

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

OŚWIADCZENIE

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

WYKAZ SKRÓTÓW I OZNACZEŃ.....	6
WSTĘP.....	10
1. Cel pracy.....	12
1.1 Konkurencyjne rozwiązania na rynku.....	13
1.2 Wady i zalety systemów feature-flag.....	15
1.3 Zastosowanie.....	15
1.4 Alternatywne rozwiązania.....	15
2. PROJEKT APLIKACJI	17
2.1 Określenie wymagań	18
2.1.1 Wymagania funkcjonalne.....	19
2.1.2 Wymagania нефункционалне	19
2.2 Analiza i Projektowanie	21
2.2.1 Analiza	21
2.2.2 Technologie i biblioteki użyte do realizacji projektu.....	23
3. BUDOWA APLIKACJI	28
3.1 Implementacja.....	28
3.1.1 Aplikacja internetowa	29
3.1.2 Biblioteka.....	49
3.2 Testy	52
3.3 Eksploatacja.....	53
3.3.1 Proces tworzenia funkcjonalności.....	53
3.3.2 Proces implementacji	53
3.3.3 Przykład implementacji	54
PODSUMOWANIE	55
Wnioski	55
Możliwe ścieżki rozwoju aplikacji.....	55

LITERATURA.....	57
SPIS RYSUNKÓW	61
SPIS TABEL	63
STRESZCZENIA	64

WYKAZ SKRÓTÓW I OZNACZEŃ

JavaScript	„JavaScript to język programowania, który umożliwia wdrożenie na stronie internetowej skomplikowanych elementów, dzięki którym strona ta może nie tylko wyświetlać statyczne informacje, ale również obsługiwać zmianę treści odpowiednio do sytuacji, wyświetlać interaktywne mapy i animacje grafiki 2D/3D , wyświetlać video” [1].
TypeScript	Jest „statycznym kontrolerem typów dla programów napisanych w JavaScript - innymi słowy, narzędziem, które w statyczny sposób działa przed właściwym kodem, sprawdzając poprawność typów w programie” [2].
Node.js	„Wieloplatformowe środowisko uruchomieniowe o otwartym kodzie do tworzenia aplikacji typu server-side napisanych w języku JavaScript” [3].
UML	„Unified Modeling Language (...) - język pół-formalny zaprojektowany, by definiować, wizualizować, konstruować i dokumentować systemy kładące nacisk na oprogramowanie” [4].
JSON,	„JavaScript Object Notation – lekki format wymiany danych komputerowych. JSON jest formatem tekstowym, bazującym na podzbiorze języka JavaScript” [5].
XML,	“(ang. Extensible Markup Language) – uniwersalny język znaczników przeznaczony do reprezentowania różnych danych w strukturalizowany sposób” [6].
Key-Value	„powszechnie stosowany w informatyce abstrakcyjny typ danych, który przechowuje pary (unikatowy klucz, wartość)” [7]. Dostęp do wartości możliwy jest poprzez podanie klucza.
wget,	„narzędzie do ściągania plików z internetu. Obsługuje protokoły HTTP, HTTPS oraz FTP” [8].
cURL,	„cURL – sieciowa biblioteka programistyczna, (...) działająca po stronie klienta, z interfejsami dla ponad 30 innych języków” [9].
axios().	„Axios jest to prosty klient HTTP, za pomocą którego tworzymy zapytania w przeglądarce oraz NodeJS” [10].
mikrousługi	„Małe, luźno powiązane usługi, które mogą być

	używane do tworzenia aplikacji opartych na chmurze” [11]
API	„(ang. application programming interface, API) – zbiór reguł ściśle opisujący, w jaki sposób programy lub podprogramy komunikują się ze sobą” [12].
Google Trends	„Serwis Google udostępniający informacje na temat liczby, pochodzenia, zależności od czasu i głównych regionów zapytań kierowanych do wyszukiwarki Google. Usługa pozwala m.in. na porównanie częstości różnych zapytań na wykresie” [13].
Ruby on Rails	„Darmowy framework do tworzenia aplikacji webowych napisany w języku Ruby” [14]
framework	„Framework albo platforma programistyczna – szkielet do budowy aplikacji. Definiuje on strukturę aplikacji oraz ogólny mechanizm jej działania, a także dostarcza zestaw komponentów i bibliotek ogólnego przeznaczenia do wykonywania określonych zadań. Programista tworzy aplikację, rozbudowując i dostosowując poszczególne komponenty do wymagań realizowanego projektu, tworząc w ten sposób gotową aplikację” [15].
next.js	„Next.js to specjalny, minimalistyczny framework, który pozwala w łatwy sposób stworzyć aplikację React posiadającą obsługę renderowania po stronie serwera” [16].
protokołu TLS	„TLS zapewnia poufność i integralność transmisji danych, a także uwierzytelnienie serwera, a niekiedy również klienta” [17].
html	„(ang. HyperText Markup Language, hipertekstowy język znaczników) – język znaczników stosowany do tworzenia dokumentów hipertekstowych” [18].
UUID	„Uniwersalnie unikalny identyfikator (ang. UUID) to 128-bitowa liczba, która identyfikuje informacje w systemie komputerowym” [19].
DOM	„(ang. Document Object Model) - Model ten opisuje jak zaprezentować tekstowe dokumenty HTML (te, które piszesz sobie jako tekst w edytorze) w postaci modelu obiektowego w pamięci komputera” [20].

UI	„(ang. User Interface)– to zbiór elementów i reguł w kontekście projektowania wizualnego danego produktu” [21].
sól	„(ang. salt) lub ciąg zaburzający – dane losowe dodawane do hasła podczas obliczania funkcji skrótu przechowywanej w systemach informatycznych. Celem soli jest ochrona systemowej bazy haseł przed atakami słownikowymi” [22].
URL	„(ang. Uniform Resource Locator) – ujednolicony format adresowania zasobów stosowany w Internecie i w sieciach lokalnych” [23].
SQL	„(ang. Structured Query Language wym.) – strukturalny oraz deklaratywny język zapytań. Jest to język dziedzinowy używany do tworzenia, modyfikowania relacyjnych baz danych oraz do umieszczania i pobierania danych z tych baz” [24].
Hermetyzacja	„Hermetyzacja polega na ukrywaniu pewnych danych składowych lub metod obiektów danej klasy tak, aby były one dostępne tylko metodom wewnętrznym danej klasy lub funkcjom zaprzyjaźnionym” [25].
Wyjątek	„(ang. exception) - mechanizm przepływu sterowania używany w procesorach oraz współczesnych językach programowania do obsługi zdarzeń wyjątkowych, a w szczególności błędów, których wystąpienie zmienia prawidłowy przebieg wykonywania programu” [26].
IDE	„(ang. integrated development environment) – zintegrowane środowisko programistyczne. program lub zespół programów (środowisko) służących do tworzenia, modyfikowania, testowania i konserwacji oprogramowania” [27].
Stack trace	„(ang. dosł. ślad stosu) – lista aktywnych ramek stosu w pewnym momencie działania aplikacji. Nazywana także zrzutem stosu” [28].

WSTĘP

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 (ang. *coding standard*), którego muszą przestrzegać pracownicy firmy. Pomaga to w zachowaniu ogólnego porządku w kodzie programu, przyspiesza proces wdrażania nowych pracowników (ang. *on boarding*), ułatwia przeprowadzanie inspekcji kodu (en. *code review*) oraz wyszukiwanie i naprawę błędów. Stosuje się również narzędzia kontrolujące jakość kodu (ang. *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, trzeba 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. Bezpieczeństwo programu to sprawa najwyższej wagi. Firmy budują swoje zaufanie wśród klientów przez dziesiątki lat, jeden błąd może spowodować katastrofalne skutki.

Istnieje pewne kryterium oceny błędu – ważność (ang. *Severity*). Określa jak duży wpływ na system ma dany defekt i jakie konsekwencje za sobą niesie. W skali ważności wyróżniamy: [29]

- 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 wciąż jest w pełni funkcjonalny

- błąd mało istotny – zazwyczaj błędy ortograficzne, interpunkcyjne, zły kolor

1. Cel pracy

„Czasu nie cofniesz, ale błędy możesz naprawić.”

- autor nieznan

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:

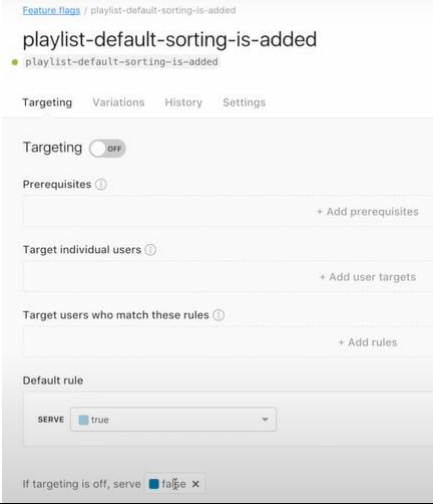
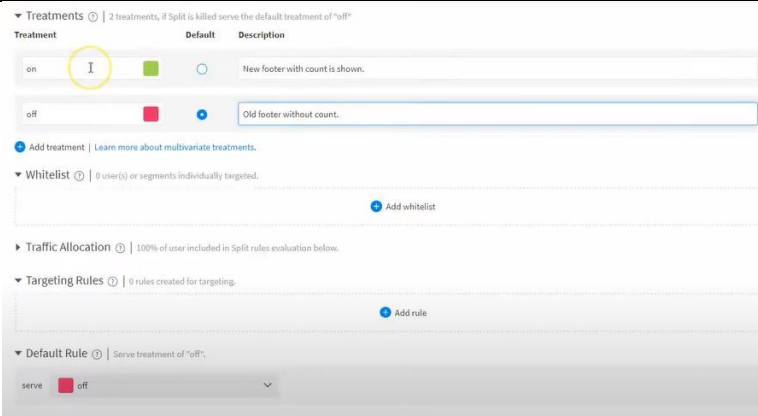
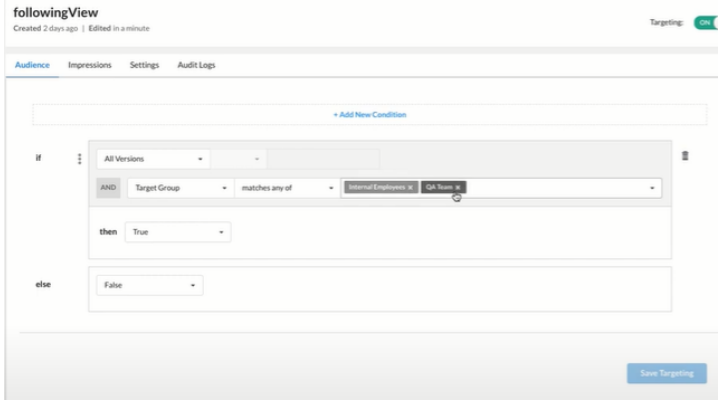
- Między 1985 a 1987 rokiem 6 osób uległo poparzeniu w wyniku naświetlań maszyną Therac-25 Była to maszyna stosowana w latach 80 do radioterapii nowotworów. Trzy z nich zmarły w następstwie wypadku. A oto przypadek jednego z pacjentów. W 1985 r. – jedna z maszyn uległa awarii, wyświetlając komunikat 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 [30].

- 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 [31].

1.1 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.

Tabela 1. Zestawienie konkurencyjnych narzędzi [1]

<p>LaunchDarkly</p> <p>Wady:</p> <ul style="list-style-type: none"> - skomplikowana dokumentacja - brak ciemnego interfejsu <p>Zalety:</p> <ul style="list-style-type: none"> - zawiera bibliotekę zaimplementowaną w wielu językach - niska cena 	 <p>The screenshot shows the LaunchDarkly configuration page for a feature flag named 'playlist-default-sorting-is-added'. The 'Targeting' tab is active, showing a toggle switch set to 'off'. Below the toggle are sections for 'Prerequisites', 'Target individual users', and 'Target users who match these rules', each with a '+ Add' button. A 'Default rule' section shows a dropdown set to 'SERVE' with a value of 'true'. At the bottom, it states 'If targeting is off, serve false'.</p>
<p>Split.io</p> <p>Wady:</p> <ul style="list-style-type: none"> - konfiguracja flagi wymaga dużo czasu - duży próg wejścia <p>Zalety</p> <ul style="list-style-type: none"> - łatwa implementacja w kodzie - darmowa dla użytkowników indywidualnych 	 <p>The screenshot shows the Split.io configuration page. It displays two treatments: 'on' (green circle) and 'off' (red square). The 'on' treatment is selected. Below the treatments are sections for 'Whitelist', 'Traffic Allocation' (set to 100%), 'Targeting Rules', and 'Default Rule' (set to 'off').</p>
<p>CloudBees Feature Management</p> <p>Wady:</p> <ul style="list-style-type: none"> - wysoka cena - system skupia się w dużym stopniu na permisjach <p>Zalety:</p> <ul style="list-style-type: none"> - czytelność i przejrzystość witryny - platforma zdobywa popularność 	 <p>The screenshot shows the CloudBees Feature Management interface for a feature named 'followingView'. It displays a targeting rule configuration with a dropdown for 'All Versions' and a condition 'Target Group matches any of Internal Employees'. The 'then' clause is set to 'True' and the 'else' clause is set to 'False'. A 'Save Targeting' button is at the bottom right.</p>

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.

1.2 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

1.3 Zastosowanie

Mechanizm flag warto stosować gdy:

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

1.4 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 1. Podejście to ma jedną główną wadę - aby przełączyć flagę musimy edytować kod.


```

app > config.ts > ...
1 | import { Lang } from "../lang/available"
2 |
3 | export const MULTI_LANGUAGE = true
4 | export const DEFAULT_LANGUAGE = Lang.PL
5 |
6 | export const USE_SIGNUP_VALIDATOR = false

```

```

if (USE_SIGNUP_VALIDATOR)
  if (validate(email, password) == false) {
    setErrorView(true)
    return
  }
await signupMutation({ email, password })

```

Rys. 1. Implementacja flagi w postaci zmiennej statycznej [opracowanie własne]

Innym ze sposobów jest utworzenie tabeli w bazie danych, i przechowywanie tam nazwy funkcjonalności i jej wartości. Strukturę takiej tabeli zaprezentowano na rysunku 2. 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.

Feature
+ nazwa: string
+ wartosc: boolean

Rys. 2. Model tabeli *Feature* w bazie danych [opracowanie własne]

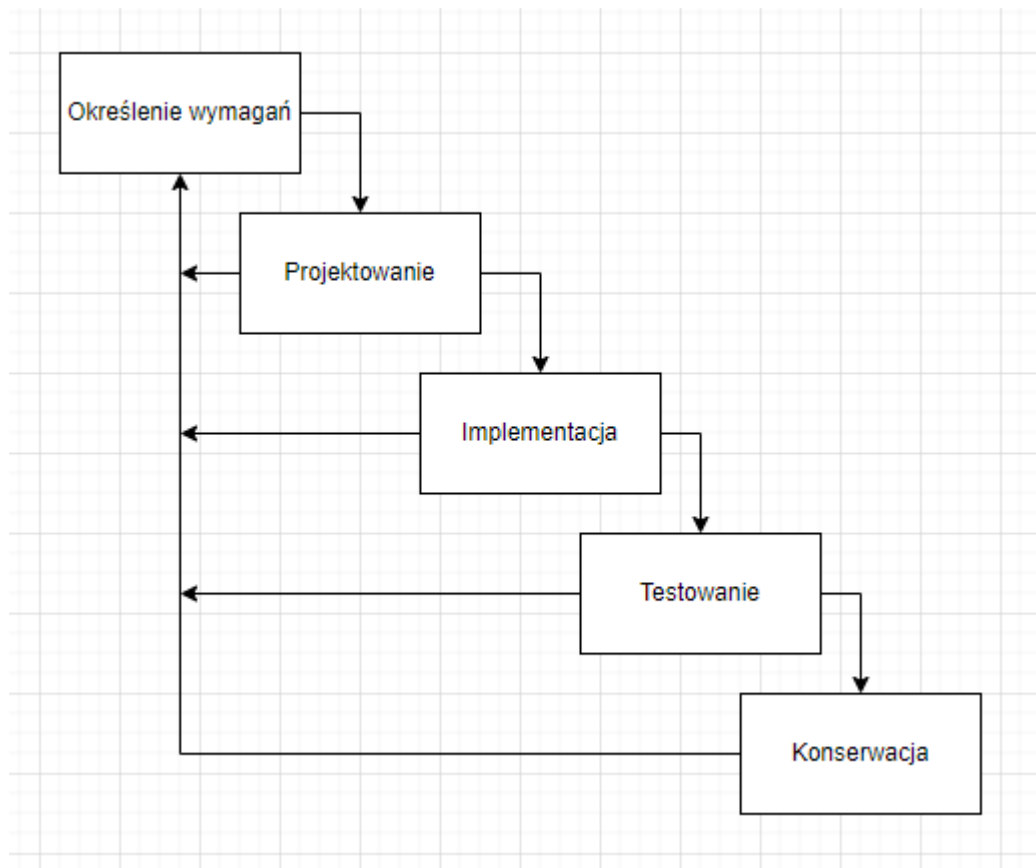
2. PROJEKT APLIKACJI

Proces wytwarzania aplikacji jest złożony. Na początku ery komputerów, gdy stacje robocze posiadały 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 [32].

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*) „dyscyplina informatyczna stosująca podejście inżynierskie do tworzenia oprogramowania, począwszy od analizy i określenia wymagań, przez projektowanie i wdrożenie, aż do ewolucji gotowego oprogramowania” [Przypis 33]

„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” [33].

Dziedzina ta wprowadza wiele procesów („Proces jest zbiorem aktywności wykonywanych z myślą o osiągnięciu pewnego celu” [34]) które należy dobrać indywidualnie do tworzonego projektu. Do realizacji niniejszego projektu użyto modelu kaskadowego (ang. *waterfall*) ukazanego na rysunku 3.



Rys. 3. Model kaskadowy [opracowanie własne]

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.

2.1 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” [34].

2.1.1 Wymagania funkcjonalne

„Opisują jakie funkcje powinien wykonywać system” [34]. Częstym sposobem definiowania takich funkcji jest formularz. Tabele 2-4 opisują pewną część wymagań funkcjonalnych względem serwisu Beheer.

Tabela 2. Wymagania funkcjonalne – rejestracja

NAZWA FUNKCJI	Rejestracja użytkownika
OPIS	Funkcja pozwala na tworzenie konta użytkownika
DANE WEJŚCIOWE	Adres email, hasło
POWÓD	Korzystanie z serwisu Beheer wymaga utworzenia konta

Tabela 3. Wymagania funkcjonalne – logowanie

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

Tabela 4. Wymagania funkcjonalne – tworzenie flagi

NAZWA FUNKCJI	Tworzenie flagi
OPIS	Funkcja pozwala na stworzenie nowej flagi
DANE WEJŚCIOWE	Nazwa flagi
POWÓD	Kontrolowanie funkcjonalności wymaga utworzenia flagi

2.1.2 Wymagania niefunkcjonalne

„Należy rozumieć jako rozmaite ograniczenia które narzucamy na system” [34].

- 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 powinien mieć ciemną szatę graficzną
- Informacje o błędach powinny być umieszczone w widocznym dla użytkownika miejscu

- Stworzenie nowej flagi nie powinno zajmować więcej niż 1 minutę
- Hasła w bazie danych powinny być przechowywane w bezpieczny sposób

2.2 Analiza i Projektowanie

„*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” [35].

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” [36].

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 (ang. *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” (ang. „*use the right tool for the right job*”) oraz własnymi preferencjami wybierał język adekwatny do danego zadania [37]. 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” [37]

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.

2.2.1 Analiza

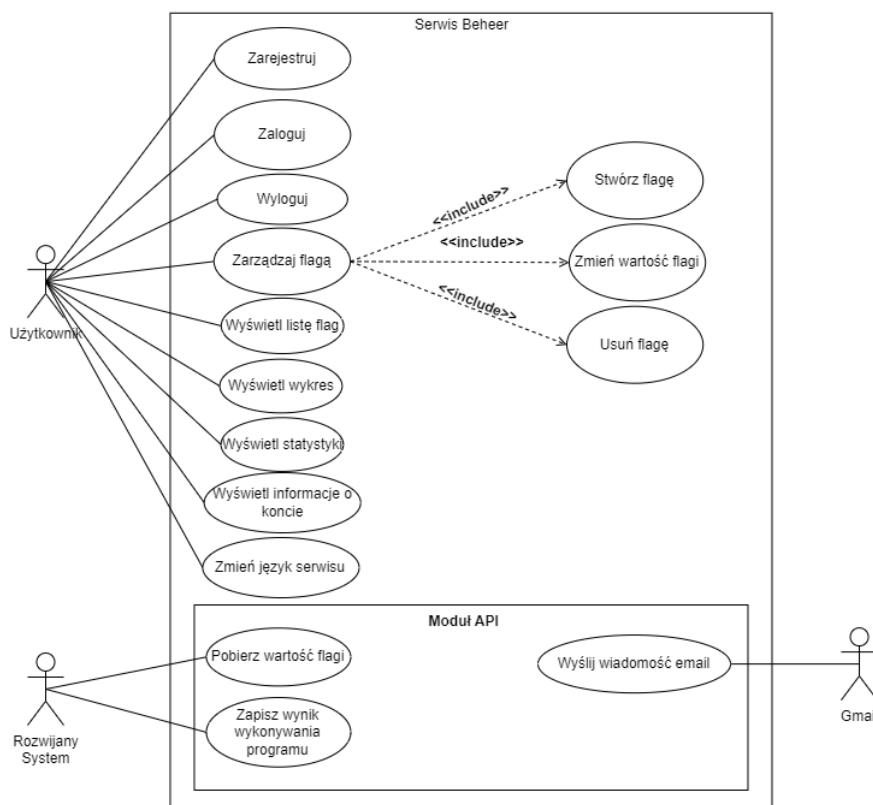
Faza analizy dokonuje podziału systemu na obiekty, wyznacza granicę funkcjonalności poszczególnych obiektów systemu. Ustala kolejność kroków które system

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, нефункционального produktu.

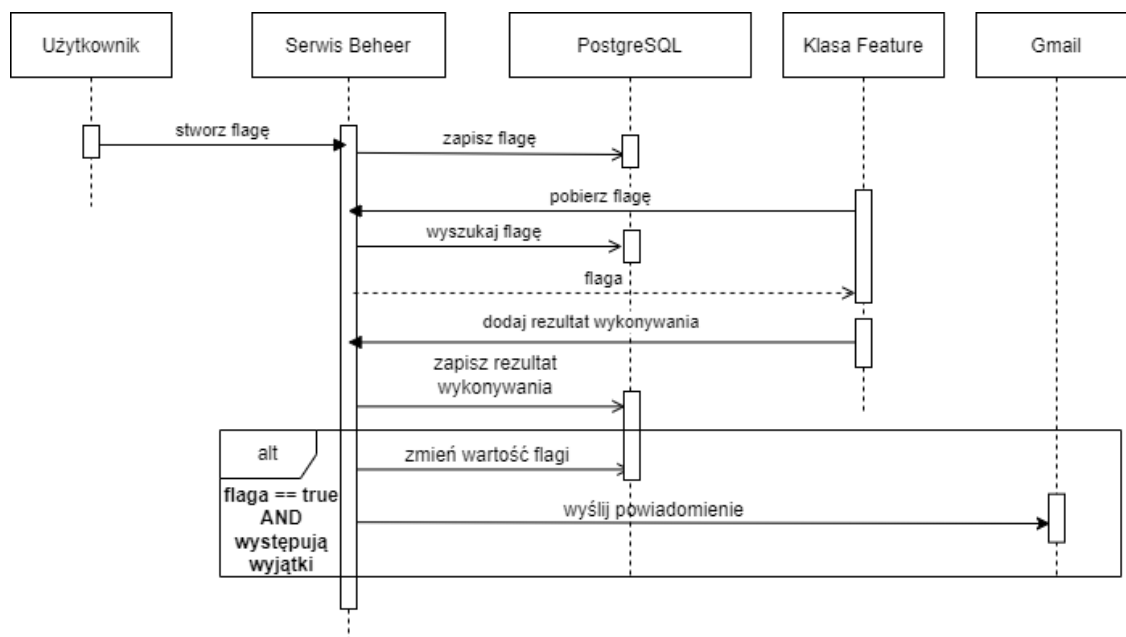
„Notacja UML umożliwia opisanie założeń, jak i dokumentowanie istniejącego systemu w sposób graficzny w postaci dedykowanych diagramów. Diagramy te pozwalają na formalne opisywanie i modelowanie struktur czy procesów w sposób zrozumiały zarówno dla zespołu technicznego, jak i interesariuszy” [38].

Jedną z metod definiowania wymagań funkcjonalnych systemu jest diagram przypadków użycia. Występują w nich aktorzy – czyli użytkownicy i systemy powiązane z naszym produktem, oraz przypadki użycia – czyli określone funkcje systemu. Aktorzy inicjują przypadki użycia korzystając z określonych funkcji systemu. Rysunek 4 przedstawia diagram przypadków użycia dla projektu serwisu Beheer.



Rys. 4. Diagram przypadków użycia [opracowanie własne]

Na rysunku 5 został przedstawiony diagram sekwencji projektu Beheer. Ukazuje on zakres obowiązków poszczególnych aktorów systemu. Identyfikowanie zakresu zachodzi poprzez definiowanie zdarzeń przepływających w systemie.



Rys. 5. Diagram sekwencji [opracowanie własne]

2.2.2 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 przyzwolecie na modyfikacje kodu i redystrybuowanie zmodyfikowanej wersji programu. Znajduje się w top 4 najczęściej wybieranych baz danych [39]. W rankingu pod uwagę brana, była liczba ofert pracy, zainteresowanie systemem mierzone systemem Google Trends, częstość wzmiankowania na forach dyskusyjnych i mediach społecznościowych [40]. Do głównych jego zalet należą najwyższy stopień zaawansowania [41]. Spełnia aż 170 na 179 głównych zasad określonych dla SQL [42]. Może przechowywać typy danych takie jak JSON, XML, Key-Value. Posiada rozbudowaną dokumentację. 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 platformy .Net należą:

- wieloplatformowość
- open-source – otwarty kod źródłowy
- jeden kod źródłowy (ang. „*single codebase*”)
- bezpieczeństwo wynikające z typowanie danych
- dedykowany manager pakietów Nuget

- kompilator JIT (ang. „*Just in time compiler*”) – kompilacja kodu odbywa się podczas wykonywania aplikacji, w ten sposób można ograniczyć czas kompilacji dużych i złożonych systemów
- Odśmiecacz pamięci (ang. „*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 [43]
- 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.

- 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 na 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. 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”” [44].

„React niesie ze sobą wiele korzyści, takich jak reużywalność komponentów, witralne drzewo DOM, jednokierunkowy przepływ danych, lekkość i stabilność, niski próg wejścia, łatwość w przetrzuceniu się na aplikacje mobilne” [45]. 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, 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. BUDOWA APLIKACJI

3.1 Implementacja

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” [33].

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 termin (ang. *deadline*) oddania gotowego produktu, a system wciąż wymaga nakładu sił i czasu programistów nazywamy „Crunch mode”.

W takiej sytuacji jest kilka wyjść:

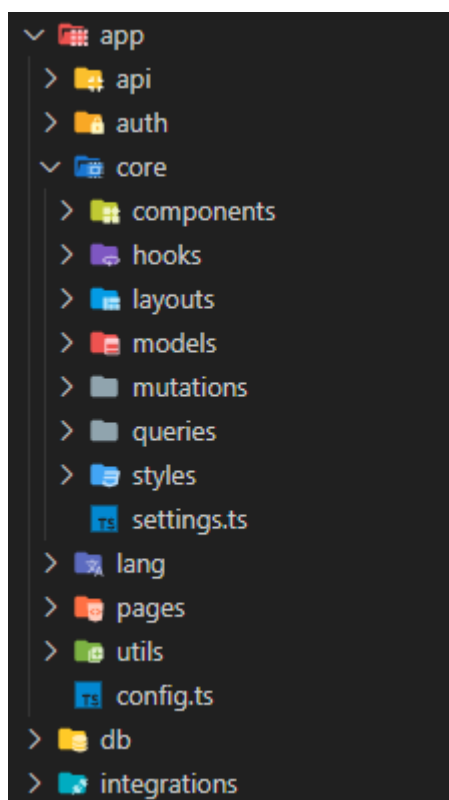
- nadgodziny
- wykonanie najbardziej znaczących funkcjonalności z perspektywy zleceniodawcy w pierwszej kolejności
- przydzielenie doświadczonych zespołów do trudniejszych zadań
- stworzenie prototypowego rozwiązania, a następnie stopniowe nadbudowywanie do wersji w pełni funkcjonalnej
- używanie zewnętrznych 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.

3.1.1 Aplikacja internetowa

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

Przedstawiona na rysunku 6 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 jakiegokolwiek zmiany, można być pewnym, że reszta modułów nie zmieni sposobu swojego działania.



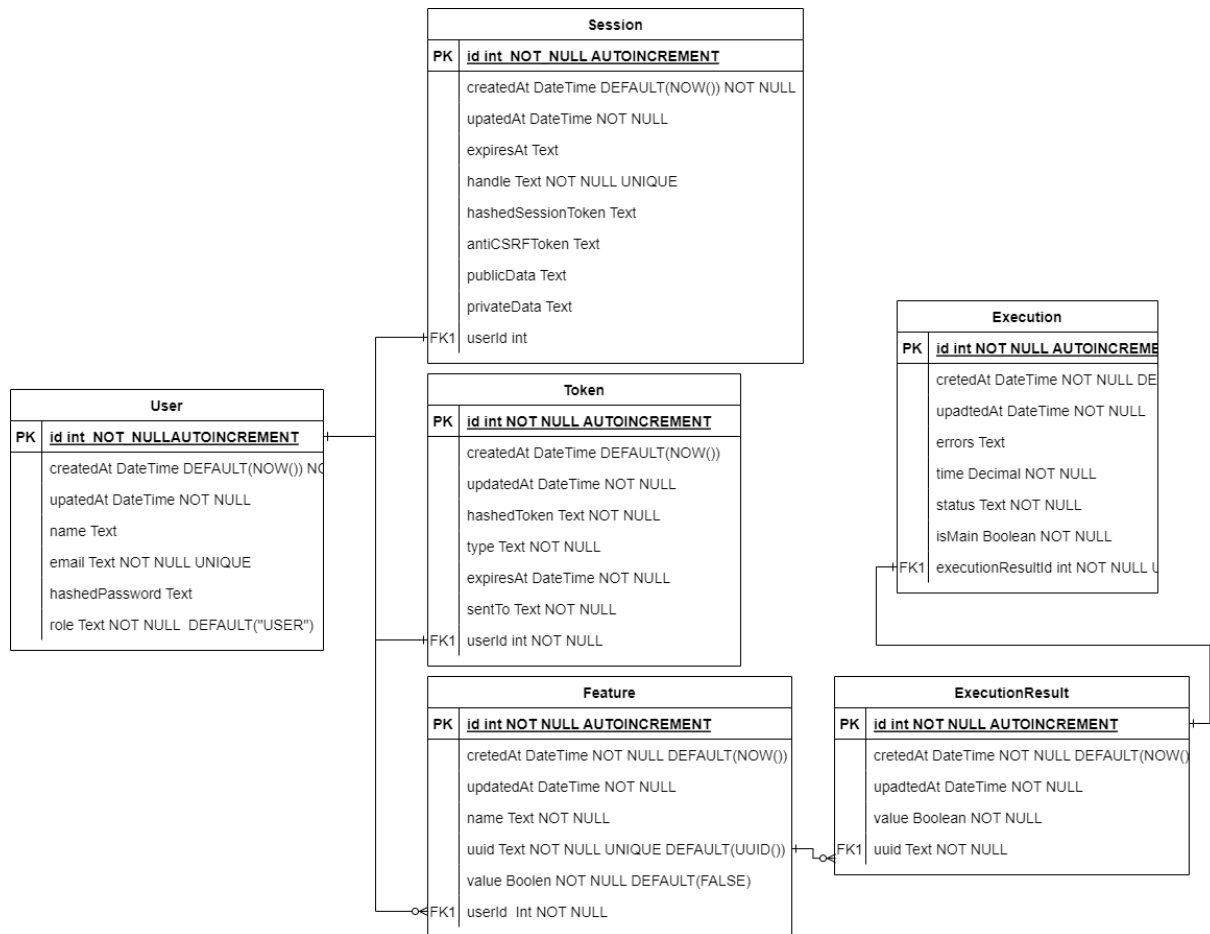
Rys. 6. Struktura plików w projekcie [opracowanie własne]

– 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 między nimi. Diagram ten ukazuje pewne związki między encjami. Relacje przedstawiane są za pomocą linii łączących. Na zakończeniach tych linii opisuje się związki.

Na rysunku 7 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.



Rys. 7. Diagram ERD [opracowanie własne]

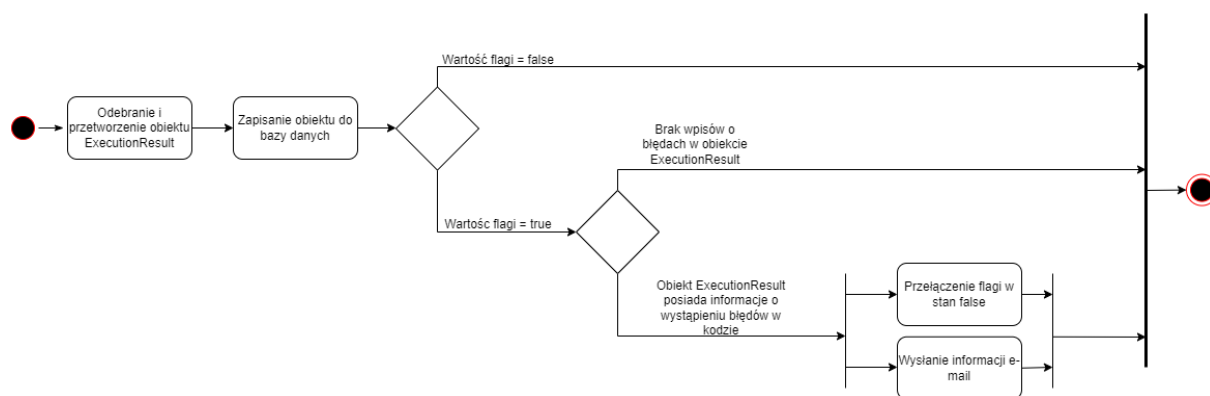
– Katalog „api”

Zawiera pliki odpowiadające za komunikację pomiędzy biblioteką programisty a systemem. To w nim realizowana jest odpowiedź na zapytanie o wartość flagi, odebranie wyniku z uruchomienia kodu, automatyczne przełączenie flagi w stan false czy wysłanie powiadomienia email. W tabeli 5 przedstawiono dwie struktury danych które służą do komunikacji serwis – klasa *Feature*.

Tabela 5. Struktury danych wymieniane w module API

<div data-bbox="354 264 639 425"> <pre> class FeatureInfo { + name: string + value: boolean } </pre> </div> <p>Struktura zwraca serwis na zapytanie o wartość flagi</p>	<div data-bbox="836 297 1398 528"> <pre> class ExecutionResult { + uuid: string + value: boolean + name: string + execution: Execution } class Execution { + status: string + errors: string + time: number + isMain: boolean } </pre> </div> <p>Struktura danych zawierająca podstawowe informacje o rezultacie wykonywania kodu. Struktura wysyłana do serwisu po zakończeniu wykonywania kodu.</p>
---	--

Aby lepiej zobrazować działanie tej części systemu przygotowano diagram aktywności ukazany na rysunku 8.



Rys. 8. Diagram aktywności – przetwarzanie wyniku uruchomienia kodu [opracowanie własne]

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 przestaje działać i wyświetla alert, informujący 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 posiada wartość false, to zostanie uruchomiony pierwotny kod programu. Także nie jest sprawdzana zawartość pola errors obiektu *ExecutionResult* – nie nastąpi przełączenie flagi w stan true. Takie rozwiązanie

podyktowane jest tym, że pierwotny kod powinien być niezawodny. Nie mniej 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 pocztce email poszkodowanych użytkowników, aż po włamania na konta do innych platform w sieci.

Ze strony użytkownika 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. [46]

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. 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 (ang. *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 [47].

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

Rys. 9. Fragment kodu realizujący hashowanie hasła [opracowanie własne]

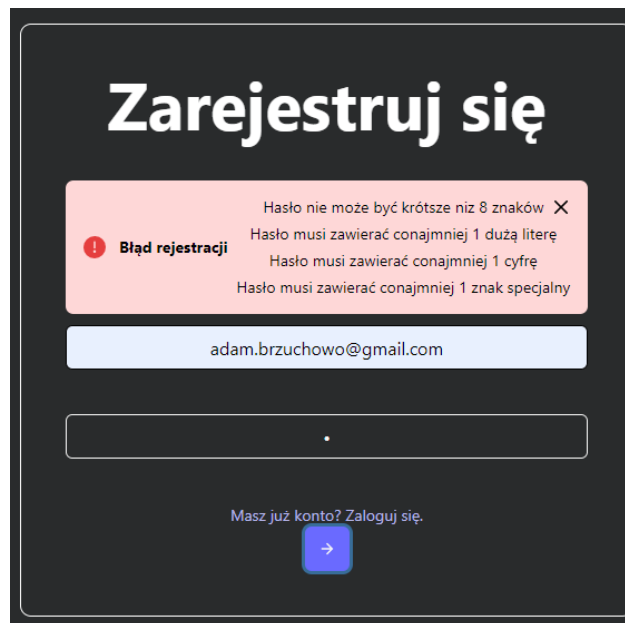
W projekcie zastosowano bibliotekę ZOD, wspomagającą walidowanie danych, zapewnia to, że programista otrzyma dokładnie takie dane jakich się spodziewa. Korzystając z zod tworzymy chain-method, z perspektywy dewelopera, jest to niezwykle czytelny i wygodny rodzaj API. Zastosowanie tejże biblioteki ukazano na rysunku 10.

```
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
}
```

Rys. 10. Funkcja walidująca dane wprowadzone w formularzu rejestracji [opracowanie własne]

Wyświetlanie błędów przedstawiono na rysunku 11. Funkcjonalność tą zrealizowano za pomocą komponentu Alert udostępnionego przez bibliotekę ChakraUI.



Rys. 11. Wyświetlenie błędów [opracowanie własne]

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 rysunku 12.

```
core > settings.ts > ...
const AppSettings = {
  errorCloseTimeout: 10_000,
}
export default AppSettings
```

Rys. 12. Fragment pliku z ustawieniami aplikacji – settings.ts [opracowanie własne]

- 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” [48].

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 13.

```

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>
  )
}

```

Rys. 13. Fragment komponentu wyświetlającego błędy [opracowanie własne]

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 opuszczeniu witryny czy przejściu do innej podstrony. Dobrym przykładem ukazującym wykorzystanie hooka useState jest, rysunek 14, 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>

```

Rys. 14. Fragment komponentu FeatureList [opracowanie własne]

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

- useEffect – wykonuje pewien fragment kodu zaraz po wyrenderowaniu strony
- useCurrentUser – zwraca obiekt zalogowanego użytkownika
- useClipboard – kopiuje tekst 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 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 (ang. *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 posiadamy informacje

o budowie obiektu (jego polach, metodach). Na rysunku 15 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.

```
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
  }
}
```

Rys. 15. Model klasy Feature [opracowanie własne]

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

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 16. Autor wykorzystał ten funkcję *random()* 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
  }
}

```

Rys. 16. Model klasy PostExecutionData [opracowanie własne]

- Moduł „mutations”

Katalog mutations, przechowuje wszystkie funkcje, użyte w systemie do modyfikacji danych przechowywanych w bazie danych.

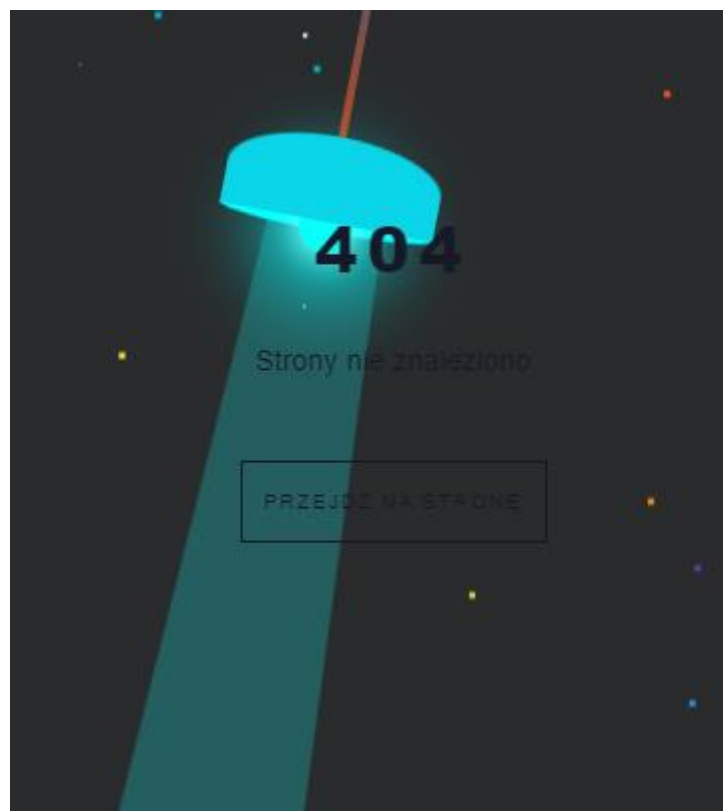
- Moduł „queries”

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 spełnić wymagania opisane w fazie określania wymagań, co do interfejsu graficznego, należało wyedytować domyślne style komponentów, lub stworzyć własny komponent i odpowiednio opisać formę prezentacji. Do tego służą właśnie kaskadowe arkusze stylów – w skrócie CSS (ang. *Cascading Style Sheets*).

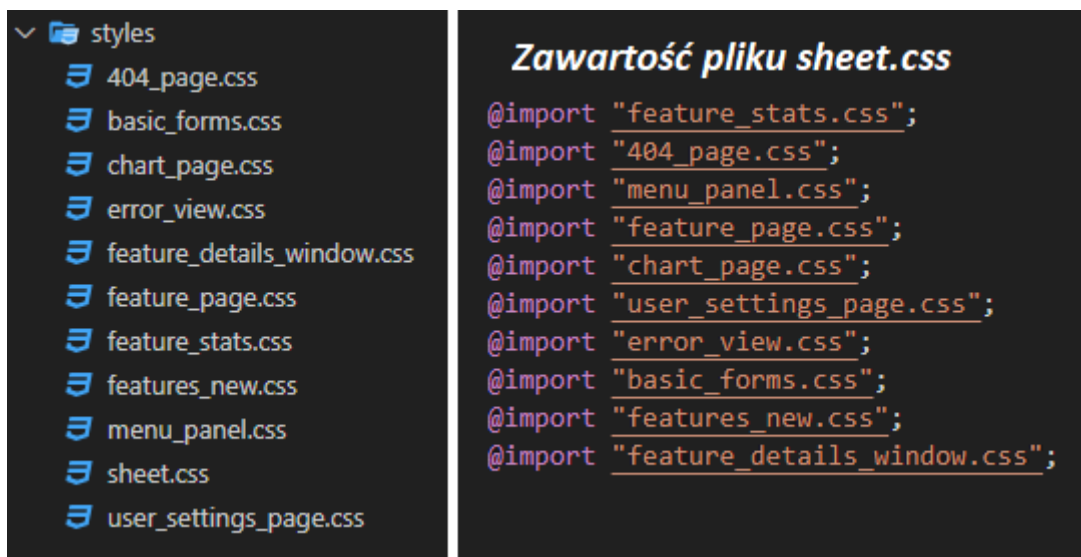
Należy wspomnieć o pliku 404_page.css którego zawartość pobrano i wdrożono z darmowego źródła, a efekt końcowy przedstawiono na rysunku 17 [49].



Rys. 17. Strona internetowa wyświetlająca błąd 404 [opracowanie własne]

Starano się stworzyć oddzielny plik css dla każdego głównego komponentu.

Plik przedstawiony na rysunku 18 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 ostylewanie elementów bazowych serwisu, które widoczne są na wszystkich widokach witryny.



Rys. 18. Widok katalogu styles i głównego pliku sheet.css [opracowanie własne]

– Plik Settings.ts

Myśląc o dalszym rozwoju aplikacji utworzono globalny plik konfiguracyjny aplikacji internetowej o nazwie - settings.ts. Przedstawiono go na rysunku 19, Parametr *errorCloseTimeout* określa po jakim czasie ma się ukryć komponent informujący o błędzie.

```
app > core > settings.ts > ...
1  const AppSettings = {
2    |   errorCloseTimeout: 10_000,
3  }
4  export default AppSettings
```

Rys. 19. Zawartość pliku z ustawieniami aplikacji internetowej [opracowanie własne]

– 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 20.

```

pl.views[AppViews.menu] = {
  features: { name: "Funkcjonalności" },
  settings: { name: "Ustawienia" },
  about: { name: "O Stronie" },
  logout: { name: "Wyloguj" },
  home: { name: "Strona Główna" },
}

en.views[AppViews.menu] = {
  features: { name: "Features" },
  settings: { name: "Settings" },
  about: { name: "About" },
  logout: { name: "Logout" },
  home: { name: "Home" },
}

```

Rys. 20. Implementacja tłumaczenia dla komponentu MenuWindow [opracowanie własne]

Tłumaczenie trafia do komponentów jako obiekt typu *Lang* za pośrednictwem parametru *props*. Typ *Lang* ukazany na rysunku 21, zawiera interfejs przez który programista dokonuje pobrania słownika – metoda *get()*, zmiany języka witryny – metoda *update()*, a parametr *currentLang* przechowuje aktualnie wybrany język.

```

export type Lang = {
  get: (view: AppViews) => any
  update: () => void
  currentLang: Languages
}

```

Rys. 21. Interfejs udostępniony do obsługi wielu języków [opracowanie własne]

Rysunek 22 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.

```

const MenuWindow = (props: MenuProps) => {
  const translation = props.lang.get(AppViews.menu)
  const updateLang = props.lang.update

  const [logoutMutation] = useMutation(logout)
}

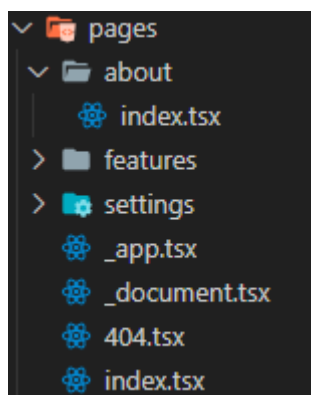
export enum AppViews {
  join = "join",
  home = "home",
  login = "login",
  signUp = "signUp",
  register = "register",
  about = "about",
  menu = "menu",
  features = "features",
  featureDetails = "featureDetails",
  settings = "settings",
}

```

Rys. 22. Prezentacja pobrania tłumaczenia dla komponentu *MenuWindow* [opracowanie własne]

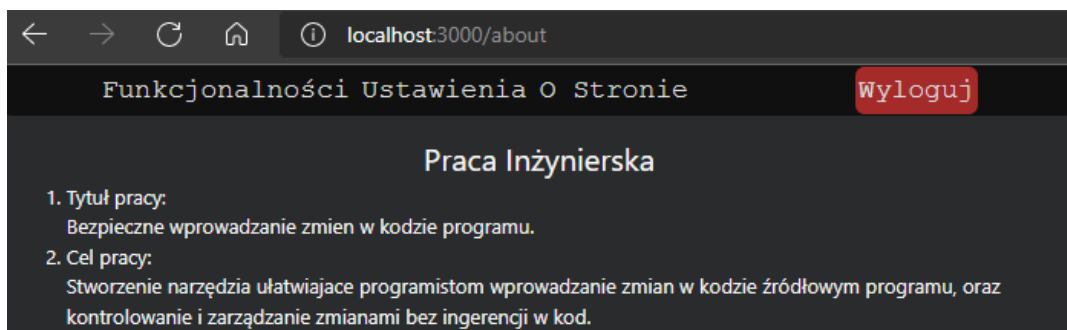
- „pages”

Katalog „pages”, przechowuje pliki stron internetowych. Na rysunku 23 ukazano strukturę katalogów, w jakiej są one przechowywane.



Rys. 23. Prezentacja zawartości katalogu „pages” [opracowanie własne]

Nie jest to przypadkowe ustawienie, ponieważ specyficzną cechą frameworka Blitz.js jest tak zwane dynamiczne trasowanie (ang. *Dynamic Routes*). To mechanizm który konwertuje fizyczną ścieżkę do pliku na ścieżkę URL (ang. *Uniform Resource Locator*). Przykładem może być witryna przedstawiona na rysunku 24. Ścieżka URL /about, która prowadzi do pliku index.tsx znajdującego się w folderze about.



Rys. 24. Prezentacja fragmentu strony - menu [opracowanie własne]

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 25. Do generowania tabeli użyto tablicy obiektów typu *Feature*, przechowywanych w stanie komponentu *FeatureList* (rysunek 14). Wybór tego rozwiązania pozwolił na umieszczenie przycisków kopiuń – zielony z ikoną dwóch kartek nakładających na siebie, przycisk typu przełącznik (ang. „switch”) – odpowiadający za przełączanie stanu flagi, oraz przycisk usuń – czerwony

opatrzone ikoną 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 ostrylować i wdrożyć, urozmaicając 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 28.



Rys. 25. Prezentacja fragmentu listy flag [opracowanie własne]

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 26, aby utworzyć flagę wymagane jest wprowadzenie nazwy funkcjonalności i zatwierdzenie poprzez kliknięcie zielonego przycisku.

Rys. 26. Prezentacja formularza tworzenia flagi [opracowanie własne]

Implementację wyszukiwania funkcjonalności zaprezentowano na rysunku 27.

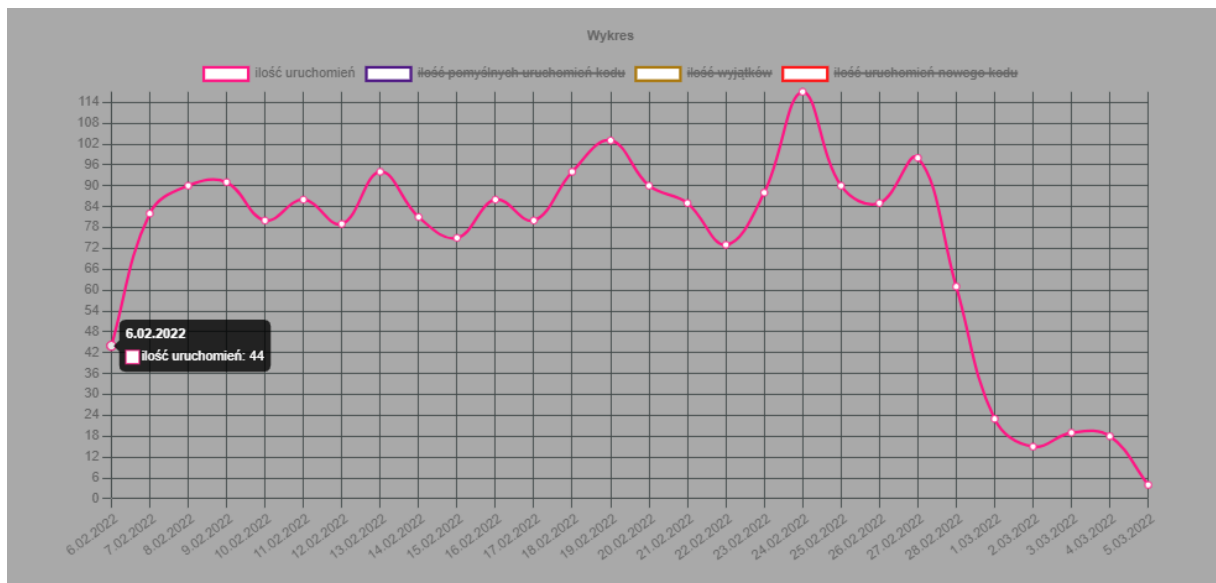
Lista obiektów typu *Feature*, na podstawie której tworzone tabelkę (rysunek 25) przechowywana jest w stanie komponentu *FeatureList*. Za każdym razem gdy następuje przeładowanie strony lista dostępnych funkcjonalności zostaje pobrana z bazy danych i zapisana do stanu komponentu - zmiennej *features*. 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.

```
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())
      )
    )
  }
}
```

Rys. 27. Implementacja wyszukiwania flagi [opracowanie własne]

Wdrożono graficzne przedstawienie danych w postaci wykresu, znacznie ułatwia to analizę. Domyślnie wykres prezentuje dane z ostatnich 30 dni. Przedstawiony na rysunku 28 wykres posiada 4 zbiory danych. Każdym z nich można manipulować - włączając lub wyłączając poprzez przyciski znajdujące się na górze wykresu.



Rys. 28. Widok wykresu [opracowanie własne]

Linie wykresu tworzone są na podstawie obiektu *DataSet* z rysunku 29. 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.

```
type DataSet = {
  label: string
  data: number[]
}
```

Rys. 29. Obiekt *DataSet* – definiuje pola potrzebne do utworzenia linii wykresu [opracowanie własne]

Autor aby zachować przejrzystość i z myślą o dalszym rozwoju, zbudował klasę, która przyjmuje tablicę obiektów *PostExecuteData*. Wewnątrz obiektu *ChartDataAdapter*, odbywa się szereg kalkulacji. Korzystając z metod udostępnionych przez interfejs api obiektu, który przedstawiono na rysunku 30, otrzymujemy dane potrzebne do sporządzenia wykresu. Zastosowano uniwersalną konstrukcję funkcji. Produkując dane do wykresu z przekazanego parametru, to programista decyduje jaki okres czasu przedstawi na wykresie.

```
export interface IChartDataAdapter {
  prepareChartData: (fromData: PostExecutionData[]) => ChartData
}
```

Rys. 30. Interfejs udostępniający metodę tworzącą dane do przygotowania wykresu [opracowanie własne]

Obiekt *ChartData* przedstawiono na rysunku 31. *ChartEntity* to podstawowy byt nakładany na wykres. Posiada opis (label) oraz pola typu number.

```
class ChartData {  
  data = [] as ChartEntity[]  
}  
  
class ChartEntity {  
  label: string  
  countOfExecutions: number  
  countOfSuccess: number  
  countOfErrors: number  
  countOfTrueValue: number  
  countOfFalseValue: number  
  
  constructor(label?: string, numOfExecutions = 0) {  
    if (label) this.label = label  
    if (numOfExecutions) this.countOfExecutions = numOfExecutions  
  
    this.countOfSuccess = 0  
    this.countOfTrueValue = 0  
    this.countOfErrors = 0  
  }  
}
```

Rys. 31. Widok struktur danych potrzebnych do przygotowania wykresu [opracowanie własne]

Konwertowanie danych do postaci *DataSet* fragment ukazano to na rysunku 32

```
label: "ilość uruchomień",  
data: chart.data.map((i) => i.countOfExecutions),
```

Rys. 32. Konwertowanie danych na obiekt *DataSet* [opracowanie własne]

– 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 biblioteką 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 33.

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

Rys. 33. Definiowanie bazy danych [opracowanie własne]

Prisma do generowania tabel potrzebuje pewnego schematu. Na rysunku 34 ukazano model tabeli Feature. Deklaracja takiego modelu zawiera wszystkie potrzebne informacje do stworzenia tabeli w bazie danych.

Pole oznaczone za pomocą *@updatedAt*, będzie zawierać dokładną datę i czas edycji rekordu. Proces aktualizacji tego pola ten jest automatyczny.

Znacznik *@relation* opisuje relacje jakie zachodzą między tabelami. Z rysunku 34 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*.


```

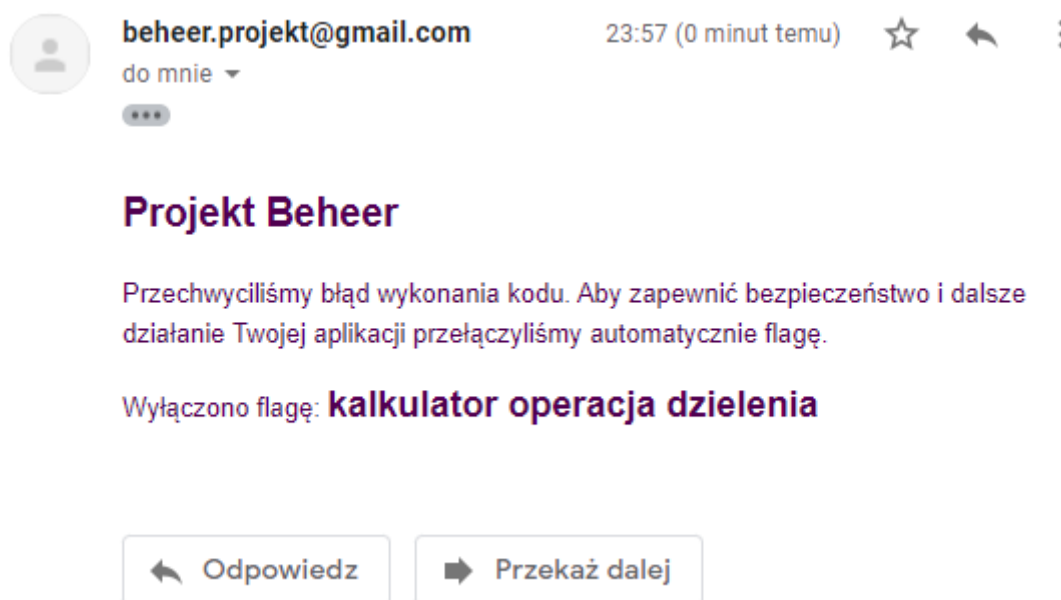
model Feature {
  id          Int          @id @default(autoincrement())
  createdAt   DateTime     @default(now())
  updatedAt   DateTime     @updatedAt
  name        String
  uuid        String       @unique @default(uuid())
  value       Boolean       @default(false)
  user        User         @relation(fields: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

```

Rys. 34. Schemat tabeli Feature [opracowanie własne]

– 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.



Rys. 35. Widok wiadomości email [opracowanie własne]

Aby zapewnić większą wydajność autor postanowił zastosować jeden statyczny obiekt służący do wysyłania wiadomości email. Statyczne pole transporter jest inicjowane

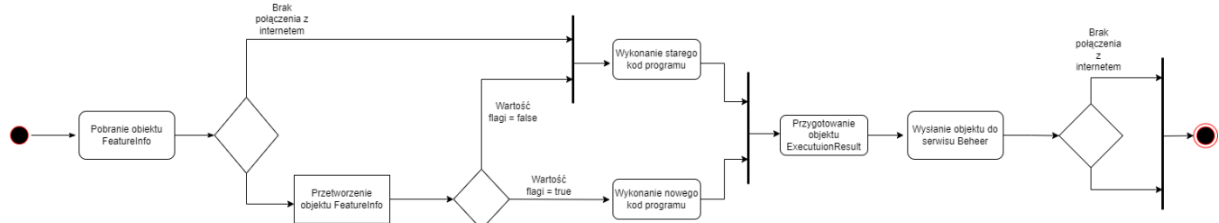
tylko raz w czasie całego działania aplikacji. Taki wzorec projektowy nazwano – singleton. Dodatkowo implementacja statycznej metody `send()`, pozwala na wysyłanie maili z każdego miejsca w programie. Fragment opisywanego kodu źródłowego ukazano na rysunku 36.

```
class Gmail {  
    private static transporter  
    private static isSetup = false  
  
    public static init() {  
        if (this.isSetup) return  
  
        this.transporter = nodemailer.createTransport({  
            service: "gmail",  
            auth: {  
                user: GMAIL.ADRES_EMAIL,  
                pass: GMAIL.PASSWORD,  
            },  
        })  
        this.isSetup = true  
    }  
  
    public static send(to: string, featurName: string) {  
    }  
}
```

Rys. 36. Implementacja wzorca singleton w klasie Gmail [opracowanie własne]

3.1.2 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 zobrazowano na rysunku 37.



Rys. 37. Diagram aktywności dla klasy Feature [opracowanie własne]

Klasa *Feature* implementuje interfejs *IFeature*, przedstawiony na rysunku 38.

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

Rys. 38. Interfejs udostępniony do komunikacji z klasą *Feature* [opracowanie własne]

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 hermetyzacji mówi, aby udostępniać na zewnątrz możliwie jak najmniej [50]. 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 39. 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 i nie osiągnie celu biznesowego.

```
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;
}
```

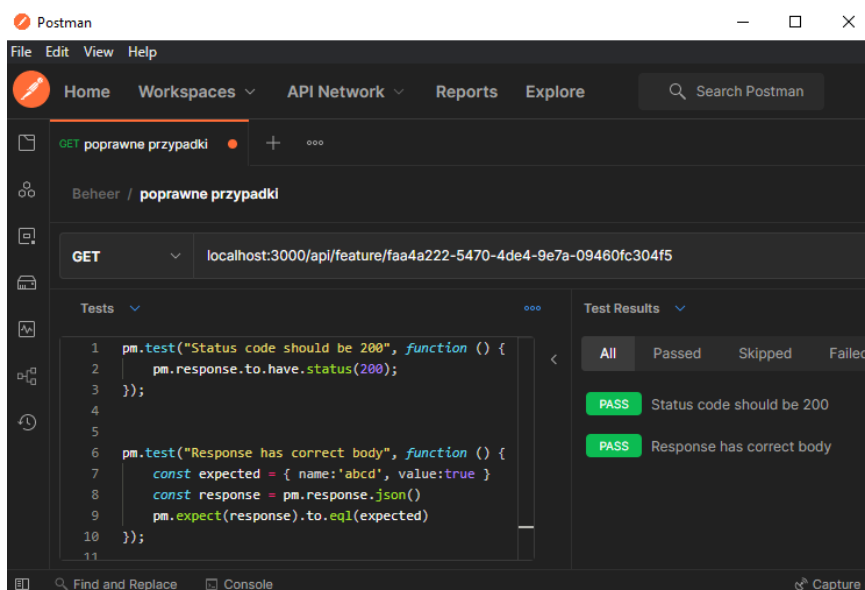
Rys. 39. Implementacja wykonywania kodu [opracowanie własne]

3.2 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 działania systemu wymagane jest sprawdzenie wszystkich możliwych przypadków danych wejściowych. Każdy zestaw parametrów wejściowych przekazywanych do systemu, można nazwać przypadkiem testowym (ang. *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 40. Pierwszy z nich sprawdza status odpowiedzi protokołu http - 200 oznacza poprawną realizację żądania. Natomiast drugi porównuje pobrane dane do tych, których oczekujemy.



Rys. 40. Widok wykonanych testów [opracowanie własne]

3.3 Eksploatacja


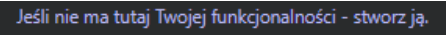

Produkt budowany był z myślą o 5 aspektach które opisane są jako definicja używalności (ang. *usability*) sformułowana przez Jakob Nielsen [52]

- 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. Natomiast jedną z miar efektywności systemu jest lista kroków – im dłuższa lista tym system jest mniej efektywny.


3.3.1 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 

3.3.2 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

3.3.3 Przykład implementacji

Dobrym przykładem jest sytuacja w której wykonujemy dzielenie przez zero. Matematyka udowadnia, że nie ma to „sensu obliczeniowego” [53]. Na rysunku 41, 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.

```
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.ControlledBy("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();
        });
    }
}
```

Rys. 41. Sekcja zmiennych programu Kalkulator [opracowanie własne]

PODSUMOWANIE

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 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 rollout. Wprowadzenie nowej funkcjonalności do serwisu Beheer, umożliwiającego procentowe wdrażanie zmian. Przykładowy widok realizacji zaprezentowano na rysunku 42.



Rys. 42. Widok funkcjonalności - rollout [opracowanie własne]

- Wprowadzenie witryny wyświetlającej ramki stosu (ang. *Stack trace*). Wyświetlenie ramek stosu umożliwia szybszą nawigację do pliku i dokładne prześledzenie

- ścieżki wykonywanego kodu. Jest to kolejna funkcjonalność która zwiększyłaby 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 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 pewnego zakresu danych z wykresu (rysunek 28), w formatach CSV / JSON / XML. Funkcjonalność wspomógłby analityków systemu, w przeprowadzaniu analiz..
 - Responsywność aplikacji internetowej – wsparcie dla urządzeń mobilnych
 - Sortowanie tabeli z funkcjonalnościami po nazwie, dacie utworzenia, wartościach flagi
 - Pełne wsparcie językowe. Podstawowa wersja aplikacji implementuje, funkcjonalny mechanizm zmiany języka który ukazano na rysunku 43, lecz tłumaczenie posiada tylko menu aplikacji.



Rys. 43. Widok komponentu MenuWindow – zmiana języka strony [opracowanie własne]

LITERATURA

- [1] „MDN web docs” [Online] Available:
https://developer.mozilla.org/pl/docs/Learn/JavaScript/First_steps/What_is_JavaScript
[Data uzyskania dostępu: 15.03.2022]
- [2] „TypeScript - Documentation” [Online] Available:
<https://www.typescriptlang.org/pl/docs/handbook/intro.html> [Data uzyskania dostępu:
15.03.2022]
- [3] „Wikipedia – Node.js” [Online] Available: <https://pl.wikipedia.org/wiki/Node.js>
[Data uzyskania dostępu: 15.03.2022]
- [4] „Wikipedia - UML” [Online] Available:
https://pl.wikipedia.org/wiki/Unified_Modeling_Language [Data uzyskania dostępu:
15.03.2022]
- [5] „Wikipedia - JSON” [Online] Available: <https://pl.wikipedia.org/wiki/JSON> [Data
uzyskania dostępu: 15.03.2022]
- [6] „Wikipedia - XML” [Online] Available: <https://pl.wikipedia.org/wiki/XML> [Data
uzyskania dostępu: 15.03.2022]
- [7] „Wikipedia – Key-Value” [Online] Available:
https://pl.wikipedia.org/wiki/Tablica_asocjacyjna [Data uzyskania dostępu: 15.03.2022]
- [8] „Wget” [Online] Available: <https://osworld.pl/wget-nie-tylko-dla-poczatkujacych/>
[Data uzyskania dostępu: 15.03.2022]
- [9] „Wikipedia - cURL” [Online] Available: <https://pl.wikipedia.org/wiki/CURL> [Data
uzyskania dostępu: 15.03.2022]
- [10] „Axios HTTP i Node.js” [Online] Available: <https://boringowl.io/tag/axios/> [Data
uzyskania dostępu: 15.03.2022]
- [11] „Czym są mikrousługi” [Online] Available:
<https://www.intel.pl/content/www/pl/pl/cloud-computing/microservices.html> [Data
uzyskania dostępu: 15.03.2022]
- [12] „Wikipedia - API” [Online] Available:
https://pl.wikipedia.org/wiki/Interfejs_programowania_aplikacji [Data uzyskania dostępu:
15.03.2022]
- [13] „Wikipedia” [Online] Available: https://pl.wikipedia.org/wiki/Google_Trends
[Data uzyskania dostępu: 15.03.2022]

- [14] „Ruby on Rails” [Online] Available: <https://boringowl.io/tag/ruby-on-rails/> [Data uzyskania dostępu: 15.03.2022]
- [15] „Wikipedia - Framework” [Online] Available: <https://pl.wikipedia.org/wiki/Framework> [Data uzyskania dostępu: 15.03.2022]
- [16] „Next.js” [Online] Available: https://geek.justjoin.it/next-js-czyli-react-server-side-rendering-cz-1#Czym_jest_Nextjs [Data uzyskania dostępu: 15.03.2022]
- [17] „Wikipedia – protokół TLS” [Online] Available: https://pl.wikipedia.org/wiki/Transport_Layer_Security [Data uzyskania dostępu: 15.03.2022]
- [18] „Wikipedia - HTML” [Online] Available: <https://pl.wikipedia.org/wiki/HTML> [Data uzyskania dostępu: 15.03.2022]
- [19] „Co to jest UUID” [Online] Available: <https://ichi.pro/pl/co-to-jest-uuid-i-jak-sa-generowane-26322457686579> [Data uzyskania dostępu: 15.03.2022]
- [20] „Kurs JavaScript” [Online] Available: <http://kursjs.pl/kurs/dom/dom.php> [Data uzyskania dostępu: 15.03.2022]
- [21] „Czym się różni UX design od UI design” [Online] Available: <https://ux.marszalkowski.org/ux-vs-ui-czym-sie-rozni-ux-design-od-ui-design/> [Data uzyskania dostępu: 15.03.2022]
- [22] „Wikipedia - sól” [Online] Available: [https://pl.wikipedia.org/wiki/S%C3%B3l_\(kryptografia\)](https://pl.wikipedia.org/wiki/S%C3%B3l_(kryptografia)) [Data uzyskania dostępu: 15.03.2022]
- [23] „Wikipedia - URL” [Online] Available: https://pl.wikipedia.org/wiki/Uniform_Resource_Locator [Data uzyskania dostępu: 15.03.2022]
- [24] „Wikipedia - SQL” [Online] Available: <https://pl.wikipedia.org/wiki/SQL> [Data uzyskania dostępu: 15.03.2022]
- [25] „Wikipedia - Hermetyzacja” [Online] Available: [https://pl.wikipedia.org/wiki/Hermetyzacja_\(informatyka\)](https://pl.wikipedia.org/wiki/Hermetyzacja_(informatyka)) [Data uzyskania dostępu: 15.03.2022]
- [26] „Wikipedia - Wyjątek” [Online] Available: <https://pl.wikipedia.org/wiki/Wyj%C4%85tek> [Data uzyskania dostępu: 15.03.2022]
- [27] „Wikipedia - IDE” [Online] Available: https://pl.wikipedia.org/wiki/Zintegrowane_%C5%9Brodowisko_programistyczne [Data uzyskania dostępu: 15.03.2022]

- [28] „Wikipedia – Stack trace” [Online] Available: https://pl.wikipedia.org/wiki/Stack_trace [Data uzyskania dostępu: 15.03.2022]
- [29] „Jak obsługiwać i zgłaszać błędy” [Online] Available: <https://gogomedia.pl/blog/zarzadzanie-projektami/jak-obslugiwac-i-zglaszac-bledy-w-web-aplikacji/> [Data uzyskania dostępu: 15.03.2022]
- [30] „4 największe katastrofy programistyczne” [Online] Available: <https://tech.wp.pl/4-najwieksze-katastrofy-programistyczne-w-historii-drobne-bledy-o-tragicznych-konsekwencjach,6034853152010881a> [Data uzyskania dostępu: 15.03.2022]
- [31] „Najgorsze skutki błędów programistów” [Online] Available: <https://www.computerworld.pl/news/Najgorsze-skutki-bledow-programistow,369992.html> [Data uzyskania dostępu: 15.03.2022]
- [32] „Wikipedia – inżynieria oprogramowania” [Online] Available: https://pl.wikipedia.org/wiki/In%C5%BCynieria_oprogramowania [Data uzyskania dostępu: 15.03.2022]
- [33] K. Sacha, „Inżynieria oprogramowania” Wydawnictwo Naukowe PWN 2021
- [34] B. Bruegge, A. H. „Inżynieria oprogramowania w ujęciu obiektowym. UML, wzorce projektowe i Java” Helion
- [35] „Program komputerowy” [Online] Available: <https://www.korektortekstu.pl/definicja/program-komputerowy> [Data uzyskania dostępu: 15.03.2022]
- [36] „Język programowania” [Online] Available: https://www.szkolnictwo.pl/szukaj.J%C4%99zyk_programowania [Data uzyskania dostępu: 15.03.2022]
- [37] G. Coldwind „Zrozumieć programowanie” Wydawnictwo Naukowe PWN 2022
- [38] „Jak stosować UML” [Online] Available: <https://www.jcommerce.pl/jpro/artykuly/podstawy-uml-czyli-modelowanie-dla-kazdego> [Data uzyskania dostępu: 15.03.2022]
- [39] „DB-Engines Ranking” [Online] Available: <https://db-engines.com/en/ranking> [Data uzyskania dostępu: 15.03.2022]
- [40] „DB-Engines Ranking - Method” [Online] Available: https://db-engines.com/en/ranking_definition [Data uzyskania dostępu: 15.03.2022]
- [41] „PostgreSQL” [Online] Available: <https://www.postgresql.org/> [Data uzyskania dostępu: 15.03.2022]

- [42] „Wikipedia – SQL compliance” [Online] Available:
https://en.wikipedia.org/wiki/SQL_compliance [Data uzyskania dostępu: 15.03.2022]
- [43] „Blitz.js” [Online] Available: <https://blitzjs.com/> [Data uzyskania dostępu: 15.03.2022]
- [44] „Samouczek: wstęp do Reacta” [Online] Available:
<https://pl.reactjs.org/tutorial/tutorial.html> [Data uzyskania dostępu: 15.03.2022]
- [45] „ReactJS I React Native – czym się różnią?” [Online] Available:
<https://geek.justjoin.it/reactjs-i-react-native-czym-sie-roznia> [Data uzyskania dostępu: 15.03.2022]
- [46] K. Sacha, „Inżynieria oprogramowania” Wydawnictwo Naukowe PWN 2021.
- [47] „Jak utworzyć bezpieczne hasło” [Online] Available: <https://www.gov.pl/web/baza-wiedzy/jak-tworzyc-bezpieczne-hasla> [Data uzyskania dostępu: 15.03.2022]
- [48] „Argon2 Hash Generator” [Online] Available: <https://argon2.online/> [Data uzyskania dostępu: 15.03.2022]
- [49] „Komponenty i właściwości - React” [Online] Available:
<https://pl.reactjs.org/docs/components-and-props.html> [Data uzyskania dostępu: 15.03.2022]
- [50] „58 HTML 404 Page Templates” [Online] Available <https://freefrontend.com/html-css-404-page-templates/> [Data uzyskania dostępu: 15.03.2022]
- [51] - „Co to jest hermetyzacja” [Online] Available
<https://www.modestprogrammer.pl/co-to-jest-hermetyzacja-w-programowaniu-objektowym> [Data uzyskania dostępu: 15.03.2022]
- [52] „Wikipedia – Użyteczność (informatyka)” [Online] Available:
[https://pl.wikipedia.org/wiki/U%C5%BCyteczno%C5%9B%C4%87_\(informatyka\)](https://pl.wikipedia.org/wiki/U%C5%BCyteczno%C5%9B%C4%87_(informatyka)) [Data uzyskania dostępu: 15.03.2022]
- [53] „Wikipedia – Dzielenie przez zero” [Online] Available:
https://pl.wikipedia.org/wiki/Dzielenie_przez_zero [Data uzyskania dostępu: 15.03.2022]

SPIS RYSUNKÓW

- Rys. 1. Implementacja flagi w postaci zmiennej statycznej [opracowanie własne]
- Rys. 2. Model tabeli Feature w bazie danych [opracowanie własne]
- Rys. 3. Model kaskadowy [opracowanie własne]
- Rys. 4. Diagram przypadków użycia [opracowanie własne]
- Rys. 5. Diagram sekwencji [opracowanie własne]
- Rys. 6. Struktura plików w projekcie [opracowanie własne]
- Rys. 7. Diagram ERD [opracowanie własne]
- Rys. 8. Diagram aktywności – przetwarzanie wyniku uruchomienia kodu [opracowanie własne]
- Rys. 9. Fragment kodu realizujący hashowanie hasła i zapisanie do bazy danych
- Rys. 10. Funkcja walidująca dane wprowadzone w formularzu rejestracji [opracowanie własne]
- Rys. 11. Wyświetlenie błędów [opracowanie własne]
- Rys. 12. Fragment pliku z ustawieniami aplikacji – settings.ts [opracowanie własne]
- Rys. 13. Fragment komponentu wyświetlającego błędy [opracowanie własne]
- Rys. 14. Fragment komponentu FeatureList [opracowanie własne]
- Rys. 15. Model klasy Feature [opracowanie własne]
- Rys. 16. Model klasy PostExecutionData [opracowanie własne]
- Rys. 17. Strona internetowa wyświetlająca błąd 404 [opracowanie własne]
- Rys. 18. Widok katalogu styles i głównego pliku sheet.css [opracowanie własne]
- Rys. 19. Zawartość pliku z ustawieniami aplikacji internetowej [opracowanie własne]
- Rys. 20. Implementacja tłumaczenia dla komponentu Menu [opracowanie własne]
- Rys. 21. Interfejs udostępniony do obsługi wielu języków [opracowanie własne]
- Rys. 22. Prezentacja pobrania tłumaczenia dla komponentu MenuWindow [opracowanie własne]
- Rys. 23. Prezentacja zawartości katalogu pages [opracowanie własne]
- Rys. 24. Prezentacja fragmentu strony - menu [opracowanie własne]
- Rys. 25. Prezentacja fragmentu listy flag [opracowanie własne]
- Rys. 26. Prezentacja formularza tworzenia flagi [opracowanie własne]

Rys. 27. Implementacja wyszukiwania flagi [opracowanie własne]

Rys. 28. Widok wykresu [opracowanie własne]

Rys. 29. Obiekt DataSet – definiuje pola potrzebne do utworzenia linii wykresu [opracowanie własne]

Rys. 30. Interfejs udostępniający metodę tworzącą dane do przygotowania wykresu [opracowanie własne]

Rys. 31. Widok struktur danych potrzebnych do przygotowania wykresu [opracowanie własne]

Rys. 32. Konwertowanie danych na obiekt DataSet [opracowanie własne]

Rys. 33. Definiowanie bazy danych [opracowanie własne]

Rys. 34. Schemat tabeli Feature [opracowanie własne]

Rys. 35. Widok wiadomości email [opracowanie własne]

Rys. 36. Implementacja wzorca singleton w klasie Gmail [opracowanie własne]

Rys. 37. Diagram aktywności dla klasy Feature [opracowanie własne]

Rys. 38. Interfejs udostępniony do komunikacji z klasą Feature [opracowanie własne]

Rys. 39. Implementacja wykonywania kodu [opracowanie własne]

Rys. 40. Widok wykonanych testów [opracowanie własne]

Rys. 41. Sekcja zmiennych programu Kalkulator [opracowanie własne]

Rys. 42. Widok funkcjonalności - Rollout [opracowanie własne]

Rys. 43. Widok komponentu MenuWindow – zmiana języka strony [opracowanie własne]

SPIS TABEL

Tabela 1. Zestawienie konkurencyjnych narzędzi [DODAĆ LINK]

Tabela 2. Wymagania funkcjonalne – rejestracja

Tabela 3. Wymagania funkcjonalne – logowanie

Tabela 4. Wymagania funkcjonalne – tworzenie flagi

Tabela 5. Struktury danych wymieniane w module API

STRESZCZENIA

Streszczenie w języku polskim

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.

Summary in English

The topic of this thesis is "Safe implementation of changes to the program code."

The goal of the author is to create a fully functioning web application to control the introduced functionalities, and to build a Feature class that enables safe implementation of these functionalities in the target program code.

The author follows good software engineering practices, selects a cascade model and guides readers through each phase to finally deliver the finished product. This is

mainly due to the carefully conducted project analysis phase. Artifacts in the form of numerous diagrams and a reasonable choice of technology contributed greatly to the success achieved.

To implement this system, he chose the high-level TypeScript languages together with the Blitz.js framework - Fullstack React Framework, and C # used to build the Feature class. He also applied many solutions to accelerate the creation of the product, such as: validation with the ZOD library, use of default styled components and icons from the ChakraUI library.

The use of design patterns for classes ensures the extensibility of the system.

A very important aspect was building a coherent, comfortable and intuitive website. When designing the user interface, the author took into account 5 features that define the usability of the application as defined by Jakob Neilson.