

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

Projekt dyplomowy

*System zarządzania finansami oraz oszczędnościami, tworzący prognozy na podstawie danych statystycznych.*

*Finance and savings management system with forecasting based on statistic data.*

Autor: *Marcin Głodziak*

Kierunek studiów: Informatyka i Systemy Inteligentne

Opiekun pracy: *dr inż. Grzegorz Rogus*

Kraków, 2024

Spis treści

[1. Wprowadzenie 4](#_Toc154758955)

[2. Architektura systemu 5](#_Toc154758956)

[2.1. Wzorzec architektoniczny 5](#_Toc154758957)

[2.2. Warstwa wizualna (front-end) 5](#_Toc154758958)

[2.2.1. Budowa części wizualizacyjnej 5](#_Toc154758959)

[2.2.2. Wykorzystanie serwisów 5](#_Toc154758960)

[2.2.3. Widok dla niezalogowanych użytkowników 5](#_Toc154758961)

[2.2.4. Widok dla zalogowanych użytkowników 5](#_Toc154758962)

[2.3 Warstwa serwerowa (back-end) 7](#_Toc154758963)

[2.4 Baza danych 7](#_Toc154758964)

[3. Moduły oraz funkcje systemu 7](#_Toc154758965)

[3.1 Rejestr wpływów 7](#_Toc154758966)

[3.2 Rejestr wydatków 7](#_Toc154758967)

[3.3 Łączenie użytkowników 7](#_Toc154758968)

[3.4 Analiza opcji oszczędnościowych 7](#_Toc154758969)

[3.5 Prognozy 7](#_Toc154758970)

[4. Możliwości dalszego rozwoju systemu 7](#_Toc154758971)

[4.1 Rozwój istniejących funkcji systemu 7](#_Toc154758972)

[4.2 Dodanie nowych funkcjonalności 7](#_Toc154758973)

[5. Podsumowanie i wnioski 7](#_Toc154758974)

[6. Wykaz literatury 7](#_Toc154758975)

[7. Załączniki 8](#_Toc154758976)

# 1. Wprowadzenie

Egzystencja w dzisiejszym świecie nieodłącznie wiąże się z zarządzaniem finansami. Zewsząd otaczają nas informacje z zakresu ekonomii, a zmiany w jednej części globu są w stanie w zauważalny sposób oddziaływać na drugą, np. powodując zmiany poziomu cen w Polsce. Mimo że jako osoby fizyczne jesteśmy wyłącznie niewielkim ogniwem łańcucha światowej gospodarki, każda jednostka jest w stanie w pewnym stopniu wpływać na swoje otoczenie i podejmować korzystne dla siebie decyzje przy zmieniających się warunkach makroekonomicznych.

Jednym z istotnych czynników wspomagających podejmowanie korzystnych decyzji finansowych jest świadomość na temat posiadanego kapitału oraz możliwości jego wykorzystania. Elementy planowania budżetu domowego czy monitorowania wpływów i wydatków są w stanie dostarczyć rzetelnych danych w zakresie efektywności tegoż budżetu oraz usprawnić dystrybucję środków. Wym­­­­iernym efektem tego podejścia może być zmiana nawyków związanych z wydatkami, a co za tym idzie – powiększenie oszczędności.

Głównym celem planowania budżetu nie powinno być jednak generowanie oszczędności *sensu stricto*. Same oszczędności nie stanowią o dobrostanie jednostki, czego przykładem może być obecna sytuacja, kiedy inflacja rok po roku trawi sporą część niepracującego na siebie kapitału. Sto złotych sprzed kilku lat nie pozwoli dziś kupić takiej samej ilości dóbr jak obecnie. Z tego powodu przy planowaniu budżetu warto poświęcić też chwilę na zastanowienie się nad możliwościami zainwestowania posiadanych środków.

Jak zostało wspomniane powyżej, nasze decyzje finansowe nie są jednak niezależne od sytuacji na świecie. Z tego powodu nawet najbardziej optymalne kroki podejmowane bez choćby pobieżnej analizy mogą okazać się nietrafione w dłuższym ujęciu czasowym. Symulacje oraz prognozy na podstawie danych statystycznych są w stanie dostarczyć danych choćby na temat istotnych czynników ekonomicznych, takich jak wskaźnik CPI czy kursy walut. Dzięki temu decyzje jednostki mają szansę stać się bardziej świadome, a przez to korzystne.

Celem niniejszej pracy jest przygotowanie systemu informatycznego wspierającego zarządzanie oszczędnościami przez osoby fizyczne. Dzięki odpowiednim modułom aplikacja będzie w stanie wspomagać planowanie budżetu oraz monitorowanie wpływów i wydatków, czego wymiernym efektem powinna być wyższa świadomość użytkownika w zakresie przepływu środków na koncie. System oferuje również moduł pozwalający na ocenę dostępnych opcji ulokowania kapitału oraz obliczenia przewidywanej stopy zwrotu, co pomoże użytkownikowi walczyć z inflacją oraz ułatwi wykorzystanie już posiadanego kapitału do jego pomnażania. Przygotowano również moduł odpowiedzialny za wykonywanie prostych prognoz na podstawie danych statystycznych, dzięki czemu użytkownik będzie posiadał większą świadomość na temat zmienności światowego oraz polskiego rynku, a także przewidywanych kierunków zmian.

Praca jest odpowiedzią na problemy typowe dla młodych ludzi, wkraczających w dorosłość. Stanowi też kompleksowe rozwiązanie, którego obecnie brak jest na ogólnodostępnym rynku.

# 2. Architektura systemu

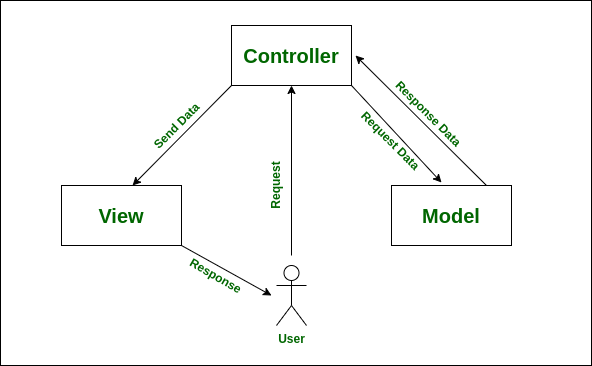
System został przygotowany w modelu aplikacji webowej. Wybór ten dyktuje głównie natura projektu – wymaganie interakcji z istniejącymi systemami, dostępność z dowolnego urządzenia czy możliwość łączenia użytkowników w większe całostki. Poniżej opisano szczegóły oraz technologie użyte do wykonania kolejnych części architektury systemu.

## 2.1. Wzorzec architektoniczny

Architektura systemu bazuje na rozpowszechnionym wzorcu Model-Widok-Kontroler (Model-View-Controller, MVC). Komunikacja między warstwą widoku oraz kontrolerami wykorzystuje REST API. Aby umożliwić wykonywanie określonych operacji wyłącznie uprawnionym użytkownikom, zastosowano mechanizm uwierzytelniania oparty o JWT. Wyjaśnienia poszczególnych terminów oraz szczegóły rozwiązań technicznych znajdują się w kolejnych podrozdziałach.

### 2.1.1. Architektura MVC

Prezentowany system został przygotowany w oparciu o wzorzec architektoniczny MVC. Jest to bardzo wygodna architektura, ponieważ pozwala na rozdzielenie odpowiedzialności pomiędzy poszczególnymi częściami aplikacji. Schemat współpracy poszczególnych elementów przedstawia rysunek 2.1.1.



**Rys. 2.1.1.** Schemat wzorca MVC[[1]](#footnote-1).

Najłatwiej prześledzić proces interakcji z systemem rozpoczynając od użytkownika. W momencie wykonania przez niego określonego działania wysłane zostaje zapytanie HTTP do odpowiedniego kontrolera. W tym momencie część serwerowa aplikacji (back-end) rozpoczyna przetwarzanie otrzymanego zapytania HTTP. W wielu przypadkach wymaga ono ingerencji w model aplikacji, które stanowi źródło danych. Model zwraca żądane dane do kontrolera, który następnie po zakończeniu przetwarzania wysyła je w odpowiedzi na otrzymane zapytanie. Dane zostają przesłane do warstwy wizualizacyjnej, dzięki czemu są w stanie wypełnić właściwy widok. Można zatem powiedzieć, że kontrolery przyjmują żądania od użytkownika, przetwarzają je korzystając z danych zgromadzonych w modelu, by następnie zwrócić wyniki na właściwy widok.

Architektura MVC ma wiele zalet, wśród których należy wymienić między innymi separację poszczególnych części składowych. Użytkownik nie ma możliwości bezpośredniej interakcji z modelem, dzięki czemu w dobrze napisanym systemie ryzyko nieautoryzowanego dostępu do danych jest zminimalizowane. Ponadto dzięki rozdzieleniu odpowiedzialności możliwe jest utrzymanie większego porządku w kodzie, co jest jedną z fundamentalnych zasad tworzenia oprogramowania. Czytelność kodu bardzo mocno wpływa na produktywność programisty, a także na skuteczność projektowania całego systemu [1]. Z tego powodu wybór architektury MVC znacznie ułatwił konstruowanie aplikacji.

Literatura:

* zasady MVC
* wady i zalety - rozwinięcie

### 2.1.2. REST API

Część serwerowa aplikacji została stworzona według popularnego zestawu zaleceń REST API. Nie stanowią one ujednoliconego standardu, jednak dzięki szerokiemu rozpowszechnieniu stanowią dobrą praktykę w programowaniu oraz rozpoznawalny styl architektoniczny [2].

REST to akronim od angielskiej nazwy Representational State Transfer. Jest to styl, który może występować razem ze standardami HTTP oraz URI. Dzięki architekturze REST możliwe jest bezproblemowe modyfikowane zgromadzonych w modelu zasobów i wyświetlanie użytkownikowi aktualnego stanu aplikacji. Sam REST jest bezstanowy, podobnie jak protokół HTTP. Oznacza to, że w każde zapytanie wysłane na serwer musi zawierać pełen zestaw informacji niezbędnych do jego obsłużenia. Z jednej strony zwiększa to narzut na niezbędne do przesłania dane (np. wielokrotne przesłanie identyfikatora danego zasobu), jednak oferuje znacznie większe korzyści w zakresie skalowalności czy przejrzystości obsługi zapytań.

W REST obowiązują liczne zalecenia i dobre praktyki, które ułatwiają tworzenie skomplikowanych systemów. Jednym z podstawowych wymagań jest korzystanie z czasowników HTTP, tj. GET, POST, PUT, DELETE. Dzięki nim możliwe jest wysłanie do kontrolerów jasnych zapytań, które po obsłużeniu odpowiednio zwrócą, dodadzą, zmodyfikują lub usuną dane.

Czasowniki HTTP pomagają zrozumieć sens wysyłanych zapytań. Co prawda REST nie warunkuje dokładnej reprezentacji słownej danego zasobu, jednak określa pewne zasady nazewnictwa. Przyjęło się, że dane reprezentowane są przez rzeczowniki. W połączeniu z czasownikami HTTP można zatem łatwo zrozumieć w jaki sposób modyfikujemy dany zasób. Zasada konstrukcji URI, czyli ujednoliconych identyfikatorów zasobów, opiera się na strukturze drzewiastej, w której kolejne węzły od korzenia stanowią specyfikację poprzednika. Niniejszą zależność obrazują przykłady 1-3.

* **Przykład 1.** URI /cars może reprezentować wszystkie samochody, znajdujące się w bazie danych.
* **Przykład 2.** URI/cars/ford może reprezentować wszystkie samochody marki Ford.
* **Przykład 3.** URI /cars/opel/van może reprezentować wszystkie Vany marki Opel.

REST dopuszcza istnieniu wielu poprawnych reprezentacji danego zasobu, dlatego przedstawiony powyżej mechanizm może zostać z łatwością zmieniony. Zobrazowana zależność pomaga jednak zrozumieć sam mechanizm działania zapytań REST – przykładowo, żądanie DELETE wysłane na URI /cars/opel/van powinno usunąć z bazy wszystkie samochody marki Opel o kategorii Van.

W prezentowanym systemie zastosowano zasady REST do konstrukcji interfejsu sieciowego (Application Programming Interface, API). Interfejs sieciowy stanowi udostępniony użytkownikowi zbiór procedur obsługujących nadsyłane żądania, na które następnie generuje i odsyła odpowiedzi [literatura – De Sanctis ASP NET CORE lub co innego]. Na API składa się wiele punktów wejścia (ang. entry points, alternatywnie endpoints), przez które możliwa jest komunikacja na linii widok-kontroler.

API stanowi bardzo wygodny model architektoniczny, ponieważ umożliwia ustrukturyzowaną komunikację w oparciu o języki znacznikowe (np. HTML czy JSON) niekoniecznie jednorodnym systemom. Dzięki API możliwe jest połączenie z systemem zewnętrznym wyłącznie poprzez odpowiednie zapytania HTTP, nie występuje potrzeba ingerencji w kod docelowej aplikacji. API pomaga zatem rozdzielić tworzoną platformę na kilka całkowicie niezależnych od siebie modułów – przekłada się to na mniejszą złożoność poszczególnych części systemu, możliwość wykorzystania wyspecjalizowanych technologii lub zaangażowanie większej liczby osób do równoległej pracy nad danym projektem.

### 2.1.3. Uwierzytelnienie użytkownika

JWT

## 2.2. Warstwa wizualna (front-end)

Wizualizacja systemu została przygotowana na platformie Angular w wersji 16. Zgodnie z literaturą, Angular stanowi framework, a zatem [literatura, cytat]. Dzięki temu programista zyskuje możliwość szybszego przygotowania projektowanego systemu, ponieważ nie musi tworzyć podstawowych funkcji odpowiedzialnych np. za obsługę modułowej budowy systemu. Frameworki takie jak Angular udostępniają również wiele przydatnych rozszerzeń (bibliotek), które znacznie ułatwiają pracę.

Warstwa wizualna projektowanego systemu składa się z dwóch głównych widoków, których projekty zostały pokazane na rysunkach 1 oraz 2.

### 2.2.1. Budowa części wizualizacyjnej

Aplikacja została rozdzielona pomiędzy dwa główne moduły, dostępne odpowiednio dla użytkowników zalogowanych oraz niezalogowanych. Posiadają one fragment wspólny w postaci stopki, która stanowi jedyny globalny element systemu. Zajmuje ona 6% wysokości ekranu i jest umieszczona przy dolnej krawędzi wyświetlacza. Pozostałe miejsce zajmuje aktualnie aktywny widok.

Architektura widoku dla użytkowników niezalogowanych zbudowana jest na utworzonym w tym celu module AuthLayoutModule, dostępnym pod trasą /authentication. Deklaruje on dwa komponenty, LoginComponent oraz RegisterComponent. Adresy URL poszczególnych komponentów określa tablica tras AuthLayoutRoutes – pozostałe adresy zostają przekierowywane do komponentu logowania. Moduł nie posiada ogólnych elementów, w związku z czym zawartość strony określa aktualnie wyświetlany komponent.

Analogiczną budowę posiada drugi z widoków, który został określony w module UserLayoutModule. Jest on jednak bardziej złożony, ponieważ wewnątrz zadeklarowano większą liczbę komponentów, odpowiedzialnych za obsługę poszczególnych funkcji systemu. Dostępne miejsce rozdysponowano pomiędzy statyczny pasek nawigacyjny z lewej strony ekranu, pasek użytkownika w górnej części centralnego obszaru oraz zawartość aktualnie wyświetlanego komponentu. Komponenty są połączone z zakładkami w pasku nawigacyjnym. Wiązania określone są przez tablicę tras UserLayoutRoutes. Pozostałe adresy są przekierowywane do głównego komponentu tego widoku, którym jest DashboardComponent.

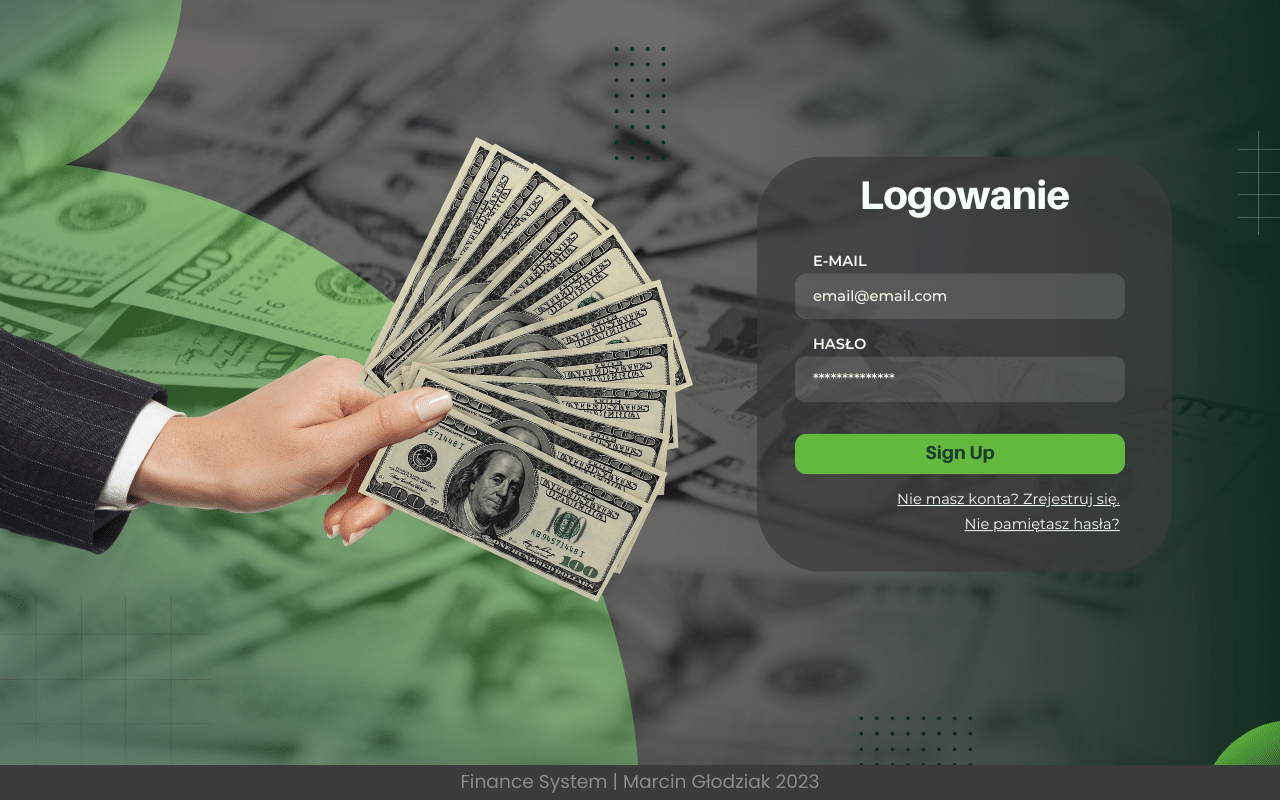
Auth Guardy

### 2.2.2. Wykorzystanie serwisów

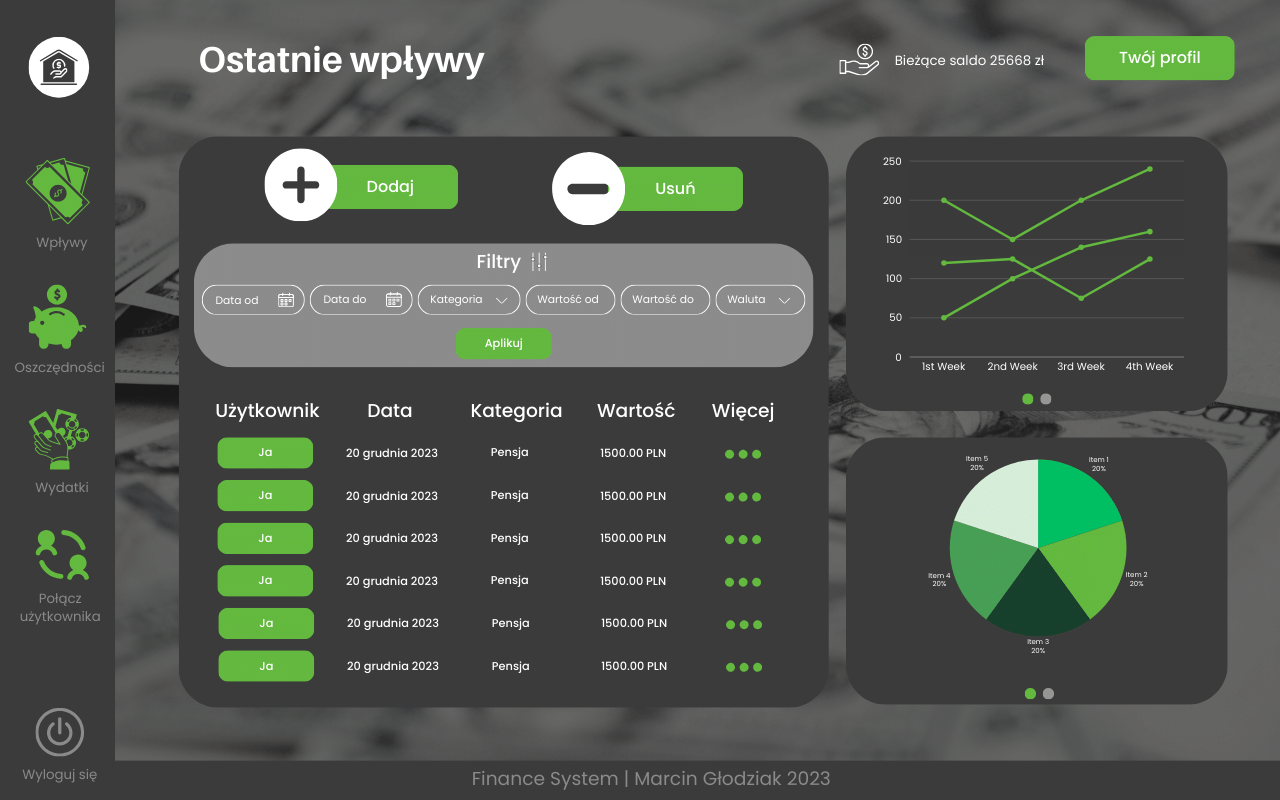
### 2.2.3. Widok dla niezalogowanych użytkowników

Pierwszy z widoków jest dostępny wyłącznie dla niezalogowanych użytkowników. Główną funkcjonalnością dostępną w tej części systemu jest możliwość założenia konta oraz zalogowania się. Po stronie wizualizacji dane wejściowe zbierane są od użytkownika poprzez formularz. Po weryfikacji poprawności danych [dorobić walidację] wysyłane jest zapytanie HTTP do części serwerowej, które może zakończyć się założeniem konta, zalogowaniem lub zwróceniem informacji o błędzie, w zależności od podanych danych oraz wybranej zakładki w początkowym ekranie.

### 2.2.4. Widok dla zalogowanych użytkowników



**Rys 2.1.** Projekt widoku dostępnego dla niezalogowanego użytkownika.



**Rys 2.2.** Projekt jednego z widoków dostępnych dla zalogowanych użytkowników.

## 2.3. Warstwa serwerowa (back-end)

REST API - endpointy

## 2.4. Baza danych

Procedury, CRUD,

# 3. Moduły oraz funkcje systemu

## 3.1. Rejestr wpływów

## 3.2. Rejestr wydatków

## 3.3. Łączenie użytkowników

## 3.4. Analiza opcji oszczędnościowych

## 3.5. Prognozy

# 4. Możliwości dalszego rozwoju systemu

## 4.1. Rozwój istniejących funkcji systemu

## 4.2. Dodanie nowych funkcjonalności

# 5. Podsumowanie i wnioski

# 6. Wykaz literatury

1. R. C. Martin. *Czysty kod: podręcznik dobrego programisty*. Gliwice: Wydawnictwo Helion, 2014.
2. B. Mehta. *REST: najlepsze praktyki i wzorce w języku Java.* Gliwice: Wydawnictwo Helion, 2015.
3. <https://developer.mozilla.org/en-US/docs/Glossary/MVC> (moze)

# 7. Załączniki

* jak ogarnąć odniesienia do rysunków (jak pokazano na rys. 1…?)
* odniesienie do infy – algo zachłanne, kółko i krzyżyk…

1. Źródło obrazu: https://www.geeksforgeeks.org/mvc-design-pattern/ dostęp z 29.12.2023r. [↑](#footnote-ref-1)