A screenshot of a computer

Description automatically generated

**PROJEKT INŻYNIERSKI**

Internetowa platforma crowdfundingowa

**Maciej Maciejewski**

**Nr albumu 290398**

**Kierunek:** Informatyka

**Specjalność:** Grafika komputerowa i oprogramowanie

**PROWADZĄCY PRACĘ**

**Dr inż. Artur Pasierbek**

**KATEDRA Algorytmiki i Oprogramowania**

**Wydział Automatyki, Elektroniki i Informatyki**

**GLIWICE Rok 2023/24**

**Spis treści**

Rozdział 1 Wstęp

* + 1. Wprowadzenie do tematu projektu
    2. Cel pracy
    3. Charakterystyka zawartości rozdziałów

Rozdział 2 Analiza tematu

2.1 Sformułowanie problemu

2.2 Opis istniejących rozwiązań

Rozdział 3 Wymagania i narzędzia

3.1 Wymagania funkcjonalne

3.2 Wymagania niefunkcjonalne

3.3 Przypadki użycia

3.4 Opis narzędzi

3.4.1 Języki programowania

3.4.2 Frameworki

3.4.3 Baza danych

3.4.4 Środowisko i Narzędzia

3.5 Metodyka pracy

Rozdział 4 Specyfikacja zewnętrzna

4.1 Wymagania sprzętowe i instalacja

4.1.1 Instalacja Lokalna

4.1.2 Instalacja z użyciem Dockera

4.2 Sposób uruchomienia

4.2.1 Uruchomienie aplikacji zainstalowanej lokalnie

4.2.2 Uruchomienie aplikacji zainstalowanej na Dockerze

4.3 Realizacja głównych przypadków użycia

4.3.1 Rejestracja i logowanie

4.3.2 Tworzenie nowego projektu jako użytkownik

4.3.3 Wsparcie wybranego projektu

4.3.4 Przeszukanie projektów przez użytkownika

4.4 Administracja serwisu

Rozdział 5 Specyfikacja wewnętrzna

5.1 Architektura serwisu

5.1.1 Architektura po stronie backendu

5.1.2 Architektura po stronie frontendu

5.2 Bezpieczeństwo aplikacji

5.2.1 Json Web Token

5.2.2 Mechanizm ról użytkownika

5.2.3 Mechanizm canActivate w Angular

5.2.4 Obsługa błędu 404

5.3 Schemat bazy danych

5.4 Wykorzystane biblioteki

5.3.1 Backend

5.3.2 Frontend

5.5 Istotne fragmenty kodu

5.4.1 Backend

5.4.2 Frontend

Rozdział 6 Weryfikacja i walidacja poprawności działania

6.1 Sposób testowania

6.2 Zakres testowania

6.3 Przypadki testowe

6.4 Wykryte i usunięte błędy

Rozdział 7 Podsumowanie i wnioski

Bibliografia

Spis skrótów i symboli

Źródła

Lista dodatkowych plików, uzupełniających tekst pracy

Spis rysunków

**1.Wstęp**

**1.1 Wprowadzenie do Tematu Projektu**

W dobie cyfrowej transformacji oraz globalizacji, której początek datuje się na moment upublicznienia Internetu i która nieprzerwanie trwa do dzisiaj, znalezienie ludzi o podobnych zainteresowaniach lub problemach stało się zadaniem niezwykle łatwym. Ta nowa rzeczywistość cyfrowa umożliwia łączenie osób z różnych zakątków świata, tworząc społeczności oparte na wspólnych pasjach, celach czy potrzebach. Niniejsza praca inżynierska koncentruje się na wykorzystaniu tego fenomenu w kontekście internetowej platformy finansowania społecznościowego.

Tradycyjnie inwestorem w wielu projektach jest jeden lub kilka podmiotów które już posiadają znaczący kapitał. Największą wadą takiego modelu finansowania jest to, że inwestycja powinna się zwrócić inwestorom, przez co twórca lub zespół muszą zwracać durzą uwagę na to ile są w stanie zarobić na projekcie, co może prowadzić do zmniejszenia jakości produktu lub agresywnego modelu monetyzacji, przykładowo zablokowanie niektórych elementów przez subskrypcją lub dodatkową opłatą, co potrafi zrazić wiele osób, które na początku były podekscytowane projektem.

Crowdfunding stanowi doskonałą opcję dla tych, którzy większą wagę przykładają do jakość i wartość tworzonego produktu, niż do zysku finansowego. Ten model finansowania społecznościowego nie opiera się na wsparciu pojedynczych inwestorów dysponujących dużym kapitałem, ale na zaangażowaniu szerszej społeczności (ang. Crowd), która wspiera projekt mniejszymi kwotami. Idea ta jest dobrze ujęta w przysłowiu „grosz do grosza, a będzie kokosza”, podkreślając, że małe wkłady mogą w sumie przyczynić się do dużego sukcesu.

Wspierający, przyciągnięci samym pomysłem, nie oczekują zwrotu z inwestycji, lecz angażują się finansowo, gdyż produkt ich autentycznie interesuje. Crowdfunding otwiera drzwi dla innowacyjnych koncepcji, które w tradycyjnych modelach finansowania mogłyby nie uzyskać potrzebnego wsparcia. Sukces takiego przedsięwzięcia zależy od skutecznego dotarcia do wystarczająco dużej liczby zainteresowanych osób.

**1.2 Cel Pracy**

Celem niniejszej pracy jest zaprojektowanie funkcjonalnej i intuicyjnej platformy crowdfundingowej. Platforma ta ma za zadanie umożliwić łączenie różnorodnych projektów z potencjalnymi donatorami i inwestorami, tworząc wspólnotę wspierającą się nawzajem w realizacji kreatywnych i innowacyjnych pomysłów. Dodatkowym kluczowym aspektem projektu jest opracowanie przejrzystego, zrozumiałego i intuicyjnego interfejsu użytkownika, co ma kluczowe znaczenie dla zapewnienia wygodnego i efektywnego korzystania z serwisu. Równie istotne jest zapewnienie wysokiego poziomu bezpieczeństwa, zarówno od strony frontendowej, jak i backendowej serwisu, co jest niezbędne do ochrony danych użytkowników oraz zapewnienia stabilności i wiarygodności platformy.

**1.3 Charakterystyka Zawartości Rozdziałów**

W kolejnych rozdziałach przedstawiona zostanie głęboka analiza tematu crowdfundingu (Rozdział 2), szczegółowe wymagania i narzędzia wykorzystane do stworzenia platformy (Rozdział 3), a także specyfikacja zewnętrzna i wewnętrzna projektu (Rozdziały 4 i 5). Następnie, omówione zostaną procesy weryfikacji i walidacji platformy (Rozdział 6), po czym praca zostanie podsumowana, a wyniki i wnioski będą przedstawione w Rozdziale 7.

**2.** **Analiza tematu**

**2.1 Sformułowanie problemu**

W ostatnich latach obserwuje się znaczny wzrost zainteresowania crowdfundingiem oraz projektami, które wykorzystują ten model finansowania. Równocześnie zauważalny jest wzrost popularności platform internetowych wspierających crowdfunding. W trakcie analizy dostępnych serwisów crowdfundingowych zidentyfikowano lukę rynkową, którą niniejsza praca ma na celu wypełnić.

Obecne polskojęzyczne platformy crowdfundingowe często koncentrują się na zbiórkach charytatywnych lub projektach o bardziej prywatnym charakterze. Wiele z nich oferuje ograniczone możliwości prezentacji celów – zazwyczaj poprzez pojedyncze zdjęcie i tekstowy opis. W ramach tego projektu planowane jest stworzenie platformy z rozbudowanymi opcjami opisu projektu, skoncentrowanej na bardziej kreatywnych inicjatywach. Chociaż istnieją zaawansowane platformy crowdfundingowe, większość z nich dostępna jest wyłącznie w języku angielskim, co może ograniczać ich zasięg w Polsce.

Ponadto, crowdfunding wiąże się z pewnymi komplikacjami, w tym z możliwością, że twórca projektu nie zrealizuje obietnic mimo zebrania funduszy. Chociaż całkowite wyeliminowanie tego ryzyka może być trudne, w projekcie przewidziano implementację rozwiązań mających na celu zminimalizowanie ryzyka oszustw i zwiększenie ochrony dla osób wspierających projekty.

**2.1 Opis istniejących rozwiązań**

W ramach analizy rynku, istniejące rozwiązania crowdfundingowe można podzielić na dwie główne grupy: międzynarodowe i polskie. Wśród polskich stron, które przeanalizowałem, znalazły się takie platformy jak polakpotrafi.pl, siepomaga.pl i wspieram.to, każda z nich charakteryzująca się swoimi zaletami i wadami.

Polakpotrafi.pl, będąca jedną z pierwszych tego typu platform w Polsce, obecnie boryka się z problemem niejednolitego interfejsu użytkownika. Siepomaga.pl, skupiająca się na akcjach charytatywnych, odbiega od celów mojej platformy, która ma koncentrować się na bardziej kreatywnych projektach.

Najbardziej zbliżona do mojej koncepcji jest platforma wspieram.to, która stosuje model kampanii „wszystko albo nic”. Jednakże, ten model, mimo że bezpieczny w kontekście unikania oszustw, może stwarzać ryzyko dla projektodawców, którzy inwestują własne środki. Oznacza on że zbiórkę uznaje się za udaną dopiero od 100% celu. Jeśli zbiórka jest nieudana cały zebrany fundusz zwracany jest do inwestorów. Alternatywny model „bierzesz ile zbierzesz” na wspieram.to wiąże się ze wzrostem prowizji z 8.5% do 9.5%, co jest stosunkowo wysokim oprocentowaniem w porównaniu do innych platform. Taki system prowizji może wpływać na decyzje projektodawców oraz sukces finansowy kampanii. Chociaż ten model ma swoje ograniczenia, regularnie obserwuje się na wspieram.to udane kampanie, co świadczy o jego skuteczności w pewnych warunkach.

**3.Wymagania i narzędzia**

**3.1 Wymagania funkcjonalne**

Portal będzie posiadał różną funkcjonalność zależnie od typu użytkownika. Rozpoznaję cztery typy, użytkownik nie zalogowany, użytkownik zalogowany z statusem USER, z statusem ADMIN i statusem SUPER.

Użytkownik nie zalogowany będzie miał jedynie dostęp do funkcjonalności publicznej. Ma dostęp do strony domowej gdzie wyświetla się kilka proponowanych projektów na podstawie jak blisko są do daty zakończenia, jak niewiele wsparcia brakuje do celu oraz te najnowsze. Może też szukać konkretnego projektu na liście projektów lub ją przefiltrować w celu przejrzenia tylko tych które mogą go zainteresować. Z strony konkretnego projektu można przejść na stronę włąsciciela projektu która zawiera szczegółowe informacje, w tym dane o inicjatorze zbiórki oraz innych projektach z nim związanych. Oprócz tego mogą skorzystać z opcji która wyświetli losowy, obecnie trwający projekt oraz opcji założenia konta lub zalogowania się na już istniejącego użytkownika. W trakcie dodawania konta użytkownika sprawdzane są odpowiednie warunki takie jak zgodność nazwy użytkownika i hasła z regułami bezpieczeństwa, poprawność adresu email.

Użytkownik typu USER ma dostęp do funkcjonalności publicznej z dodatkowymi opcjami. Przy przeglądaniu konkretnego projektu stworzonego prze innego użytkownika może on zdecydować się na wsparcie projektu wpłatą lub jeśli zdecyduje, że projekt jest podejrzany może o tym poinformować administrację serwisu. Jeśli użytkownik jest twórcą projektu nie może go wesprzeć kwotą ani poddać pod wątpliwość jego uczciwość. Zamiast tego może go modyfikować w ograniczonym zakresie. Dodatkowo może pod projektem napisać notatkę w której może poinformować innych użytkowników o postępie albo inną informację związaną z projektem.

Oczywiście normalny użytkownik, czyli użytkownik typu USER, ma możliwość stworzenia własnego projektu poprzez wykorzystanie specjalnie przygotowanej do tego strony i formularzu który zawiera .Projekty są weryfikowane na dwóch etapach. Najpierw w frontedndzie gdzie sprawdzane jest czy poszczególne wartości spełniają odpowiednie wymagania. Jak na przykład czy planowana data zakończenia jest później niż dzisiejsza data albo czy nazwa projektu nie zawiera żadnych zakazanych znaków. Drugi etap odbywa się po stronie backendu gdzie sprawdzane jest czy projekt spełnia wymagania pod względem bazy danych, jeśli nie to do frontu wysyłana jest wiadomość o konkretnym będzie na podstawie której strona informuje użytkownika co należy poprawić.

Po stworzeniu zbiórki musi ona być najpierw sprawdzona przez administrację serwisu czy nie łamie żadnych z zasad i czy nie wygląda zbyt podejrzanie.

Zalogowany użytkownik USER ma dostęp do strony z swoimi zbiórkami, tymi które się udały i które się nie udały, gdzie może je łatwo i szybko podejrzeć. Jeśli projekt został odrzucony przez moderację nie znajdzie się na liście publicznej projektów ale znajdzie się tutaj.

Dodatkowo każdy zalogowany użytkownik ma dostęp do strony z informacjami o swoim koncie. Gdzie może podać dodatkowe informacje o sobie lub do pewnego stopnia zmodyfikować te które już podał.

Użytkownicy typu ADMIN są administratorami serwisu. Mają dostęp do pełnej funkcjonalności publicznej. Poza tym mają dostęp do listy projektów które zostały zgłoszone prze innych użytkowników jako podejrzane. Na podstawie informacji podanych na stronie projektu lub ich braku mogą podjąć decyzje o przedwczesnym zamknięciu akcji jeśli uznają ją za dostatecznie podejrzaną. Dodatkowo zatwierdzają lub odrzucają stworzone projekty zanim staną się publiczne.

Ostatnim typem użytkownika jest typ SUPER którego jedynym celem jest dodawanie i blokowanie użytkowników typu ADMIN.

**3.2 Wymagania niefunkcjonalne**

W trakcie realizowania projektu największą wagę przyłożono do bezpieczeństwa serwisu, po stronie frontednu jak i backendu. Zaimplementowano kilka rodzajów zabezpieczeń takich jak: JWT (Jason Web Token), routr guards i własna configuracja CORS.

Dzięki JWT spora część endpointów przyjmie zapytanie tylko wtedy kiedy otrzyma także nagłówek zawierający poprawny i aktywny token identyfikujący dla istniejącego użytkownika. Dodatkowo dzięki ustawieniu limitu czasu na JWT, token przypisany konkretnemu użytkownikowi ma ważność 8 godzin, po tym czasie użytkownik musi logować się ponownie do serwisu.

Mechanizm Router Guard () przekierowania miedzy stronami będą odbywać się tylko jeśli zostaną spełnione odpowiednie warunki, a własna konfiguracja CORS zapewnia że zapytania mogą być przyjmowane jedynie od zaufanego źródła.

Dodatkowo aby zapewnić bezpieczeństwo transmisji ograniczono ilość przesyłanych informacji do wymaganego minimum, a wrażliwe dane np. hasło są szyfrowane.

Aby przestrzegać zasad RODO użytkownicy którzy nie skorzystali z żadnej znaczącej funkcjonalności są niewidoczni dla reszty użytkowników. Jeśli osoba korzystająca z serwisu zdecydowała się wesprzeć konkretny projekt, jego twórca ma wgląd tylko i wyłącznie do nazwy użytkownika i adresu email w celu ewentualnego kontaktu. Natomiast jeśli, użytkownik utworzył projekt inne osoby korzystające z strony mają wgląd w jego profil, gdzie podanie informacji prywatnych jest opcjonalne.

Durzą wagę przyłożono również do stworzenia intuicyjnego i łatwego w użyciu interfejsu z którego mógłby skorzystać jak najwięcej użytkowników. Istotne jest aby do każdej funkcji aplikacji można było uzyskać dostęp w jak najmniejszej liczbie przejść i żeby każde przejście było świadome, jeśli użytkownik jest czymś zainteresowany, to powinien od razu wiedzieć gdzie tego szukać. Spora część obecnie dostępnych rozwiązań strony croudfundingowej miała opcje które znajdowały się dla mnie w nie oczywistych miejscach albo opcja była niewidoczna z powodu jej umieszczenia lub kolorystyki, jak na przykład szary przycisk na czarnym tle.

W celu zapewnienia wysokiej jakości kodu zastosowana się do reguł czystego kodu (ang. Clean code). Aby kod był jak najbardziej czytelny starałem się aby był on w jak największym stopniu samo komentujący się, oznacza to że w kodzie nie powinno się stosować komentarzy a same nazwy funkcji i obiektów powinny tłumaczyć do czego służą i co robią, niestety znaczenie komplikuje to ich nazewnictwo. Dodatkowo trzymałem się zasady aby każda funkcjonalność była odpowiednio podzielona na pod funkcję, co pozytywnie wpływa na łatwość prowadzania modyfikacji do kodu. Starłem się także ograniczyć liczbę przesyłanych parametrów pakując je w obiekt dla łatwiejszego zrozumienia działania.

W celu ułatwienia diagnostyki zastosowałem Logback (logowanie informacji w springu) oraz własne błędy i mechanizm ich obsługi, dzięki czemu nie tylko łatwiej jest wyłapać błędy w trakcie działania aplikacji, ale możliwe jest obsłużenie błędu w taki sposób aby aplikacja w dalszym ciągu mogła pracować poprawnie.

**3.3 Przypadki użycia**

**3.4 Opis narzędzi**

**3.4.1 Języki programowania**

Aplikacja została stworzona z wykorzystaniem dwóch języków programowania: Java dla strony serwerowej i TypeScript dla interfejsu użytkownika.

W trakcie wybierania technologii do stworzenia frontendu głównie rozpatrywane były dwie opcje: JavaScript i TypeScript. JavaScript jest jednym z najbardziej popularnych narzędzi do tworzenia stron internetowych. Jest łatwo dostępny, nie wymaga żadnych dodatkowych narzędzi do uruchomienia i cechuje się dużym wsparciem społeczności, dzięki czemu istnieje wiele gotowych materiałów pomocniczych. Jego największą zaletą jest prostota i szybkość tworzenia kodu, wynikająca z dynamicznego typowania, zwalnia to programistę z obowiązku deklarowania typów w trakcie pisania programu, przypisanie typu odbywa się w trakcie działania aplikacji.

TypeScript, rozwinięcie JavaScriptu, umożliwia integrację typowanego oraz tradycyjnego kodu JavaScript w jednym projekcie. Jako nadzbiór JavaScriptu, TypeScript poszerza możliwości języka poprzez wprowadzenie statycznego typowania. W praktyce, każda zmienna, parametr funkcji czy obiekt muszą być jasno zdefiniowane typami, co znacząco przyczynia się do zwiększenia przejrzystości kodu. Taki system typowania ułatwia zarządzanie kompleksowymi projektami, minimalizuje ryzyko błędów związanych z niezamierzoną konwersją typów i wspiera efektywną współpracę programistów. Statyczne typowanie wprowadza większy porządek w strukturze kodu, co z kolei pozytywnie wpływa na jego jakość, ułatwiając refaktoryzację i testowanie

Ostateczny wybór TypeScriptu dla frontendu został podyktowany jego zaletami w zakresie skalowalności i utrzymania dużych baz kodu. Dzięki statycznemu typowaniu, TypeScript zapewnia lepszą kontrolę nad kodem i redukuje częstość występowania subtelnych błędów, które mogą być trudne do wykrycia w JavaScript. Dodatkowo, TypeScript oferuje zaawansowane funkcje programowania obiektowego oraz kompatybilność z nowoczesnymi frameworkami front-endowymi, co czyni go idealnym wyborem dla złożonych i dynamicznych aplikacji internetowych.

Decyzja o użyciu Javy dla backendu została podjęta z uwagi na jej wysoką wydajność i skalowalność, co jest niezbędne w aplikacjach serwerowych obsługujących liczne równoległe żądania. Java, dzięki swojej maszynie wirtualnej (JVM), gwarantuje łatwą skalowalność aplikacji oraz ich optymalną wydajność. Ponadto, uniwersalność JVM sprawia, że Java zapewnia wysoką kompatybilność i przenośność – kod źródłowy może być skompilowany do uniwersalnego kodu bajtowego, który następnie jest uruchamiany na dowolnej platformie z zainstalowanym JVM.

Kolejną zaletą Javy na którą zwrócono uwagę jest jej bezpieczeństwo i niezawodność. Podobnie jak TypeScript jest to język silnie typowany, co zapobiega wielu możliwym błędom w trakcie pracy programu. Dodatkowo ma mocno rozbudowany system obsługi wyjątków z możliwością definicji własnych oraz ich obsługi. Wyposażony jest również w automatyczny mechanizm garbage collection dzięki czemu ryzyko jakichkolwiek przecieków pamięci jest minimalizowane.

Podsumowując, wybór TypeScriptu dla frontendu i Javy dla backendu stanowi strategiczne połączenie, które zapewnia aplikacji zarówno łatwość modyfikacji, jak i niezawodność. TypeScript zwiększa jakość i bezpieczeństwo kodu dzięki silnemu typowaniu, co jest kluczowe dla nowoczesnych interfejsów użytkownika. Java, z kolei, wnosi wydajność i stabilność do obsługi serwerowej, z niezawodnym zarządzaniem pamięcią i mocnym wsparciem dla wielowątkowości. Razem tworzą one solidną bazę dla rozbudowanej, bezpiecznej i łatwej w utrzymaniu aplikacji.

**3.4.2 Frameworki**

W ramach projektu, kluczową rolę odgrywają nie tylko języki programowania TypeScript i Java ale też dwa nowoczesne i wydajne frameworki stworzone dla tych języków: Angular i Spring Boot. Wybór tych technologii nie jest przypadkowy, oba frameworki oferują bogaty zestaw funkcji, które znacząco przyspieszają rozwój aplikacji, a także zapewniają jej skalowalność, bezpieczeństwo i łatwość utrzymania.

Angular, wykorzystywany w warstwie frontendowej, jest frameworkiem, oferując wszechstronne rozwiązania dla zarządzania stanem aplikacji, routingiem oraz dwustronnym wiązaniem danych. Te funkcje sprawiają, że proces tworzenia interaktywnych interfejsów użytkownika jest znacznie szybszy i bardziej efektywny. Dodatkowo, modularna architektura Angulara pozwala na tworzenie dobrze zorganizowanego i łatwego w utrzymaniu kodu, co jest nieocenione w projektach o dużym zakresie. Za największą zaletę Angulara uważam wbudowaną zdolność do włączania instrukcji warunkowych i pętli bezpośrednio w kodzie HTML, co znacznie upraszcza tworzenie dynamicznych interfejsów użytkownika.

Spring Boot, używany do budowy backendu, umożliwia szybkie tworzenie aplikacji serwerowych dzięki automatycznym konfiguracjom i prostemu wdrożeniu. Jest to idealne rozwiązanie dla szybkich iteracji i prototypowania. Spring Boot jest również doskonałym wyborem dla tworzenia bezpiecznych i łatwych w skalowaniu mikro serwisów, oferując jednocześnie wsparcie dla szerokiej gamy funkcji bezpieczeństwa.

Zarówno Angular, jak i Spring Boot są znane z ich zdolności do radzenia sobie z dużymi, złożonymi aplikacjami, oferując niezbędne narzędzia do budowy, testowania i utrzymania wydajnych i niezawodnych systemów. Wybrano je nie tylko z powodu podanych wyżej podanych zalet ale są to też technologię z którymi Autor ma największe doświadczenie.

**3.4.3 Baza danych**

W projekcie postanowiłem wykorzystać PostgreSQL, otwarto źródłowy system zarządzania bazami relacyjnymi bazami danych (RDBMS). Cechuje się on durzą rozszerzalnością poprzez dodatkowe moduły, co pozwala na implementację niestandardowych funkcji i typów danych. PostgreSQL swojej cechuje się dużą wydajnością, szczególnie w obsłudze dużych baz danych i skomplikowanych operacji. Mechanizmy takie jak indeksowanie GIN (**Generalized Inverted Index)** i GIST (**Generalized Search Tree**), partycjonowanie tabel oraz możliwość optymalizacji zapytań, przyczyniają się do szybkiego przetwarzania danych oraz możliwości przechowywania bardziej złożonych danych w jednym polu, jak na przykład json.

**3.4.4 Środowisko i narzędzia**

W procesie tworzenia aplikacji, oprócz głównych technologii takich jak języki programowania i frameworki, kluczowe znaczenie mają również środowiska deweloperskie i narzędzia pomocnicze. W projekcie wykorzystano narzędzia, które nie tylko ułatwiają pisanie kodu aplikacji ale umożliwiają testowanie i zapewniają konsystencję środowiska.

**Środowiska Deweloperskie (IDE)**

* **IntelliJ IDEA Ultimate (Backend):** Dla backendu użyto IntelliJ IDEA Ultimate na licencji studenckiej, co jest jednym z najbardziej popularnych IDE dla Javy obecnie na rynku. Jest to kompleksowe środowisko, które oferuje zaawansowane funkcje takie jak refaktoryzacja kodu, wsparcie dla Maven i Gradle, integracja z systemami kontroli wersji oraz wsparcie dla frameworków takich jak Spring. Jego inteligentne uzupełnianie kodu i możliwości debugowania znacznie przyspieszają i ułatwiają pracę. Dodatkowo oferuje całą gamę możliwych dodatków do zainstalowania, które dodają funkcjonalność domyślnie nie zawarte w IDE
* **Visual Studio Code:** Dla Angulara zastosowano VS Code ponieważ jest on szybkim i intuicyjnym edytorem tekstu, którego największa zaleta są rozszerzenia które można dowolnie dodawać i konfigurować. Dzięki tym rozszerzeniom łatwo go skonfigurować pod dowolny język i technologię. Nie było konieczne użycie dedykowanego IDE dla Angulara ponieważ zawiera on narzędzie ng do tworzenia i modyfikowania projektu.

**Narzędzia do testowania i zarządzania kodem**

* **Git:** Do zarządzania wersją kodu oraz jej ewentualnego przesyłania między maszynami wykorzystałem Git, a dokładniej platformę GitHub. Pozwala to na efektywne zarządzanie kodem, śledzenie zmian i wersji, rozsyłanie między różnymi komputerami oraz przechowywanie w razie awarii lokalnej maszyny. W tym celu pomocny był GitBash, jest to terminal który ułatwia korzystanie z komend git oraz oferuje dodatkową funkcjonalność jak na przykład wyświetlanie z jakiej gałęzi obecnie jest projekt.
* **PgAdmin:** Do zarządzania bazą danych zastosowałem narzędzie PgAdmin. Jest to zaawansowane narzędzie graficzne służące do zarządzania i projektowania baz danych PostgreSQL. Jest to jedno z najpopularniejszych narzędzi tego typu, używane zarówno przez początkujących, jak i doświadczonych administratorów baz danych. Umożliwia wizualizację struktur i relacji w bazie danych, łatwe modyfikowanie wartości w tabelach oraz wydajności bazy danych, co pomaga w identyfikacji i rozwiązywaniu problemów związanych z wydajnością i optymalizacją.
* **Postman:** Dla testowania poprawności endpitów stworzonych w Springu przed przystąpieniem do ich implementacji w froncie należałoby je przetestować aby niepotrzebnie implementować coś co nie dział. Do tego celu wykorzystano postmana, aplikację która pozwala testować wysyłanie zapytań i odbieranie informacji z backendu. Dodatkowo postman jest wyposażony w opcję pisania skryptów, które są w stanie automatycznie sprawdzić czy odpowiedź spełnia warunki jakie zadamy.

**Docker**

Docker odgrywa kluczową rolę w zapewnieniu spójności środowiskowej i ułatwia wdrożenie aplikacji. Aby zachować jednolite działanie aplikacji niezależnie od maszyny, na jakiej jest uruchamiana, zdecydowałem no się na wykorzystanie Dockera i Docker Desktop. Oto kilka kluczowych zalet, które Docker wnosi do projektu:

* **Izolacja Środowiskowa:** Docker używa kontenerów do izolacji aplikacji, co zapewnia, że działa ona w identyczny sposób na każdej maszynie. Kontenery Docker eliminują problem „u mnie działa” poprzez zapewnienie, że wszyscy programiści pracują na tym samym środowisku, co znacznie ułatwia współpracę i testowanie.
* **Elastyczność i Skalowalność:** Docker pozwala na łatwe skalowanie aplikacji, umożliwiając uruchomienie wielu instancji kontenera bez konieczności konfigurowania dodatkowych maszyn. Jest to niezwykle przydatne w zarządzaniu obciążeniem i dostosowaniu aplikacji do zmiennego ruchu.
* **Łatwość Wdrożenia:** Docker ułatwia wdrażanie aplikacji na różnych środowiskach, od lokalnych maszyn deweloperskich po serwery produkcyjne. Dzięki zastosowaniu Dockera, proces wdrożenia jest szybki i niezawodny, co jest kluczowe w środowiskach, gdzie regularne aktualizacje i szybka reakcja na zmiany są ważne.
* **Zarządzanie Zależnościami:** Docker pozwala na definiowanie i zarządzanie zależnościami aplikacji za pomocą plików Dockerfile. Każdy kontener może zawierać wszystkie niezbędne zależności, co zapewnia, że aplikacja będzie działała poprawnie w każdym środowisku.
* **Komunikacja między kontenerami: Gdy w projekcie znajduje się więcej niż jeden element korzystający z osobnego** Dockerfile można je połączyć za pomocą docker-compose, jest to plik który definiuje jakie kontenery stworzyć, jak powinny się one między sobą komunikować oraz zależności między nimi. Ponieważ zrealizowany projekt składa się z trzech kontenerów (frontend, backend i postgreSQL) docker-compose zapewnia, że backend zostanie uruchomiony dopiero po pełnym uruchomieniu postgreSQL oraz to, że kontenery z frontem i backiem będą się mogły między sobą komunikować dzięki docker-network.

**3.5 Metodyka pracy**

Aby ujednolicić pracę nad projektem jego tworzenie zostało podzielone na kilka etapów. Na początku, utworzono ogólny obraz aplikacji, jakie strony powinna zawierać, co na tych stronach powinno się znaleźć, jaka funkcjonalność powinna być dostępna na poszczególnej stronie i dla jakiego typu użytkownika powinna ona być dostępna. Następnie zaprojektowano diagram UML reprezentujący bazę danych, jakie tabele i jakie pola powinny się w niej znaleźć oraz jakie relacje powinny zachodzić pomiędzy poszczególnymi tabelami. Na podstawie ogólnego obrazu aplikacji i diagramu UML cała funkcjonalność została podzielona na niezależne od siebie fragmenty które następnie zostały zaimplementowane w zaplanowanej kolejności.

W trakcie wdrażania nowej funkcjonalności aplikacji, proces rozpoczynał się od opracowania części serwerowej. Następnie, wykorzystując aplikację Postman i narzędzie PgAdmin, przeprowadzane były testy w celu potwierdzenia prawidłowości działania. Ewentualne błędy były identyfikowane i naprawiane. Po weryfikacji poprawności backendu, przechodzono do implementacji w Angularze. Początkowo skupiano się na kodowaniu logiki strony, takiej jak wysyłanie zapytań do backendu i konfiguracja nawigacji. Następnie, w celu wizualizacji projektu, projektowano układ poszczególnych elementów interfejsu użytkownika (rys. 3.1), co stanowiło podstawę do stworzenia odpowiednich plików HTML i CSS.

Obraz zawierający Prostokąt, zrzut ekranu, tekst, diagram

Opis wygenerowany automatycznie

rys. 3.1 Przykład schematu rozmieszczenia elementów interfejsu użytkownika

Po zaimplementowaniu części serwerowej i klienckiej przeprowadzono testy funkcjonalne wykorzystując różnorodne dane wejściowe oraz scenariusze użytkowania, takie jak interakcje między różnymi użytkownikami i projektami. Sukcesywne testowanie w lokalnym środowisku było następnie uzupełniane o weryfikację działania aplikacji w konteneryzowanym środowisku Docker, co pozwoliło na sprawdzenie zachowania systemu w izolowanej instancji.

**4. Specyfikacja zewnętrzna**

**4.1 Wymagania sprzętowe i instalacja**

Podstawowym wymaganie do instalacji projektu jest posiadanie jego kodu. W tym celu można skorzystać z GitHuba i ściągnąć plik .zip z kodem lub korzystając z komend gitowych sklonować repozytorium na maszynę lokalną.

**4.1.1 Instalacja i wymagania Lokalne**

**Dla Backendu:**

* **Java JDK:** Wymagana jest instalacja Java JDK w wersji 8 lub wyższej, co można zweryfikować przy użyciu polecenia java -version w terminalu.
* **IntelliJ IDEA:** Rekomendowane jest pobranie IntelliJ IDEA, zawierającego Gradle, ze strony JetBrains.
* **Alternatywne IDE:** W przypadku innych środowisk deweloperskich należy samodzielnie zainstalować Gradle w wersji 8, dostępny na gradle.org.

**Dla Frontendu:**

* **Node.js:** Powinien być zainstalowany Node.js w wersji 20, dostępny na nodejs.org.
* **NPM:** Wersja npm, instalowana razem z Node.js, powinna być minimum 10.
* **Zależności:** Aby zainstalować wszelkie zależności dla Agulara należy w katalogu projektu FundRippleGUI uruchomić komendę npm install.

**4.1.2 Instalacja z użyciem Dockera**

* **Docker Desktop:** Dla użytkowników systemów Windows i macOS zalecane jest zainstalowanie Docker Desktop. Można go pobrać z oficjalnej strony Dockera.
* **Docker na Linux:** Użytkownicy systemów Linux mogą zainstalować Dockera bezpośrednio z repozytorium swojej dystrybucji.

Powyższe instrukcje powinny umożliwić efektywne skonfigurowanie i zainstalowanie aplikacji w środowisku lokalnym oraz z użyciem Dockera. Jest to niezbędne, aby umożliwić przyszłym deweloperom i użytkownikom pracę nad aplikacją oraz jej testowanie.

**4.2 Sposób uruchomienia**

Metoda uruchomienia aplikacji zależy od wybranego sposobu instalacji, opisanego w podrozdziale 4.1.

**Dla Instalacji Lokalnej:**

* **Backend:**
  + Należy przejść do katalogu FundRippleAPI, zawierającego pliki gradlew i build.gradle.
  + W wierszu poleceń wykonuje się polecenie ./gradlew bootrun. Zaleca się, aby przy pierwszym uruchomieniu lub w przypadku problemów, wykonać najpierw ./gradlew clean build.
  + Alternatywnie, jeśli środowisko IDE na to pozwala, aplikacja może być uruchomiona bezpośrednio z poziomu IDE.
* **Frontend:**
  + Przechodzi się do katalogu FundRippleGUI.
  + W wierszu poleceń wpisuje się ng serve.
  + Należy zwrócić uwagę, że pierwsze uruchomienia mogą zająć więcej czasu z powodu pobierania i instalowania zależności.

**Dla Instalacji z Użyciem Dockera:**

* W głównym katalogu projektu uruchamia się skrypt start.sh, który odpowiedzialny jest za uruchomienie poszczególnych kontenerów oraz docker-compose, koordynującego ich działanie.

**4.3 Realizacja głównych przypadków użycia**

**4.3.1 Rejestracja i logowanie**

* **Opis Procesu:** Dostęp do ekranów rejestracji i logowania dla niezalogowanych użytkowników jest zapewniony poprzez pasek menu na górze każdej dostępnej strony. W trakcie rejestracji użytkownik jest proszony o podanie podstawowych informacji, takich jak nazwa użytkownika, hasło, adres email oraz imię i nazwisko (rys. 4.1). W przypadku niespełnienia wymogów serwisu przez podane dane, wyświetlane jest ostrzeżenie i proces rejestracji jest wstrzymywany. Po pomyślnej rejestracji następuje automatyczne przekierowanie na stronę domową. W procesie logowania, z kolei, wymagane jest podanie nazwy użytkownika i hasła (rys. 4.2), przy czym błędne dane skutkują pojawieniem się komunikatu o błędzie. Po udanym zalogowaniu, podobnie jak w przypadku rejestracji, użytkownik jest przekierowywany na stronę główną. Pomyślne zakończenie procesu rejestracji lub logowania skutkuje otrzymaniem tokena JWT, który jest przechowywany lokalnie w przeglądarce aż do momentu wylogowania się użytkownika lub utraty ważności tokenu. Token ten służy do weryfikacji statusu logowania użytkownika oraz dostępu do różnych funkcji serwisu.
* **Interfejs Użytkownika:**

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

rys. 4.1 Formularz rejestracji użytkownika

Obraz zawierający zrzut ekranu, oprogramowanie, design

Opis wygenerowany automatycznie

rys. 4.2 Formularz logowania użytkownika

* **Backend:** Proces rejestracji rozpoczyna się od weryfikacji, czy podane przez użytkownika dane spełniają określone wymogi. Istotnym elementem jest unikalność nazwy użytkownika w całym serwisie, która jest warunkiem koniecznym do zapisu użytkownika w bazie danych. Hasło przesłane z interfejsu użytkownika jest szyfrowane przed zapisaniem w tabeli autoryzacji, która powiązana jest relacją jeden do jeden z tabelą użytkowników. Taki podział na dwie tabele ułatwia ewentualne przyszłe zmiany w systemie autoryzacji, na przykład wprowadzenie pytań bezpieczeństwa, bez konieczności modyfikacji głównej tabeli użytkowników. Po pomyślnym zarejestrowaniu, generowany jest token JWT, który następnie jest wysyłany do frontendu. Podczas logowania, backend najpierw sprawdza, czy istnieje użytkownik o podanej nazwie. Następnie weryfikowana jest poprawność hasła. Jeśli obie te czynności zakończą się pomyślnie, podobnie jak w przypadku rejestracji, użytkownik otrzymuje token JWT. W przypadku nieudanej próby logowania lub rejestracji, backend wysyła do frontendu komunikat o błędzie, informujący o przyczynie niepowodzenia.

**4.3.2 Tworzenie nowego projektu jako użytkownik**

* **Opis Procesu:** Dostęp do funkcji tworzenia nowego projektu jest ograniczony wyłącznie do zalogowanych użytkowników. Proces ten składa się z czterech etapów. Na pierwszym etapie użytkownik wprowadza podstawowe informacje o projekcie, takie jak nazwa, krótki opis, zakładany cel finansowy oraz planowaną datę zakończenia zbiórki (rys. 4.4). W drugim etapie następuje szczegółowe opisanie projektu, gdzie treść dzielona jest na różne elementy, takie jak paragrafy tekstu lub zdjęcia, które mogą być dowolnie modyfikowane, usuwane lub dodawane (rys. 4.5). Po dodaniu, te elementy opisu będą prezentowane na stronie zbiórki w tej samej kolejności, w jakiej zostały wprowadzone, co umożliwia tworzenie dynamicznych i atrakcyjnych opisów. Następnie, użytkownik ma możliwość określenia dodatkowych celów projektu, które mogą zostać zrealizowane po osiągnięciu określonej kwoty finansowania (rys. 4.6). Można także określić benefity dla wspierających, określając minimalną kwotę wsparcia niezbędną do ich otrzymania (rys. 4.7). Liczba celów dodatkowych i benefitów jest elastyczna, a ich dodanie nie jest obowiązkowe. W trakcie dodawania projektu, proces jest podzielony na cztery interaktywne strony, między którymi można się przemieszczać za pomocą odpowiednich przycisków. Na ostatniej stronie znajduje się przycisk finalizujący proces tworzenia projektu. Przed wysłaniem danych do backendu, są one weryfikowane pod kątem spełnienia podstawowych wymagań, na przykład kompletności danych czy odpowiedniej daty zakończenia. Dane są przesyłane do backendu w czterech etapach: informacje o projekcie, opis, cele dodatkowe i benefity. W przypadku wystąpienia błędów na dowolnym etapie, proces jest przerywany, a użytkownik otrzymuje informacje o przyczynie niepowodzenia. W przypadku pomyślnego zakończenia procesu, użytkownik zostaje przekierowany na stronę projektu.
* **Interfejs Użytkownika:**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie**Obraz zawierający tekst, elektronika, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

rys. 4.4 Pierwsz strona dodawania projektu rys. 4.5 Druga strona dodawania projektu

Obraz zawierający tekst, zrzut ekranu, numer

Opis wygenerowany automatycznie

rys. 4.6 Trzecia strona dodawania projektu

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

rys. 4.4 Pierwsz strona dodawania projektu rys. 4.5 Druga strona dodawania projektu

* **Backend:** Przed przystąpieniem do jakiejkolwiek operacji, backend sprawdza, czy otrzymane zapytanie zawiera poprawny token JWT. W przypadku braku tokenu lub jego nieprawidłowości, zwracany jest błąd z kodem 403. Gdy token jest zatwierdzony jako poprawny, następuje dodanie odpowiednich elementów do bazy danych. Proces dodawania projektu odbywa się w czterech etapach, zgodnie z opisem procedury. W sytuacji, gdy wystąpi błąd na którymkolwiek z tych etapów, wszystkie dane wprowadzone w ramach poprzednich kroków są usuwane z bazy danych. Czynność ta ma na celu zapewnienie spójności danych. Istotne jest również uwzględnienie ograniczeń niektórych tabel, takich jak wymóg unikalności nazwy projektu w tabeli projektów. Dzięki temu unika się dezorganizacji w przypadku, gdy konieczne jest oczyszczenie tabeli.

**4.3.3 Wsparcie wybranego projektu**

* **Opis Procesu:** Zalogowany użytkownik typu USER, który zdecyduje się wesprzeć znaleziony projekt, może to zrobić poprzez kliknięcie odpowiedniego przycisku, który otwiera wyskakujące okno. W tym oknie użytkownik wpisuje kwotę, którą chce przekazać na zbiórkę (rys. 4.6). Po określeniu kwoty, płatność realizowana jest za pośrednictwem serwisu PayPal. Proces ten przekierowuje użytkownika na stronę PayPal, gdzie wymagane jest zalogowanie do konta użytkownika. Po zalogowaniu i potwierdzeniu szczegółów płatności (rys. 4.7), w przypadku pomyślnej transakcji, użytkownik jest przekierowywany na tymczasową pustą stronę gdzie w tle obsługiwany jest zapis płatności do bazy danych, następnie następuje przekierowanie na stronę wspartego projektu, gdzie można sprawdzić łączną kwotę jaką do tej pory użytkownik wsparł zbiórkę. W sytuacji, gdy transakcja zakończy się błędem, użytkownik również zostaje przekierowany na stronę projektu, lecz otrzymuje informację o nieudanej płatności.
* **Interfejs Użytkownika:**

Obraz zawierający tekst, zrzut ekranu, Prostokąt, design

Opis wygenerowany automatycznie

rys. 4.6 Interfejs wspierania zbiórki w Angular

Obraz zawierający tekst, elektronika, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

rys. 4.7 Interfejs PayPal do potwierdzenia danych płatności

* **Backend:** Strona serwerowa, przed przystąpieniem do obsługi płatności, inicjuje proces weryfikacji zgodności tokena JWT. Po potwierdzeniu poprawności nagłówka JWT, serwer rozpoczyna proces tworzenia transakcji płatniczej. W trakcie tego procesu, obok kwoty płatności, rejestrowane są dodatkowe informacje, takie jak waluta oraz adresy URL, na które użytkownik ma zostać przekierowany po udanej lub nieudanej operacji płatności na stronie PayPal. Następnie generowany jest specyficzny link przekierowujący bezpośrednio do interfejsu PayPal dedykowanego dla danej płatności. Obiekt zawierający wszystkie niezbędne informacje do przeprowadzenia płatności jest przygotowywany po stronie serwera, a następnie przekazywany do frontendu Angular jako część odpowiedzi.

**4.3.4 Przeszukanie projektów przez użytkownika**

* **Opis Procesu:** Do przeszukania projektów w serwisie służy dedykowana strona (rys. 4.8). Funkcjonalność ta umożliwia wyszukiwanie projektów poprzez pole tekstowe, które filtruje projekty według nazwy. Użytkownik ma również do dyspozycji drop-down menu umożliwiające wybór do trzech tagów oraz opcję filtrowania projektów w zależności od ich statusu – czy są zamknięte, w trakcie realizacji, lub oba. Dodatkowo, strona oferuje opcje sortowania wyników według różnych kryteriów, w porządku rosnącym lub malejącym. Wyniki wyszukiwania nie są prezentowane wszystkie jednocześnie; zaimplementowano stronicowanie, dzięki któremu użytkownik może wybrać liczbę projektów wyświetlanych na jednej stronie oraz nawigować między stronami za pomocą przycisków. Po wybraniu konkretnego projektu, użytkownik jest przekierowywany na dedykowaną stronę tego projektu. Operacje filtrowania i sortowania odbywają się po stronie frontendu, co ma na celu ograniczenie komunikacji z serwerem. Należy jednak zaznaczyć, że na wolniejszych urządzeniach może to prowadzić do krótkich opóźnień w przetwarzaniu.
* **Interfejs Użytkownika:**

Obraz zawierający tekst, zrzut ekranu, diagram, design

Opis wygenerowany automatycznie

rys. 4.8 Interfejs przeszukiwania projektów

* **Backend:** W backendzie główną operacją jest przesyłanie początkowej listy projektów do frontendu. W celu zminimalizowania przesyłania niepotrzebnych danych, zaimplementowano mapowanie projektów na uproszczone obiekty zawierające jedynie niezbędne informacje do wyświetlenia w interfejsie użytkownika. Dodatkowo, w celu dalszej optymalizacji, przesyłane są wyłącznie te projekty, które są w trakcie zbiórki lub zakończyły się pozytywnie. Ta metoda pozwala na ograniczenie ilości przesyłanych danych, jednocześnie zapewniając efektywne i celowane dostarczanie informacji.

**4.4 Administracja serwisu**

Administracja serwisu jest zadaniem użytkowników o statusie ADMIN. Ich obowiązki obejmują przeglądanie nowo utworzonych zbiórek, które jeszcze nie zostały upublicznione, w celu weryfikacji zgodności z zasadami serwisu. Ponadto, zajmują się oni analizą akcji zgłoszonych jako podejrzane przez innych użytkowników. Proces przeglądania zbiórek przez administratorów różni się od standardowego przeglądania projektów przez zwykłych użytkowników tym, że pobierany jest inny zestaw danych z serwera. W przypadku weryfikacji projektów pobierane są tylko te z oznaczeniem TO\_VERIFY, natomiast przy kontroli zgłoszeń użytkowników bierze się pod uwagę te projekty, które zgromadziły określoną liczbę zgłoszeń.

Administratorzy, sprawdzający zbiórki po raz pierwszy, stają przed wyborem: zaakceptować lub odrzucić daną akcję. W przypadku decyzji o odrzuceniu, muszą podać powód tej decyzji w wyświetlającym się oknie (zobacz rys. 4.9), co skutkuje zmianą statusu projektu na EARLY\_CLOSE. Jeżeli zbiórka zostanie zaakceptowana, jej status zmienia się z TO\_VERIFY na OPEN, co umożliwia jej dostępność dla pozostałych użytkowników serwisu.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, design

Opis wygenerowany automatycznie

rys. 4.9 Interfejs weryfikacji projektu

Podczas rozpatrywania akcji zgłoszonych przez użytkowników jako podejrzane, administratorzy mają do dyspozycji te same opcje, co w przypadku weryfikacji nowych projektów. Proces wczesnego zamknięcia akcji odbywa się w identyczny sposób. Jednakże, proces akceptacji różni się w zależności od statusu projektu.

Gdy projekt jest już publiczny, ale został zgłoszony jako podejrzany przez wystarczającą liczbę użytkowników, a administrator oceni go jako bezpieczny, status projektu pozostaje niezmieniony. Liczba użytkowników, którzy uznały projekt za podejrzany, jest stała i nadal jest wyświetlana. Jednakże, licznik zgłoszeń podejrzeń zostaje zresetowany. Oznacza to, że aby projekt był ponownie rozpatrywany przez administrację, musi być ponownie zgłoszony jako podejrzany określoną liczbę razy.

Dodatkowym elementem administracji serwisu jest konto typu SUPER. W bazie danych powinno istnieć tylko jedno takie konto, a jego jedyną funkcjonalnością jest tworzenie nowych kont administratorów(rys. 4.10).

Obraz zawierający tekst, zrzut ekranu, design

Opis wygenerowany automatycznie

rys. 4.10 Interfejs funkcjonalności użytkownika typu SUPER

**5 Specyfikacja wewnętrzna**

**5.1 Architektura serwisu**

**5.1.1 Architektura po stronie backendu**

Jak wspomniano w rozdziale 3.4, serwerowa część aplikacji została zbudowana głównie w języku Java z użyciem frameworku Spring Boot, co ułatwia tworzenie bezpiecznych aplikacji typu REST API, znanych również jako RESTful API.

Aplikacja typu RESTful API łączy dwa kluczowe elementy: REST (Representational State Transfer) i API (Application Programming Interface). API to zestaw reguł i protokołów, które umożliwiają aplikacjom komunikowanie się ze sobą, działając jak most umożliwiający dostęp do funkcji, danych lub zasobów innego oprogramowania. Definiuje ono sposób współpracy programów, określając metody wymiany danych.

REST to styl architektury oprogramowania, powszechnie stosowany w tworzeniu usług sieciowych, zwłaszcza dla aplikacji internetowych. Opiera się on na zestawie zasad, takich jak:

1. **Bezstanowość:** Każde żądanie HTTP zawiera wszystkie informacje potrzebne do jego zrozumienia i wykonania, a serwer nie przechowuje stanu klienta między żądaniami.
2. **Jednolity Interfejs:** Umożliwia separację klienta od serwera poprzez ujednolicenie komunikacji.
3. **Cacheable:** Odpowiedzi serwera mogą być cache'owane, co zwiększa wydajność.
4. **Klient-Serwer:** Oddziela interfejs użytkownika od przechowywania danych, co umożliwia niezależny rozwój obu.
5. **Warstwowy System:** Pozwala na ukrycie szczegółów serwera przed klientem, co zwiększa skalowalność i bezpieczeństwo.

REST API wykorzystuje standardowe metody HTTP, jak GET, POST, PUT, DELETE, do zarządzania zasobami reprezentowanymi w formatach JSON lub XML.

Sama aplikacja REST API stworzona w Spring Boot popularnie składa się głównie z trzech elementów. Kontrolery które opisują adres, metodę (GET, POST, PUT, DELETE) i dane jakie należy wysłać w zapytaniu do serwera aby otrzymać konkretną odpowiedź. Usługi które spełniają określone funkcjonalności, są wykorzystywane w kontrolerach aby otrzymać oczekiwany wynik. Modle które służą między innymi do poprawnego odbierania danych, ich przechowywania, modyfikowania oraz wysyłania jako odpowiedzi na zapytania dla serwera. Taka struktura została zastosowana w projekcie, plus dodatkowo rozbudowana na potrzeby serwisu.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

rys. 5.1 Struktura projektu w Java Spring Boot, pokazująca podział na pakiety

W pakiecie controller (rys. 5.1) zamieszczone są wszystkie kontrolery, w których obsługiwane są poszczególne endpointy.

W linijkach 1 do 3 (rys. 5.2) znajdują się kluczowe adnotacje, @RestController wskazuje, że klasa jest używana do obsługi żądań HTTP. Adnotacja @RequestMapping("/project") określa, że wszystkie żądania HTTP na ścieżce /project będą obsługiwane przez metody w tej klasie. Adnotacja @RequiredArgsConstructor generuje konstruktor z wymaganymi zależnościami, które są oznaczone jako finalne lub oznaczone adnotacją @NonNull, co umożliwia wstrzykiwanie zależności przez Springa.

Fragment kodu w liniach 8 do 23 (rys. 5.2) reprezentuje metodę w kontrolerze Spring Boot, która obsługuje żądania POST na ścieżce /project/add. Używa ona adnotacji @PostMapping("/add") do mapowania żądań HTTP POST na metodę addProject. Metoda ta przyjmuje dwa argumenty: obiekt ProjectWriteModel, który jest oczekiwanym ciałem żądania (@RequestBody), oraz ciąg header, który jest wartością nagłówka HTTP Authorization (@RequestHeader("Authorization")). Wewnątrz metody, próbuje ona dodać nowy projekt za pomocą serwisu projectService. Jeśli operacja się powiedzie, zwraca ResponseEntity z kodem odpowiedzi HTTP 200 (OK). W przypadku wystąpienia wyjątku ProjectException, zwraca ResponseEntity z kodem błędu HTTP 400 (BAD\_REQUEST) i komunikatem wyjątku.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | @RestController @RequestMapping("/project") @RequiredArgsConstructor public class ProjectController {   private final ProjectService projectService;  @PostMapping("/add")  public ResponseEntity<?> addProject(  @RequestBody ProjectWriteModel projectWriteModel,  @RequestHeader("Authorization") String header  ) {  try {  return ResponseEntity.*ok*(projectService.addProject(projectWriteModel, header));  } catch (ProjectException e) {  return ResponseEntity  .*status*(HttpStatus.*BAD\_REQUEST*)  .body(e.getMessage());  }  } |

rys. 5.2 Przykład kontrolera w aplikacji Spring Boot z jednym endpointem

W pakiecie service (rys. 5.1) umieszczone są serwisy, które pełnią rolę klas zawierających funkcje biznesowe. Te klasy są wykorzystywane przez inne elementy programu, na przykład kontrolery. Istotnym elementem serwisu jest adnotacja @Service, umieszczona przed definicją klasy, która informuje kontener Springa o traktowaniu klasy jako serwisu. Dzięki temu, Spring zarządza tą klasą jako beanem, co umożliwia wstrzykiwanie zależności.

Adnotacja @Service pozwala Springowi na automatyczne wykrywanie serwisu jako komponentu, który może być wstrzyknięty do innych klas, w tym do kontrolerów. Serwisy, umiejscowione w pakiecie service, są zazwyczaj miejscem, gdzie znajduje się logika biznesowa aplikacji, co pozwala na jej oddzielenie od warstwy prezentacji i dostępu do danych. Wykorzystanie mechanizmów transakcyjnych oferowanych przez Spring umożliwia deklaratywne zarządzanie transakcjami bazodanowymi.

Dodatkowo, adnotacja @Autowired, stosowana na polach klasy, konstruktorach, setterach oraz innych metodach, umożliwia automatyczne wstrzykiwanie odpowiednich instancji komponentów, takich jak beany serwisów czy DAO (Data Access Object), do zadeklarowanych miejsc w klasie. Umożliwia to tworzenie bardziej luźno powiązanych, łatwych w utrzymaniu i skalowalnych aplikacji. Może się to jednak wiązać z utrudnieniami w testowaniu i zmniejszeniu przejrzystości kodu.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | @Service public class SubGoalService {  private final ProjectRepository projectRepository;  private final SubGoalMapper subGoalMapper;  private final SubGoalRepository subGoalRepository;  @Autowired  private CleanService cleanService;   public SubGoalService(ProjectRepository projectRepository, SubGoalMapper subGoalMapper, SubGoalRepository subGoalRepository) {  this.projectRepository = projectRepository;  this.subGoalMapper = subGoalMapper;  this.subGoalRepository = subGoalRepository;  }   public List<SubGoal> getAllSubGoalsForProject(String projectName) {  Project project = projectRepository.findProjectByProjectName(projectName);  return subGoalRepository.getSubGoalsByProject(project);  } |

rys. 5.3 Przykład serwisu z funkcją zwracającą listę dodatkowych celów dla danego projektu

W pakiecie model (rys. 5.1) znajdują się różnego rodzaju obiekty wykorzystywane w projekcie. W tym miejscu umieszcza się obiekty Data Transfer Object (DTO), reprezentacje elementów bazy danych, enumeratory oraz inne struktury danych niezbędne w aplikacji.

W liniach od 1 do 6 znajdują się adnotację do całej klasy Project. Adnotacja @Entity jest używana do oznaczenia klasy jako encji JPA (Java Persistence API), co oznacza, że jej instancje są przechowywane w bazie danych i każda instancja encji odpowiada wierszowi w tabeli. Następnie, adnotacja @Table(name = “projects”) precyzuje, że encja ma być mapowana na tabelę o nazwie „projects” w bazie danych, co pozwala na szczegółowe odwzorowanie między klasą a tabelą. Dodatkowo, adnotacje @NoArgsConstructor i @AllArgsConstructor, dostarczone przez bibliotekę Lombok, automatycznie generują odpowiednio konstruktor bezargumentowy i konstruktor przyjmujący argumenty dla każdego pola klasy. To umożliwia łatwą inicjalizację i tworzenie nowych instancji encji. Ponadto, adnotacje @Getter i @Setter, również z Lomboka, generują metody dostępowe i modyfikujące dla każdego pola w klasie, co ułatwia zarządzanie stanem encji i zapewnia wygodny dostęp do jej pól. Łącząc te adnotacje, tworzymy kompletną i efektywną definicję klasy encji JPA, która jest nie tylko funkcjonalna, ale także przejrzysta i łatwa w utrzymaniu.

Przy użyciu adnotacji @Entity w JPA (Java Persistence API), konieczne jest zawsze zdefiniowanie pola z adnotacją @Id (linia 9). Adnotacja @Id wskazuje, które pole klasy encji jest unikalnym identyfikatorem (kluczem głównym) w mapowanej tabeli bazy danych. Jest to wymagane, aby JPA mogło jednoznacznie identyfikować każdy wiersz (instancję encji) w tabeli.

Adnotacja @GeneratedValue (linia 10) jest używana w połączeniu z @Id do automatycznego generowania wartości klucza głównego encji. Parametr strategy adnotacji @GeneratedValue określa, jak wartość klucza głównego ma być generowana. Jedną z dostępnych strategii generowania jest GenerationType.IDENTITY, która wskazuje, że wartość klucza głównego powinna być generowana przez bazę danych.

@Colum (linia 13) jest używana do określenia, jak pole encji ma być mapowane na kolumnę w tabeli bazy danych. Pozwala ona na precyzyjne określenie szczegółów mapowania, w tym nazwy kolumny, jej długości, czy też unikalności i innych ograniczeń.

Sping Boot obsługuje adnotacje typu @ManyToOne (linia 17) , @OneToMany, @OneToOne i @ManyToMany (linia 57) które opisują w łatwy sposób relacje między tabelami w bazie danych. Przy korzystaniu z nich, za wyjątkiem @ManyToMany, adnotację @Colum zastępowana zostaje przez, @JoinColum która mówi o tym że ta kolumna jest kluczem obcym. W przypadku @ManyToMany reprezentacja w Javie znacznie odbiega od reprezentacji w bazie danych, jest ona bardziej skomplikowana (rys. 5.5). Implementację połączenia typu wiele do wielu składa się z czterech głównych elementów:

1. **Definicja Relacji:** Adnotacja @ManyToMany jest stosowana na polu w obu klasach encji, aby wskazać, że istnieje relacja wiele-do-wielu między nimi.
2. **Tabela Łącząca:** Standardowo, JPA tworzy tabelę łączącą, aby zarządzać relacją wiele-do-wielu. Ta tabela zawiera klucze obce obu encji.
3. **Opcjonalne Modyfikacje Tabeli Łączącej:** Za pomocą adnotacji @JoinTable można dostosować nazwę tabeli łączącej, nazwy kolumn kluczy obcych oraz inne właściwości tabeli łączącej.
4. **Kierunkowość Relacji:** Relacje @ManyToMany mogą być jednokierunkowe lub dwukierunkowe. W relacji dwukierunkowej obie encje mają świadomość swojego powiązania.

Adnotacje @GeneratedValue działają w połączeniu z @PrePersist. Takie połączenie sprawie że wartości oznaczone jako wartości wygenerowane nie muszą być jawnie podawany przy tworzeniu obiektu. Są one generowane i przypisywane w metodzie oznaczonej @PrePersist tuż przed zapisaniem obiektu do bazy danych, co ułatwia tworzenie obiektów których wartości początkowe są z góry znane, Jak na przykład data stworzenia.

Ostatnią adnotacją wykorzystywaną w modelach jest @Enumerated( EnumType.STRING). W tym wypadku oznacza to że pole które w Javie obsługiwane jest jako obiekt enumeratora, w bazie danych powinno być zapisane jako ciąg znaków. Nie zastosowanie adnotacji @Enumerated sprawi że wartości enuma będą zapisane cyfra przedstawiająca pozycję wartości w enumeratorze (rys. 5.6).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68 | @Entity @Table(name = “projects”) @NoArgsConstructor @AllArgsConstructor @Getter @Setter public class Project {   @Id  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  private Long id;   @Column(name = “project\_name", nullable = false,unique = true)  private String projectName;  @ManyToOne  @JoinColumn(name = "responsible\_user", nullable = false)  private User responsibleUser;   @Column(nullable = false)  private Double goal;   @Column(name = "money\_collected",nullable = false)  @GeneratedValue()  private Double moneyCollected;   @Column(name = "number\_of\_supporters",nullable = false)  @GeneratedValue()  private Long numberOfSupporters;   @Column(nullable = false,name = "date\_created")  @GeneratedValue()  private LocalDate dateCreated;   @Column(nullable = false)  @GeneratedValue()  private Long suspicions;   @Column(name = "date\_closed")  private LocalDate dateClosed;   @Column(name = "planed\_date\_of\_closing", nullable = false)  private LocalDate planedDateOfClosing;   @Column(name="banner\_image")  private String bannerURL;   @Column(nullable = false)  private String summery;   @Column(name = "status",nullable = false)  @GeneratedValue()  @Enumerated(EnumType.*STRING*)  private ProjectStatus status;   @ManyToMany(mappedBy = "projects")  private Set<Tag> tags = new HashSet<>();   @PrePersist  public void prePersist() {  dateCreated = LocalDate.*now*();  moneyCollected = 0.00;  numberOfSupporters = 0L;  status = ProjectStatus.*TO\_VERIFY*;  suspicions=0L;  } } |

rys. 5.4 Przykład obiektu reprezentującego wiersz tablicy w bazie danych

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | @Entity @Table(name = "tags") @NoArgsConstructor @AllArgsConstructor @Getter @Setter public class Tag {  @Id  @GeneratedValue(strategy = GenerationType.*AUTO*)  @Column()  private Long id;   @ManyToMany  @JoinTable(  name = "tag\_project",  joinColumns = @JoinColumn(name = "tag\_id"),  inverseJoinColumns = @JoinColumn(name = "project\_id")  )  private Set<Project> projects = new HashSet<>();   @Column(unique=true,nullable = false,name = "tag\_name")  private String tagName; } |

rys. 5.5 Przykład obiektu reprezentującego wiersz tablicy w bazie danych, przybliżenie relacji wiele do wielu

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public enum ProjectStatus {  *OPEN*, *//0  CLOSED*, *//1  EARLY\_CLOSE*, *//2  TO\_VERIFY //3* } |

rys. 5.6 Przykład enumeratora, i numery pozycji poszczególnych elementów

W pakiecie mapper (rys. 5.1) znajdują się mappery których zadaniem jest ułatwić proces transformacji danych między różnymi formatami lub strukturami. Mappery, czyli klasy zawarte w tym pakiecie, są odpowiedzialne za przekształcanie danych z jednego typu obiektu na inny. Jest to szczególnie przydatne w sytuacjach, gdy struktura danych używana wewnętrznie przez aplikację serwerową różni się od formatu, który ma być przesłany do klienta lub otrzymany od klienta.

Mappery w Javie można zrealizować na kilka sposobów; ja zdecydowałem się na wykorzystanie adnotacji @Mapper z biblioteki MapStruct, która służy do automatycznego generowania kodu realizującego mapowanie między różnymi klasami Java. Biblioteka ta umożliwia zdefiniowanie różnych opcji, takich jak componentModel, który pozwala na integrację generowanych mapperów z różnymi frameworkami i kontenerami wstrzykiwania zależności, w tym z Springiem. Dodatkowo, MapStruct oferuje opcję unmappedTargetPolicy, która określa postępowanie w sytuacji, gdy nie wszystkie pola są zmapowane, oraz uses, informującą mapper o konieczności użycia innego mappera, jeśli natrafi na obiekt, który jest już przez niego mapowany.

Adnotacja @Mapping w MapStruct jest używana do definiowania sposobu mapowania poszczególnych pól z obiektów źródłowych na pola w obiektach docelowych. Jest szczególnie przydatna, gdy nazwy pól w mapowanych obiektach różnią się od siebie, wymagane jest niestandardowe mapowanie wartości, lub gdy istnieje potrzeba, aby nie wszystkie pola były mapowane. Ta elastyczność umożliwia precyzyjne i efektywne zarządzanie danymi przekazywanymi między różnymi warstwami aplikacji.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @Mapper(componentModel = "spring",  unmappedTargetPolicy = ReportingPolicy. *WARN,*  uses = {ProjectMapper.class) public interface UserMapper {  UserMapper *INSTANCE* = Mappers.*getMapper*(UserMapper.class);   @Mapping(source = "username", target = "userName")  UserReadModel toDto(User user); |

rys. 5.7 Przykład mappera

Pakiet Repository zawiera repozytoria które są kluczowym elementem architektury aplikacji, umożliwiającym elegancką i efektywną komunikację z bazą danych. W Spring Boot, repozytoria są zazwyczaj interfejsami, które korzystają z Spring Data JPA (Java Persistence API) do zapewnienia abstrakcji nad operacjami bazodanowymi.

Główne funkcje repozytoriów w Spring Boot obejmują:

* **Abstrakcja Operacji Bazodanowych:** Repozytoria ukrywają szczegóły implementacji operacji na bazie danych, takich jak zapytania, wstawianie, aktualizacja i usuwanie, co ułatwia zarządzanie danymi w aplikacji.
* **Automatyczne Generowanie Prostych Metod:** Spring Data JPA oferuje zestaw gotowych metod, takich jak findAll(), findById(), save(), co znacznie przyspiesza pracę z danymi. Te metody są automatycznie dostępne w repozytoriach, które rozszerzają interfejsy Spring Data JPA, np. CrudRepository lub JpaRepository.
* **Definiowanie Własnych Zapytań:** Oprócz standardowych metod, Spring Boot umożliwia tworzenie własnych zapytań SQL lub JPQL. Można to zrobić za pomocą adnotacji @Query, która pozwala na określenie niestandardowego zapytania dla danej metody w repozytorium.
* **Integracja z Mechanizmami Springa:** Repozytoria integrują się z resztą ekosystemu Spring, w tym z transakcjami, cachowaniem i bezpieczeństwem, co zapewnia spójność i niezawodność operacji danych.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public interface PaymentRepository extends JpaRepository<Payment, Long> {  @Query(value = "SELECT \* FROM payments where project\_id = (SELECT project\_id FROM projects WHERE project\_name=?1) and " +  "user\_id=(select user\_id from users where user\_name = ?2)",nativeQuery = true)  List<Payment> getPaymentForProjectByUser(String projectName,String userName);  ); |

rys 5.8 Przykład repozytorium z własnym zapytaniem

Pakiet Error jest przeznaczony do obsługi błędów w aplikacji, zapewniając odpowiednią reakcję na wyjątki i nieoczekiwane sytuacje. Z kolei pakiet Config zawiera klasy konfiguracyjne, takie jak CorsConfiguration, która zarządza zasadami Cross-Origin Resource Sharing (CORS), oraz PayPalConfiguration, odpowiedzialna za konfigurację integracji z PayPal.

Podsumowując, działanie aplikacji serwerowej można opisać w następujący sposób: gdy na jeden z endpointów zdefiniowanych w kontrolerach przychodzi zapytanie spełniające określone kryteria, uruchamiana jest funkcja serwisu, która pobiera i przetwarza dane. Następnie, za pomocą mappera, dane są transformowane na odpowiedni format, a wynik operacji jest zwracany jako odpowiedź na zapytanie do frontendu. Ta struktura pozwala na efektywne i uporządkowane przetwarzanie zapytań, co jest kluczowe dla płynnego działania aplikacji.

**5.1.2 Architektura po stronie frontendu**

Architektura po stronie frontendu jest stosunkowo mniej skomplikowana niż po stronie backendu, jest tak w dużej mierze dzięki narzędziu ng. Ng jest skrótem od "Angular" i odnosi się do narzędzia wiersza poleceń (CLI – Command Line Interface) Angulara, pozwala ono łatwo tworzyć projekty, dodawać do niego nowe elementy oraz symulować lokalny serwer na którym aplikacja może być testowana.

Oto kilka kluczowych funkcji, które ng (Angular CLI) oferuje:

1. **Tworzenie Nowego Projektu:** Polecenie ng new pozwala na stworzenie nowego projektu Angular od podstaw. Generuje ono wszystkie niezbędne pliki, strukturę katalogów oraz konfigurację podstawową.
2. **Generowanie Komponentów i Usług:** Angular CLI umożliwia generowanie różnych części aplikacji, takich jak komponenty (ng generate component), usługi (ng generate service), dyrektywy, moduły itp., co znacząco przyspiesza proces tworzenia aplikacji.
3. **Budowanie i Uruchamianie Aplikacji:** Za pomocą polecenia ng serve, można uruchomić lokalny serwer deweloperski, który automatycznie odświeża aplikację przy zmianie kodu. Polecenie ng build służy do budowania aplikacji

Budowa projektu w Anularze dzieli się w dużej mierze na dwie części, część konfiguracyjną i część deweloperską.

Część konfiguracyjna obejmuje ustawienia narzędzi takich jak Webpack i TypeScript, które są kluczowe dla działania aplikacji. W przypadku Angulara, wiele z tych konfiguracji jest zarządzanych automatycznie przez Angular CLI, co znacznie ułatwia pracę programisty.

Część deweloperska koncentruje się na tworzeniu komponentów, serwisów, modułów i innych artefaktów, które razem tworzą aplikację. Komponenty w Angularze zawierają logikę, szablony i style, które definiują, jak poszczególne części aplikacji są wyświetlane i jak działają. Serwisy umożliwiają współdzielenie logiki biznesowej między komponentami. Moduły zaś grupują powiązane funkcjonalności i pomagają w organizacji aplikacji.

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

rys 5.9 Struktura projektu w Angularze

W projektach Angular, część konfiguracyjna jest zarządzana głównie przez kilka kluczowych plików. Oto te, które zwykle odgrywają najważniejszą rolę w projekcie:

* **angular.json**: Ten plik jest centralnym miejscem konfiguracji dla Angular CLI. Określa szczegóły projektu, opcje budowania, testowania, oraz inne konfiguracje związane z różnymi aspektami projektu Angular.
* **package.json**: Ten plik należy do Node.js i jest używany do określania zależności projektu, skryptów uruchamiających, wersji projektu i innych metadanych. Zawiera również zależności od bibliotek używanych w projekcie.
* **tsconfig.json**: Jest to plik konfiguracyjny TypeScript, który określa, jak kompilator TypeScript ma przetwarzać pliki TS. Zawiera opcje takie jak target ECMAScript, moduły, ścieżki, opcje ścisłości i inne.
* **src/environments/environment.ts**: W tych plikach znajdują się konfiguracje specyficzne dla środowiska, takie jak URL do API w produkcji i w środowisku deweloperskim. Pozwalają one na łatwe przełączanie między różnymi ustawieniami w zależności od środowiska, w którym działa aplikacja.

W projekcie Angular, od strony developerskiej, istotną rolę odgrywają pliki takie jak app.module.ts – główny moduł aplikacji, który grupuje komponenty, serwisy, filtry, i inne zależności, oraz app-routing.module.ts, który definiuje trasy nawigacji w aplikacji (rys. 5.10). Moduł routingu pozwala na konfigurację ścieżek dostępu do różnych komponentów i obsługę nawigacji w aplikacji jednostronicowej (SPA).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | const routes: Routes = [    {path:'home', component:HomePageComponent},    {path:'', redirectTo: '/home', pathMatch: 'full' },    {path:'login', component:LoginPageComponent},    {path:'register', component:RegisterPageComponent},    {path:'register/admin', component:RegisterPageComponent,canActivate:[UserGuardService]},    {path:'createProject',component:CreateProjectComponent,canActivate:[UserGuardService]},    {path:'search',component:SearchProjectComponent},    {path:'list/suspicion',component:SearchProjectComponent,canActivate:[UserGuardService]},    {path:'list/verify',component:SearchProjectComponent,canActivate:[UserGuardService]},    {path:'list/user/:userName',component:SearchProjectComponent,canActivate:[UserGuardService]},    {path:'project/:projectName',component:SpecificProjectComponent},    {path:'profile/:userName',component:ProfileComponent,canActivate:[UserGuardService]},    {path:'payment/:status/:projectName/:amount',component:PaymentComponent},    { path: '\*\*', component: Page404Component },  ]; |

rys. 5.10 Przykład nawigacji w aplikacji Angular

Nawigacja w app-routing.module.ts w Angularze jest kluczowa dla zarządzania przepływem użytkownika w aplikacji. Plik ten zawiera definicje tras, składające się z ścieżek i odpowiadających im komponentów. Na przykład, ścieżka path: 'project/:projectName' w linii 19 mapuje na konkretny komponent, component:SpecificProjectComponent, który odpowiada za wyświetlanie odpowiedniej treści HTML. Ścieżki mogą zawierać zmienne parametry, oznaczone znakiem dwukropka (:), co pozwala na dynamiczne dostosowanie treści wyświetlanych w komponencie. Dodatkowo, wiele ścieżek może prowadzić do tego samego komponentu który zmienia swoje działanie w zależności od ścieżki, umożliwiając wielokrotne wykorzystanie kodu i zmniejszenie liczby plików w projekcie.

Nowo stworzone komponenty w Angularze zazwyczaj składają się z trzech głównych plików, ts, css i html.

Plik ts (TypeScript) zawiera logikę komponentu, w tym jego właściwości, metody i dekoratory, takie jak @Component, który pełni kluczową rolę w definiowaniu właściwości i zachowania komponentu. Dekorator @Component jest funkcją, która przyjmuje konfigurację w formie obiektu i przekształca daną klasę TypeScript w komponent Angulara.

W konfiguracji @Component możemy określić kilka istotnych właściwości:

* **selector:** Określa niestandardowy tag HTML, który będzie używany do umieszczania komponentu w szablonach. Na przykład, selector: 'app-my-component' pozwala na użycie <app-my-component></app-my-component> w szablonach HTML.
* **templateUrl:** Ścieżka do pliku HTML, który zawiera szablon komponentu. Ten szablon definiuje strukturę i układ elementów UI, które będą wyświetlane przez komponent.
* **styleUrls:** Tablica ścieżek do plików CSS, które zawierają style dla komponentu. Stylowanie może być również zdefiniowane bezpośrednio w dekoratorze za pomocą właściwości styles.

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Component({    selector: 'app-home-page',    templateUrl: './home-page.component.html',    styleUrls: ['./home-page.component.css']  }) |

rys 5.11 Przykład adnotacji @Component w pliku ts komponentu

Dodatkową zaletą Angulara jest łatwe wykorzystanie tzw. "web hooków" (częściej nazywanych "lifecycle hooks") w plikach .ts (TypeScript) komponentów. Lifecycle hooks to specjalne metody, które pozwalają na interakcję z różnymi etapami cyklu życia komponentu lub dyrektywy. Dzięki nim można kontrolować i reagować na zmiany w aplikacji w odpowiednich momentach. Oto niektóre z najważniejszych hooks dostępnych w Angularze:

* **ngOnInit:** Wywoływany po utworzeniu komponentu i inicjalizacji wejść (@Input). Jest to idealne miejsce do inicjalizacji danych komponentu.
* **ngOnChanges:** Uruchamiany za każdym razem, gdy zmieniają się dane wejściowe komponentu. Przydatny do reagowania na zmiany z zewnątrz.
* **ngDoCheck:** Wywoływany przy każdej wykrytej zmianie detekcji, pozwala na ręczną kontrolę nad zmianami.
* **ngAfterViewInit:** Wywoływany po zainicjowaniu widoku komponentu i jego dzieci. Używany do manipulacji elementami DOM.
* **ngOnDestroy:** Uruchamiany tuż przed zniszczeniem komponentu. Idealny do czyszczenia zasobów, takich jak subskrypcje czy timery.

Plik CSS w komponentach Angulara odpowiada za stylizowanie wyświetlanego HTML, nadając wygląd i układ elementów interfejsu użytkownika. Natomiast plik HTML jest odpowiedzialny za strukturę i zawartość, które mają zostać wyświetlone. Angular, dzięki swojej zaawansowanej integracji, oferuje dodatkowe możliwości w plikach HTML poprzez dyrektywy i bindingi.

Jedną z takich funkcji jest dyrektywa \*ngFor, która jest przykładem pętli używanej w Angularze do renderowania listy elementów. Pozwala ona na iterację przez kolekcję danych (np. array) i generowanie powtarzających się elementów HTML dla każdego obiektu w kolekcji. Na przykład, \*ngFor="let item of items" stworzy blok HTML dla każdego elementu w tablicy items.

Inne przydatne dyrektywy to:

* \*ngIf: Służy do warunkowego renderowania elementów w zależności od wartości wyrażenia logicznego.
* [ngClass] i [ngStyle]: Umożliwiają dynamiczne przypisywanie klas CSS i stylów do elementów na podstawie warunków logicznych.
* [(ngModel)]: Zapewnia dwustronne wiązanie danych, pozwalając na synchronizację danych między modelem a widokiem.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <div class="project-banner">  <div \*ngIf="loading" class="loading-image">    <img    mat-card-image     class="resized-banner-image"     src="assets/No-Image-Placeholder.png"     alt="Descriptive Image Text"    />   </div>   <div \*ngIf="!loading">    <img     mat-card-image     class="resized-banner-image"     src="assets/{{ currentItem1.responsibleUser.userName }}/{{                    currentItem1.projectName                  }}/banner/{{ currentItem1.bannerURL }}"     alt="Banner Image Text"    />  </div>  </div> |

rys. 5.12 Przykład wykorzystanie ngIf w kodzie HTML

Poza komponentami, istotną część aplikacji Angular stanowią również serwisy. Podobnie jak w części serwerowej, serwisy w Angularze przechowują metody, które są używane w całym projekcie. Większość z tych serwisów to funkcje odpowiedzialne za komunikację z serwerem: wysyłają zapytania do serwera i obsługują odpowiedzi.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | @Injectable({    providedIn: 'root'  })  export class ProjectService {    constructor(private http:HttpClient) { }    public addProject(project: Project): Observable<Project> {      const storedToken = localStorage.getItem('token');      const headers = new HttpHeaders({        Authorization: `Bearer ${storedToken}`,      });      return this.http.post<Project>(`${environment.apiBaseUrl}/project`, project, { headers })        .pipe(          catchError(this.handleError)        );    } |

rys. 5.13 Przykład serwisu i jednej funkcji komunikującej się z serwerem

**5.2 Bezpieczeństwo aplikacji**

**5.2.1 Json Web Token**

SON Web Token (JWT) to kompaktowy i bezpieczny sposób na przekazywanie informacji między stronami jako obiekt JavaScript Object Notation (JSON), który jest szeroko stosowany w uwierzytelnianiu i autoryzacji w aplikacjach internetowych. JWT składa się z trzech kluczowych części: nagłówka (header), ładunku (payload) i podpisu (signature).

* **Nagłówek:** Ta część zawiera informacje o typie tokena, czyli JWT, oraz o algorytmie kryptograficznym używanym do jego podpisu. W projekcie użyto algorytmu HMAC (Hash-based Message Authentication Code) z użyciem SHA-256 (Secure Hash Algorithm 256-bit).
* **Ładunek:** Zawiera rzeczywiste dane, takie jak uprawnienia użytkownika, jego identyfikator czy informacje o czasie wygaśnięcia tokena. Dane te są zazwyczaj prezentowane w formacie Claims (zastrzeżenia), które umożliwiają przekazywanie ważnych informacji.
* **Podpis:** Jest tworzony przez zastosowanie wybranego algorytmu kryptograficznego do zakodowania nagłówka i ładunku wraz z tajnym kluczem. Tajny klucz jest znany tylko serwerowi, co zapewnia, że token nie został zmieniony podczas transmisji.

JWT jest używany w autoryzacji, ponieważ po zalogowaniu się użytkownika, serwer generuje token JWT i wysyła go do klienta. Następnie klient przesyła ten token w nagłówkach HTTP (Hypertext Transfer Protocol) przy każdym zapytaniu, umożliwiając serwerowi weryfikację tożsamości użytkownika i uprawnień dostępu do zasobów. Główną zaletą JWT jest jego samowystarczalność - nie wymaga dodatkowego przechowywania stanu sesji na serwerze.

W aplikacji JWT jest wykorzystywany nie tylko do autoryzacji, ale również do przesyłania innych informacji, takich jak nazwa użytkownika czy jego typ. Umieszczenie tych danych w tokenie zwiększa bezpieczeństwo i ułatwia ich przetwarzanie po stronie serwera.

Dodatkowo, zastosowano ograniczony czas ważności tokenu, aby w przypadku jego wycieku, był on ważny tylko przez określony czas lub do momentu ponownego zalogowania się użytkownika.

**5.2.2 Mechanizm ról użytkownika**

W celu odpowiedniego podzielenia funkcjonalności aplikacji zastosowałem mechanizm ról użytkowników. Jak było powiedziane w 3.1 użytkownik może nie mieć roli, lub jedną z trzech dostępnych w serwisie (USER, ADMIN i SUPER). Role te nie są jawnie podawane przy rejestracji użytkownika, lecz są ustawiane po stronie serwerowej zelżenie od kontekstu w jakim dany endpoint został wywołany, a użytkownik SUPER jest tworzony tylko raz przy pierwszym uruchomieniu aplikacji dzięki czemu nie da się w normalny sposób stworzyć drugiego takiego konta. Raz przypisana nie możliwa jest też jej zmiana bez bezpośredniej ingerencji w bazę danych.

Zależnie od tego jaka rola jest przypisana do konta dostępna jest inna funkcjonalność, jak w przypadku pobierania listy wszystkich projektów o statusie OPEN lub CLOSED (rys. 5.14). Jeśli zapytanie wysłał normalny użytkownik dostanie on listę tylko i wyłącznie swoich projektów, natomiast jeśli zapytanie wysłał administrator serwisu, ponieważ nie mogą oni tworzyć własnych projektów, w odpowiedzi dostanie listę wszystkich takich zbiórek w bazie danych.

|  |
| --- |
| public List<ProjectSLElement> getOpenAndClosedProjects(String header){  User user = userService.getUserEntityByToken(header);  List<Project> projects = new ArrayList<>();  if(user.getRole()== Role.*USER*){  projects = projectRepository.getOpenAndClosedProjectsForUser(user.getUsername());  } else if (user.getRole()==Role.*ADMIN*) {  projects = projectRepository.getOpenAndClosedProjects();  }  return projectMapper.mapSLE(projects); } |

rys. 5.14 Przykład zmiennej funkcjonalności zależnie od typu użytkownika

Podobnie w przypadku interfejsu użytkownika, jeśli zalogowany użytkownik chciałby wpłacić na cel konkretnego projektu zostanie uruchomiony cały mechanizm obsługi wpłaty, jeśli natomiast użytkownik nie jest zalogowany zostanie od przeniesiony na stronę do zalogowania się (rys. 5.15).

|  |
| --- |
| support(){      if(this.role=='USER'){        const dialogRef = this.dialog.open(PopupComponent, {          width: '500px',          data: { myParam: 'SP' }        });          dialogRef.afterClosed().subscribe(result => {          let ammount:number=result          let payment:Payment={            userName:this.user!.userName,            projectName:this.projectName,            money:ammount          }          this.payPalService.initPayPalPayment(ammount,payment).subscribe(            (resp:PayPal)=>{              window.location.href = resp.approvalUrl            }          )        });      }      else{        this.router.navigate(['login']);      }    } |

rys. 5.15 Przykład funkcjonalności zależnej od typu użytkownika w frontedzie

**5.2.3 Mechanizm canActivate w Angular**

Jedną z zalet Angulara jest wbudowany mechanizm strażnika ścieżki (Router Guard). Jest to funkcjonalność dzięki której przed wykonaniem przejścia z jednego adresu na drugi jesteśmy w stanie sprawdzić czy takie przejście ma prawo się wykonać. W tym celu do dyspozycji mamy sześć opcje strażników drogi.

1. **canActivate:** Decyduje, czy użytkownik może przejść do danej ścieżki, np. na podstawie statusu zalogowania.
2. **canActivateChild:** Stosowany dla ścieżek podrzędnych, umożliwiając wspólną kontrolę dostępu dla całego segmentu ścieżki.
3. **canDeactivate:** Sprawdza, czy użytkownik może opuścić stronę, np. po niewypełnieniu formularza.
4. **canLoad:** Kontroluje, czy można załadować moduł leniwego ładowania, na przykład w celu ochrony dostępu do modułu.
5. **resolve:** Pobiera dane niezbędne do wyświetlenia strony przed jej załadowaniem.
6. **canMatch:** Używany do zaawansowanego dopasowywania ścieżek, uwzględniający asynchroniczne operacje.

W projekcie został wykorzystany wyłącznie canActivate do sprawdzenia czy użytkownik który chce dostać się pod konkretny adres jest zalogowany, nie wszystkie adresy wymagają takiego sprawdzenia.

|  |
| --- |
| {path:'createProject',component:CreateProjectComponent,canActivate:[UserGuardService]} |

rys. 5.16 Przykład połączenia ścieżki z konkretnym strażnikiem ścieżki

|  |
| --- |
| export class UserGuardService implements CanActivate {      constructor(      private authService: AuthenticationService,      private router: Router    ) {}    canActivate(      route: ActivatedRouteSnapshot,      state: RouterStateSnapshot    ): Observable<boolean> | boolean {      return this.authService.testUserByHeaderInStorage().pipe(        map(response => {          if (response) {            console.log("Guard OK");            return true;          } else {            console.log("Guard Bad");            this.router.navigate(['/home']);            return false;          }        }),        catchError((error) => {          console.log("Error occurred", error);          this.router.navigate(['/home']);          return of(false);        })      );    }  } |

rys. 5.17 Przykład implementacji strażnika ścieżki sprawdzającego czy użytkownik jest poprawnie zalogowany

**5.2.4 Obsługa błędu 404**

Aby zapewnić bezpieczeństwo i klarowność nawigacji w serwisie, wprowadzono mechanizm domyślnego przekierowania na stronę błędu 404. Ta funkcjonalność jest aktywowana w sytuacji, kiedy żądany adres URL nie odpowiada żadnej z zdefiniowanych wcześniej ścieżek. Ważne jest umieszczenie takiego przekierowania na samym końcu listy ścieżek, ponieważ system sprawdza je sekwencyjnie, od pierwszej do ostatniej. Jeżeli przekierowanie na stronę błędu 404 zostanie zdefiniowane przed innymi ścieżkami, uniemożliwi to dostęp do nich, przechwytując wszystkie żądania przejścia.

|  |
| --- |
| { path: '\*\*', component: Page404Component }, |

rys. 5.18 Domyślne przekierowanie w wypadku braku dopasowania ścieżki

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

rys. 5.19 Strona błędu 404

**5.3 Schemat bazy danych**

**5.4 Wykorzystane biblioteki**

**5.3.1 Backend**

**5.3.2 Frontend**

**5.5 Istotne fragmenty kodu**

**5.4.1 Backend**

**5.4.2 Frontend**

**6 Weryfikacja i walidacja poprawności działania**

**6.1 Sposób testowania**

**6.2 Zakres testowania**

**6.3 Przypadki testowe**

**6.4 Wykryte i usunięte błędy**

**7 Podsumowanie i wnioski**

**Bibliografia**

**Spis skrótów i symboli**

**Źródła**

**Lista dodatkowych plików, uzupełniających tekst pracy**

**Spis rysunków**