



Politechnika Łódzka
Wydział Fizyki Technicznej, Informatyki
i Matematyki Stosowanej

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.....	5
1.1.	Problematyka i zakres pracy	5
1.2.	Cele pracy.....	5
1.3.	Przegląd literatury.....	6
1.4.	Układ pracy	6
2.	Przegląd wybranych aplikacji wspomagających proces zamawiania i projektowanie gier planszowych.....	7
2.1.	Aplikacja webowa StoryboardThat	7
2.2.	Aplikacja webowa TableTopia	8
2.3.	Aplikacja webowa Olx	10
2.4.	Zalety i wady wybranych aplikacji do zamawiania i wspomagania projektowania gier planszowych.....	11
3.	Stos technologiczny.....	12
3.1.	Uzasadnienie wyboru technologii.....	12
3.2.	Język programowania TypeScript.....	13
3.3.	Framework NestJs	14
3.4.	Framework React	15
3.5.	Baza danych PostgreSQL	16
3.6.	System kolejkowania BullMQ.....	16
3.7.	Pakiet Mailer	17
3.8.	Pakiet JWT	18
3.9.	Pakiet Schedule.....	19
4.	Cykl projektowy aplikacji webowej GameFlow	20
4.1.	Sylwetka klienta i jego wymagania	20
4.2.	Wymagania funkcjonalne.....	22
4.3.	Wymagania niefunkcjonalne.....	24
4.4.	Architektura aplikacji	25
4.4.1.	Warstwa prezentacji	26
4.4.2.	Warstwa logiki biznesowej	27
4.4.3.	Warstwa danych	29
4.5.	Implementacja – punkty kluczowe	33
4.5.1.	Modularna budowa projektu.....	33

4.5.2.	Implementacja personalnych dekoratorów	39
4.5.3.	Implementacja personalnych guards.....	40
4.5.4.	Realizacja kolejkowania	44
4.5.5.	Realizacja JWT.....	46
4.6.	Testy	47
4.7.	Instalacje i konserwacja	47
5.	Podręcznik użytkowania aplikacji webowej GameFlow	50
5.1.	Instrukcja.....	50
5.2.	Wprowadzenie do widoku użytkownika niezalogowanego	52
5.2.1.	Pasek nawigacyjny	52
5.2.2.	Stopka strony	52
5.2.3.	„Home” – Strona główna.....	53
5.2.4.	„Games” – Strona z produktami	55
5.2.5.	„About” – Strona zawierająca informację o firmie.....	56
5.2.6.	„Contact” – Strona zawierająca informację kontaktowe	57
5.2.7.	„Sign in” – Formularz logowania	58
5.2.8.	„Create an account” – Formularz rejestracji	59
5.3.	Wprowadzenie do widoku klienta	60
5.3.1.	„User Data” – Dane użytkownika.....	60
5.3.2.	„Orders” – Złożone zamówienia	60
5.3.3.	„Order” – Złożenie zamówienia na produkt	62
5.4.	Wprowadzenie do widoku pracownika.....	63
5.4.1.	„Manage Orders” – Zarządzaj zamówieniami	64
5.4.2.	„Manage Games” – Zarządzaj grami	67
5.4.3.	„Manage Tags” – Zarządzaj tagami.....	69
5.4.4.	„Manage Projects” – Zarządzaj projektami	70
5.4.5.	„My workspace” – Moja przestrzeń robocza.....	73
5.5.	Wprowadzenie do widoku administratora.....	77
5.5.1.	„Custom options” – Opcje niestandardowe	78
5.5.2.	„Manage employees” – Zarządzaj pracownikami	78
5.5.3.	„Manage Users” – Zarządzaj użytkownikami.....	81
6.	Podsumowanie	83
6.1.	Wnioski.....	83

6.2. Perspektywy dalszego rozwoju tematyki	83
Spis rysunków	85
Spis tabel.....	88
Bibliografia	89

Streszczenie

Słowa kluczowe

1. Wstęp

1.1. Problematyka i zakres pracy

Prowadzenie działalności gospodarczej od zawsze było wymagającym wyzwaniem. Od czasów powstania pierwszych sklepów przedsiębiorcy starają się dotrzeć do jak najszerzego 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 staje 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.

1.2. 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.

1.3. Przegląd literatury

Nest.js: A Progressive Node.js Framework – oficjalna dokumentacja techniczna framework'a *Nest.js* [2]. Zasób ten wybrano ze względu, że jest to jedna z lepiej napisanych dokumentacji na rynku. Opisuje ona działanie całej platformy programistycznej, jak i tego w jaki sposób w danym szkielecie aplikacyjnym należy dodawać oraz wykorzystać zadane zależności oraz biblioteki.

Dav Vanderkam, TypeScript: Skuteczne programowanie - książka zawierająca porady dotyczące dobrych praktyk i skutecznego posługiwanego się językiem programowania *TypeScript* [3, 4]. Zasób ten wybrano 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 wybrano ze względu na lepsze zrozumienie potencjału inżynierii oprogramowania oraz opisu krok po kroku wytwarzania profesjonalnego oprogramowania od wymagań klienta, aż po fazę konserwacji i utrzymania docelowego programu.

1.4. Układ pracy

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

2. 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.

2.1. Aplikacja webowa StoryboardThat

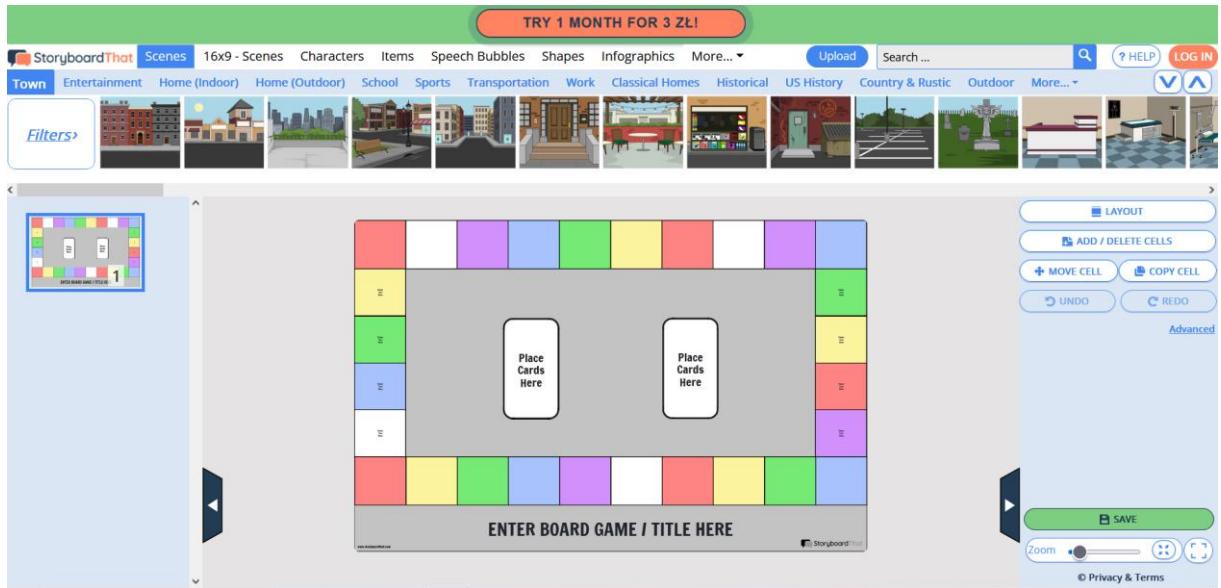


*Rysunek 2.1: Logo systemu StoryboardThat.
Źródło: [49]*

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 prezentuje 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 oferuje 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 oferuje 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 jednorazowo, 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.



Rysunek 2.2: Ekran aplikacji webowej StoryboardThat.
Źródło: [53]

2.2. Aplikacja webowa TableTopia



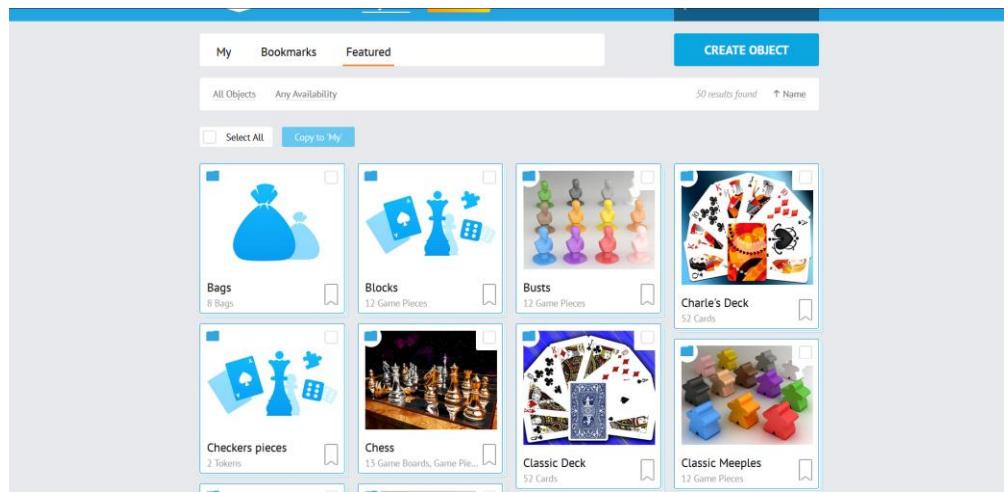
Rysunek 2.3: Logotyp systemu TableTopia.
Źródło: [50]

Aplikacja webowa *TableTopia* (rysunek 2.3) jest jednym z najbardziej rozbudowanych serwisów, jeżeli chodzi o projektowanie gier planszowych. Swoim użytkownikom oferuje 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 prezentuje 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 tworzenie nowych elementów od zera.

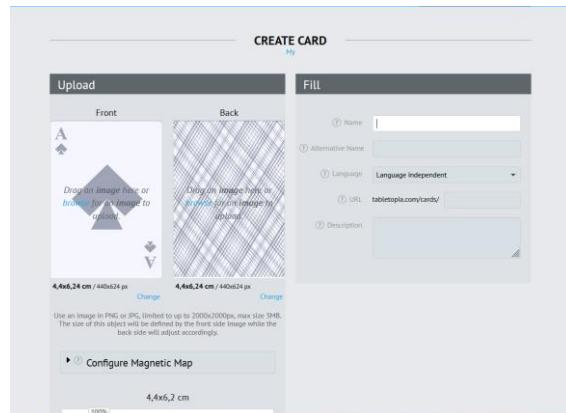
Aplikacja udostępnia również możliwość tworzenia własnych nowych obiektów o zadanej grafice, rozmiarze czy też nazwie, jednak nowe elementy muszą być elementami

danego typu na przykład karta lub kostka, co przedstawia 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.



Rysunek 2.4: Ekran aplikacji internetowej TableTopia.

Źródło: [54]



Rysunek 2.5: Tworzenie nowego obiektu w aplikacji internetowej TableTopia.

Źródło: [54]

2.3. Aplikacja webowa Olx



Rysunek 2.6: Logotyp systemu Olx.
Źródło: [51]

Aplikacja webowa *Olx* (rysunek 2.6) 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 2.7. 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 zadawanego 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 szczegół 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 emocje 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.

Znaleźliśmy ponad 1000 ogłoszeń

Produkt	Opis	Cena
Apple Macbook Pro 15 A1707 i7-7820HQ 16GB 512SSD Gwar FV23%	Używane Wrocław, Wola - Odtworzono dnia 18 listopada 2023	3 399 zł
KOMPUTER DO GIER RYZEN 5 5500/ RTX 3060/ 16GB RAM/ SSD 500GB/ Win11	Nowe Warszawa, Mokotów - Odtworzono dnia 17 listopada 2023	2 599 zł
Komputer gamingowy, RTX 3060 bardzo wydajny	Nowy Gdańsk - Odtworzono dnia 17 listopada 2023	4 000 zł

Rysunek 2.7: Ekran aplikacji webowej Olx.
Źródło: [55]

2.4. 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 można zaobserwować, 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 oferuje 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.

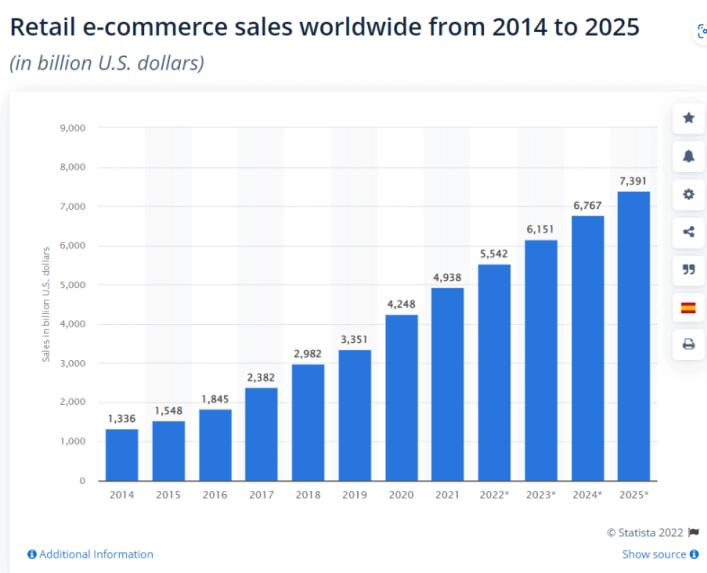
Tabela 2-1: Porównanie funkcjonalności oferowanych przez obecnie dostępne rozwiązania na rynku

	Tworzenie projektu	Edycja elementów	Dodawanie potrzebnych właściwości	Integracja z zewnętrznym API	Zarządzanie projektem	priortyzowanie i ustawianie statusu danych zadań	Sprawdzenia statusu z perspektywy klienta	Zamieszczanie produktów na sprzedaż	Zamieszczanie informacji o swojej działalności
StoryBoardThat	✓	x	x	x	x	x	x	x	x
TableTopia	✓	ograniczone	ograniczone	x	ograniczone	x	x	x	x
Olx	x	x	x	x	x	x	ograniczone	✓	ograniczone

3. Stos technologiczny

3.1. Uzasadnienie wyboru technologii

Przez ostatnie parę lat można zauważać zachodzącą rewolucję, w której Internet staje się drugą rzeczywistością handlu zwiększać swoją wartość o miliardy dolarów każdego roku, co przedstawia wykres dostępny rysunku 3.1. Oczywiście 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. Generuje to tym samym coraz więcej potrzeb *komputeryzacji* sklepów oraz systemów. Wynikiem tego procesu jest ogromny rozwój przeróżnych *frameworków* webowych i coraz to nowszych języków programowania.

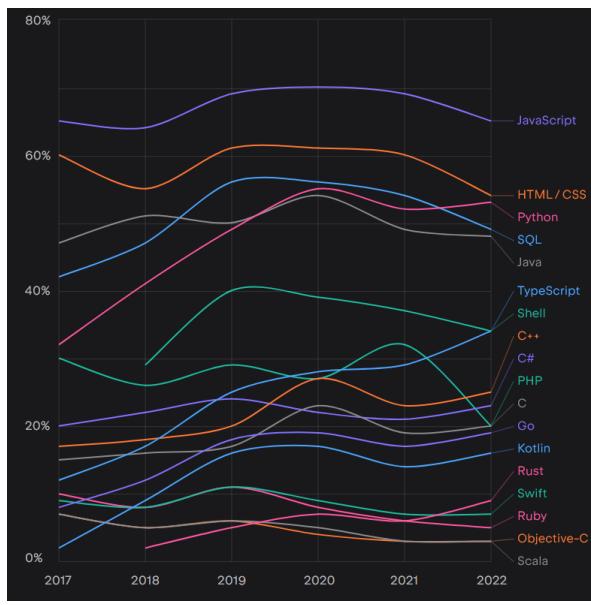


Rysunek 3.1: Wykres wartości rynkowej i prognozowanej w handlu internetowym na świecie według portalu ecommerce guide.

Źródło: [7]

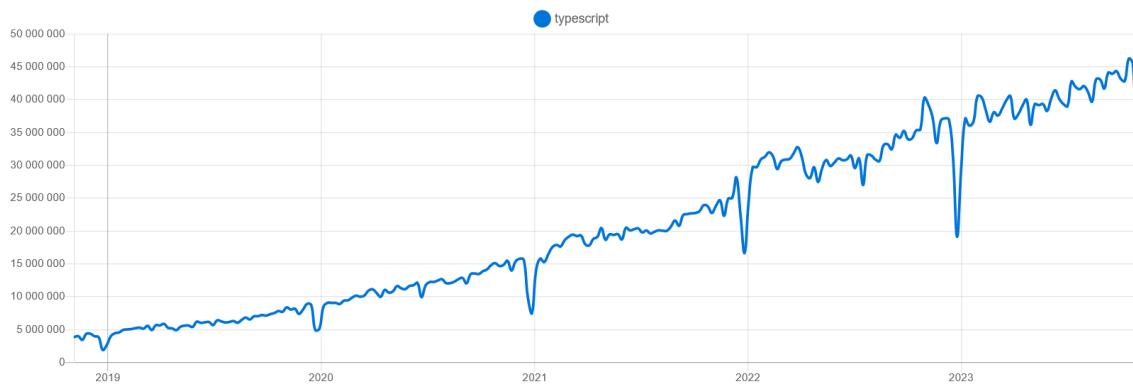
Jednym z takich języków programowania, który co roku notuje coraz większy udział na rynku jest język programowania *TypeScript*, którego wzrost popularności i liczba pobrań przedstawiona jest na wykresach na rysunkach 3.2 oraz 3.3 [4]. Wynika z tego jasno, że technologia ta zyskuje coraz to większą renomę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

- wprowadzanie typowania przez nadawanie zmiennym określonego typu danych. Pozwala to uniknąć wielu błędów z odwoływaniami 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 jawniej i prostej enkapsulacji zmiennych i metod klas, co pozwala zachować kontrolę nad udostępnianiem wewnętrznej logiki na zewnątrz obiektu [11].



Rysunek 3.2: Rozkład użycia języków programowania w ciągu ostatnich 12 miesięcy.

Źródło: [8]



Rysunek 3.3: Liczba pobrań języka programowania TypeScript.

Źródło: [9]

3.2. 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 zaprojektowany 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.

3.3. Framework NestJs

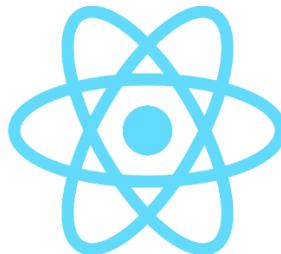


Rysunek 3.5: Logotyp frameworka NestJs.
Źródło: [43]

NestJS (rysunek 3.5) jest platformą programistyczną do budowy aplikacji serwerowych w środowisku uruchomieniowym *Node.js* [4, 14]. Umożliwia programistom programowanie w czystym języku programowania *JavaScript*, ale przede wszystkim przy użyciu języka programowania *TypeScript* [4, 12]. Łączy w sobie zasady programowania 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 wcześniejszej 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 monolith, mikroservisy jak i również aplikację *CLI* przy użyciu *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 [18, 21].

3.4. Framework React



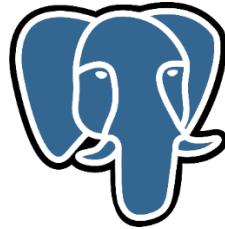
Rysunek 3.6: Logotyp frameworka React.
Źródło: [44]

React (rysunek 3.6) jest platformą programistyczną, a nawet bardziej biblioteką języka programowania *JavaScript* służącą do tworzenia interfejsów graficznych w środowisku uruchomieniowym *Node.js* [12, 14, 22]. Dysproporcja ta wynika z faktu, iż twór ten nie rozwiązuje 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 reprezentację komponentu;
- funkcji stanów, które pozwalają przetrzymać pewien zdefiniowany, zmieniany stan w cyklu życia *React*;
- funkcji ubocznej, 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 progranicie na bycie zgodnym z jedną z głównych zasad *SOLID* – „single responsibility” [23]. Warto również zauważyć, że *React* udostępnia możliwość programowania opartego o klasy, jednak nowoczesne podejście opiera się głównie na funkcjijnym podejściu do komponentów.

3.5. Baza danych PostgreSQL



Rysunek 3.7: Logotyp bazy danych PostgreSQL.
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].

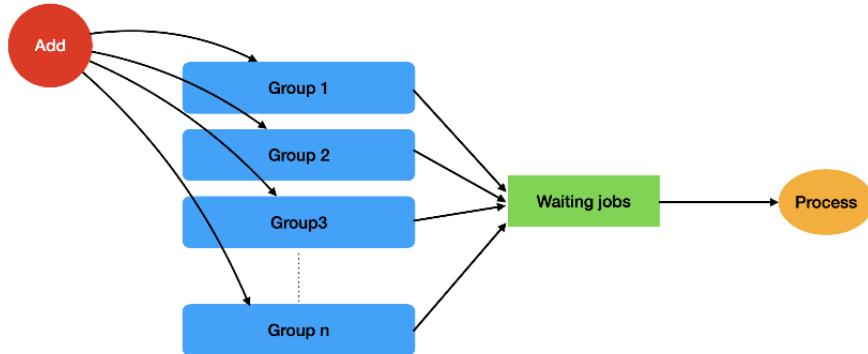
3.6. System kolejkowania BullMQ



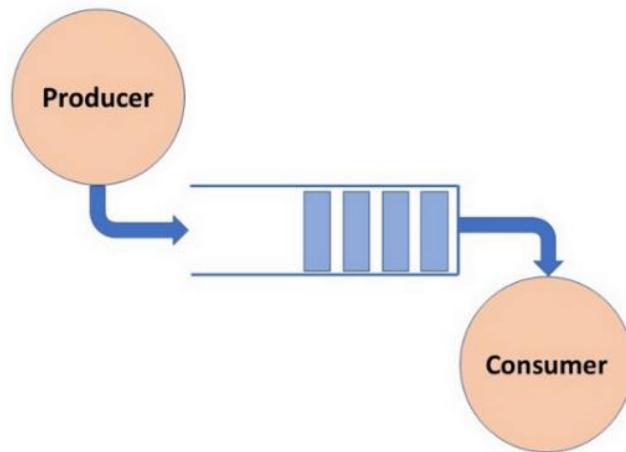
Rysunek 3.8: Logotyp systemu kolejkowania BullMQ.
Źródło: [46]

BullMQ (rysunek 3.8) jest biblioteką do obsługi kolejkowania zadań zbudowaną na bazie *Redis* 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 ogólną 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* - rysunek 3.10, gdzie następuje wykonanie i zakończenie danego zadania [29]. Sama kolejka może być skonfigurowana na wiele sposobów *LIFO*, *FIFO*, ale przede wszystkim poprzez przyznanie priorytetu danym zadaniom [30, 31].



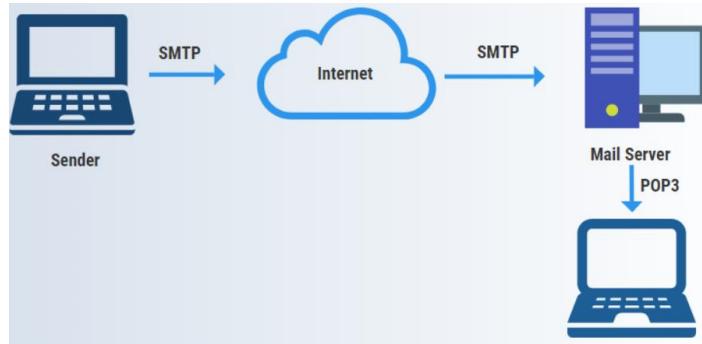
Rysunek 3.9: Schemat działania systemu kolejkowania BullMQ.
Źródło: [32]



Rysunek 3.10: Schemat wzorca Producer Consumer.
Źródło: [33]

3.7. 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 3.11 [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* [37, 38]. Warto również zauważyć, że wysyłanie e-mail jest operacją czasochlonną narażoną na wiele potencjalnych opóźnień, przez co moduł zachowuje się w sposób asynchroniczny.



Rysunek 3.11: Schemat działania komunikacji z zastosowaniem protokołu SMTP.
 Źródło: [36]

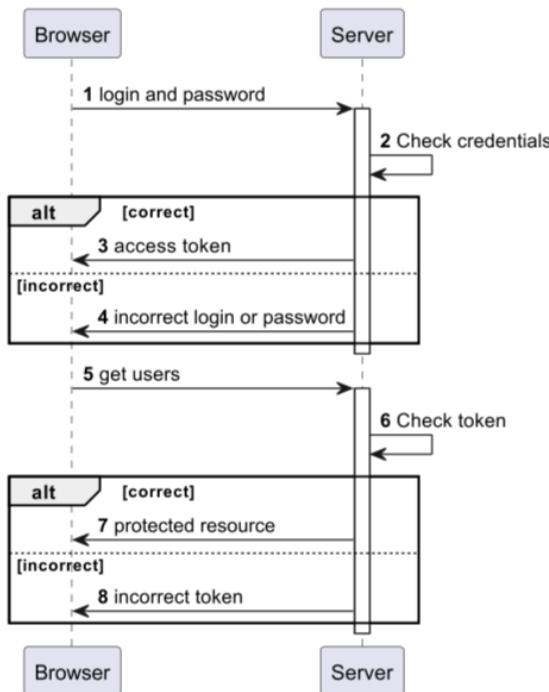
3.8. Pakiet JWT

Pakiet *JWT* jest modułem realizującym funkcjonalności zgodne z realizacją standardu *JSON Web Token*, czyli uwierzytelnieniem użytkownika [39]. Powyższa strategia polega na generowaniu tokenu dostępu zawierającego pewne dane, na podstawie których system przyznaje dostęp do zasobów użytkownikowi. Struktura *JSON Web Token* zbudowana jest z trzech części nagłówka, zawartości oraz podpisu. Nagłówek zazwyczaj zawiera informacje 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 diagramem sekwencji (rysunek 3.12):

- 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 otrzymuje 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.

Correct login and access to the protected resource



Rysunek 3.12: Diagram sekwencji dla możliwych scenariuszy strategii JWT.
 Źródło: opracowanie własne

3.9. 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* [14, 41]. Pakiet ten pozwala odciążyć użytkownika z pewnych powtarzalnych zadań w systemie, takich jak czyszczenie przedawnionych kodów lub wysyłanie cyklicznych emaliów. 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.

4. Cykl projektowy aplikacji webowej GameFlow

4.1. Sylwetka klienta i jego wymagania

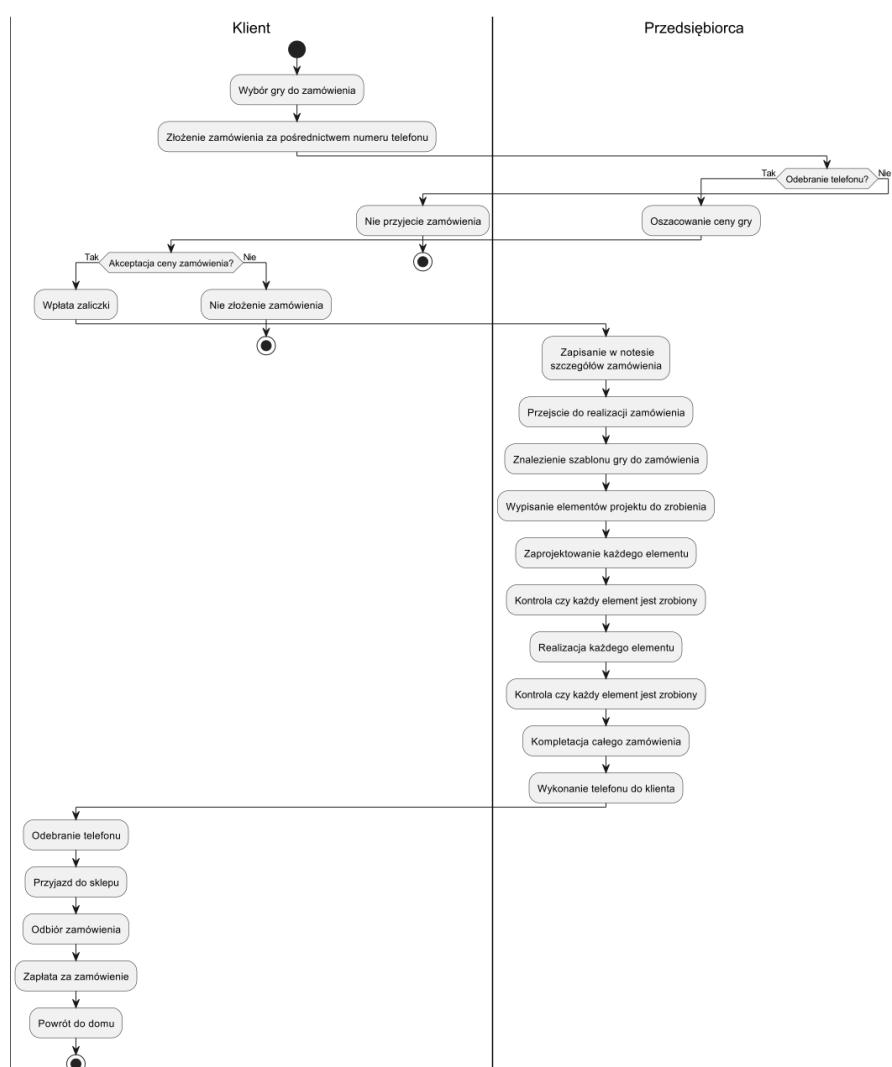
W celu lepszego zrozumienia klienta, analizę wymagań należy rozpocząć od określenia sylwetki klienta. Docelowym klientem aplikacji webowej *GameFlow* jest przedsiębiorca chcący zautomatyzować dotychczasowy cykl wytwarzania danego zamówienia na grę planszową od złożenia zamówienia po zaprojektowanie schematu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

Obecnie przedsiębiorca realizuje ten proces według diagramu czynności przedstawionego na rysunku 4.1. Analizując diagram czynności można zauważać 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 przez przedsiębiorcę o cenie dopiero wtedy i może na nią 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 modyfikacje lub dopytywać o postępy, co również jest dodatkowym *rozpraszaczem*.

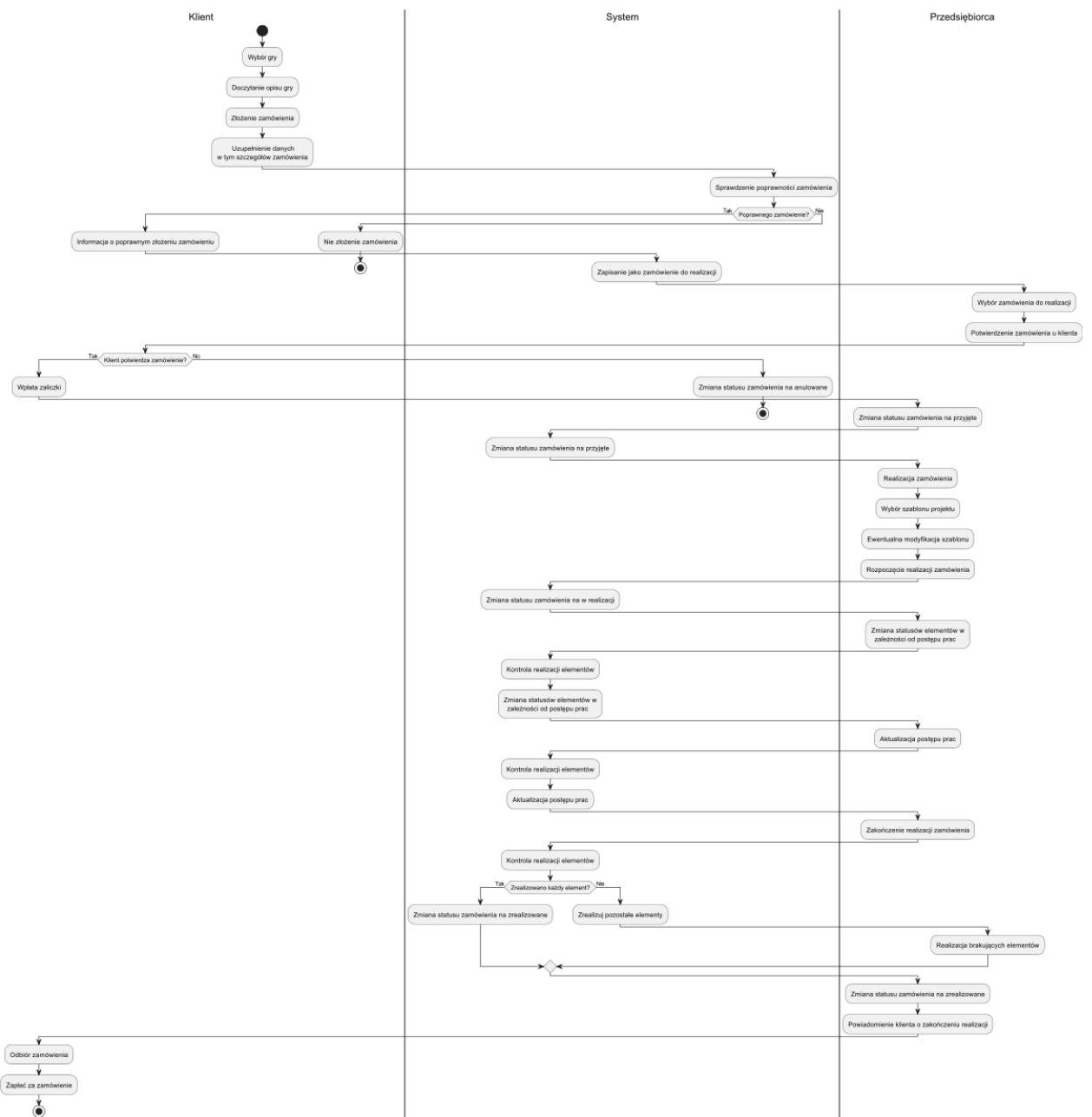
W opozycji do przeprowadzonej analizy, przedsiębiorca prowadzący swoją działalność za pomocą aplikacji webowej *GameFlow* realizuje zupełnie inny cykl wytwarzania zamówienia przedstawiony na diagramie czynności zamieszczonym na rysunku 4.2. Analizując wymieniony diagram czynności już na wstępnie możemy zauważyć, że część odpowiedzialności spoczywających na przedsiębiorcy została przeniesiona na system informatyczny, co za tym idzie cały cykl realizacji zamówienia jest w mniejszym stopniu narażony na porażkę przez współczynnik ludzki. Klient składając zamówienie już od samego początku jest świadomymi 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 składaże złożone zamówienie nad którym prace mogą rozpoczęć się w dowolnej chwili. Przedsiębiorca realizując kolejne elementy zna ich dokładne właściwości, jak wymiary, kolor jak i ewentualne opisy lub notatki. System cały czas kontroluje status danych elementów, które traktuje 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. Warto zaznaczyć, że system wspiera możliwość realizacji projektu bez złożonego zamówienia umożliwiając realizację projektu ze zgłoszenia telefonicznego lub czysto hobbystycznego powodu, jak projekt do portfolio.

Na podstawie 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



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



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

4.2. Wymagania funkcjonalne

Wymagania funkcjonalne to specyficzne funkcje, 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 twórcom systemów spełnić oczekiwania, które dane rozwiązanie ma spełniać.

W przypadku aplikacji webowej *GameFlow* 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 liczbie 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 udostępniać możliwość ciągłego rozszerzania oferty danego przedsiębiorstwa poprzez dodawanie nowych gier. 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 znajduje się 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 kryje się za zasłonami prowadzenia danej działalności może skłonić klienta do skorzystania 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 jawnych informacji na stronie. Pracownik ma możliwość wprowadzania nowych produktów do systemu, tworzenia nowych projektów gier, zarządzania użytkownikami. Administrator rozszerza uprawnienia pracownika o zarządzanie samymi pracownikami i ogólnymi informacjami zamieszczanymi na 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 liczby fałszywych kont przed możliwością zalogowania do systemu, użytkownik musi zweryfikować swoje konto poprzez aktywowanie go 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 do pewnego stopnia powinna udostępniać możliwość wyboru języka, w której możemy ją obsługiwać. Możliwość wyboru języka pozwoli dostosować się pod preferencję użytkownika.

4.3. 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 GameFlow wymagania niefunkcjonalne wraz z opisem również zostały zdefiniowane:

Architektura modularnego monolitu

Aplikacja została zbudowana na wzór architektury modularnej wydzielając tym samym luźno powiązane moduły, 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.

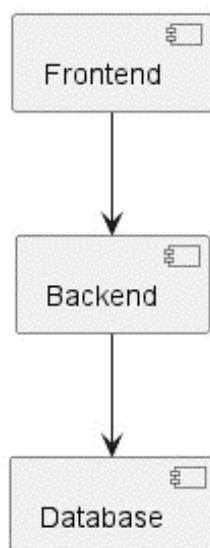
Bezpieczeństwo

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

4.4. Architektura aplikacji

Aplikacja *GameFlow* 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.



Rysunek 4.3: Diagram komponentów dla architektury trójwarstwowej.
 Źródło: opracowanie własne

4.4.1. Warstwa prezentacji

Warstwa prezentacji aplikacji *GameFlow* została 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 *GameFlow* 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 *GameFlow*;
- 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.

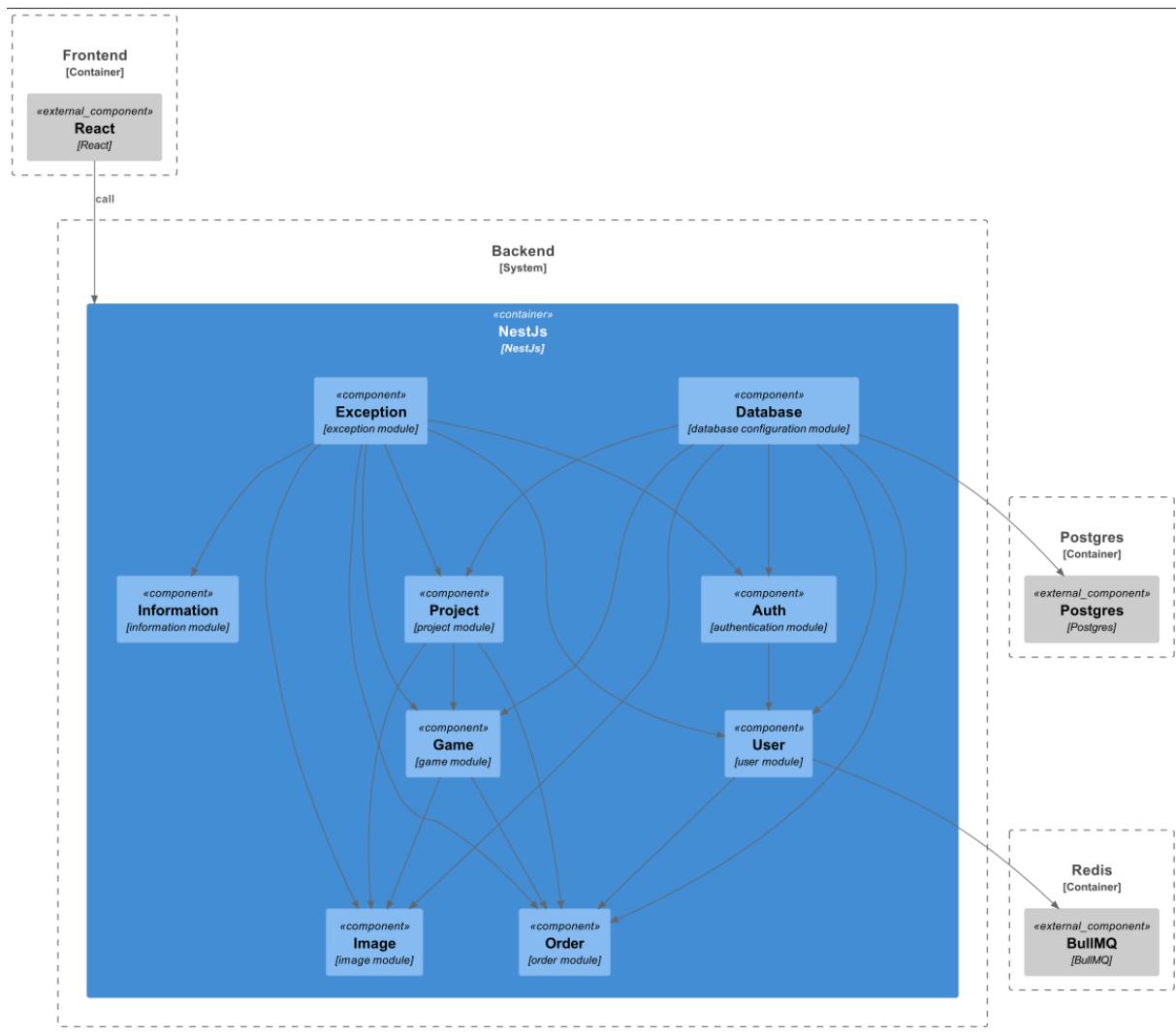
Więcej informacji o poszczególnych widokach i ich funkcjonalnościach zostało opisane w rozdziale 5.

4.4.2. Warstwa logiki biznesowej

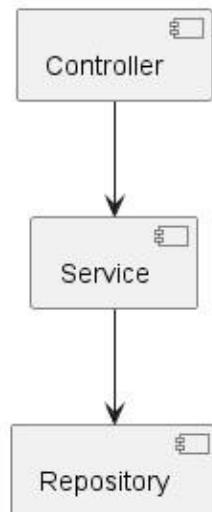
Warstwa logiki biznesowej aplikacji *GameFlow* została zbudowana według architektury modularnego monolitu rysunek 4.4. Oznacza to, że projekt jest pojedynczym serwisem z wewnętrznym podziałem na pojedyncze niezależne moduły, których logika jest odseparowana. Zastosowanie takiego podejścia pozwala wyodrębnić pełnione role do poszczególnych modułów na zasadzie jeden moduł, jedna pełniona odpowiedzialność. Każdy moduł aplikacji ma swoją własną logikę biznesową, która modyfikuje 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 podmianę tych już używanych przez ustanowione interfejsy, które są udostępniane przez poszczególne moduły. Architektura modularnego monolitu jest krokiem wprzód względem klasycznego monolitu, w którym 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 modyfikacjami danych;
- *repozytoria* – odpowiada za dostęp do danych warstw wyższych poprzez dostarczanie możliwości przeprowadzania modyfikacji danych w bazie danych.



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



Rysunek 4.5: Diagram komponentów dla modelu MVC.
 Źródło: opracowanie własne

4.4.3. Warstwa danych

Warstwa danych aplikacji *GameFlow* w architekturze trójwarstwowej odpowiada za przechowywanie, dostęp oraz zarządzanie danymi używanymi przez aplikację. W kontekście aplikacji *GameFlow* 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. Eliminuje to tym samym konieczność korzystania z surowych zapytań języka *SQL* w aplikacji, zastępując je interfejsem umożliwiającym wykonywanie operacji na bazie. Diagram klas przedstawiający relacje dla poszczególnych obiektów został zaprezentowany na rysunku 4.6. Analizując układ tych obiektów i zależności między nimi można wyróżnić 3 główne encje 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:

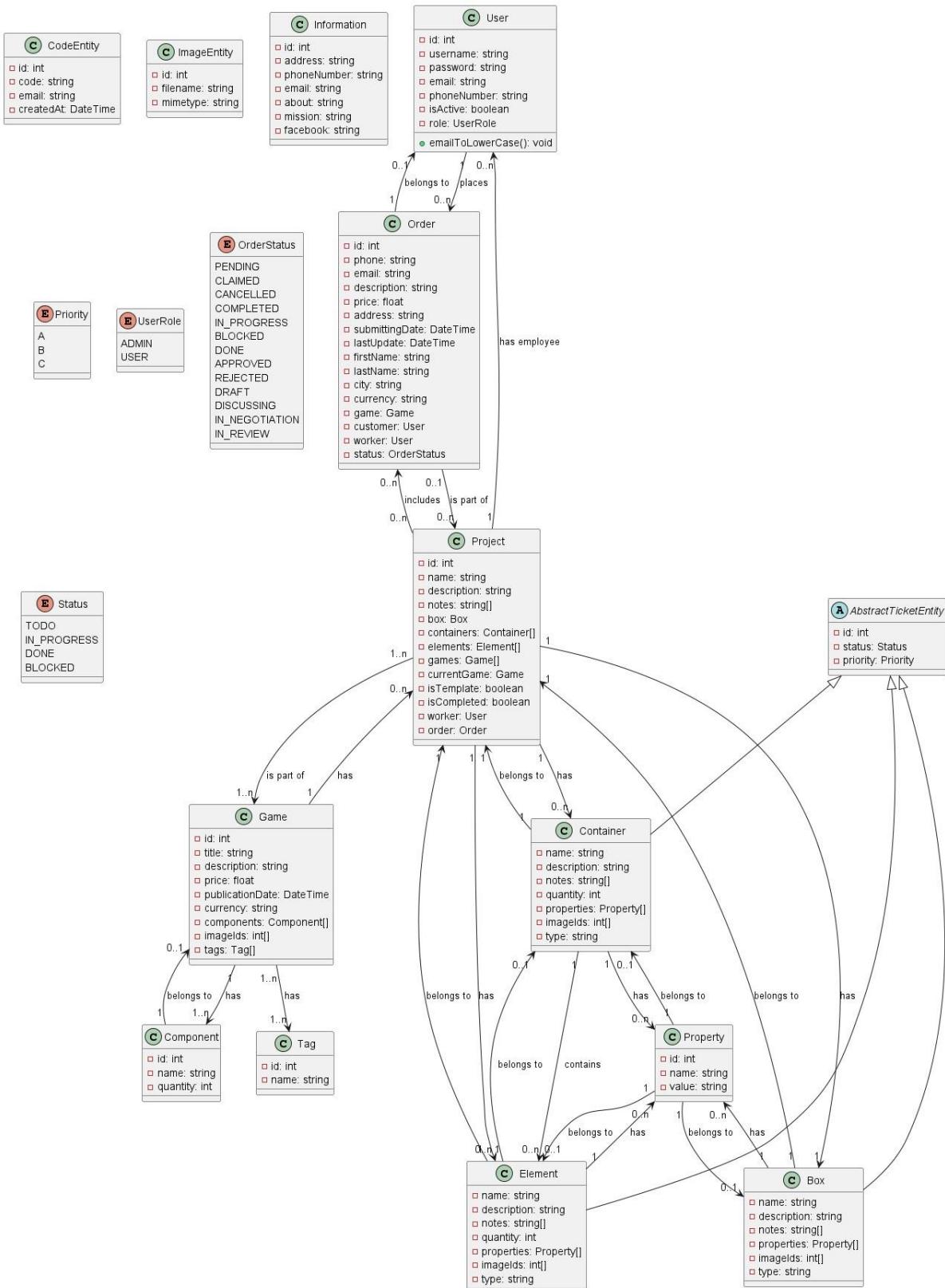
- pudełko („Box”) – reprezentujący pudełko, w którym znajduje się gra, relacja między projektem a grą to 1:1, ponieważ każda gra musi mieć swoje opakowanie, a dane opakowanie może być elementem tylko jednej gry;
- element („Element”) – reprezentuje element danej gry do zrobienia, czyli niezbędną rzecz 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”) – reprezentuje 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 reprezentuje produkt gry planszowej, który jest dostępny do zakupu jako finalny produkt procesu projektowania danej gry. W jego skład wchodzą elementy takie jak:

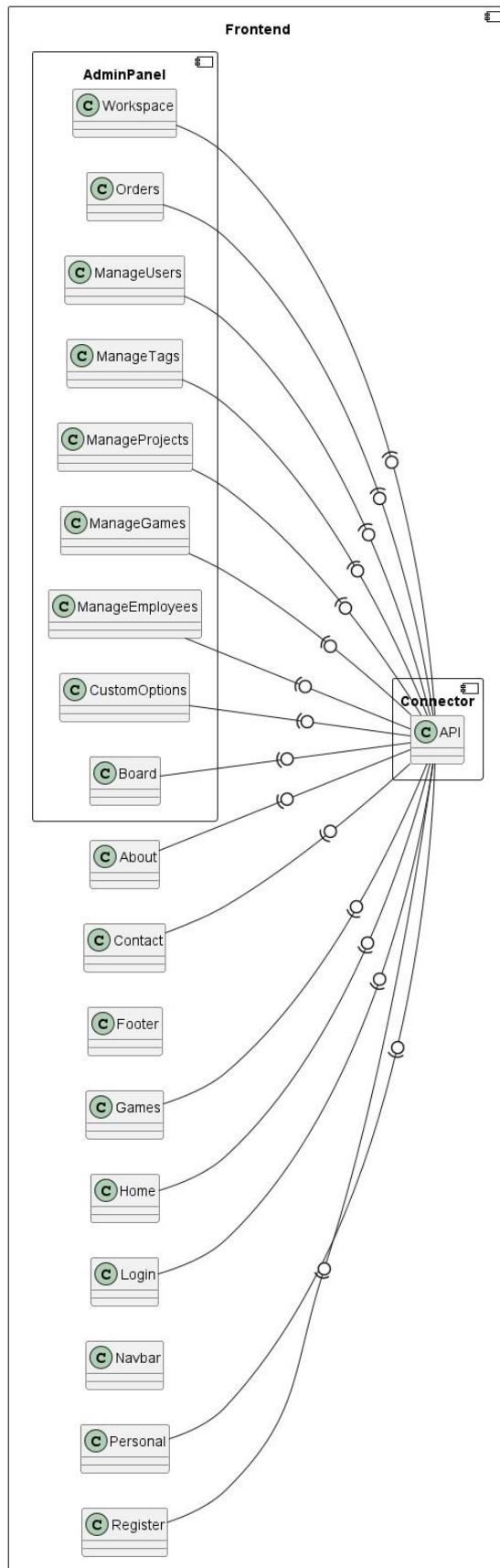
- tagi („Tag”) – reprezentujące kategorię, do której dana gra przynależy. Ułatwia to wyszukiwanie danej gry po kategoriach, do których gra przynależy. Relacja między grą a tagiem, to wiele do wielu, ponieważ dana kategoria może należeć do wielu gier, a dana gra może mieć wiele kategorii;
- komponenty („Components”) – reprezentujące składowe będące elementami 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.

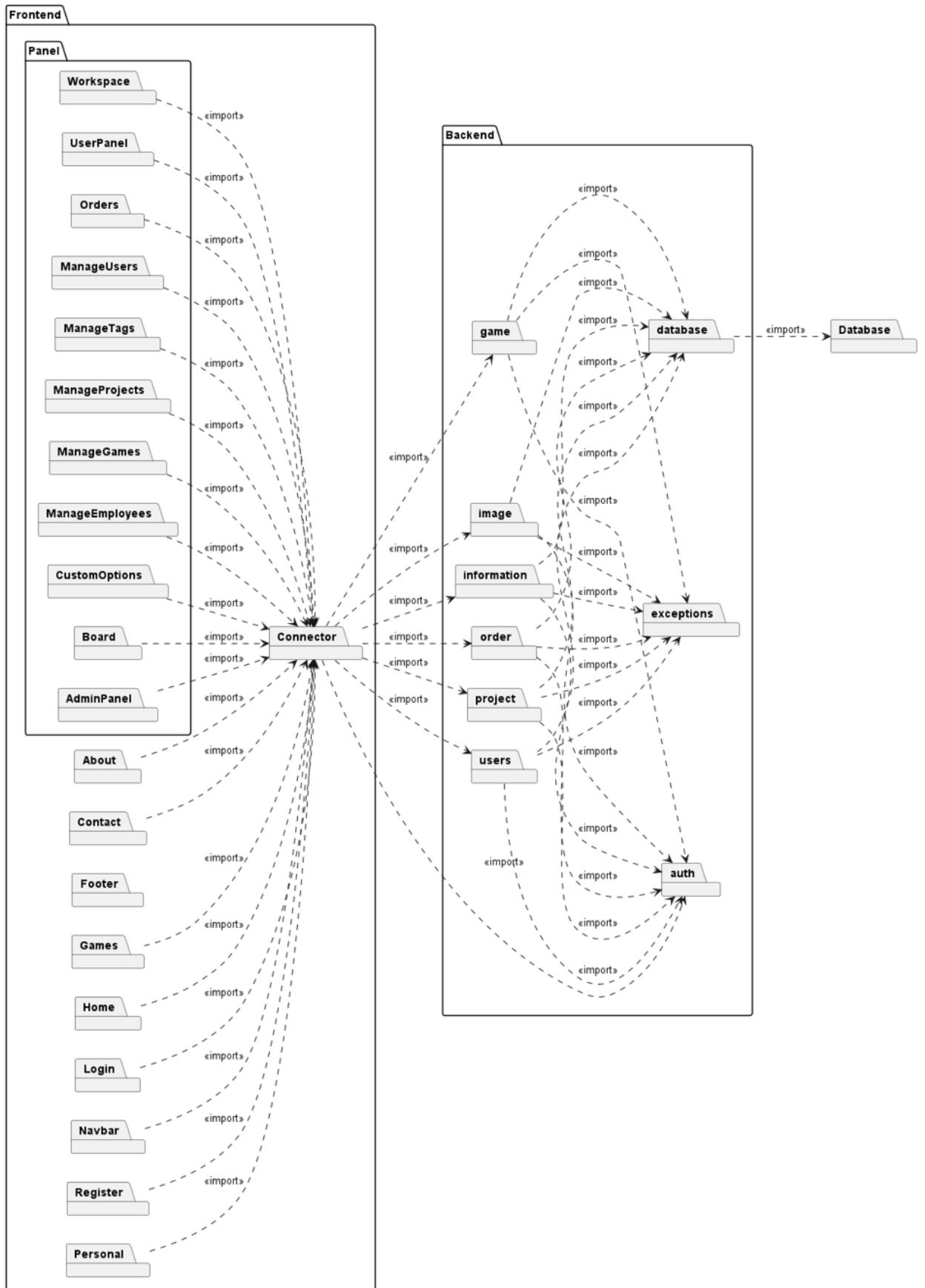
Wszystkie przeanalizowane fragmenty całej aplikacji można przedstawić na diagram pakietów przedstawiającym wizualizację i rozmieszczenie elementów w systemie - rysunek 4.8.



Rysunek 4.6: Diagram klas dla aplikacji GameFlow.
 Źródło: opracowanie własne



Rysunek 4.7: Diagram klas przedstawiający komponenty aplikacji GameFlow.
 Źródło: opracowanie własne



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

Źródło: opracowanie własne

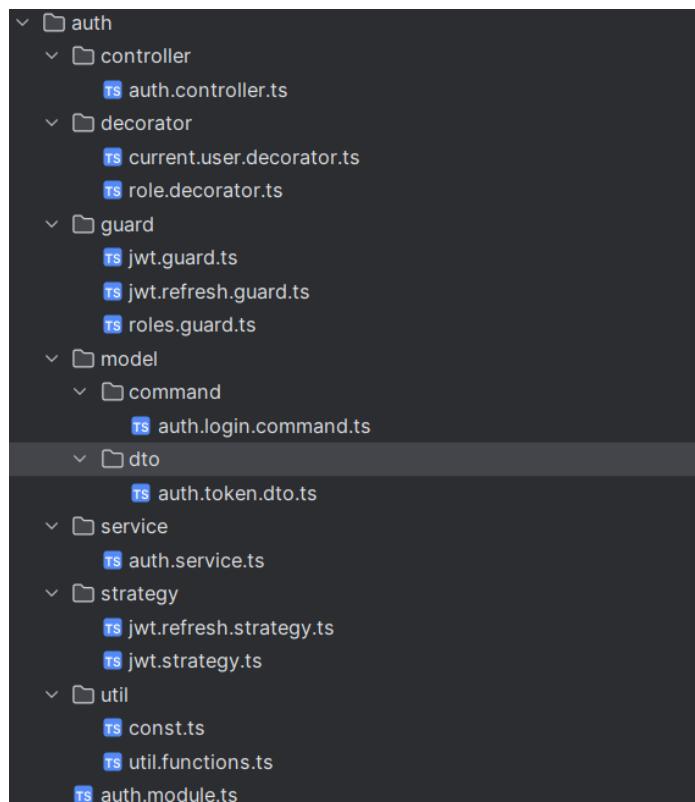
4.5. Implementacja kluczowych punktów aplikacji GameFlow

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 [4, 10, 39, 58]. Te najważniejsze punkty implementacji rozwiązania zostały opisane w tym rozdziale.

4.5.1. 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:

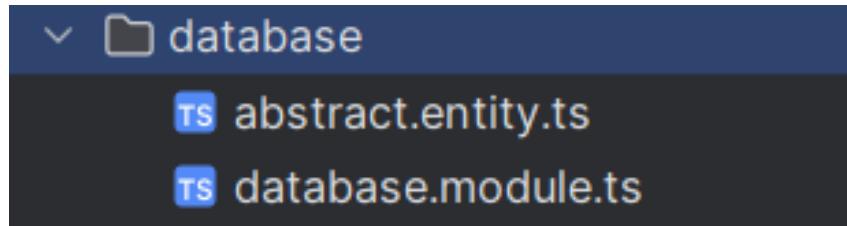
- moduł *auth* – (rysunek 4.9) moduł odpowiedzialny za obsługę funkcjonalności związanych z uwierzytelnieniem żądania i późniejszym autoryzowaniem go, przyznając mu dostęp do zasobów. Zastosowano w nim strategie *JWT* wraz z uwierzytelnieniem opartym na rolach w celu zabezpieczenia zasobów. Wykorzystuje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



Rysunek 4.9: Budowa modułu auth.

Źródło: opracowanie własne

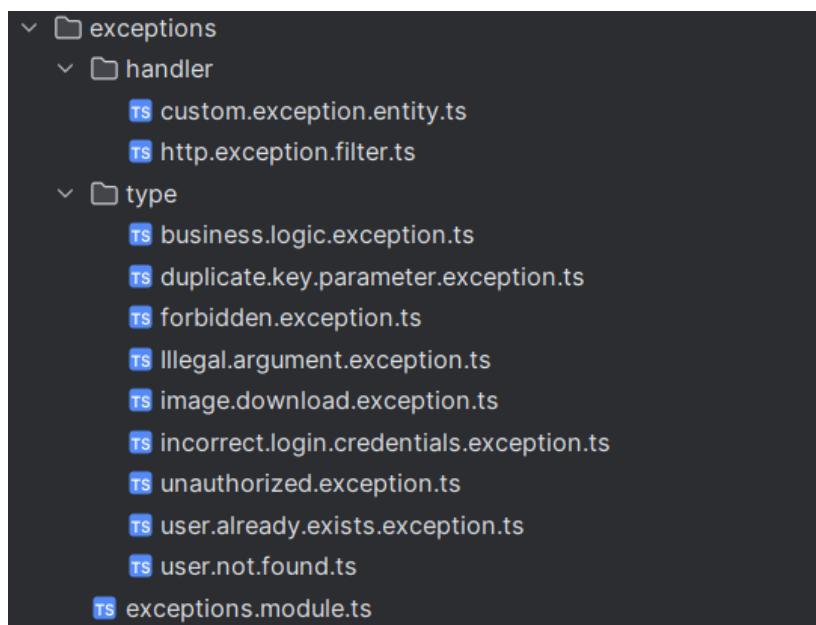
- moduł *database* – (rysunek 4.10) moduł odpowiedzialny za konfigurację połączenia z bazą danych. Zastosowano w nim bibliotekę *TypeORM* do realizacji podejścia „Object Relational Mapping” umożliwiającego mapowanie obiektów na rekordy bazy danych [59], [57];



Rysunek 4.10: Budowa modułu *database*.

Źródło: opracowanie własne

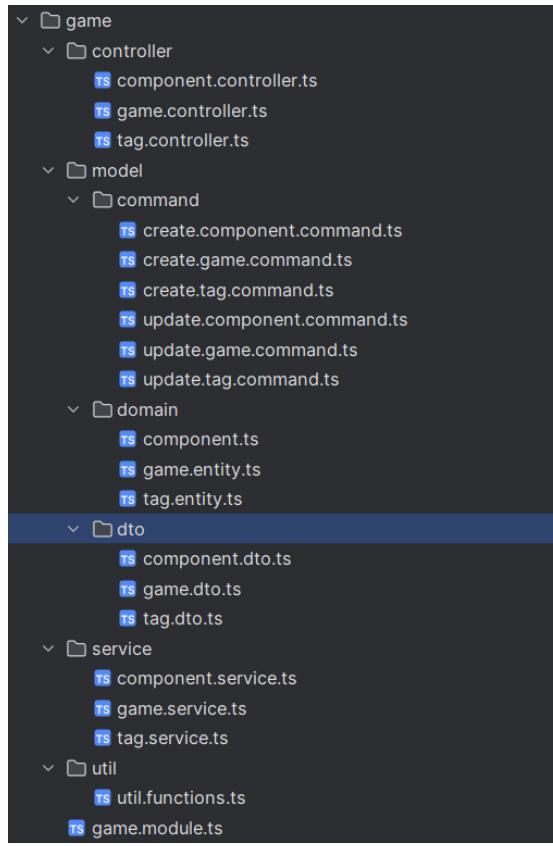
- moduł *exceptions* – (rysunek 4.11) moduł odpowiedzialny za obsługę wyjątków w całej aplikacji, pozwala on w łatwy sposób zwracać różnego typu błędy w momencie wystąpienia w jakikolwiek innym module. Mapuje on dany błąd na wiadomość i przypisuje mu odpowiedni kod błędu w zależności, czego dany wyjątek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



Rysunek 4.11: Budowa modułu *exceptions*.

Źródło: opracowanie własne

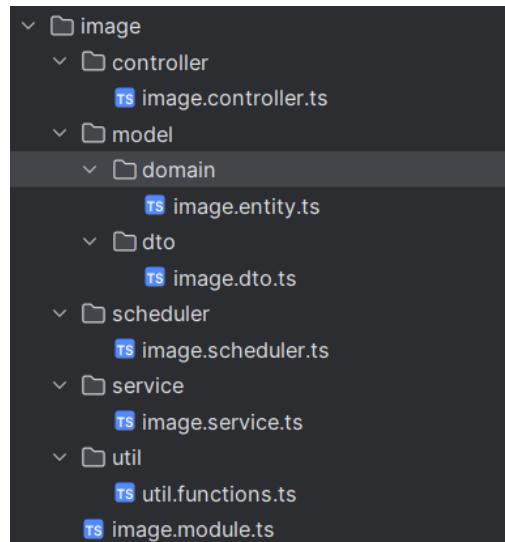
- moduł *game* – (rysunek 4.12) moduł odpowiedzialny za obsługę funkcjonalności związanych z logiką dotyczącą zarządzania dostępnymi grami planszowymi w systemie. Wykorzystuje w tym celu zależności do modułów autoryzacji, bazy danych i wyjątków;



Rysunek 4.12: Budowa modułu game.

Źródło: opracowanie własne

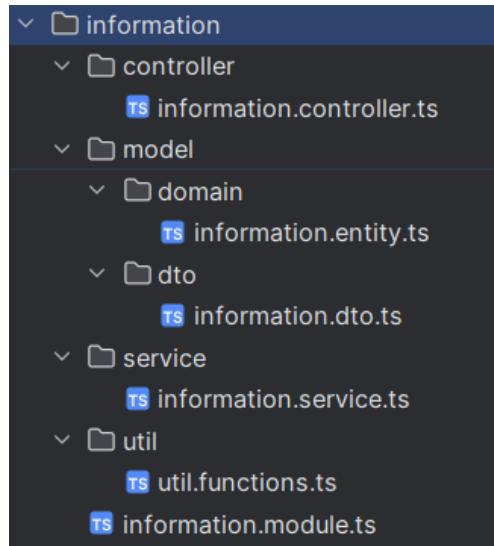
- moduł *image* – (rysunek 4.13) moduł odpowiedzialny za obsługę przechowywania plików obrazów w aplikacji. Wykorzystuje on predefiniowane komponenty obsługi przesyłu i odbioru plików w framework'u *NestJs* do realizacji funkcjonalności związanych z przesyłem plików w żądaniach [2, 60]. Wykorzystuje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



Rysunek 4.13: Budowa modułu image.

Źródło: opracowanie własne

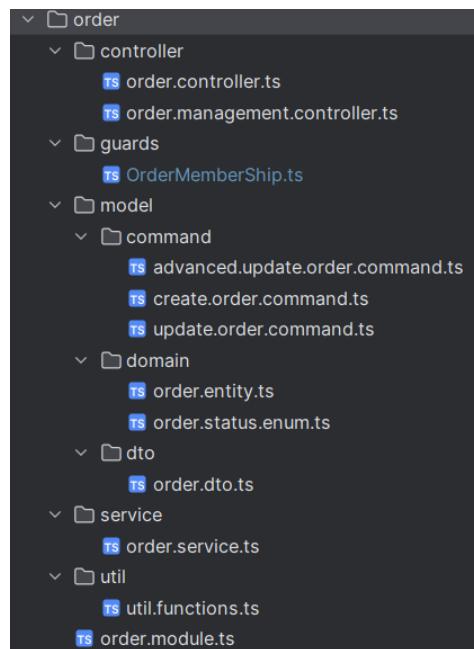
- moduł *information* – (rysunek 4.14) moduł odpowiedzialny za ustawianie aktualnie wyświetlanych informacji na stronie, takich jak opisy, cele firmy ale przede wszystkim dane kontaktowe do przedsiębiorstwa. Wykorzystuje w tym celu zależności do modułów autoryzacji, bazy danych i wyjątków;



Rysunek 4.14: Budowa modułu *information*.

Źródło: opracowanie własne

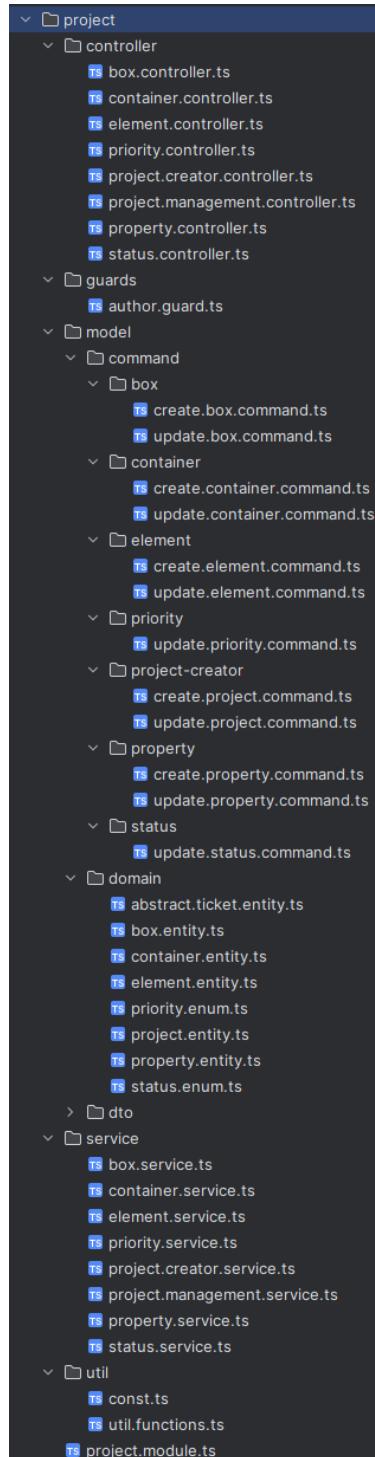
- moduł *order* – (rysunek 4.15) moduł odpowiedzialny za realizację funkcjonalności związanych z obsługą składania oraz zarządzania zamówieniami. Wykorzystuje w tym celu zależności do modułów autoryzacji, bazy danych i wyjątków;



Rysunek 4.15: Budowa modułu *order*.

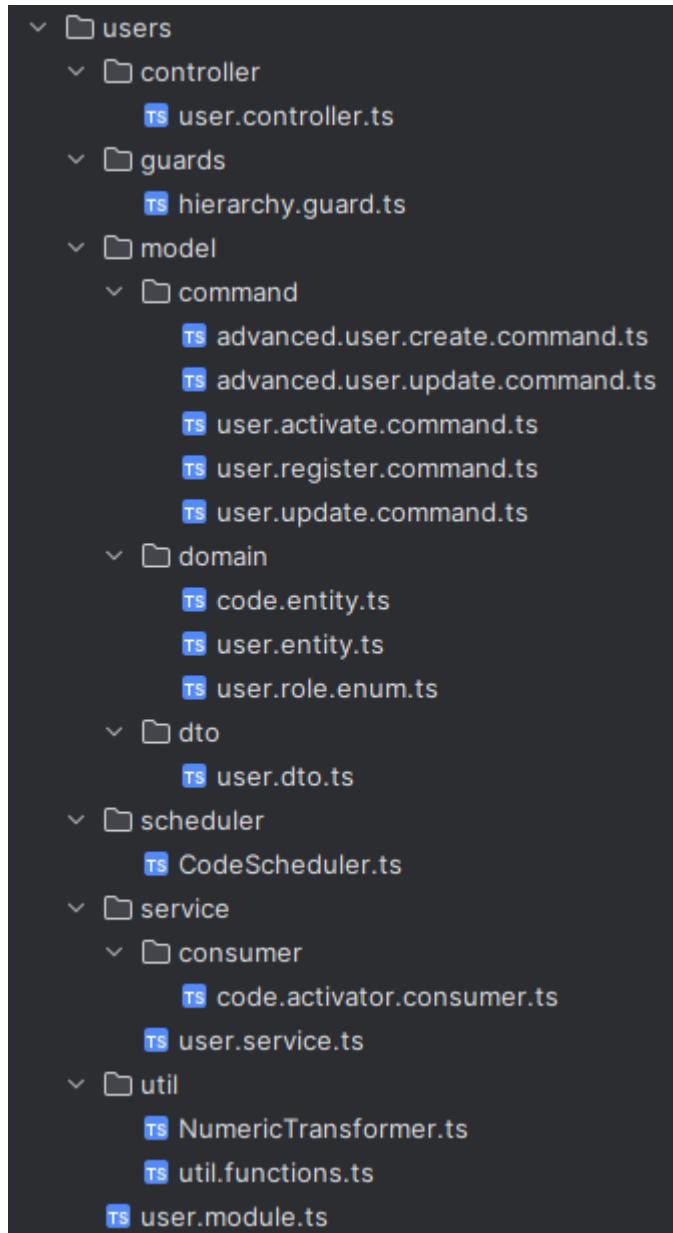
Źródło: opracowanie własne

- moduł *project* – (rysunek 4.16) 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. Wykorzystuje 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. Wykorzystuje w tym celu zależności do modułów autoryzacji, bazy danych, wyjątków, gier oraz użytkowników;

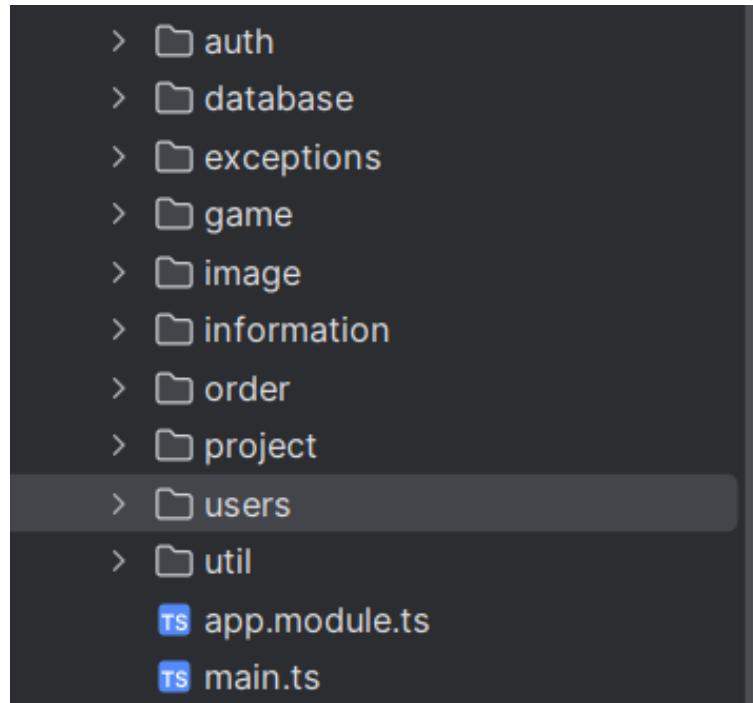


Rysunek 4.16: Budowa modułu *project*.
 Źródło: opracowanie własne

- moduł *users* – (rysunek 4.17) moduł odpowiedzialny za realizację funkcjonalności związanych z obsługą użytkowników systemu. Realizuje również obsługę kodów aktywacyjnych za pośrednictwem serwera *SMTP* [35]. W tym module również swoje zastosowanie znalazły system kolejkowania, opisany w rozdziale 3.6, realizujący wzorzec *producer consumer* [33]. Wykorzystuję on w tym celu zależności do modułów autoryzacji, bazy danych i wyjątków.



Rysunek 4.17: Budowa modułu *users*.
Źródło: opracowanie własne



Rysunek 4.18: Przedstawienie modularnej budowy projektu.
 Źródło: opracowanie własne

4.5.2. 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 komplikacji [4, 10]. W projekcie w ramach pracy dyplomowej użyto własne *dekoratory* opisane w dalszej części tego rozdziału.

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.

```
export const HasRoles = (...hasRoles: string[]) => SetMetadata(ROLES, hasRoles);
```

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 do wyciągnięcia z kontekstu, czyli żądania personaliów użytkownika, które mogą być użyte na przykład przy automatycznym przypisaniu zamówienia do właściciela żądania.

```

export const GetCurrentUser = createParamDecorator(
  factory: (_: undefined, context: ExecutionContext): number => {
    const request = context.switchToHttp().getRequest();
    return request.user;
  },
);

```

Rysunek 4.20: Listing kodu źródłowego implementacji dekoratora `GetCurrentUser`.
 Źródło: opracowanie własne

4.5.3. Implementacja personalnych strażników

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. W projekcie w ramach pracy dyplomowej użyto własne *guard'sy* opisane w dalszej części tego rozdziału.

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 wyłącznie jego właścicielem. Takie podejście było konieczne ze względu na konieczność weryfikacji tożsamości przed umożliwieniem aktualizacji danych, co pokazano na diagramie sekwencji przedstawionym na rysunku 4.22.

```

@Injectable()
export class OrderMembership implements CanActivate {

  no usages ▾ Damiaan
  CodiumAI: Test this method.
  constructor(
    private reflector: Reflector,
    private readonly userService: UserService,
    private readonly orderService: OrderService
  ) {}

  no usages ▾ Damiaan
  CodiumAI: Test this method.
  async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest();
    const params = request.params;

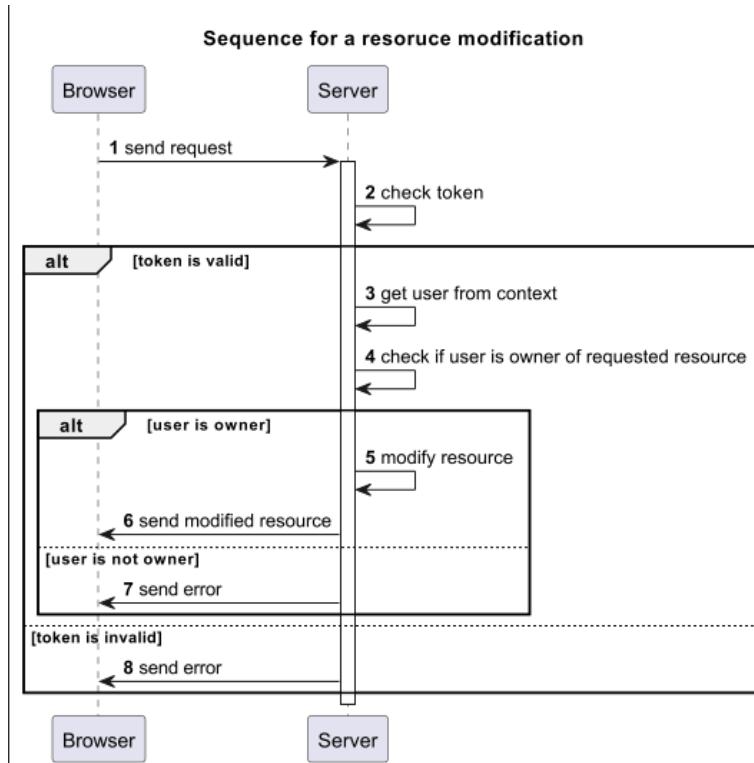
    const order: Order = await this.orderService.getOrderById(params.id);

    return this.isOrderOwner(request.user, order);
  }

  1 usage ▾ Damiaan
  private isOrderOwner(user, order: Order): boolean {
    return user.id === order.customer.id;
  }
}

```

Rysunek 4.21: Listing kodu źródłowego implementacji `OrderMembership guard`.
 Źródło: opracowanie własne



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 rolę w systemie, która zezwala na dostęp do danego fragmentu w aplikacji. Zastosowanie *guard’sa „RolesGuard”* wraz z *dekoratorem „HasRoles”* realizuje 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 weryfikuje, 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.

```

@Injectable()
export class RolesGuard implements CanActivate {

  no usages  ↳ Damaan
  CodiumAI: Test this method.
  constructor(
    private reflector: Reflector,
    @Inject(forwardRef( fn: () => UserService))
    private userService: UserService
  ) { }

  no usages  ↳ Damaan
  CodiumAI: Test this method.
  canActivate(context: ExecutionContext): boolean | Promise<boolean> | Observable<boolean> {
    const roles : string[] = this.reflector.get<string[]>(ROLES, context.getHandler());
    if (!roles) {
      return true;
    }

    const request = context.switchToHttp().getRequest();
    const user: User = request.user;

    if (!user) {
      return false;
    }

    return this.userService.findOne(user.id).then((user: User) => {
      1 usage  ↳ Damaan
      const hasRole = (o : boolean => roles.indexOf(user.role) > -1);
      let hasPermission: boolean = false;

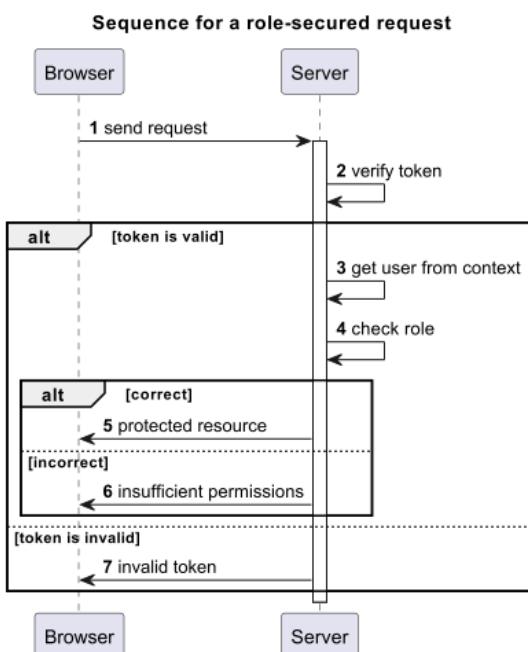
      if (hasRole()) {
        hasPermission = true;
      }

      return user && hasPermission;
    });
  }
}

```

Rysunek 4.23: Listing kodu źródłowego implementacji RolesGuard.

Źródło: opracowanie własne



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*. Realizuje tym samym funkcjonalność sprawdzenia, czy dane żądanie powinno uzyskać dostęp do modyfikacji projektu co przedstawia diagram sekwencji ukazany na rysunku 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.

```
@Injectable()
export class AuthorGuard implements CanActivate {

    no usages  ▲ Damiaan
    CodiumAI: Test this method.
    constructor(
        private readonly userService: UserService,
        private readonly projectService: ProjectCreatorService,
    ) {}

    no usages  ▲ Damiaan
    CodiumAI: Test this method.
    async canActivate(context: ExecutionContext): Promise<boolean> {
        const request = context.switchToHttp().getRequest();
        const params = request.params;

        const sender: User = await this.userService.findOne(request.user.id);
        const project : ProjectDto = await this.projectService.getProjectDtoById(params.id);

        if (this.canAccessTemplate(project)) {
            return true;
        }

        if (this.isProjectCompleted(project)) {
            return false;
        }

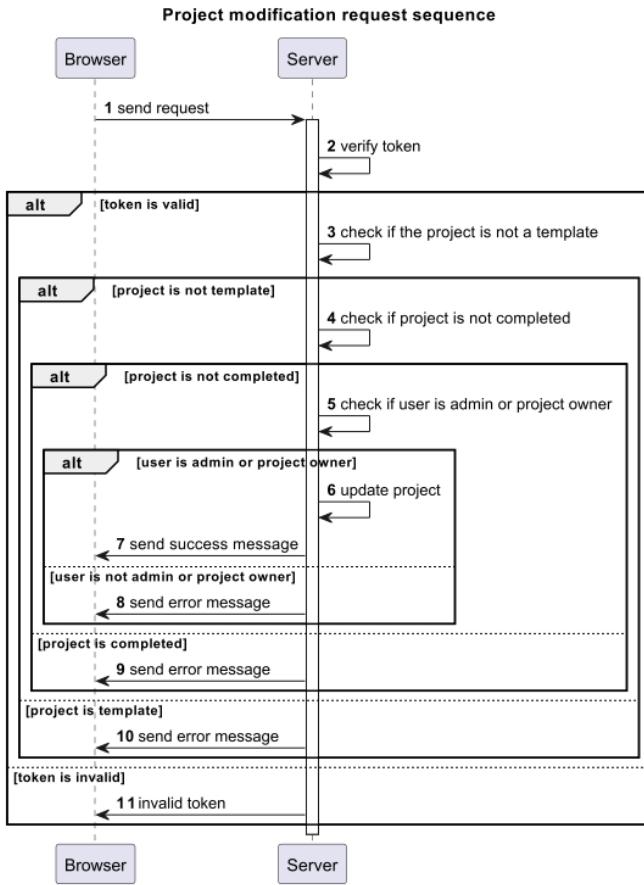
        return this.isAdminOrOwner(sender, project);
    }

    1 usage  ▲ Damiaan
    private canAccessTemplate(project: ProjectDto): boolean {
        return project.isTemplate;
    }

    1 usage  ▲ Damiaan
    private isProjectCompleted(project: ProjectDto): boolean {
        return project.isCompleted;
    }

    1 usage  ▲ Damiaan
    private isAdminOrOwner(sender: User, project: ProjectDto): boolean {
        return sender.role === UserRole.ADMIN || sender.id === project.user.id;
    }
}
```

Rysunek 4.25: Listing kodu źródłowego implementacji *AuthorGuard*.
Źródło: opracowanie własne



Rysunek 4.26: Diagram sekwencji dla sprawdzenia, czy użytkownik może modyfikować projekt.
 Źródło: opracowanie własne

4.5.4. Realizacja kolejkowania

W projekcie występuje również kolejkowanie procesów przy użyciu biblioteki *BullMQ*. Implementacja tego fragmentu kodu została przedstawiona na listingach zaprezentowanych na rysunkach 4.27 oraz 4.28. Można zauważać na nich użycie wzorca projektowego *Producer-Consumer*, w którym *UserService* pełni rolę *producer'a*, a *CodeActivatorConsumer* rolę *consumer'a*. Podczas rejestracji użytkownika zadanie do wykonania zostaje wrzucone do kolejki o nazwie *CODE_SEND_EMAIL*, które następnie trafia do *consumer'a* kolejki o tej samej nazwie, w której następuje wykonanie zadania, co pokazano na diagramie sekwencji na rysunku 4.29.

```

async register(command: UserRegisterCommand): Promise<Result> {
    if (await this.findOneByEmail(command.email) == null) {
        await this.userRepository.save(await mapUserCommandToUser(command));
        this.sendEmailQueue.add({ data: {
            email: command.email
        }};
        return new Result({ result: {affected: 1}})
    }
    throw new UserAlreadyExistsException(`User with email: ${command.email} already exists!`);
}

```

Rysunek 4.27: Listing kodu źródłowego implementacji kolejkowania, producer.
 Źródło: opracowanie własne

```

@Processor(CODE_SEND_EMAIL)
export class CodeActivatorConsumer {

    private readonly logger : Logger = new Logger(CodeActivatorConsumer.name);

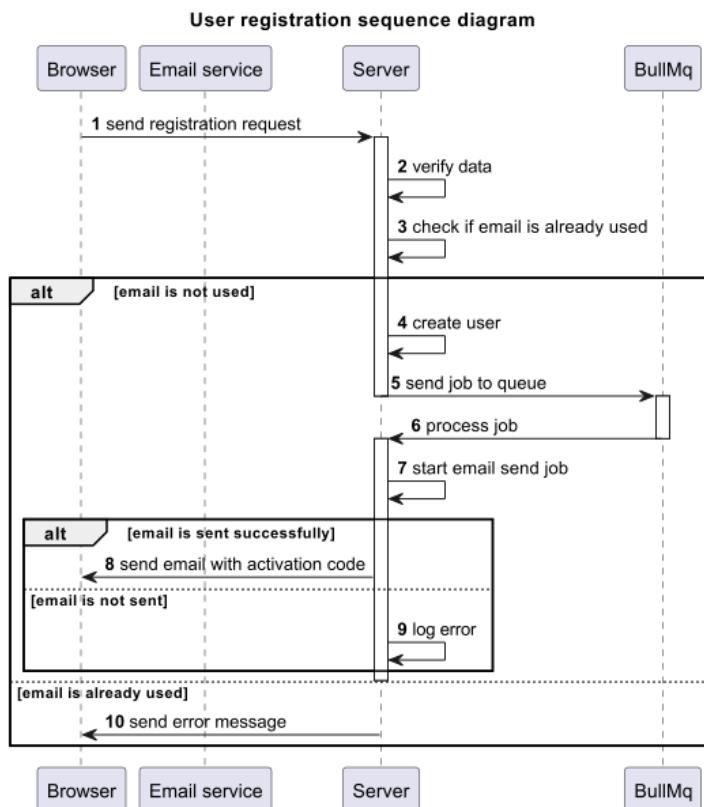
    no usages ▲ Damiaan
    CodiumAI: Test this method.
    constructor(
        private readonly mailerService: MailerService,
        @InjectRepository(CodeEntity)
        private readonly codeRepository: Repository<CodeEntity>
    ) {}

    no usages ▲ Damiaan
    CodiumAI: Test this method.
    @Process()
    async sendEmail(job: Job<any>) : Promise<void> {
        const code: string = this.generateCode(job.data.email);
        this.mailerService.sendMail({ sendMailOptions: {
            to: job.data.email,
            from: process.env.SMTP_FROM,
            subject: 'Code Activator',
            text: `Activation code: ${code}`
        }}).then(() => {
            this.logger.log(`Activation email sent to: ${job.data.email}`);
        }).catch((error) => {
            this.logger.error(error);
        });
        await this.codeRepository.save(new CodeEntity(code, job.data.email, new Date().toISOString()));
    }

    1 usage ▲ Damiaan
    private generateCode(email: string): string {
        const hash = crypto.createHash('algorithm: 'sha256');
        hash.update(email);
        return hash.digest('encoding: hex').slice(0, 6);
    }
}

```

Rysunek 4.28: Listing kodu źródłowego implementacji kolejkowania, consumer.
 Źródło: opracowanie własne



Rysunek 4.29: Diagram sekwencji dla rejestracji użytkownika.
 Źródło: opracowanie własne

4.5.5. 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 rysunkach 4.30 oraz 4.31. Polega ona na ekstrakcji *Bearer token'u* z nagłówka autoryzacyjnego danego zapytania oraz jego rozkodowaniu i sprawdzeniu, czy token jest poprawny i nie został zmodyfikowany [62]. W przypadku użycia *framework'a 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 wstrzygnięcie różnego typu strategii autoryzacji użytkowników do danych zasobów, dlatego do implementacji ich funkcjonalności zastosowano wzorzec projektowy strategii.

```
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {

    no usages  ↳ Damiaan
    constructor() {
        super({
            jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
            ignoreExpiration: false,
            secretOrKey: process.env.JWT_SECRET
        });
    }

    no usages  ↳ Damiaan
    async validate(payload: any): Promise<any> {
        return { ...payload.user };
    }
}
```

Rysunek 4.30: Listing kodu źródłowego implementacji strategii JWT.
Źródło: opracowanie własne

```
@Injectable()
export class JwtGuard extends AuthGuard(JWT) {}
```

Rysunek 4.31: Listing kodu źródłowego implementacji JWT guard.
Źródło: opracowanie własne

4.6. Testy

W projekcie w celu weryfikacji poprawności działania kodu oraz utrzymania ogólnej jakości oprogramowania zostały zastosowane testy integracyjne [66]. Odpowiadają one za weryfikację, czy poszczególne elementy składowe systemu w prawidłowy sposób ze sobą współpracują. Zastosowanie takiego podejścia pozwala sprawdzić, czy każdy fragment systemu spełnia określone wymagania. Dodatkowo pozwalają na symulację działania systemu, co za tym idzie dodając nową funkcjonalność w bezpieczniejszy sposób jesteśmy w stanie sprawdzić, czy zmiana jest kompatybilna wstecznie, co oznacza brak negatywnego wpływu na istniejące już funkcję systemu [66].

W kontekście aplikacji *GameFlow* testy integracyjne zostały wykonane przy użyciu kontenera testowego wraz z zastosowaniem *framework'u Jest*, który ułatwia testowanie oprogramowania [68, 67]. Dla każdego kontrolera aplikacji zostały utworzone pozytywne i negatywne przypadki testowe, które pozwalają weryfikować zachowanie systemu w różnych sytuacjach. W związku z tym testy aplikacji zostały podzielone na 13 zestawów testowych dających w sumie 88 przypadków testowych, których uruchomienie zostało przedstawione na rysunku 4.32.

```
PASS test/integration/users/user.spec.ts (8.065 s)
PASS test/integration/project/element.spec.ts
PASS test/integration/project/container.spec.ts
PASS test/integration/game/game.spec.ts
PASS test/integration/project/property.spec.ts
PASS test/integration/game/component.spec.ts
PASS test/integration/auth/auth.spec.ts (5.726 s)
PASS test/integration/game/tag.spec.ts
PASS test/integration/project/priority.spec.ts
PASS test/integration/project/status.spec.ts
PASS test/integration/image/image.spec.ts

Test Suites: 13 passed, 13 total
Tests:       88 passed, 88 total
Snapshots:   0 total
Time:        64.218 s
Ran all test suites.
```

Rysunek 4.32: Zrzut ekranu ze stanu odpalenia wszystkich testów dla aplikacji *GameFlow*.

Źródło: opracowane własne

4.7. Instalacje i konserwacja

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

- *docker compose*, wersja $\geq 2.15.1$ [64];
- *docker*, wersja $\geq 20.10.22$ [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 jednak zwrócić uwagę, że przez zastosowanie architektury klient-serwer warstwa widoku oraz warstwa logiki biznesowej będą korzystać z dwóch odrębnych instalacji niezbędnych modułów, a które to moduły 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 znajduje się kod źródłowy aplikacji. Wywołanie komendy „*docker-compose up*” spowoduje uruchomienie pliku konfiguracyjnego używanego przez *Docker Compose* przedstawionego na rysunku 4.33, w tym utworzenie 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 $\geq 7.0.11$ [28];
- środowisko uruchomieniowe *Node.js*, wersja $\geq 18.16.1$ [14];
- menadżer pakietów *npm*, wersja $\geq 9.8.0$ [65].

Naturalną koleją rzeczy są również ciągłe wychodzące aktualizacje 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 wydające polecenie

„npm update”. Wykonanie tej komendy spowoduje aktualizację wszystkich użytych pakietów w projekcie za pomocą menadżera pakietów *npm* [65].

```
version: "3.8"
volumes:
  postgres_data:
    driver: local
services:
  postgres:
    container_name: GameBoard-Postgres
    image: postgres
    env_file: .env
    environment:
      POSTGRES_DB: ${POSTGRES_SQL_DB}
      POSTGRES_USER: ${POSTGRES_SQL_USER}
      POSTGRES_PASSWORD: ${POSTGRES_SQL_PASS}
    ports:
      - ${POSTGRES_LOCAL_PORT}:${POSTGRES_DOCKER_PORT}
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./backend/init.sql:/docker-entrypoint-initdb.d/init.sql
    healthcheck:
      test: [ "CMD-SHELL", "pg_isready -U postgres" ]
      interval: 10s
      timeout: 5s
      retries: 3
  redis:
    container_name: GameBoard-Redis
    image: redis:7.0.11
    restart: unless-stopped
    env_file: .env
    ports:
      - ${REDIS_LOCAL_PORT}:${REDIS_DOCKER_PORT}
    healthcheck:
      test: [ "CMD", "redis-cli", "ping" ]
      interval: 1s
      timeout: 3s
      retries: 30
  backend:
    image: designer/backend
    env_file: .env
    build:
      context: backend
      dockerfile: Dockerfile
    ports:
      - "8080:3000"
    depends_on:
      postgres:
        condition: service_healthy
      redis:
        condition: service_healthy
    volumes:
      - ./designer/backend
      - /designer/backend/node_modules
  frontend:
    image: designer/frontend
    env_file: .env
    build:
      context: frontend
      dockerfile: Dockerfile
```

Rysunek 4.33: Zawartość pliku konfiguracyjnego *docker-compose.yaml*.
 Źródło: opracowanie własne

5. Podręcznik użytkowania aplikacji webowej GameFlow

5.1. Instrukcja

Aplikacja *GameFlow*, jak większość aplikacji korzysta z pewnych danych niejawnych przechowywanych jako zmienne środowiskowe. Jak zostało wspomniane w rozdziale 4.7, w głównym katalogu projektu znajduje 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.

```

1 #Backend config
2 BACKEND_PORT=3001
3
4 #MULTER config
5 MULTER_STORAGE_PATH=./upload
6
7 #MailerModule config
8 SMTP_HOST=sandbox.smtp.mailtrap.io
9 SMTP_PORT=465
10 SMTP_USER=a20cbf7a874b13
11 SMTP_PASS=157937455221c4
12 SMTP_FROM=GameFlow@gmail.com
13
14 #Frontend config
15 FRONTEND_PORT=3000
16
17 #JWT config
18 JWT_SECRET=example_secret
19 JWT_EXPIRATION_TIME=30d
20 JWT_REFRESH_EXPIRATION_TIME=7d
21
22 #Database config
23 POSTGRESQL_DB=gamedb
24 POSTGRESQL_USER=postgres
25 POSTGRESQL_PASS=postgres
26 POSTGRES_LOCAL_PORT=5333
27 POSTGRES_DOCKER_PORT=5432
28 POSTGRESQL_HOST=localhost
29
30 #Redis config
31 REDIS_HOST=localhost
32 REDIS_LOCAL_PORT=6379
33 REDIS_DOCKER_PORT=6379|
```

Rysunek 5.1: Zawartość pliku konfiguracyjnego .env.
 Źródło: opracowanie własne

5.2. Wprowadzenie do widoku użytkownika niezalogowanego

Warstwa prezentacji aplikacji *GameFlow* z widoku użytkownika niezalogowanego została stworzona z myślą o jak najszybszym i intuicyjnym dostępie odwiedzającego do treści wprowadzających do zawartości strony.

5.2.1. Pasek nawigacyjny

Zaprezentowany na rysunku 5.2 pasek nawigacyjny aplikacji *GameFlow* znajduje się w górnej części strony i zawsze podąża za użytkownikiem niezależnie od podstrony, do której przeszedł. Pomaga on użytkownikowi w szybki sposób przemieszczać się między zakładkami. Zawiera on następujące elementy:

- *Home* – stronę główną;
- *Games* – stronę z zaprezentowanymi planszówkami;
- *About* – stronę zawierającą informację o firmie;
- *Contact* – stronę z informacjami o firmie;
- *Wybór języków* – listę rozwijaną z dostępnymi tłumaczeniami strony;
- *Sign in* – przekierowanie do strony z logowaniem.

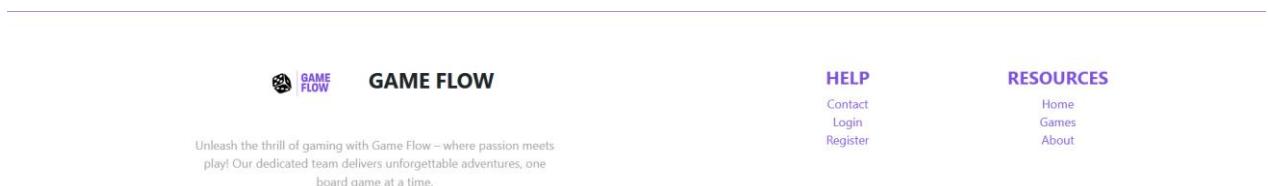


Rysunek 5.2: Pasek nawigacyjny aplikacji *GameFlow*.
Źródło: opracowanie własne

5.2.2. Stopka strony

Zaprezentowana na rysunku 5.3 stopka strony aplikacji *GameFlow* znajduje się na samym dole każdej z głównych podstron aplikacji. Zawiera ona logo oraz nazwę serwisu wraz z krótkim opisem oraz odnośnikami do następujących elementów:

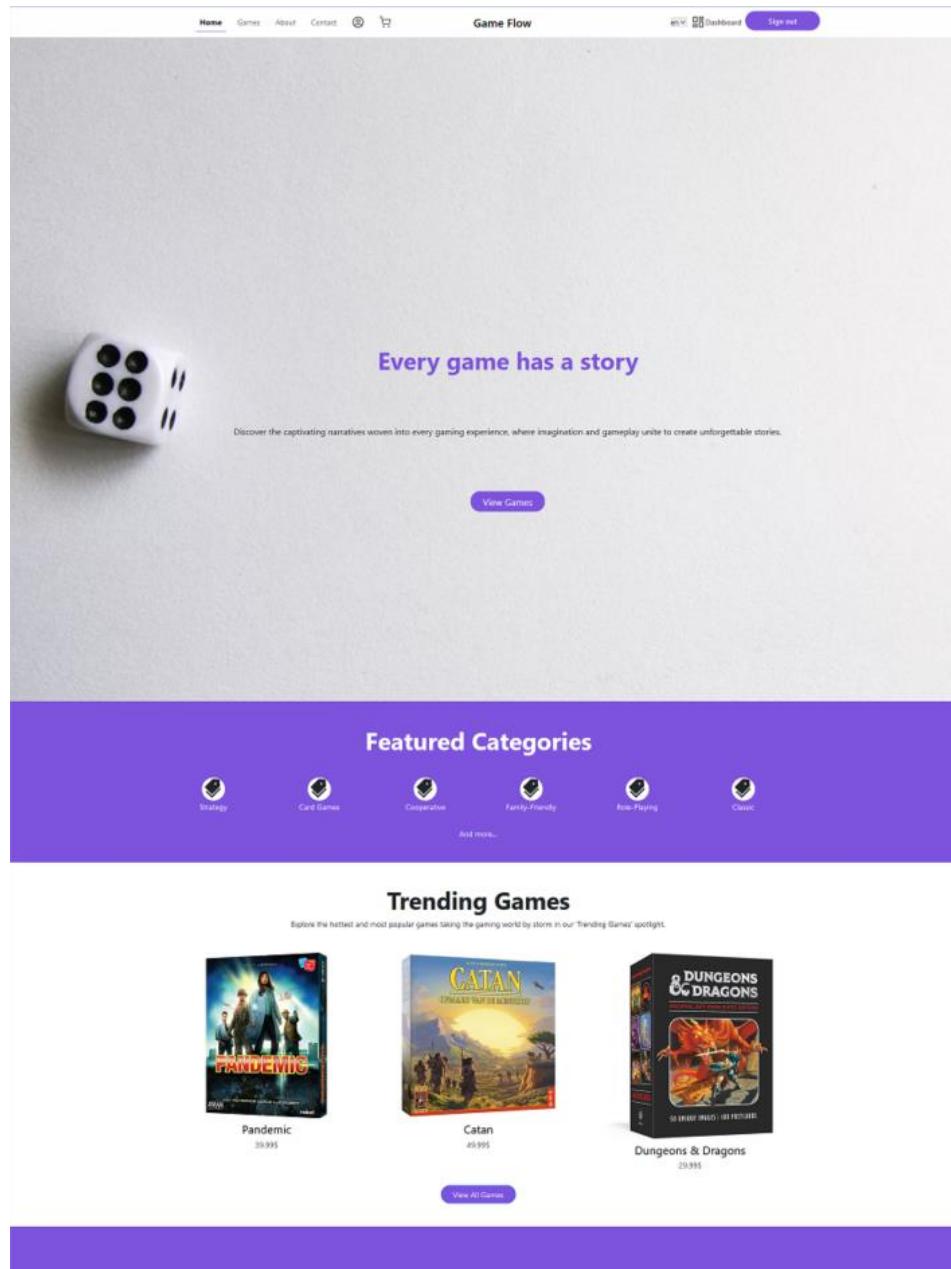
- *Contact* – przekierowanie do strony z informacjami o firmie;
- *Login* – przekierowanie do formularza z logowaniem;
- *Register* – przekierowanie do formularza rejestracji;
- *Home* – przekierowanie do strony głównej aplikacji;
- *Games* – przekierowanie do strony z zaprezentowanymi planszówkami;
- *About* – przekierowanie do strony zawierającej informację o firmie.



Rysunek 5.3: Stopka strony aplikacji *GameFlow*.
Źródło: opracowanie własne

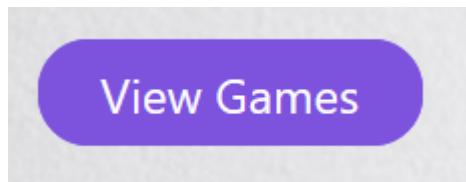
5.2.3. „Home” – Strona główna

Zaprezentowana na rysunku 5.4 strona główna aplikacji *GameFlow* jest pierwszą rzeczą ukazującą się po wejściu do serwisu. Została ona zbudowana w ten sposób, by jak najszybciej wprowadzić odwiedzającego w tematykę oferowanych przez firmę produktów, w tym przypadku gier planszowych. Użytkownik jest wprowadzany w daną tematykę przez tło z kostką do gry oraz sentencję, które naprowadzają potencjalnego odwiedzającego na branżę gier planszowych.



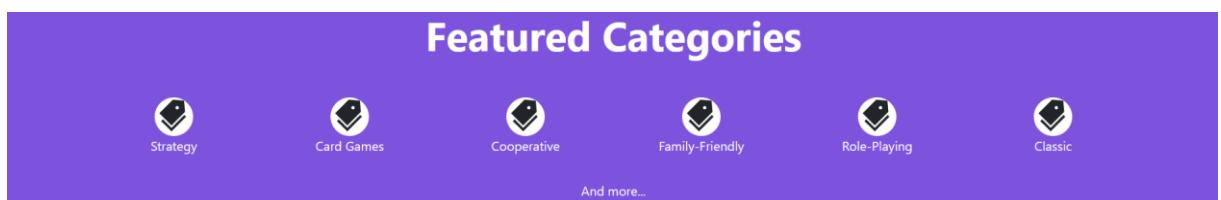
Rysunek 5.4: Strona główna aplikacji *GameFlow*.
Źródło: opracowanie własne

Na środku strony widoczny jest również przycisk *View Games* rysunek 5.5, który przekierowuje na podstronę *Games* zawierającą oferowane gry planszowe. Przycisk ten został umieszczony na środku strony w widocznym miejscu w celu jak najszybszego umożliwienia użytkownikowi przejścia do oferowanych produktów.



Rysunek 5.5: Przycisk przekierowujący do strony z produktami.
Źródło: opracowanie własne

W środkowej sekcji strony poniżej przycisku *View Games* znajduje się sześć pierwszych kategorii, po których użytkownik może wyszukiwać gry – rysunek 5.6. Zostały one wylistowane również w celu szybszej prezentacji odwiedzającemu zawartości strony.



Rysunek 5.6: Sekcja kategorii prezentująca potencjalne kategorie w aplikacji GameFlow.
Źródło: opracowanie własne

W dolnej sekcji strony znajdują się popularne produkty, czyli produkty które zawierają największą liczbę zamówień – rysunek 5.7. Zaprezentowane zostały one jako sugestia produktów, które sprzedają się najlepiej, a co za tym idzie mają największą szansę na przyciągnięcie uwagi odwiedzającego.

Trending Games

Explore the hottest and most popular games taking the gaming world by storm in our 'Trending Games' spotlight.



Rysunek 5.7: Sekcja popularnych produktów.
Źródło: opracowanie własne

5.2.4. „Games” – Strona z produktami

Zaprezentowana na rysunku 5.8 strona przedstawia podstronę *Games* aplikacji *GameFlow*. Głównym zadaniem tego elementu serwisu jest prezentowanie dostępnych produktów na stronicowanej liście produktów. Użytkownik za jej pomocą może przeglądać produkty dostępne w serwisie zmieniając strony za pośrednictwem guzików dostępnych w dolnej części strony. Warto również zaznaczyć, że odwiedzający stronę ma również możliwość filtrowanie wyników (rysunek 5.9) po nazwie oraz tagach, czyli kategoriach przypisanych do danych gier planszowych. Użytkownik najeżdżając na dany produkt, rysunek 5.10, ma również możliwość zobaczenia dokładniejszych informacji o nim - rysunek 5.11. Informację jakie użytkownik może uzyskać po kliknięciu w daną grę, to więcej zdjęć, kategorię oraz opis gry.



Rysunek 5.8: Strona z produktami dostępnymi w aplikacji GameFlow.
 Źródło: opracowanie własne

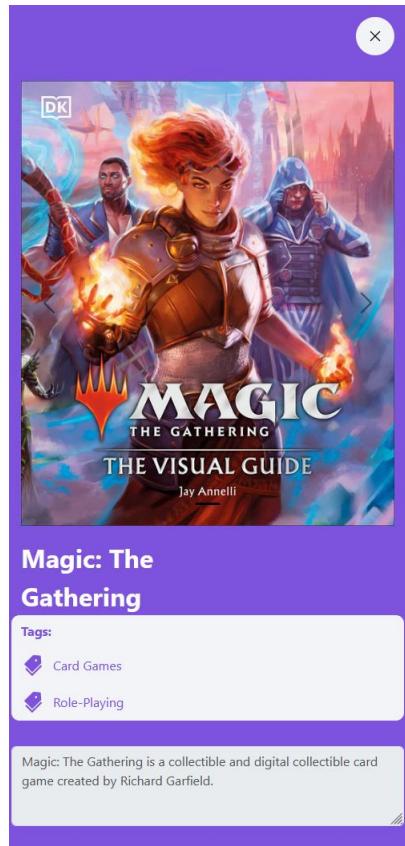


Rysunek 5.9: Prezentacja filtrowania produktów w aplikacji GameFlow.
 Źródło: opracowanie własne



Rysunek 5.10: Prezentacja przycisku informacji po najechaniu na dany produkt.

Źródło: opracowanie własne



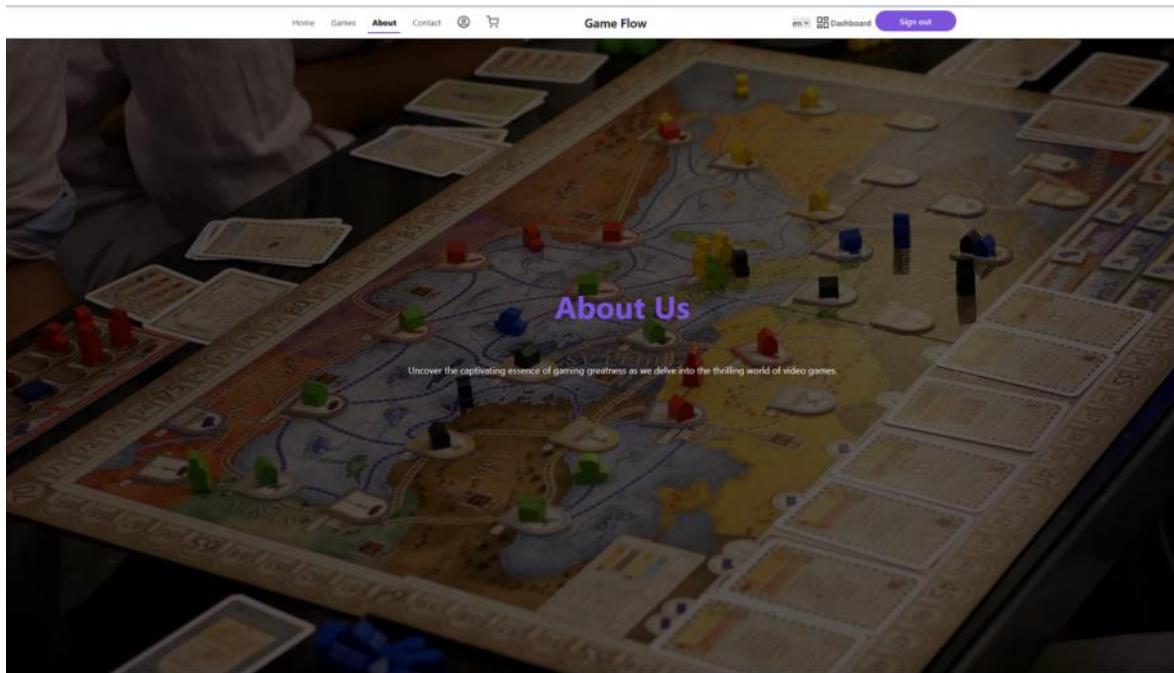
Rysunek 5.11: Prezentacja informacji o wybranym produkcie.

Źródło: opracowanie własne

5.2.5. „About” – Strona zawierająca informacje o firmie

Zaprezentowana na rysunku 5.12 strona zawiera informacje, którymi przedsiębiorca chce się podzielić z potencjalnymi klientami. Przedsiębiorca ma możliwość ustawienia tam opisu swojej działalności oraz celu, który kryje się za jego pomysłem na biznes. Warto zaznaczyć, że opis *About us* oraz *Our Mission* mogą być konfigurowane z poziomu panelu administracyjnego (rozdział 5.5.) umożliwiając tym samym administratorowi systemu na łatwą zmianę tekstu dostosowując się do obecnie panujących trendów oraz zmieniających się celów firmy.

Niniejsza podstrona ma skłonić odwiedzającego do między innymi szerszego zapoznania się z działalnością i zgłębienia, czym jest dane przedsiębiorstwo oraz jakie są jego kluczowe wartości. Taki zabieg może dodatkowo skłonić klienta do potencjalnego zakupu wpływając na jego przekonanie, że dana marka nie oferuje swoich produktów jako zwykłego towaru na sprzedaż, ale kryje za sobą również cel i misję, którą przelewa w każde swoje dzieło. Sekcja ta również pozwala na zamieszczenie potencjalnego celu, do którego może dążyć dana działalność, a co pozwala potencjalnemu klientowi lepiej zrozumieć działania danego przedsiębiorstwa.



Our Mission

Embark on our inspiring journey as we strive to fulfill our mission - pushing the boundaries of innovation and delivering immersive gaming experiences that captivate and delight players worldwide.



GAME FLOW

Unleash the thrill of gaming with Game Flow - where passion meets play! Our dedicated team delivers unforgettable adventures, one board game at a time.

HELP

Contact
Login
Register

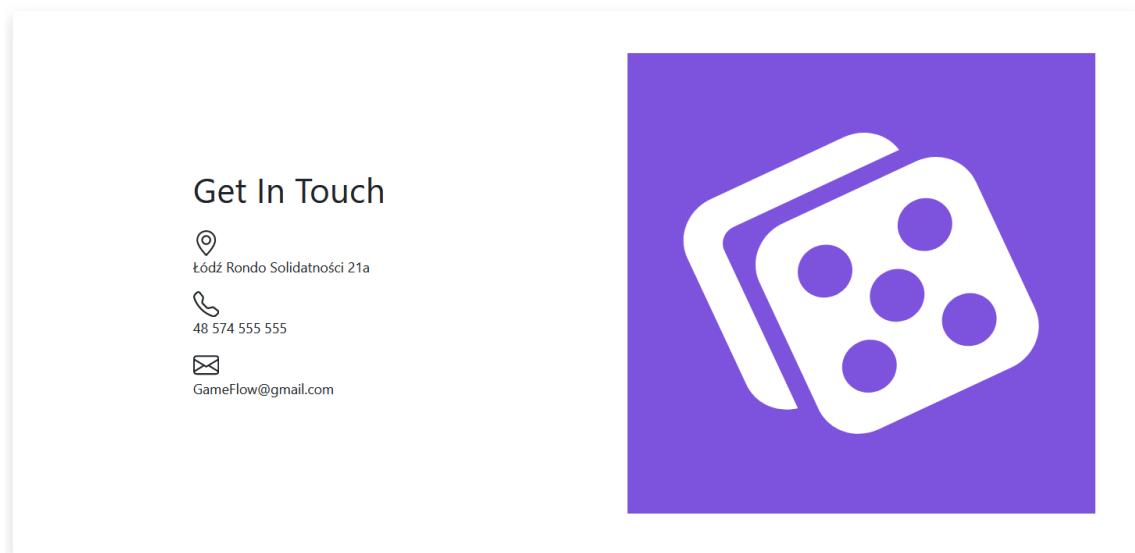
RESOURCES

Home
Games
About

Rysunek 5.12: Strona przedstawiająca informację o firmie.
Źródło: opracowanie własne

5.2.6. „Contact” – Strona zawierająca informacje kontaktowe

Zaprezentowana na rysunku 5.13 strona zawiera informacje kontaktowe do danego przedsiębiorstwa. Strona została dodana ze względu na zapewnienie potencjalnemu użytkownikowi szybkiego dostępu do danych kontaktowych, takich jak adres, numer telefonu oraz adres email. Warto zaznaczyć, że każda z zaprezentowanych na tej stronie informacja jest konfigurowalna z poziomu panelu administratora (rozdział 5.5.) zapewniając tym samym administratorowi systemu łatwą możliwość modyfikacji danych w razie potrzeby uaktualnienia jakiejkolwiek z informacji.



Rysunek 5.13: Strona zawierająca informacje kontaktowe aplikacji GameFlow.
Źródło: opracowanie własne

5.2.7. „Sign in” – Formularz logowania

Zaprezentowany na rysunku 5.14 formularz przedstawia logowanie do aplikacji *GameFlow*. Użytkownik z tego poziomu może się zalogować podając swój email oraz hasło, a następnie klikając przycisk *Sign in*. Z tego poziomu użytkownik ma również możliwość stworzenia nowego konta za pośrednictwem hiperłącza dostępnego pod napisem *Create an account*.

A screenshot of a sign-in form titled "Sign In". It includes fields for "Email" (containing "admin2@gmail.com") and "Password" (containing a redacted password). A "Sign In" button is at the bottom. To the right of the form is a decorative illustration of a person standing in front of a window, holding a ball. There is also a small "X" icon in the top right corner of the form area.

Rysunek 5.14: Formularz logowania aplikacji GameFlow.
Źródło: opracowanie własne

5.2.8. „Create an account” – Formularz rejestracji

Zaprezentowany na rysunku 5.15 formularz przedstawia rejestrację nowego użytkownika w aplikacji *GameFlow*. Odwiedzający stronę za pośrednictwem tego widoku może stworzyć nowe konto podając takie dane jak nazwę użytkownika, numer telefonu, adres email oraz hasło. Następnie klikając przycisk *Sign up*, znajdujący się w dolnej części formularza, przesyła dane do serwera tworząc tym samym konto. Dane oczywiście są walidowane pod kątem prawidłowości. Jednak w celu aktywacji konta użytkownik musi potwierdzić konto za pośrednictwem kodu aktywacyjnego, który w chwili utworzenia konta zostaje wysłany na adres email podany przez użytkownika. Formularz aktywacji konta został przedstawiony na rysunku 5.16, pojawia się on w momencie poprawnej rejestracji, jednak użytkownik może aktywować konto również później za pośrednictwem hiperłącza dostępnego pod napisem *Activate*.

The screenshot shows a registration form titled "Create An Account". It includes fields for Username (admin2@gmail.com), Phone number (+48 321 321 321), Email (example@mail.com), and Password (represented by a redacted yellow bar). There are links for "Sign In" and "Activate account? Activate". A large smartphone icon on the right displays a user profile. A small figure of a person stands next to the phone. A purple "Sign Up" button is at the bottom.

Rysunek 5.15: Formularz rejestracji do systemu *GameFlow*.
Źródło: opracowanie własne

The screenshot shows an activation form titled "Enter activation code". It has a field labeled "Enter Code:" with a placeholder "Enter Code" and a blue "Activate" button below it.

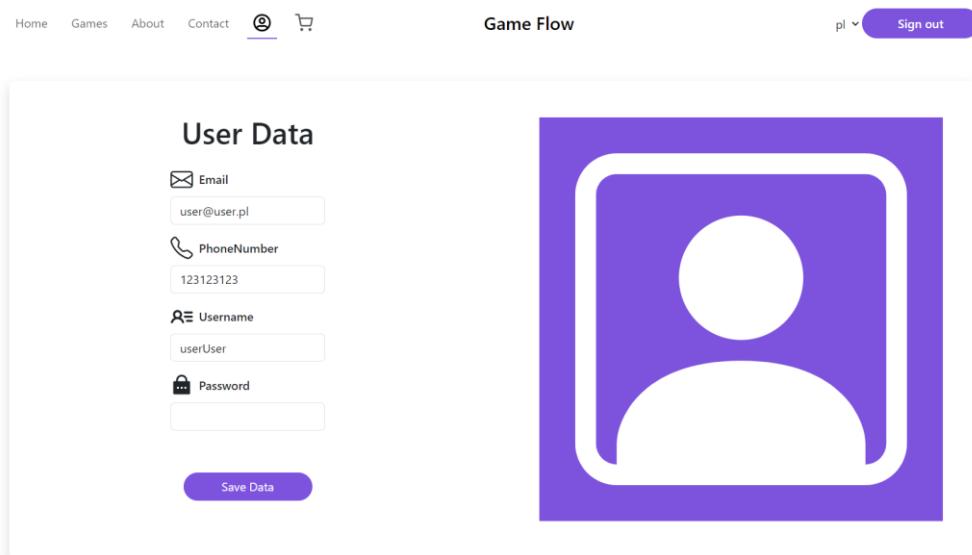
Rysunek 5.16: Formularz aktywacji konta w aplikacji *GameFlow*.
Źródło: opracowanie własne

5.3. Wprowadzenie do widoku klienta

Użytkownik po zalogowaniu się, czyli potwierdzeniu swojej tożsamości zyskuje kilka dodatkowych widoków i możliwości, które ze względu na różne aspekty nie powinny być widoczne lub możliwe do uzyskania przez użytkownika niezalogowanego.

5.3.1. „User Data” – Dane użytkownika

Zaprezentowana na rysunku 5.17 strona zawiera aktualne informacje o zalogowanym użytkowniku. Dodatkowo udostępnia możliwość aktualizacji takich danych jak adres email, numer telefonu, nazwę użytkownika oraz hasło. Wymienione parametry są danymi, które z biegiem czasu mogą się zmieniać, a użytkownik może tracić do nich dostęp, tak jak w przypadku numeru telefonu, który wraz ze zmianą operatora zazwyczaj przepada. Udostępnienie takiej funkcjonalności jest kluczowe w kontekście nie wymuszania na użytkowniku ciągłego zakładania nowych kont w przypadku zmiany danych oraz trzymania wszystkich swoich personalnych informacji w obrębie jednego konta. Po zmodyfikowaniu któreś z wymienionych wartości, tylko zaktualizowane wartości są wysyłane do aktualizacji oraz są walidowane pod kątem prawidłowości danych.



Rysunek 5.17: Strona udostępniająca możliwość aktualizacji swoich danych w aplikacji GameFlow.
 Źródło: opracowanie własne

5.3.2. „Orders” – Złożone zamówienia

Zaprezentowana na rysunku 5.18 strona zawiera wszystkie złożone przez zalogowanego użytkownika zamówienia. Za pośrednictwem tego widoku użytkownik może przeglądać wszystkie złożone zamówienia na produkcie w serwisie. Widok ten posiada bardzo ważne pole, którym jest status zamówienia, za pośrednictwem którego można sprawdzać postępy w wykonywaniu naszego zamówienia. Każdy wiersz posiada również przycisk *info*, wyświetlający formularz zaprezentowany na rysunku 5.19, z pomocą którego można sprawdzić dodatkowe

szczegóły dotyczące złożonego zamówienia. Dodatkowo każdy wiersz posiada przycisk *Edit*, pokazujący formularz zaprezentowany na rysunku 5.20 do edycji zamówienia, dodany w celu ewentualnej poprawy szczegółów dotyczących danego zamówienia. Warto jednak zauważyć, że możliwość edycji zamówienia przez użytkownika istnieje tylko do momentu, kiedy zamówienie posiada status *PENDING* w momencie potwierdzenia zamówienia przez pracownika, użytkownik traci możliwość zmiany statusu.

ORDERED GAME	STATUS	DETAILS	EDIT
Catan	PENDING	Info	Edit
Pandemic	PENDING	Info	Edit
Magic: The Gathering	PENDING	Info	Edit

Rysunek 5.18: Strona zawierająca złożone zamówienia w aplikacji GameFlow.
Źródło: opracowanie własne

Order ID: 6

Status: PENDING
 Phone Number: 311131131
 Email: johnwick@gmail.com
 Address: Street of london
 City: London

Games:	Price:
Catan	49.99 \$

Please order with blue inserts

Rysunek 5.19: Formularz zawierający informacje o złożonym zamówieniu.
Źródło: opracowanie własne

The form is titled "Edit order ID: 6". It includes fields for Phone Number, Address, Email, City, and a large Description area. The Description field contains the text "Please order with blue inserts". A "Save changes" button is located at the bottom.

Rysunek 5.20: Formularz umożliwiający możliwość edycji zamówienia.
 Źródło: opracowanie własne

5.3.3. „Order” – Złożenie zamówienia na produkt

Jako użytkownik zalogowany również dostajemy możliwość złożenia zamówienia za pomocą przycisku *Order*, który pojawia się po najechaniu na dany produkt rysunek 5.21. Po kliknięciu wskazanego przycisku ukaże się formularz złożenia zamówienia na dany produkt zaprezentowany na rysunku 5.22. Posiada on wylistowany produkt, którego dotyczy zamówienie oraz szereg pól tekstowych służących do uzupełnienia informacji kluczowych, jeżeli chodzi o złożenie zamówienia takich jak imię oraz nazwisko osoby, na którą składa się zamówienie, miasto, adres zamieszkania, numer kontaktowy, adres email oraz opis zamówienia, w którym to klient może zamieścić szczegóły oraz uwagi, które chciałby, żeby zostały uwzględnione w zamówieniu. Warto jednak zauważyć, że cena zamówienia jest tylko ceną sugerowaną, ostateczne koszty mogą ulec zwiększeniu w zależności od wspomnianej personalizacji, ostateczną cenę ustala pracownik. Dodatkowo warto podkreślić, pojedyncze zamówienie może być złożone tylko na jedną grę. Zostało to stworzone w taki sposób, gdyż wytworzenie każdego z produktów pochłania dłuższy okres czasu i nie jest obiektem handlu hurtowego. Przekłada się to tym samym na pewną trudność w realizacji wielu gier naraz, będąc procesem trudnym w realizacji, przez co w praktyce wytworzenie kolejnych gier przyjmuje raczej formę pracy synchronicznej, czyli realizacji gier po kolei.



Rysunek 5.21: Prezentacja przycisku informacji oraz zamówienia po najechaniu na dany produkt.
Źródło opracowanie własne

Name	Count	Price
Magic: The Gathering	1	19.99 \$

firstname:

lastname:

City:

address:

PhoneNumber:

Email:

Description:

Submit Order

Rysunek 5.22: Formularz złożenia zamówienia na dany produkt.
Źródło: opracowanie własne

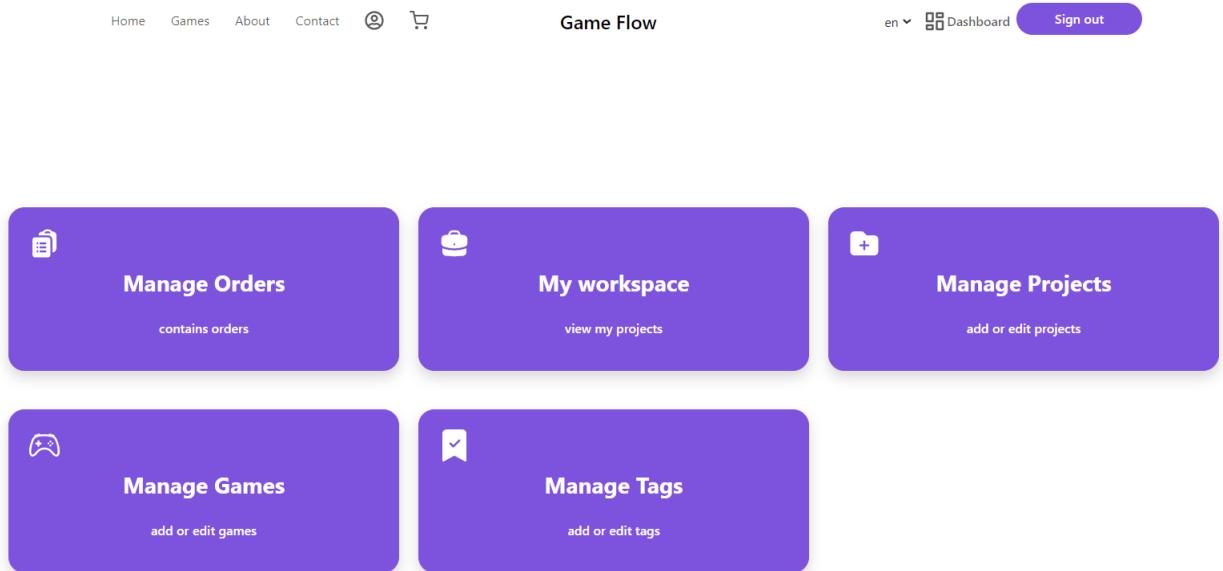
5.4. Wprowadzenie do widoku pracownika

Po zalogowaniu się na konto z przypisaną rolą pracownik *Employee*, użytkownik uzyskuje dodatkowy przycisk na pasku nawigacyjnym Panel lub też *DashBoard*, przedstawiony na rysunku 5.23. Za pośrednictwem tego przycisku pracownik serwisu może przejść do widoku pracownika przedstawionego na rysunku 5.24. Widok pracownika umożliwia wykonywanie pewnego rodzaju interakcji, takich jak zarządzanie zamówieniami, projektami grami oraz tagami. Dodatkowo na tym ekranie znajduje się kafelek przekierowujący do przestrzeni

roboczej danego pracownika, czyli miejsca gdzie znajdują się wszystkie bieżąco wykonywane projekty danego pracownika.



Rysunek 5.23: Pasek nawigacyjny aplikacji GameFlow po zalogowaniu się jako pracownik.
 Źródło: opracowanie własne



Rysunek 5.24: Widok panelu pracownika aplikacji GameFlow.
 Źródło: opracowanie własne

5.4.1. „Manage Orders” – Zarządzaj zamówieniami

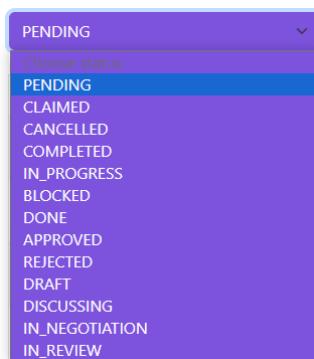
Kafelek zarządzaj zamówieniami jest odnośnikiem przenoszącym użytkownika do podstrony zaprezentowanej na rysunku 5.25. Za pośrednictwem tego widoku pracownicy aplikacji mogą wykonywać różnego typu akcję na złożonych zamówieniach na produkcie. Wszystkie zamówienia widoczne są na stronicowanej liście stanowiącej główny element tego komponentu. Użytkownik może oczywiście również filtrować zamówienia po statusie - rysunek 5.26.

Orders

ID	ORDERED GAME	SUBMIT DATE	LAST UPDATE	STATUS	WORKER	EDIT	INFO	CLAIM
4	Pandemic	2023-10-18	2023-10-18	PENDING	Not claimed	<button>Edit</button>	<button>Info</button>	<button>Claim</button>
5	Pandemic	2023-11-29	2023-11-29	PENDING	Not claimed	<button>Edit</button>	<button>Info</button>	<button>Claim</button>
6	Catan	2023-12-02	2023-12-02	PENDING	Not claimed	<button>Edit</button>	<button>Info</button>	<button>Claim</button>
7	Pandemic	2023-12-02	2023-12-02	PENDING	Not claimed	<button>Edit</button>	<button>Info</button>	<button>Claim</button>
8	Magic: The Gathering	2023-12-02	2023-12-02	PENDING	Not claimed	<button>Edit</button>	<button>Info</button>	<button>Claim</button>

previous 1 next

Rysunek 5.25: Widok edycji zamówienia aplikacji GameFlow.
Źródło: opracowanie własne



Rysunek 5.26: Lista zawierająca możliwe statusy zamówień.
Źródło: opracowanie własne

Dodatkowo za pośrednictwem przycisków znajdujących się obok każdego złożonego zamówienia pracownik może:

Edytować zamówienie – pracownik za pośrednictwem przycisku *Edit*, może otworzyć formularz zaprezentowany na rysunku 5.27. Pozwala on na edycję danych danego zamówienia poprzez edycję informacji wprowadzonych przez klienta, ale również tych przypisywanych automatycznie przez pobranie ich z gry takich jak cena oraz waluta. Pracownik może również zmienić z tego widoku status zamówienia oraz przypisać zamówienie do któregoś z innych pracowników.

Edit Order

firstname aaaaa	lastname aaaaa
Email aaaaa@gmail.com	PhoneNumber 123123123
address aaaaa	City aaaaa
Description aaaaa	
Price 39,99 PLN	
Status PENDING	Assigned to None
Save Data	

Rysunek 5.27: Formularz edycji zamówienia w aplikacji GameFlow.
 Źródło: opracowanie własne

Przeglądać informacje o zamówieniu – pracownik za pośrednictwem przycisku *info* może otworzyć formularz zaprezentowany na rysunku 5.28. Pozwala on na przejrzenie informacji podanych przez klienta przy składaniu zamówienia, jak również jakiej gry dotyczy złożone zamówienie, kim jest klient, który złożył zamówienie oraz kto obecnie zajmuje się danym zamówieniem.

Pandemic Order: 4

Price: 39.99\$	Description: aaaa			
Submit Date: 2023-10-18	User Data:			
Last Update: 2023-10-18	firstname: aaaa			
Status: PENDING	lastname: aaaa			
address aaaaa	City aaaaa	Email aaaaa@gmail.com	PhoneNumber 123123123	Customer Info
Assigned to: None				
Game Details				

Rysunek 5.28: Formularz informacji o zamówieniu w aplikacji GameFlow.
 Źródło: opracowanie własne

Przypisywać zamówienie do siebie – pracownik za pośrednictwem przycisku *claim*, może przypisać zamówienie do siebie. Ta funkcjonalność pozwala w szybki sposób na ustawienie aktualnie zalogowanego pracownika jako osoby, która będzie wykonywać dane zamówienie.

5.4.2. „Manage Games” – Zarządzaj grami

Kafelek zarządzaj grami jest odnośnikiem przenoszącym użytkownika do podstrony zaprezentowanej na rysunku 5.29. Za pośrednictwem tego widoku pracownicy aplikacji mogą wykonywać różnego typu operacje na produktach, czyli grach w systemie. Wszystkie gry widoczne są na stronicowanej liście.

ID	TITLE	PRICE	DETAILS	EDIT
1	Catan	49.99 \$	<button>Info</button>	<button>Edit</button>
2	Pandemic	39.99 \$	<button>Info</button>	<button>Edit</button>
3	Magic: The Gathering	19.99 \$	<button>Info</button>	<button>Edit</button>
4	Dungeons & Dragons	29.99 \$	<button>Info</button>	<button>Edit</button>

Rysunek 5.29: Widok zarządzania produktami w aplikacji GameFlow.
Źródło: opracowanie własne

Za pośrednictwem przycisków znajdujących się na tym widoku oraz przy każdej grze w liście, pracownik może:

Dodać nową grę – pracownik za pośrednictwem przycisku *add new game*, może dodać nowy produkt do listy dostępnych produktów w systemie. Przycisk ten otwiera formularz tworzenia nowej gry przedstawiony na rysunku 5.30. Za jego pomocą pracownik może wprowadzić informacje o danej planszówce, takie jak tytuł, opis, cena, data premiery, waluta, kategorie, zdjęcia oraz komponenty, czyli elementy, które składają się na daną grę.

New Game

Amazing arcade game

11

12.02.2025

EUR

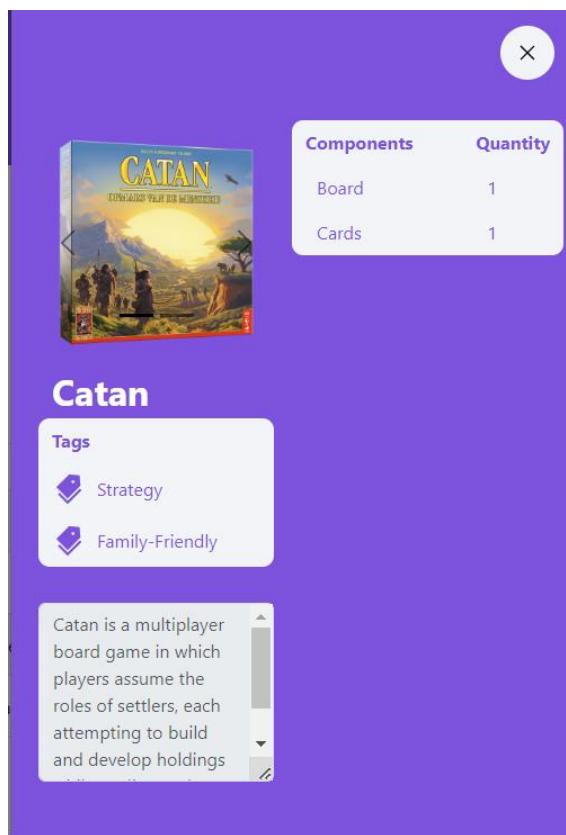
Family-Friendly Card Games

Create

Name	Quantity
example_component	5

Rysunek 5.30: Formularz dodania nowej gry w aplikacji GameFlow.
Źródło: opracowanie własne

Przeglądać informacje o danej grze – pracownik za pośrednictwem przycisku *info*, może otworzyć formularz zaprezentowany na rysunku 5.31. Pozwala on na przejrzenie informacji o danej grze, takich jak tytuł, opis, kategorie, zdjęcia oraz elementy.



Rysunek 5.31: Formularz informacji o grze w aplikacji GameFlow.
Źródło: opracowanie własne

Edytować grę – pracownik za pośrednictwem przycisku *Edit*, może otworzyć formularz zaprezentowany na rysunku 5.32. Pozwala on na edycję informacji o danej grze w tym tytule, ceny, daty premiery, waluty, kategorie, zdjęć oraz elementów danej gry.

Name	Quantity	Edit	Remove
Board	1	<button>Edit</button>	<button>-</button>
Cards	1	<button>Edit</button>	<button>-</button>

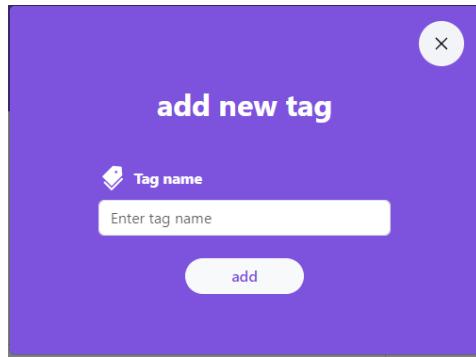
Rysunek 5.32: Formularz edycji gry w aplikacji GameFlow.
 Źródło: opracowanie własne

5.4.3. „Manage Tags” – Zarządzaj tagami

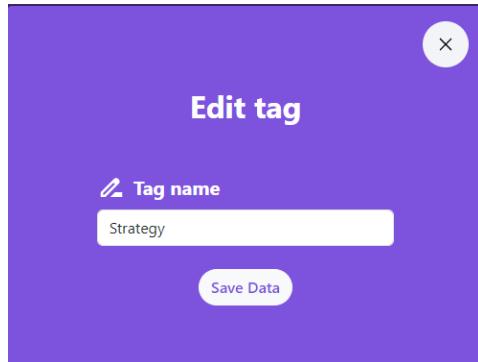
Kafelek zarządzaj tagami jest odnośnikiem przenoszącym użytkownika do podstrony zaprezentowanej na rysunku 5.33. Za pośrednictwem tego widoku pracownicy aplikacji mogą dodawać nowe kategorie, co przedstawia rysunek 5.34, które możliwe są do przypisania do gier, jak i edytować obecnie już istniejące - rysunek 5.35.

ID	NAME	EDIT
1	Strategy	<button>Edit</button>
2	Card Games	<button>Edit</button>
3	Cooperative	<button>Edit</button>
4	Family-Friendly	<button>Edit</button>
5	Role-Playing	<button>Edit</button>
6	Classic	<button>Edit</button>

Rysunek 5.33: Widok zarządzania tagami w aplikacji GameFlow.
 Źródło: opracowanie własne



Rysunek 5.34: Formularz dodania nowego tagu w aplikacji GameFlow.
Źródło: opracowanie własne



Rysunek 5.35: Formularz edycji tagu w aplikacji GameFlow.
Źródło: opracowanie własne

5.4.4. „Manage Projects” – Zarządzaj projektami

Kafelek zarządzaj projektami jest odnośnikiem przenoszącym użytkownika do podstrony zaprezentowanej na rysunku 5.36. Za pośrednictwem tego widoku pracownicy aplikacji mogą wykonywać różnego typu operacje na projektach, czyli schematach w oparciu o które można tworzyć gry. Wszystkie projekty widoczne są na stronicowanej liście. Użytkownik może oczywiście również filtrować projekty po zadanych kategoriach - rysunek 5.37, takich jak:

- typ gry, którego dotyczą projekty;
- przypisany pracownik, który wykonuje dany projekt;
- szablon projektu, czy projekt jest realnie wykonywalnym schematem;
- status projektu, czy projekt jest skończony, czy dalej trwa nad nim praca.

The screenshot shows the 'Projects' screen in the GameFlow application. At the top right is a close button (X). Below it is the title 'Projects'. In the center is a table with the following data:

ID	NAME	BOX	CONTAINERS	ELEMENTS	TEMPLATE	COMPLETED	INFO	EDIT
5	catan project	catanBox	1	1	No	No	<button>Info</button>	<button>Edit</button>
6	dnd project	dndBox	1	1	No	No	<button>Info</button>	<button>Edit</button>
7	dnd project	dndBox	1	1	No	No	<button>Info</button>	<button>Edit</button>
8	dnd project	dndBox	1	1	No	No	<button>Info</button>	<button>Edit</button>
9	pandemic project	pandemicBox	1	1	No	Yes	<button>Info</button>	<button>Edit</button>
10	catan project	catanBox	1	1	No	No	<button>Info</button>	<button>Edit</button>

At the bottom of the table are navigation buttons: 'previous', '1', and 'next'.

At the top of the screen are four filter dropdowns:

- Choose Game
- Choose worker
- Choose template status
- Choose complete status

A 'Add new project' button is located above the filters.

Rysunek 5.36: Widok zarządzania projektami w aplikacji GameFlow.
 Źródło: opracowanie własne

This screenshot shows the filter dropdowns from Rysunek 5.36. The dropdowns are expanded to show their options:

- Choose Game: None, Catan, Pandemic, Magic: The Gathering, Dungeons & Dragons
- Choose worker: None, admin@gmail.com, admin2@gmail.com, admin3@gmail.com, employee@gmail.com, user@user.pl
- Choose template status: Choose template status, Yes, No
- Choose complete status: Choose complete status, Yes, No

Rysunek 5.37: Kategorie filtrowania dla projektów w aplikacji GameFlow.
 Źródło: opracowanie własne

Za pośrednictwem przycisków znajdujących się na tym widoku oraz przy każdym projekcie w liście pracownik może:

Przeglądać informacje o danym projekcie – pracownik za pośrednictwem przycisku *info*, może otworzyć formularz zaprezentowany na rysunku 5.38. Pozwala on na przejrzenie opisu, notatek, statusu o danego projektu oraz elementów, które wchodzą w jego skład, takich jak opakowanie, przedstawione na rysunku 5.39, elementy, przedstawione na rysunku 5.40, kontenery przedstawione na rysunku 5.41.

Project: catan project

Description: Project for catan game	Container: <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th>CONTAINER</th> <th>QUANTITY</th> <th>INFO</th> </tr> </thead> <tbody> <tr> <td>Container 1</td> <td>2</td> <td>Show</td> </tr> </tbody> </table>	CONTAINER	QUANTITY	INFO	Container 1	2	Show	Elements: <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th>ELEMENT</th> <th>QUANTITY</th> <th>INFO</th> </tr> </thead> <tbody> <tr> <td>Dice</td> <td>3</td> <td>Show</td> </tr> </tbody> </table>	ELEMENT	QUANTITY	INFO	Dice	3	Show
CONTAINER	QUANTITY	INFO												
Container 1	2	Show												
ELEMENT	QUANTITY	INFO												
Dice	3	Show												
Notes: "catan project notes"														
Template: No Completed: No Games: GAME Catan														
box images														

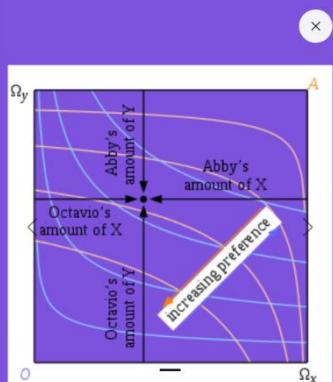
Rysunek 5.38: Formularz zawierający informację o wybranym projekcie.
 Źródło: opracowanie własne

OLD KENT ROAD

House	£10
House with color set	£40
House with	£10
House with	£30
House with	£60
House with	£160
House with	£250

Dice

Quantity	3
Status	TODO
Priority	C
Name	Die



catanBox

Status TODO	
Priority B	
Name	catan box desc

Rysunek 5.39: Formularz informacji o danym pudełku.
 Źródło: opracowanie własne

Rysunek 5.40: Formularz informacji o danym elemencie.
 Źródło: opracowanie własne

Container: Container 1

Description:		Elements:			Properties	
		ELEMENT	QUANTITY	INFO	NAME	VALUE
Box for cards		Cards	2	Info	height	5
Notes: "empty"					width	5
Status: TODO Priority: C					length	5
Images						

Rysunek 5.41: Formularz informacji o danym pojemniku.
Źródło: opracowanie własne

Edytować projekt – pracownik za pośrednictwem przycisku *Edit*, może otworzyć formularz zaprezentowany na rysunku 5.42. Pozwala on na edycję wymienionych wyżej elementów wchodzących w skład projektu odpowiednio za pomocą przycisków *Images*, *Containers*, *Box* oraz *Elements*. Dodatkowo za pomocą tego formularza można zmienić nazwę, opis, dodać notatki, zmienić status projektu na szablon, przypisać projekt do pracownika oraz dodać gry, do których szablon może przynależeć.

Edit Project

catan project	Images
Project for catan game	Containers
	Box
	Elements

Notes Completed: No Template: No

Assigned to: None

Choose Game: Catan

Save Data

Rysunek 5.42: Formularz edycji projektu.
Źródło: opracowanie własne

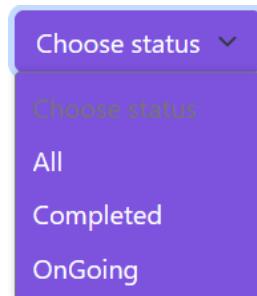
5.4.5. „My workspace” – Moja przestrzeń robocza

Kafelek moja przestrzeń robocza jest odnośnikiem przenoszącym użytkownika do podstrony zaprezentowanej na rysunku 5.43. Jak nazwa mówi strona ta pełni funkcję przestrzeni, w której realizowane są projekty danego zalogowanego użytkownika, będąc główną funkcjonalnością serwisu. Pracownik oczywiście za pośrednictwem panelu może filtrować zadane projekty po statusie rysunek 5.44.

ID	NAME	CONTAINERS	ELEMENTS	COMPLETED	GAME	INFO	CONTINUE WORK
10	catan project	1	1	OnGoing	Catan	Info	Continue

Rysunek 5.43: Widok okna przestrzeni roboczej.

Źródło: opracowanie własne



Rysunek 5.44: Kategorie filtrowania dla projektów w aplikacji GameFlow.

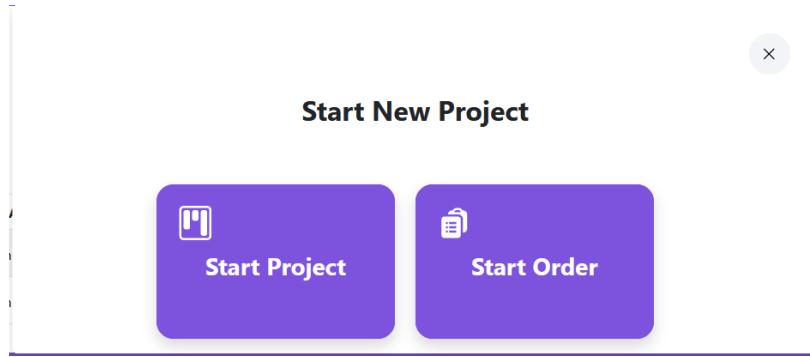
Źródło: opracowanie własne

Za pośrednictwem przycisków znajdujących się na tym widoku oraz przy każdym projekcie w liście pracownik może:

Przeglądać informacje o danym projekcie – pracownik za pośrednictwem przycisku *info*, może otworzyć formularz zaprezentowany już wcześniej na rysunku 5.38 otrzymując tym samym możliwość przypomnienia sobie, co wchodziło w skład danego projektu.

Zacząć nowy projekt – pracownik za pośrednictwem przycisku *Start New Project*, może otworzyć formularz pozwalający rozpoczęć projekt na dwa sposoby - rysunek 5.45:

- *Start project* – udostępnia możliwość rozpoczęcia projektu bez konieczności posiadania na nie zamówienia rysunek 5.46. Wystarczy, że pracownik odnajdzie projekt, który go interesuje upewniając się za pomocą możliwości przejrzenia informacji o danym projekcie, że odnaleziony projekt jest tym właściwym, kliknie przycisk *Start project* rozpoczęt projekt. Po kliknięciu tego przycisku docelowy projekt przestrzeni się w przestrzeni roboczej;
- *Start Order* – udostępnia możliwość rozpoczęcia realizacji przypisanego do siebie zamówienia przez pracownika - rysunek 5.47. Użytkownik wybiera zamówienie, którego chce rozpoczęć realizację, a następnie wybiera szablon, w oparciu o który można realizować daną grę - rysunek 5.48.



Rysunek 5.45: Formularz rozpoczęcia realizacji nowego projektu.
Źródło: opracowanie własne

A screenshot of a user interface for selecting a project. The title is "Projects". A dropdown menu shows "Catan". Below it is a table with one row, showing details for a project named "catan project" with ID 1. The table has columns: ID, NAME, BOX, CONTAINERS, ELEMENTS, INFO, and START. The "INFO" and "START" buttons are purple. Navigation buttons at the bottom are labeled "previous", "1", and "next".

ID	NAME	BOX	CONTAINERS	ELEMENTS	INFO	START
1	catan project	catanBox	1	1	<button>Info</button>	<button>Start</button>

Rysunek 5.46: Formularz wyboru projektu do realizacji.
Źródło: opracowanie własne

A screenshot of a user interface for starting an order. The title is "Orders". A table shows one order for "Pandemic" with ID 4. The table columns are: ID, ORDERED GAME, STATUS, SUBMIT DATE, LAST UPDATE, INFO, and START. The "INFO" and "START" buttons are purple.

ID	ORDERED GAME	STATUS	SUBMIT DATE	LAST UPDATE	INFO	START
4	Pandemic	CLAIMED	2023-10-18	2023-12-11	<button>Info</button>	<button>Start</button>

Rysunek 5.47: Formularz rozpoczęcia realizacji danego zamówienia.
Źródło: opracowanie własne

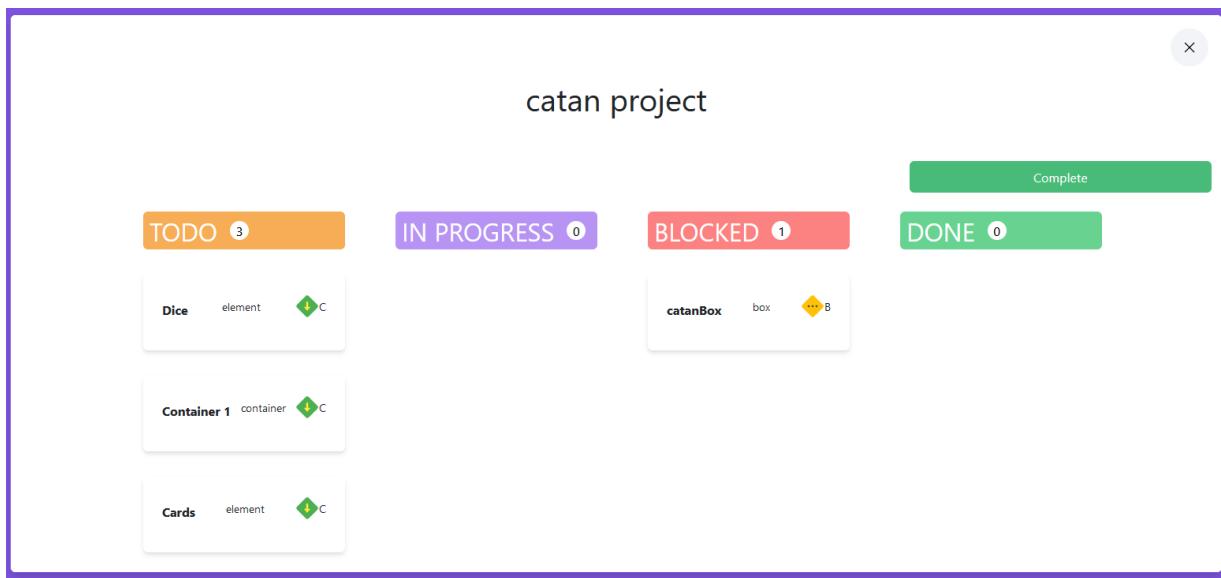
A screenshot of a user interface for selecting a project from an order. The title is "Select Project". A table shows one project for "pandemic project" with ID 2. The table columns are: ID, NAME, BOX, CONTAINERS, ELEMENTS, INFO, and START. The "INFO" and "START" buttons are purple.

ID	NAME	BOX	CONTAINERS	ELEMENTS	INFO	START
2	pandemic project	pandemicBox	1	1	<button>Info</button>	<button>Start</button>

Rysunek 5.48: Formularz wyboru projektu do złożonego zamówienia.
Źródło: opracowanie własne

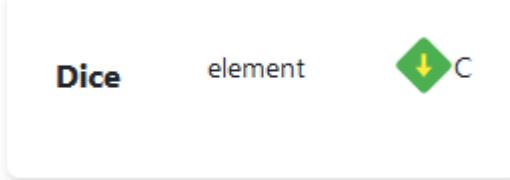
Kontynuować prace nad danym projektem – pracownik za pośrednictwem przycisku *Continue*, może przejść do widoku realizacji danego projektu - rysunek 5.49. W tym widoku widzimy 4 kolumny:

- *TODO* – do zrobienia, zawierająca elementy, które należy wykonać, żeby zrealizować dany projekt;
- *IN PROGRESS* – w trakcie pracy, zawierająca aktualnie wykonywane elementy;
- *BLOCKED* – zablokowane, zawierająca zablokowane elementy, które nie są aktualnie wykonywane;
- *DONE* – zrobione, zawierająca elementy, które zostały już wykonane.

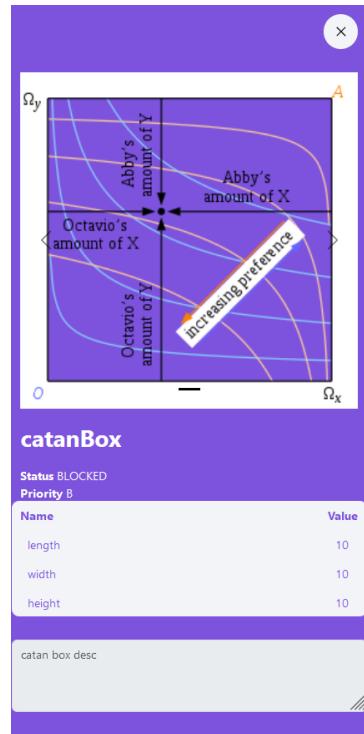


Rysunek 5.49: Formularz realizacji danego projektu.
Źródło: opracowanie własne

Każdy komponent danego projektu traktowany jest jako osobne zadanie - rysunek 5.50 posiadające swoją nazwę, typ, priorytet. Dodatkowo klikając w dane zadanie możemy otworzyć formularz zawierający informacje o nim, co pokazuje rysunek 5.51.



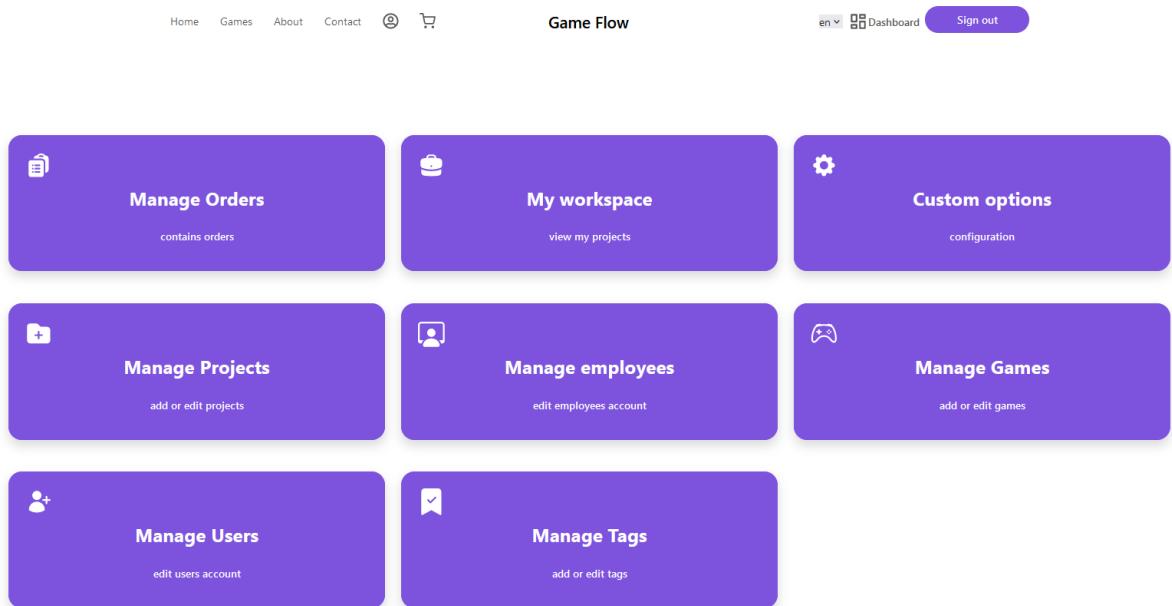
Rysunek 5.50: Reprezentacja pojedynczego zadania do realizacji.
Źródło: opracowanie własne



Rysunek 5.51: Formularz informacji o zadanym elemencie do realizacji.
 Źródło: opracowanie własne

5.5. Wprowadzenie do widoku administratora

Panel administratora jest rozszerzeniem panelu pracownika o kilka opcji wymagających specjalnych uprawnień do ich wykonywania, a do których pracownik nie powinien mieć dostępu.

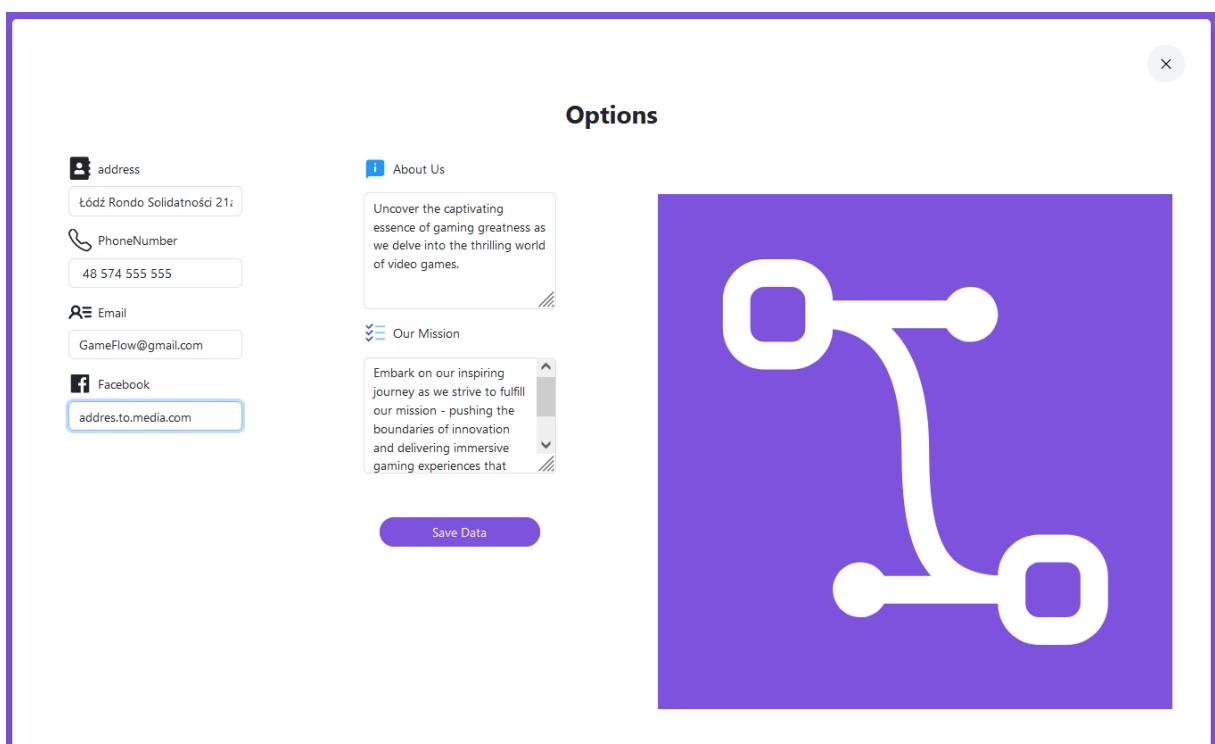


Rysunek 5.52: Widok panelu administratora.
 Źródło: opracowanie własne

5.5.1. „Custom options” – Opcje niestandardowe

Kafelek opcje niestandardowe jest odnośnikiem przenoszącym użytkownika do podstrony zaprezentowanej na rysunku 5.53. Za pośrednictwem tego widoku administrator serwisu może edytować w szybki i łatwy sposób takie dane jak:

- *Address* – adres, pod którym znajduje się instytucja;
- *PhoneNumber* – numer telefonu do przedsiębiorstwa;
- *Email* – email do przedsiębiorstwa;
- *Facebook* – adres do mediów społecznościowych przedsiębiorstwa;
- *About us* – informacje o przedsiębiorstwie;
- *Our Mission* – cel, do którego obecnie dąży przedsiębiorstwo.



Rysunek 5.53: Widok edycji opcji niestandardowych.
 Źródło: opracowanie własne

5.5.2. „Manage employees” – Zarządzaj pracownikami

Kafelek zarządzaj pracownikami jest odnośnikiem przenoszącym użytkownika do podstrony zaprezentowanej na rysunku 5.54. Za pośrednictwem tego widoku administrator serwisu może, zarządzać pracownikami w tym aktywować lub deaktywować ich konta, przeglądać informacje o danym pracowniku – rysunek 5.55, edytować konto danego pracownika – rysunek 5.56 oraz stworzyć nowe konto wypełniając odpowiednie pola formularza zaprezentowanego na rysunku 5.57.

Employees

Add new employee

USERNAME	EMAIL	ROLE	DETAILS	EDIT	ACTIVATION
admin	admin@gmail.com	admin	<button>Info</button>	<button>Edit</button>	<button>Deactivate</button>
employee	employee@gmail.com	employee	<button>Info</button>	<button>Edit</button>	<button>Deactivate</button>
admin2User	admin2@gmail.com	admin	<button>Info</button>	<button>Edit</button>	<button>Deactivate</button>
AdminUser123	admin3@gmail.com	admin	<button>Info</button>	<button>Edit</button>	<button>Deactivate</button>
userUser	user@user.pl	employee	<button>Info</button>	<button>Edit</button>	<button>Deactivate</button>
Bobadd	asdsaadad@gmail.com	admin	<button>Info</button>	<button>Edit</button>	<button>Deactivate</button>

previous 1 next

Rysunek 5.54: Widok zarządzania pracownikami w aplikacji GameFlow.
 Źródło: opracowanie własne

Employee: 1

Username: admin

Email: admin@gmail.com

PhoneNumber: 555555555

Role: admin

Active: Yes

Rysunek 5.55: Formularz informacji o pracowniku.
 Źródło: opracowanie własne

The screenshot shows a mobile application interface titled "Edit Employee". The form fields are as follows:

- Username**: admin
- Email**: admin@gmail.com
- PhoneNumber**: 55555555
- Password**: (empty field)
- Role**: admin

A toggle switch labeled "Active: Yes" is present. At the bottom is a "Save Data" button.

Rysunek 5.56: Formularz edycji pracownika.

Źródło: opracowanie własne

The screenshot shows a mobile application interface titled "Add Employee". The form fields are as follows:

- Email**: Enter email
- Username**: s@m1n@gmail.com
- PhoneNumber**: Enter phone number
- Password**: (empty field)
- Role**: employee

A toggle switch labeled "Active: Yes" is present. At the bottom is a "Save" button.

Rysunek 5.57: Formularz dodania nowego pracownika.

Źródło: opracowanie własne

5.5.3. „Manage Users” – Zarządzaj użytkownikami

Kafelek zarządzaj użytkownikami jest odnośnikiem przenoszącym użytkownika do podstrony zaprezentowanej na rysunku 5.58. Za pośrednictwem tego widoku administrator serwisu może zarządzać kontami użytkowników w serwisie w tym aktywować lub deaktywować ich konta, przeglądać informacje o danym użytkowniku – rysunek 5.59 oraz edytować konto danego użytkownika.

USERNAME	EMAIL	ROLE	DETAILS	EDIT	ACTIVATION
user	user@gmai.com	user	<button>Info</button> <button>Edit</button>	<button>Info</button> <button>Edit</button>	<button>Deactivate</button>
inactive	user2@gmai.com	user	<button>Info</button> <button>Edit</button>	<button>Info</button> <button>Edit</button>	<button>Activate</button>

Rysunek 5.58: Widok zarządzania użytkownikami w aplikacji GameFlow.
Źródło: opracowanie własne

User: 2

Username: user

Email: user@gmai.com

PhoneNumber: 444444444

Role: user

Active: Yes

Rysunek 5.59: Formularz informacji o użytkowniku.
Źródło: opracowanie własne

The screenshot shows a mobile application interface titled "Edit User". The background is a solid purple color. At the top right is a white circular button with a black "X" icon. Below the title, there are five input fields with labels and icons: "Username" (user), "Email" (user@gmai.com), "PhoneNumber" (444444444), "Password" (empty field), and "Role" (user). Underneath these fields is a toggle switch labeled "Active: Yes" with its switch turned on. At the bottom right is a white button with the text "Save Data".

Rysunek 5.60: Formularz edycji użytkownika.
 Źródło: opracowanie własne

6. Podsumowanie

6.1. Wnioski

Założeniem pracy była analiza wybranych istniejących aplikacji na rynku wspierających proces projektowania poprzez porównanie ich mocnych i słabych stron oraz analiza procesów towarzyszących obecnie w procesie wytwarzania i zamawiania produktu oraz pozostałych potrzeb konsumenckich i przełożenie tego na wymagania funkcjonalne i niefunkcjonalne oprogramowania. W wyniku czego przeprowadzając cały cykl projektowy udało się stworzyć aplikację *GameFlow* stanowiącą odpowiedź na potrzebę posiadania oprogramowania spełniającego wszystkie wymagania określone w rozdziałach 4.2 oraz 4.3. Porównując powstałe rozwiązanie do swoich poprzedników wymienionych w rozdziale 2, udało się nie popełnić drugi raz błędów tych systemów, łącząc tym samym najlepsze cechy każdego z nich w jednym programie. Rezultaty te zostały przedstawione w tabeli 6.1. Podsumowując można jednoznacznie stwierdzić, że cały cykl projektowy został przeprowadzony poprawnie i pozwolił na stworzenie oprogramowania spełniającego wszystkie ustalone założenia.

Tabela 6-1: Porównanie funkcjonalności oferowanych przez obecnie dostępne rozwiązania na rynku z powstającym rozwiązaniem

	Tworzenie projektu	Edycja elementów	Dodawanie potrzebnych właściwości	Integracja z zewnętrznym API	Zarządzanie projektem	priorytyzowanie i ustawianie statusu danych zadań	Sprawdzenia statusu z perspektywy klienta	Zamieszczania produktów na sprzedaż	Zamieszczanie informacji o swojej działalności
StoryBoardThat	✓	x	x	x	x	x	x	x	x
TableTopia	✓	ograniczone	ograniczone	x	ograniczone	x	x	x	x
Olx	x	x	x	x	x	x	ograniczone	✓	ograniczone
GameFlow	✓	✓	✓	✓	✓	✓	✓	✓	✓

6.2. 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 mogłoby pozwolić optymalnie wykorzystać czas operacyjny drukarki poprzez drukowanie elementów drobnych, a co za tym idzie, których czas powstania 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. Powoduje 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 generuje to stratę dla firmy pod kątem zamrożenia zainwestowanych pieniędzy w dany materiał. Tak samo w drugą stronę, zamówienie za mało danego materiału generuje 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.....	7
Rysunek 2.2: Ekran aplikacji webowej StoryboardThat.....	8
Rysunek 2.3: Logotyp systemu TableTopia.....	8
Rysunek 2.4: Ekran aplikacji webowej TableTopia.....	9
Rysunek 2.5: Tworzenie nowego obiektu w aplikacji webowej TableTopia.....	9
Rysunek 2.6: Logotyp systemu Olx.....	10
Rysunek 2.7: Ekran aplikacji webowej Olx.....	10
Rysunek 3.1: Wykres wartości rynkowej i prognozowanej w handlu internetowym na świecie według portalu ecommerce guide.....	12
Rysunek 3.2: Rozkład użycia języków programowania w ciągu ostatnich 12 miesięcy.....	13
Rysunek 3.3: Liczba pobrań języka programowania TypeScript.....	13
Rysunek 3.4: Logotyp języka programowania TypeScript.....	13
Rysunek 3.5: Logotyp frameworka NestJs.....	14
Rysunek 3.6: Logotyp frameworka React.....	15
Rysunek 3.7: Logotyp bazy danych PostgreSQL.....	16
Rysunek 3.8: Logotyp systemu kolejkowania BullIMQ.....	16
Rysunek 3.9: Schemat działania systemu kolejkowania BullMQ.....	17
Rysunek 3.10: Schemat wzorca Producer Consumer.....	17
Rysunek 3.11: Schemat działania komunikacji z zastosowaniem protokołu SMTP.....	18
Rysunek 3.12: Diagram sekwencji dla możliwych scenariuszy strategii JWT.....	19
Rysunek 4.1: Diagram czynności dla przedsiębiorcy nie korzystającego z systemów informatycznych.....	21
Rysunek 4.2: Diagram czynności dla przedsiębiorcy korzystającego z systemów informatycznych.....	22
Rysunek 4.3: Diagram komponentów dla architektury trójwarstwowej.....	25
Rysunek 4.4: Diagram komponentów prezentujący architekturę modularną w kontekście aplikacji GameFlow. Źródło: opracowanie własne	28
Rysunek 4.5: Diagram komponentów dla modelu MVC.....	28
Rysunek 4.6: Diagram klas dla aplikacji GameFlow.....	30
Rysunek 4.7: Diagram klas przedstawiający komponenty aplikacji GameFlow.....	31
Rysunek 4.8: Diagram pakietów dla rozmieszczenia elementów dla całego systemu.....	32
Rysunek 4.9: Budowa modułu auth.	33
Rysunek 4.10: Budowa modułu database.	34
Rysunek 4.11: Budowa modułu exceptions.....	34
Rysunek 4.12: Budowa modułu game.....	35
Rysunek 4.13: Budowa modułu image.....	35
Rysunek 4.14: Budowa modułu information.....	36
Rysunek 4.15: Budowa modułu order.	36
Rysunek 4.16: Budowa modułu project.	37
Rysunek 4.17: Budowa modułu users.	38
Rysunek 4.18: Przedstawienie modularnej budowy projektu.....	39
Rysunek 4.19: Listing kodu źródłowego implementacji dekoratora HasRole.	39
Rysunek 4.20: Listing kodu źródłowego implementacji dekoratora GetCurrentUser.....	40
Rysunek 4.21: Listing kodu źródłowego implementacji OrderMemberShip guard.	40
Rysunek 4.22: Diagram sekwencji dla sprawdzenia, czy użytkownik jest właścicielem zasobu.....	41
Rysunek 4.23: Listing kodu źródłowego implementacji RolesGuard.....	42
Rysunek 4.24: Diagram sekwencji dla uwierzytelnienia opartego na rolach.	42

Rysunek 4.25: Listing kodu źródłowego implementacji AuthorGuard.	43
Rysunek 4.26: Diagram sekwencji dla sprawdzenia, czy użytkownik może modyfikować projekt.....	44
Rysunek 4.27: Listing kodu źródłowego implementacji kolejkowania, producer.	44
Rysunek 4.28: Listing kodu źródłowego implementacji kolejkowania, consumer.	45
Rysunek 4.29: Diagram sekwencji dla rejestracji użytkownika.....	45
Rysunek 4.30: Listing kodu źródłowego implementacji strategii JWT.	46
Rysunek 4.31: Listing kodu źródłowego implementacji JWT guard.	46
Rysunek 4.32: Zrzut ekranu ze stanu odpalenia wszystkich testów dla aplikacji GameFlow.	47
Rysunek 4.33: Zawartość pliku konfiguracyjnego docker-compose.yaml.	49
Rysunek 5.1: Zawartość pliki konfiguracyjnego .env.	51
Rysunek 5.3: Pasek nawigacyjny aplikacji GameFlow.	52
Rysunek 5.4: Stopka strony aplikacji GameFlow.	52
Rysunek 5.5: Strona główna aplikacji GameFlow.	53
Rysunek 5.6: Przycisk przekierowujący do strony z produktami.	54
Rysunek 5.7: Sekcja kategorii prezentująca potencjalne kategorie w aplikacji GameFlow.	54
Rysunek 5.8: Sekcja popularnych produktów.	54
Rysunek 5.9: Strona z produktami dostępnymi w aplikacji GameFlow.	55
Rysunek 5.10: Prezentacja filtrowania produktów w aplikacji GameFlow.	55
Rysunek 5.11: Prezentacja przycisku informacji po najechaniu na dany produkt.	56
Rysunek 5.12: Prezentacja informacji o wybranym produkcie.	56
Rysunek 5.13: Strona przedstawiająca informację o firmie.	57
Rysunek 5.14: Strona zawierająca informacje kontaktowe aplikacji GameFlow.	58
Rysunek 5.15: Formularz logowania aplikacji GameFlow.	58
Rysunek 5.16: Formularz rejestracji do systemu GameFlow.	59
Rysunek 5.17: Formularz aktywacji konta w aplikacji GameFlow.	59
Rysunek 5.18: Strona udostępniająca możliwość aktualizacji swoich danych w aplikacji GameFlow...	60
Rysunek 5.19: Strona zawierająca złożone zamówienia w aplikacji GameFlow.	61
Rysunek 5.20: Formularz zawierająca informację o złożonym zamówieniu.	61
Rysunek 5.21: Formularz umożliwiający możliwość edycji zamówienia.	62
Rysunek 5.22: Prezentacja przycisku informacji oraz zamówienia po najechaniu na dany produkt....	63
Rysunek 5.23: Formularz złożenia zamówienia na dany produkt.	63
Rysunek 5.24: Pasek nawigacyjny aplikacji GameFlow po zalogowaniu się jako pracownik.	64
Rysunek 5.25: Widok panelu pracownika aplikacji GameFlow.	64
Rysunek 5.26: Widok edycji zamówienia aplikacji GameFlow.	65
Rysunek 5.27: Lista zawierająca możliwe statusy zamówień.	65
Rysunek 5.28: Formularz edycji zamówienia w aplikacji GameFlow.	66
Rysunek 5.29: Formularz informacji o zamówieniu w aplikacji GameFlow.	66
Rysunek 5.30: Widok zarządzania produktami w aplikacji GameFlow.	67
Rysunek 5.31: Formularz dodania nowej gry w aplikacji GameFlow.	68
Rysunek 5.32: Formularz informacji o grze w aplikacji GameFlow.	68
Rysunek 5.33: Formularz edycji gry w aplikacji GameFlow.	69
Rysunek 5.34: Widok zarządzania tagami w aplikacji GameFlow.	69
Rysunek 5.35: Formularz dodania nowego tagu w aplikacji GameFlow.	70
Rysunek 5.36: Formularz edycji tagu w aplikacji GameFlow.	70
Rysunek 5.37: Widok zarządzania projektami w aplikacji GameFlow.	71
Rysunek 5.38: Kategorię filtrowania dla projektów w aplikacji GameFlow.	71
Rysunek 5.39: Formularz zawierający informację o wybranym projekcie.	72
Rysunek 5.40: Formularz informacji o danym pudełku.	72

Rysunek 5.41: Formularz informacji o danym elemencie.....	72
Rysunek 5.42: Formularz informacji o danym pojemniku.....	73
Rysunek 5.43: Formularz edycji projektu.....	73
Rysunek 5.44: Widok okna przestrzeni roboczej.....	74
Rysunek 5.45: Kategorię filtrowania dla projektów w aplikacji GameFlow.....	74
Rysunek 5.46: Formularz rozpoczęcia realizacji nowego projektu.....	75
Rysunek 5.47: Formularz wyboru projektu do realizacji.....	75
Rysunek 5.48: Formularz rozpoczęcia realizacji danego zamówienia.....	75
Rysunek 5.49: Formularz wyboru projektu do złożonego zamówienia.....	75
Rysunek 5.50: Formularz realizacji danego projektu.....	76
Rysunek 5.51: Reprezentacja pojedynczego zadania do realizacji.....	76
Rysunek 5.52: Formularz informacji o zadanym elemencie do realizacji.....	77
Rysunek 5.53: Widok panelu administratora.....	77
Rysunek 5.54: Widok edycji opcji niestandardowych.....	78
Rysunek 5.55: Widok zarządzania pracownikami w aplikacji GameFlow.....	79
Rysunek 5.56: Formularz informacji o pracowniku.....	79
Rysunek 5.57: Formularz edycji pracownika.....	80
Rysunek 5.58: Formularz dodania nowego pracownika.....	80
Rysunek 5.59: Widok zarządzania użytkownikami w aplikacji GameFlow.....	81
Rysunek 5.60: Formularz informacji o użytkowniku.....	81
Rysunek 5.61: Formularz edycji użytkownika.....	82

Spis tabel

Tabela 2-1: Porównanie funkcjonalności oferowanych przez obecnie dostępne rozwiązania na rynku	11
Tabela 6-1: Porównanie funkcjonalności oferowanych przez obecnie dostępne rozwiązania na rynku z powstały rozwiązańiem	83

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] Model kaskadowy, [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%2FLUuDmt_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/lZbR5N9NRi4JZmiBkGsp7pUQikm8cQMZtnC2RN1e7xhU3u3cObSYUSquVoqgeuRQw>
- [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://d1wqxts1xzle7.cloudfront.net/50526307/MVC-libre.pdf?1480020702=&response-content-disposition=inline%3B+filename%3DModel_View_Controller_MVC_Architecture.pdf&Expires=1700915009&Signature=LDU2eYvi5Qkazm6h2gg0I9QR6nQGZ6mqkmkfNdZPr-peZm-skHuqaf576wq2z2Dvxrvu3JAeX9-uFQNNIg21pHTQz0CIIWG9cyihM3ol-7bO~1nFgzvdiUSCoN1r--o91-yNj9ThC~S3ZSSmgGWfZnfQDQcMbr1p~E1rJqP8eIEqPpf6X29xADxvfEfXxHrA5ccVNXeUpYwuE
- [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/>
- [66] Testy integracyjne, [online], [dostęp 23.12.2023] <https://ieeexplore.ieee.org/abstract/document/131377>
- [67] Framework Jest, [online], [dostęp 23.12.2023] <https://jestjs.io/>
- [68] Kontener testowy, [online], [dostęp 23.12.2023] <https://testcontainers.com/>