

**AKADEMIA MARYNARKI WOJENNEJ**

**im. BOHATERÓW WESTERPLATTE**

**Wydział Mechaniczno-Elektryczny**

Institut Elektroniki i Automatyki Okrętowej

Katedra Informatyki

## **PRACA DYPLOMOWA**

### **INŻYNIERSKA**

Temat:

Wykonawca: Adam SZREIBER

Kierownik pracy

dr inż. Jan Masiejczyk

Kierownik katedry

kmdr dr hab. inż. Andrzej Żak

prof. nadzw. AMW

Ocena pracy dyplomowej

.....  
słownie

.....  
data i podpis  
Przewodniczącego Komisji Egzaminacyjnej



Gdynia, dnia ..... r.

Adam Szeiber

21662

### O Ś W I A D C Z E N I E

Oświadczam, że przedłożoną do egzaminu pracę dyplomową pt.

.....  
.....

kończącą studia I stopnia napisałem samodzielnie. Przy wykonywaniu pracy nie zlecałem jej opracowania ani żadnej jej części innym osobom, jak też nie skopiowałem cudzych opracowań

i przestrzegałem postanowień Ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. z 2018 r. poz. 1191 z późn. zmianami).

Ponadto oświadczam, iż treści zaczerpnięte z literatury przedmiotu są oznaczone w tekście oraz w przypisach, w sposób ogólnie przyjęty dla prac naukowych.

Jednocześnie przyjmuję do wiadomości, że gdyby powyższe oświadczenie okazało się nieprawdziwe, jestem świadomy(a) zasadności cofnięcia decyzji o wydaniu mi dyplomu.

Wyrażam zgodę na udostępnienie mojej pracy dyplomowej czytelnikom.

Przekazuję moją pracę dyplomową do Ogólnego Repozytorium Prac Dyplomowych.

.....  
podpis osoby składającej oświadczenie



## Spis treści

1.	Wykaz skrótów i oznaczeń .....	5
2.	Wprowadzenie .....	6
2.1	Komercyjne aplikacje.....	6
2.2	Cel pracy .....	6
2.3	Konkurencyjne rozwiązania na rynku.....	8
2.4	Wady i zalety feature-flag .....	9
3.	Budowa Aplikacji .....	10
3.1	Określenie wymagań .....	11
3.1.1	Wymagania funkcjonalne .....	11
3.1.2	Wymagania нефункционалне .....	11
3.2	Analiza i Projektowanie .....	11
3.2.1	Wstęp .....	12
3.2.2	Technologie i biblioteki użyte do realizacji projektu .....	12
3.3	Implementacja.....	16
3.3.1	Aplikacja internetowa.....	16
3.3.2	Biblioteka.....	30
3.4	Testy .....	32
3.5	Eksploatacja.....	33
3.5.1	Proces tworzenia funkcjonalności .....	33
3.5.2	Proces implementacji .....	33
3.5.3	Przykład implementacji .....	33
4.	Podsumowanie .....	35
4.1	Wnioski .....	35
4.2	Możliwe ścieżki rozwoju aplikacji .....	36
5.	Literatura / Załączniki .....	37
6.	Streszczenie .....	38

## 1. Wykaz skrótów i oznaczeń

JavaScript	
TypeScript	

## 2. Wprowadzenie

### 2.1 Komercyjne aplikacje

Firmy informatyczne wytwarzające oprogramowanie pracują nad swoimi aplikacjami przez kilkadziesiąt lat. Ich kod źródłowy może liczyć wiele milionów linii kodu, zawarty w tysiącach plików. Starają się zapewnić funkcjonalności jakie oczekują od nich użytkownicy, usprawniają to co kiedyś zostało zrobione lub wprowadzają nowe funkcje, tylko po to, aby ich produkt był konkurencyjny na rynku, pozyskać nowych użytkowników, utrzymać dotychczasowych dużych, stałych klientów. Na przestrzeni wielu lat struktura, organizacja i pracownicy firmy mogą się zmieniać, a jakość wytwarzanego kodu degraduje się. Aby temu zapobiec każda firma posiada swój standard kodowania (eng. coding standard) – którego muszą przestrzegać pracownicy firmy. Pomaga to w zachowaniu ogólnego porządku w kodzie programu, ułatwia nowym pracownikom wdrożenie/zapoznanie się z nim, ułatwia przeprowadzanie inspekcji kodu (en. Code review), ułatwia wyszukiwanie i naprawę błędów. Stosuje się również narzędzia typu SOLARLINT – który kontroluje jakość kodu, a w razie niedostosowania się do zadanych restrykcji informuje o tym na przykład, poprzez podkreślenie źle napisanego kodu i wyświetlenie dokładnego błędu. Firmy stosują również pryncypia które narzucone przez sam język programowania w którym tworzą swoje systemy. Dlatego tak ważne jest aby styl kodu danego programu był spójny, czytelny i otwarty na rozbudowę.

Wprowadzanie zmian i rozwój produktu jest kluczowym elementem aby zapewnić firmie byt na dynamicznie zmieniającym się rynku oprogramowania. Lecz nie należy to do najłatwiejszych rzeczy – gdy pracujemy nad kodem napisanym przez innego dewelopera lub który jest bardzo przestarzały – napisany przed wprowadzeniem wewnętrznych firmowych kryteriów kodowania. Aby wprowadzić jakiegokolwiek zmiany, najpierw musimy poświęcić wiele godzin aby znaleźć odpowiednie miejsce w kodzie, następnie dokładnie przeanalizować dany fragment kodu, opracować plan wdrożenia nowej funkcjonalności, i ostatecznie przejść do implementacji. Tak złożony proces może przysporzyć wiele trudności, szczególnie tym mniej doświadczonym programistom.

### 2.2 Cel pracy

*„Czasu nie cofniesz, ale błędy możesz naprawić.”*

Bezpieczeństwo programu to sprawa najwyższej wagi. Firmy budują swoje zaufanie wśród klientów przez dziesiątki lat, jeden krytyczny błąd może spowodować katastrofalne skutki.

Istnieją dwa kryteria oceny błędu, pierwszy z nich to – ważność. Określa jak duży wpływ na system ma dany defekt i jakie konsekwencje za sobą niesie. Wyróżniamy:

- błąd krytyczny – uniemożliwia korzystania podstawowych funkcji systemu
- błąd pilny - znacząco utrudnia korzystanie z systemu, istnieje możliwość obejścia błędu w celu osiągnięcia oczekiwanego wyniku. Jako przykład może posłużyć alternatywna metoda płatności w sklepie internetowym.
- błąd normalny – utrudnia korzystanie, nie mniej jednak system wciąż jest w pełni funkcjonalny
- błąd mało istotny – zazwyczaj literówki, źle dobrany kolor

Serwis Beheer (słowo pochodzi z języka holenderskiego i oznacza „zarządzanie”) to projekt który ma wspomóc programistów bezpiecznie wprowadzać zmiany w kodzie programu poprzez mechanizm flag (ang. feature-flag), automatyczne wyłączanie flag gdy wystąpi błąd wykonywania kodu, wykresy i statystyki. Współczesne programy są niezwykle rozbudowane i nie sposób prześledzić wszystkich

**Komentarz [S1]:** [Jak obsługiwać i zgłaszać błędy w web aplikacji?](#)  
(gogomedia.pl)

ścieżek wykonywanego kodu, co za tym idzie, zmiany w jednym obszarze aplikacji mogą spowodować wystąpienie w innym module aplikacji. Szybkie reagowanie na takie zachowania mogą uratować firmę. Skutki wystąpienia błędów krytycznych niosą za sobą poważne konsekwencje - w najgorszym przypadku spowodują zamknięcie firmy, sprawy karne, czy zadość uczynienie. Z historii znamy już kilka takich przypadków:

- Między 1985 a 1987 rokiem 6 osób uległo poparzeniu w wyniku naświetlań maszyną Therac-25. Była to maszyna stosowana w latach 80 do [radioterapii](#) nowotworów. Trzy z nich zmarły w następstwie wypadku. A oto przypadek jednego z pacjentów. W 1985 r. – jedna z maszyn uległa awarii, wyświetlając komunikat o błędzie i niepodjęciu naświetlania. Operator, przyzwyczajony do humorów urządzenia, wymusił wykonanie procedury. Maszyna pięciokrotnie podejmowała próbę wykonania naświetlenia, po czym zupełnie odmówiła posłuszeństwa. 3 miesiące później pacjent, który brał udział w zabiegu, zmarł w związku z powikłaniami napromieniowania.

**Komentarz [S2]:** [4 największe katastrofy programistyczne w historii - drobne błędy o tragicznych konsekwencjach \(wp.pl\)](#)

- 15 stycznia 1990 r. ponad 60 tys. klientów AT&T nie mogło zestawić połączeń międzymiastowych. Przyczyną była awaria oprogramowania w przełącznikach 4ESS. Miesiąc przed awarią AT&T postanowiło przyspieszyć ten proces i zmieniło nieco parametry. Ponieważ reakcja była zbyt szybka, druga wiadomość trafiała podczas restartu przełącznika, zatem oprogramowanie stwierdzało awarię, wystawiało sygnał przepełnienia i wykonywało restart. Skutkiem błędów była lawina restartów i odmowa obsługi.

**Komentarz [S3]:** [Najgorsze skutki błędów programistów - Computerworld - Wiadomości IT, biznes IT, praca w IT, konferencje](#)



## 2.3 Konkurencyjne rozwiązania na rynku

Feature toggle – launchdarkly

[Feature Flags: Faster software deployment and safer code releases | Split](#)

[The Top 6 Feature Flag Management Tools | Harness](#)

[Do czego przydaje się technika Feature Flag \(bulldogjob.pl\)](#)

Zgłębiając temat feature-flag można znaleźć wiele komercyjnych produktów na rynku. Niektóre z nich bardzo zaawansowane, zawierające [...] złożone wykresy, możliwość przypisywania różnych typów danych do flag, czy serwowania różnych wartości flagi na podstawie geolokalizacji.

## 2.4 Wady i zalety feature-flag

### ZALETY:

Łatwość przywrócenia poprzedniej ścieżki kodu bez ingerencji w kod

Możliwość automatycznego wyłączania flagi gdy wystąpi błąd

Możliwość serwowania różnych typów flag – wartość string, boolean, integer

Możliwość wprowadzania większej grupy zmian, kontrolowanej za pomocą jednej flagi

### WADY:

Brak połączenia z Internetem spowoduje wykonywanie starej ścieżki programu

Spowolnienie wykonywania programu – komunikacja przez Ethernet jest stosunkowo wolna

Testowanie aplikacji staje się trudniejsze

Brak możliwości wykorzystania flag w aplikacjach wymagających wysokiej wydajności

### 3. Budowa Aplikacji

Proces wytwarzania aplikacji jest złożony. Na początku ery komputerów, gdy stacje robocze były słabe/wolne i ograniczone przez zasoby, programy pisało się i konserwowało bez większych trudności. Na **przełomie lat 50/60 XIX** wieku rozwój technologiczny przyspieszył do takiego stopnia, że zaczęto tworzyć coraz bardziej skomplikowane oprogramowanie, którego implementacja i utrzymanie zaczęło sprawiać trudności.

**Komentarz [AS4]:** [Inżynieria oprogramowania – Wikipedia, wolna encyklopedia](#)

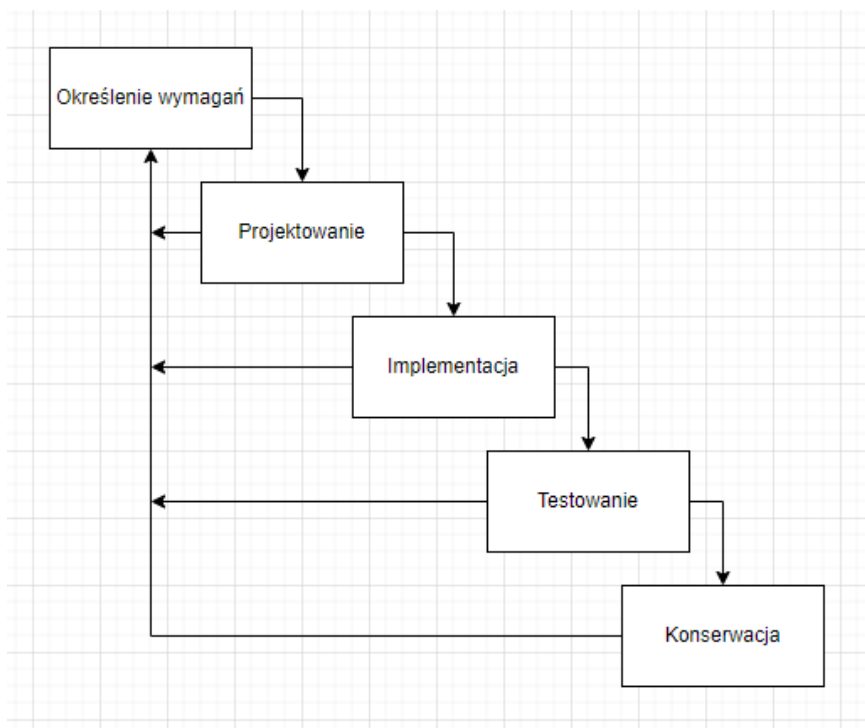
W latach 1968 i 1969 miały miejsce dwie konferencje w Rzymie i Garmich na których po raz pierwszy wprowadzono termin „INŻYNIERIA OPROGRAMOWANIA” (ang. Software engineering, SE) dyscyplina informatyczna stosująca podejście inżynierskie do tworzenia oprogramowania: od analizy i określenia wymagań, przez projektowanie i wdrożenie, aż do ewolucji gotowego oprogramowania.”

Inżynieria oprogramowania zajmuje się metodami wytwarzania, oceniania i utrzymywania oprogramowania systemów komputerowych oraz metodami zarządzania realizacją projektów informatycznych. Celem stosowania tych metod jest zapewnienie wysokiej jakości oprogramowania oraz doprowadzenie do terminowej i zgodnej z budżetem realizacji projektu. Znaczenie metod inżynierii oprogramowania rośnie wraz z wielkością projektu

**Komentarz [AS5]:** [Sacha K. - Inżynieria oprogramowania.pdf](#)

Dziedzina ta wprowadza wiele procesów (Proces jest zbiorem aktywności wykonywanych z myślą o osiągnięciu pewnego celu) które należy dobrać indywidualnie do tworzonego projektu. Do realizacji niniejszego projektu użyto modelu kaskadowego (eng. Waterfall).

**Komentarz [AS6]:** [Inżynieria oprogramowania.pdf](#)



Jest to szeroko stosowany model charakteryzującym się jasno uporządkowanym przebiegiem pracy, brakiem powtarzania aktywności oraz posiada wbudowany mechanizm weryfikacji wyników każdej z faz. Każda faza następuje po sobie, a błąd w jakiegokolwiek faz jest bardzo kosztowne w usunięciu, ponieważ skutkuje powrotem do pierwszej fazy czyli określenia wymagań.

### 3.1 Określenie wymagań

<<<<<CO TO JEST>>>>>

#### 3.1.1 Wymagania funkcjonalne

**Komentarz [S7]:** Tylko wspomnieć że się szło tą drogą

NAZWA FUNKCJI	
OPIS	
DANE WEJŚCIOWE	
POWÓD	

NAZWA FUNKCJI	Rejestracja
OPIS	Funkcja pozwala stworzyć konto nowe użytkownika
DANE WEJŚCIOWE	Adres email, hasło
POWÓD	Aby korzystać z serwisu Beheer użytkownik musi posiadać konto

NAZWA FUNKCJI	Logowanie
OPIS	Funkcja pozwala określić tożsamość użytkownika i przydzielić mu odpowiednie zasoby
DANE WEJŚCIOWE	Adres email, hasło
POWÓD	Aby móc korzystać z serwisu należy być zalogowanym

#### 3.1.2 Wymagania niefunkcjonalne

- Aplikacja ma być dostępna 24/7
- Wygląd aplikacji powinien być przyjazny użytkownikowi
- Wdrożenie w system nie powinien trwać więcej niż 15min
- Interfejs graficzny powinien być wyraźny i czytelny
- Interfejs graficzny powinien być w stonowanych kolorach
- Interfejs graficzny powinien mieć ciemną szatę graficzną
- Informacje o błędach powinny być umieszczone w widocznym dla użytkownika miejscu
- Stworzenie nowej flagi nie powinno zajmować więcej niż 1min
- Hasła w bazie danych powinny być przechowywane w bezpieczny sposób

## 3.2 Analiza i Projektowanie

<<<<<Trochę o UML>>>>>

### 3.2.1 Wstęp

**Program komputerowy** to sekwencja instrukcji składająca się z deklaracji i instrukcji, która jest zgodna z zasadami określonego języka programowania w celu przetworzenia i rozwiązywania pewnych funkcji, zadań lub problemów za pomocą komputera.

Program realizowany jest jako sekwencja maszynowa, tj. polecenia procesora przetwarzane są przez procesor lub procesory komputera, które są następnie wykonywane. Przez program komputerowy rozumie się również kod źródłowy programu, z którego w trakcie tworzenia oprogramowania tworzony jest plik wykonywalny.

Proces tworzenia programu komputerowego nazywamy programowaniem. Programista do procesu wytwarzania programu komputerowego używa specjalistycznych narzędzi jakimi są języki programowania.

Na świecie istnieją tysiące języków programowania i każdego roku powstają nowe. Od języków naturalnych odróżniają się wysoką precyzją oraz jednoznacznością. Człowiek podczas komunikacji między sobą stale popełnia niewielkie błędy lub pozostawia niedomówienia wiedząc, że drugi rozmówca najczęściej go zrozumie. Maszyny wykonują zadania dokładnie, dlatego każdą czynność trzeba opisać ściśle krok po kroku, ponieważ komputer nie potrafi domyślić się, co programista miał na myśli.

Dla programisty język programowania to swoiste narzędzie do komunikacji z komputerem. Dobierając język programowania powinniśmy zwrócić uwagę na takie aspekty jak: wielkość i złożoność projektu, szybkość działania, czas życia projektu (eng. Project Lifetime), niezawodność, doświadczenie programisty, docelowy system operacyjny. Tak więc nie ma jednego idealnego języka – warto, by programista znał ich kilka i zgodnie z zasadą – „używaj odpowiednich narzędzi do danego zadania” (eng. „use the right tool for the right job”) oraz własnymi preferencjami wybierał język adekwatny do danego zadania. Zazwyczaj doświadczeni programiści to poligłoci, bardzo dobrze znają i posługują się wieloma językami programowania. W większości języków programowania występują te same podstawowe mechanizmy, których używa się w bardzo podobny sposób niezależnie od wybranego języka programowania, a różnice często sprowadzają się wyłącznie do nazw bibliotek, funkcji, klas itp.

Doskonałym tego przykładem jest niniejsza praca w której wykorzystano dwa języki programowania, C# w którym wykonano bibliotekę do wprowadzania nowych funkcjonalności, oraz TypeScript w którym wykonano serwis internetowy Beheer.

### 3.2.2 Technologie i biblioteki użyte do realizacji projektu

**Technologie:**

- **PostgreSQL**

Darmowa, open-source relacyjna baza danych znajdująca się w **top 4** najczęściej wybieranych baz danych. Do głównych jego zalet należą

**Komentarz [AS8]:** [Co to znaczy Program komputerowy? Definicja, co oznacza \(korektortekstu.pl\)](#)

**Komentarz [AS9]:** [Język programowania - Szkolnictwo.pl](#)

**Komentarz [AS10]:** Gynveal

**Komentarz [S11]:** Przegląd, i dlaczego wybrałem, to że pracowałem i się w tym czuje

**Komentarz [AS12]:** [PostgreSQL baza danych roku 2020 według DB-Engines po raz kolejny \(linuxpolska.pl\)](#)

**Komentarz [AS13]:** [DB-Engines Ranking - popularity ranking of database management systems](#)

Jak oceniano systemy bazodanowe w tym rankingu?

Podstawą była aktualna popularność mierzona:

- ilością rezultatów w wyszukiwarkach internetowych (Google i Bing),
  - ogólnym zainteresowaniem systemem mierzone w Google Trends,
  - częstością występowania w dyskusjach technicznych (Stack Overflow i DBA Stack Exchange),
  - liczbą ofert pracy, w których dany system był wymieniany (Indeed i Simply Hired),
  - liczbą profili w sieciach profesjonalnych, gdzie system był wzmiankowany (LinkedIn),
  - liczbą wzmianek w sieciach społecznościowych (Twitter).
1. PostgreSQL jest znany i bardzo wysoko ceniony za stabilność i szeroki zestaw funkcji.
  2. PostgreSQL to baza nie tylko relacyjna, uwzględnia wiele rozwiązań post-relacyjnych np. typy danych: JSON, XML, wieloelementowe tablice (arrays), typy wierszy (row types), pełnotekstowe, szeregi czasowe, rekurencyjne zapytania grafowe, dane przestrzenne (GIS); wraz z funkcjami operującymi na tych nowych typach danych i – co więcej – z możliwością tworzenia zapytań mieszanych tj. obejmujących dane relacyjne i post-relacyjne.
  3. PostgreSQL używa standardowego SQL bez specyficznych dodatków utrudniających przeniesienie kodu lub przestawienie się programisty z innej bazy danych (ograniczenie vendor lock-in).
  4. W ciągu ostatnich lat kolejne wydania PostgreSQL skupiały się na wzmocnieniu back-endu, silnika bazy danych, dzięki czemu uzyskano istotny wzrost wydajności i skuteczności.
  5. Nowe cechy funkcjonalne i użytkowe są idealne dla wdrożenia środowiska DevOps.
  6. Migracja do PostgreSQL jest łatwa, a dzięki temu atrakcyjna dla wielu użytkowników Oracle, zmagających się z polityką cenową i licencyjną tej firmy.
  7. PostgreSQL ma [najbardziej wolnościową i o ile można to wiarygodnie ustalić najkrótszą licencję](#) spośród wszelkich licencji open source, co skutkuje także [szeroką popularnością wśród społeczności deweloperów](#) i co za tym idzie także [szeroką gamą sponsorów](#).
  8. PostgreSQL ma jedną z najszerzych ofert wsparcia komercyjnego wymaganego przez przedsiębiorstwa dla rozwiązań produkcyjnych w tym [najpopularniejsze w Europie EDB i 2ndQuadrant](#). Rozwiązania EDB posiadają wszystkie zalety bazy danych PostgreSQL, wzbogacając ją o zaawansowane narzędzia do migracji, integracji i zarządzania, a także funkcjonalności klasy enterprise oraz support 24/7.
- Node.js

Node.js to wieloplatformowe, asynchroniczne, sterowane zdarzeniami środowisko uruchomieniowe dla języka JavaScript i TypeScript. To oznacza, że za jego pomocą można uruchomić kod JavaScript bezpośrednio na maszynie lokalnej. Domyślny manager pakietów NPM sprawia, że deweloperzy łatwo zarządzają bibliotekami i zależnościami w projektach, a ogromna baza bibliotek ułatwia tworzenie skomplikowanych aplikacji. Język programowania jakim jest JavaScript odznacza się niskim progiem wejścia, przez co w ostatnim czasie platforma zyskała ogromną popularność. Node.js tworzony był z myślą o szybkim budowaniu skalowalnych aplikacji internetowych.

- C# .Net

To bezpłatna platforma deweloperska, którego językiem programowania jest między innymi C#. Pozwala tworzyć oprogramowanie takie jak:

- aplikacje internetowe

- mikroustugi
- aplikacje desktopowe
- aplikacje mobilne
- gry
- aplikacje konsolowe
- internet rzeczy (IoT)

Do głównych zalet .Net należą:

- wieloplatformowość
- może być uruchamiany na wielu architekturach procesorów
- open-source
- jeden kod źródłowy (eng „single codebase” )
- bezpieczeństwo wynikające z typowanie danych
- dedykowany manager pakietów Nuget
- narzędzie .Net interactive – pozwalający na wykonywanie fragmentów kodu w konsoli
- kompilator JIT (eng „Just in time compiler”) – kompilacja kodu odbywa się podczas wykonywania aplikacji, w ten sposób można ograniczyć czas kompilacji dużych i złożonych systemów
- Odśmieczacz pamięci (eng „Garbage collector”) - Automatyczne zarządzanie pamięcią
- Linq – technologia pozwalająca posługiwać się składnią przypominającą zapytania SQL odnosząc się do źródła danych **jakim jest** tablica w kodzie programu.

#### Biblioteki i frameworki

- Blitz js

To framework inspirowany na Ruby on Rails przeznaczony dla programistów full-stack, bazuje na innym JavaScriptowym frameworku Next.js. Tworząc nowy projekt w Blitz otrzymujemy na start wiele domyślnych rozwiązań które mają ułatwić pracę programiście i sprawi, że **zajmiemy** się pracą nad rozwijaniem projektu a nie budowaniem i przygotowywaniem. Jest to możliwe dzięki wbudowanym rozwiązaniom takim jak:

- „zero-api” data layer
- autentykacja
- autoryzacja
- domyślna struktura podziału plików
- Faker

Biblioteka do generowania ogromnych ilości losowych, ale wciąż realistycznych danych. Stosowana między innymi w celach wypełnienia baz danych przykładowymi danymi, testowania aplikacji, przygotowywania wersji demo aplikacji. Posiada dobrze udokumentowany i czytelny interfejs API, a generowanie danych następuje po wywołaniu odpowiedniej metody.

- ChakraUi

Jest to prosta, modułowa, udostępniająca gotowe komponenty biblioteka, która pozwala na budowanie wyglądu aplikacji w sposób blokowy. Posiada domyślnie zaimplementowane style dla udostępnianych komponentów, co sprawia, że korzystając podstawowych komponentów aplikacja będzie wyglądać czytelnie i schludnie.

- Nodemailer

To moduł dla aplikacji budowanych w środowisku Node.js umożliwiający łatwe wysyłanie wiadomości e-mail. Autorzy tej biblioteki stawiali dużą wagę na bezpieczeństwo, co przekłada się ogromną popularność i wszechstronność. Moduł udostępnia szereg funkcjonalności takich jak: wysyłanie emaili jako zwykły tekst lub dokumenty html, dodawanie załączników, autentykacje OAuth2, bezpieczne dostarczanie emaili za pomocą protokołu TLS. Biblioteka nie jest zamknięta, oznacza to że można tworzyć dodatkowe wtyczki.

- Uuid

Moduł pozwalający tworzyć identyfikatory UUID (en. Universally Unique Identifier). Jest to ciąg 128 bitów, który gwarantuje unikalność w przestrzeni i czasie. Stosowany często w ścieżkach URL.

- React

React jest deklaratywną, wydajną i elastyczną biblioteką javascriptową do budowania interfejsów użytkownika. Pozwala na tworzenie złożonych UI przy użyciu małych i odizolowanych od siebie kawałków kodu, zwanych "komponentami".

**Komentarz [S14]:** [Samouczek: Wstęp do Reacta – React \(reactjs.org\)](#)

React niesie ze sobą wiele korzyści, takich jak reużywalność komponentów, witalne drzewo DOM, jednokierunkowy przepływ danych, lekkość i stabilność, niski próg wejścia, łatwość w przerzuceniu się na aplikacje mobilne.

**Komentarz [S15]:** [ReactJS i React Native - czym się różnią? - Just Geek IT \(justjoin.it\)](#)

React znacznie ułatwia tworzenie interaktywnych UI. Przeznaczony jest do projektowania widoków obsługujących stan aplikacji. React domyślnie zajmie się aktualizacją danych i ponownym renderowaniem odpowiednich komponentów.

Deklaratywne widoki sprawiają, że kod staje się bardziej przewidywalny i łatwiejszy do debugowania.

- Prisma

Prisma to biblioteka przeznaczona dla programistów backend ułatwiająca odczytywanie i zapisywanie informacji do bazy danych w intuicyjny i bezpieczny sposób. Wspiera zarówno relacyjne jak i nierelacyjne bazy danych. Najpopularniejsze z nich to, PostgreSQL, MySQL, SQL Server, MongoDB.

Należy wspomnieć, domyślnie otrzymujemy również narzędzie o nazwie „Prisma studio” które umożliwia edytowanie i przeglądanie danych w graficznej formie - dane wyświetlane są w przeglądarce w formie tabel.

- Chart.js



Darmowa, oparta na elemencie <canvas> html, biblioteka open-source napisana w języku JavaScript, przeznaczona do wizualizacji danych. Tworzenie wykresów jest łatwe i elastyczne. Polityka open-source sprawia, że biblioteka wciąż dynamicznie się rozwija, zdobywając popularność i rzeszę stałych użytkowników. Natomiast dostępny system wtyczek to najbardziej efektywny sposób aby spersonalizować lub zmienić domyślne zachowanie wykresów. Biblioteka wspiera takie wykresy jak: wykres warstwowy, liniowy, słupkowy, bąbelkowy, punktowy, kołowy, typu radar.

### 3.3 Implementacja

“The ultimate Inspiration is the Deadline” - Nolan Bushnell

Po całym żmudnym i czasochłonnym, ale jakże ważnym procesie planowania systemu przychodzi czas na fazę implementacji, w którym to tworzony jest gotowy produkt. Faza implementacji to jedna z najbardziej kosztowych faz wytwarzania oprogramowania. W firmach zajmujących się tworzeniem oprogramowania pochłania na ogół dwie trzecie zasobów i połowę czasu trwania projektu. Duża koncentracja zasobów w stosunkowo krótkim czasie wynika z możliwości równoległego prowadzenia prac nad wieloma komponentami przez niezależnych ludzi lub zespoły. Faza konstrukcji jest zwykle okresem największego wysiłku i najwyższego zatrudnienia po stronie wykonawcy.

**Komentarz [S16]:** [Sacha K. - Inżynieria oprogramowania.pdf](#)

Autor pisząc pracę osobiście nie zważa na takie czynniki jak koszt, daje to częściową swobodę, natomiast ograniczony czas, przysparza problemów nie tylko autorowi ale i całemu zespołom programistycznym. Sytuację w której zbliża się **deadline** oddania gotowego produktu, a system wciąż wymaga nakładu sił i czasu programistów nazywamy „**Crunch mode**”. W takiej sytuacji jest kilka wyjść:

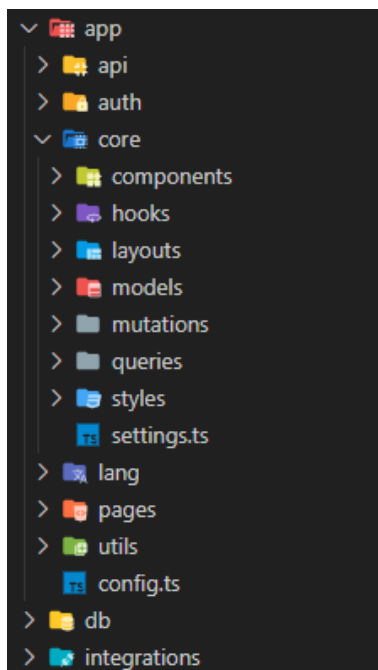
- nadgodziny
- skupienie jako pierwsze się na najbardziej znaczących funkcjonalnościach z perspektywy zleceniodawcy
- przydzielenie doświadczonych zespołów do trudniejszych zadań
- stworzenie prototypowego rozwiązania, a następnie stopniowe nadbudowywanie do wersji w pełni funkcjonalnej
- używanie zewnętrznych bibliotek zamiast tworzenia własnych rozwiązań
- zwiększenie liczebności drużyn pracujących nad danym projektem
- tymczasowe przydzielenie doświadczonego programisty do mniej doświadczonej drużyny znacząco zwiększa produktywność i szerzenie wiedzy
- dokładne określenie „**kamieni milowych**” – głównych celów do osiągnięcia

Projekt Beheer zakłada stworzenie aplikacji internetowej oraz udostępnienia biblioteki dla deweloperów umożliwiającej wprowadzanie w bezpieczny sposób funkcjonalności.

#### 3.3.1 Aplikacja internetowa

Aplikację internetową stworzono w node.js, a głównym środowiskiem programistycznym był Blitz.js.

Podstawowa struktura folderów aplikacji wygląda następująco:



Każdy z folderów jest integralną, zawiera kompleksową część systemu. Podział pomaga w łatwym nawigowaniu, a także wprowadzając jakiekolwiek zmiany, można być pewnym, że reszta modułów nie zmieni zachowania ani sposobu swojego działania.

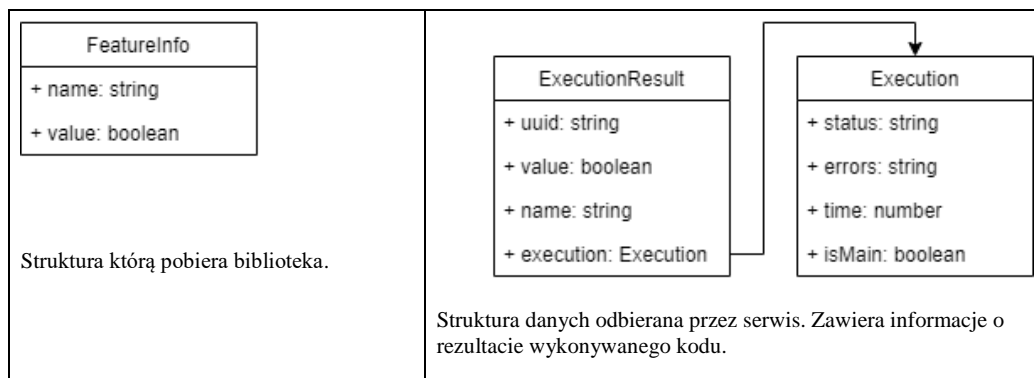
- Struktura bazy danych

#### DIAGRAM

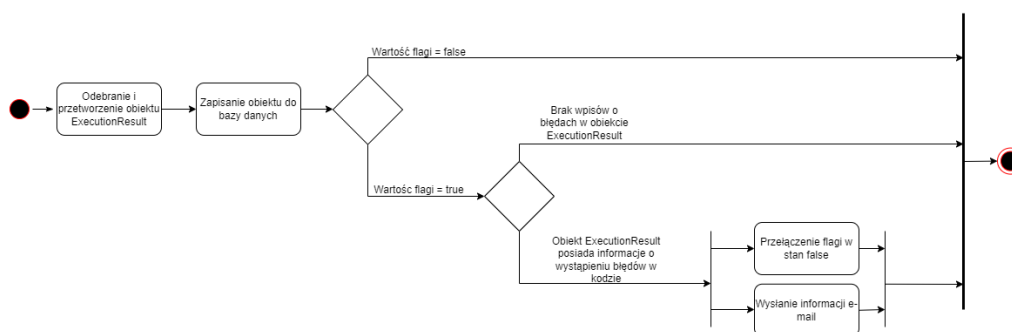
#### OPIS DLACZEGO TAK PODZIELONE

- Katalog „api” zawiera pliki odpowiadające za komunikację pomiędzy biblioteką programisty a systemem. To w nim zachodzi odpytanie serwisu o wartość flagi, odebranie wyniku z uruchomienia kodu, automatyczne przełączenie flagi w stan false czy wysłanie powiadomienie email.

Moduł ten wymienia z dołączoną biblioteką 2 struktury danych:



Aby lepiej zobrazować działanie tej części systemu przygotowano poniższy diagram aktywności.



Funkcja automatycznego przełączania flagi w przypadku wystąpienia błędu przy uruchomieniu nowo zaimplementowanego kodu, pozwala na szybkie reagowanie deweloperów. Z punktu widzenia dużych firm bezpieczeństwo i niezawodność jest najważniejsza, co za tym idzie, lepszym rozwiązaniem jest wycofanie wadliwej funkcjonalności, aby nie ucierpiało zaufanie klientów do firmy. Kolejnym powodem dla którego zdecydowano się wprowadzić ten mechanizm jest często spotykana sytuacja w której aplikacja się **crachuje**, lub wyświetla alert zawierający informacje o błędzie. Używając tej biblioteki unikniemy wszelakich niepożądanych skutków. Gdy wystąpi błąd, zostanie on zgłoszony do serwisu, a użytkownik nie zobaczy niczego niepokojącego, zwyczajnie nie osiągnie zamierzonego celu.

Z powyższego diagramu wynika, że gdy flaga posiada wartość false, to nie jest sprawdzana zawartość pola errors obiektu ExecutionResult. Wynika to z **natury**, pierwotny kod powinien być niezawodny. Nie mniej jednak wciąż gromadzimy dane w bazie danych. Pozwala to na dalsze udoskonalanie kodu np. gdy luka w nowej funkcjonalności wymaga dużego nakładu czasowego programistów, warto wtedy naprawić podstawowy kod.

- Katalog „auth” zawiera pliki odpowiedzialne jest za logowanie i rejestrowanie nowych użytkowników.  
Logowanie i rejestracja to skomplikowany proces, od programisty zależy bezpieczeństwo systemu, a tutaj są przekazywane wrażliwe dane. Wyciek takich

danych niesie za sobą szereg niepożądanych skutków, począwszy od spamu na poczcie email, aż po włamania na konta użytkowników na innych platformach w sieci.

**Ze strony użytkownika ważne jest aby hasła do konta nie powtarzały się, posiadały co najmniej 8 znaków, a także zawierały co najmniej 1 znak specjalny i cyfrę. To zapewnia podstawowe bezpieczeństwo.**

**Komentarz [S17]:** ZNAJDZ ZRODLO

Ze strony programisty należy wprowadzić wymienione wyżej ograniczenia co do jakości wprowadzanego hasła. Nie należy również zapisywać haseł do bazy danych w postaci zwykłego tekstu. Zamiast tego należy wprowadzić mechanizm haszowania hasła. Zapewnia to bezpieczeństwo na poziomie przesyłania danych przez sieć, a także potencjalnych wycieków z bazy danych. W ten sposób atakujący zamiast typowego hasła będzie widział pewien ciąg znaków, a rozszyfrowanie jest niezwykle czasochłonne. Zazwyczaj to wystarcza aby zniechęcić potencjalnego włamywacza do czynienia zła.

Mechanizm haszowania został zaimplementowany przy użyciu biblioteki `secure-password`. Biblioteka ta wykorzystuje algorytm kryptograficzny Argon2id, który jest parametryzowany poprzez: tekst, salt, wyznaczone zużycie pamięci, czas wykonywania algorytmu, ilość wątków, wynikowa długość ciągu znaków. Algorytm został zaprezentowany na zawodach Secure Password Hashing w 2019, co dodatkowo potwierdza jego bezpieczeństwo.

**Komentarz [S18]:** [Argon2 Hash Generator, Validator & Verifier](#)

```
export default resolver.pipe(async ({ email, password }, ctx) => {
  password = password.trim()
  const hashedPassword = await SecurePassword.hash( String password)

  const user = await db.user.create( args: {
    data: { email: email.toLowerCase().trim(), hashedPassword, role: "USER" },
    select: { id: true, name: true, email: true, role: true },
  })

  await ctx.session.$create({ userId: user.id, role: user.role as Role })
  return user
})
```

[funkcja realizująca rejestrację - wykorzystanie biblioteki SecurePassword i proces haszowania]

W projekcie zastosowano bibliotekę ZOD, wspomagającą walidowanie danych, zapewnia to, że programista otrzyma dokładnie takie dane jakich się spodziewa. Korzystając z zod tworzymy **chain-method**, z perspektywy **developer**, jest to niezwykle czytelny i wygodny rodzaj **api**. Zastosowanie tejże biblioteki ukazano na rysunku XXX.

```
const SignUpValidation = (email: string, password: string) => {
  const passwValidation = z.string()
    .min( Number minLength: 8, message: "Hasło nie może być krótsze niż 8 znaków")
    .max( Number maxLength: 30, message: "Hasło nie może być dłuższe niż 30 znaków")
    .regex( RegExp regex: /^[A-Z]/, message: "Hasło musi zawierać conajmniej 1 dużą literę")
    .regex( RegExp regex: /^[0-9]/, message: "Hasło musi zawierać conajmniej 1 cyfrę")
    .regex( RegExp regex: /^[a-z]/, message: "Hasło musi zawierać conajmniej 1 małą literę")
    .regex( RegExp regex: /^[^A-Za-z0-9]/, message: "Hasło musi zawierać conajmniej 1 znak specjalny")
  const emailValidation = z.string().email( message: "Wprowadzony email wygląda na niepoprawny")

  try{
    passwValidation.parse( data: password)
    emailValidation.parse( data: email)
  }
  catch(e){
    setErrors( value: () => e.errors.map(e => e.message).map(str => <p key={Math.random()}> {str} </p>))
    return false
  }
  return true
}
```

Wyświetlanie błędów przedstawiono na rysunku XXX. Funkcjonalność tą zrealizowano za pomocą komponentu Alert udostępnionego przez bibliotekę ChakraUI.

**Zarejestruj się**

**! Błąd rejestracji**

- Hasło nie może być krótsze niż 8 znaków ✕
- Hasło musi zawierać conajmniej 1 dużą literę
- Hasło musi zawierać conajmniej 1 cyfrę
- Hasło musi zawierać conajmniej 1 znak specjalny

adam.brzuchowo@gmail.com

.

Masz już konto? Zaloguj się.

→

W projekcie zaimplementowano mechanizm automatycznego ukrywania alertu o błędzie. Domyślnie wygaśnięcie następuje po 10 sekundach.

**Konfiguracja tego ustawienia zachodzi w pliku <<<PLIK>>>**

- Folder „components”  
Zawiera komponenty stworzone przez autora, na potrzeby implementacji systemu.

Koncepcyjnie, komponenty są jak javascriptowe funkcje. Przyjmują one arbitralne wartości na wejściu (nazywane “właściwościami” (ang. props)) i zwracają reactowe elementy opisujące, co powinno się pojawić na ekranie.

**Komentarz [S19]:** [Komponenty i właściwości – React \(reactjs.org\)](#)

Komponenty są reużywalne, dlatego przed ich stworzeniem, warto się zastanowić co dokładnie będziemy prezentować i jakie parametry przekazać, by później móc je użyć kolejny raz. Przykładem takiego komponentu może być zastosowany do wyświetlania błędów `ErrorViewComponent` ukazany na rysunku XXX

```
type ErrorProps = {
  error: any
  statusCode: number
  title: string
  closeCb: () => void
}

const ErrorViewComponent = (props: ErrorProps) => {
  return (
    <Alert id="ErrorView" status="error">
      <AlertIcon />
      <AlertTitle mr={2}>
        {props.title} {props.statusCode}
      </AlertTitle>
      <AlertDescription>{props.error}</AlertDescription>
      <CloseButton position="absolute" right="8px" top="8px" onClick={props.closeCb} />
    </Alert>
  )
}
```

Dzięki komponentom możemy budować skomplikowane strony internetowe jak z klocków, reużywalność pomaga w zmniejszeniu ilości kodu aplikacji. Dodatkowo gdy chcemy wprowadzić zmianę w jakimś elemencie witryny, zmieniamy w jednym miejscu w kodzie programu, a efekt będzie widoczny w każdym miejscu wystąpienia komponentu. Podział na komponenty sprzyja także szybkiemu poruszaniu się w projekcie i wyszukiwaniu dokładnie tego co nas interesuje.

React domyślnie udostępnia szereg hooków, które umożliwiają dynamiczną zmianę kontentu zawartego na stronie internetowej. Najczęściej używanym jest **hook** „`useState`”, który przechowuje zapisaną wartość, aż do usunięcia danego komponentu z drzewa DOM, co może nastąpić na przykład: po opuszczeniu witryny czy przejściu do innej podstrony. Dobrym przykładem ukazującym wykorzystanie **hooka** `useState` jest, rysunek XXX, w którym to tworzony jest widok listy funkcjonalności, na podstawie tablicy ‘features’ - za każdym razem gdy opuścimy witrynę chcemy na nowo pobrać z bazy danych, tablicę wszystkich funkcjonalności użytkownika.

```

const FeatureList = (props) => {
  const currentUser = useCurrentUser()
  const featuresBase = useQuery( Function queryFn: getFeatures, params: { userId: currentUser?.id }))[0]
  const sortedFeatures = sortFeaturesByDateFromLatestToOldest( Feature ListOfFeatures: featuresBase)
  const [features, setFeatures] = useState( Feature initialState: sortedFeatures)

  [...]

  <Table variant="simple" size="sm" id="featureTable">
    <Tbody>
      {features.map( Function callbackfn: (f) => {
        return (
          <FeatureView
            key={Date.now() + Math.random() * 1000000}
            item={f}
            updateCallback={updateFeatureFunction}
            deleteCallback={removeFeature}
          />
        )
      })}
    </Tbody>
  </Table>

```

- **Moduł „hooks”**

System do budowy wymagał zastosowania następujących hooków

- useState
- useMutation
- useRouter
- useRouterQuery
- useParm
- useCurrentUser
- useQuery
- useEffect
- useClipboard
- useSession

Jakich potrzebowałem w aplikacji

- **Folder Layout**

Komponent w nim zawarty opakowuje całą stronę, uzupełniając jedynie sekcję head dokumentu html.

- **Katalog „models”**

Autor tworząc ten projekt zdecydował się użyć specyficznej odmiany języka Javascript jakim jest TypeScript. Jest to język obiektowy, opierający się na Javascript, którego głównym atutem jest typowanie zmiennych. Tworząc duże projekty należy zwracać szczególną uwagę na kod który wytwarzamy, musi być on czytelny i zrozumiały nie tylko dla autora, **samodeskryptywny**. Wprowadzając typy danych znacznie łatwiej jest się odnaleźć w kodzie, zostaje zachowany pewien ład, zostaje zmniejszona możliwość popełnienia błędu przez programistę (eng. Room for error). W TypeScript otrzymujemy wszystkie atuty programowania obiektowego, takie jak dziedziczenie, interfejsy, polimorfizm. Wybierając ten język programowania, nie jest równoznaczne z całkowitym porzuceniem Javascript, wręcz przeciwnie, to właśnie stosowanie obu tych języków naprzemiennie stanowi prawdziwą potęgę. Katalog „models” zawiera modele danych używanych w kodzie źródłowym aplikacji. Programowanie staje się znacznie łatwiejsze gdy **wiemy** dokładnie jakie pola zawiera

**Komentarz [S20]:** Kilka nawazniejszych

obiekt. Na rysunku XXX zamieszczono model obiektu Feature. Tworząc w aplikacji nową flagę tak naprawdę tworzymy obiekt typu Feature, który kolejno zostaje zapisany do bazy danych. Jak później będzie można zauważyć niewiele się on różni od struktury tabeli w bazie danych.

```
export class Feature {  
  id: number  
  userId: number  
  name: string  
  uuid: string  
  value: boolean  
  createdAt: Date  
  updatedAt: Date  
  
  constructor(userId: number, name: string, value: boolean) {  
    this.userId = userId  
    this.name = name  
    this.value = value  
  }  
}
```

Dzięki zastosowaniu modeli mamy podstawową walidację danych – nie możemy przypisać wartości „abc” do zmiennej typu „number” – już w procesie kompilacji otrzymamy błąd.

Kolejnym atutem tworzenia modeli obiektów jest możliwość stworzenia statycznej metody wewnątrz obiektu, który zwróci instancję obiektu w raz z wypełnionymi losowymi danymi. Ułatwia to proces tworzenia oprogramowania, gdy potrzebujemy przetestować zachowanie systemu przy zadanych parametrach. Jest to popularna i często stosowana praktyka.

Zastosowano ten mechanizm w obiekcie PostExecutionData, co przedstawiono na rysunku XXX. Autor wykorzystał ten mechanizm do wypełnienia bazy danych.



```

export class PostExecutionData {
  uuid: string
  value: boolean
  execution: ExecutionData
  createdAt: number
  name?: string

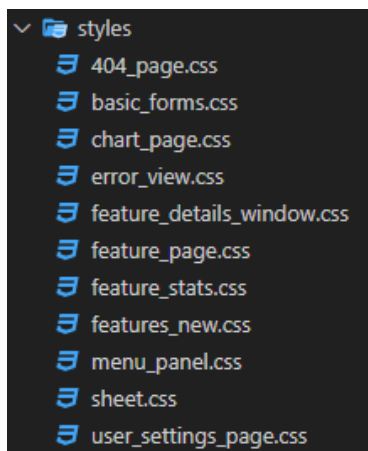
  static random(feature: Feature) {
    let x = new PostExecutionData()
    x.uuid = feature.uuid
    x.value = feature.value
    x.execution = ExecutionData.random( Boolean featureValue: x.value)
    x.createdAt = DateAddDays(
      Number | Date startDate: new Date(),
      days: RandomInt( Number min: -90, Number max: 0)).getTime()

    return x
  }
}

```

- Moduł „mutations”  
Katalog mutations, przechowuje wszystkie funkcje użyte w systemie do modyfikacji danych przechowywanych w bazie danych.
- Moduł „queries”  
Katalog queries, przechowuje wszystkie funkcje użyte w systemie do pobierania danych z bazy danych
- Moduł „styles”  
Tworząc witrynę internetową autor zdecydował się użyć dedykowanej biblioteki do tworzenia interfejsów graficznych, nie mniej jednak nie został on całkowicie zwolniony z stosowania kodu css. Aby osiągnąć oczekiwany rezultat, należało wyedytować domyślne style komponentów, lub stworzyć własny komponent i nadać mu odpowiednie cechy wyglądu. Do tego służą właśnie kaskadowe arkusz stylów – w skrócie CSS.  
Podział jaki zastosowano spławia że każdy z plików zawiera nie więcej niż 100 linii, przez co odnalezienie się i edycja nie sprawia większego kłopotu. Starano się stworzyć oddzielny plik css dla każdego głównego komponentu. **Należy wspomnieć o pliku 404\_page.css którego zawartość pobrano i wdrożono z darmowego źródła.**  
<<<SCREEN STORNY 404>>>

**Komentarz [S21]:** [58 HTML 404 Page Templates \(freefrontend.com\)](#)



```
@import "feature_stats.css";  
@import "404_page.css";  
@import "menu_panel.css";  
@import "feature_page.css";  
@import "chart_page.css";  
@import "user_settings_page.css";  
@import "error_view.css";  
@import "basic_forms.css";  
@import "features_new.css";  
@import "feature_details_window.css";
```

Z myślą o rozszerzalności autor zastosował jeden główny plik sheet.css w którym importuje wszystkie pozostałe pliki stylów. W pliku tym znajduje się także ostylowanie elementów bazowych serwisu, które widoczne są na wszystkich widokach witryny.

- **Plik Settings.ts**

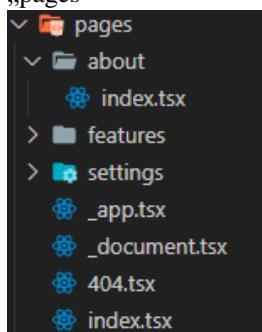
Myśląc o dalszym rozwoju aplikacji sporządzono globalny plik konfiguracyjny całej aplikacji nazwany - settings.ts

<<< OPISAĆ CO OZNACZAJĄ WPISY >>>

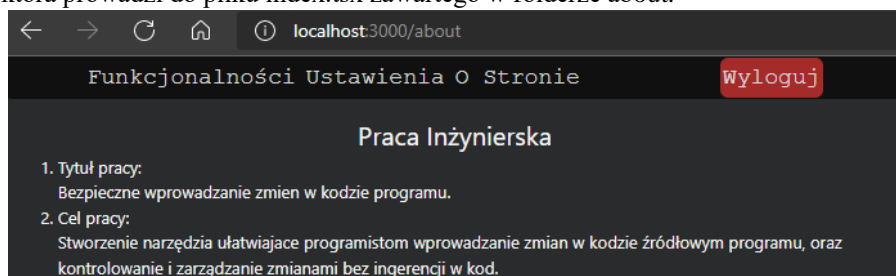
```
const AppSettings = {
  bgColor: "#21618C",
  color: "#FFFFFF",
  fontFamily: "'Courier New', monospace",
  menu: {
    bgColor: "#283747",
    height: "5vh",
  },
  button: {
    color: "#000000",
  },
  defaultHorizontalMargin: "10vw",
  errorCloseTimeout: 5_000,
}
```

- Moduł „lang”  
System został zaprojektowany, w taki sposób, aby nie ograniczać się tylko do rynku polskiego. Dlatego też wprowadzono moduł lang, w którym znajdują się tłumaczenia napisów zamieszczonych w interfejsie użytkownika. <<<OPISAĆ ZE MODUŁ NIE JEST SKONCZONY A JEDYNIE PRZETŁUMACZONO MENU>>>

- „pages”



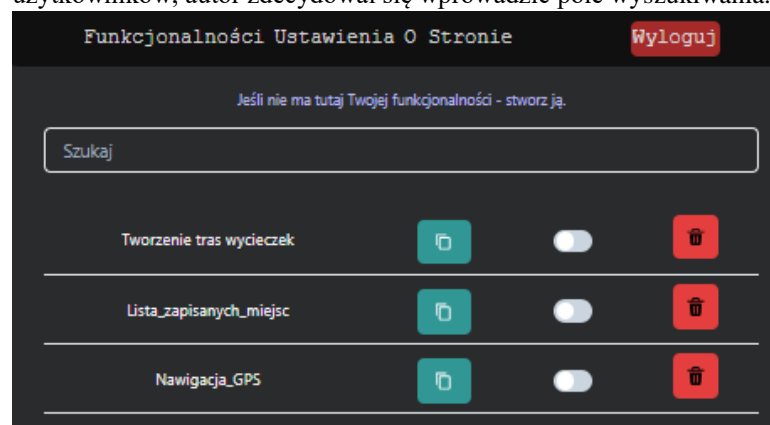
Katalog „pages”, przechowuje pliki stron internetowych. Na rysunku XXX ukazano strukturę katalogów, w jakiej są one przechowywane. Nie jest to przypadkowe ustawienie, ponieważ specyficzną cechą frameworka Blitz.js jest tak zwane dynamiczne trasowanie (eng. Dynamic Routes). Jest to mechanizm który konwertuje fizyczną ścieżkę do pliku na ścieżkę URL (ang. Uniform Resource Locator). Przykładem może być witryna przedstawiona na rysunku XXX. Ścieżka URL /about, która prowadzi do pliku index.tsx zawartego w folderze about.



Główną założeniem projektu Beheer jest tworzenie i zarządzanie funkcjonalnościami. Jako iż użytkownicy będą spędzać w tej części aplikacji, interfejs użytkownika powinien być czytelny i intuicyjny. Autor zdecydował się aby listę funkcjonalności przedstawić w tabeli, jak ukazano na rys XXX. Wybór tego rozwiązania pozwolił na umieszczenie przycisków kopiuj – zielony z ikoną dwóch kartek nakładających na siebie, przycisk typu przełącznik (eng. „switch”) – odpowiadający za przełączanie stanu flagi, oraz przycisk usuń – czerwony opatrzony w ikonę kosza. W odróżnieniu od tekstu stosowanie ikon przykuwa uwagę użytkownika, zajmują mniej miejsca na stronie – szczególnie ważny aspekt tworząc witrynę przeznaczoną na urządzenia mobilne, łatwo je ostrylować i wdrożyć, urozmaicając treść – nie zlewają się z resztą strony.

Na górze strony można zauważyć tekst „Jeśli nie ma tutaj Twojej funkcjonalności – stwórz ją.”. Jest to odnośnik który przenosi użytkownika do podstrony z formularzem tworzenia funkcjonalności – wymagane jest tylko wprowadzenie nazwy nowej funkcjonalności.

Biorąc pod uwagę obszerne systemy deweloper może chcieć tworzyć dziesiątki, jeżeli nie setki flag. Aby zwiększyć komfort korzystania z serwisu i zaoszczędzić czas użytkowników, autor zdecydował się wprowadzić pole wyszukiwania.



Implementację wyszukiwania funkcjonalności zaprezentowano na rysunku XXX.

Wspomniano wcześniej, że lista typu Feature przechowywana jest w stanie komponentu FeatureList, za każdym razem gdy następuje przeładowanie strony lista dostępnych funkcjonalności zostaje zaciągnięta z bazy danych i zapisana do zmiennej features. Podczas wyszukiwania nie możemy stracić pierwotnej listy funkcjonalności, a pobieranie jej z bazy danych, za każdym razem gdy użytkownik wprowadzi tekst w pole wyszukiwania byłoby zbyt powolne i **tworzyło** niepotrzebne zapytania do bazy danych. Zdecydowano się utworzyć kopię oryginalnej listy funkcjonalności – i zapisać pod zmienną allFeatures. Inicjalizowana jest w momencie wpisywania tekstu w polu wyszukiwania. Kolejno na podstawie wartości jaką wpisał użytkownik zapada decyzja, jeśli pole jest puste – należy wyświetlić wszystkie funkcjonalności, w przeciwnym wypadku należy wybrać tylko te funkcjonalności których nazwa rozpoczyna się od wprowadzonego tekstu. Dla wygody użytkownika wyszukiwanie nie uwzględnia wielkości liter (**case insensitive**).

```
const onSearchChange = (e) => {
  if (allFeatures.length == 0 && allFeatures.length != features.length) {
    setAllFeatures(features)
  }

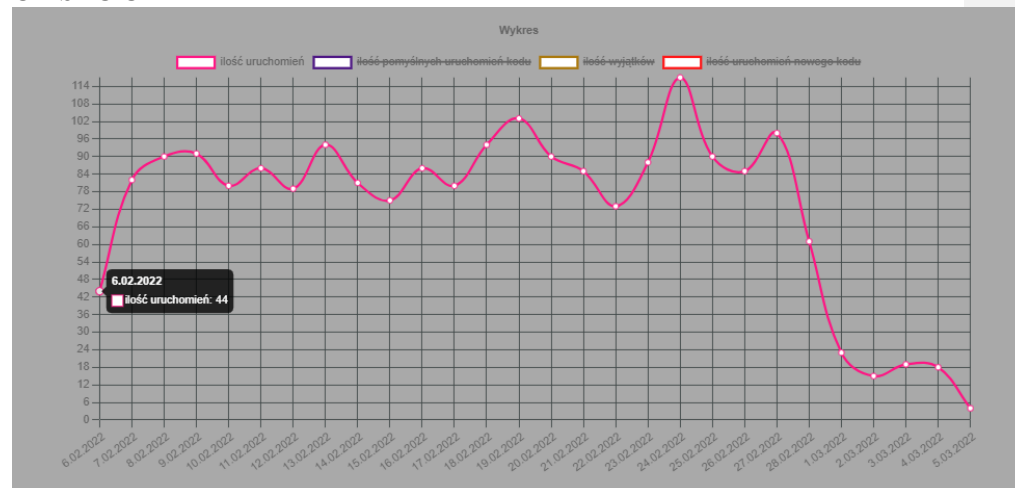
  if (e.target.value == "") {
    setFeatures(allFeatures)
  } else {
    setFeatures(() =>
      allFeatures.filter((f) =>
        f.name.toLowerCase().startsWith(e.target.value.toLowerCase())
      )
    )
  }
}
}
```

Klikając na funkcjonalność w tabeli z rysunku XXX, zostaniemy przeniesieni do widoku wykresu, przedstawiono to na rysunku XXX. Graficzne przedstawienie danych znacznie lepiej wpływa na odbiorcę niżeli czyste dane. Wykres domyślnie sporządzany jest z ostatnich 30 dni. Autor aby zachować przejrzystość i z myślą o dalszym rozwoju, **zbudował obiekt DataAdapter, który przyjmuje tablicę obiektów PostExecutionData – obiekty te otrzymujemy w wyniku uruchomienia kodu.**

Wewnątrz obiektu DataAdapter, odbywa się szereg kalkulacji. Korzystając z metod udostępnionych przez interfejs api obiektu, który przedstawiono na rysunku XXX, otrzymujemy dane potrzebne do sporządzenia wykresu.

<<<PRZEDSTAWIĆ SCREEN - INTERFACE DATA\_ADAPTERA>>>

## OPISAĆ CHART



- katalog „utils”

W katalogu utils zamieszczone są głównie funkcje które wspomagały autora w tworzeniu oprogramowania. Przykładem takiej funkcji może być RandomInt(), która zwraca losową liczbę całkowitą z przedziału sprecyzowanego w parametrach funkcji.

Tworzenie takich funkcji bezpośrednio w pliku z komponentem, zwiększałoby znacząco ilość linii kodu, a importowanie do kolejnych komponentów, wpływałoby negatywnie na jakość i czytelność kodu.

- Moduł „db”

Jeżeli chcemy gromadzić i składować dane to nieodłączną częścią systemu jest baza danych. W bazach danych dane przechowywane są w tabelach, dzięki temu są usystematyzowane, co usprawnia ich przetwarzanie i modyfikowanie.

Każda tabela posiada nazwę oraz szereg atrybutów, co odzwierciedla model w programowaniu obiektowym. Na rysunku XXX zaprojektowano bazę danych wykorzystaną w niniejszym projekcie.

<<<DB DIAGRAM>>>

Zazwyczaj do komunikacji z bazą danych używany jest specjalny język SQL, natomiast Blitz.js przychodzi z narzędziem zwanym Prisma. Prisma jest biblioteką która wspomaga programistę w budowaniu bazy danych poprzez deklarowanie schematów. Ułatwia także wykonywanie zapytań – programista dostarcza pewną strukturę która następnie konwertowana jest na zapytanie SQL. Zmniejsza to ilość błędów popełnianych przez programistę – zaawansowane zapytania SQL potrafią być bardzo obszerne, nie sposób stworzyć zapytanie pobierające z więcej niż jednej tabeli bez pomyłki, a także umożliwia łatwą zamianę na inną relacyjną bądź nierelacyjną bazę danych nie zmieniając struktur zapytań.

```
datasource db {
  provider = "postgres"
  url      = env("DATABASE_URL")
}
```

```
model Feature {
  id          Int      @id @default(autoincrement())
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  name        String
  uuid        String   @unique @default(uuid())
  value       Boolean   @default(false)
  user        User      @relation(fields: [userId], references: [id])
  userId      Int
  execution   ExecutionResult[]
}
```

@updatedAt, pole oznaczone w taki sposób będzie przechowywało czas, kiedy ostatnio została dokonana edycja danego rekordu. Proces ten jest całkowicie automatyczny.

Znacznik @relation opisuje relacje jakie zachodzą między tabelami. Ze rysunku XXX wynika, że tabela Feature połączona jest z tabelą User polami Feature.userId = User.id – zachodzi relacja „jeden do jeden”.

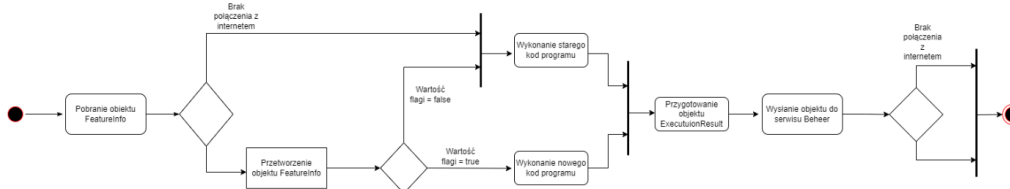
Ostatnia kolumna – executions to typ tablicowy – zachodzi relacja jeden do wielu. Oznacza to tyle, że jeden obiekt Feature, może być powiązany z wieloma obiektami ExecutionResult.

- Moduł „integration”  
Katalog „integration” przechowuje funkcje które w jakiś sposób korzystają z zewnętrznych systemów. Wysyłanie powiadomienia email było bardzo ważną częścią aplikacji, a realizacja tego wymagała skorzystać z zewnętrznego serwisu Gmail.

### Jak działa – pojedyncza instancja Pokazanie wiadomości email

#### 3.3.2 Biblioteka

Biblioteka udostępnia jedną klasę o nazwie Feature. Do implementacji biblioteki autor zdecydował się użyć obiektowego języka C#. Konstrukcja samej biblioteki jest prosta, a jej działanie zobrażono na rysunku XXX.



Klasa Feature implementuje interfejs IFeature, przedstawiony na rysunku XXX

```

interface IFeature{
    1 reference
    public IFeature Replace(Action code);
    1 reference
    public void With(Action code);
}
  
```

Do utworzenia instancji klasy Feature zastosowano wzorec projektowy **factory method**. Metoda **ControlledBy()** tworzy obiekt klasy feature i zwraca interfejs IFeature. Każda instancja klasy obsługuje pojedynczą funkcjonalność poprzez wskazanie identyfikatora UUID w parametrze metody **ControlledBy()**.

Zastosowanie interfejsu spowodowało, że deweloper będzie się komunikować się z klasą Feature poprzez metody udostępnione w interfejsie. Zasada enkapsulacji mówi, aby udostępniać na zewnątrz tylko to co niezbędne do poprawnego funkcjonowania obiektu. Ułatwia to programiście korzystanie z klasy. Prowadzi do ograniczenia niepożądanych zmian w kodzie klasy bazowej.

**Komentarz [S22]:** ZRODŁO

Metody interfejsu były implantowane zgodnie z wzorcem **chain method**. W którym to łączymy wywoływania metod w swoisty łańcuch. Dzięki temu poprawimy czytelność kodu.

Programista stosując tę bibliotekę uniknie niepożądanych skutków wywoływanych przez błędów w aplikacji. Zaprojektowano ją w taki sposób, że działanie aplikacji w której zaimplementowano jej użycie nie przerwie się z powodu wystąpienia błędu w przekazanym kodzie.

Kod programu kryjący się pod parametrami funkcji Replace() oraz With(), przekazywany jest dalej do metody RunCode(), a następnie wykonywany w bloku try{} catch{} - ukazano to na rysunku XXX. W przypadku wystąpienia błędu, **wyjątek zostanie złapany, a następnie wynik wykonania kodu (obiekt Execution) przekazany do serwisu. Użytkownik końcowy nie będzie poinformowany o wystąpieniu błędu, nie osiągnie też spodziewanego efektu.**

```
private Execution RunCode(Action code)
{
    Execution exec = new();
    try
    {
        code();
        exec.status = Statuses.SUCCESS;
    }
    catch (Exception ex)
    {
        exec.status = Statuses.FAILED;
        exec.errors = ex.Message;
    }
    return exec;
}
```

<<<DIAGRAM SEKWENCJI>>>



### 3.4 Testy

#### Testy Api

- GET – zły uuid
- POST na zły uuid jak zachowuje się platforma
- zła struktura excrestult

### 3.5 Eksploatacja

Produkt budowany był z myślą o 5 aspektach które opisane są jako definicja używalności (eng usability) sformułowana przez [Jakob Nielsen](#)


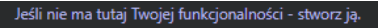

**Komentarz [S23]:** [Użyteczność \(informatyka\) – Wikipedia, wolna encyklopedia](#)

- nauczalność – miara trudności progu wejścia w dany system
- efektywność – miara szybkości posługiwania się systemem, przez doświadczonych użytkowników systemu
- zapamiętywalność – miara łatwości powrotu do systemu po dłuższej przerwie
- błędy – miara częstotliwości popełniania błędów w obsłudze z systemem (wynikająca z braku znajomości)
- satysfakcja – miara satysfakcji korzystania z systemu

Rzetelne zmierzenie każdego z tych aspektów wymaga przeprowadzenia bezpośredniego testu na docelowych użytkownikach. Z wiadomych przyczyn autor nie jest w stanie tego wykonać. Natomiast jedną z miar efektywności systemu jest lista kroków – im dłuższa lista tym system jest mniej efektywny.


#### 3.5.1 Proces tworzenia funkcjonalności

Lista kroków

1. Przejdź do rejestracji
2. Wypełnij formularz
3. Kliknij 
4. Przejdź do zakładki „Funkcjonalności” z menu głównego
5. Przejdź do widoku tworzenia nowej flagi klikając w link znajdujący się na górze strony  

6. Wypełnij formularz
7. Kliknij 

#### 3.5.2 Proces implementacji

Lista kroków

1. Przejdź do zakładki „Funkcjonalności” z menu głównego
2. Wyszukaj funkcjonalność
3. Kliknij przycisk  znajdujący się przy funkcjonalności
4. Przejdź do edytora IDE
5. Kliknij kombinację CTRL + V, aby wkleić skopiowaną funkcjonalność
6. Odwołując się do nazwy wklejonej zmiennej wywołaj metodę Replace() z odpowiednim parametrem
7. Używając wzorca łańcucha metod i wywołaj metodę With() z odpowiednim parametrem

#### 3.5.3 Przykład implementacji

Sytuacja w której dzielimy przez zero jest dobrym przykładem. **Z matematyki wiadomo, że takiej operacji nie możemy wykonywać.** Na rysunku XXX, przedstawiono fragment kalkulatora – jest to

program okienkowy. Flagę należy umieścić w sekcji zmiennych. Następnie przejść do fragmentu kodu w którym wykonywany jest nowy – potencjalnie „niebezpieczny” kod. Kolejno należy odwołać się do nazwy zmiennej jaką nosi flaga i wywołać obie funkcje Replace() a następnie With(). Funkcja Replace() przyjmuje jako parametr typu void, kod przekazywany w parametrze Replace() powinien być bezpieczny. W przykładzie ustawiono tekst w polu tekstowym na „Dzielenie przez 0 nie jest jeszcze obsługiwane”. Należy podkreślić, że ten kod zostanie uruchomiony gdy flaga będzie posiadała wartość false. Funkcja With() przyjmuje jako parametr typ void – umieszczamy tam „niebezpieczny” kod, w tym przykładzie - operację dzielenia.

```
public partial class Kalkulator : Form
{
    31 references
    string input = string.Empty;        //String storing user input
    6 references
    String operand1 = string.Empty;     //String storing first operand
    3 references
    String operand2 = string.Empty;     //String storing second operand
    8 references
    char operation;                     //Char to store operator
    8 references
    double result = 0.0;                //Get result
    1 reference
    private IFeature FLAG_DZIELENIE_PRZEZ_ZERO = Feature.CcontrolledBy("325e739a-4039-4f04-bb56-51981908:
[...]
```

```
    else if (operation == '/')
    {
        FLAG_DZIELENIE_PRZEZ_ZERO.Replace()=>{
            textBox1.Text = "Dzielenie przez 0 nie jest jeszcze obsługiwane";
        }.With(()=>{
            result = num1 / num2;
            textBox1.Text = result.ToString();
        });
    }
}
```

## 4. Podsumowanie

### 4.1 Wnioski

Wykorzystanie wielu dodatkowych bibliotek znacząco przyspieszyło implementację systemu. Ostatecznie udało się dostarczyć w pełni funkcjonalną wersję oprogramowania z wieloma perspektywami rozwoju. System nie jest idealny, ale spełnia założone wymagania zdefiniowane w fazie określania wymagań.

Z perspektywy autora, był to ciekawy i wymagający temat pracy. Przed zbudowaniem systemu Beheer, autor często używał statycznych zmiennych jako flagi, a przełączanie wymagało manualną edytując kod programu. Było to również dodatkową motywacją do stworzenia tego systemu.

Do implementacji zastosowano szereg technologii stosowanych przy budowie komercyjnych aplikacji. Dzięki wiedzy z zakresu protokołów internetowych, programowania obiektowego, inżynierii oprogramowania, budowy aplikacji internetowych zdobytej podczas studiów, uniknięto **przestojów**. Jednak największe problemy sprawiło pozycjonowanie elementów podczas tworzenia aplikacji internetowej. Niewielkie doświadczenie w tym obszarze dało się we znaki. Autor, ma nadzieję, że realizacja tego projektu będzie miała wpływ na przyszły rozwój kariery programisty/dewelopera.

Rosnąca popularność systemów typu toogle-feaure, a zarazem ograniczona liczba rozwiązań na rynku tego rodzaju, sprawia że w dłuższej perspektywie, system Beeher ma możliwość wypełnienia niszy i zyskania popularności.

## 4.2 Możliwe ścieżki rozwoju aplikacji

- Rozbudowanie o Rolout

Wprowadzenie nowej funkcjonalności do serwisu Beheer, umożliwiającego procentowe wdrażanie zmian. Przykładowy widok realizacji zaprezentowano na rysunku XXX.

<<<<UI DESIGN>>>>

- Wprowadzenie witrynę wyświetlającą **stacktrace**

Wyświetlenie stacktrace umożliwia szybszą nawigację do pliku i dokładne prześledzenie ścieżki wykonywanego kodu. Jest to kolejna funkcjonalność która zwiększyła by satysfakcję z użycia systemu.

- Wprowadzić witrynę internetową zawierającą kod pliku

Z perspektywy dewelopera wygodnym rozwiązaniem byłoby otrzymanie widoku zawartości pliku w którym wystąpił błąd i zaznaczenia linii.

Dostęp do takiej strony wymagałby autentykacji. Wyciek kodu źródłowego posiadającego luki, jest skrajnie niebezpieczne.

Wymagało by to również integracji z serwisu systemem kontroli wersji.

- Zbudowanie systemu grup i uprawnień, w celu dawania kontroli nad flagą innym użytkownikom

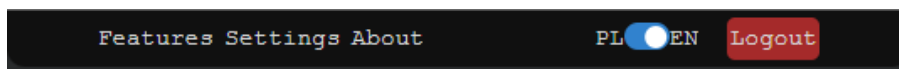
Stworzenie systemu grup użytkowników. Wszyscy użytkownicy z takiej grupy mogliby kontrolować flagę. Rozwiązałoby to problem z zmieniającym się zespołem deweloperów. Funkcjonalność ta wykazuje wysoką użyteczność w okresie trwania praktyk studenckich.

- Możliwość pobierania danych z tabeli XYZ, w formatach <CSV / JSON / XML>

Funkcjonalność wspomaga analityków systemu, w przeprowadzaniu analiz. Pracownikom na stanowisku <<<<XYZ>>>> lepiej przydzielać pracowników do zadań.

- Responsywność aplikacji internetowej – wsparcie dla urządzeń mobilnych
- Sortowanie tabeli z funkcjonalnościami po nazwie, dacie utworzenia, wartościach flagi
- Rozbudowa tabeli statystyk
- **Pełne** wsparcie dla wielu języków

Podstawowa wersja aplikacji implementuje, funkcjonalny mechanizm zmiany języka, lecz tłumaczenie jest tylko menu aplikacji – fragment ten ukazano na Rysunku XXX.



## 5. Literatura / Załączniki

Cos tu będzie .....

## 6. Streszczenie

Temat niniejszej pracy inżynierskiej jest „Bezpieczne wprowadzanie zmian w kodzie programu.”

Za cel, autor obrał sobie stworzenie w pełni działającej aplikacji internetowej do kontrolowania wprowadzanych funkcjonalności, oraz budowę klasy Feature umożliwiającej bezpieczne wdrażanie tych funkcjonalności w docelowym kodzie programu.

Autor postępuje zgodnie z dobrymi praktykami inżynierii oprogramowania, **obiera metodykę waterfall** i przeprowadza czytelników przez każdą z faz aby ostatecznie dostarczyć gotowy produkt. Zawdzięcza to przede wszystkim dokładnie przeprowadzonej fazie analizy projektu. Artefakty w postaci licznych diagramów i uzasadniony obór technologii, w dużym stopniu przyczyniły się do osiągniętego sukcesu.

Do wdrożenia tego systemu wybrał języki wysokopoziomowe TypeScript w raz z frameworkiem Blitz.js – full stack development react, oraz C# wykorzystany do budowy klasy Feature. Zastosował także wiele rozwiązań przyspieszających stworzenie produktu takich jak, walidowanie za pomocą biblioteki ZOD, stosowanie domyślnie ostylowanych komponentów i ikon z frameworka ChakraUi.

Zastosowanie wzorców projektowych i odpowiedniej konstrukcji klas zapewnia spójność i rozszerzalność systemu.

Bardzo ważnym aspektem była budowa spójnego, wygodnego, intuicyjnego serwisu internetowego. Projektując interfejs użytkownika autor miał na uwadze 5 cech które **określają używalność aplikacji zdefiniowane przez Jakoba Neilsona**.

## Spis Rysunków