**AKADEMIA MARYNARKI WOJENNEJ**

**im. BOHATERÓW WESTERPLATTE**

**Wydział Mechaniczno-Elektryczny**

Instytut Elektroniki i Automatyki Okrętowej

Katedra Informatyki

**PRACA DYPLOMOWA**

**INŻYNIERSKA**

Temat: Bezpieczne wprowadzanie zmian w kodzie programu.

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 Szreiber

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](#_Toc97820803)

[2. Wprowadzenie 6](#_Toc97820804)

[2.1 Komercyjne aplikacje 6](#_Toc97820805)

[2.2 Cel pracy 6](#_Toc97820806)

[2.3 Konkurencyjne rozwiązania na rynku 8](#_Toc97820807)

[2.4 Wady i zalety feature-flag 9](#_Toc97820808)

[3. Budowa Aplikacji 10](#_Toc97820809)

[3.1 Określenie wymagań 11](#_Toc97820810)

[3.1.1 Wymagania funkcjonalne 11](#_Toc97820811)

[3.1.2 Wymagania niefunkcjonalne 11](#_Toc97820812)

[3.2 Analiza i Projektowanie 11](#_Toc97820813)

[3.2.1 Wstęp 12](#_Toc97820814)

[3.2.2 Technologie i biblioteki użyte do realizacji projektu 12](#_Toc97820815)

[3.3 Implementacja 16](#_Toc97820816)

[3.3.1 Aplikacja internetowa 16](#_Toc97820817)

[3.3.2 Biblioteka 30](#_Toc97820818)

[3.4 Testy 32](#_Toc97820819)

[3.5 Eksploatacja 33](#_Toc97820820)

[3.5.1 Proces tworzenia funkcjonalności 33](#_Toc97820821)

[3.5.2 Proces implementacji 33](#_Toc97820822)

[3.5.3 Przykład implementacji 33](#_Toc97820823)

[4. Podsumowanie 35](#_Toc97820824)

[4.1 Wnioski 35](#_Toc97820825)

[4.2 Możliwe ścieżki rozwoju aplikacji 36](#_Toc97820826)

[5. Literatura / Załączniki 37](#_Toc97820827)

[6. Streszczenie 38](#_Toc97820828)

# Wykaz skrótów i oznaczeń

|  |  |
| --- | --- |
| JavaScript |  |
| TypeScript |  |
|  |  |

# Wprowadzenie

## 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 użytkownik, 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, przyspiesza proces wdrażania nowych pracowników (eng. On boarding), ułatwia przeprowadzanie inspekcji kodu (en. Code review) oraz wyszukiwanie i naprawę błędów. Stosuje się również narzędzia kontrolujące jakość kodu (eng. code quality checker) – a w razie niedostosowania się do zadanych restrykcji informuje o tym, poprzez podkreślenie źle napisanego kodu i wyświetlenie dokładnego opisu błędu. Firmy stosują również pryncypia które narzuca 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 programistę, który jest bardzo przestarzały – wdrożony przed wprowadzeniem wewnętrznych firmowych kryteriów kodowania. Dlatego, aby wprowadzić jakiekolwiek zmiany, najpierw musimy poświęcić wiele godzin, aby znaleźć odpowiednie miejsce w kodzie, następnie dokładnie przeanalizować dany fragment, opracować plan wdrożenia nowej funkcjonalności, żeby ostatecznie przejść do implementacji. Tak złożony proces może przysporzyć wiele trudności, szczególnie tym mniej doświadczonym programistom.

## Cel pracy

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

*- autor nieznany*

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

Istnieje pewne kryterium oceny błędu – ważność (eng. Severity). Określa jak duży wpływ na system ma dany defekt i jakie konsekwencje za sobą niesie. W skali ważności wyróżniamy:

- błąd krytyczny – uniemożliwia korzystania podstawowych funkcji systemu

- błąd pilny - znacząco utrudnia korzystanie z systemu, istnieje możliwość obejścia błędu w celu osiągnięcia oczekiwanego wyniku. Przykładem może być – wykorzystanie alternatywnej metody płatności w sklepie internetowym.

- błąd normalny – utrudnia korzystanie, niemniej jednak system ciąż jest w pełni funkcjonalny

- błąd mało istotny – zazwyczaj błędy ortograficzne, interpunkcyjne, zły 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 ścieżek wykonywanego kodu, co za tym idzie, zmiany w jednym obszarze aplikacji mogą spowodować wystąpienia błędów w innym module aplikacji. Szybkie reagowanie na takie zachowania mogą uratować mienie firmy. Skutki wystąpienia błędów krytycznych niosą za sobą poważne konsekwencje - w najgorszym przypadku spowodują zamknięcie firmy, sprawy karne, czy zadość uczynienie. W niedalekiej przeszłości zdarzyło się kilka tragicznych przypadków spowodowanym wystąpieniem wyjątku w programie. Oto wybrane:

- 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.

- 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.

## Konkurencyjne rozwiązania na rynku

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 różnych wartości flagi decydując podstawie geolokalizacji.

|  |  |
| --- | --- |
| LaunchDarkly  Wady:  - skomplikowana dokumentacja  - brak ciemnego interfejsu  Zalety:  - zawiera bibliotekę zaimplementowaną w wielu językach  - niska cena |  |
| Split.io  Wady:  - konfiguracja flagi wymaga dużo czasu  - duży próg wejścia  Zalety  - łatwa implementacja w kodzie  - darmowa dla użytkowników indywidualnych |  |
| CloudBees Feature Management  Wady:  - wysoka cena  - system skupia się w dużym stopniu na permisjach  Zalety:  - czytelność i przejrzystość witryny  - platforma zdobywa popularność |  |

Wszystkim powyższym rozwiązaniom brakuje ciemnego motywu i żadna ze stron nie kładła nacisku na minimalizację ilości kroków którą trzeba wykonać, aby stworzyć flagę. Niektóre serwisy nie są darmowe, bezpłatny jest tylko okres próbny.

## Wady i zalety systemów 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

## Zastosowanie

Mechanizm flag warto stosować gdy:

* Wdrażamy nową funkcjonalność
* Poprawiamy istniejącą funkcjonalność
* Wycofujemy funkcjonalność
* Rozszerzamy daną funkcjonalność
* Chcemy w wygodny sposób przełączać wygląd aplikacji

## Alternatywne rozwiązania

Programiści tworząc oprogramowanie często spotykają się z sytuacją w której chcą chwilowo wyłączyć pewną funkcjonalność, aby móc skupić się na wyznaczonym zadaniu. Przykładowo może być to obejście walidacji hasła. Jedno z podejść mówi, aby umieścić w pliku konfiguracyjnym, zmienną statyczną typu boolean, nadać jej wartość false, a następnie umieścić w konstrukcji if, tuż przed funkcją którą **chcemy** wyłączyć. Przykład takiego rozwiązania ukazano na rysunku XXX. Podejście to ma jedną główną wadę - aby przełączyć flagę musimy edytować kod.



Innym ze sposobów jest utworzenie tabeli w bazie danych, i przechowywanie tam nazwy funkcjonalności i jej wartości. Strukturę takiej tabeli zaprezentowano na rysunku XXX. Rozwiązanie to nie jest optymalne, ponownie pojawia się problem z manualnym przełączaniem flagi, na programiście spoczywałaby odpowiedzialność wdrożenia własnego mechanizmu pobierania wartości flagi, w raz z zwiększeniem liczby flag zarządzanie nimi stałoby się trudniejsze.



# Budowa Aplikacji

Proces wytwarzania aplikacji jest złożony. Na początku ery komputerów, gdy stacje robocze posiadły małą moc obliczeniową i w dużym stopniu były 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.

W latach 1968 i 1969 miały miejsce dwie konferencje w Rzymie i Garmich na których po raz pierwszy wprowadzono termin „INZYNIERIA OPROGRAMOWANIA” (ang. Software engineering, SE) dyscyplina informatyczna stosująca podejście inżynierskie do tworzenia oprogramowania, począwszy 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

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) ukazanego narysunku XXX.



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 jakiejkolwiek faz jest bardzo kosztowne w usunięciu, ponieważ skutkuje powrotem do pierwszej fazy czyli określenia wymagań.

## Określenie wymagań

Zbieranie wymagań opiera się na komunikacji między programistami, klientem i użytkownikami, w celu wypracowania spójnej definicji nowego systemu. Załamanie tej komunikacji i niedostateczne zrozumienie czyichś koncepcji prowadzi nieuchronnie w stronę systemu, który w najlepszym razie będzie niewygodny w użytkowaniu, a prawdopodobnie nie będzie w ogóle użyteczny. Z tego względu metody zbierania wymagań zmierzają do usprawnienia wspomnianej komunikacji

### Wymagania funkcjonalne

Opisują jakie funkcje powinien wykonywać system. Częstym sposobem definiowania takich funkcji jest formularz. Tabele XYZ, XYZ, XYZ pewną część wymagań funkcjonalnych względem serwisu Beheer.

|  |  |
| --- | --- |
| NAZWA FUNCKCJI | Rejestracja użytkownika |
| OPIS | Funkcja pozwala na tworzenie konta użytkownika |
| DANE WEJSCIOWE | Adres email, hasło |
| POWÓD | Korzystanie z serwisu Beheer wymaga utworzenia konta |

|  |  |
| --- | --- |
| NAZWA FUNCKCJI | Logowanie |
| OPIS | Funkcja pozwala określić tożsamość użytkownika i przydzielić mu odpowiednie zasoby |
| DANE WEJSCIOWE | Adres email, hasło |
| POWÓD | Dostęp do zasobów wymaga ustalenia tożsamości użytkownika |

|  |  |
| --- | --- |
| NAZWA FUNCKCJI | Tworzenie flagi |
| OPIS | Funkcja pozwala na stworzenie nowej flagi |
| DANE WEJSCIOWE | Nazwa flagi |
| POWÓD | Kontrolowanie funkcjonalności wymaga utworzenia flagi |

### Wymagania niefunkcjonalne

Należy rozumieć jako rozmaite ograniczenia które narzucamy na system.

- Aplikacja ma być dostępna 24/7

- Wygląd aplikacji powinien być przyjazny użytkownikowi

- Zapoznanie z systemem nie powinno trwać dłużej niż 15min

- Interfejs graficzny powinien być wyraźny i czytelny

- Interfejs graficzny powinien być w stonowanych kolorach

- Interfejs graficzny pewien 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

## Analiza i Projektowanie

### Wstęp

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

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 poligloci, 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.

### Analiza

Faza analizy dokonuje podziału systemu na obiekty, wyznacza granicę funkcjonalności poszczególnych obiektów systemu. Ustala kolejność kroków które system musi podjąć aby wykonać postawione zadanie. Podczas tej fazy tworzone są liczne modele, pozwalające zobrazować i lepiej zrozumieć co się dzieje wewnątrz systemu. W tym celu wykorzystuje się diagramy UML.

Faza analizy jest jedną z najistotniejszych czynności w procesie wytwarzania oprogramowania. Tworzone w tej fazie diagramy są podstawą do implementacji systemu. Pominięcie jakiegokolwiek scenariusza skutkuje powstaniem niepełnego, niefunkcjonalnego produktu.

**UML (eng Unified Modeling Language**) - język pół-formalny zaprojektowany, by definiować, wizualizować, konstruować i dokumentować systemy kładące nacisk na [oprogramowanie](https://pl.wikipedia.org/wiki/Oprogramowanie),

**Notacja UML** umożliwia opisanie założeń, jak i dokumentowanie istniejącego systemu w sposób graficzny w postaci dedykowanych diagramów. Diagramy te pozwalają na formalne opisywanie i modelowanie struktur czy procesów w sposób zrozumiały zarówno dla zespołu technicznego, jak i interesariuszy.

Jedną z metod definiowania wymagań funkcjonalnych systemu jest diagram przypadków użycia. Występują w nich aktorzy – czyli użytkownicy i systemy powiązana z naszym produktem, oraz przypadki użycia – czyli określone funkcje systemu. Aktorzy inicjują przypadki użycia korzystając z określonych funkcji systemu. Rysunek XXX przedstawia diagram przypadków użycia dla projektu serwisu Beheer.

****

Na rysunku XXX został przedstawiony diagram sekwencji projektu Beheer. Ukazuje on zakres obowiązków poszczególnych aktorów systemu. Identyfikowanie zakresu zachodzi poprzez definiowanie zdarzeń przepływających w systemie.



### Technologie i biblioteki użyte do realizacji projektu

Technologie:

* PostgreSQL

Darmowa, relacyjna baza danych. Rozpowszechniana na liberalnej licencji wolnego oprogramowania czasami nazywanej BSD. Cechą szczególną tej licencji jest przyzwolenie na modyfikacje kodu i redystrybuowanie zmodyfikowanej wersji programu. Znajduje się w top 4 najczęściej wybieranych baz danych. W rankingu pod uwagę brana, była liczba ofert pracy, zainteresowanie systemem mierzone systemem Google Trends, częstość wzmiankowania na forach dyskusyjnych i mediach społecznościowych. Do głównych jego zalet należą najwyższy stopień zaawansowania . Spełnia aż 170 na 179 głównych zasad określonych dla SQL. Może przechowywać typy danych takie jak JSON, XML, Key-Value. Posiada rozbudowaną dokumentacje. Jest bardzo ceniona za stabilność.

* Postman

To aplikacja do wykonywania zapytań API. Posiada wygodny graficzny interfejs użytkownika. Umożliwia tworzenie kolaboracji, tworzenie zmiennych globalnych, generowanie kodu do narzędzi takich wget, cURL, axios() i wiele innych. Nowością wprowadzoną do programu Postman jest tworzenie testów API. Testowanie odbywa się przy pomocy biblioteki chai.js.

* 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

- mikrousługi

- aplikacje desktopowe

- aplikacje mobilne

- gry

Do głównych zalet .Net należą:

- wieloplatformowość

- open-source

- jeden kod źródłowy (eng „single codebase” )

- bezpieczeństwo wynikające z typowanie danych

- dedykowany manager pakietów Nuget

- 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

- Odśmiecacz pamięci (eng „Garbage collector”) - Automatyczne zarządzanie pamięcią

Głównym powodem wyboru przez autora języka C# jest dobra znajomość technologii. Przygotowanie biblioteki z klasą Feature w języku obiektowym ułatwia zrozumienie kodu źródłowego, a zaimplementowanie biblioteki w innych językach staje się łatwiejsze.

Biblioteki i frameworki

* Blitz js

To framework inspirowany na Ruby on Rails przeznaczony dla programistów full-stack, bazuje na innym JavaScriptowym frameworku Next.js. Autor wybrał tę technologię ponieważ, 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 podstawowych jego funkcjonalności. Blitz domyślnie posiada zaimplementowane rozwiązania takie jak:

**- „zero-api” data layer – dane są pobierane z serwera, w czasie budowania strony**

- uwierzytelnianie

- autoryzacja

- domyślna struktura podziału plików

* Faker

Biblioteka do generowania ogromnych ilości losowych, 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. Dzięki temu korzystając podstawowych komponentów aplikacja będzie wyglądać czytelnie i schludnie.

* Nodemaier

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 emalii jako zwykły tekst lub dokumenty html, dodawanie załączników, autentykacje OAuth2, bezpieczne dostarczanie emalii za pomocą protokołu TSL. Biblioteka nie jest zamknięta, oznacza to, ze można tworzyć dodatkowe wtyczki. Motywacją autora projektu do wyboru tego pakietu była popularność i czytelna dokumentacja.

* Uuid

Moduł pozwalający tworzyć identyfikatory UUID (en. Universally Unique Identifier). Jest to ciąg 128 bitów, który gwarantuje unikalność - prawdopodobieństwo wygenerowania dwóch tych samych identyfikatorów jest bliska zeru. 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”.

React niesie ze sobą wiele korzyści, takich jak reużywalność komponentów, witralne drzewo DOM, jednokierunkowy przepływ danych, lekkość i stabilność, niski próg wejścia, łatwość w przerzuceniu się na aplikacje mobilne. Znacznie ułatwia tworzenie interaktywnych UI (en. User Interface). Przeznaczony jest do projektowania widoków obsługujących stan aplikacji. Domyślnie zajmuje się aktualizacją danych i ponownym renderowaniem odpowiednich komponentów.

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

* Prisma

Prisma to narzędzie przeznaczone dla programistów backend ułatwiająca odczytywanie i zapisywanie informacji do bazy danych w intuicyjny i bezpieczny sposób. Domyślnie jest częścią architektury frameworka Blit.js. Wspiera zarówno relacyjne jak i nierelacyjne bazy danych. Najpopularniejsze z nich to, PostgreSQL, MySQL, SQL Server, MongoDB.

Narzędzie to wbudowane jest do architektury frameworka Blitz.Js.

Instalując prismę domyślnie otrzymujemy również narzędzie o nazwie „Prisma studio” które umożliwia edytowanie i przeglądanie danych w graficznej formie – dane są wyświetlane w przeglądarce w formie tabel.

* Chart.js

Darmowa, oparta na elemencie <canvas> html, bliblioteka 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.

## Implementacja

“The ultimate Inspiration is the Deadline”

“największą inspiracją jest zbliżający się termin”

- Nolan Bushnell

W fazie implementacji dochodzi do wdrożenia systemu - 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.

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 całym zespołom programistycznym. Sytuację w której zbliża się termin (eng .**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.

### Aplikacja internetowa

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

Przedstawiona na rysunku XXX struktura katalogów jest domyślnym układem proponowanym przez framework. Każdy z folderów jest nierozłączną częścią systemu i zawiera implementację kompleksowej części systemu. Podział pomaga w łatwym nawigowaniu, a także wprowadzając jakiekolwiek zmiany, można być pewnym, że reszta modułów nie zmieni sposobu swojego działania.



* Struktura bazy danych

Autor implementacje serwisu internetowego rozpoczął od diagramu związków encji (ang. entity-relationship diagram – ERD). Pozwala on przedstawić graficznie struktury danych i relacje jakie zachodzą miedzy nimi. Diagram ten ukazuje pewne związki między encjami. Relacje przedstawiane są za pomocą linii łączących. Na zakończeniach tych linii opisuje się związki.

Na rysunku XXX przedstawiono diagram struktur bazy danych projektu Beheer. Z wyznaczonych relacji możemy wywnioskować, że jeden użytkownik może utworzyć wiele funkcjonalności – zachodzi relacja 1 do wielu. Jedna funkcjonalność może być uruchamiana wiele razy – relacja 1 do wielu pomiędzy Feature a ExecutionResult. Natomiast encja ExecutionResult połączona jest z encją Execution w relacji 1 do 1.

****

* Katalog „api” zawiera pliki odpowiadające za komunikację pomiędzy biblioteką programisty a systemem. To w nim realizowana jest odpowiedz na zapytanie o wartość flagi, odebranie wyniku z uruchomienia kodu, automatyczne przełączenie flagi w stan false czy wysłanie powiadomienia email. W tabeli XXX przedstawiono dwie struktury danych które służą do komunikacji serwis – klasa Feature.

|  |  |
| --- | --- |
| Struktura zwraca serwis na zapytanie o wartość flagi | Struktura danych zawierająca podstawowe informacje o rezultacie wykonywania kodu. Struktura wysyłana do serwisu po zakończeniu wykonywania kodu. |

Aby lepiej zobrazować działanie tej części systemu przygotowano diagram aktywności ukazany na rysunku XXX.



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, funkcjonalność zostanie wyłączona, a użytkownik nie zobaczy niczego niepokojącego - nie **osiągnie zamierzonego celu biznesowego.**

Z powyższego diagramu wynika, że gdy flaga posiad wartość false, to zostanie uruchomiony pierwotny kot programu. Także nie jest sprawdzana zawartość pola errors obiektu ExecutionResult – nie nastąpi przełączenie flagi w stan true. Takie rozwiązanie podyktowane jest tym, ze pierwotny kod powinien być niezawodny. Nie miej jednak wciąż zostają gromadzone dane o wykonaniu. 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 poszkodowanych użytkowników, aż po włamania na konta do innych platform w sieci.

**Ze strony użytkowania ważne jest aby hasła do kont w sieci nie powtarzały się,** posiadały co najmniej 8 znaków, a także zawierały co najmniej 1 znak specjalny i cyfrę. Zapewnia to podstawowe bezpieczeństwo.

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 haszowana hasła. Dzięki temu mamy zapewnione bezpieczeństwo na poziomie przesyłania danych przez sieć, a także potencjalnych wycieków z bazy danych. Haszując tekst atakujący zamiast typowego hasła zobaczy pewien ciąg znaków, a rozszyfrowanie jest niezwykle czasochłonne. Ta technika 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, **sól (eng. Salt)**, przydział ilości 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.



[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 **dewelopera**, jest to niezwykle czytelny i wygody rodzaj **api**. Zastosowanie tejże biblioteki ukazano na rysunku XXX.



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



W projekcie zaimplementowano mechanizm automatycznego ukrywania alertu. Domyślnie wygaśnięcie następuje po 10 sekundach. Odpowiada za to parametr „errorCloseTimeout” zawarty w ustawieniach aplikacji internetowej. Plik ukazany został na rynuku XXX

****

* 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.

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



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.

* **hooks**

React domyślnie udostępnia szereg hooków, które umożliwiają dynamiczną zmianę treści zawartej 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 opuszczenia 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.



**Podczas implementacji projektu autor użył szeregu hooków, a to niektóre z nich:**

**- useEffect – wykonuje pewien fragment kodu po zaraz wyrenderowaniu strony**

**- useCurrentUser – zwraca obiekt zalogowanego użytkownika**

**- useClipboard – kopiuje text do pamięci**

* **Folder Layout**

**Komponent w nim zawarty opakowuje całą witrynę, uzupełniając o sekcje <head> w dokumencie 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 główmy atutem jest typowanie zmiennych. Tworząc duże projekty należy zawracać szczególną uwagę na kod który wytwarzamy, mysi 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.

Katalog „models” zawiera modele danych używanych w kodzie źródłowym aplikacji. Programowanie staje się znacznie łatwiejsze gdy **wiadomo** jakie dokładnie pola zawiera obiekt. Na rysunku XXX zamieszczono model obiektu Feature. Tworząc w aplikacji nową flagę, tworzymy obiekt typu Feature, który kolejno zostaje zapisany do bazy. Definicja klasy w TypeScript niewiele różni się od struktury tabeli w bazie danych.



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

Kolejnym atutem tworzenia modeli jest możliwość stworzenia statycznej metody wewnątrz obiektu, który zwróci instancję jego instancję w raz z wypełnionymi losowymi danymi. Ułatwia to proces tworzenia oprogramowania, gdy potrzebujemy przetestować zachowanie systemu z pewnymi parametrami. Jest to popularna i często stosowana praktyka. Zastosowano ten mechanizm w obiekcie PostExecutionData. co przedstawiono na rysunku XXX. Autor wykorzystał ten funkcję random() do wypełnienia bazy danych.



* 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 witryną internetową autor zdecydował się użyć dedykowanej bibliteki do tworzenia interfejsów graficznych, nie mniej jednak nie został on całkowicie zwolniony z stosowania kodu css. Aby spełnić wymagania opisane w fazie określania wymagań, co do interfejsu graficznego, należało wy edytować domyślne style komponentów, lub stworzyć własny komponent i odpowiednio opisać formę prezentacji. Do tego służą właśnie kaskadowe arkusz stylów – w skrócie CSS (ang. *Cascading Style Sheets)*.

**Należy wspomnieć o pliku 404\_page.css którego zawartość pobrano i wdrożono z darmowego zródła.**

****

.

Starano się stworzyć oddzielny plik css dla każdego głównego komponentu.

Plik przedstawiony na rysunku XXX został wprowadzony przez autora z myślą o rozszerzalności. Podział jaki zastosowano spławia, że każdy z plików zawiera nie więcej niż 100 linii, przez co odnalezienie się i edycja nie sprawia problemu. Jest to wygodne rozwiązanie, ponieważ jeden główny plik sheet.css, w który importuje wszystkie pozostałe pliki stylów. W pliku sheet.css znajduje się także ostylowanie elementów bazowych serwisu, które widoczne są na wszystkich widokach witryny.



* **Plik Settings.ts**

**Myśląc o dalszym rozwoju aplikacji utworzono globalny plik konfiguracyjny aplikacji internetowej o nazwie - settings.ts. Przedstawiono go na rysunku XXX, Parametr errorCloseTimeout określa po jakim czasie ma się ukryć komponent informujący o błędzie.**



* Moduł „lang”

System został zaprojektowany, w taki sposób, aby nie ograniczyć produktu tylko do rynku polskiego. Wprowadzono moduł lang, w którym znajdują się tłumaczenia napisów zamieszczonych w interfejsie użytkownika.

W podstawowej wersji aplikacji wdrożone zostało tłumaczenie panelu menu ukazane na rysunku XXX.



Tłumaczenie trafia do komponentów jako obiekt typu Lang za pośrednictwem parametru props. Typ Lang ukazany na rysunku XXX, zawiera interfejs przez który programista dokonuje pobrania słownika – metoda get(), zmiany języka witryny – metoda update(), a parametr currentLang przechowuje aktualnie wybrany język.



Przedstawione na rysunku XXX ilustruje użycie modułu lang w komponencie MenuWindow. Każdy komponent pobiera tylko swoją paczkę z tłumaczeniami poprzez wywołanie metody get() z parametrem AppViews. Typ AppViews – to enum w którym zarejestrowane są nazwy widoków.



* „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). 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 znajdującego się w folderze about.



Główną założeniem projektu Beheer jest tworzenie i zarządzanie funkcjonalnościami. Ponieważ 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. Do generowania tabeli użyto tablicy obiektów typu Feature, przechowywanych w stanie komponentu FeatureList **RYC^^^**. 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 – jest szczególnie ważny aspekt tworząc witrynę przeznaczoną na urządzenia mobilne, łatwo je ostylować i wdrożyć, urozmaicają treść – nie występuje efekt zlewania z resztą strony.

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. Klikając na funkcjonalność w tabeli zostaniemy przeniesieni do widoku wykresu, przedstawionego na rysunku XXX



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.

W formularzu przedstawionym na rysunku XXX, aby utworzyć flagę wymagane jest wprowadzenie nazwy funkcjonalności i zatwierdzenie poprzez kliknięcie zielonego przycisku



Implementację wyszukiwania funkcjonalności zaprezentowano na rysunku XXX.

Lista obiektów typu Feature, na podstawie której tworzono tabelkę RYC^^^, przechowywana jest w stanie komponentu FeatureList. Za każdym razem gdy następuje przeładowanie strony lista dostępnych funkcjonalności zostaje pobrana z bazy danych i zapisana do stanu komponentu - zmiennej features. Podczas wyszukiwania nie możemy stracić pierwotnej listy funkcjonalności. Pobieranie jej z bazy danych, za każdym razem gdy użytkownik wprowadzi tekst w pole wyszukiwania jest zbyt wolne i generuje niepotrzebne zapytania do bazy danych. Zdecydowano się utworzyć kopię oryginalnej listy funkcjonalności i zapisać do zmiennej allFeatures za pomocą funkcji setAllFeatures(). 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 z pola wyszukiwarki. Z myślą o wygodnym interfejsie użytkownika, zaimplementowano wyszukiwanie które nie uwzględnia wielkości liter**.**



Wdrożono graficzne przedstawienie danych w postaci wykresu, znacznie ułatwia analizę. Domyślnie wykres prezentuje dane z ostatnich 30 dni. Przedstawiony na rysunku XXX wykres posiada 4 zbiory danych. Każdym z nich można manipulować - włączając lub wyłączając poprzez przyciski znajdujące się na górze wykresu.



Linie wykresu tworzone są na podstawie obiektu DataSet z rysunku XXX. Opis danych (pole label) wyświetlany jest tuż obok przycisków, natomiast tablica z wartościami numerycznymi zawarta w polu data, umieszcza się jako kolejne punkty na wykresie.



Autor aby zachować przejrzystość i z myślą o dalszym rozwoju, zbudował klasę , która przyjmuje tablicę obiektów PostExecutionData. Wewnątrz obiektu ChartDataAdapter, odbywa się szereg kalkulacji. Korzystając z metod udostępnionych przez interfejs api obiektu, który przedstawiono na rysunku XXX, otrzymujemy dane potrzebne do sporządzenia wykresu. Zastosowano uniwersalną konstrukcję funkcji. Produkuje na dane do wykresu z przekazanego parametru, to programista decyduje jaki okres czasu przedstawi na wykresie.



Obiekt ChartData przedstawiono na rysunku XXX. ChartEntity to podstawowy byt nakładany na wykres. Posiada opis (label) oraz pola typu number.



Konwertowanie danych do postaci DataSet fragment ukazano to na rysunku XXX



* 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, wpłynęłoby 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. Usprawnia to, ich przetwarzanie i modyfikowanie.

Każda tabela posiada nazwę oraz szereg atrybutów, co odzwierciedla model w programowaniu obiektowym.

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



Prisma do generowania tabel potrzebuje pewnego schematu. Na rysunku XXX ukazano model tabeli Feature. Deklaracja takiego modelu zawiera wszystkie potrzebne informacje do stworzenia tabeli w bazie danych.



@updatedAt - pole oznaczone w taki sposób będzie ostatni czas edycji rekordu Proces aktualizacji tego pola ten jest całkowicie automatyczny.

Znacznik @relation opisuje relacje jakie zachodzą między tabelami. Z rysnunku XXX wynika, że tabela Feture 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. Podstawowa wersja projektu zakłada, wysłanie wiadomości email w celu poinformowania. Deweloper w ten sposób dowie się wystąpieniu wyjątku w wdrożonym kodzie i przełączeniu flagi.

****

Aby zapewnić większą wydajność autor postanowił zastosować jeden statyczny obiekt służący do wysyłania wiadomości email. Statyczne pole transporter jest inicjowane tylko raz w czasie całego działania aplikacji. Taki wzorzec projektowy nazwano – singleton. Dodatkowo implementacja statycznej metody send(), pozwala na wysyłanie maili z każdego miejsca w programie. Fragment opisywanego kodu źródłowego ukazano na rysunku XXX.



### Biblioteka

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



Klasa Feature implementuje interfejs IFeature, przedstawiony na rysunku XXX



Do utworzenie instancji Klasy Feature zastosowano wzorzec projektowy **factory method . Metoda ControlledBy() tworzy obiekt klasy Feature i zwraca Interfejs IFeature. Każda instancja klasy obsługuje pojedyńczą funkcjonalność poprzez wskazanie indetyfikatora 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.

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.**



## Testy

Celem testowania jest wyszukiwanie odchyleń zaimplementowanego systemu od oczekiwanego (definiowanego w fazie określenia wymagań i analizy). Jest to proces w którym eksperymentalnie badamy pojedyncze moduły aplikacji lub cały system, poprzez uruchamianie pewnego modułu z zadanymi parametrami wejściowymi i przy określonych warunkach. Testując system możemy ocenić jego stabilność, szybkość działania, wydajność, poprawność.

Dokładne przetestowanie systemu wymaga objęcie każdego możliwego przypadku. Aby zagwarantować poprawność działa systemu wymagane jest sprawdzenie wszystkich możliwych przypadków danych wejściowych. Każdy zestaw parametrów wejściowych przekazywanych do systemu, można nazwać przypadkiem testowym (eng. Test case). Testowanie produktu staje się trudniejsze w raz wzrostem złożoności systemu i integracji, ponieważ zwiększa się pula przypadków testowych. Dokładne przetestowanie systemu nie jest to możliwe ponieważ, wymagałoby to sprawdzenia wszystkich możliwych parametrów.

Testy powinny przygotowywać osoby które nie znają wewnętrznej budowy systemu. Zwiększa to wiarygodność wykonywanych testów. Autor jako twórca całego projektu Beheer przygotował kilka przypadków testowych ukazanych na rysunku XXX. Pierwszy z nich sprawdza status odpowiedzi protokołu http - 200 oznacza poprawną realizację żądania. Natomiast drugi porównuje pobrane dane do tych, których oczekujemy.



## Eksploatacja

Produkt budowany był z myślą o 5 aspektach które opisane są jako definicja używalności (eng usability) sormułowana przez [Jakob Nielsen](https://pl.wikipedia.org/wiki/Jakob_Nielsen)

- nauczalność – miara trudności progu wejścia w dany system

- efektywność – miara szybkości posługiwania sie 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. Natomiast jedną z miar efektywności systemu jest lista kroków – im dłuższa lista tym system jest mniej efektywny.

### 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 

### Proces implementacji

Lista kroków

1. Przejdź do zakładki „Funkcjonalności” z menu głównego
2. Wyszukaj funkcjonalność
3. Kliknij przycisk  - kopiuj, 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

### Przykład implementacji

Dobrym przykładem jest sytuacja w której wykonujemy dzielenie przez **zero**. **Matematyka udowadnia, dlaczego nie ma to sensu obliczeniowego**. 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 interfejsu IFeature - Replace() a następnie With(). Funkcja Replace() przyjmuje jako parametr typ void, kod umieszczony w tym parametrze powinien być bezpieczny. W przykładzie wykonanie kodu z funkcji Replace() ustawi wyświetlany tekst 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.



# Podsumowanie

## Wnioski

Wykorzystanie wielu dodatkowych bibliotek znacząco przyśpieszyły 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 edytuję 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 feaure-flag, a zarazem ograniczona liczba rozwiązań na rynku tego rodzaju sprawia, że w dłuższej perspektywie, system Beeher ma możliwość wypełnienia niszy i zyskania popularności.

## Możliwe ścieżki rozwoju aplikacji

* Rozbudowanie o Roloout

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



* Wprowadzenie witrynę wyświetlającą **stacktrace**

Wyświetlenie stacktrace umożliwia szybszą nawigację do pliku i dokładne prześledzenie ścieżki wykonywanego kodu. Jest to kolejna funkcjonalność która zwiększyła by satysfakcje 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 odpowiedniej linii.

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

Wymagałoby 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 wspomoże 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 utworzenie, wartościach flagi
* Rozbudowa tabeli statystyk
* Pełne wsparcie językowe

Podstawowa wersja aplikacji implementuje, funkcjonalny mechanizm zmiany języka który ukazano na rysunku XXX, lecz tłumaczenie posiada tylko menu aplikacji.



# Literatura / Załączniki

Cos tu będzie ………………………………………………………………………..

# 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 model** kaskadowy 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 dobór technologii, w dużym stopniu przyczyniły się do osiągniętego sukcesu.

Do wdrożenia tego systemu wybrał języki wysokopoziomowe TypeScript w raz z frameworkiem Blitz.js - Fullstack React Framework, oraz C# wykorzystany do budowy klasy Feature. Zastosował także wiele rozwiązań przyspieszających stworzenie produktu takich jak: walidowanie z użyciem biblioteki ZOD, stosowanie domyślnie ostylowanych komponentów i ikon z biblioteki ChakraUI.

Zastosowanie wzorców projektowych, dla klas zapewnia 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