ma działać na urządzeniach nie wspierających javascript
* Aplikacja ma działać na urządzeniach o niskiej mocy obliczeniowej

## Wymagania funkcjonalne i niefunkcjonalne

Wymagania funkcjonalne:

* Aplikacja umożliwia oglądanie zaimportowanych zdjęć, przy użyciu przycisków pozwalających się obracać.
* Aplikacja umożliwia poruszanie się po mapie stworzonej z wielu zdjęć.

Wymagania niefunkcjonalne:

* Aplikacja funkcjonuje na urządzeniach nie wspierających javasript.
* Aplikacja funkcjonuje na urządzeniach o niskiej mocy obliczeniowej, takich jak telefony czy tablety.

## Przypadki użycia

Interfejs aplikacji jest na tyle prosty, że nie ma potrzeby na przypadki użycia.

# Definicja architektury

Diagram architektury:

Diagram

Description automatically generated

# Dane trwałe

Obrazy będą przechowywane na komputerze host, a podczas wdrażania kontenera dockera, katalog ten jest montowany w kontenerze, wraz z plikiem mapy w formacie json.

# Specyfikacja analityczna i projektowa

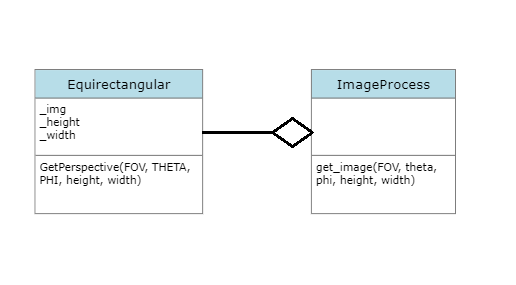
Odnośnik do repozytorium: https://github.com/congminh1254/360-webview-nojs

Aplikacja zbudowana przy użyciu Pythona i Flask. Biblioteka OpenCV używana do renderowania zdjęć.

Wykorzystana została architektura mikroserwisów z mikroserwisem Flask odpowiedzialnym za renderowanie zdjęć i zwracanie ich do klienta

Zdjęcia przechowywane jako pliki w systemie plików. Wygenerowana mapa zapisana w formacie JSON. Aplikacja uzyskuje dostęp do plików i map w celu renderowania zdjęć po stronie serwera i zwracania ich klientowi.

Diagram klas:

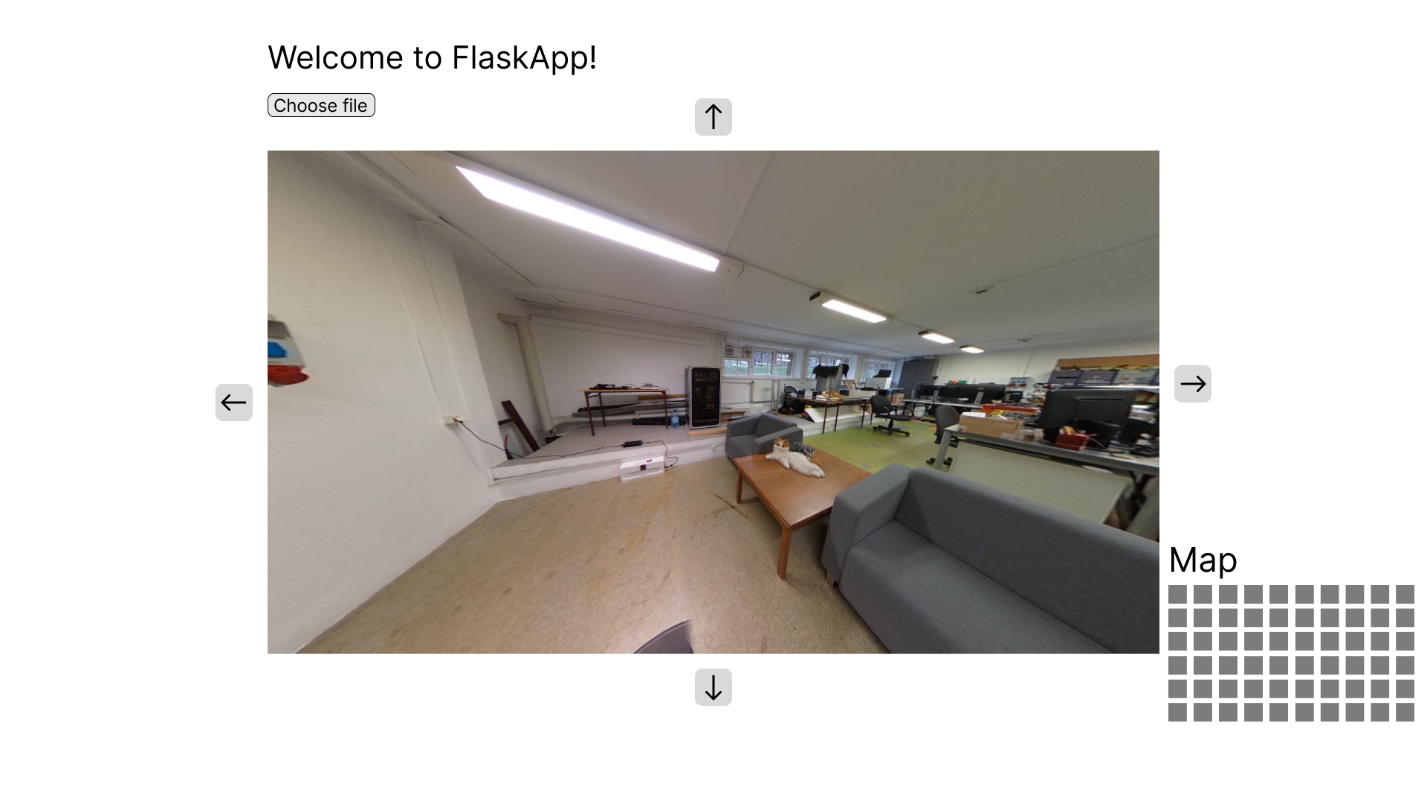


Statystyki:

* Liczba plików: 13
* Liczba linii kodu: 192
* Liczba testów jednostkowych: 0

# Projekt standardu interfejsu użytkownika

Przy użyciu programu Figma, stworzyliśmy mockup ekranu aplikacji:



Interfejs jest na tyle prosty, że nie ma potrzeby użycia wireframes, wireflows itd.

# Specyfikacja testów

Przeprowadzimy testy jednostkowe, które weryfikują:

* czy mikrousługa Flask może otrzymać od klienta żądanie renderowania zdjęcia i zwrócić wyrenderowane zdjęcie klientowi przy użyciu OpenCV.
* czy OpenCV może poprawnie renderować zdjęcie z określonym punktem widzenia.
* czy mikrousługa Flask może poprawnie obsługiwać błędy i zwracać odpowiednie komunikaty o błędach do klienta.
* czy mikrousługa Flask może obsługiwać żądania od wielu klientów jednocześnie bez zakłóceń.

# *Wirtualizacja/konteneryzacja*

Docker i kontenery będą używane w tym projekcie do łatwego wdrażania i zarządzania aplikacją w różnych środowiskach. Aplikacja zostanie umieszczona w kontenerze i spakowana w obraz Dockera, który można łatwo wdrożyć na dowolnym komputerze hosta z zainstalowanym Dockerem. Kontener będzie miał preinstalowane wszystkie niezbędne zależności i konfiguracje, co ułatwi uruchamianie aplikacji bez obaw o problemy ze zgodnością. Takie podejście zapewni również izolację i bezpieczeństwo, ponieważ kontener działa niezależnie od komputera hosta i innych kontenerów.

# Bezpieczeństwo

Cała komunikacja między klientem a korzysta z protokołu HTTPS, który szyfruje przesyłane dane. Zapobiega to podsłuchiwaniu i manipulowaniu danymi.

# Podręcznik użytkownika

Użytkownik może obracać widok używając przycisków ze strzałkami. Aby zmienić zdjęcie/lokacje w której się znajduje, należy kliknąć na jeden z przycisków reprezentujących mapę.

# Podręcznik administratora

Instrukcja dla administratora, jak zainstalować projekt:

1. Sklonuj repozytorium z https://github.com/congminh1254/360-webview-nojs na lokalną maszynę.

2. Pobierz i rozpakuj zestaw danych zdjęć 360 stopni na swoim komputerze głównym.

3. Otwórz terminal i przejdź do katalogu głównego sklonowanego repozytorium.

4. Aby wyczyścić poprzedni kontener Docker, uruchom następujące polecenie:

```

docker stop 360-webview && docker rm 360-webview

```

5. Aby skompilować bieżącą wersję projektu do obrazu Dockera, uruchom następującą komendę:

```

docker build . -t 360-webview

```

6. Aby wdrożyć kontener Docker z zamontowanym katalogiem zestawu danych w /python-docker/data, uruchom następującą komendę:

```

docker run -d -it --name 360-webview --restart always --mount type=bind,source=$HOME/data\_folder,target=/python-docker/data -p 9001:9001 360-webview

```

To polecenie uruchomi kontener Docker o nazwie `360-webview`, który automatycznie uruchomi się ponownie po ponownym uruchomieniu systemu (`--restart always`). Zamontuje również katalog zestawu danych znajdujący się w `$HOME/data\_folder` w katalogu `/python-docker/data` kontenera, w którym aplikacja będzie szukać zdjęć 360 stopni. Na koniec zmapuje port 9001 kontenera na port 9001 hosta (`-p 9001:9001`), w którym aplikacja będzie dostępna z przeglądarki internetowej.

7. Opcjonalnie: aby skonfigurować Nginx do ruchu proxy z Internetu do portu 9001, wykonaj następujące kroki:

- Zainstaluj Nginx, jeśli nie jest jeszcze zainstalowany w twoim systemie.

- Otwórz plik konfiguracyjny Nginx, zwykle znajdujący się w `/etc/nginx/nginx.conf`, za pomocą edytora tekstu.

- Dodaj następujący blok kodu w bloku `http`:

```

server {

listen 80;

server\_name example.com;

location / {

proxy\_pass http://localhost:9001;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

}

```

- Zastąp `example.com` nazwą swojej domeny lub adresem IP.

- Zapisz i zamknij plik, a następnie uruchom ponownie Nginx za pomocą następującego polecenia:

```

sudo systemctl restart nginx

```

- Teraz powinieneś mieć dostęp do aplikacji z przeglądarki internetowej, przechodząc do „http://example.com” lub swojego adresu IP.

# Podsumowanie

Mocne strony:

* Open source
* Małe wymagania sprzętowe
* Nie potrzebuje javascript
* Proste w obsłudze

Słabe strony:

* Proste, nieimponujące UI
* Brak możliwości „przeciągania” myszką, aby się obracać
* Brak możliwości „chodzenia” (tylko klikanie na mapę)

# Bibliografia

OpenCV: https://docs.opencv.org/

Flask: https://flask.palletsprojects.com/en/2.1.x/

Docker: https://docs.docker.com/