Obraz zawierający tekst, Czcionka, zrzut ekranu, logo

Opis wygenerowany automatycznie

**Damian Biskupski**

**236503**

PRACA DYPLOMOWA

inżynierska

na kierunku Informatyka Stosowana

**Aplikacja webowa do zamawiania i automatyzowania procesu wytwarzania własnoręcznie robionych gier planszowych**

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej

**Promotor:** dr inż. Joanna Ochelska-Mierzejewska

Łódź 2023

**Spis treści**

[1. Wstęp 3](#_Toc151915626)

[1.1. Problematyka i zakres pracy 3](#_Toc151915627)

[1.2. Cele pracy 3](#_Toc151915628)

[1.3. Przegląd literatury 4](#_Toc151915629)

[1.4. Układ pracy 4](#_Toc151915630)

[2. Przegląd wybranych aplikacji wspomagających proces zamawiania i projektowanie gier planszowych 5](#_Toc151915631)

[2.1. Aplikacja webowa StoryboardThat 5](#_Toc151915632)

[2.2. Aplikacja webowa TableTopia 6](#_Toc151915633)

[2.3. Aplikacja webowa Olx 8](#_Toc151915634)

[2.4. Zalety i wady wybranych aplikacji do zamawiania i wspomagania projektowania gier planszowych 9](#_Toc151915635)

[3. Stos technologiczny 10](#_Toc151915636)

[3.1. Uzasadnienie wyboru technologii 10](#_Toc151915637)

[3.2. Język programowania TypeScript 12](#_Toc151915638)

[3.3. Framework NestJs 12](#_Toc151915639)

[3.4. Framework React 13](#_Toc151915640)

[3.5. Baza danych PostgreSQL 14](#_Toc151915641)

[3.6. System kolejkowania BullMQ 15](#_Toc151915642)

[3.7. Pakiet Mailer 16](#_Toc151915643)

[3.8. Pakiet JWT 16](#_Toc151915644)

[3.9. Pakiet Schedule 17](#_Toc151915645)

[4. Cykl projektowy aplikacji webowej BoardFlow 18](#_Toc151915646)

[4.1. Sylwetka klienta i jego wymagania 18](#_Toc151915647)

[4.2. Wymagania funkcjonalne 20](#_Toc151915648)

[4.3. Wymagania niefunkcjonalne 22](#_Toc151915649)

[4.4. Architektura aplikacji 23](#_Toc151915650)

[4.4.1. Warstwa prezentacji 24](#_Toc151915651)

[4.4.2. Warstwa logiki biznesowej 25](#_Toc151915652)

[4.4.3. Warstwa danych 27](#_Toc151915653)

[4.5. Implementacja – punkty kluczowe 31](#_Toc151915654)

[4.5.1. Modularna budowa projektu 31](#_Toc151915655)

[4.5.2. Implementacja personalnych dekoratorów 37](#_Toc151915656)

[4.5.3. Implementacja personalnych guards 38](#_Toc151915657)

[4.5.4. Realizacja kolejkowania 42](#_Toc151915658)

[4.5.5. Realizacja JWT 44](#_Toc151915659)

[4.6. Testy 45](#_Toc151915660)

[4.7. Instalacje i konserwacja 45](#_Toc151915661)

[5. Podręcznik użytkowania aplikacji webowej BoardFlow 46](#_Toc151915662)

[5.1. Instrukcja 46](#_Toc151915663)

[5.2. Wprowadzenie do panelu klienta 49](#_Toc151915664)

[5.3. Wprowadzenie do panelu pracownika 49](#_Toc151915665)

[6. Podsumowanie 50](#_Toc151915666)

[6.1. Wnioski 50](#_Toc151915667)

[6.2. Perspektywy dalszego rozwoju tematyki 50](#_Toc151915668)

[Spis rysunków 51](#_Toc151915669)

[Spis tabel 52](#_Toc151915670)

[Bibliografia 53](#_Toc151915671)

# **Wstęp**

## **Problematyka i zakres pracy**

Prowadzenie działalności gospodarczej od zawsze było wymagającym wyzwaniem. Od czasów powstania pierwszych sklepów przedsiębiorcy starają się dotrzeć do jak najszerszego grona odbiorców. Z biegiem czasu ten cel osiągali coraz to nowszymi środkami masowego przekazu, gazetą, radiem, telewizją i najnowszą powstałą formą – Internetem. Ten ostatni sposób stał się normą, która jest niezbędna do przetrwania, a nawet istnienia współczesnej działalności gospodarczej. Statystyczny konsument stał się wygodniejszy przez ogólną wirtualizację świata, przez co posiadanie internetowej sprzedaży może stać się czynnikiem kluczowym w przypadku wyboru sklepu w którym dokona się zakupu produktu. E-commerce jest obecnie jedną z najbardziej dochodowych gałęzi biznesu, a co za tym idzie, chcąc przetrwać na rynku przedsiębiorcy zmuszeni są do wyboru tej formy handlu [1].

Dodatkowym czynnikiem, które również jest ważny przy prowadzeniu działalności gospodarczej jest skuteczne planowanie pracy. W dzisiejszych czasach, kiedy świat wymaga coraz bardziej niebanalnych pomysłów, trzymanie planu pracy w głowie przestaje być możliwe przez złożoność wytwarzanych produktów. Kluczowe w tym przypadku staję się miejsce w którym możemy trzymać plan naszej pracy, co już zrobiliśmy, a co należy jeszcze zrobić. Można tego dokonać na różne sposoby za pomocą specjalnej tablicy lub zwykłej kartki papieru, jednak formą, która najlepiej się sprawdzi w większości przypadków to dedykowane miejsce do tego typu aktywności. Takim miejscem są wszelkiego typu programy wspomagające zarządzanie projektami, pozwalają one kategoryzować naszą pracę i skutecznie ją zaplanować, a co ważniejsze są dostępne z każdego miejsca, a jednocześnie są szybsze w użyciu niż inne sposoby na zarządzanie projektem.

Zakresem prac będzie analiza procesów towarzyszących obecnie w procesie wytwarzania i zamawiania produktu oraz pozostałych potrzeb konsumenckich i przełożenie tego na wymagania funkcjonalne i niefunkcjonalne oprogramowania. Dodatkowo poddane analizie i porównaniu zostaną obecnie dostępne rozwiązania na rynku wspierające projektowanie i na bazie ich wad i zalet stworzona zostanie nowa aplikacja.

## **Cele pracy**

Celem niniejszej pracy jest analiza wybranych istniejących aplikacji na rynku wspierających proces projektowania poprzez porównanie ich mocnych i słabych stron. Na podstawie przeprowadzonej analizy i sylwetki klienta zostanie utworzona nowa aplikacja webowa, która będzie automatyzować proces wytwarzania i zamawiania gier planszowych poprzez łączenie najlepszych cech i omijanie popełnionych błędów w porównywanych serwisach. Powstałe rozwiązanie końcowo zostanie porównane z wcześniej analizowanymi dostępnymi serwisami na rynku, w celu podsumowania czy wszystkie założenia zostały spełnione.

## **Przegląd literatury**

Nest.js: A Progressive Node.js Framework – oficjalna dokumentacja techniczna framework’a Nest.js [2]. Zasób ten wybrałem ze względu, że jest to jedna z lepiej napisanych dokumentacji na rynku. Opisuję ona działanie całej platformy programistycznej, jak i tego w jaki sposób można używać danych zależności i biblioteki współpracują z danym szkieletem aplikacyjnym.

Dav Vanderkam, TypeScript: Skuteczne programowanie - książka zawierająca porady dotyczące dobrych praktyk i skutecznego posługiwania się językiem programowania TypeScript [3], [4]. Zasób ten wybrałem ze względu na praktyczne przepisy oraz wskazówki, które mogą przynieść korzyści w celu optymalnego wykorzystania potencjału języka TypeScript [4].

Ian Sommerville, Software Engineering Ninth Edition – opis cyklu projektowego w procesie wytwarzania oprogramowania [5]. Zasób ten wybrałem ze względu na lepsze zrozumienie potencjału inżynierii oprogramowania oraz jak wygląda krok po kroku wytwarzanie profesjonalnego oprogramowania od wymagań klienta, aż po fazę konserwacji i utrzymania docelowego programu.

## **Układ pracy**

Praca zbudowana jest z dwóch części, część teoretyczna – rozdziały 1-3, część praktyczna – rozdziały 4-6. Pierwsza część pracy opisuję podstawy napotkanego problemu oraz obecnie dostępne na rynku rozwiązania oraz technologię, które zostaną przeanalizowane. Drugi rozdział opisuje cykl projektowy aplikacji wraz z wszystkimi jego fazami, według metodologii waterfall [6]. W tej części znajduję się również podręcznik użytkowania aplikacji oraz podsumowanie przeprowadzonego cyklu projektowego oraz perspektywy dalszego rozwoju tematyki.

# **Przegląd wybranych aplikacji wspomagających proces zamawiania i projektowanie gier planszowych**

Stworzona aplikacja łączy w sobie cechy systemu do sprzedaży rzeczy jak i do wspomagania procesu wytwarzania projektu. W związku z tym porównywane istniejące już rozwiązania będą pochodzić z tych dwóch dziedzin. Taka perspektywa zostanie użyta w celu wskazania, że wytworzone nowe oprogramowanie będzie łączyć cechy z obydwóch tych zakresów.

## **Aplikacja webowa StoryboardThat**

Obraz zawierający Czcionka, logo, Grafika, tekst

Opis wygenerowany automatycznie

Rysunek 2.1: Logotyp systemu StoryboardThat.

Źródło: [49]

Aplikacja webowa StoryboardThat (rysunek 2.1) jest systemem wspomagającym proces projektowania gier planszowych poprzez możliwość projektowania gry planszowej za pośrednictwem kreatora gier planszowych. Interfejs użytkownika przedstawiony na rysunku 2.2 prezentuję widok aplikacji i jej kluczowe elementy takie jak:

* Obszar roboczy – przestrzeń na której następuje wizualizacja projektu,
* Pasek menu – umożliwiający dodanie różnego typu obiektów gry.

Aplikacja oferuję możliwość budowy gry planszowej z predefiniowanych elementów, które niestety nie są możliwe do edycji, a co za tym idzie użytkownik nie może zmienić ich właściwości. Gotowe projekty gier zamykają się niestety tylko w modelach 2D bez możliwości żadnych rozszerzeń o często kluczowe opisy lub notatki. Aplikacja nie oferuję żadnego API za pośrednictwem, którego można by było spróbować rozszerzyć działanie aplikacji. Nie jest również możliwe prezentowanie statusu projektu, ani dzielenie się projektem w formie do edycji z innymi potencjalnymi pracownikami, raz zapisany szablon nie ulega już zmianie. Przekłada się to tym samym na brak możliwości skutecznego planowania czasu pracy. Potencjalny użytkownik musi wykonać cały projekt naraz, gdzie w przypadku zaistnienia możliwości utworzenia bardziej skomplikowanego projektu trzeba by było zostawić odpalony komputer na długie godziny. Warto również zaznaczyć fakt, że tworzy to tym samym problem z podziałem danego projektu na zadania do wykonania, a co za tym idzie może statystycznie wydłużyć czas pracy nad projektem. Warto również podkreślić fakt, że aplikacja mimo znikomej liczby oferowanych funkcjonalności jest płatna.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Rysunek 2.2: Zdjęcie ekranu z aplikacji webowej StoryboardThat.

Źródło: [53]

## **Aplikacja webowa TableTopia**

Obraz zawierający Grafika, logo, Czcionka, Jaskrawoniebieski

Opis wygenerowany automatycznie

Rysunek 2.3: Logotyp systemu TableTopia.

Źródło: [50]

Aplikacja webowa TableTopia (rysunek 2.3) jest jednym z najbardziej rozbudowanych serwisów jeżeli chodzi o projektowanie gier planszowych. Swoim użytkownikom oferuję szereg udogodnień, jak tworzenie rozbudowanych modeli składających się z wielu elementów w tym modeli 3D zbudowanych z predefiniowanych materiałów. Interfejs użytkownika przedstawiony na rysunku 2.4 prezentuję widok aplikacji i jej kluczowe elementy takie jak:

* Predefiniowane elementy – elementy które mogą posłużyć jako szablony do tworzenia elementów gry,
* Dotychczas stworzone elementy – elementy, które stworzyliśmy dotychczas,
* Kreator nowych elementów – umożliwiający nam tworzenie nowych elementów od zera.

Aplikacja tak jak zostało wspomniane udostępnia nam również możliwość tworzenia własnych nowych obiektów o zadanej grafice, rozmiarze czy też nazwie, jednak nowe elementy muszą być elementami danego typu na przykład karta lub kostka rysunek 2.5. Aplikacja udostępnia możliwość zapisu projektu w formie do edycji co umożliwia w pewnym stopniu dzielenie się pracą jednak tylko w obrębie jednego konta, co za tym idzie potencjalni współpracownicy zmuszeni są do dzielenia się jednym kontem. Warto również podkreślić fakt, że system ten nie ma możliwości stworzenia zadań na utworzenie danego elementu, z czego wynika konieczność korzystania dalej z jakiegoś miejsca do przetrzymywania elementów do stworzenia. Tak samo w momencie przekładania modelu aplikacji na świat rzeczywisty użytkownik nie ma możliwości pilnowania za pośrednictwem systemu, które elementy zostały już utworzone, a które nie, co może prowadzić do problemów z organizacją wytwarzania projektu. Serwis również nie udostępnia możliwości kategoryzacji elementów na te bardziej i mniej priorytetowe, tym samym nie spełniając założeń aplikacji do zarządzania wytwarzaniem projektu w całości. Dodatkowo system nie udostępnia API do rozszerzenia swoich funkcjonalności i jest płatny.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

Rysunek 2.4: Zdjęcie ekranu z aplikacji webowej TableTopia.

Źródło: [54]

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Rysunek 2.5: Zdjęcie z aplikacji webowej TableTopia prezentujące tworzenie nowego obiektu.

Źródło: [54]

## **Aplikacja webowa Olx**

Obraz zawierający Grafika, symbol, projekt graficzny, logo

Opis wygenerowany automatycznie

Rysunek 2.6: Logotyp systemu Olx.

Źródło: [51]

Aplikacja webowa Olx (rysunek 2.5) jest serwisem ogłoszeniowym, który udostępnia swoim użytkownikom możliwość publikacji usług, wystawiania rzeczy na sprzedaż, jak i w drugą stronę, możliwość przeglądania usług i zakupu wystawionych rzeczy. Posiada ona bardzo intuicyjny interfejs użytkownika przedstawiony na rysunku 2.6. Po wybraniu interesującej nas kategorii na ekranie możemy zobaczyć wylistowane kafelki pasujące do naszego wyszukiwania z możliwością stronicowania tuż za ostatnią rzeczą na liście. Serwis udostępnia bardzo przyjemne przefiltrowanie rzeczy po nazwie lub też innych kategoriach, które mogą być przypisane do danej rzeczy. Po wejściu w daną rzecz dostajemy szereg informacji jak opis, cena i więcej zdjęć zadanego produktu. Aplikacja ta jest bardzo prosta w obsłudze, a co za tym idzie potencjalny klient nie zniechęca się po wejściu na stronę do serwisu, bo nie wie jak coś zrobić. Ciekawym aspektem tej aplikacji jest czat ze sprzedającym, który może okazać się pomocny w przypadku dopytania o szczegóły zamówienia. Jednak z perspektywy wykupienia danej usługi, która jest realizowana przez dłuższy okres czasu, to na sprzedającym spoczywa odpowiedzialność wejścia w dany dymek czatu i informowania klienta o statusie. Jest to opcja narażona na potencjalne niebezpieczeństwo przez współczynnik ludzki, co może wywoływać negatywne emocję u klienta, który złożył dane zamówienie, a nie wie co się z nim dzieje. Dodatkowo z perspektywy przedsiębiorcy oferującego swoje usługi lub produkty na platformie nie ma możliwości obszernego zareklamowania się. Jedyną opcją jest zdjęcie profilowe. Tym samym przedsiębiorca nie ma możliwości przedstawienia swojej firmy, czy jakichkolwiek szczegółów z nią związanych.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

Rysunek 2.7: Zdjęcie ekranu z aplikacji webowej Olx.

Źródło: [55]

## **Zalety i wady wybranych aplikacji do zamawiania i wspomagania projektowania gier planszowych**

Po przeanalizowaniu istniejących już rozwiązań można dojść do jednoznacznych wniosków, że na rynku nie ma oprogramowania spełniającego w 100% cechy serwisu sprzedaży, jak i wspomagania procesu projektowania gier planszowych oraz zarządzania wytwarzanym projektem. Wady i zalety istniejących rozwiązań zostały zagregowane w tabeli 2.1. Jak możemy zauważyć wszystkim systemom wspomagającym cały proces projektowania gier planszowych daleko do ideału, a co za tym idzie nie spełniają założenia sprawnego projektowania i zarządzania wytwarzanym projektem. Z kolei serwis oferujący prezentacji swoich usług nie oferuję nic więcej poza ramami sprzedaży danego produktu. Z związku z tym stworzenie oprogramowania łączącego w sobie cechy zamawiania i wspomagania projektowania gier i planszowych jest zasadne i uzupełni brakującą lukę w rynku.

Tabela .: Porównanie funkcjonalności oferowanych przez obecnie dostępne rozwiązania na rynku

Obraz zawierający tekst, zrzut ekranu, linia, Równolegle

Opis wygenerowany automatycznie

# **Stos technologiczny**

## **Uzasadnienie wyboru technologii**

Przez ostatnie paręnaście lat można zauważyć zachodzącą rewolucję w której Internet staje się drugą rzeczywistością handlu zwiększając swoją wartość o miliardy dolarów każdego roku, co przedstawia wykres dostępny rysunku 3.1. Oczywistą odpowiedzią na zachodzące zmiany, jak i prognozy rynkowe z których jasno wynika, że proces ten będzie postępował coraz szybciej jest przenoszenie się każdego przedsiębiorstwa do Internetu. Generuję to tym samym coraz więcej potrzeb „komputeryzacji” sklepów oraz systemów. Wynikiem tego procesu jest ogromny rozwój przeróżnych framework’ów webowych i coraz to nowszych języków programowania.

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Rysunek 3.1: Wykres wartości rynkowej i prognozowanej w handlu internetowym na świecie według portalu ecommerce guide.

Źródło: [7]

Jednym z takich języków programowania, który co roku notuję coraz większy udział na rynku jest język programowania TypeScript, którego to wzrost popularności i ilości pobrań przedstawiony jest na wykresach na rysunkach 3.2 oraz 3.3 [4]. Wynika z tego jasno, że technologia ta zyskuję coraz to większą renomę notując nawet 3 krotny wzrost w ciągu ostatnich czterech lat. Dzieję się tak ze względu na szereg korzyści płynący z używania tej semantyki, między innymi:

* Wprowadzanie typowania przez nadawanie zmiennym określonego typu danych. Pozwala to uniknąć wielu błędów z odwoływania się do nieistniejących instancji obiektów klas,
* Dostęp do dekoratorów, które pozwalają dopisać dodatkową logikę do klas metod parametrów, zwiększając tym samym czytelność kodu [10],
* Wprowadzenie jawnej i prostej enkapsulacji zmiennych i metod klas, co pozwala zachować kontrolę nad udostępnianiem wewnętrznej logiki na zewnątrz obiektu [11].

Obraz zawierający zrzut ekranu, linia, Wielobarwność, diagram

Opis wygenerowany automatycznie

Rysunek 3.2: Wykres reprezentujący rozkład użycia języków programowania w ciągu ostatnich 12 miesięcy.

Źródło: [8]

Obraz zawierający linia, Wykres, diagram, stok

Opis wygenerowany automatycznie

Rysunek 3.3: Wykres reprezentujący ilość pobrań języka programowania TypeScript.

Źródło: [9]

## **Język programowania TypeScript**



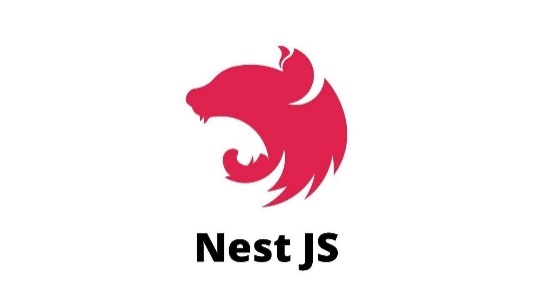
Rysunek 3.4: Logotyp języka programowania TypeScript.

Źródło: [42]

TypeScript (rysunek 3.4) jako obudowa języka programowania JavaScript jest wysokopoziomowym językiem zorientowanym obiektowo umożliwiającym również statyczne typowanie [4], [12]. Został on zaprojekowany przez korporację Microsoft w 2012 roku głównie w celu umożliwienia definiowania przez programistę typów zmiennych [13]. W praktyce oznacza to, że TypeScript rozszerza JavaScript o dodatkowe elementy ułatwiając tym samym tworzenie dużych projektów. Wprowadza on również szereg usprawnień z których część została wymieniona w rozdziale 3.1 oraz takich jak [47]:

* Interfejsy – umożliwiające tworzenie kontraktów między warstwami w aplikacji,
* Klasy wraz z dziedziczeniem – umożliwiające tym samym tworzenie hierarchii obiektów oraz ułatwiając dzielenie się funkcjami i właściwościami,
* Moduły – umożliwiające podział całej aplikacji na niezależne części, ułatwiając tym samym lepsze zarządzanie udostępnianą logiką z danego modułu,
* Typy generyczne – umożliwiające tworzenie uniwersalnych fragmentów kodu, które możliwe są do użycia w różnych miejscach programu.

## **Framework NestJs**



Rysunek 3.5: Logotyp frameworka NestJs.

Źródło: [43]

NestJS (rysunek 3.5) jest platformą programistyczną do budowy aplikacji serwerowych w środowisku uruchomieniowym Node.js [4], [14]. Umożliwia programistą programowanie w czystym języku programowania JavaScript, ale przede wszystkim przy użyciu języka programowania TypeScript [12], [4]. Łączy w sobie zasady programowanie obiektowego poprzez wspieranie klas oraz ich instancji, ale również programowania funkcyjnego poprzez możliwość definiowania metod bez konieczności tworzenia obiektów, jak i również umożliwia programowanie reaktywne poprzez wspieranie operacji asynchronicznych opartych o promise oraz funkcje zwrotne [15], [16]. Zbudowana została przy użyciu TypeScript na bazie wcześniej już wspomnianego Node.js oraz serwera HTTP Express [4], [14], [17]. Platforma zapewnia specyficzną architekturę projektu, wprowadzając skalowalne, luźno połączone, ale przede wszystkim wysoce testowalne moduły aplikacyjne poprzez zastosowanie mechanizmy wstrzykiwania zależności [18]. Zapewnia ona możliwość budowy różnych aplikacji, takich jak monolity, mikroserwisy jak i również aplikację CLI przy użycia API opartego na różnych protokołach, jak i również wsparcie GraphQL [19], [20]. Dzięki takiej architekturze pomimo zmieniającego się sposobu komunikacji między modułami zapewniony jest pewnego rodzaju kontrakt między modułami ułatwiający zrozumienie, jak i wejście w projekt nowemu programiście. Warto również zauważyć, że budowa tej platformy jest mocno inspirowana platformą programistyczną Angular poprzez zastosowanie modułów i wcześniej wspomnianego wstrzykiwania zależności [21], [18].

## **Framework React**

Obraz zawierający Grafika, krąg, sztuka, symbol

Opis wygenerowany automatycznie

Rysunek 3.6: Logotyp frameworka React.

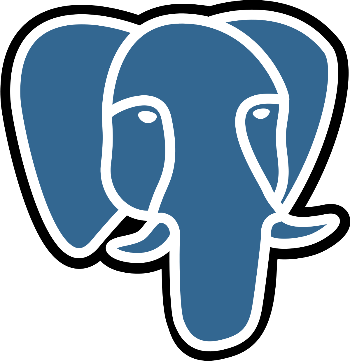
Źródło: [44]

React (rysunek 3.6) jest platformą programistyczną, a nawet bardziej biblioteką języka programowania JavaScript służącą do tworzenia interfejsów graficznych w środowisku uruchomieniowym Node.js [22], [12], [14]. Dysproporcja ta wynika z faktu, iż twór ten nie rozwiązuję problemów strukturalnych i architektonicznych, oferuje za to nowe podejście do tworzenia interfejsów oparte na budowie komponentowej. Nie narzuca on konkretnego stylu programowania dając pełną swobodę, jeżeli chodzi o sposób implementacji projektu. Fenomen tej platformy polega na tworzeniu wielu izolowanych komponentów, które same zarządzają własnym stanem, tworząc wspólnie jednolity i spójny interfejs graficzny. Pojedynczy komponent zbudowany jest z następujących elementów:

* Funkcji komponentu pełniącej reprezentacje komponentu,
* Funkcji stanów, które pozwalają przetrzymać pewien zdefiniowany, zmieniany stan w cyklu życia React,
* Funkcji ubocznej w która pozwala na wykonanie pewnych zdarzeń asynchronicznie poza komponentem na przykład podczas renderowania danego komponentu,
* Zwracanego elementu react, czyli fragmentu, który ma zostać wyrenderowany.

Taka budowa umożliwia pewien sposób izolowania logiki na mniejsze fragmenty pozwalając programiście na bycie zgodnym z jedną z głównych zasad SOLID – „single responsibility” [23]. Warto również zauważyć, że React udostępnia możliwość programowana opartego o klasy, jednak nowoczesne podejście opiera się głównie na funkcyjnym podejściu do komponentów.

## **Baza danych PostgreSQL**

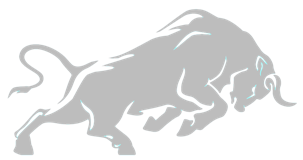


Rysunek 3.7: Logotyp bazy danych PosgreSQL.

Źródło: [45]

PostgreSQL (rysunek 3.7) jest system zarządzania bazą danych oferującą obiektowo-relacyjne podejście [24]. Oznacza to nie wiele więcej niż, połączenie cech relacyjnych baz danych z elementami programowania obiektowego. Takie podejście umożliwia elastyczne modelowanie skomplikowanych struktur danych poprzez możliwość korzystania z zapytań obiektowych ułatwiając tym samym operacje na samych obiektach, jak i relacjach między nimi. Sam system jest oprogramowaniem typu open source oferując swoim użytkownikom wieloplatformowość poprzez dostępność na wszystkich dystrybucjach systemów typu UNIX oraz Windows oraz skalowalność poprzez tabele o rozmiarach nawet do 32 TB [25]. Warto również zaznaczyć, że system ten udostępnia możliwość tworzenia zaawansowanych zapytań poprzez wsparcie dla proceduralnego SQL PL/pgSQL [26], [48].

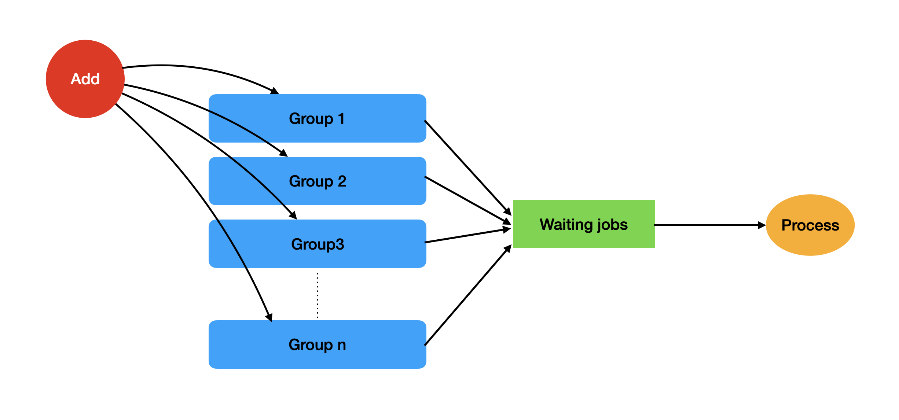
## **System kolejkowania BullMQ**



Rysunek 3.8: Logotyp systemu kolejkowania BullMQ.

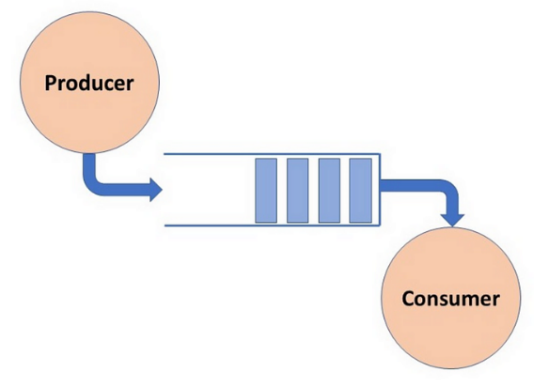
Źródło: [46]

BullMQ (rysunek 3.8) jest biblioteką do obsługi kolejkowania zadań zbudowaną na bazie Redis’a w celu wykorzystania jego przepustowości [27], [28]. Pozwala ona rozwiązać wiele problemów komunikacyjnych między poszczególnymi modułami lub mikroserwisami, odciążając tym samym sam serwer. W praktyce oznacza to, że biblioteka pozwala zakolejkować, czyli odłożyć na stos zadań do zrobienia pewną pracę bez konieczności czekania innego procesu na jej zakończenie. Idealnie sprawdza się w przypadku operacji zakolejkowania wysyłki maila, operacji na plikach lub dostępu do usług w której w jednym czasie może znajdować się tylko jedno zadanie przez ogromną konsumpcję zasobów przez ten proces takich jak generowanie obrazów przez sztuczną inteligencję. Budowa tego systemu przedstawiona na rysunku 3.9 jest dość prosta, a co za tym idzie przyjazna jeżeli chodzi o obsługę lub ewentualne poszukiwanie błędów. Proces czyli Producer dodaje zadanie do utworzonej kolejki lub grupy zadań, następnie zadanie czeka na wejście do Consumer'a rysunek 3.10, gdzie następuję wykonanie i zakończenie danego zadania [29]. Sama kolejka może być skonfigurowana na wiele sposobów LIFO, FIFO, ale przede wszystkim poprzez przyznanie priorytetu danym zadaniom [31], [30].



Rysunek 3.9: Schemat działania systemu kolejkowania BullMQ.

Źródło: [32]



Rysunek 3.10: Schemat wzorca Producer Consumer.

Źródło: [33]

## **Pakiet Mailer**

Pakiet Mailer jest modułem odpowiadającym i umożliwiającym łatwe wysyłania wiadomości typu e-mail z poziomu aplikacji webowej [34]. Wykorzystuje on protokół SMTP „Simple Mail Transfer Protocol” do komunikacji z serwerem poczty, którego działanie możemy zobaczyć na rysunku 6 [35]. Posiada on również pewnego rodzaju elastyczność będąc pakietem wysoce konfigurowalnym obsługując praktycznie wszystkich usługodawców oferujących serwery poczty poprzez konfigurację parametrów zabezpieczeń SSL oraz TLS [38], [37]. Warto również zauważyć, że wysyłanie e-mail jest operacją czasochłonną narażoną na wiele potencjalnych opóźnień przez co moduł zachowuję się w sposób asynchroniczny.

Obraz zawierający tekst, zrzut ekranu, diagram, design

Opis wygenerowany automatycznie

Rysunek 3.11: Schemat działania komunikacji z zastosowaniem protokołu SMTP.

Źródło: [36]

## **Pakiet JWT**

Pakiet JWT jest modułem realizującym funkcjonalności zgodne z realizacją standardu „Json Web Token”, czyli uwierzytelnieniem użytkownika [39]. Powyższa strategia polega na generowania tokenu dostępu zawierającego pewne dane na podstawie których system przyznaję dostęp do zasobów użytkownikowi. Struktura Json web tokena zbudowana jest z trzech części nagłówka, zawartości oraz podpisu. Nagłówek zazwyczaj zawiera informację o rodzaju tokena oraz użytego algorytmu, który posłużył do podpisania. Zawartość zawiera dane, które chcemy zakodować, a które mogą być potrzebne przy rozkodowywaniu tokena w celu identyfikacji kim jest dany użytkownik i czy jego rola jest wystarczająca, żeby uzyskać dostęp do danego zasobu. Podpis powstaje poprzez „podpisanie” nagłówka oraz zakodowanej zawartości poprzez użycie danego algorytmu z pewnym tajemnym ciągiem znaków, który możemy nazwać kluczem. Jego głównym zadaniem jest sprawdzenie, czy token nie został zmodyfikowany po drodze. Warto również zauważyć, że JWT umożliwia funkcjonalność ważności tokenu, czyli nadania mu określonego czasu kiedy token jest akceptowany przez system, a po którego upływie token uznawany jest za przeterminowany.

Działanie tego standardu można przedstawić diagramie sekwencji(rysunek 7):

* Użytkownik podaje poprawne dane login i hasło otrzymując tym samym token dostępu, który jest umieszczany w nagłówku zapytania do którego dostęp jest chroniony, a użytkownik otrzymuję dostęp do zasobu,
* Użytkownik podaje niepoprawny login i hasło, czego skutkiem jest nie przyznanie tokenu,
* Użytkownik posiada token, jednak jego termin ważności minął, czego skutkiem jest brak uzyskania dostępu do zasobu.

Obraz zawierający tekst, zrzut ekranu, paragon, diagram

Opis wygenerowany automatycznie

Rysunek 3.12: Diagram sekwencji dla możliwych scenariuszy strategii JWT.

Źródło: opracowanie własne

## **Pakiet Schedule**

Pakiet Schedule jest pakietem udostępniającym możliwość zaplanowania wykonywania się danego fragmentu kodu w określonym terminie o konkretnej godzinie w sposób cykliczny [40]. Można określić go jako linuksową adaptację pakietów Cron, ale w środowisku Node.js [41], [14]. Pakiet ten pozwala odciążyć użytkownika z pewnych powtarzalnych zadań w systemie takich jak czyszczenie przedawnionych kodów lub wysyłanie cyklicznych emalii. Programista za pomocą tej funkcjonalności może zaplanować wykonywanie się pewnych zadań „w tle” poza świadomością użytkownika. Jego główną zaletą jest to, że dane wydarzenie można zaplanować do wykonania o dowolnej porze, co pozwala na wykonanie kosztownych obliczeń lub przeglądu bazy danych w godzinach kiedy na przykład najmniej użytkowników korzysta z danego systemu.

# **Cykl projektowy aplikacji webowej BoardFlow**

## **Sylwetka klienta i jego wymagania**

W celu lepszego zrozumienia klienta, analizę wymagań należy rozpocząć od określenia sylwetki klienta. Docelowym klientem aplikacji webowej BoardFlow jest przedsiębiorca chcący zautomatyzować dotychczasowy cykl wytwarzania danego zamówienia na grę planszową od złożenia zamówienia po zaprojektowanie schematu gry planszowej aż po jego realizację.

Obecnie przedsiębiorca realizuję ten proces według diagramu czynności przedstawionego na rysunku 4.1. Analizując diagram czynności można zauważyć bardzo niepokojącą rzecz, iż cała odpowiedzialność realizacji projektu spoczywa na przedsiębiorcy i jego dostępności. Problemy pojawiają się już przy próbie złożenia zamówienia, gdyż zysk potencjalnego klienta już od początku zależy od odebrania telefonu. Samo odebranie telefonu nie świadczy już o pewnym potencjalnym zysku, klient jest informowany o cenie dopiero przez przedsiębiorcę na którą to cenę może nie przystać. Jednak w sytuacji akceptacji warunków koniecznych do realizacji zamówienia przedsiębiorca może przejść do przyjęcia zamówienia zapisując tym samym szczegóły i uwagi klienta, co może zająć sporo czasu. Jednakowoż najbardziej czasochłonną czynnością podczas realizacji całego projektu jest ciągła kontrola, czy w procesie realizacji znajdują się wszystkie elementy projektu, co w przypadku skomplikowanych projektów zawierających dziesiątki elementów jest wymagającą czynnością. Po skompletowaniu zamówienia klient odbiera finalny produkt. Należy jednak zaznaczyć, że proces realizacji gry planszowej jest czasochłonny i może zająć nawet do kilku tygodni, a klient w między czasie może chcieć wprowadzać modyfikację lub dopytywać o postępy co również jest dodatkowym rozpraszaczem.

W opozycji do przeprowadzonej analizy przedsiębiorca prowadzący swoją działalność za pomocą aplikacji webowej BoardFlow realizuję zupełnie inny cykl wytwarzania zamówienia przedstawiony na diagramie czynności zamieszczonym na rysunku 4.2. Analizując wymieniony diagram czynności już na wstępnie możemy zauważyć, że część odpowiedzialności spoczywających na przedsiębiorcy została przeniesiona na system informatyczny, co za tym idzie cały cykl realizacji zamówienia jest w mniejszym stopniu na porażkę przez współczynnik ludzki. Klient składając zamówienie już od samego początku jest świadomy szczegółów zamawianej gry, jej ceny oraz ewentualnych modyfikacji, których chciałby dokonać, oszczędza to tym samym czas przedsiębiorcy na przyjęcie zamówienia do jedynie jego potwierdzenia i przyjęcia zaliczki od klienta. System następnie składuję złożone zamówienie, które nad którym prace mogą rozpocząć się w dowolnej chwili. Przedsiębiorca realizując kolejne elementy zna ich dokładne właściwości, jak wymiary kolor jak i ewentualne opisy lub notatki. System cały czas kontroluję status danych elementów, które traktuję jako zadania do wykonania. W systemie istnieją statusy pojedynczych zadań jak i ich priorytety pozwalające osobie realizującej projekt lepiej rozplanować pracę na przykład wykonując element pudełka przed wykonaniem elementu jego zawartości. Osoba realizująca zamówienie może cały czas aktualizować status całego zamówienia w celu informowania klienta o poczynionych pracach, dodatkowo system sam nanosi znacznik czasu ostatniej aktualizacji w przypadku wykrycia zmian na danym projekcie, dzięki czemu klient może w pewnym sensie również po tym wywnioskować jak postępują pracę. Po zrealizowaniu całego projektu i jego skomplementowaniu na przedsiębiorcy spoczywa jedynie obowiązek poinformowania klienta, że dane zamówienie jest już gotowe do odebrania, zaznaczenie tego faktu w systemie i przyjęcie zapłaty. Sam system również udostępnia możliwość realizacji danego zamówienia przez wielu pracowników naraz, jak i przepisywania realizacji danego zamówienia na innego pracownika w przypadku oddania zamówienia. Takie rozwiązania pozwalają dzielić się pracą i tym samym realizować kolejne zadania skuteczniej. System jednak nie odcina się całkowicie od starej drogi składania zamówień dalej umożliwia realizacja projektu bez zamówienia ze zgłoszenia telefonicznego lub czysto hobbystycznego powodu jak projekt do portfolio.

Na podstawię przeprowadzonych powyżej analiz, jak i przeglądu istniejących już aplikacji w rozdziale 2 można wysnuć jednoznaczne wnioski, że docelowe rozwiązanie powinno wspierać funkcjonalności z zakresu zamawiania, projektowania i wspomagania wytwarzania projektów. Oznacza to, że kluczowymi wymaganiami dla klienta będzie dokładne informowanie użytkownika o oferowanych usługach, wspieranie klienta na poziomie składania zamówienia oraz ciągłe wspomaganie wykonawcy w organizacji wykonywanych zadań

Obraz zawierający tekst, zrzut ekranu, dokument, diagram

Opis wygenerowany automatycznie

Rysunek 4.1: Diagram czynności dla przedsiębiorcy nie korzystającego z systemów informatycznych.

Źródło: opracowanie własne

Obraz zawierający tekst, diagram, paragon, Równolegle

Opis wygenerowany automatycznie

Rysunek 4.2: Diagram czynności dla przedsiębiorcy korzystającego z systemów informatycznych.

Źródło: opracowanie własne

## **Wymagania funkcjonalne**

Wymagania funkcjonalne to specyficzne funkcję, zachowania oraz usługi, które tworzony system powinien obsługiwać. Definiują one pewne założenia, które powinny zostać spełnione jako reakcja na pewne czynności lub zdarzenia. Określenie wymagań funkcjonalnych pomaga twórcą systemów spełnić oczekiwania, które dane rozwiązanie ma spełniać.

W przypadku aplikacji webowej BoardFlow wymagania funkcjonalne wraz z opisem również zostały zdefiniowane w tym samym celu:

**Wyświetlanie i możliwość przechowania dokładnych informacji o danej grze planszowej**

System powinien zapewnić możliwość dokładnego opisania danej gry, jej ceny jak i zdefiniowania każdego pojedynczego elementu w zadanej ilości egzemplarzy wraz ze zdjęciem każdego elementu. Pozwoli to stworzyć jednolite i spójne miejsce do przechowywania wyżej wymienionych informacji z którego będzie mógł korzystać klient w celu pozyskania informacji o potencjalnym produkcie, jak i przedsiębiorca tworząc projekt dla zadanej z góry liczby elementów.

**Tworzenie, modyfikacja i przechowywanie wielu szablonów dla jednej gry**

System przechowując wiele szablonów dla jednej gry będzie w stanie zwiększyć ich reużywalność. Przełoży się to tym samym na to, że raz utworzony specyficzny zmodyfikowany szablon dla danej gry będzie mógł być użyty w przyszłości do innej gry lub powtórzonych takich samych wymagań klienta.

**Wprowadzanie nowych i modyfikacja istniejących gier planszowych**

System powinien umożliwiać ciągłe dodawanie nowych gier w ciągłego rozszerzania oferty danego przedsiębiorstwa. Modyfikacja dodanych elementów również jest kluczowa w przypadku pomyłki lub potrzeby modyfikacji danej gry o dodatkowe elementy.

**Tworzenie nowych kategorii**

Tworzenie nowych kategorii może okazać się kluczowe w przypadku poszukiwania przez klienta gry z zadanej tematyki, ale bez konkretnego tytułu. Dobre przyporządkowanie kategorii może pomóc w tym procesie.

**Priorytetyzacja i kategoryzacja elementów gry jako zadania**

Elementy w systemie będą traktowane jako pojedyncze zadania do wykonania. Pozwoli to tym samym na lepszą organizację pracy poprzez utworzenie bardziej priorytetowych elementów na początku pracy z danym projektem. Z kolei kategoryzacja elementów pozwoli sprecyzować na jakim etapie produkcji jest zadany element. Pozwoli to tym samym na lepszą kontrolę w jakim stanie znajduję się nasz wytwarzany projekt.

**Składanie zamówień poprzez stronę**

Aplikacja powinna oferować możliwość złożenia zamówienia na daną grę wraz z podaniem kluczowych informacji potrzebnych do kontaktu z potencjalnym zalogowanym klientem. Dodatkowym atutem będzie również pole komentarzy i ewentualnych modyfikacji, które pozwolą dostosować grę pod konkretne wymagania. Tylko zalogowany użytkownik powinien móc złożyć zamówienie.

**Wyświetlanie informacji na stronie**

Strona internetowa systemu powinna być miejscem, które będzie zachęcać potencjalnych klientów do zakupu oferowanych produktów. Zawarcie różnego typu bestsellerów danego miesiąca, nowych hitów lub chociażby opisu i misji, która kryję się za zasłonami prowadzenia danej działalności może skłonić klienta do skorzystanie z usług. Dodatkowym atutem będzie możliwość dynamicznej modyfikacji tych informacji z poziomu administratora.

**Panel administratora i pracownika**

System powinien obsługiwać trzy poziomy dostępu klient, administrator oraz pracownik. Klient ma możliwość przeglądania gier, złożonych zamówień oraz jawnym informacji na stronie. Pracownik obsługiwać wprowadzanie nowych produktów do systemu, tworzyć nowe projekty gier, zarządzać użytkownikami. Administrator rozszerzać uprawnienia pracownika o zarządzanie również pracownikami i generalnymi informacji o stronie.

**Logowanie użytkownika**

Funkcjonalność logowania zapewnia użytkownikowi weryfikację jego tożsamości i dostęp do autoryzowanych zasobów zależnych od roli użytkownikami.

**Rejestracja użytkownika**

Aplikacja powinna zapewniać możliwość utworzenia nowego konta, które w momencie zatwierdzenia będzie posiadać role zwykłego klienta.

**Weryfikacja konta za pomocą adresu email**

W celu ograniczenia ilości fałszywych kont przed możliwością zalogowania do systemu użytkownik musi zweryfikować swoje konto poprzez aktywowanie swojego konta kodem, który otrzyma na adres email utworzonego konta. Dopiero po aktywacji konta powinien móc się zalogować i korzystać z usług serwisu.

**Dodawanie właściwości określających elementy gry**

System powinien umożliwiać dodanie dla zadanego elementu różnego typu opisów, notatek, kolorów oraz wymiarów. Obowiązkowym elementem również są zdjęcia danego elementu, które ułatwiają jego wierne odzwierciedlenie.

**Zmiana języka na stronie głównej**

Strona główna chociaż w pewnej mierzę powinna udostępniać możliwość wyboru języka w której możemy ją obsługiwać. Umożliwia to osobą, które nie są wielojęzykowe również z niej korzystanie.

## **Wymagania niefunkcjonalne**

Wymagania niefunkcjonalne to specyficzne cechy danego systemu, którego jednak nie są powiązane z jego funkcjonalnościami, jednak wpływają na jego odbiór, użyteczność lub też wydajność.

W przypadku aplikacji webowej BoardFlow wymagania niefunkcjonalne wraz z opisem również zostały zdefiniowane:

**Architektura modularnego monolitu**

Aplikacja została zbudowana na wzór architektury modularnej wydzielającym tym samym luźno powiązane modułu, które nie są ze sobą ściśle powiązane. Taka budowa umożliwia podział logiki na osobne komponenty, które dotyczą tylko zadanego fragmentu ułatwiając tym samym łatwą edycję lub też wymianę danej logiki na inną, więcej w rozdziale 4.4.

**Łatwość użycia, intuicyjny interfejs**

Z perspektywy klienta jak najszybsze uzyskanie informacji o produkcie, a następnie złożenie na nie zamówienia są czynnikami które często przeważają w skorzystaniu z usług danego przedsiębiorstwa lub też nie.

**Bezpieczeństwo**

Aplikacja powinna zapewniać bezpieczeństwo danych użytkowników oraz zarządzać dostępem do nich, weryfikując dostęp do różnych funkcji systemu.

## **Architektura aplikacji**

Aplikacja BoardFlow jako cały system posiada architekturę trójwarstwową lub inaczej mówiąc klient-serwer. Oznacza to, że system zbudowany jest z warstwy prezentacji, warstwy logiki biznesowej oraz warstwy danych rysunek 4.3.

* Warstwa prezentacji (frontend) zawiera elementy interfejsu użytkownika, tym samym odpowiadając za interakcję z nim,
* Warstwa logiki biznesowej (backend) zawiera serwer aplikacji obsługujący żądania klienta przetwarzając tym samym dane. Odpowiada za szeroko pojęte zachowanie, logikę w udostępnianych oraz modyfikowanych danych,
* Warstwa danych (database) zawiera przechowywane dane oraz poprzez różne mechanizmy bazo-danowe zarządza nimi.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rysunek 4.3: Diagram komponentów dla architektury trójwarstwowej.

Źródło: opracowanie własne

### **Warstwa prezentacji**

Warstwa prezentacji aplikacji BoardFlow został zbudowana według architektury komponentowej. Oznacza to, że poszczególne widoki są oparte na komponentach, które zawierają swój własny stan i logikę. Taka architektura pozwala w łatwy sposób dodawać oraz modyfikować już istniejące komponenty. Dodatkowo jest to naturalny wybór w przypadku użycia framework’a React, którego budowa również oparta jest na komponentach [22]. Komponenty dla warstwy prezentacji aplikacji BoardFlow zostały zaprezentowane na diagramie klas widocznym na rysunku 4.7.

* Komponent Workspace – odpowiada za obsługę przestrzeni roboczej dla pracownika przedsiębiorstwa, czyli modyfikację aktualnie wykonywanego projektu oraz zamówienia, które może być przypisane do projektu, przez zmianę jego statusu i informacji o nim,
* Komponent Orders – odpowiada za logikę zarządzania zamówieniami oraz przypisywania ich do poszczególnych pracowników w tym ich edycję,
* Komponent ManageUsers – odpowiada za logikę zarządzania użytkownikami w tym ich dezaktywacje, modyfikację oraz dodawanie,
* Komponent ManageTags – odpowiada za logikę zarządzania tagami, czyli kategoriami, które można przypisywać do gier, poszerzając tym samym możliwość lepszej kategoryzacji gier,
* Komponent ManageProjects – odpowiada za logikę zarządzania projektami, czyli szablonami według których powstają gry planszowe. Umożliwia dodawanie nowych oraz modyfikację istniejących szablonów. Dodatkowo umożliwia zarządzanie obecnie trwającymi projektami, czyli tymi które obecnie są wykonywane przez pracowników,
* Komponent ManageGames – odpowiada za logikę zarządzania grami, czyli dostępnymi produktami w systemie, w tym dodawanie nowych, modyfikację oraz usuwanie produktów w systemie,
* Komponent ManageEmployees – odpowiada za logikę zarządzania użytkownikami w systemie, w tym zmianę ich roli, modyfikację danych, dezaktywację kont,
* Komponent CustomOptions – odpowiada za ustawianie różnego typu opisów widocznych w warstwie prezentacji aplikacji BoardFlow,
* Komponent Board – odpowiada za logikę zarządzania elementami w systemie traktowanymi jako zadania do zrobienia poprzez zarządzanie ich priorytetem oraz kategorią,
* Komponent About – odpowiada za prezentację tekstów, które zostały ustawione do wyświetlenia,
* Komponent Contact – odpowiada za prezentację danych kontaktowych, które zostały ustawione do wyświetlenia,
* Komponent Footer – odpowiada za wyświetlenie szybkich przekierowań do poszczególnych podstron aplikacji,
* Komponent Games – odpowiada za prezentację oraz filtrowanie produktów dostępnych w systemie,
* Komponent Home – odpowiada za stronę główną warstwy prezentacji aplikacji tym samym za wprowadzenie nowego użytkownika w możliwości oferowane przez serwis,
* Komponent Login – odpowiada za logowanie użytkownika do systemu, czyli obsługę formularza logowania, tym samym modyfikując stan aplikacji wyświetlając dodatkowe komponenty w zależności od przyznanych uprawnień,
* Komponent NavBar – odpowiada za nawigację w aplikacji, czyli intuicyjne przechodzenie do odpowiednich widoków przez użytkownika,
* Komponent Personal – odpowiada za logikę wyświetlenia informacji o obecnie zalogowanym użytkowniku w systemie i ewentualną modyfikację przez zalogowanego użytkownika swoich danych,
* Komponent Register – odpowiada za logikę formularza rejestracyjnego dla nowego użytkownika w systemie oraz aktywację konta przez podanie kodu aktywacyjnego.

Więcej informacji o poszczególnych widokach i ich funkcjonalnościach w rozdziale 5.

### **Warstwa logiki biznesowej**

Warstwa logiki biznesowej aplikacji BoardFlow została zbudowana według architektury modularnego monolitu rysunek 4.4. Oznacza to, że projekt jest pojedynczym serwisem z wewnętrznym podziałem na pojedyncze niezależne moduły, których logika jest odseparowana. Zastosowanie takiego podejścia pozwala wyodrębniać pełnione role do poszczególnych modułów na zasadzie jeden moduł, jedna pełniona odpowiedzialność. Każdy moduł aplikacji ma swoją własną logikę biznesową, która modyfikuję stan aplikacji, tym samym zapewniając obsługę danego typu zdarzeń w obrębie jednego modułu. Podejście to pozwala na dość szybkie i łatwe dodawanie nowych modułów oraz podmiankę tych już używanych przez ustanowione interfejsy, które są udostępniane przez poszczególne moduły. Architektura modularnego monolitu jest krokiem wprzód względem klasycznego monolitu, w którym to cała aplikacja jest traktowana jako jeden moduł. Warto również zauważyć, że modularny monolit w pewnym sensie jest inspirowany architekturą mikroserwisów. Każdy serwis, moduł ma swoją pojedynczą odpowiedzialność będąc niezależnym od pozostałych, a jedynie wymieniając informację w razie potrzeby w obrębie części składowych.

Przechodząc jednak do architektury pojedynczego modułu oparta jest ona na modelu MVC „Model-View-Controller”, która przedstawia podział aplikacji na kontrolery, logikę oraz repozytoria rysunek 4.5 [56].

* Kontroler – odpowiada za obsługę operacji związanych z obsługą przychodzących żądań oraz routowaniem ich do odpowiednich metod logiki biznesowej,
* Logika – odpowiada za przeprowadzanie różnego typu operacji i zarządzanie modyfikację danych,
* Repozytoria – odpowiada za dostęp do danych warstw wyższych poprzez dostarczanie możliwości przeprowadzania modyfikacji danych w bazie danych .

Obraz zawierający tekst, zrzut ekranu, diagram, Równolegle

Opis wygenerowany automatycznie

Rysunek 4.4: Diagram komponentów prezentujący architekturę modularną w kontekście aplikacji BoardFlow. Źródło: opracowanie własne

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rysunek 4.5: Diagram komponentów dla modelu MVC.

Źródło: opracowanie własne

### **Warstwa danych**

Warstwa danych aplikacji BoardFlow w architekturze trójwarstwowej odpowiada za przechowywanie, dostęp oraz zarządzanie danymi używanymi przez aplikację. W kontekście aplikacji BoardFlow wykorzystana została relacyjna baza danych w celu utworzenia odpowiednich zależności między encjami w aplikacji. Takie podejście jest możliwe dzięki użyciu bazy danych PostgreSQL wraz z biblioteką ORM „Object Relational Mapping” wykorzystaną do wykonywania operacji na wspomnianej bazie [57]. Takie podejście umożliwia odzwierciedlenie struktury bazy danych za pomocą encji, czyli obiektów już w samych warstwach wyższych. Eliminuję to tym samym konieczność korzystania z surowych zapytań języka SQL w aplikacji zastępując je interfejsem, umożliwiającym wykonywanie operacji na bazie. Diagram klas przedstawiający relację dla poszczególnych obiektów został zaprezentowany na rysunku 4.6. Analizując układ tych obiektów i zależności między nimi można wyróżnić 3 główne encję projekt(„Project”), gra(„Game”) oraz użytkownik(„User”).

Projekt reprezentuje szablon w oparciu o który wytwarzany jest dany produkt. W jego skład wchodzi kilka innych obiektów będących jego składowymi takich jak:

* Pudełko („Box”) – reprezentujący pudełko w którym znajduję się gra, relacja między projektem, a grą to 1:1, ponieważ każda gra musi mieć swoje opakowanie, a dane opakowanie może być elementem tylko jednej gry,
* Element („Element”) – reprezentuję element danej gry do zrobienia, czyli niezbędną rzecz z elementów do wykonania na przykład kostka, karta itp. Relacja między nimi to 1 do wielu, ponieważ dany projekt może mieć wiele elementów, ale dany element musi należeć tylko do jednego projektu,
* Kontener („Container”) – reprezentuję pojemnik, który jest potrzebny do przetrzymywania danych elementów w grze na przykład pojemnik na kostki do gry, karty itp. Relacja między pojemnikiem a projektem to 1 do wielu, ponieważ dany projekt może mieć wiele pojemników, ale dany pojemnik musi należeć tylko do jednego projektu,

Gra reprezentuję produkt gry planszowej, który jest dostępny do zakupu jako finalny produkt procesu projektowania danej gry. W jego skład wchodzą elementy takie jak:

* Tagi („Tag”) – reprezentujące kategorię do której dana gra przynależy. Ułatwia to wyszukiwanie danej gry po kategoriach do których gra przynależy. Relacja między grą a tagiem, to wiele do wielu, ponieważ dana kategoria może należeć do wielu gier, a dana gra może mieć wiele kategorii.
* Komponenty („Komponents”) – reprezentujące składowe, które wchodzą w elementy gry planszowej. Pozwalają one użytkownikowi zobaczyć z jakich elementów składa się dana gra.

Użytkownik („User”) – reprezentujący zarówno użytkowników, jak i pracowników systemu na podstawie roli do której przynależą.

Wszystkie przeanalizowane fragmenty całej aplikacji można przedstawić na diagram pakietów przedstawiającym wizualizację i rozmieszczenie elementów w systemie rysunek 4.8.

Obraz zawierający tekst, diagram, Plan, wykres

Opis wygenerowany automatycznie

Rysunek 4.6: Diagram klas dla aplikacji BoardFlow.

Źródło: opracowanie własne

Obraz zawierający tekst, diagram, zrzut ekranu, linia

Opis wygenerowany automatycznie

Rysunek 4.7: Diagram klas przedstawiający komponenty aplikacji BoardFlow.

Źródło: opracowanie własne

Obraz zawierający tekst, diagram, mapa

Opis wygenerowany automatycznie

Rysunek 4.8: Diagram pakietów dla rozmieszczenia elementów dla całego systemu.

Źródło: opracowanie własne

## **Implementacja – punkty kluczowe**

Kluczowymi elementami w przedstawionym rozwiązaniu na pewno mogą być wspomniane w rozdziale 3.2. dekoratory, strategia JWT, system wprowadzonych strażników(„Guards”) oraz sama modularna struktura projektu [10], [4], [39], [58]. Elementy wraz z opisem wchodzące w kluczowe punkty implementacji rozwiązania:

### **Modularna budowa projektu**

Architektura projektu, tak jak już zostało opisane w rozdziale 4.4. to modularny monolit. W celu realizacji tego wzorca zastosowane zostały moduły języka programowania TypeScript, które podzieliły aplikację na 9 osobnych modułów rysunek 4.18 [4]. Każdy z nich ma swoją własną logikę i budowę spójną z modelem MVC [56]. Odpowiedzialności poszczególnych modułów:

* Moduł auth – (rysunek 4.9) moduł odpowiedzialny za obsługę funkcjonalności związanych z uwierzytelnieniem żądania i późniejszym autoryzowaniem go, przyznając mu dostęp do zasobów. Zastosowano w nim strategie JWT wraz z uwierzytelnieniem opartym na rolach w celu zabezpieczenia zasobów. Wykorzystuję w tym celu zależności do modułów użytkowników, bazy danych i wyjątków,

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, design

Opis wygenerowany automatycznie

Rysunek 4.9: Budowa modułu auth.

Źródło: opracowanie własne

* Moduł database – (rysunek 4.10) moduł odpowiedzialny za konfigurację połączenia z bazą danych. Zastosowano w nim bibliotekę TypeORM do realizacji podejścia „Object Relational Mapping” umożliwiającego mapowanie obiektów na rekordy bazy danych [59], [57],

Obraz zawierający tekst, Czcionka, zrzut ekranu, Jaskrawoniebieski

Opis wygenerowany automatycznie

Rysunek 4.10: Budowa modułu database.

Źródło: opracowanie własne

* Moduł exceptions – (rysunek 4.11) moduł odpowiedzialny za obsługę wyjątków w całej aplikacji pozwala on w łatwy sposób zwracać różnego typu błędy w momencie wystąpienia w jakikolwiek innym module. Mapuję on dany błąd na wiadomość i przypisuję mu odpowiedni kod błędu w zależności, czego dany wyjątek dotyczył,

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rysunek 4.11: Budowa modułu exceptions.

Źródło: opracowanie własne

* Moduł game – (rysunek 4.12) moduł odpowiedzialny za obsługę funkcjonalności związanych z logiką dotyczącą zarzadzania dostępnymi grami planszowymi w systemie. Wykorzystuję w tym celu zależności do modułów autoryzacji, bazy danych i wyjątków,

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Rysunek 4.12: Budowa modułu game.

Źródło: opracowanie własne

* Moduł image – (rysunek 4.13) moduł odpowiedzialny za obsługę przechowywania plików obrazów w aplikacji. Wykorzystuje on predefiniowane komponenty obsługi przesyłu i odbioru plików w framework’u NestJs do realizacji funkcjonalności związanych z przesułem plików w żadaniach [60], [2]. Wykorzystuję w tym celu zależności do modułów autoryzacji, bazy danych i wyjątków

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rysunek 4.13: Budowa modułu image.

Źródło: opracowanie własne

* Moduł information – (rysunek 4.14) moduł odpowiedzialny za ustawianie aktualnie wyświetlanych informacji na stronie, takich jak opisy, cele firmy ale przede wszystkim dane kontaktowe do przedsiębiorstwa. Wykorzystuję w tym celu zależności do modułów autoryzacji, bazy danych i wyjątków,

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rysunek 4.14: Budowa modułu information.

Źródło: opracowanie własne

* Moduł order – (rysunek 4.15) moduł odpowiedzialny za realizację funkcjonalności związanych z obsługą składania oraz zarządzania zamówieniami. Wykorzystuję w tym celu zależności do modułów autoryzacji, bazy danych i wyjątków,

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Rysunek 4.15: Budowa modułu order.

Źródło: opracowanie własne

* Moduł project – moduł odpowiedzialny za realizację funkcjonalności związanych z obsługą zarządzania projektami. Jest to największy i jednocześnie najważniejszy moduł realizujący przeprowadzanie cyklu projektowego poprzez zarządzanie stanami zadań. Wykorzystuję do tego celu szereg kontrolerów w celu podziału warstwy domenowej na mniejsze klasy, tak samo dzieląc warstwę logiki biznesowej na mniejsze części w celu lepszej jej separacji. Wykorzystuję w tym celu zależności do modułów autoryzacji, bazy danych, wyjątków, gier oraz użytkowników,

Obraz zawierający zrzut ekranu, tekst, oprogramowanie

Opis wygenerowany automatycznie

Rysunek 4.16: Budowa modułu project.

Źródło: opracowanie własne

* Moduł users – moduł odpowiedzialny za realizację funkcjonalności związanych z obsługą użytkowników systemu. Realizuję również obsługę realizacji kodów aktywacyjnych za pośrednictwem serwera SMTP [35]. W tym module również swoje zastosowanie znalazł system kolejkowania opisany w rozdziale 3.6 realizujący wzorzec producer consumer [33]. Wykorzystuję w tym celu zależności do modułów autoryzacji, bazy danych i wyjątków,

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

Rysunek 4.17: Budowa modułu users.

Źródło: opracowanie własne

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

Rysunek 4.18: Przedstawienie modularnej budowy projektu.

Źródło: opracowanie własne

### **Implementacja personalnych dekoratorów**

W projekcie warstwy logiki biznesowej zostały również zastosowane własne dekoratory języka programowania TypeScript. Są one funkcjami pozwalającymi modyfikować właściwości metod w czasie kompilacji [10], [4]. Użyte własne dekoratory w projekcie:

Dekorator „HasRole”, którego implementacja została przedstawiona na listingu na rysunku 4.19. Przypisuje on metadane do metod przy użyciu klucza ROLES. Używany jest jako składowa implementacji „RolesGuard” guard, listing na rysunku 4.23. Wszystkie wspomniane elementy implementują funkcjonalność kontroli dostępu opartej na rolach, która to polega na sprawdzeniu, czy dane żądanie przychodzące do aplikacji ma wystarczające uprawnienia.



Rysunek 4.19: Listing kodu źródłowego implementacji dekoratora HasRole.

Źródło: opracowanie własne

Dekorator „GetCurrentUser”, którego implementacja została przedstawiona na listingu widocznym na rysunku 4.20. Służy on do wyciągnięcia z kontekstu, czyli żądania personaliów użytkownika, które mogą być użyte na przykład przy automatycznym przypisaniu zamówienia do właściciela żądania.



Rysunek 4.20: Listing kodu źródłowego implementacji dekoratora GetCurrentUser.

Źródło: opracowanie własne

### **Implementacja personalnych guards**

W projekcie warstwy logiki biznesowej występuje również koncepcja specjalnych serwisów pozwalających regulować kontrolę dostępu do określonych fragmentów aplikacji. Pozwalają one na określenie, czy przychodzące żądanie powinno posiadać dostęp do określonych fragmentów aplikacji. Użyte własne guard’sy:

* Guard „OrderMemberShip”, którego implementacja została przedstawiona na listingu na rysunku 4.21. Został on zastosowany w celu umożliwienia edycji złożonego już zamówienia przez użytkownika będącego tylko i wyłączenie jego właścicielem. Takie podejście było koniecznie ze względu na konieczność weryfikacji tożsamości przed umożliwieniem aktualizacji danych diagram sekwencji przedstawiony na rysunku 4.22.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Rysunek 4.21: Listing kodu źródłowego implementacji OrderMemberShip guard.

Źródło: opracowanie własne

Obraz zawierający tekst, zrzut ekranu, Równolegle, numer

Opis wygenerowany automatycznie

Rysunek 4.22: Diagram sekwencji dla sprawdzenia czy użytkownik jest właścicielem zasobu.

Źródło: opracowanie własne

* Guard „RolesGuard”, którego implementacja została przedstawiona na listingu widocznym na rysunku 4.23. Używa on warstwy logiki z modułu user w celu pobierania najbardziej aktualnego stanu użytkownika w systemie. Jego odpowiedzialność polega na sprawdzeniu czy wysłane żądanie powinno mieć dostęp do danego zasobu. Polega to na niczym innym jak sprawdzeniu, czy użytkownik posiada odpowiednią role w systemie, która zezwala na dostęp do danego fragmentu w aplikacji. Zastosowanie „RolesGuard” wraz z dekoratorem „HasRoles” realizuję funkcjonalność kontroli dostępu opartej na roli. Dekorator ustawia metadane danego zasobu na role niezbędne w procesie uzyskiwania dostępu do danego fragmentu aplikacji. Następnie guard pobiera metadane z używanego zasobu i weryfikuję, czy żądanie, które wysyła prośbę o uzyskanie dostępu powinno mieć do niego dostęp. Dla potrzeb przedstawienia interakcji między systemem, a zasobem wysyłającym żądanie został stworzony diagram sekwencji zaprezentowany na rysunku 4.24. Przedstawia on kolejność wywołań operacji w przypadku próby uzyskania dostępu do zasobu, który jest chroniony w wspomniany wyżej sposób. Posiada on trzy możliwe zakończenia swojej sekwencji:
  + Uzyskanie dostępu do zasobu – żądanie miało ważny token i odpowiednie permisje by uzyskać dostęp,
  + Brak uzyskania dostępu do zasobu – wysłane żądanie posiadało token, który stracił swoją ważność,
  + Brak uzyskania dostępu, próba nieautoryzowanego dostępu do zasobu – wysłane żądanie posiada ważny token, lecz nie posiada wystarczających pozwoleń by uzyskać dostęp do zasobu.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Rysunek 4.23: Listing kodu źródłowego implementacji RolesGuard.

Źródło: opracowanie własne

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rysunek 4.24: Diagram sekwencji dla uwierzytelnienia opartego na rolach.

Źródło: opracowanie własne

* Guard „AuthorGuard”, którego implementacja została przedstawiona na listingu zaprezentowanym na rysunku 4.25. Posiada on zależności do warstw logiki z modułów user i project. Realizuję tym samym funkcjonalność sprawdzenia, czy dane żądanie powinno uzyskać dostęp do modyfikacji projektu diagram sekwencji rysunek 4.26. Sprawdza tym samym trzy warunki:
  + Czy edytowany projekt nie jest szablonem, który nie powinien podlegać edycji,
  + Czy edytowany projekt nie został już zakończony, zakończone projekty również nie powinny być już edytowane,
  + Czy wysłane żądanie jest administratorem systemu lub właścicielem żądanego zasobu.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Rysunek 4.25: Listing kodu źródłowego implementacji AuthorGuard.

Źródło: opracowanie własne

Obraz zawierający tekst, zrzut ekranu, Równolegle, numer

Opis wygenerowany automatycznie

Rysunek 4.26: Diagram sekwencji dla sprawdzenia czy użytkownik może modyfikować projekt.

Źródło: opracowanie własne

### **Realizacja kolejkowania**

W projekcie występuję również kolejkowanie procesów przy użyciu biblioteki BullMQ. Implementacja tego fragmentu kodu została przedstawiona na listingach zaprezentowanym na rysunkach 4.27 oraz 4.28. Można zauważyć na nich użycie wzorca projektowego Producer-Consumer, w którym „UserService” pełni role producera, a „CodeActivatorConsumer” role consumera. Podczas rejestracji użytkownika zadanie do wykonania zostaje wrzucone do kolejki o nazwie „CODE\_SEND\_EMAIL”, które następnie trafia do consumera kolejki o tej samej nazwie w której następuje wykonanie zadania, diagram sekwencji na rysunku 4.29.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Rysunek 4.27: Listing kodu źródłowego implementacji kolejkowania, producer.

Źródło: opracowanie własne

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, System operacyjny

Opis wygenerowany automatycznie

Rysunek 4.28: Listing kodu źródłowego implementacji kolejkowania, consumer.

Źródło: opracowanie własne

Obraz zawierający tekst, zrzut ekranu, Równolegle, diagram

Opis wygenerowany automatycznie

Rysunek 4.29: Diagram sekwencji dla rejestracji użytkownika.

Źródło: opracowanie własne

### **Realizacja JWT**

W projekcie autoryzacja odbywa się poprzez zastosowanie strategii JSON web Token’a [39]. Realizacja tego fragmentu implementacji została przedstawiona na listingach zaprezentowanych na 4.30 oraz 4.31. Polega ona na ekstrakcji Bearer tokenu z nagłówka autoryzacyjnego danego zapytania oraz jego rozkodowaniu i sprawdzeniu czy token jest poprawny i nie został zmodyfikowany [62]. W przypadku użycia frameworka NestJs praktycznie całość kodu jest realizowana przy użyciu pakietów passport-jwt oraz passport [2], [61]. Dzięki temu realizacja nawet tak skomplikowanego mechanizmu sprowadza się do użycia predefiniowanych komponentów oraz wspomnianych wcześniej guards’ów, w tym przypadku „JwtGuard” [58]. Warto jednak zaznaczyć, że wspomniane pakiety umożliwiają wstrzyknięcie różnego typu strategii autoryzacji użytkowników do danych zasobów, dlatego do implementacji ich funkcjonalności zastosowano wzorzec projektowy strategii.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

Rysunek 4.30: Listing kodu źródłowego implementacji strategii JWT.

Źródło: opracowanie własne



Rysunek 4.31: Listing kodu źródłowego implementacji JWT guard.

Źródło: opracowanie własne

## **Testy**

## **Instalacje i konserwacja**

W celu zestawienia projektu na swoim lokalnym środowisku należy posiadać następujące poprawnie zainstalowane oprogramowania wraz z daną wersją od której rozwiązani na pewno jest wspierane:

* Docker compose, wersja >= 2.15.1 [64],
* Docker, wersja >= 20.10.22 [63].

W celu weryfikacji poprawności i wersji wymienionego wyżej oprogramowania należy wykonać następujące komendy na terminalu „docker --version” oraz „docker-compose --version”. Ze względu na proces instalacji pakietów używanych w projekcie niezbędny będzie również dostęp do internetu, bez którego nie ma możliwości przeprowadzenia procesu poprawnej instalacji aplikacji w środowisku.

Pierwszym krokiem na drodze instalacji projektu jest ściągnięcie kodu źródłowego aplikacji do wybranej lokalizacji na swoim lokalnym środowisku. Należy zwrócić uwagę jednak, że przez zastosowanie architektury klient-serwer, warstwa widoku oraz warstwa logiki biznesowej będą korzystać z dwóch odrębnych instalacji niezbędnych modułów, a które to modułu zajmują dość sporo przestrzeni dyskowej przez co ich kontenery będą dość ciężkie.

Następnym krokiem będzie konfiguracja zmiennych środowiskowych, których przykładowe wartości znajdują się w pliku „.env” w głównym katalogu projektu. Ustawione wartości są nieedytowalne podczas działania aplikacji, więc należy je ustawić przed uruchomieniem aplikacji, więcej w rozdziale 5.1. Plik konfiguracyjny jest wstrzykiwany do kontenerów w momencie ich tworzenia.

Ostatnim krokiem będzie otwarcie konsoli i przejście do lokalizacji w której znajduję się kod źródłowy aplikacji. Wywołanie komendy „docker-compose up” spowoduję uruchomienie pliku konfiguracyjnego używanego przez Docker Compose przedstawionego na rysunku 5.2. w tym „utworzenie się 4 kontenerów:

* „GameFlow-Postgres” – kontener zawierający relacyjną bazę PostgreSQL [24],
* „GameFlow-Redis” – kontener zawierający specjalną bazę danych Redis [28],
* „GameFlow-Backend” – kontener zawierający warstwę logiki biznesowej,
* „GameFlow-Frontend” – kontener zawierający warstwę prezentacji.

Wymienione wyżej kontenery będą posiadać oprogramowanie w następującej wersji od której na pewno instalacja będzie działać:

* Baza danych PostgreSql, wersja >= 15.0 [24]
* Baza danych Redis, wersja >= 7.0.11 [28]
* Środowisko uruchomieniowe Node.js, wersja >= 18.16.1 [14]
* Menadżer pakietów Npm, wersja >= 9.8.0 [65]

Naturalną koleją rzeczy są również ciągłe wychodzące aktualizację pakietów używanych w aplikacji. Co za tym idzie chcąc wciąż utrzymać projekt w jak najlepszym stanie potrzebna jest skuteczna konserwacja oprogramowania. W tym kontekście polegałaby ona na zwyczajnej aktualizacji używanych bibliotek. Takie podejście pozwoli na potencjalne bierne ulepszanie naszego rozwiązania przez coraz to lepsze zastosowane rozwiązania w coraz to nowszych wersjach bibliotek. W celu aktualizacji warstwy prezentacji oraz logiki biznesowej wystarczy przejść odpowiednio do lokalizacji w której znajdują się dane warstwy wydając polecenie „npm update”. Wykonanie tej komendy spowoduję aktualizację wszystkich użytych pakietów w projekcie za pomocą menadżera pakietów npm [65].

# **Podręcznik użytkowania aplikacji webowej BoardFlow**

## **Instrukcja**

Aplikacja BoardFlow, jak większość aplikacji korzysta z pewnych danych nie jawnych przechowywanych jako zmienne środowiskowe. Jak zostało wspomniane w rozdziale 4.7, w głównym katalogu projektu znajduję się plik „.env”, którego zawartość została przedstawiona na rysunku 5.1. Zmienne te przechowują wartości nieedytowalne w momencie kiedy aplikacja jest uruchomiona. Pozwalają one w sposób dynamiczny, ale przede wszystkim bezpieczny przechowywać wartości takie, jak loginy i hasła używane w aplikacji. Zmiana portu lub danych logowania nie wymusza edytowania wartości w samym kodzie aplikacji. Objaśnienie znaczenia poszczególnych zmiennych środowiskowych:

* „BACKEND\_PORT” – port na którym jest uruchomiony kontener z logiką biznesową aplikacji,
* „MULTER\_STORAGE\_PATH” – ścieżka w której zapisywane są otrzymywane pliki,
* „SMTP\_HOST” – adres hosta serwera SMTP używanego do przesyłania wiadomości email [35],
* „SMTP\_PORT” – port używany do komunikacji za pośrednictwem protokołu SMTP [35],
* „SMTP\_USER” – nazwa użytkownika używanego do uwierzytelnienia połączenia z serwerem SMTP [35],
* „SMTP\_PASS” – hasło użytkownika używanego do uwierzytelnienia połączenia z serwerem SMTP [35],
* „SMTP\_FROM” – adres email, który będzie używany jako adres nadawcy wiadomości email [35],
* „FRONTEND\_PORT” – port na którym uruchomiony jest kontener z warstwą prezentacji aplikacji,
* „JWT\_SECRET” – tajny klucz używany do podpisywania tokenów JWT [39],
* „JWT\_EXPIRATION\_TIME” – czas po którym token dostępu traci swoją ważność,
* „JWT\_REFRESH\_EXPIRATION\_TIME” – czas po którym token odświeżający traci swoją ważność,
* „POSTGRESQL\_DB” – nazwa bazy danych używanej jako warstwa danych [24],
* „POSTGRESQL\_USER” – nazwa użytkownika używanego do uwierzytelnienia połączenia z bazą danych [24],
* „POSTGRESQL\_PASS” – hasło użytkownika używanego do uwierzytelnienia połączenia z bazą danych [24],
* „POSTGRESQL\_LOCAL\_PORT” – port na którym uruchomiony jest kontener z bazą danych, ten port jest widoczny na zewnątrz konteneru [24],
* „POSTGRESQL\_DOCKER\_PORT” – port na którym rzeczywiście chodzi uruchomiona baza danych [24],
* „POSTGRESQL\_HOST” – adres hosta bazy danych dzięki któremu można nawiązać połączenie z bazą danych [24],
* „REDIS\_HOST” – adres hosta bazy danych dzięki któremu można nawiązać połączenie z bazą danych Redis [28],
* „REDIS\_LOCAL\_PORT” – nazwa użytkownika używanego do uwierzytelnienia połączenia z bazą danych Redis [28],
* „REDIS\_DOCKER\_PORT” – hasło użytkownika używanego do uwierzytelnienia połączenia z bazą danych Redis [28].

Aplikacja w momencie uruchomienia posiada tylko jednego użytkownika z rangą administratora o emailu: „admin@admin.com” i hasłem: „password123”. Dane te wprowadzone do formularza logowania pozwolą na uzyskanie dostępu do panelu administratora aplikacji.

Obraz zawierający tekst, zrzut ekranu, menu

Opis wygenerowany automatycznie

Rysunek 5.1: Zawartość pliki konfiguracyjnego .env.

Źródło: opracowanie własne

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Rysunek 5.2: Zawartość pliku konfiguracyjnego docker-compose.yaml.

Źródło: opracowanie własne

## **Wprowadzenie do widoku klienta**

## **Wprowadzenie do widoku pracownika**

## **Wprowadzenie do widoku administratora**

# **Podsumowanie**

## **Wnioski**

Podsumowywując przeprowadzony cały cykl projektowy dla aplikacji webowej BoardFlow

## **Perspektywy dalszego rozwoju tematyki**

Tematykę procesów związanych z automatyzacją wszelkiego typu czynności można rozwijać na wiele sposobów w celu coraz to lepszego odciążania człowieka z odpowiedzialności na rzecz mechanizmów, które mogą wykonywać czynności za niego. W przypadku automatyzacji procesów wytwarzania własnoręcznie robionych gier planszowych nie jest inaczej. Przedstawione rozwiązanie można by było dalej rozwijać między innymi w przedstawiony niżej sposób.

Inteligentnego planowania pracy – wytworzenie każdego elementu zabiera cenny czas, który często potrafi być na wagę złota. Co za tym idzie złe rozplanowania kolejności wytwarzanych elementów może prowadzić do znaczących strat w tym obszarze. W szczególności kiedy uwzględnimy fakt, że część elementów powstaje na drukarce 3D, która potrafi godzinami drukować bardziej skomplikowane elementy. Usprawnienie rozwiązania w inteligentnego planistę mogło by pozwolić optymalnie wykorzystać czas operacyjny drukarki poprzez drukowanie elementów drobnych, a co za tym idzie których czas powstanie trwa krótko za dnia, kiedy człowiek może obsłużyć drukarkę. Z kolei elementów dużych lub skomplikowanych przez noc, kiedy to czas nie gra istotnej roli, bo nie ma w pracy osoby, która mogłaby wyjąć element z drukarki i przygotować pod druk kolejny. Zastosowanie automatycznego planisty dodatkowo mogłoby pomóc oszacować średni czas, kiedy można spodziewać się ukończenia projektu, a co za tym idzie być bardziej świadomym mocy przerobowej swojej firmy oraz pracowników.

Automatyczne zamawianie materiałów na bazie analizy trendów sprzedaży – wytwarzanie jakichkolwiek fizycznych rzeczy zawsze wymaga dostępności materiałów z których dany produkt ma powstać. Powoduję to ciągłe zapotrzebowanie na dany materiał. Jednak co jeżeli materiału do wytworzenia produktu, który niespodziewanie przestał się sprzedawać, zamówi się za dużo. W tym momencie generuję to stratę dla firmy pod kątem zamrożenia zainwestowanych pieniędzy w dany materiał. Tak samo w drugą stronę zamówienie za mało danego materiału generuję przestój, przez brak możliwości dalszej pracy. Oba przykłady dowodzą, że skuteczne zarządzanie stanem materiału w magazynie może mieć kluczowe znaczenie w prowadzeniu przedsiębiorstwa. Funkcjonalność usprawniająca ten proces mogłaby analizować sprzedaż danych gier planszowych i na tej podstawie przewidywać, którego materiału może zabraknąć i na tej podstawie sugerować jego dokup.

Architektura mikroserwisów – kolejną z możliwych ścieżek rozwoju tematyki może być przejście z architektury modularnego monolitu na mikroserwisy. Wspomniana architektura dzieli aplikację na niezależne serwisy, które są w stanie dalej przeprowadzać operację mimo przykładowego błędu jednego serwisu. Przedstawione podejście jest najczęściej stosowaną architekturą w nowoczesnych i profesjonalnych serwisach. W przypadku rozwoju tematyki aplikacji webowej propozycja ta pozwoliłaby zwiększyć niezawodność serwisu oraz potencjalnie szybkość jego działania przy dużych obciążeniach. Również aktualizacja lub ewentualna wymiana danych fragmentów projektu byłaby jeszcze prostsza przez totalną niezależność serwisów do tego stopnia, że każdy może być napisany w innym języku programowania z zastosowaniem różnych technologii.

# **Spis rysunków**

[Rysunek 2.1: Logotyp systemu StoryboardThat. 5](#_Toc151915677)

[Rysunek 2.2: Zdjęcie ekranu z aplikacji webowej StoryboardThat. 6](#_Toc151915678)

[Rysunek 2.3: Logotyp systemu TableTopia. 6](#_Toc151915679)

[Rysunek 2.4: Zdjęcie ekranu z aplikacji webowej TableTopia. 7](#_Toc151915680)

[Rysunek 2.5: Zdjęcie z aplikacji webowej TableTopia prezentujące tworzenie nowego obiektu. 7](#_Toc151915681)

[Rysunek 2.6: Logotyp systemu Olx. 8](#_Toc151915682)

[Rysunek 2.7: Zdjęcie ekranu z aplikacji webowej Olx. 8](#_Toc151915683)

[Rysunek 3.1: Wykres wartości rynkowej i prognozowanej w handlu internetowym na świecie według portalu ecommerce guide. 10](#_Toc151915684)

[Rysunek 3.2: Wykres reprezentujący rozkład użycia języków programowania w ciągu ostatnich 12 miesięcy. 11](#_Toc151915685)

[Rysunek 3.3: Wykres reprezentujący ilość pobrań języka programowania TypeScript. 11](#_Toc151915686)

[Rysunek 3.4: Logotyp języka programowania TypeScript. 12](#_Toc151915687)

[Rysunek 3.5: Logotyp frameworka NestJs. 12](#_Toc151915688)

[Rysunek 3.6: Logotyp frameworka React. 13](#_Toc151915689)

[Rysunek 3.7: Logotyp bazy danych PosgreSQL. 14](#_Toc151915690)

[Rysunek 3.8: Logotyp systemu kolejkowania BullMQ. 15](#_Toc151915691)

[Rysunek 3.9: Schemat działania systemu kolejkowania BullMQ. 15](#_Toc151915692)

[Rysunek 3.10: Schemat wzorca Producer Consumer. 16](#_Toc151915693)

[Rysunek 3.11: Schemat działania komunikacji z zastosowaniem protokołu SMTP. 16](#_Toc151915694)

[Rysunek 3.12: Diagram sekwencji dla możliwych scenariuszy strategii JWT. 17](#_Toc151915695)

[Rysunek 4.1: Diagram czynności dla przedsiębiorcy nie korzystającego z systemów informatycznych. 19](#_Toc151915696)

[Rysunek 4.2: Diagram czynności dla przedsiębiorcy korzystającego z systemów informatycznych. 20](#_Toc151915697)

[Rysunek 4.3: Diagram komponentów dla architektury trójwarstwowej. 23](#_Toc151915698)

[Rysunek 4.4: Diagram komponentów prezentujący architekturę modularną w kontekście aplikacji BoardFlow. Źródło: opracowanie własne 26](#_Toc151915699)

[Rysunek 4.5: Diagram komponentów dla modelu MVC. 26](#_Toc151915700)

[Rysunek 4.6: Diagram klas dla aplikacji BoardFlow. 28](#_Toc151915701)

[Rysunek 4.7: Diagram klas przedstawiający komponenty aplikacji BoardFlow. 29](#_Toc151915702)

[Rysunek 4.8: Diagram pakietów dla rozmieszczenia elementów dla całego systemu. 30](#_Toc151915703)

[Rysunek 4.9: Budowa modułu auth. 31](#_Toc151915704)

[Rysunek 4.10: Budowa modułu database. 32](#_Toc151915705)

[Rysunek 4.11: Budowa modułu exceptions. 32](#_Toc151915706)

[Rysunek 4.12: Budowa modułu game. 33](#_Toc151915707)

[Rysunek 4.13: Budowa modułu image. 33](#_Toc151915708)

[Rysunek 4.14: Budowa modułu information. 34](#_Toc151915709)

[Rysunek 4.15: Budowa modułu order. 34](#_Toc151915710)

[Rysunek 4.16: Budowa modułu project. 35](#_Toc151915711)

[Rysunek 4.17: Budowa modułu users. 36](#_Toc151915712)

[Rysunek 4.18: Przedstawienie modularnej budowy projektu. 37](#_Toc151915713)

[Rysunek 4.19: Listing kodu źródłowego implementacji dekoratora HasRole. 37](#_Toc151915714)

[Rysunek 4.20: Listing kodu źródłowego implementacji dekoratora GetCurrentUser. 38](#_Toc151915715)

[Rysunek 4.21: Listing kodu źródłowego implementacji OrderMemberShip guard. 38](#_Toc151915716)

[Rysunek 4.22: Diagram sekwencji dla sprawdzenia czy użytkownik jest właścicielem zasobu. 39](#_Toc151915717)

[Rysunek 4.23: Listing kodu źródłowego implementacji RolesGuard. 40](#_Toc151915718)

[Rysunek 4.24: Diagram sekwencji dla uwierzytelnienia opartego na rolach. 40](#_Toc151915719)

[Rysunek 4.25: Listing kodu źródłowego implementacji AuthorGuard. 41](#_Toc151915720)

[Rysunek 4.26: Diagram sekwencji dla sprawdzenia czy użytkownik może modyfikować projekt. 42](#_Toc151915721)

[Rysunek 4.27: Listing kodu źródłowego implementacji kolejkowania, producer. 42](#_Toc151915722)

[Rysunek 4.28: Listing kodu źródłowego implementacji kolejkowania, consumer. 43](#_Toc151915723)

[Rysunek 4.29: Diagram sekwencji dla rejestracji użytkownika. 43](#_Toc151915724)

[Rysunek 4.30: Listing kodu źródłowego implementacji strategii JWT. 44](#_Toc151915725)

[Rysunek 4.31: Listing kodu źródłowego implementacji JWT guard. 44](#_Toc151915726)

[Rysunek 5.1: Zawartość pliki konfiguracyjnego .env. 47](#_Toc151915727)

[Rysunek 5.2: Zawartość pliku konfiguracyjnego docker-compose.yaml. 48](#_Toc151915728)

# **Spis tabel**

[Tabela 2.1: Porównanie funkcjonalności oferowanych przez obecnie dostępne rozwiązania na rynku 9](#_Toc151395435)

# **Bibliografia**

[1] E-Commerce, [online], [dostęp 05.11.2023] https://www.techtarget.com/searchcio/definition/e-commerce

[2] Nest.js: A Progressive Node.js Framework, [online], [dostęp 06.11.2023] https://docs.nestjs.com/

[3] Dav Vanderkam, TypeScript: Skuteczne programowanie, APN Promise, 2020, ISBN 978-83-754-1420-2

[4] TypeScript, [online], [dostęp 06.11.2023] https://www.typescriptlang.org/

[5] Ian Sommerville, Software Engineering Ninth Edition, Pearson, 2011, ISBN 978-0-13-703515-

[6] Watterfall, [online], [dostęp 05.11.2023] https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model

[7] Wykres wartości rynkowej i prognozowanej w handlu internetowym na świecie, [online], [dostęp 07.11.2023] https://ecommerceguide.com/ecommerce-statistics/

[8] Wykres reprezentujący rozkład użycia języków programowania w ciągu ostatnich 12 miesięcy, [online], [dostęp 07.11.2023] https://www.jetbrains.com/lp/devecosystem-2022/

[9] Wykres reprezentujący ilość pobrań języka programowania TypeScript, [online], [dostęp 07.11.2023] https://npmtrends.com/typescript

[10] TypeScript decorators, [online], [dostęp 07.11.2023] https://www.typescriptlang.org/docs/handbook/decorators.html

[11] Encapsulation, [online], [dostęp 07.11.2023] https://www.sumologic.com/glossary/encapsulation/

[12] JavaScript, [online], [dostęp 08.11.2023] https://developer.mozilla.org/en-US/docs/Web/JavaScript

[13] Microsoft, [online], [dostęp 08.11.2023] https://www.microsoft.com/pl-pl

[14] Node.js, [online], [dostęp 09.11.2023] https://nodejs.org/en

[15] Promise, [online], [dostęp 09.11.2023] https://docs.nestjs.com/fundamentals/async-providers

[16] Callback, [online], [dostęp 09.11.2023] https://docs.nestjs.com/techniques/events

[17] Express, [online], [dostęp 09.11.2023] https://expressjs.com/

[18] Dependency Injection, [online], [dostęp 09.11.2023] https://docs.nestjs.com/fundamentals/custom-providers

[19] CLI, [online], [dostęp 09.11.2023] https://www.w3schools.com/whatis/whatis\_cli.asp

[20] GraphQL, [online], [dostęp 09.11.2023] https://graphql.org/

[21] Angular, [online], [dostęp 09.11.2023] https://angular.io/

[22] React, [online], [dostęp 10.11.2023] https://react.dev/

[23] SOLID, [online], [dostęp 10.11.2023] https://www.bmc.com/blogs/solid-design-principles/

[24] PostgreSQL, [online], [dostęp 10.11.2023] https://www.postgresql.org/

[25] Open source, [online], [dostęp 10.11.2023] https://opensource.org/licenses/

[26] PL/pgSQL, [online], [dostęp 10.11.2023] https://www.postgresql.org/docs/current/plpgsql.html

[27] BullMQ, [online], [dostęp 10.11.2023] https://docs.bullmq.io/

[28] Redis, [online], [dostęp 10.11.2023] https://redis.io/

[29] Kolejki, [online], [dostęp 10.11.2023] https://docs.nestjs.com/techniques/queues

[30] FIFO, [online], [dostęp 10.11.2023] https://www.investopedia.com/terms/f/fifo.asp

[31] LIFO, [online], [dostęp 10.11.2023] https://www.investopedia.com/terms/l/lifo.asp

[32] Schemat działania systemu kolejkowania BullMQ, [online], [dostęp 10.11.2023] https://docs.bullmq.io/bullmq-pro/groups

[33] Producer, Consumer wzorzec, [online], [dostęp 10.11.2023] https://medium.com/@karthik.jeyapal/system-design-patterns-producer-consumer-pattern-45edcb16d544

[34] Pakiet Mailer, [online], [dostęp 10.11.2023] https://nest-modules.github.io/mailer/docs/mailer.html

[35] SMTP, [online], [dostęp 10.11.2023] https://sendgrid.com/blog/what-is-an-smtp-server/

[36] Schemat komunikacji z zastosowaniem SMTP, [online], [dostęp 10.11.2023] https://www.educba.com/smtp-protocol/

[37] SSL, [online], [dostęp 10.11.2023] https://www.cloudflare.com/learning/ssl/what-is-ssl/

[38] TLS, [online], [dostęp 10.11.2023] https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/

[39] JWT, [online], [dostęp 10.11.2023] https://docs.nestjs.com/security/authentication

[40] Pakiet Schedule, [online], [dostęp 12.11.2023] https://docs.nestjs.com/techniques/task-scheduling

[41] Cron, [online], [dostęp 12.11.2023] https://docs.oracle.com/cd/E12058\_01/doc/doc.1014/e12030/cron\_expressions.htm

[42] Logotyp języka programowania TypeScript, [online], [dostęp 15.11.2023] https://upload.wikimedia.org/wikipedia/commons/thumb/4/4c/Typescript\_logo\_2020.svg/1200px-Typescript\_logo\_2020.svg.png

[43] Logotyp frameworka NestJs, [online], [dostęp 15.11.2023] https://miro.medium.com/v2/resize:fit:1358/1\*s9kgU8F1eB7Tzs7sG0YhBg.jpeg

[44] Logotyp frameworka React, [online], [dostęp 15.11.2023] https://upload.wikimedia.org/wikipedia/commons/thumb/a/a7/React-icon.svg/1150px-React-icon.svg.png

[45] Logotyp bazy danych PostgreSQL, [online], [dostęp 15.11.2023] https://upload.wikimedia.org/wikipedia/commons/thumb/2/29/Postgresql\_elephant.svg/1200px-Postgresql\_elephant.svg.png

[46] Logotyp systemu kolejkowania BullMQ, [online], [dostęp 15.11.2023] https://876297641-files.gitbook.io/~/files/v0/b/gitbook-x-prod.appspot.com/o/spaces%2F-LUuDmt\_xXMfG66Rn1GA%2Ficon%2FHOq80FSJicAlE4bVptC9%2Fbull.png?alt=media&token=10a2ba71-db1f-4d5c-8787-3dbedc8dd3ce

[47] Zalety języka TypeScript, [online], [dostęp 15.11.2023] https://www.droptica.pl/blog/co-jest-typescript-i-dlaczego-sprawdzi-sie-w-twoich-projektach/

[48] Opis bazy danych PostgreSQL, [online], [dostęp 15.11.2023] https://vavatech.pl/technologie/bazy-danych/postgresql

[49] Logotyp systemu StoryboardThat, [online], [dostęp 15.11.2023] https://www.storyboardthat.com/create/game-posters#

[50] Logotyp systemu TableTopia, [online], [dostęp 15.11.2023] https://tabletopia.com/

[51] Logotyp systemu Olx, [online], [dostęp 15.11.2023] https://play-lh.googleusercontent.com/IZbR5N9NRi4JZmiBkGsp7pUQikm8cQMZtnC2RN1e7xhU3u3-cObSYUSquVoqgeuRQw

[52] Trello, [online], [dostęp 15.11.2023] https://trello.com/pl/tour

[53] Zdjęcie ekranu z aplikacji webowej StoryBoardThat, [online], [dostęp 15.11.2023] https://www.storyboardthat.com/storyboards/poster-templates/game-poster-2/copy

[54] Zdjęcie ekranu z aplikacji webowej TableTopia, [online], [dostęp 19.11.2023] https://tabletopia.com/workshop/objects/featured

[55] Zdjęcie ekranu z aplikacji webowej Olx, [online], [dostęp 19.11.2023] https://www.olx.pl/elektronika/komputery/

[56] MVC, [online], [dostęp 25.11.2023] https://d1wqtxts1xzle7.cloudfront.net/50526307/MVC-libre.pdf?1480020702=&response-content-disposition=inline%3B+filename%3DModel\_View\_Controller\_MVC\_Architecture.pdf&Expires=1700915009&Signature=LDU2eYvi5Qkazm6h2ggoI9QR6nQGZ6mqkmkfNdZPr-peZm-skHuqaf576wq2z2Dvxrvu3JAeX9-uFQNNIg21pHTQz0ClIWG9cyihM3ol-7bO~1nFgzvdiUSCoN1r--o91-yNj9ThC~S3ZSSmgGWfZnfQDQcMbr1p~E1rJqP8eIEqPpf6X29xADxvfEfXxHrA5ccVNXeUpYwuE-OWzm0d5gAdwVO6KZvG7ObD8ZRd9nacQdqa5GIDPxTxMSZ5hdp~qPKvdMPr4MhWBAlXf1phgDPNytu8HXow4io0Ll2ph9TRurD1~jHZvUMPDnA-v5dnioY~XvLiMHTy8z9Ly~uYAw\_\_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

[57] ORM, [online], [dostęp 25.11.2023] https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=35eefea817172677f03b12baa2916060957cfb23#page=38

[58] Guards, [online], [dostęp 25.11.2023] https://docs.nestjs.com/guards

[59] TypeOrm, [online], [dostęp 25.11.2023] https://typeorm.io/

[60] File upload, [online], [dostęp 25.11.2023] https://docs.nestjs.com/techniques/file-upload

[61] NestJs authentication, [online], [dostęp 26.11.2023] https://docs.nestjs.com/security/authentication

[62] Bearer token, [online], [dostęp 26.11.2023] https://www.rfc-editor.org/rfc/rfc6750

[63] Docker, [online], [dostęp 26.11.2023] https://www.docker.com/

[64] Docker-Compose, [online], [dostęp 26.11.2023] https://docs.docker.com/compose/

[65] npm, [online], [dostęp 26.11.2023] https://www.npmjs.com/