**Kolegium Nauk Przyrodniczych  
Uniwersytet Rzeszowski**

**Przedmiot:**

**Sieci komputerowe**

**Projekt - Implementacja serwera WWW**

**Wykonał:**

**Tomasz Nowak, 131478**

**Prowadzący: Mgr inż. Jarosław Szkoła**

**Rzeszów 2024**

## Temat projektu:

Projekt polega na stworzeniu serwera WWW, który obsługuje wiele użytkowników jednocześnie. Serwer udostępnia wyłącznie pliki statyczne w formacie .txt oraz .html. Dodatkowo, serwer implementuje prosty endpoint API, który zwraca dane w formacie JSON. Serwer posiada plik konfiguracyjny który determinuje pliki które chcemy udostępniać dla osób trzecich.

## Wprowadzenie:

Do realizacji założeń zastosowano:

* Język programowania Golang, który wyróżnia się swoją prostotą, wydajnością i nowoczesnym podejściem do tworzenia aplikacji.
* Wykorzystano Dockera oraz Docker-compose do bardzo łatwego i prostego przenoszenia projektu na inne serwery, czy komputery. Każdy system który będzie potrafił zainstalować dockera, uruchomi również naszą aplikację, uwzględniając wszystkie paczki, wszystkie ‘dodatki’ którymi uraczymy nasz projekt. Jedynym wymogiem jest umieszczenie poprawnych skryptów w Dockerfile (pliku tworzącym nasze środowisko).
* Docker-compose użyto do konteneryzacji naszej aplikacji. Dzięki kontenerom możemy tworzyć wiele niezależnych od siebie aplikacji które o sobie mogą wiedzieć, bądź nie. Wszystko możemy ustawić w pliku docker-compose.yml.
* W języku programowania Golang użyto następujących paczek:
  + “net/http”- <https://pkg.go.dev/net/http> - paczka do obsługi serwera. W tej paczce mamy strukturę (nie obiekt, nie klasę, w golangu posiadamy struktury) http która posiada funkcję FileServer (opiszę w dalszej części dokumentacji).
  + “encoding/json” - paczka w golangu do przetwarzania naszych struktur do plików typu JSON. Używana jest tylko i wyłącznie pod endpointem ‘/api’ w naszej przeglądarce, nie jest to wymóg projektu, zrobiłem to z czystej ciekawości tego języka.
* Do obsługi Hot-reload kodu jak i całej stworzonej aplikacji Golang, użyłem paczki ‘air’ dostępnej pod adresem: <https://github.com/air-verse/air>
  + Użyłem jej dla posiadania hot-reloadu aplikacji, gdy wprowadzam zmiany w plikach statycznych, czy bądź też zmiany w logice aplikacji, zmiany automatycznie są zapisywane, a cały serwer restartowany. Niesamowicie przyspiesza to proces tworzenia naszej aplikacji, jak i plików statycznych. Nie musimy dzięki temu wyłączać naszego dockerowego środowiska, i ponownie go włączać. Jedyna wada a właściwie specyfika tej paczki jest taka, że tylko zmiany w naszej aplikacji będą restartować aplikację - czyli gdy zmienimi coś w pliku docker-compose.yml, dalej będziemy musieli wyłączyć i włączyć środowisko z aplikacją.
* Repozytorium GitHub z aktualnym kodem projektu jest dostępne pod adresem <https://github.com/Tnovyloo/http-go-server>

## Elementy kluczowe oraz opis aplikacji

Struktura projektu:

* main.go: Główny plik projektu, który uruchamia serwer, obsługuje statyczne zasoby z katalogu static oraz definiuje endpoint /api.
* api/handler.go: Prosty handler, który obsługuje zapytania pod endpointem localhost/api i zwraca dane w formacie JSON (stworzony z czystej ciekawości).
* docker-compose.yml: Konfiguracja Docker Compose, pozwalająca uruchomić serwer w kontenerze oraz udostępniająca porty.
* dockerfile: Konfiguracja Dockera zawierająca ustawienie środowiska Go, narzędzia Air oraz hot-reloadingu.
* /server/static - katalog odpowiedzialny za przetrzymywanie wszystkich naszych plików statycznych.
* config.json - plik odpowiedzialny za konfigurację serwera. Od jego ustawienia możemy zarządzać udostępnianą treścią.

Jakie funkcje implementuję plik ‘main.go’ - plik odpowiedzialny za nasz serwer:

main.go implementuję paczkę http oraz jej funkcję FileServer, która posiada dokumentację pod linkiem: <https://pkg.go.dev/net/http#FileServer>

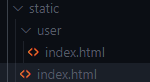
Aby funkcja działała prawidłowo należy podać katalog plików statycznych do funkcji http.FileServer(http.Dir(“scieżka do plików”)). W ten sposób nasza aplikacja w Go implementuje serwer dla plików statycznych. Aby obsługiwać ten serwer za pomocą ‘scieżek’ (np http:localhost/index”) należy go umieścić do funkcji http.Handle(tutaj podajemy nasz http.FileServer).

Dodatkowo została zrobiona funkcjonalność blokowania pobierania plików z danym rozszerzeniem z naszego statycznego katalogu. Aby blokować bądź udzielać dostęp do pobierania plików należy zmienić konfigurację naszego serwera. Konfiguracja zawiera się w pliku ‘config.json’ gdzie posiadamy zmienne do jego konfiguracji. Mianowicie “allowed\_extensions”, oraz “denied\_extensions”. W tych listach należy umieścić interesujące nas rozszerzenia plików. Np. Jeśli chcemy aby serwer nie obsługiwał plików typu ISO, należy umieścić “.ISO” w liście “denied\_extensions”, i na odwrót.

Dodałem również prosty endpoint który zwraca dane JSON dostępny pod ścieżką “localhost:80/api” do funkcji obsługującej takie ‘Handlery’ czyli http.HandleFunc. Aby prawidłowo napisać funkcję obsługującą należy do niej przyjąć strukturę żądania użytkownika, oraz strukturę ‘ResponseWriter’.

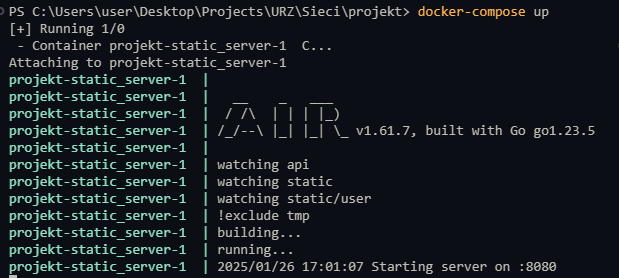
Następnie należy włączyć funkcję która zacznie nasłuchiwać naszych żądań z przeglądarki na dany port. (W aplikacji Go ustawiłem 8080, ale w pliku docker-compose przekierowuję port 80 - czyli port hosta, na port konteneru 8080 - to tak w woli niuansu)

Aby widzieć nasz plik statyczny pod daną ścieżką w przeglądarce, należy go nazwać index.html. Tak wygląda konwencja nazywania plików:

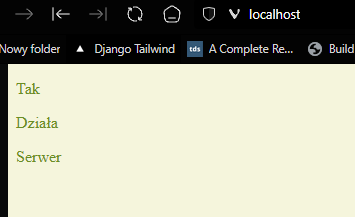


## Działanie aplikacji:

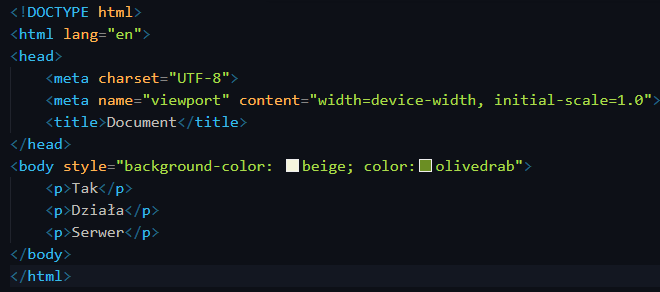
Po wpisaniu docker-compose up –build w katalogu projektu (nie w katalogu /server !!) nasz serwer będzie dostępny pod linkiem localhost:80, będziemy widzieć nasz plik index.html z folderu /static/. Poniżej przedstawiono działanie komendy uruchamiającej nasz serwer czyli: docker-compose up w terminalu:



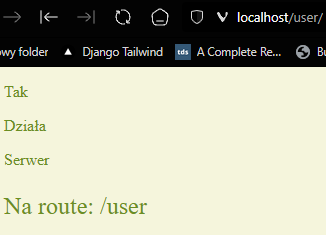
Plik index.html z folderu static.



Plik statyczny dostępny w folderze static wygląda następująco:



Aby zobaczyć plik statyczny static/user/index.html należy przejść w przeglądarce do ‘localhost:80/user’



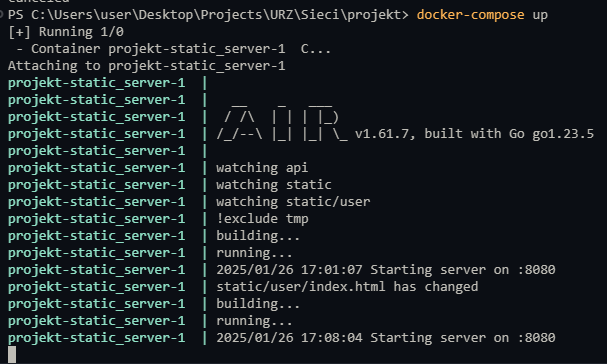
Plik user/index.html wygląda następująco:



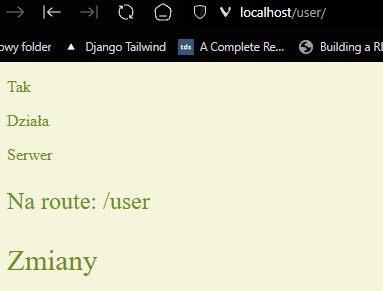
Po zmianach w pliku /static/user/index.html na:



Nasz serwer automatycznie się zrestartuje co będzie widać w logach docker-compose:



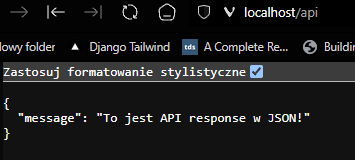
Po odświeżeniu przeglądarki treść pliku statycznego będzie zaktualizowana (bez konieczności manualnego restartu dockera, czy też aplikacji golang):



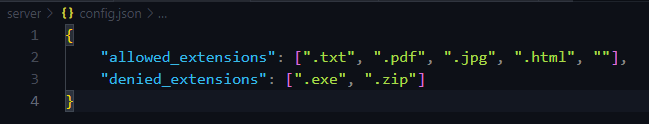
Aby zobaczyć co serwuje nasz Handler JSON należy przejść do ścieżki localhost:80/api, Poniżej przedstawiam nasz Handler do ścieżki /api:



Jak widzimy zwraca nam tylko strukturę przedstawioną w JSON. Teraz przedstawię jak wygląda to w przeglądarce:



Poniżej przedstawiono konfigurację naszego serwera który pozwala na dostęp do plików z danym rozszerzeniem. Konfiguracja serwera jest następująca:

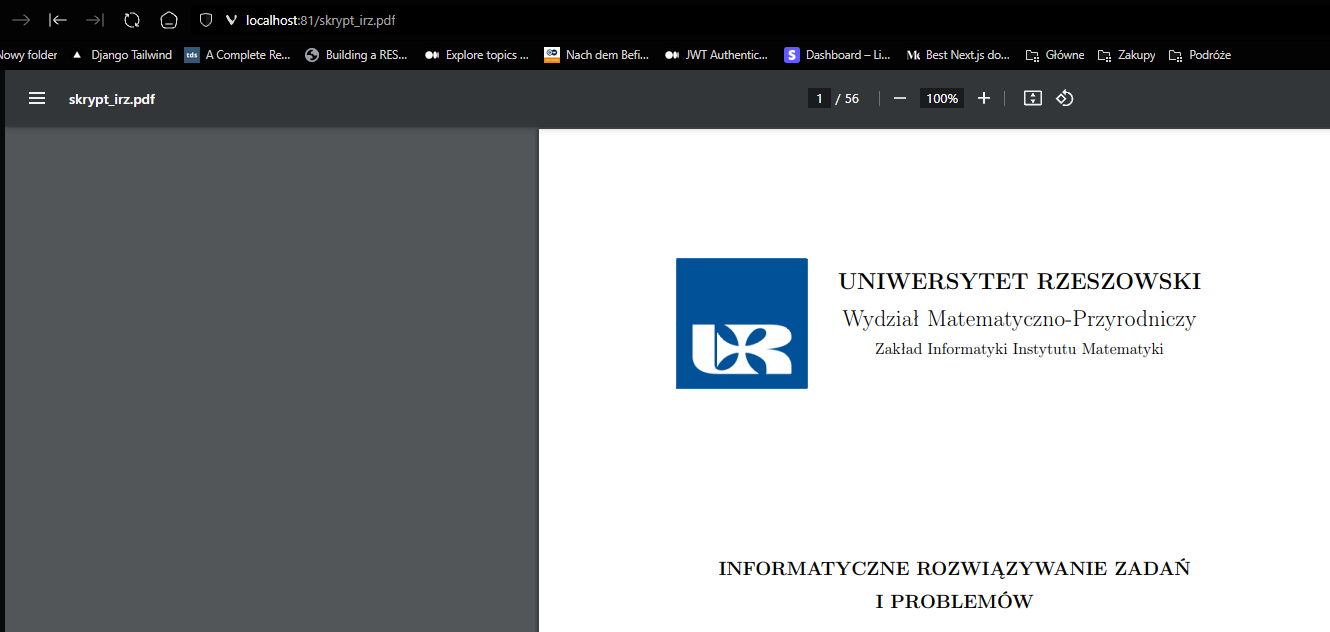


Mianowicie zezwalamy serwerowi udostępniać pliki z katalogu static, o rozszerzeniach “.txt”, “.pdf”, “.jpg”, “.html”, “”. I zabraniamy udostępniać “.exe”, “.zip”

Struktura mojego katalogu statycznego wygląda następująco (dodałem kilka przykładowych plików z różnymi rozszerzeniamy aby pokazać działanie konfiguracji serwera)



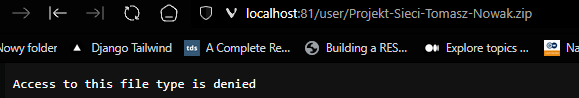
Następnie sprawdzamy ścieżkę: localhost/skrypt\_irz.pdf



Jak widzimy powyżej, uzyskaliśmy dostęp do pliku typu: PDF, poniżej przedstawiam logi serwera dla żądania pobrania pliku PDF:



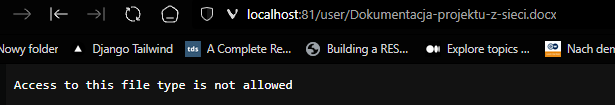
Teraz możemy sprawdzić czy nasz serwer obsługuje pliki które mają być zablokowane do pobrania:



Poniżej logi serwera:



Następnie sprawdzimy co z plikami których nie określiliśmy w pliku konfiguracyjnym (Będzie to plik o rozszerzeniu .docx):



Jak widzimy plik jest niedostępny do pobrania (pomimo tego, że fizycznie jest w pliku statycznym). Poniżej logi serwera:



Jak widzimy rozszerzenie .docx nie jest obsługiwane, serwer pokazuje nam rozszerzenia które są dostępne dla użytkowników - w nich nie ma .docx więc serwer zwraca nam błąd.

## Wnioski oraz podsumowanie

1. Efektywność i prostota: Wykorzystanie języka Go pozwoliło na szybkie stworzenie wydajnego serwera WWW obsługującego wiele połączeń jednocześnie.
2. Docker jako narzędzie rozwojowe: Dzięki konteneryzacji środowisko serwerowe jest łatwe do uruchomienia i modyfikacji, co zwiększa mobilność projektu.
3. Przystosowanie do developmentu: Implementacja hot-reloadingu za pomocą paczki Air usprawnia proces testowania zmian w kodzie oraz zmian w konfiguracji serwera.
4. Przystosowanie pliku konfiguracyjnego serwera. Dzięki niemu możemy z łatwością zarządzać typami plików które chcemy udostępniać użytkownikom, bądź też je blokować. Wszystko to za pomocą pliku config.json oraz funkcji FileExtensionMiddleWare (aby zobaczyć tą funkcję należy wejść do pliku main.go)