Internetowa gra przygodowa wspomagająca edukację wczesnoszkolną

An online adventure game supporting early school education

Poznań, 2025

Wstęp.....5
1. Aktualny stan wiedzy..... 6
1.1 Wprowadzenie..... 6
1.2. Mechanizmy dydaktyczne wykorzystywane w grach edukacyjnych.....6
1.2.1. Nauka poprzez eksplorację.....6
1.2.2. Symulacje edukacyjne.....6
1.2.3. Systemy nagród i grywalizacja.....6
1.3. Wpływ gier edukacyjnych na rozwój dzieci — przegląd badań..... 6
1.3.1 Badania dotyczące skuteczności nauczania poprzez gry.....6
1.3.2 Wpływ grywalizacji na motywację do nauki..... 7
1.4. Analiza rynku gier edukacyjnych.....7
1.4.1 Przegląd popularnych platform.....7
1.5. Technologie wykorzystywane w grach edukacyjnych.....7
1.5.1 Technologie webowe..... 7
1.5.2 Cloud computing..... 7
1.6. Podsumowanie.....8
2. Cel i zakres pracy oraz podział zadań..... 9
2.1 Cel pracy.....9
2.2 Założenia projektowe.....9
3. Metodyka pracy.....9
3.1. Wprowadzenie..... 9
3.2. Czym jest Kanban?.....9
3.3. Planowanie i podział zadań.....10
3.4. Diagram Gantta jako wsparcie planowania..... 10
3.5. Zalety stosowania Kanban w naszym projekcie..... 10
3.6. Podsumowanie..... 11
3.7 Wymagania funkcjonalne..... 11
3.7.1 Tabele wymagań funkcjonalnych..... 11
3.8. Diagramy akcji i przypadków użycia..... 19
3.9 Diagramy klas..... 31
3.9.1 Diagramy klas dla plików z backendu.....31
3.9.2 Diagramy klas dla plików z frontendu.....51
3.10 Diagram komponentów..... 60
3.11 Diagramy przypadków użycia.....61
4. Opis realizacji projektu..... 66
4.1 Wprowadzenie.....66
4.2.1 Projektowanie stron i podstron.....66
4.2.2 Backend development — podstawy.....66
4.2.3 Frontend development — podstawy..... 66
4.2.4 Backend development — główne funkcjonalności.....66
4.2.5 Frontend development — panel użytkownika.....67
4.2.6 Backend development — rozwinięcia.....67
4.2.7 Frontend development — panel administratorski..... 67
4.2.8 Testowanie i poprawki.....67
4.2.9 Finalne dopracowanie i wdrożenie..... 67

5. Wdrożenie i eksploatacja....68
5.1. Wdrożenie w procesie rozwoju oprogramowania....68
5.1.1. Wdrożenie aplikacji....68
5.1.2. Konfiguracja serwera....68
5.1.3. Kopie zapasowe i monitoring....68
5.2. Wdrożenie produkcyjne....69
5.3 Działanie frontendu....69
5.3.1 Wybór technologii....69
5.3.2 Budowanie aplikacji....71
5.3.3 Struktura frontendu....72
5.4 Działanie backendu....73
5.4.1 Technologia backendowa: Node.js....73
5.4.2 Biblioteki i moduły używane w aplikacji....73
5.4.3 Połączenie z bazą danych:....76
5.4.4 Kontrolery....77
5.4.5 Pliki cron....78
5.4.6 Automatyczna wysyłka wiadomości e-mail w aplikacji backendowej....78
5.4.7 Implementacja mechanizmu wysyłki wiadomości....78
5.4.8 Middleware....80
5.4.9 Modele....81
5.4.10 Organizacja tras (routes)....89
5.4.11 Funkcja i implementacja folderu utils....91
5.4.12 Plik głównego - app.js....92
5.4.13 Konfiguracja i uruchomienie serwera HTTPS w aplikacji backendowej (server.js)....94
5.5 Baza danych MongoDB....96
5.5.1 Struktura kolekcji i sposób przechowywania danych....96
5.5.2 Konfiguracja....98
5.2.3 Korzyści wynikające z wykorzystania MongoDB....99
5.6 Struktura Serwera....99
5.7 Instrukcja użytkowania....100
6. Testowanie oprogramowania....111
6.1 Wstęp do testów....111
6.2 Testy Postman....111
6.2.1 Testy API za pomocą aplikacji PostMan....112
6.2 Podsumowanie testów Postman....172
6.2.1 Wprowadzenie....172
6.2.2 Przeprowadzone testy....172
6.2.3 Pominięte testy....172
6.2.4 Wyniki testów....172
6.2.5 Wnioski....172
6.3 Testy manualne....173
6.4 Testy E2E....177
6.4 Podsumowanie testowania....181
Podsumowanie....182
Spis rysunków....185

Spis tabel **195**
Streszczenie **197**
Abstract **198**

Wstęp

Ciągły i niezachwiany rozwój technologii informacyjnych od początku XXI wieku nieprzerwanie otwiera nowe możliwości w dziedzinie nauki i edukacji. W erze powszechnego dostępu dzieci do smartfonów oraz komputerów można by zadać sobie pytanie jak wykorzystać upowszechnienie się tych urządzeń wśród najmłodszych w celu poprawy poziomu ich wiedzy. Stąd celem naszej pracy jest stworzenie internetowej platformy edukacyjnej, zawierającej elementy gry, w której użytkownik poprzez tworzenie spersonalizowanego bohatera i rozwiązywanie zadań z zakresu wiedzy wczesnoszkolnej, będzie w stanie poszerzyć swoje horyzonty wiedzowe w angażujący i nieodpychający dla młodego umysłu sposób.

Rozwój gier edukacyjnych na przestrzeni ostatnich lat znacząco wpłynął na sposób, w jaki dzieci zdobywają wiedzę i rozwijają swoje umiejętności. Badania wykazują, że stosowanie elementów grywalizacji w nauczaniu znacznie podnosi motywację uczniów oraz poprawia ich zdolność do przyswajania nowych informacji. Współczesne systemy edukacyjne coraz częściej wdrażają interaktywne aplikacje, które łączą tradycyjne metody nauczania z nowoczesnymi technologiami.

W tym celu zespół opracował wymagania funkcjonalne aplikacji, określił priorytety pracy oraz grupę docelową i wybrał odpowiednie narzędzia technologiczne. W celu elastycznego ale również efektywnego zarządzania projektem grupa zdecydowała się na zastosowanie metodologii Kanban. Aplikacja działa na serwerze Linux Debian a technologie zastosowane w projekcie to Express.js oraz React.

Niniejsza praca szczegółowo przedstawia trudności napotkane przez zespół w trakcie tworzenia aplikacji, omawia kluczowe funkcjonalności oraz przedstawia proces projektowania i implementacji rozwiązań zastosowanych w projekcie. W dalszych rozdziałach przedstawione zostały również opisy oraz wyniki testów przeprowadzonych w celu zapewnienia niezawodności i efektywnego działania wszystkich elementów, które finalnie składają się na internetową platformę edukacyjną — SQULIO.

1. Aktualny stan wiedzy

1.1 Wprowadzenie

Gry edukacyjne online stanowią dynamicznie rozwijający się sektor technologii edukacyjnych, który łączy interaktywne metody nauczania z zaawansowanymi rozwiązaniami technologicznymi. Współczesne aplikacje tego typu wykorzystują różne podejścia dydaktyczne, takie jak nauka poprzez eksplorację, symulacje czy systemy nagród. W niniejszym opracowaniu zostaną omówione mechanizmy wykorzystywane w grach edukacyjnych, ich wpływ na rozwój dzieci, analiza rynku oraz aspekty technologiczne.

1.2. Mechanizmy dydaktyczne wykorzystywane w grach edukacyjnych

1.2.1. Nauka poprzez eksplorację

Eksploracyjne metody nauczania pozwalają dzieciom na samodzielne odkrywanie treści edukacyjnych, co sprzyja głębszemu zrozumieniu materiału. Zgodnie z badaniami przeprowadzonymi przez Paperta (1993)[5], nauka przez odkrywanie rozwija u dzieci zdolności analityczne i umożliwia lepsze zrozumienie abstrakcyjnych koncepcji.

1.2.2. Symulacje edukacyjne

Symulacje pozwalają na interaktywne modelowanie rzeczywistych sytuacji, co ułatwia przyswajanie wiedzy w sposób praktyczny i angażujący. Jak wskazują badania Clark i Mayer (2016) [1], wykorzystanie symulacji w edukacji prowadzi do zwiększenia zdolności rozwiązywania problemów oraz poprawy wyników w nauce w porównaniu do tradycyjnych metod.

1.2.3. Systemy nagród i grywalizacja

Mechanizmy grywalizacyjne, takie jak zdobywanie punktów, odznak i poziomów, zwiększają motywację dzieci do nauki oraz wpływają na ich długofalowe zaangażowanie. Badania Deterdinga i Dixon (2011)[2] pokazują, że elementy grywalizacji mogą znacząco poprawić skuteczność nauczania, szczególnie wśród dzieci o niższym poziomie motywacji wewnętrznej.

1.3. Wpływ gier edukacyjnych na rozwój dzieci — przegląd badań

1.3.1 Badania dotyczące skuteczności nauczania poprzez gry

Liczne badania naukowe potwierdzają, że gry edukacyjne wspierają rozwój umiejętności poznawczych, logicznego myślenia oraz zdolności rozwiązywania problemów.

Przykładowe wyniki badań:

- Badania przeprowadzone przez Uniwersytet Ekonomiczny [3] w Krakowie wykazały, że dzieci korzystające z gier edukacyjnych osiągają o 20% lepsze wyniki w testach matematycznych w porównaniu do rówieśników uczących się tradycyjnymi metodami.
- Eksperymenty na Uniwersytecie Wrocławskim [6] wskazują, że interaktywne metody nauki sprzyjają długotrwałemu zapamiętywaniu materiału. Według wyników badań, uczniowie korzystający z gier edukacyjnych pamiętali 40% więcej informacji niż ich rówieśnicy uczący się bez elementów interaktywnych.

1.3.2 Wpływ grywalizacji na motywację do nauki

Gry edukacyjne, które stosują mechanizmy nagradzania i rywalizacji, są skutecznym narzędziem podtrzymującym zainteresowanie dzieci edukacją. Badania przeprowadzone przez MIT (2020) wskazują, że dzieci uczące się w środowisku grywalizacyjnym wykazują większą chęć do podejmowania wyzwań edukacyjnych oraz są bardziej skłonne do samodzielnej nauki. Cytując badanie: „Systemy nagród i interaktywne elementy w grach edukacyjnych zwiększają zaangażowanie uczniów o 35% w porównaniu do tradycyjnych metod nauczania" (MIT, 2020).

1.4. Analiza rynku gier edukacyjnych

1.4.1 Przegląd popularnych platform

Obecnie na rynku funkcjonuje wiele platform edukacyjnych, które skutecznie integrują grywalizację z procesem nauczania. Przykłady:

- Khan Academy Kids — interaktywna platforma edukacyjna dla najmłodszych, oferująca gry matematyczne, językowe i logiczne.
- Prodigy — matematyczna gra RPG, która angażuje dzieci w naukę poprzez eksplorację i system nagród.
- ABCmouse — kompleksowy system nauczania dla dzieci w wieku przedszkolnym i wczesnoszkolnym, który stosuje grywalizację do nauki czytania i matematyki.

1.4.2 Trendy i perspektywy rozwoju

Rynek gier edukacyjnych stale rośnie, co jest związane z:

- rosnącym dostępem do technologii internetowych,
- wzrostem zapotrzebowania na interaktywne metody nauczania,
- popularyzacją nauki zdalnej.

Według raportu Research and Markets (2022), rynek gier edukacyjnych osiągnie wartość 25 miliardów dolarów do 2026 roku, przy średnim rocznym wzroście na poziomie 8,5%.

1.5. Technologie wykorzystywane w grach edukacyjnych

1.5.1 Technologie webowe

- Nowoczesne gry edukacyjne opierają się na technologiach webowych, co zapewnia ich szeroką dostępność i łatwość aktualizacji.
- React, Vue.js — frameworki frontendowe umożliwiające budowanie dynamicznych interfejsów użytkownika,
- Node.js, Django — rozwiązania backendowe zapewniające płynność działania aplikacji,
- bazy danych NoSQL — np. Firebase, MongoDB, pozwalające na elastyczne zarządzanie danymi użytkowników.

1.5.2 Cloud computing

Technologie chmurowe umożliwiają skalowanie aplikacji oraz obsługę dużej liczby użytkowników bez utraty wydajności. Popularne rozwiązania, takie jak AWS, Google Cloud czy Microsoft Azure, zapewniają szybkie wdrażanie aktualizacji oraz stabilność działania gier edukacyjnych.

1.6. Podsumowanie

Gry edukacyjne stanowią skuteczne narzędzie wspierające naukę dzieci poprzez interaktywne i angażujące metody dydaktyczne. Ich rozwój jest wspierany przez nowoczesne technologie webowe oraz mechanizmy grywalizacji. Badania naukowe jednoznacznie potwierdzają pozytywny wpływ takich metod na wyniki edukacyjne i motywację dzieci. Wzrost popularności takich aplikacji wskazuje na ich istotną rolę w przyszłości edukacji cyfrowej.

2. Cel i zakres pracy oraz podział zadań

2.1 Cel pracy

Celem pracy inżynierskiej było zaprojektowanie oraz stworzenie internetowej platformy edukacyjnej służącej do polepszenia zdolności do nauki oraz poszerzenia wiedzy dzieci w wieku wczesnoszkolnym. Aplikacja ma w przyjazny dla użytkownika sposób zachęcać go do regularnej nauki poprzez system nagród.

2.2 Założenia projektowe

- Responsywność — Aplikacja powinna działać poprawnie na urządzeniach mobilnych, tabletach i komputerach (RWD).
- Poziomy trudności — Implementacja 5 poziomów trudności, dostosowujących pytania do umiejętności użytkownika.
- System nagród i grywalizacji — Użytkownicy zdobywają punkty doświadczenia (EXP), monety oraz osiągnięcia za wykonywanie zadań.
- Personalizacja postaci — Możliwość zmiany wyglądu postaci poprzez zakup ubrań i akcesoriów.
- Codzienne wyzwania — Każdego dnia użytkownik otrzymuje zestaw pytań, które musi wykonać, aby utrzymać postać w dobrym nastroju.
- Autoryzacja użytkowników — Rejestracja i logowanie odbywa się przy użyciu JWT (JSON Web Token), zapewniając bezpieczeństwo danych.
- Integracja z API — Aplikacja komunikuje się z backendem poprzez REST API, umożliwiając pobieranie i zapisywanie postępów użytkownika.
- System regeneracji życia — Jeśli użytkownik źle odpowie na pytanie, traci życie, które regeneruje się po upływie określonego czasu lub poprzez zakup w sklepie serca.
- Modularna architektura kodu — Kod aplikacji podzielony jest na moduły backendowe (Express.js, MongoDB) i komponenty frontendowe (React, TypeScript) dla lepszej organizacji i łatwej rozbudowy.
- Panel administracyjny — Administratorzy mogą zarządzać pytaniami, poziomami trudności i statystykami użytkowników.

3. Metodyka pracy

3.1. Wprowadzenie

W trakcie tworzenia założeń oraz realizacji całego projektu zdecydowaliśmy się na wykorzystanie metodyki Kanban, która jest jedną z popularnych metod zarządzania pracą w realizacji projektów programistycznych. Kanban umożliwia swobodne dostosowywanie priorytetów, poprawę efektywności oraz szybkie reagowanie na zmieniające się wymagania projektu.

3.2. Czym jest Kanban?

Kanban to elastyczna metodyka zarządzania zadaniami, która koncentruje się na wizualizacji pracy, ograniczeniu ilości zadań w toku (WIP — Work In Progress) oraz optymalizacji przepływu pracy. Jest szczególnie przydatna w zespołach, które muszą dynamicznie reagować na zmieniające się potrzeby i nie mają ścisłego podziału na iteracje, jak w przypadku metodyki Scrum.[7]

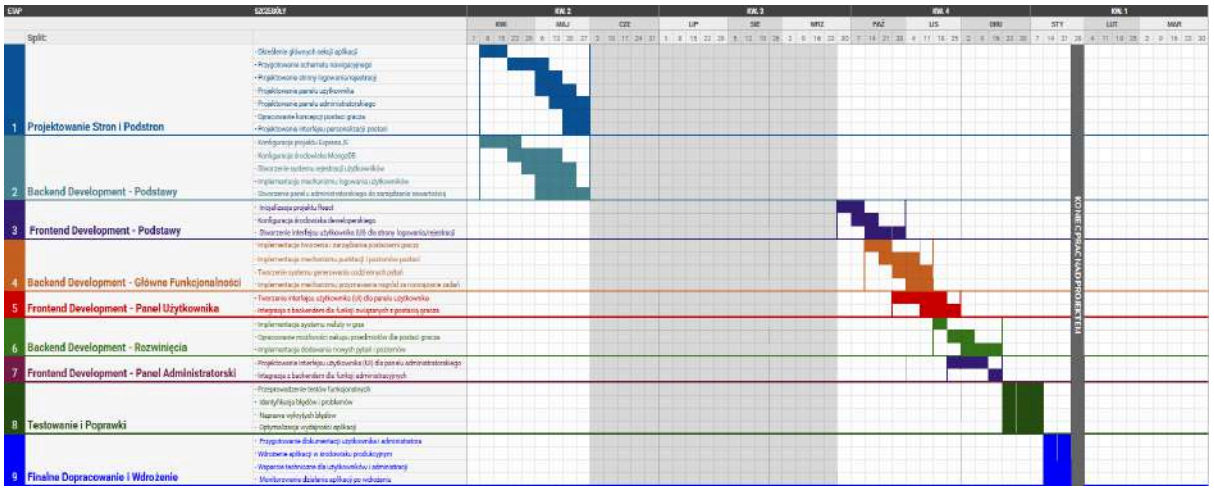
3.3. Planowanie i podział zadań

Cały zespół miał dostęp do tablicy Kanban i mógł przypisywać sobie zadania. W miarę realizacji postępu prac regularnie aktualizowaliśmy statusy zadań. Podczas cotygodniowych spotkań przeglądaliśmy tablicę i omawialiśmy postęp.

3.4. Diagram Gantta jako wsparcie planowania

W celu efektywnego monitorowania harmonogramu realizacji poszczególnych etapów projektu opracowaliśmy diagram Gantta, który pozwolił na wizualne śledzenie postępów prac. Diagram przedstawiał podział projektu na kluczowe fazy, takie jak:

- Projektowanie interfejsu i architektury systemu — obejmowało opracowanie schematów UI i wybór technologii.
- Implementacja backendu i frontend — podzielona na podstawowe i zaawansowane funkcjonalności.
- Testowanie i poprawki — identyfikacja błędów i ich eliminacja.
- Finalne wdrożenie i optymalizacja — ostatni etap obejmujący przygotowanie dokumentacji i monitorowanie działania systemu po wdrożeniu.
- Diagram Gantta umożliwił zespołowi lepsze planowanie czasu pracy i przewidywanie ewentualnych opóźnień. Połączenie Kanban z Ganttem pozwoliło na elastyczne dostosowanie harmonogramu do realnych postępów, co przyczyniło się do efektywnego zarządzania zadaniami.



Rysunek 3.1 Diagram Gantta (źródło: opracowanie własne)

3.5. Zalety stosowania Kanban w naszym projekcie

Dzięki wykorzystaniu Kanban osiągnęliśmy następujące korzyści:

- Lepsza organizacja pracy — wizualizacja tablicy pozwoliła zespołowi na szybkie zrozumienie, jakie zadania są w toku i jakie mają priorytet.
- Elastyczność — możliwość dynamicznego dodawania i reorganizacji zadań bez konieczności długich cykli planowania.
- Szybsza identyfikacja wąskich gardeł — ograniczenie liczby zadań w toku pozwoliło uniknąć blokad w procesie pracy.
- Lepsza komunikacja w zespole — codzienna aktualizacja tablicy i regularne spotkania ułatwiły współpracę i podział obowiązków.

- Stała optymalizacja — analizowaliśmy, jakie etapy procesu wymagają usprawnienia i wprowadzaliśmy zmiany na bieżąco.

3.6. Podsumowanie

Metodyka Kanban okazała się skutecznym rozwiązaniem w naszym projekcie, pozwalając nam elastycznie zarządzać zadaniami, reagować na zmiany i optymalizować przepływ pracy. W połączeniu z diagramem Gantta umożliwiła skuteczne zarządzanie harmonogramem oraz dostosowanie pracy do rzeczywistych postępów. Dzięki temu uniknęliśmy niepotrzebnych opóźnień, zwiększyliśmy przejrzystość realizowanych zadań i poprawiliśmy efektywność zespołu. W przyszłości możemy rozważyć dalsze usprawnienia, takie jak bardziej zaawansowane automatyzacje w systemie zarządzania zadaniami.

3.7 Wymagania funkcjonalne

3.7.1 Tabele wymagań funkcjonalnych

Tabela 3.1 Tabela wymagań funkcjonalnych — logowanie (źródło: opracowanie własne)

Cecha	Opis
Nazwa funkcji	Logowanie
Opis funkcji	Umożliwia użytkownikom zalogowanie się do aplikacji przy użyciu adresu e-mail i hasła.
Dane wejściowe	Adres e-mail, hasło, opcjonalnie „zapamiętaj mnie”.
Źródło danych wejściowych	Formularz logowania w aplikacji webowej.
Wynik	Użytkownik zostaje zalogowany, a system generuje token uwierzytelniający.
Wymagania	Użytkownik musi posiadać zarejestrowane konto; adres e-mail i hasło muszą być poprawne; weryfikacja stanu konta (np. aktywne, wymaga aktywacji); konto użytkownika nie może być zablokowane.
Przeznaczenie	Umożliwienie użytkownikom dostępu do aplikacji po uwierzytelnieniu.
Warunek wstępny	Konto użytkownika istnieje, dostęp do internetu i strony aplikacji.
Warunek końcowy	Użytkownik zalogowany, token uwierzytelniający nadany.
Efekty uboczne	Aktualizacja czasu ostatniego logowania użytkownika, generacja tokena JWT, ewentualne dodatkowe obciążenie serwera baz danych.
Powód wywołania funkcji	Chęć użytkownika do zalogowania się i korzystania z aplikacji.

Tabela 3.2 Tabela wymagań funkcjonalnych — rejestracja (źródło: opracowanie własne)

Cecha	Opis
Nazwa funkcji	Rejestracja
Opis funkcji	Pozwala nowym użytkownikom na tworzenie konta w aplikacji.
Dane wejściowe	Nazwa użytkownika, adres e-mail, hasło.
Źródło danych wejściowych	Formularz rejestracyjny w aplikacji webowej.
Wynik	Utworzenie nowego konta użytkownika i przesłanie kodu aktywacyjnego na podany adres e-mail, generowanie tokenu uwierzytelniającego, przekierowanie użytkownika na stronę z aktywacją
Wymagania	Wszystkie pola formularza muszą być poprawnie wypełnione. Adres e-mail musi być unikalny i nie może pochodzić z jednorazowych domen. Hasło musi spełniać kryteria bezpieczeństwa (min. 8 znaków, co najmniej jedna cyfra, brak znaków specjalnych SQL/NoSQL).
Przeznaczenie	Umożliwienie dostępu nowym użytkownikom do aplikacji.
Warunek wstępny	Dostęp do internetu i strony aplikacji.
Warunek końcowy	Konto użytkownika utworzone i gotowe do aktywacji.
Efekty uboczne	Zapis nowego rekordu w bazie danych, wysłanie e-maila z kodem aktywacyjnym, generacja dodatkowego ruchu w bazie danych.
Powód wywołania funkcji	Chęć założenia konta i korzystania z aplikacji.

Tabela 3.3 Tabela wymagań funkcjonalnych — aktywacja konta (źródło: opracowanie własne)

Cecha	Opis
Nazwa funkcji	Aktywacja konta
Opis funkcji	Pozwala użytkownikowi aktywować swoje konto po rejestracji za pomocą kodu aktywacyjnego.
Dane wejściowe	Kod aktywacyjny przesłany na e-mail użytkownika.
Źródło danych wejściowych	Formularz aktywacji w aplikacji webowej.
Wynik	Konto użytkownika zostaje aktywowane a użytkownik zostaje przekierowany na stronę z tworzeniem postaci
Wymagania	Kod aktywacyjny musi być poprawny i zgodny z danymi w bazie. Użytkownik musi być zalogowany (token JWT).
Przeznaczenie	Umożliwienie nowym użytkownikom korzystania z aplikacji po weryfikacji konta.
Warunek wstępny	Użytkownik musi mieć zarejestrowane konto i otrzymać kod aktywacyjny na e-mail. Musi być zalogowany, aby przeprowadzić aktywację.

Warunek końcowy	Konto użytkownika zostaje oznaczone jako „enabled” w bazie danych.
Efekty uboczne	Aktualizacja statusu konta w bazie danych, ewentualne logowanie zdarzeń, generowanie ruchu w bazie danych.
Powód wywołania funkcji	Chęć aktywacji konta i rozpoczęcia korzystania z aplikacji.

Tabela 3.4 Tabela wymagań funkcjonalnych — zapis postaci (źródło: opracowanie własne)

Cecha	Opis
Nazwa funkcji	Zapis postaci
Opis funkcji	Umożliwia użytkownikowi zapisanie stworzonej postaci, w tym jej wyglądu oraz nazwy, do bazy danych.
Dane wejściowe	Nazwa postaci, elementy wyglądu postaci (głowa, tułów, nogi).
Źródło danych wejściowych	Formularz kreatora postaci w aplikacji webowej.
Wynik	Zapisanie postaci użytkownika w bazie danych i umożliwienie jej wykorzystania w aplikacji, przekierowanie użytkownika na stronę główną aplikacji
Wymagania	Użytkownik musi być zalogowany (posiadać ważny token JWT). Nazwa postaci musi spełniać kryteria walidacji. Użytkownik musi mieć prawo do edycji postaci.
Przeznaczenie	Personalizacja wyglądu postaci użytkownika w aplikacji.
Warunek wstępny	Użytkownik jest zalogowany i posiada aktywne konto. Posiada prawo do edycji postaci.
Warunek końcowy	Postać użytkownika została zapisana w bazie danych.
Efekty uboczne	Aktualizacja rekordu użytkownika w bazie danych. Możliwy wzrost obciążenia serwera w przypadku wielu zapisów.
Powód wywołania funkcji	Użytkownik chce zapisać swoją postać, aby móc z niej korzystać w aplikacji.

Tabela 3.5 Tabela wymagań funkcjonalnych — resetowanie hasła (źródło: opracowanie własne)

Cecha	Opis
Nazwa funkcji	Resetowanie hasła
Opis funkcji	Umożliwia użytkownikom zresetowanie hasła do konta poprzez wysłanie kodu resetującego na e-mail i ustawienie nowego hasła.
Dane wejściowe	Adres e-mail, kod resetujący, nowe hasło, potwierdzenie nowego hasła.
Źródło danych wejściowych	Formularz resetowania hasła w aplikacji webowej.

Wynik	Użytkownik otrzymuje kod resetujący na e-mail, a po jego wprowadzeniu może ustawić nowe hasło.
Wymagania	Użytkownik musi posiadać zarejestrowane konto; adres e-mail musi być poprawny; kod resetujący musi być poprawny; nowe hasło musi spełniać wymagania dotyczące bezpieczeństwa.
Przeznaczenie	Umożliwienie użytkownikom zresetowania hasła w przypadku jego zapomnienia lub utraty.
Warunek wstępny	Konto użytkownika istnieje, dostęp do internetu i strony aplikacji, użytkownik zna swój adres e-mail.
Warunek końcowy	Użytkownik zresetował hasło, nowe hasło zostało zapisane w bazie danych.
Efekty uboczne	Wysłanie e-maila z kodem resetującym, aktualizacja hasła użytkownika w bazie danych, usunięcie kodu resetującego po użyciu.
Powód wywołania funkcji	Chęć użytkownika do zresetowania hasła w przypadku jego zapomnienia lub utraty.

Tabela 3.6 Tabela wymagań funkcjonalnych — zakup nowego serduszka (źródło: opracowanie własne)

Cecha	Opis
Nazwa funkcji	Zakup nowego serduszka
Opis funkcji	Umożliwia użytkownikom zakup dodatkowego serduszka, które zwiększa ich maksymalny limit HP.
Dane wejściowe	Token uwierzytelniający, akcja "buy_heart".
Źródło danych wejściowych	Formularz zakupu serduszka w aplikacji webowej.
Wynik	Użytkownik kupuje serduszko, jego maksymalny limit HP wzrasta, a liczba dostępnych diamencików maleje.
Wymagania	Użytkownik musi być zalogowany; użytkownik musi posiadać wystarczającą ilość diamencików; token uwierzytelniający musi być poprawny.
Przeznaczenie	Umożliwienie użytkownikom zwiększenia maksymalnego limitu HP poprzez zakup dodatkowego serduszka.
Warunek wstępny	Konto użytkownika istnieje, użytkownik jest zalogowany, dostęp do internetu i strony aplikacji, użytkownik posiada wystarczającą ilość diamencików.
Warunek końcowy	Użytkownik zakupił serduszko, maksymalny limit HP użytkownika wzrósł, liczba dostępnych diamencików użytkownika zmniejszyła się.
Efekty uboczne	Aktualizacja liczby diamencików użytkownika, aktualizacja maksymalnego limitu HP użytkownika, zapisanie zmian w bazie danych.
Powód wywołania funkcji	Chęć użytkownika do zwiększenia maksymalnego limitu HP poprzez zakup dodatkowego serduszka.

Tabela 3.7 Tabela wymagań funkcjonalnych — zakup przedmiotu w sklepie (źródło: opracowanie własne)

Cecha	Opis
Nazwa funkcji	Zakup przedmiotu w sklepie
Opis funkcji	Umożliwia użytkownikom zakup przedmiotów w sklepie, które mogą być używane do personalizacji postaci.
Dane wejściowe	Token uwierzytelniający, ID przedmiotu do zakupu.
Źródło danych wejściowych	Formularz zakupu przedmiotu w aplikacji webowej.
Wynik	Użytkownik kupuje przedmiot, jego liczba monet maleje, a przedmiot zostaje dodany do jego ekwipunku.
Wymagania	Użytkownik musi być zalogowany; użytkownik musi posiadać wystarczającą ilość monet; token uwierzytelniający musi być poprawny; użytkownik musi spełniać wymagany poziom dla przedmiotu.
Przeznaczenie	Umożliwienie użytkownikom zakupu przedmiotów w sklepie, które mogą być używane do personalizacji postaci.
Warunek wstępny	Konto użytkownika istnieje, użytkownik jest zalogowany, dostęp do internetu i strony aplikacji, użytkownik posiada wystarczającą ilość monet.
Warunek końcowy	Użytkownik zakupił przedmiot, liczba monet użytkownika zmniejszyła się, przedmiot został dodany do ekwipunku użytkownika.
Efekty uboczne	Aktualizacja liczby monet użytkownika, dodanie przedmiotu do ekwipunku użytkownika, zapisanie zmian w bazie danych.
Powód wywołania funkcji	Chęć użytkownika do zakupu przedmiotu w sklepie i personalizacji swojej postaci.

Tabela 3.8 Tabela wymagań funkcjonalnych — zakup edycji postaci (źródło: opracowanie własne)

Cecha	Opis
Nazwa funkcji	Zakup edycji postaci
Opis funkcji	Umożliwia użytkownikom zakup opcji edycji postaci, co pozwala na modyfikację wyglądu postaci.
Dane wejściowe	Token uwierzytelniający, akcja "buy_edit_character".
Źródło danych wejściowych	Formularz zakupu edycji postaci w aplikacji webowej.
Wynik	Użytkownik kupuje opcję edycji postaci, jego liczba diamencików maleje, a opcja edycji postaci zostaje odblokowana.

Wymagania	Użytkownik musi być zalogowany; użytkownik musi posiadać wystarczającą ilość diamencików; token uwierzytelniający musi być poprawny.
Przeznaczenie	Umożliwienie użytkownikom modyfikacji wyglądu postaci poprzez zakup opcji edycji postaci.
Warunek wstępny	Konto użytkownika istnieje, użytkownik jest zalogowany, dostęp do internetu i strony aplikacji, użytkownik posiada wystarczającą ilość diamencików.
Warunek końcowy	Użytkownik zakupił opcję edycji postaci, liczba diamencików użytkownika zmniejszyła się, opcja edycji postaci została odblokowana.
Efekty uboczne	Aktualizacja liczby diamencików użytkownika, odblokowanie opcji edycji postaci, zapisanie zmian w bazie danych.
Powód wywołania funkcji	Chęć użytkownika do modyfikacji wyglądu postaci poprzez zakup opcji edycji postaci.

Tabela 3.9 Tabela wymagań funkcjonalnych — ubranie postaci (źródło: opracowanie własne)

Cecha	Opis
Nazwa funkcji	Ubranie przedmiotu
Opis funkcji	Umożliwia użytkownikom ubranie przedmiotu z garderoby, co zmienia wygląd postaci.
Dane wejściowe	Token uwierzytelniający, ID przedmiotu, akcja "equip" lub "unequip".
Źródło danych wejściowych	Formularz wyboru przedmiotu w aplikacji webowej.
Wynik	Użytkownik ubiera przedmiot, a wygląd postaci zostaje zaktualizowany.
Wymagania	Użytkownik musi być zalogowany; użytkownik musi posiadać przedmiot w garderobie; token uwierzytelniający musi być poprawny.
Przeznaczenie	Umożliwienie użytkownikom zmiany wyglądu postaci poprzez ubranie przedmiotu z garderoby.
Warunek wstępny	Konto użytkownika istnieje, użytkownik jest zalogowany, dostęp do internetu i strony aplikacji, użytkownik posiada przedmiot w garderobie.
Warunek końcowy	Użytkownik ubrał przedmiot, wygląd postaci został zaktualizowany, zmiany zostały zapisane w bazie danych.
Efekty uboczne	Aktualizacja wyglądu postaci, zapisanie zmian w bazie danych, ewentualne zadania związane z ubraniem przedmiotu.
Powód wywołania funkcji	Chęć użytkownika do zmiany wyglądu postaci poprzez ubranie przedmiotu z garderoby.

Tabela 3.10 Tabela wymagań się (źródło: opracowanie własne)

Cecha	Opis
Nazwa funkcji	Wylogowanie się
Opis funkcji	Umożliwia użytkownikom wylogowanie się z aplikacji, co powoduje usunięcie tokenu uwierzytelniającego.
Dane wejściowe	Brak (akcja wylogowania jest inicjowana przez kliknięcie przycisku).
Źródło danych wejściowych	Przycisk wylogowania w aplikacji webowej.
Wynik	Użytkownik zostaje wylogowany, token uwierzytelniający zostaje usunięty z lokalnego magazynu.
Wymagania	Użytkownik musi być zalogowany.
Przeznaczenie	Umożliwienie użytkownikom zakończenia sesji i wylogowania się z aplikacji.
Warunek wstępny	Konto użytkownika istnieje, użytkownik jest zalogowany, dostęp do internetu i strony aplikacji.
Warunek końcowy	Użytkownik wylogowany, token uwierzytelniający usunięty, użytkownik przekierowany na stronę logowania.
Efekty uboczne	Usunięcie tokenu uwierzytelniającego z lokalnego magazynu, przekierowanie na stronę logowania.
Powód wywołania funkcji	Chęć użytkownika do zakończenia sesji i wylogowania się z aplikacji.

Tabela 3.11 Tabela wymagań funkcjonalnych — odpowiadanie na pytania (źródło: opracowanie własne)

Cecha	Opis
Nazwa funkcji	Odpowiadanie na pytania
Opis funkcji	Umożliwia użytkownikom odpowiadanie na pytania z różnych dziedzin, takich jak programowanie, matematyka, nauki ścisłe i angielski.
Dane wejściowe	Token uwierzytelniający, ID pytania, odpowiedź użytkownika.
Źródło danych wejściowych	Formularz odpowiedzi na pytanie w aplikacji webowej.
Wynik	Użytkownik odpowiada na pytanie, a system ocenia odpowiedź jako poprawną lub niepoprawną, aktualizując odpowiednio statystyki użytkownika.
Wymagania	Użytkownik musi być zalogowany; token uwierzytelniający musi być poprawny; użytkownik musi mieć wystarczającą ilość HP, aby odpowiedzieć na pytanie.
Przeznaczenie	Umożliwienie użytkownikom sprawdzenia swojej wiedzy i zdobywania nagród za poprawne odpowiedzi.

Warunek wstępny	Konto użytkownika istnieje, użytkownik jest zalogowany, dostęp do internetu i strony aplikacji, użytkownik ma wystarczającą ilość HP.
Warunek końcowy	Użytkownik odpowiedział na pytanie, statystyki użytkownika zostały zaktualizowane, odpowiedź została oceniona jako poprawna lub niepoprawna.
Efekty uboczne	Aktualizacja statystyk użytkownika (ELO, monety, doświadczenie, HP), zapisanie odpowiedzi w bazie danych, ewentualne zadania związane z odpowiedzią na pytanie.
Powód wywołania funkcji	Chęć użytkownika do sprawdzenia swojej wiedzy i zdobywania nagród za poprawne odpowiedzi.

Tabela 3.12 Tabela wymagań funkcjonalnych — dodaj nowe funkcje (źródło: opracowanie własne)

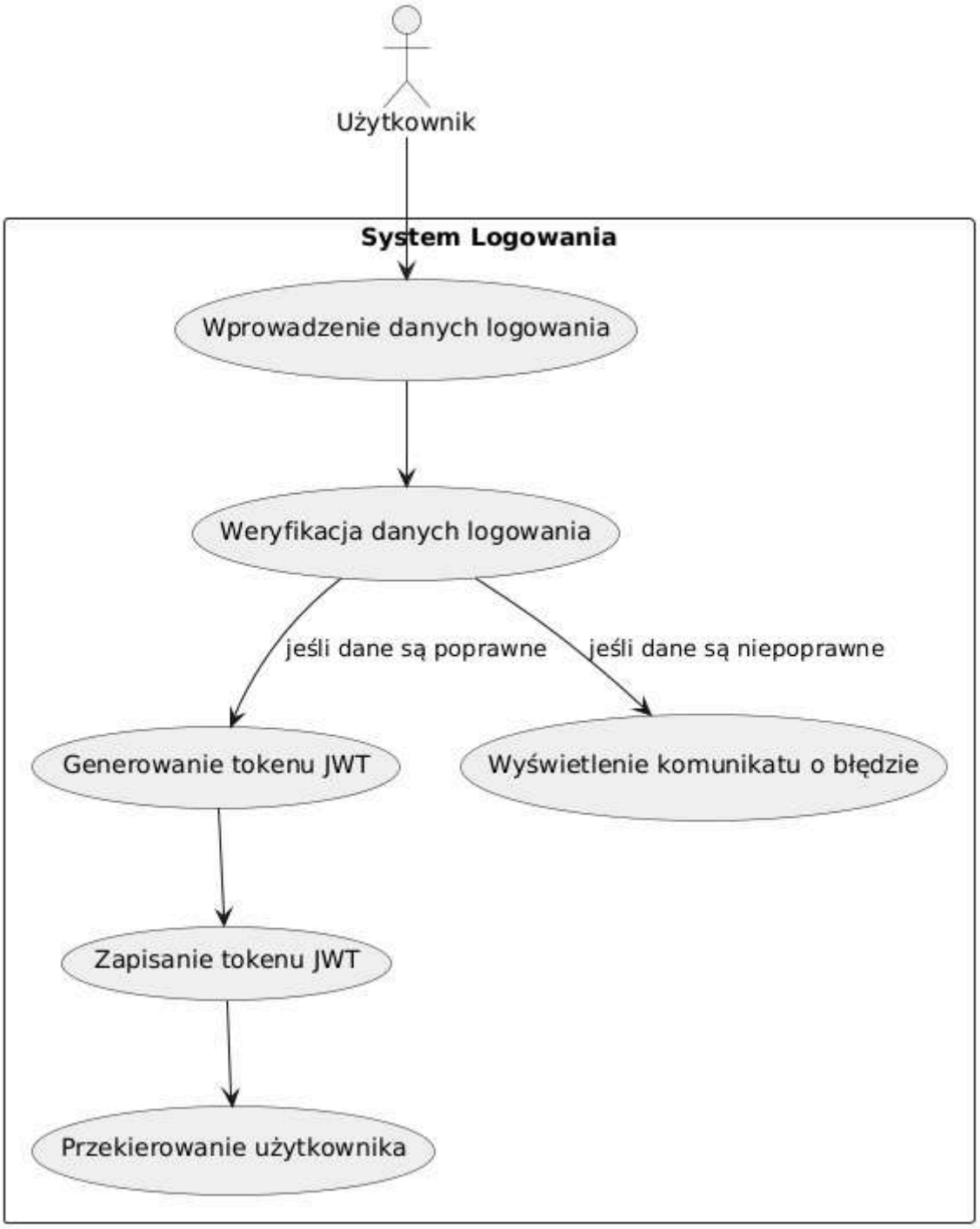
Cecha	Opis
Nazwa funkcji	Dodaj nowe pytanie
Opis funkcji	Umożliwia administratorom dodawanie nowych pytań do bazy danych dla różnych dziedzin, takich jak matematyka, programowanie, nauki ścisłe i język angielski.
Dane wejściowe	Token uwierzytelniający, pytanie, poprawna odpowiedź, błędne odpowiedzi, poziom trudności, data, ID pytania.
Źródło danych wejściowych	Formularz dodawania pytania w panelu administracyjnym aplikacji webowej.
Wynik	Nowe pytanie zostaje dodane do bazy danych, a administrator otrzymuje potwierdzenie.
Wymagania	Administrator musi być zalogowany; token uwierzytelniający musi być poprawny; wszystkie pola formularza muszą być wypełnione.
Przeznaczenie	Umożliwienie administratorom dodawania nowych pytań do bazy danych, które będą używane w quizach i testach.
Warunek wstępny	Konto administratora istnieje, administrator jest zalogowany, dostęp do internetu i strony aplikacji.
Warunek końcowy	Nowe pytanie zostało dodane do bazy danych, administrator otrzymał potwierdzenie.
Efekty uboczne	Zapisanie nowego pytania w bazie danych, ewentualne zadania związane z dodaniem nowego pytania.
Powód wywołania funkcji	Chęć administratora do dodania nowych pytań do bazy danych, aby urozmaicić quizy i testy dla użytkowników.

Tabela 3.13 Tabela wymagań funkcjonalnych — dodaj nowy przedmiot do sklepu (źródło: opracowanie własne)

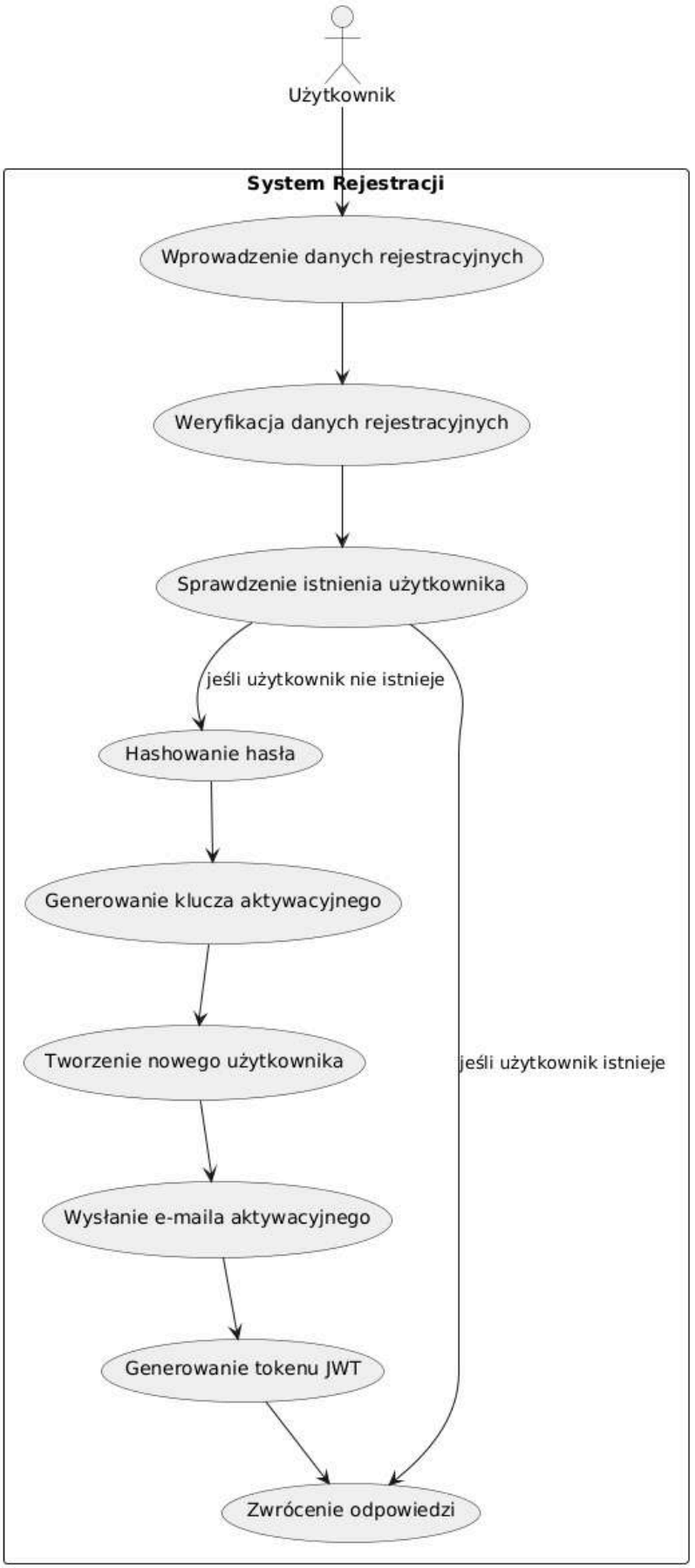
Cecha	Opis
Nazwa funkcji	Dodaj nowy przedmiot do sklepu

Opis funkcji	Umożliwia administratorom dodawanie nowych przedmiotów do sklepu, które mogą być kupowane przez użytkowników.
Dane wejściowe	Token uwierzytelniający, ID przedmiotu, typ przedmiotu, URL przedmiotu, wymagany poziom, koszt, nazwa przedmiotu.
Źródło danych wejściowych	Formularz dodawania przedmiotu w panelu administracyjnym aplikacji webowej.
Wynik	Nowy przedmiot zostaje dodany do sklepu, a administrator otrzymuje potwierdzenie.
Wymagania	Administrator musi być zalogowany; token uwierzytelniający musi być poprawny; wszystkie pola formularza muszą być wypełnione.
Przeznaczenie	Umożliwienie administratorom dodawania nowych przedmiotów do sklepu, które mogą być kupowane przez użytkowników.
Warunek wstępny	Konto administratora istnieje, administrator jest zalogowany, dostęp do internetu i strony aplikacji.
Warunek końcowy	Nowy przedmiot został dodany do sklepu, administrator otrzymał potwierdzenie.
Efekty uboczne	Zapisanie nowego przedmiotu w bazie danych, ewentualne zadania związane z dodaniem nowego przedmiotu.
Powód wywołania funkcji	Chęć administratora do dodania nowych przedmiotów do sklepu, aby urozmaicić ofertę dla użytkowników.

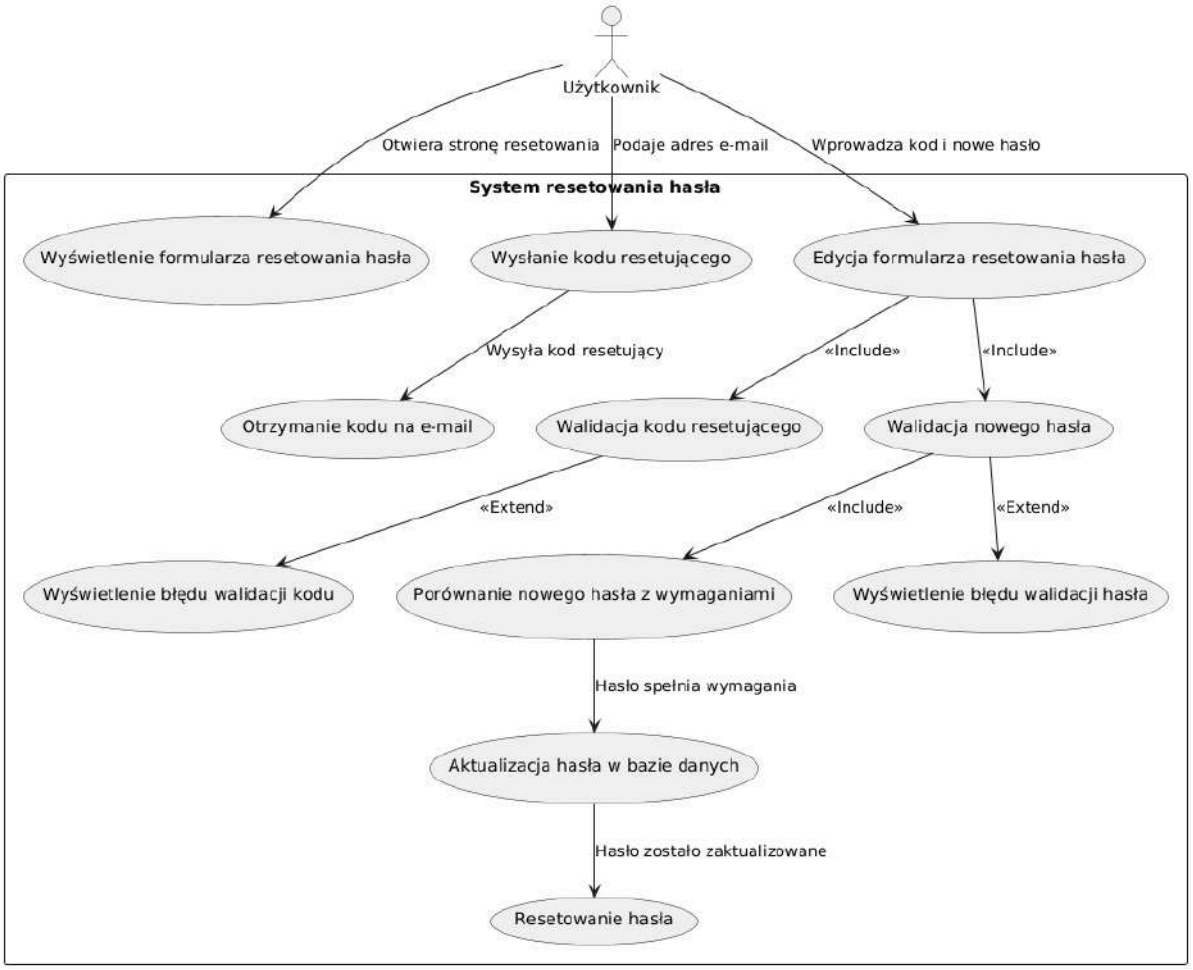
3.8. Diagramy akcji i przypadków użycia



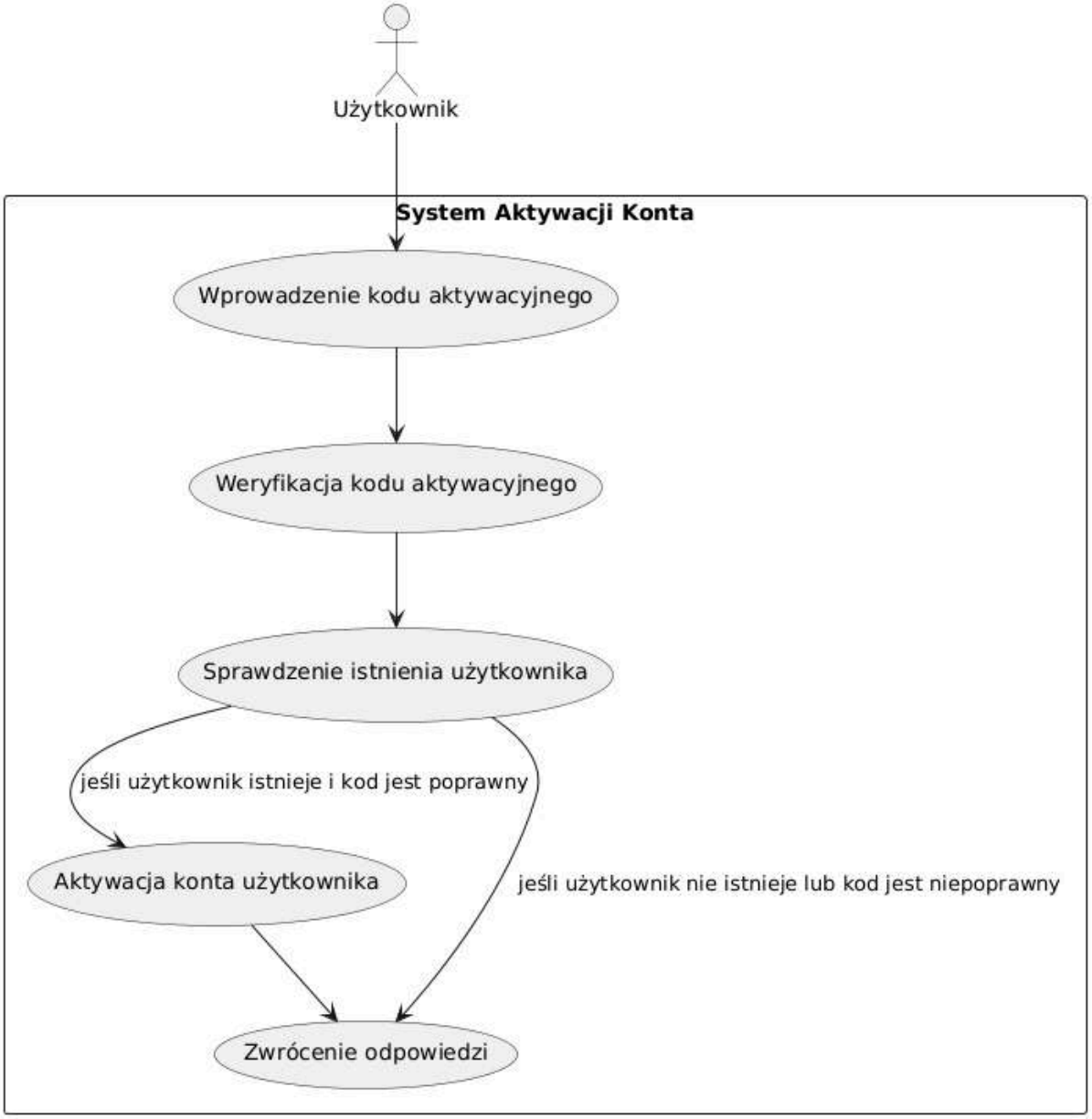
Rysunek 3.2 Diagram akcji — system logowania (źródło: opracowanie własne)



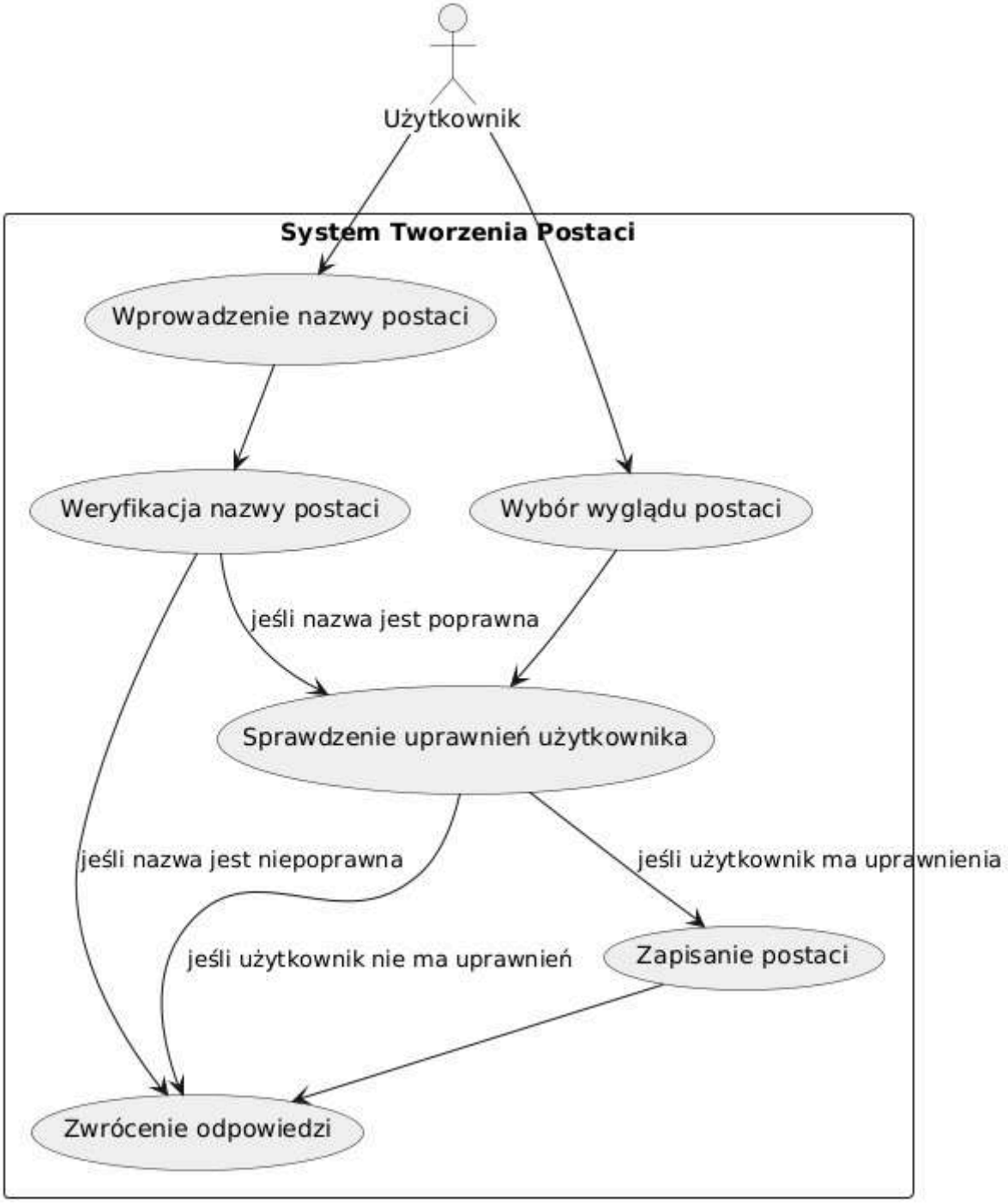
Rysunek 3.3 Diagram akcji — system rejestracji (źródło: opracowanie własne)



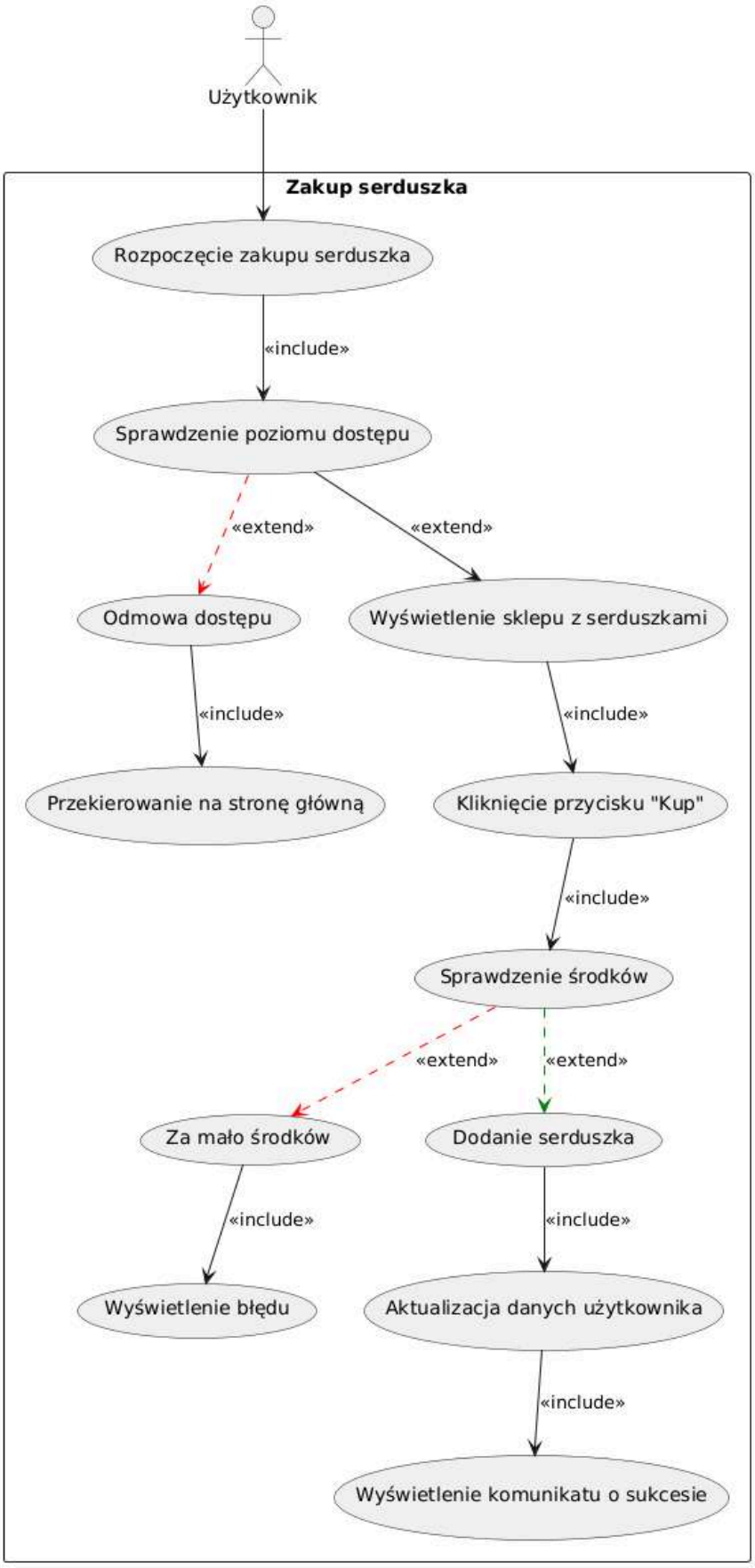
Rysunek 3.4 Diagram akcji — system resetowania hasła (źródło: opracowanie własne)



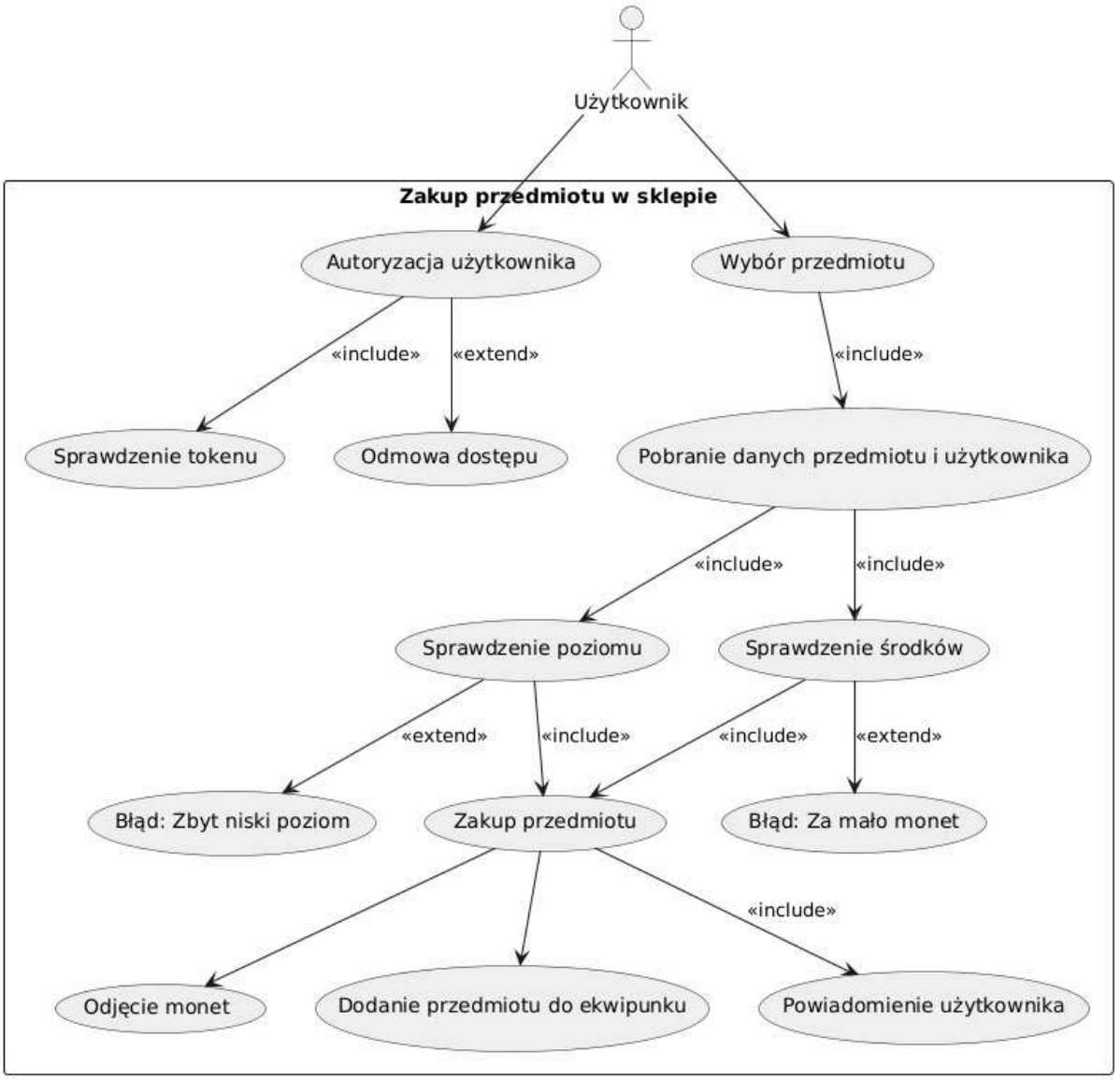
Rysunek 3.5 Diagram akcji — system aktywacji konta (źródło: opracowanie własne)



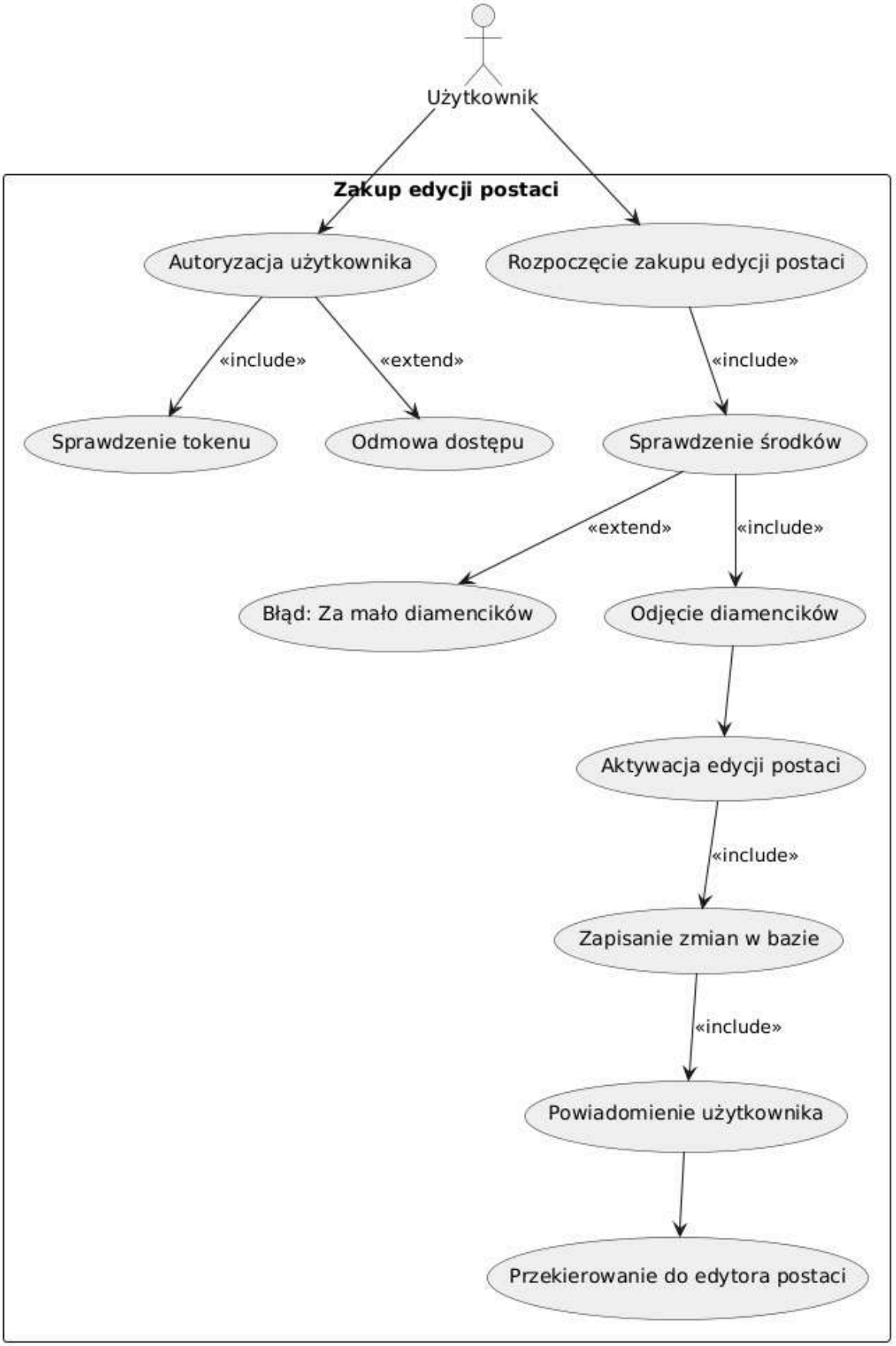
Rysunek 3.6 Diagram akcji — system tworzenia postaci (źródło: opracowanie własne)



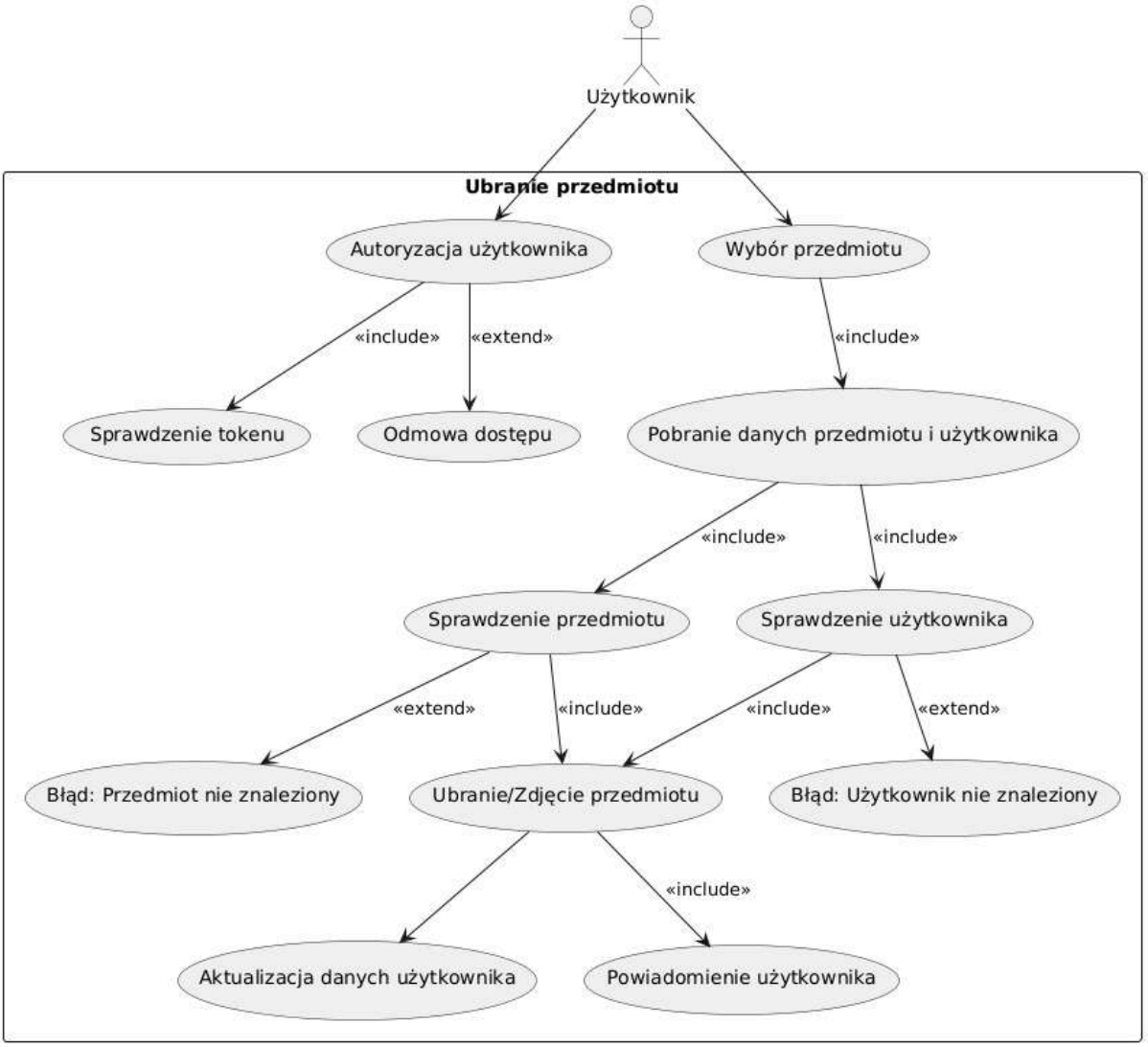
Rysunek 3.7 Diagram akcji — zakup serduszek (źródło: opracowanie własne)



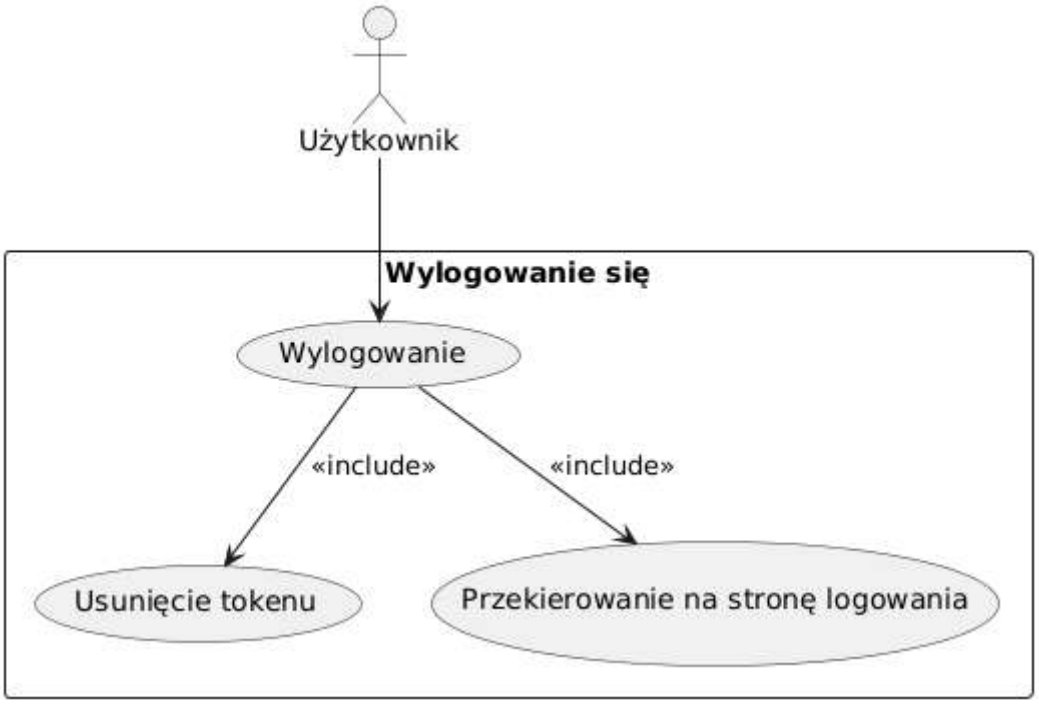
Rysunek 3.8 Diagram akcji — zakup przedmiotu w sklepie (źródło: opracowanie własne)



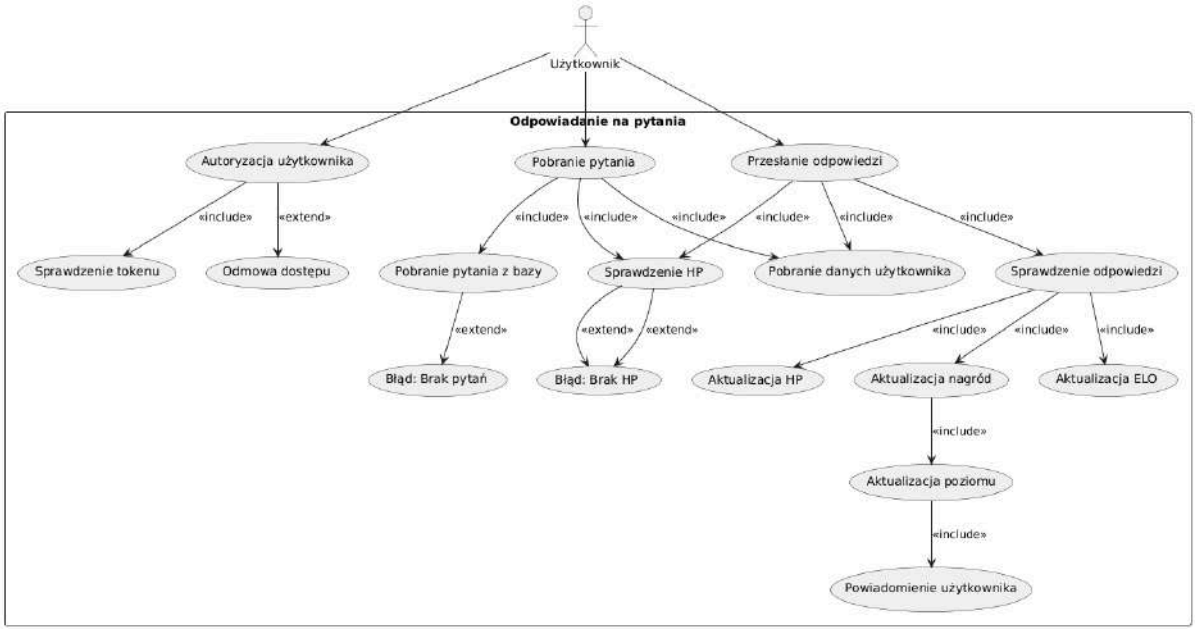
Rysunek 3.9 Diagram akcji — zakup edycji postaci (źródło: opracowanie własne)



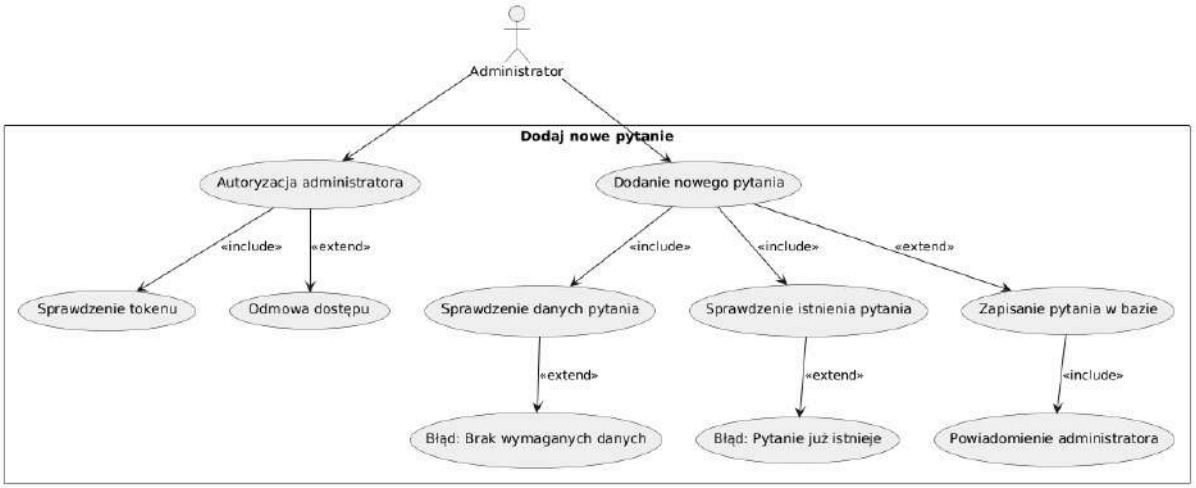
Rysunek 3.10 Diagram akcji — ubranie przedmiotu (źródło: opracowanie własne)



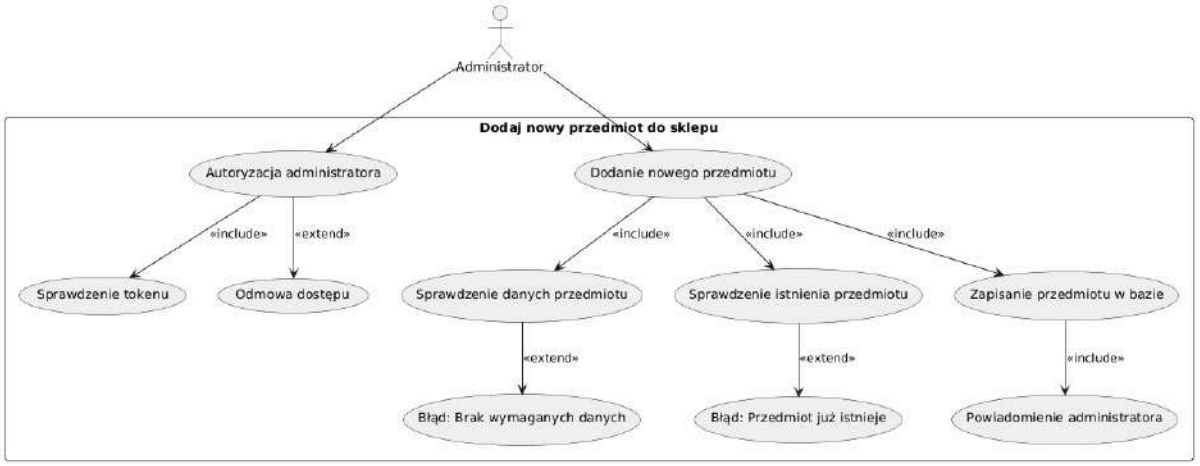
Rysunek 3.11 Diagram akcji — wylogowanie się (źródło: opracowanie własne)



Rysunek 3.12 Diagram akcji — odpowiadanie na pytanie (źródło: opracowanie własne)



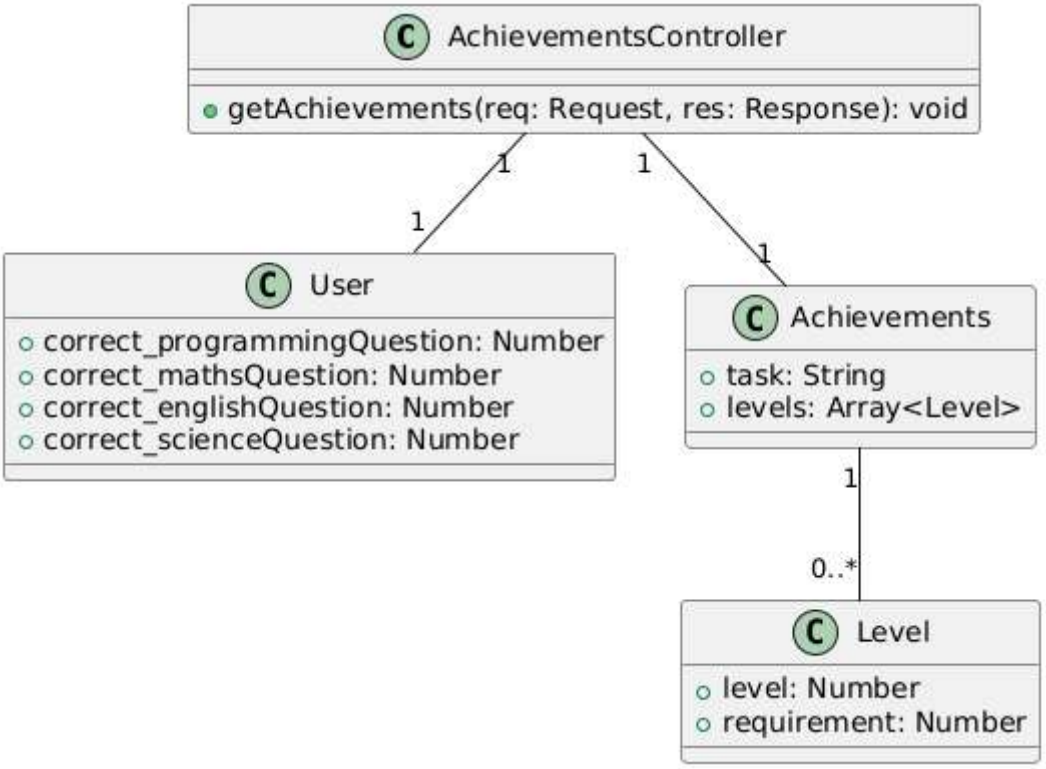
Rysunek 3.13 Diagram akcji — dodaj nowe pytanie (źródło: opracowanie własne)



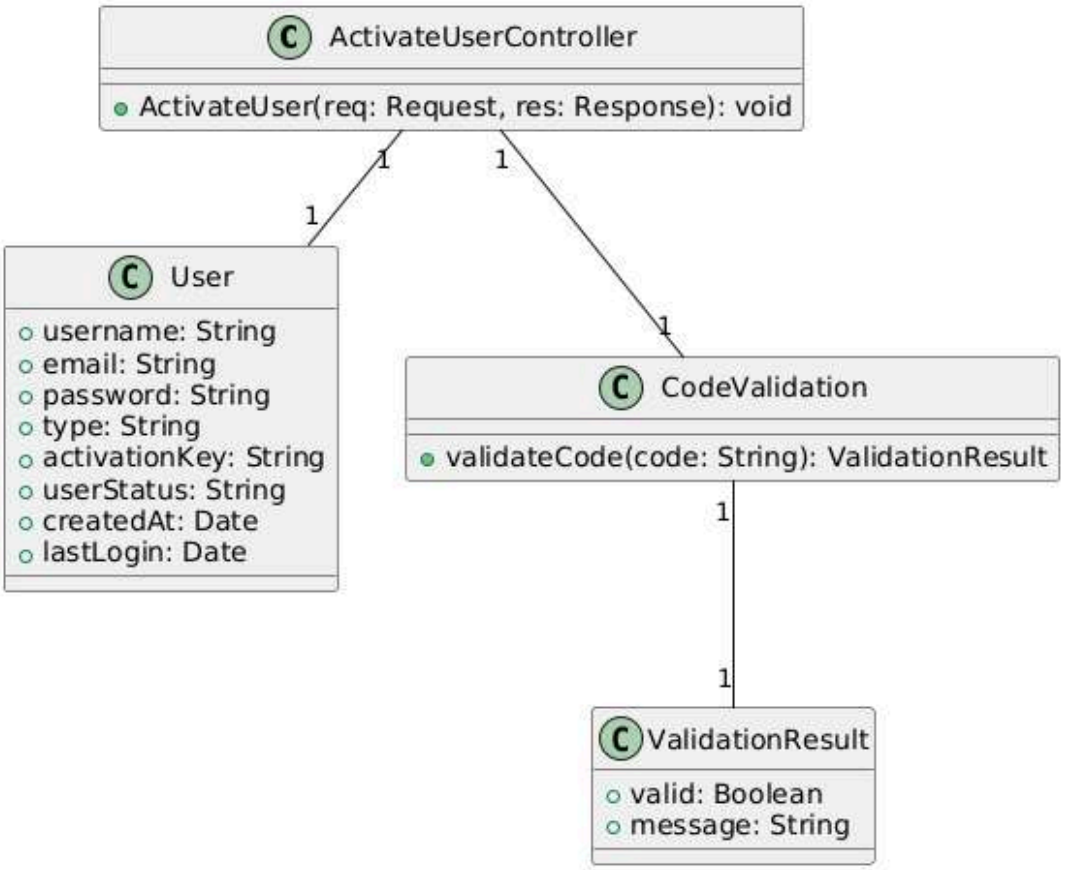
Rysunek 3.14 Diagram akcji — dodaj nowy przedmiot do sklepu (źródło: opracowanie własne)

3.9 Diagramy klas

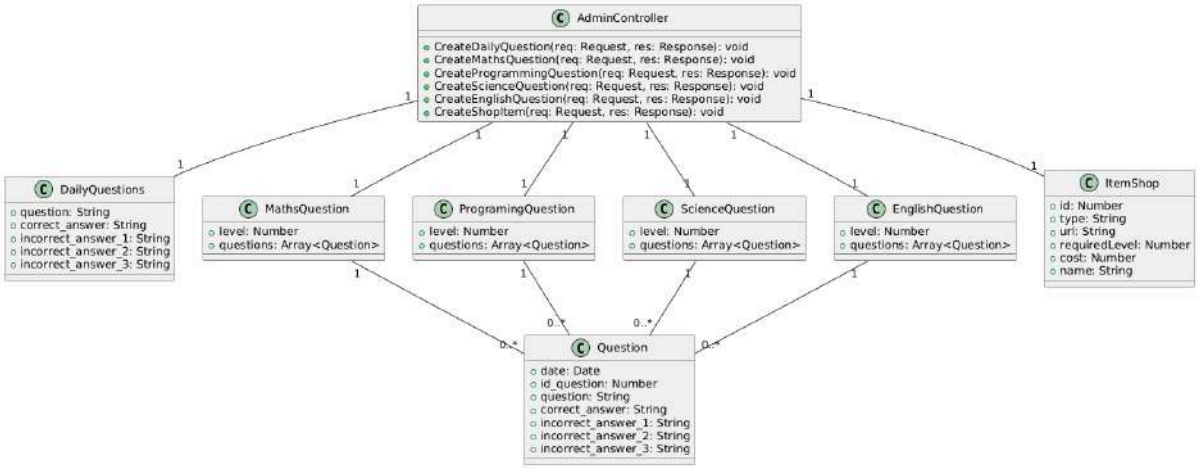
3.9.1 Diagramy klas dla plików z backendu



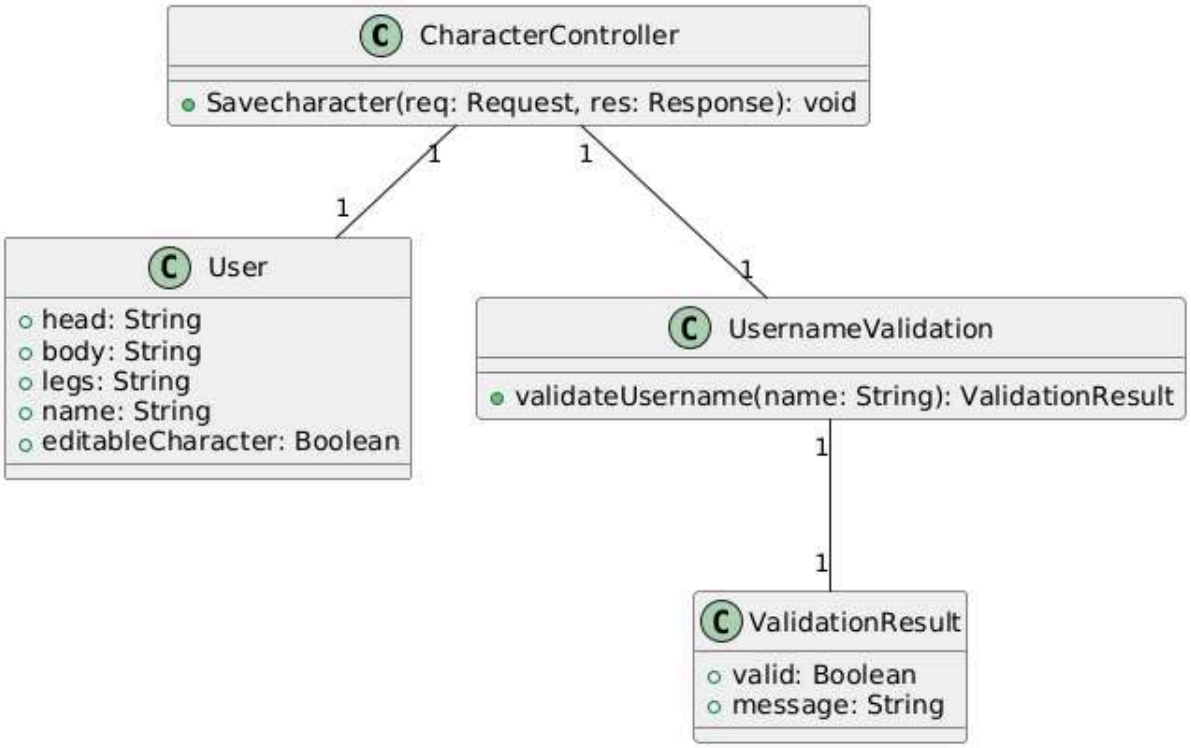
Rysunek 3.15 Diagram klas dla pliku AchievementsController.js (źródło: opracowanie własne)



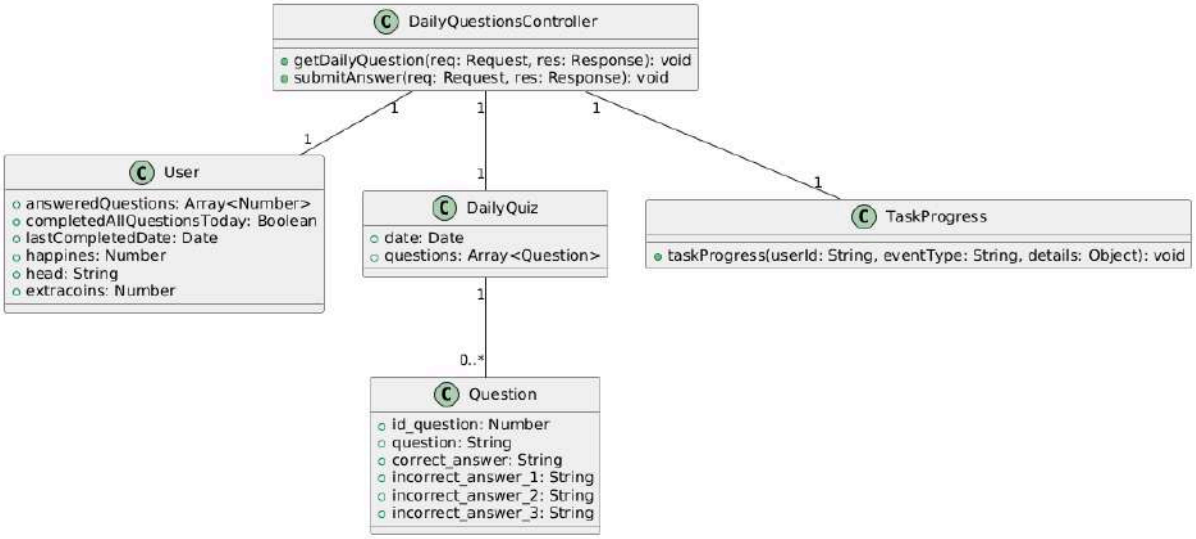
Rysunek 3.16 Diagram klas dla pliku ActivateUserController.js (źródło: opracowanie własne)



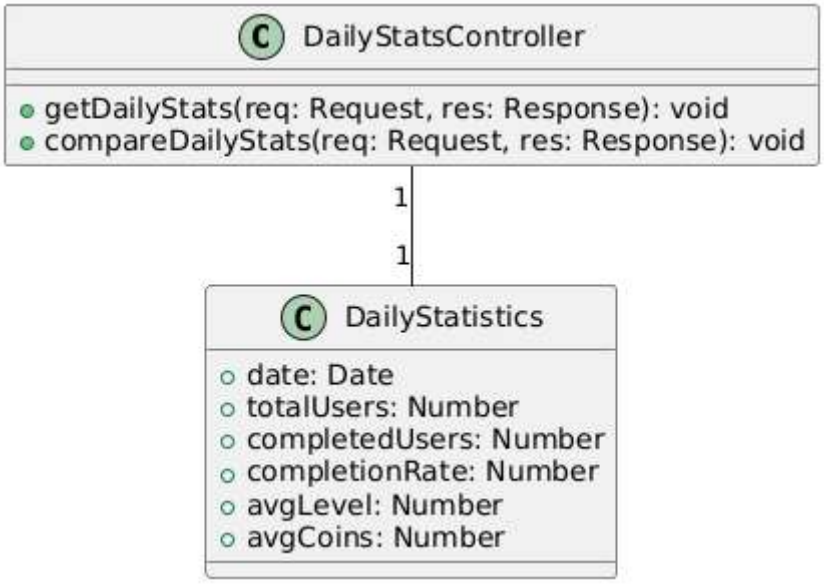
Rysunek 3.17 Diagram klas dla pliku AdminController.js (źródło: opracowanie własne)



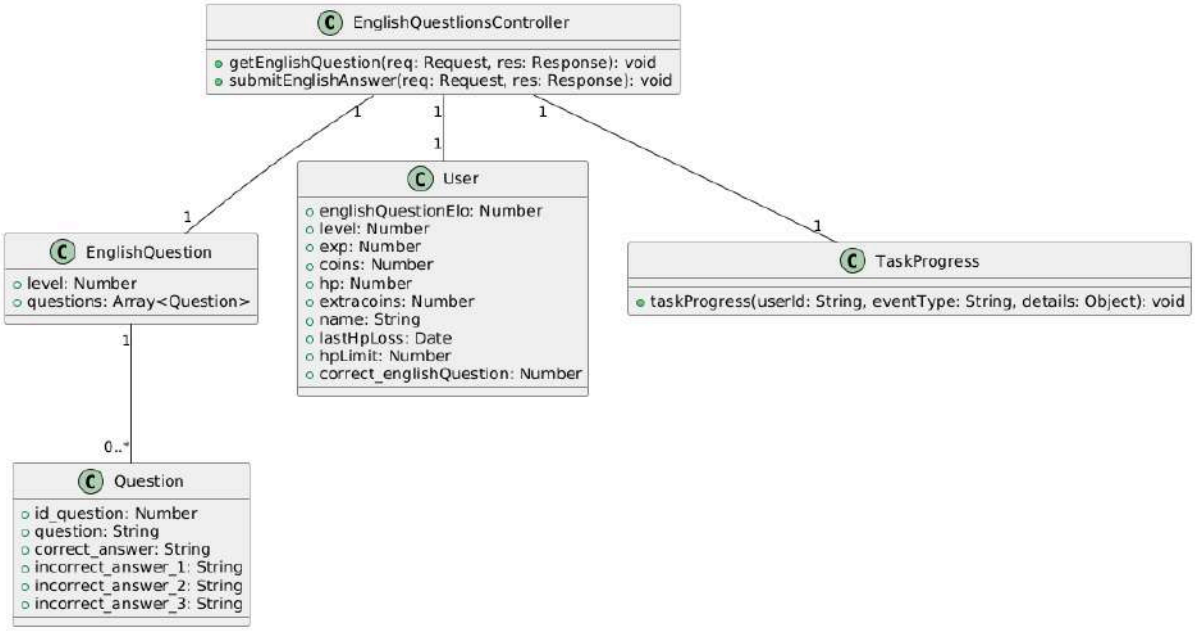
Rysunek 3.18 Diagram klas dla pliku CharacterController.js (źródło: opracowanie własne)



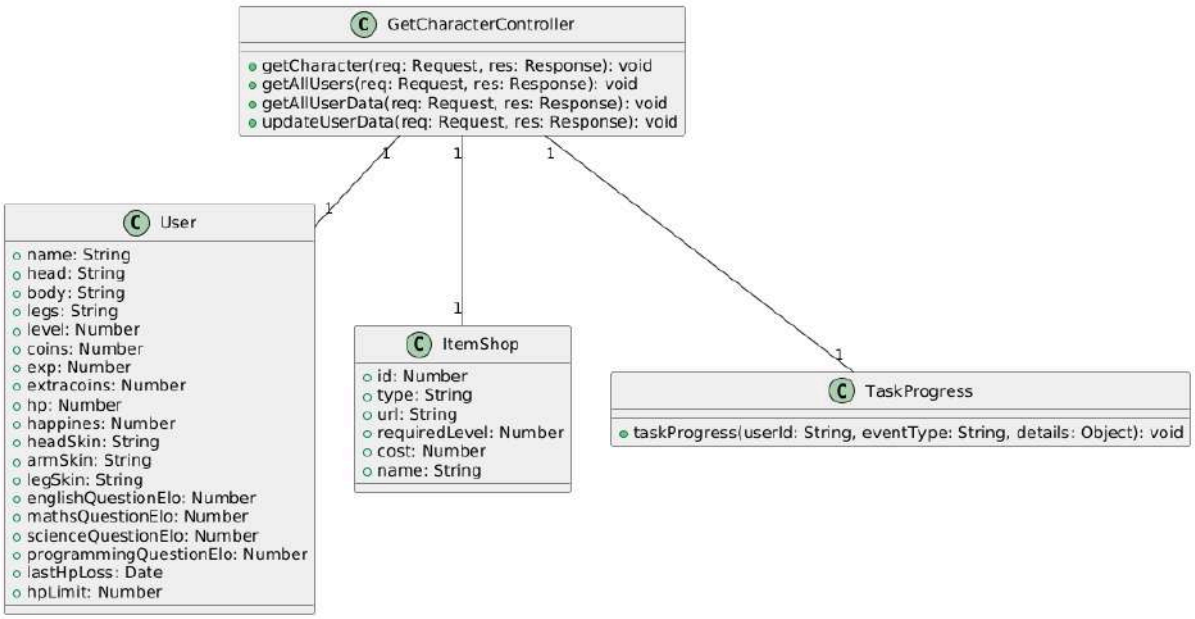
Rysunek 3.19 Diagram klas dla pliku DailyQuestionController.js (źródło: opracowanie własne)



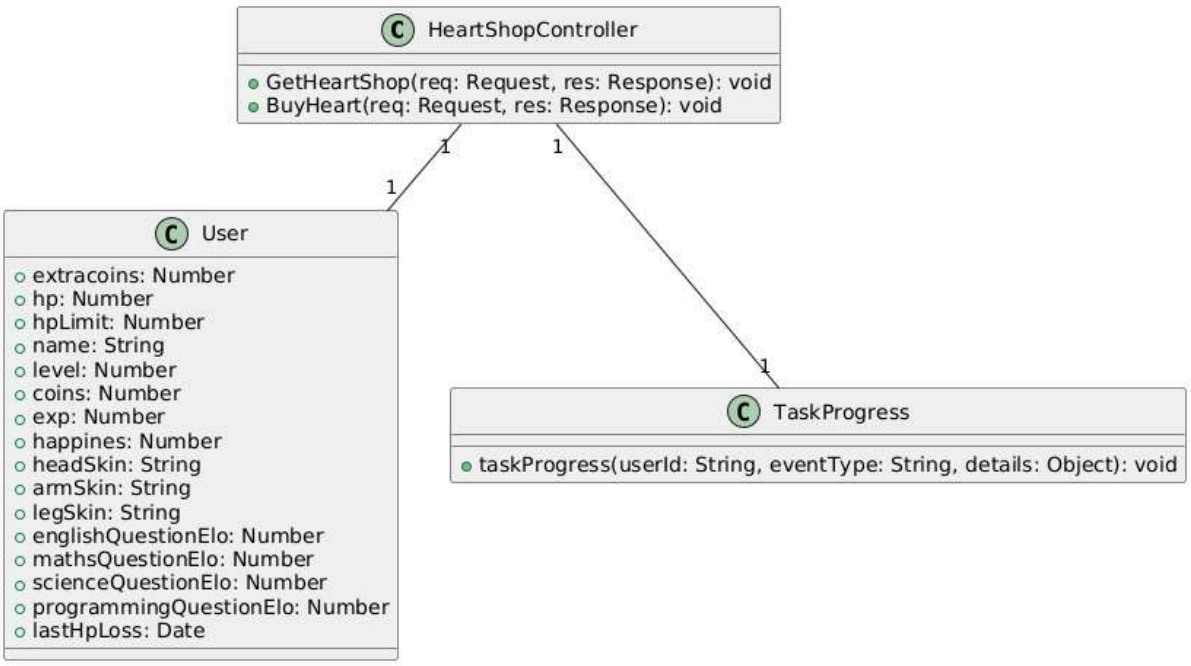
Rysunek 3.20 Diagram klas dla pliku DailyStatsController.js (źródło: opracowanie własne)



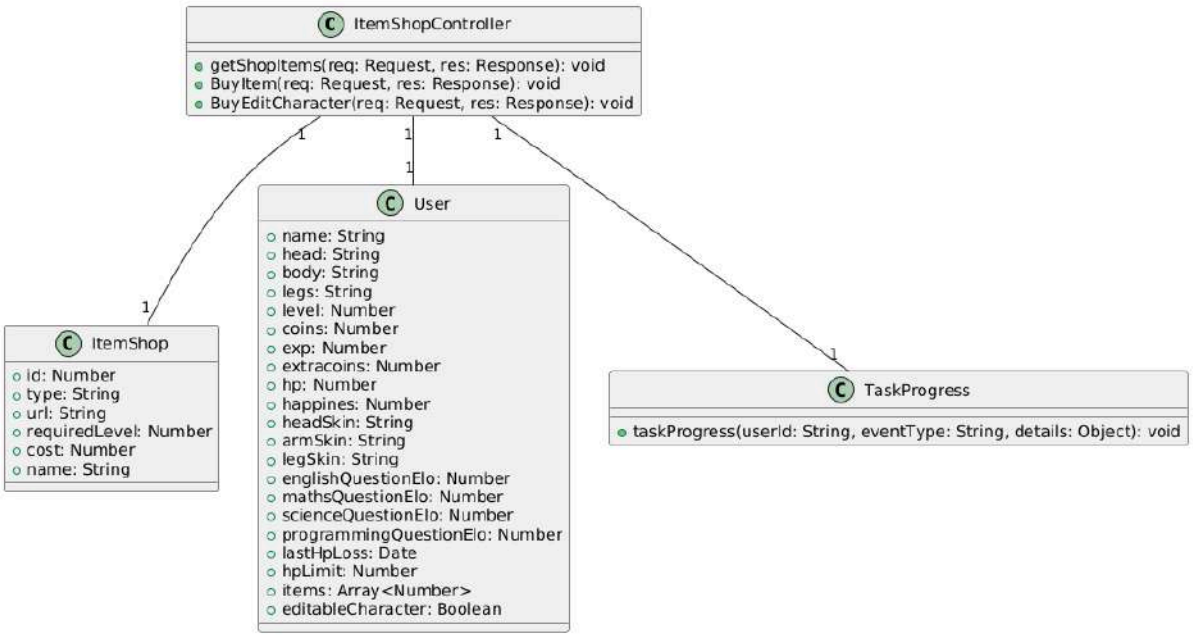
Rysunek 3.21 Diagram klas dla pliku EnglishQuestionController.js



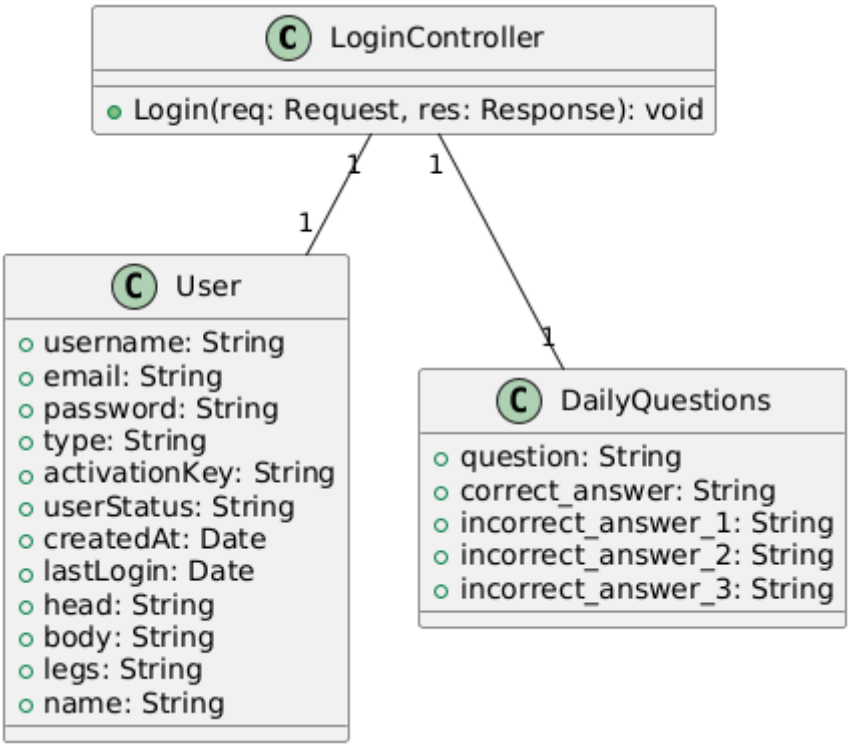
Rysunek 3.22 Diagram klas dla pliku GetCharacterController.js (źródło: opracowanie własne)



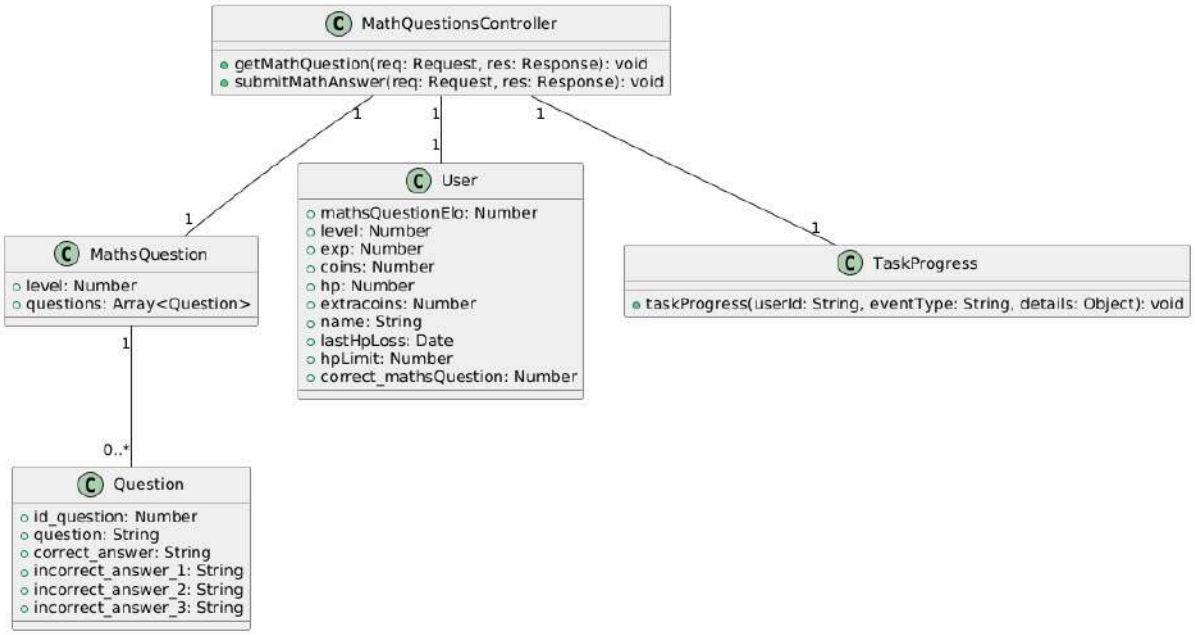
Rysunek 3.23 Diagram klas dla pliku HeartShopController.js (źródło: opracowanie własne)



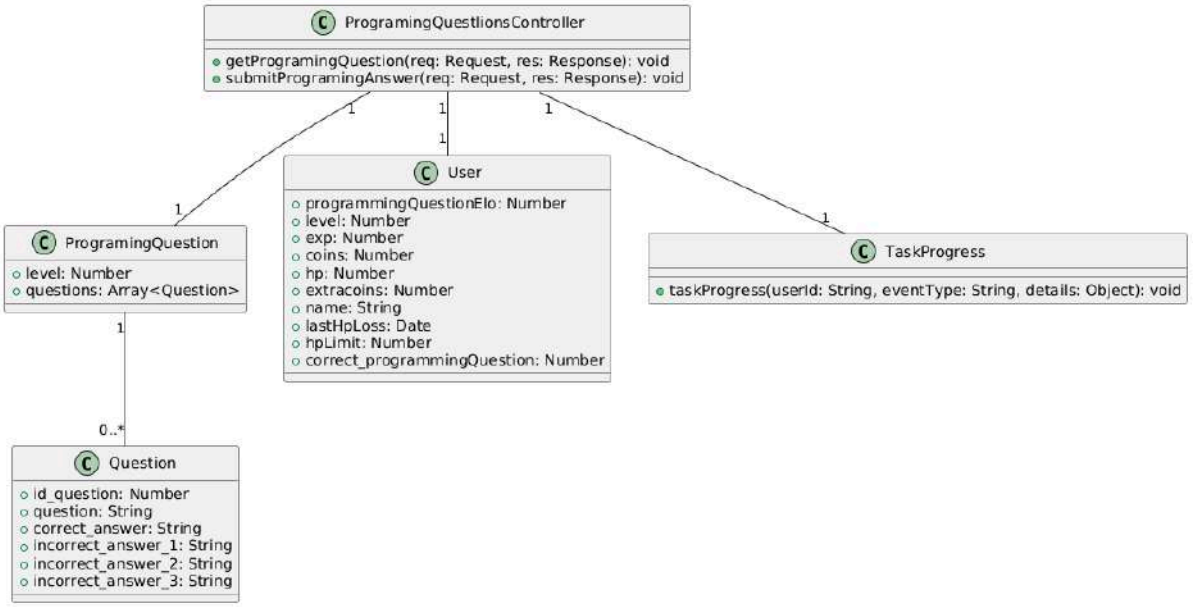
Rysunek 3.24 Diagram klas dla pliku ItemShopController.js (źródło: opracowanie własne)



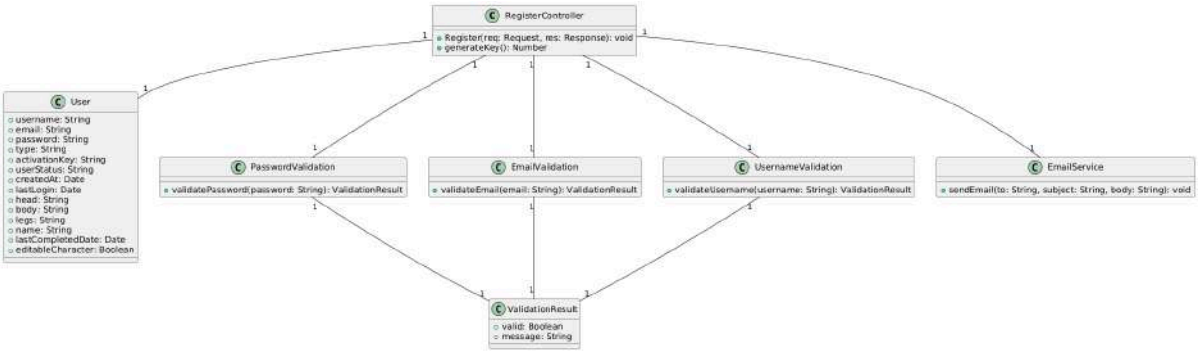
Rysunek 3.25 Diagram klas dla pliku LoginController.js (źródło: opracowanie własne)



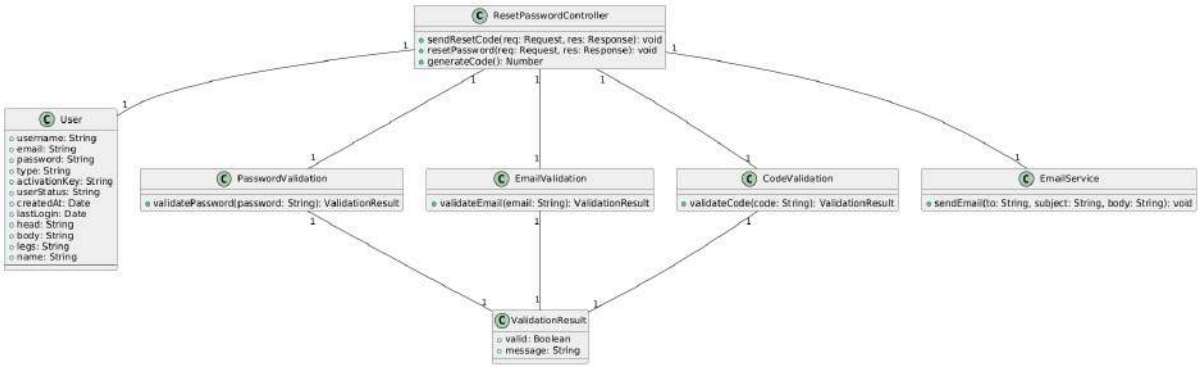
Rysunek 3.26 Diagram klas dla pliku MathQuestionsController.js (źródło: opracowanie własne)



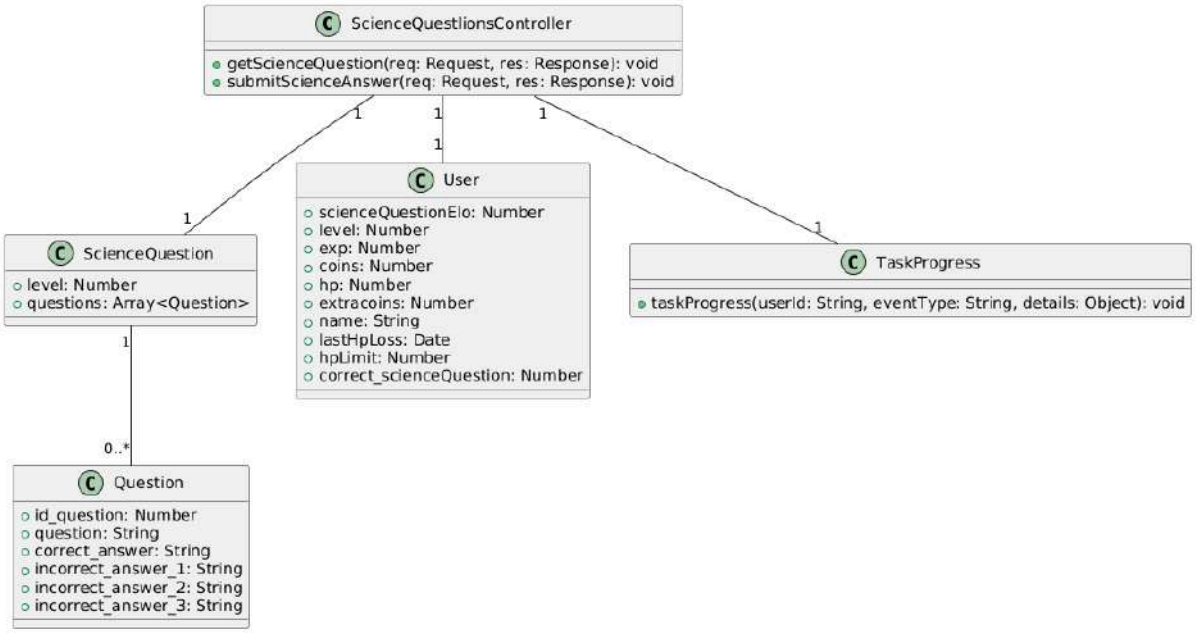
Rysunek 3.27 Diagram klas dla pliku ProgramingQuestionsController.js (źródło: opracowanie własne)



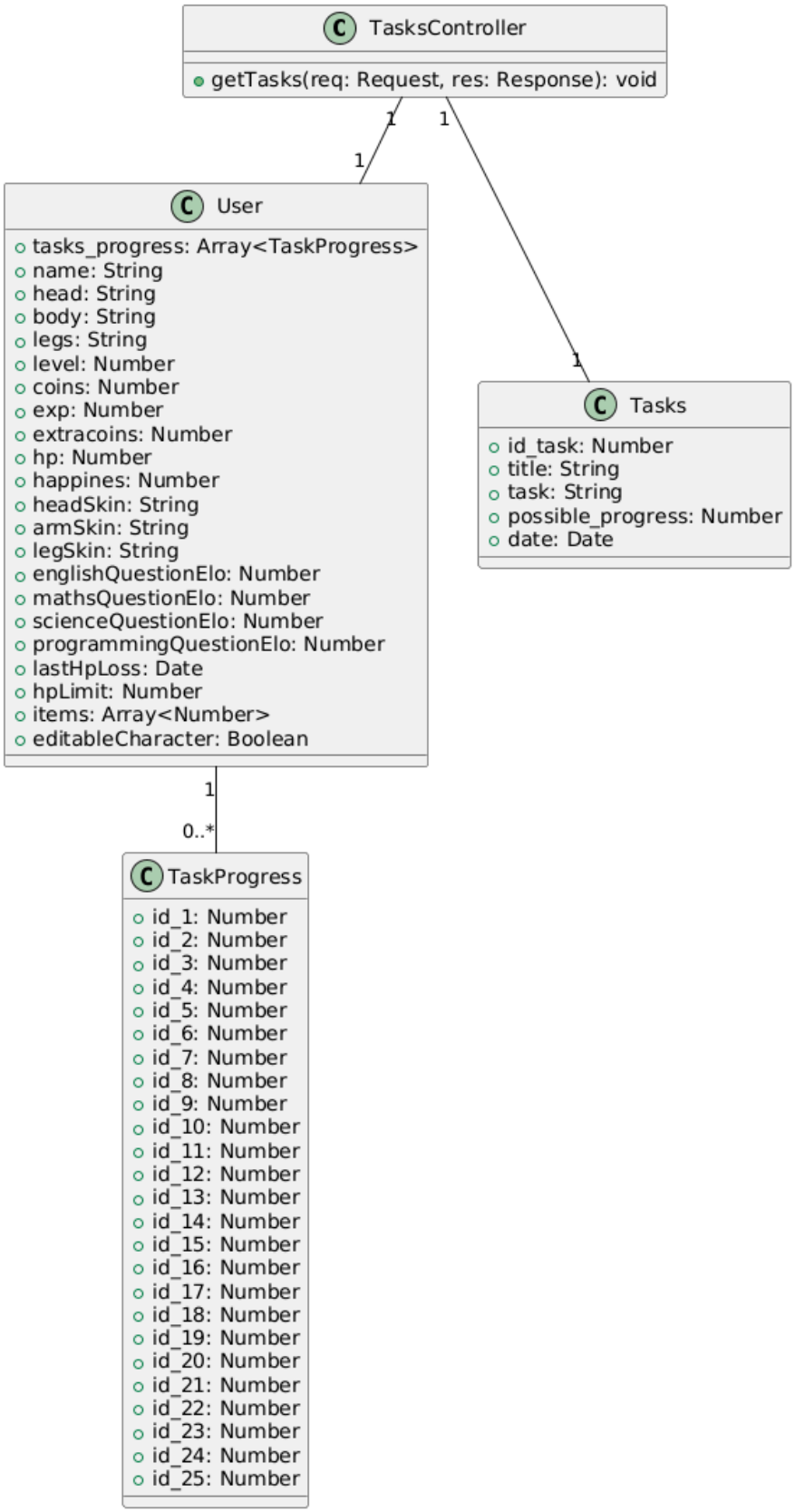
Rysunek 3.28 Diagram klas dla pliku RegisterController.js (źródło: opracowanie własne)



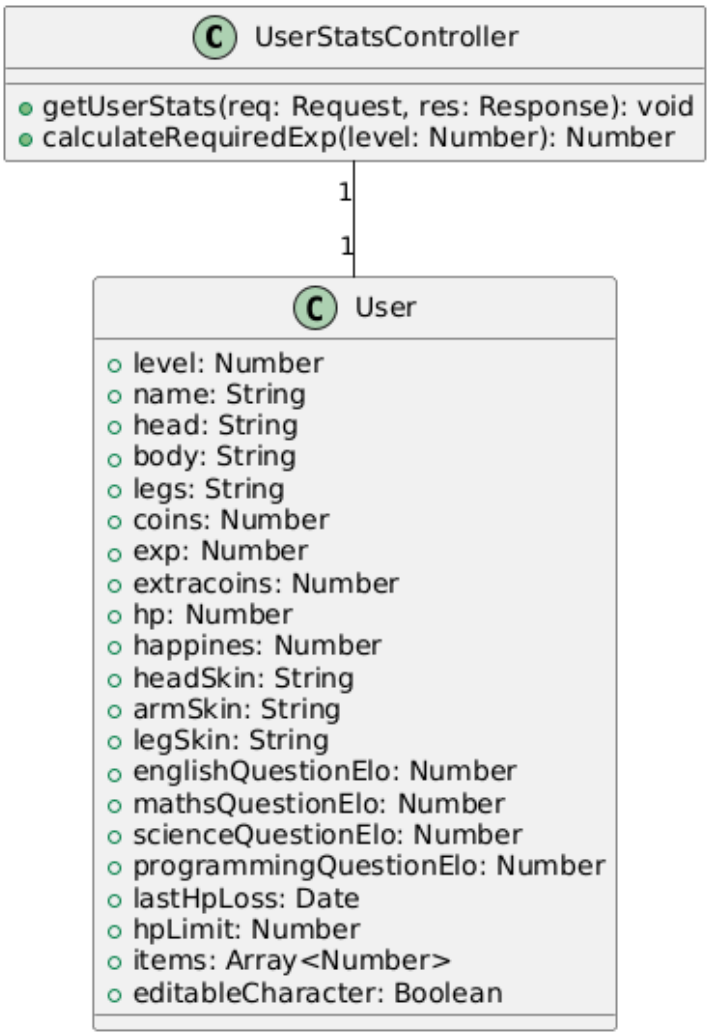
Rysunek 3.29 Diagram klas dla pliku ResetPasswordController.js (źródło: opracowanie własne)



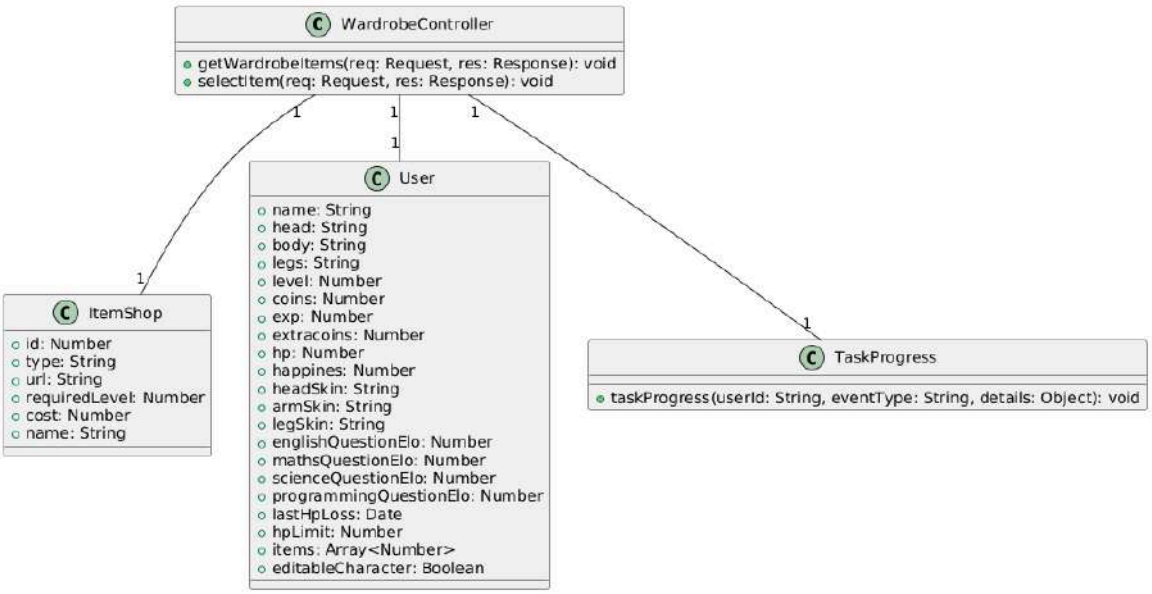
Rysunek 3.30 Diagram klas dla pliku ScienceQuestionsController.js (źródło: opracowanie własne)



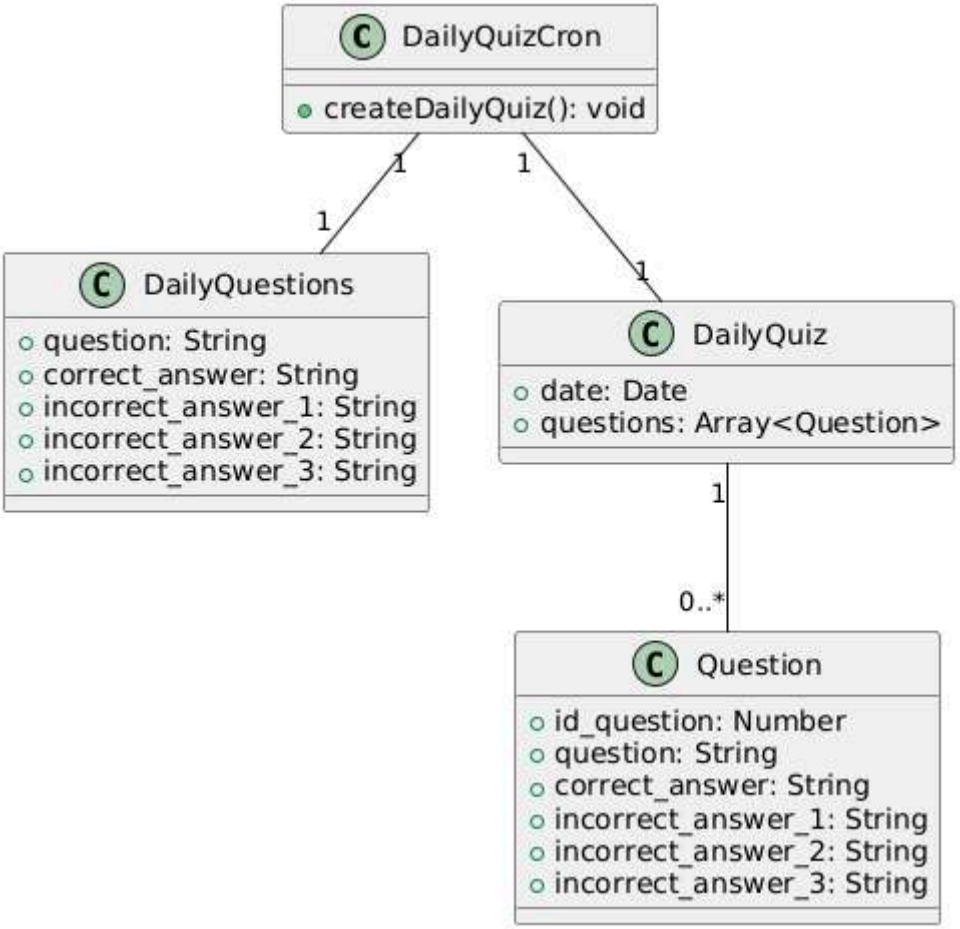
Rysunek 3.31 Diagram klas dla pliku TasksController.js (źródło: opracowanie własne)



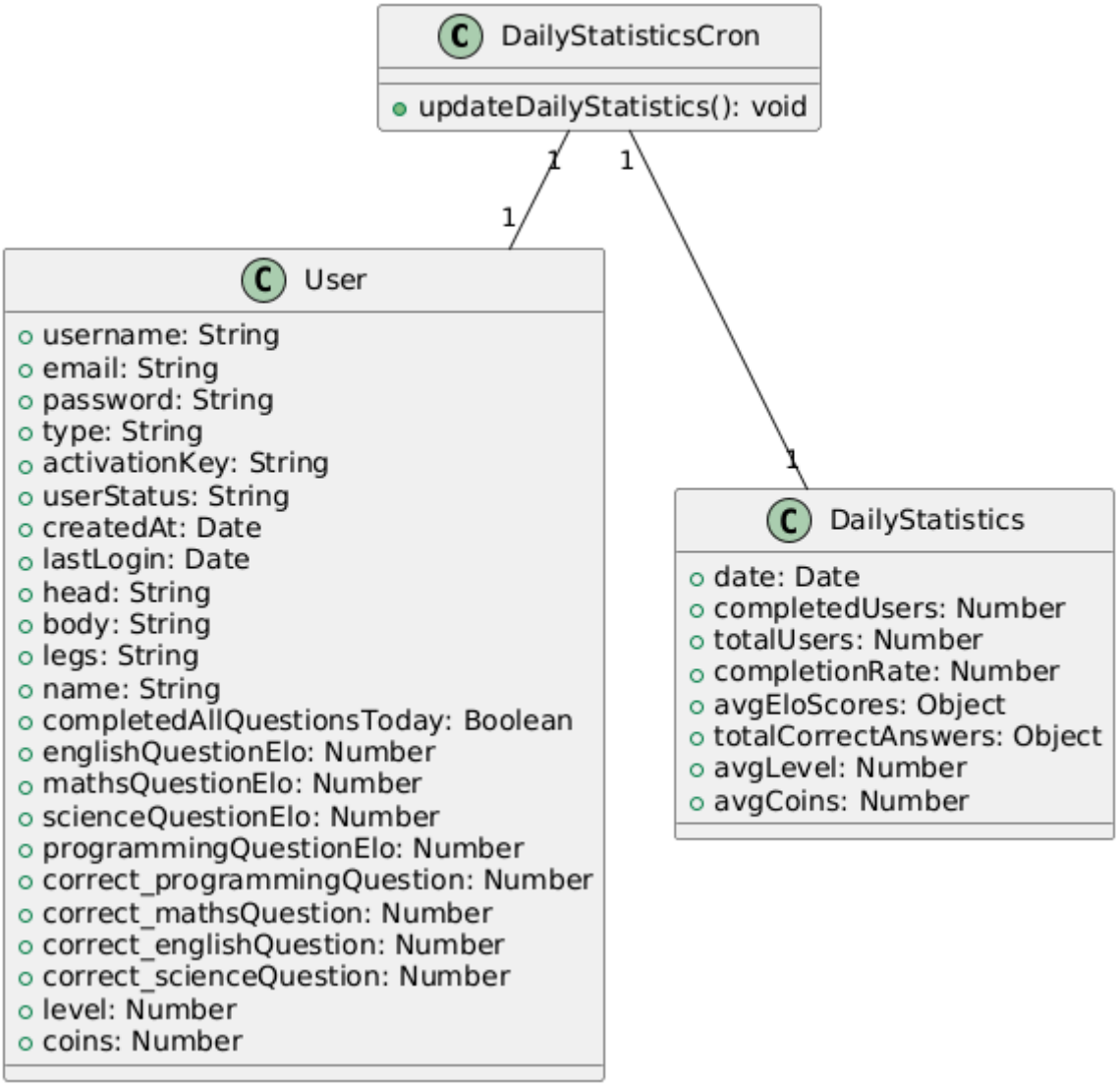
Rysunek 3.32 Diagram klas dla pliku UserStatsController.js (źródło: opracowanie własne)



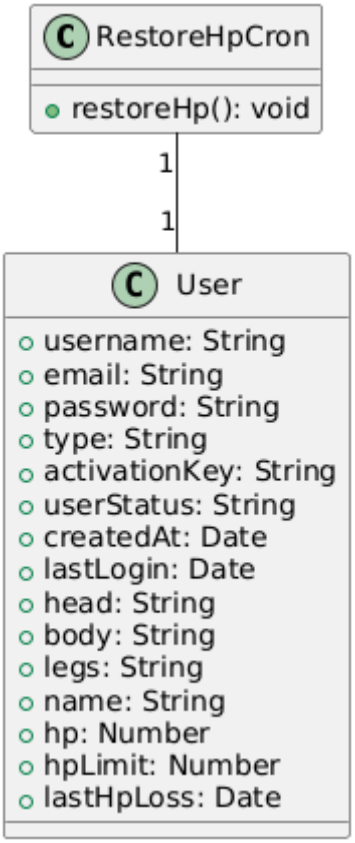
Rysunek 3.33 Diagram klas dla pliku WardrobeController.js (źródło: opracowanie własne)



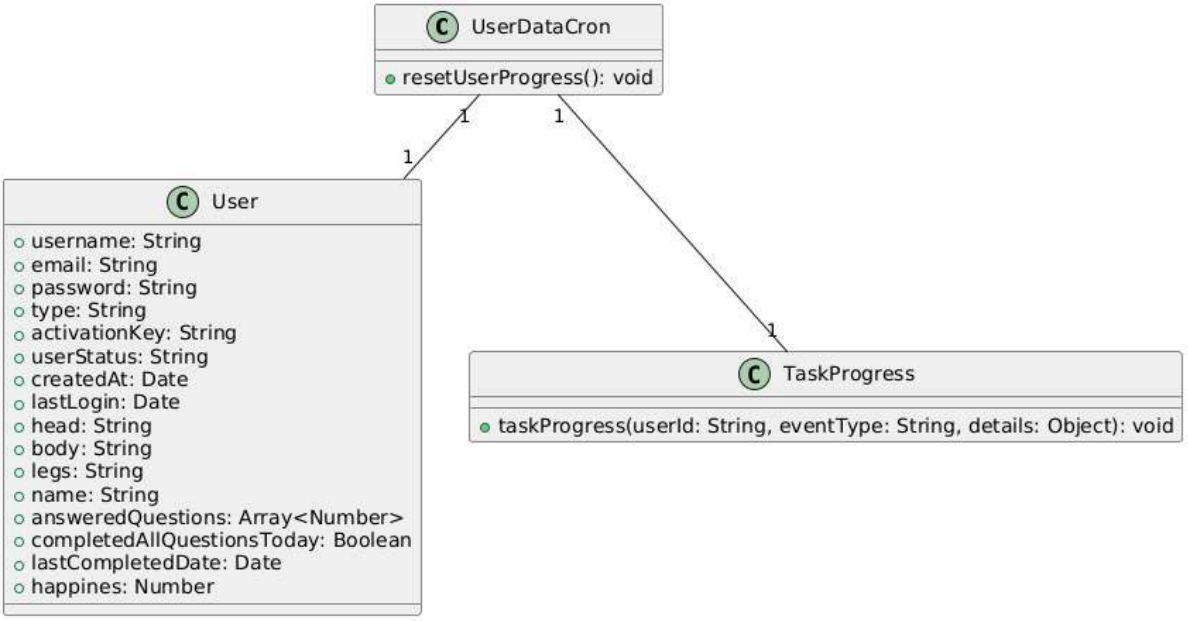
Rysunek 3.34 Diagram klas dla pliku dailyQuizCron.js (źródło: opracowanie własne)



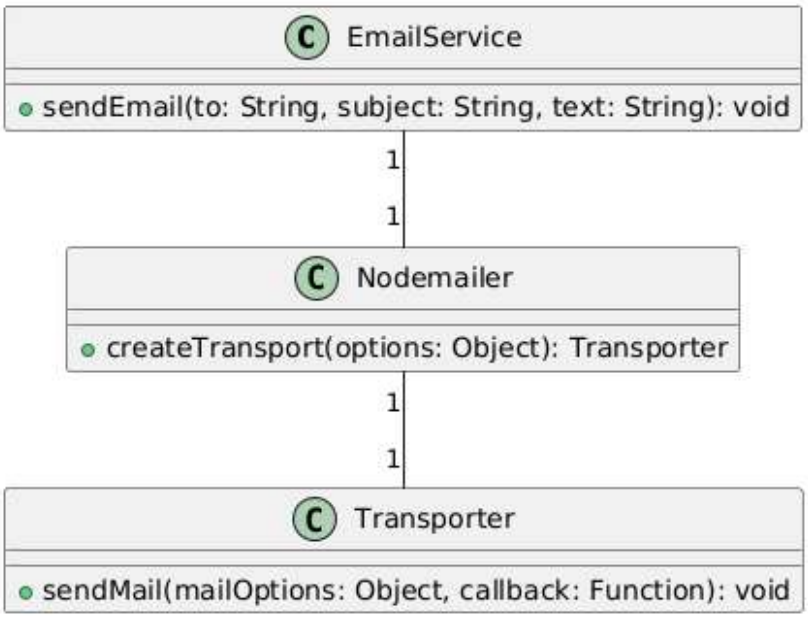
Rysunek 3.35 Diagram klas dla pliku dailyStatistics.js (źródło: opracowanie własne)



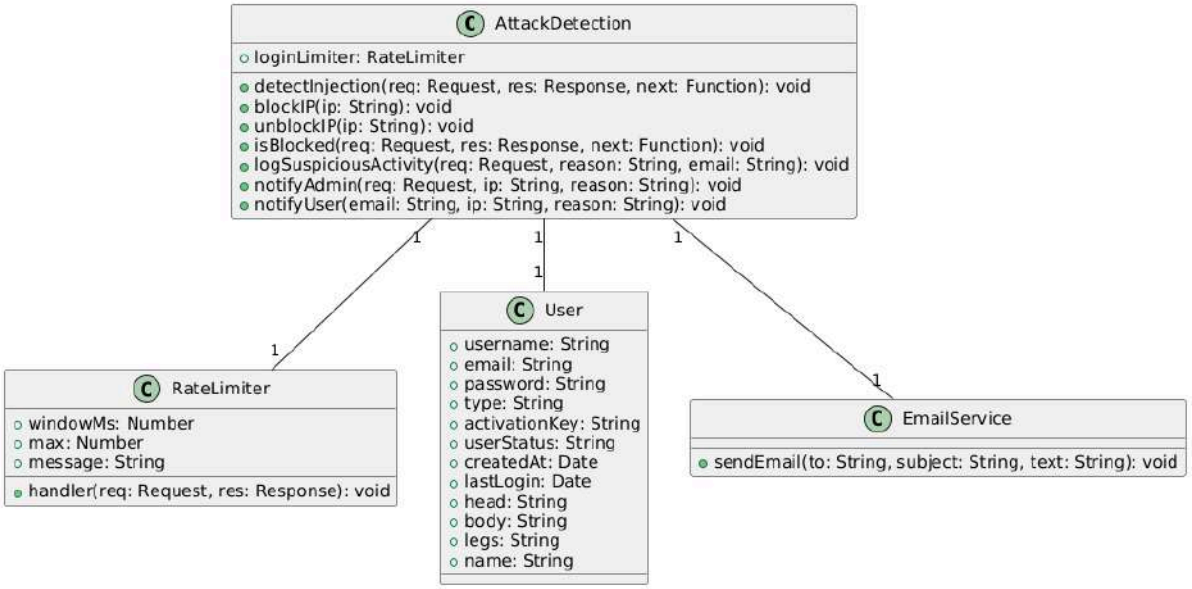
Rysunek 3.36 Diagram klas dla pliku restoreHp.js (źródło: opracowanie własne)



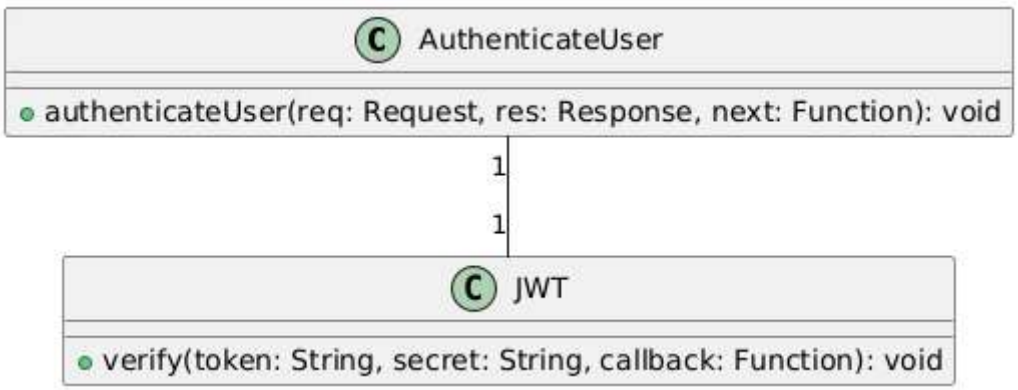
Rysunek 3.37 Diagram klas dla pliku userDataCron.js (źródło: opracowanie własne)



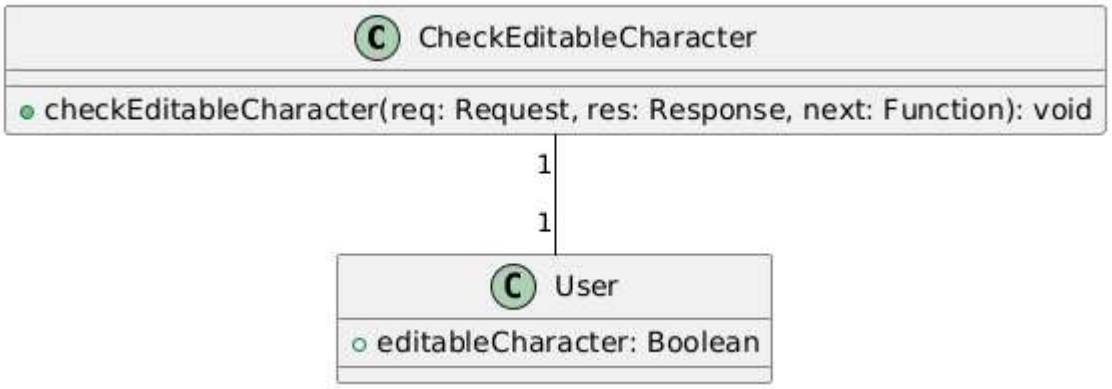
Rysunek 3.38 Diagram klas dla pliku emailservice.js (źródło: opracowanie własne)



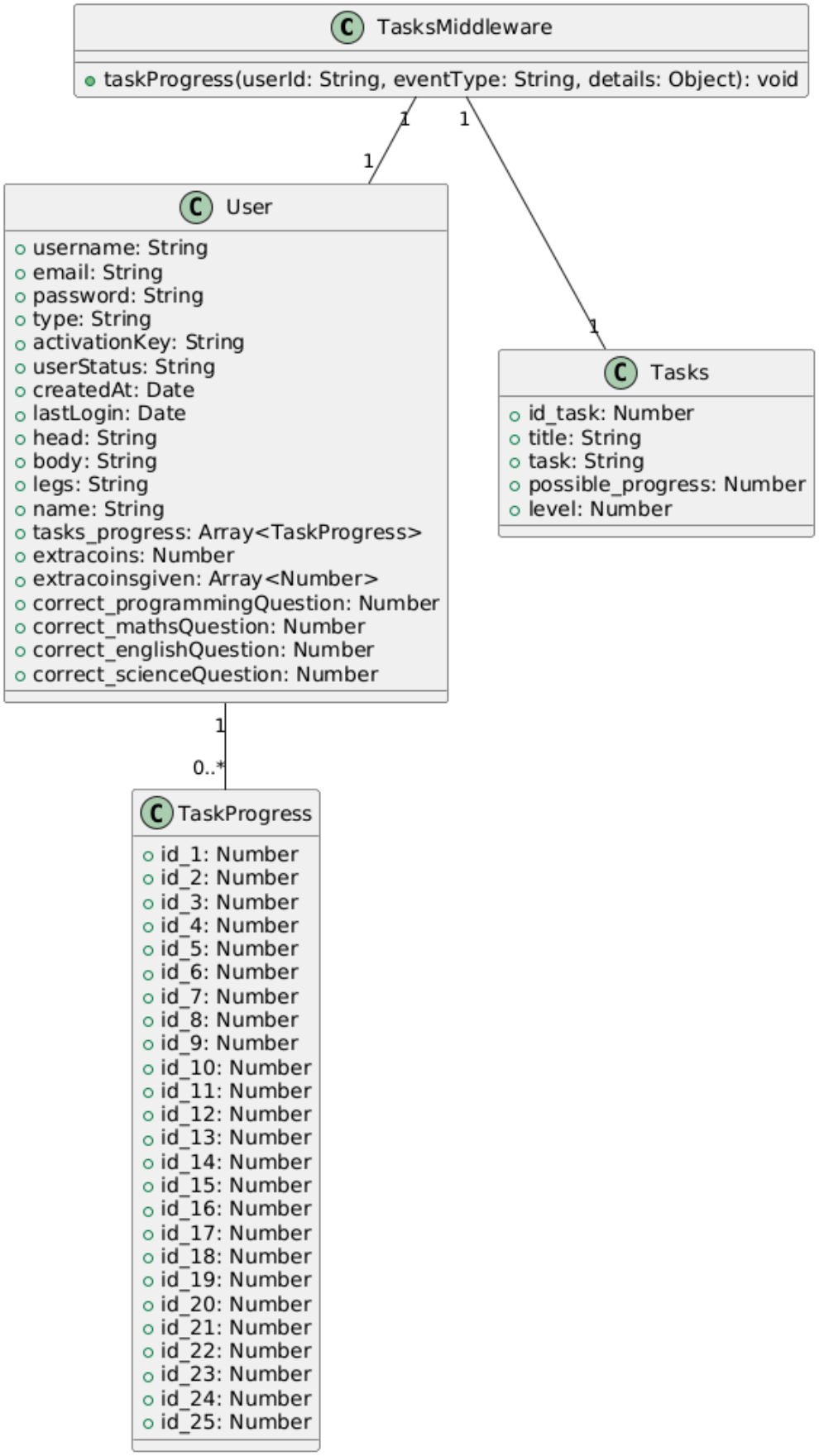
Rysunek 3.39 Diagram klas dla pliku attackDetection.js (źródło: opracowanie własne)



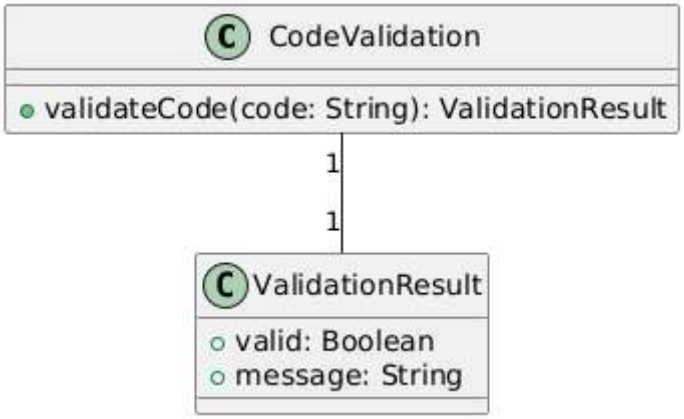
Rysunek 3.40 Diagram klas dla pliku authenticateUser.js (źródło: opracowanie własne)



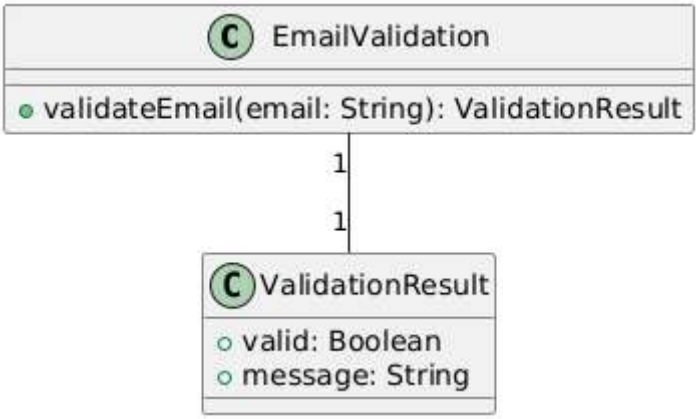
Rysunek 3.41 Diagram klas dla pliku checkEditableCharacter.js (źródło: opracowanie własne)



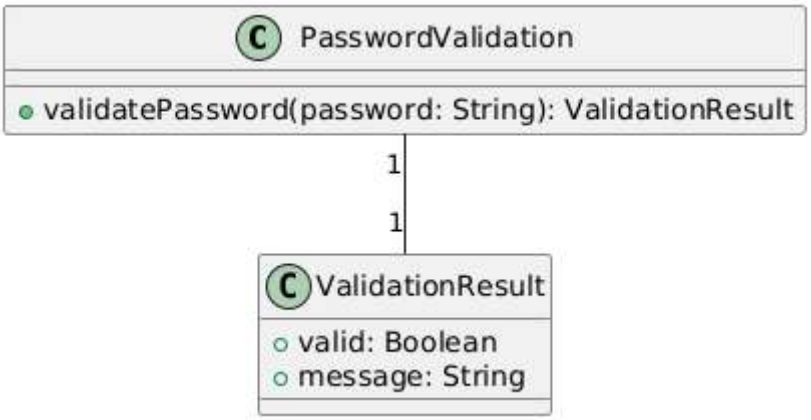
Rysunek 3.42 Diagram klas dla pliku tasks.js (źródło: opracowanie własne)



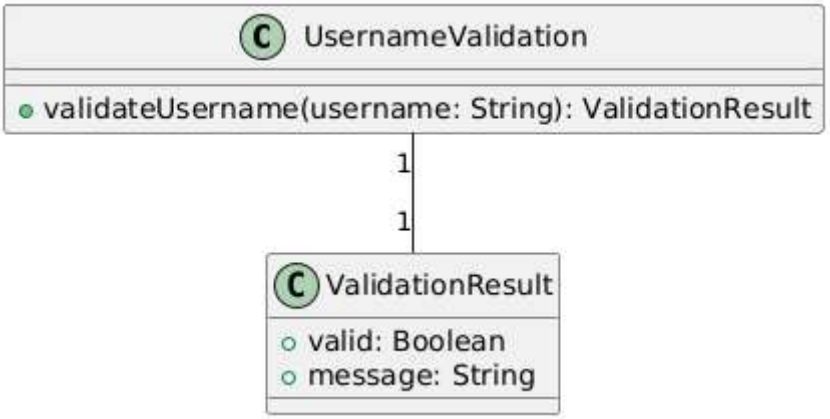
Rysunek 3.43 Diagram klas dla pliku codeValidation.js (źródło: opracowanie własne)



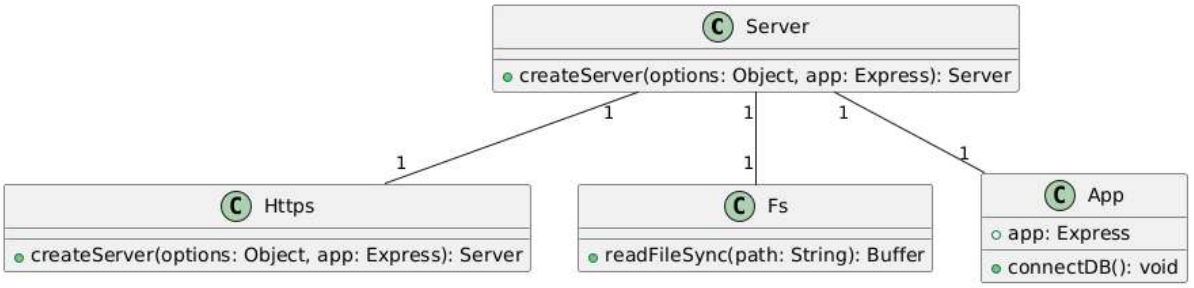
Rysunek 3.44 Diagram klas dla pliku emailValidation.js (źródło: opracowanie własne)



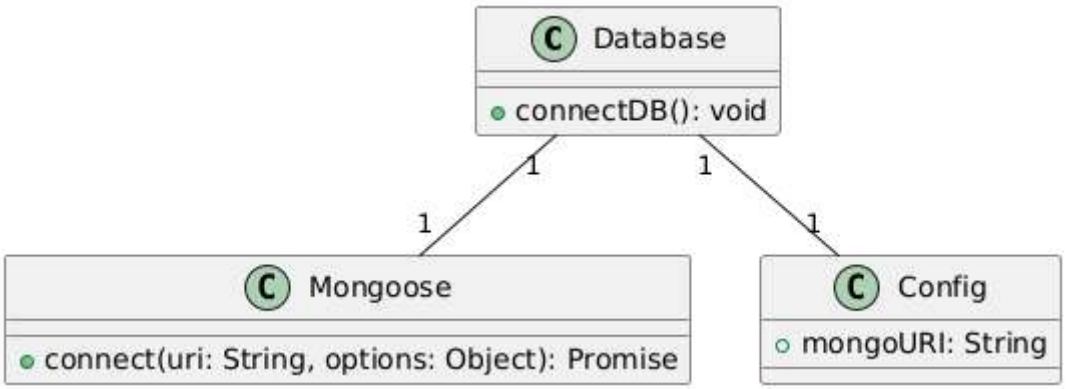
Rysunek 3.45 Diagram klas dla pliku passwordValidation.js (źródło: opracowanie własne)



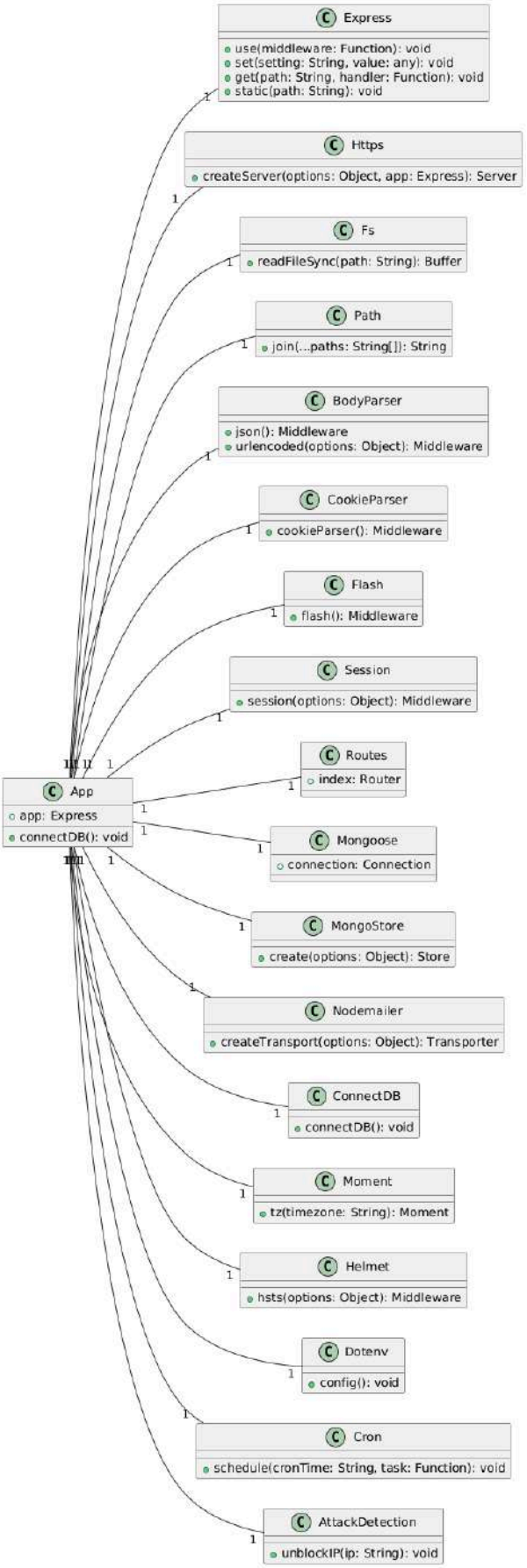
Rysunek 3.46 Diagram klas dla pliku usernameValidation.js (źródło: opracowanie własne)



Rysunek 3.47 Diagram klas dla pliku server.js (źródło: opracowanie własne)

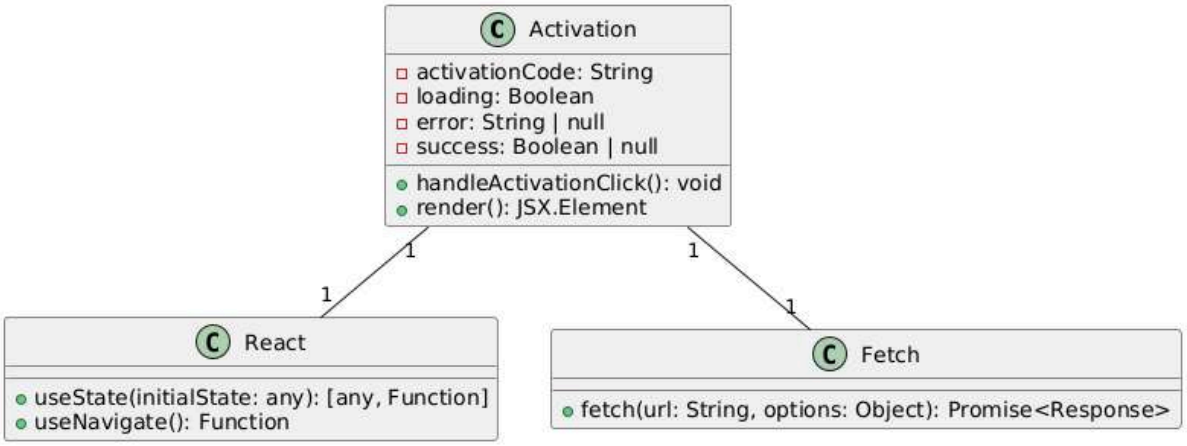


Rysunek 3.48 Diagram klas dla pliku connection.js i database.js (źródło: opracowanie własne)

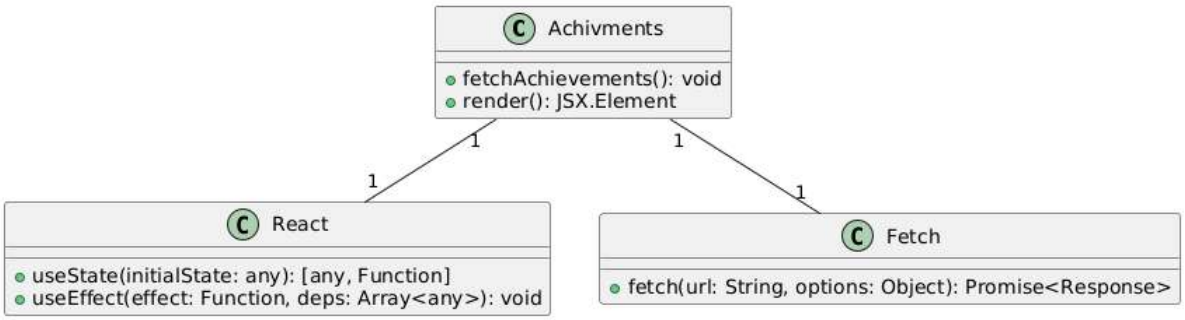


Rysunek 3.49 Diagram klas dla pliku app.js (źródło: opracowanie własne)

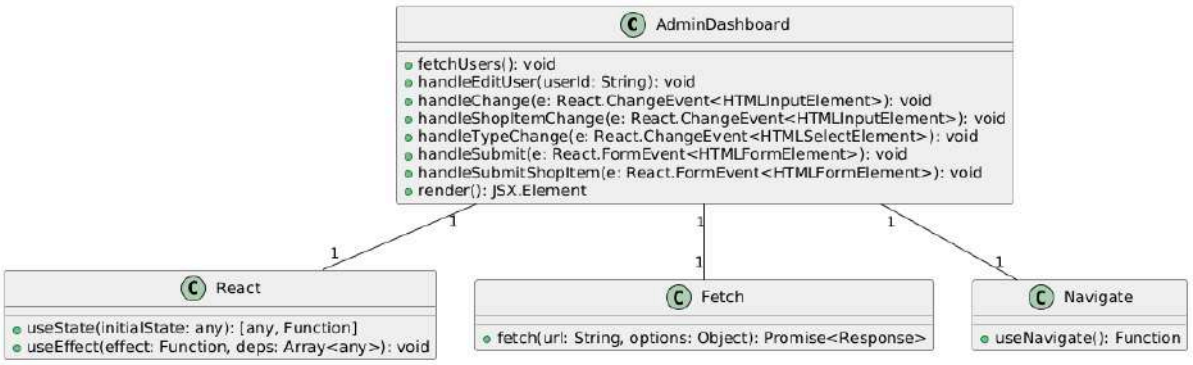
3.9.2 Diagramy klas dla plików z frontendu



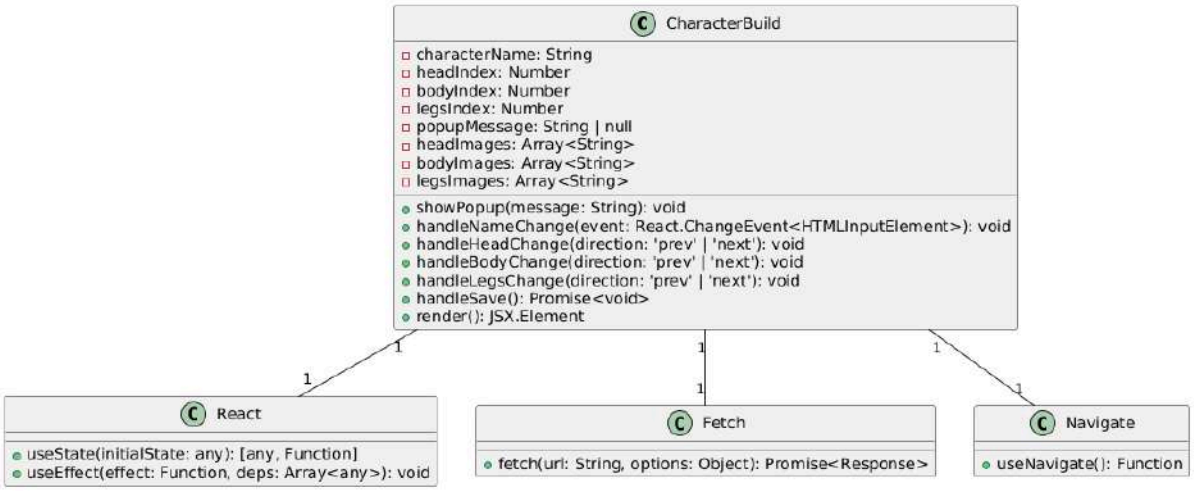
Rysunek 3.50 Diagram klas dla pliku Activation.tsx (źródło: opracowanie własne)



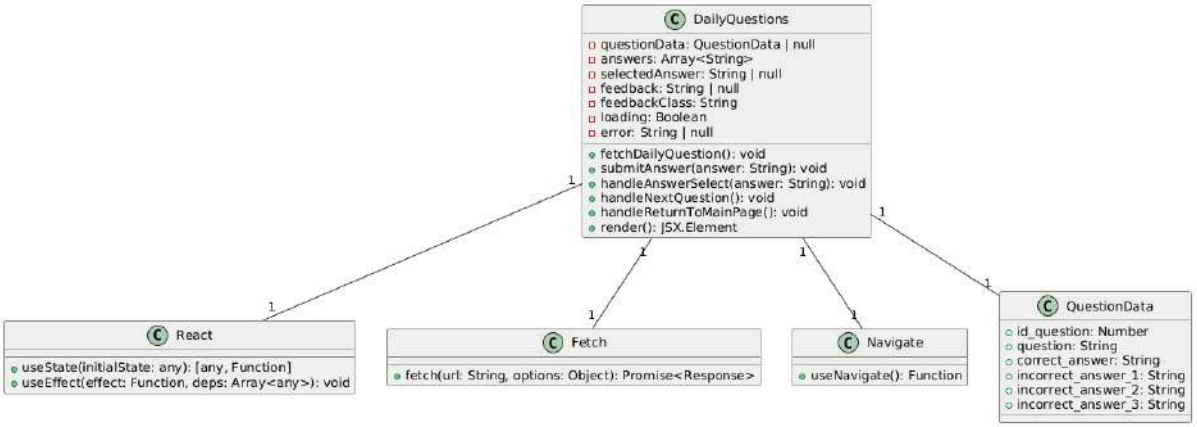
Rysunek 3.51 Diagram klas dla pliku Achivments.tsx (źródło: opracowanie własne)



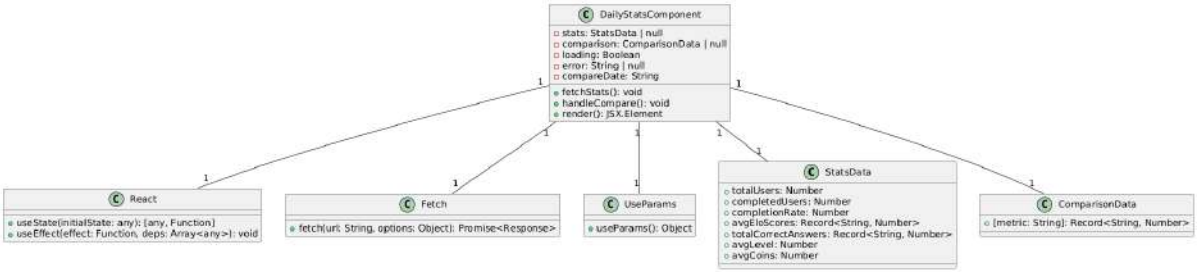
Rysunek 3.52 Diagram klas dla pliku AdminDashBoard.tsx (źródło: opracowanie własne)



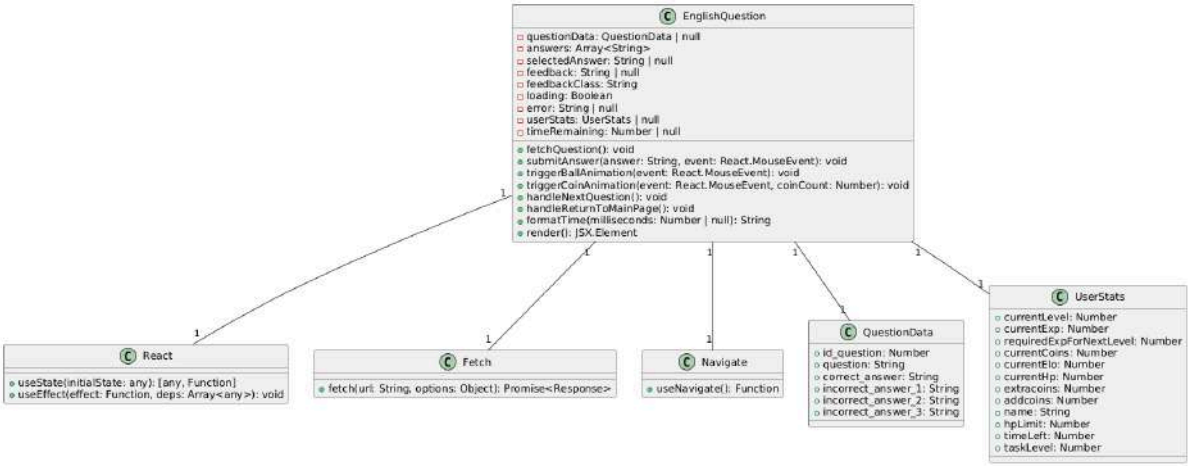
Rysunek 3.53 Diagram klas dla pliku CharacterBuild.tsx (źródło: opracowanie własne)



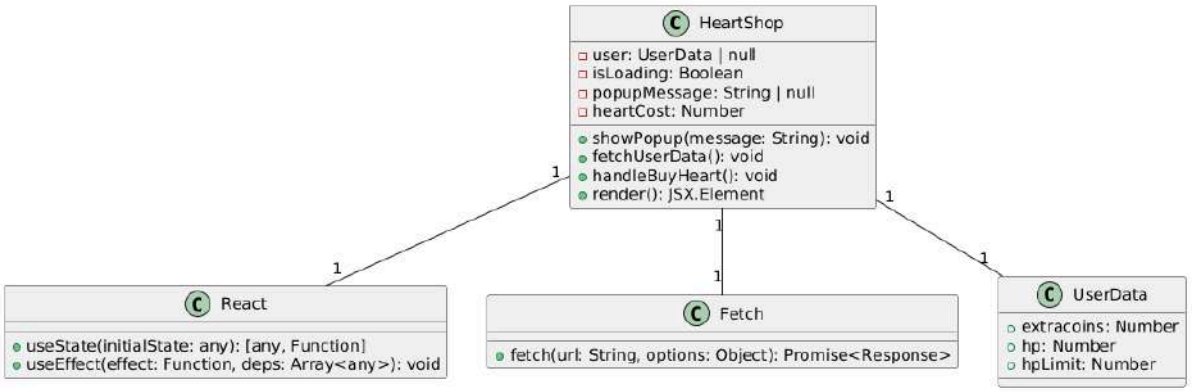
Rysunek 3.54 Diagram klas dla pliku DailyQuestions.tsx (źródło: opracowanie własne)



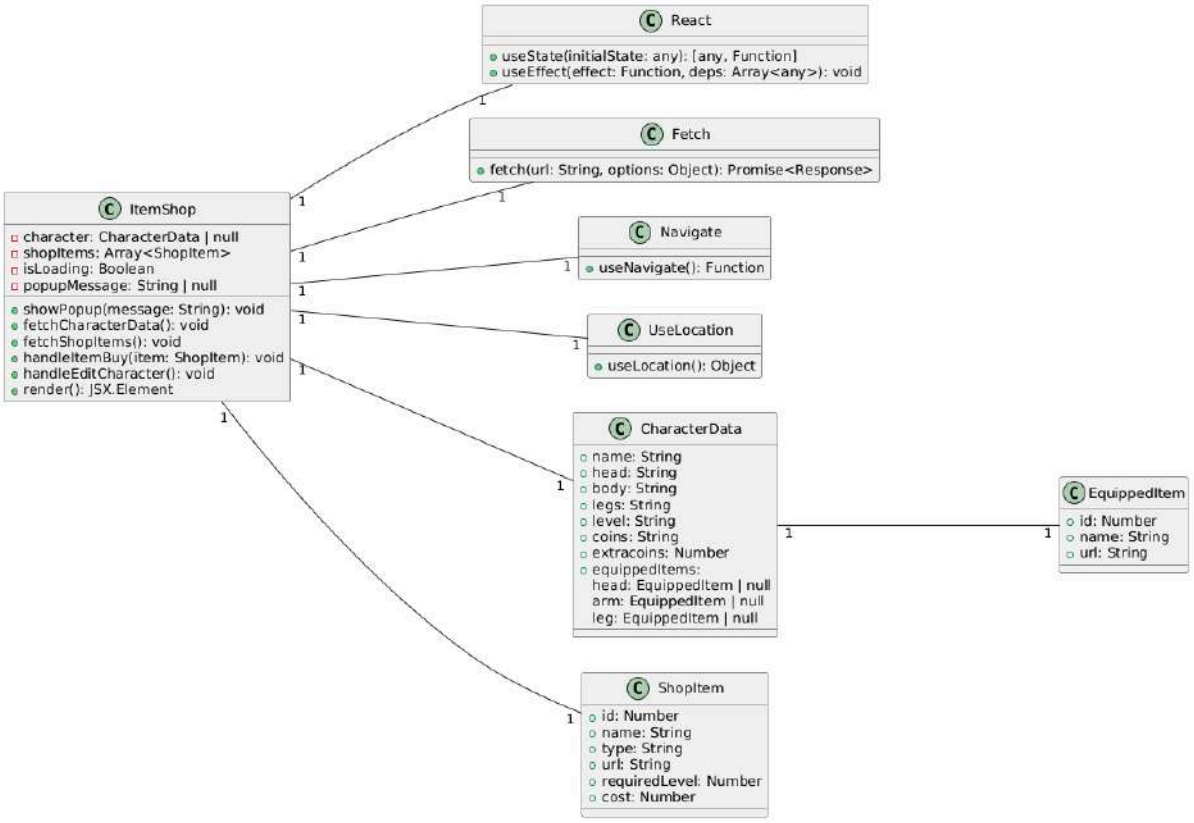
Rysunek 3.55 Diagram klas dla pliku DailyStats.tsx (źródło: opracowanie własne)



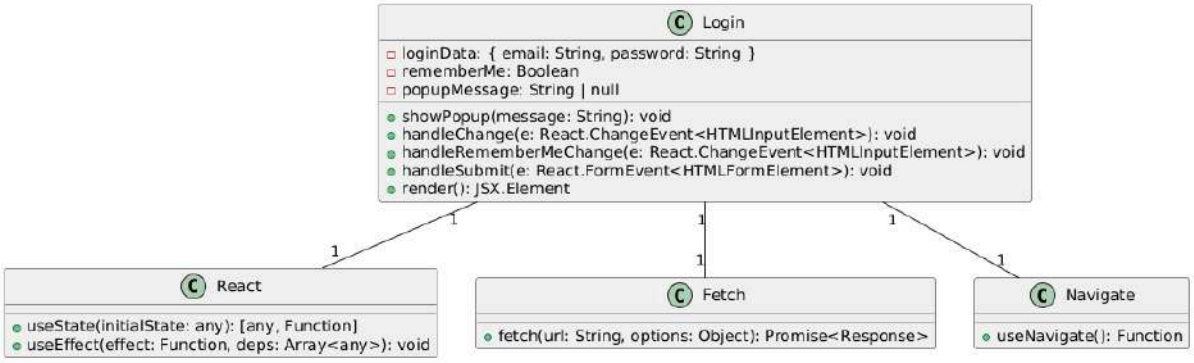
Rysunek 3.56 Diagram klas dla pliku EnglishQuestions.tsx (źródło: opracowanie własne)



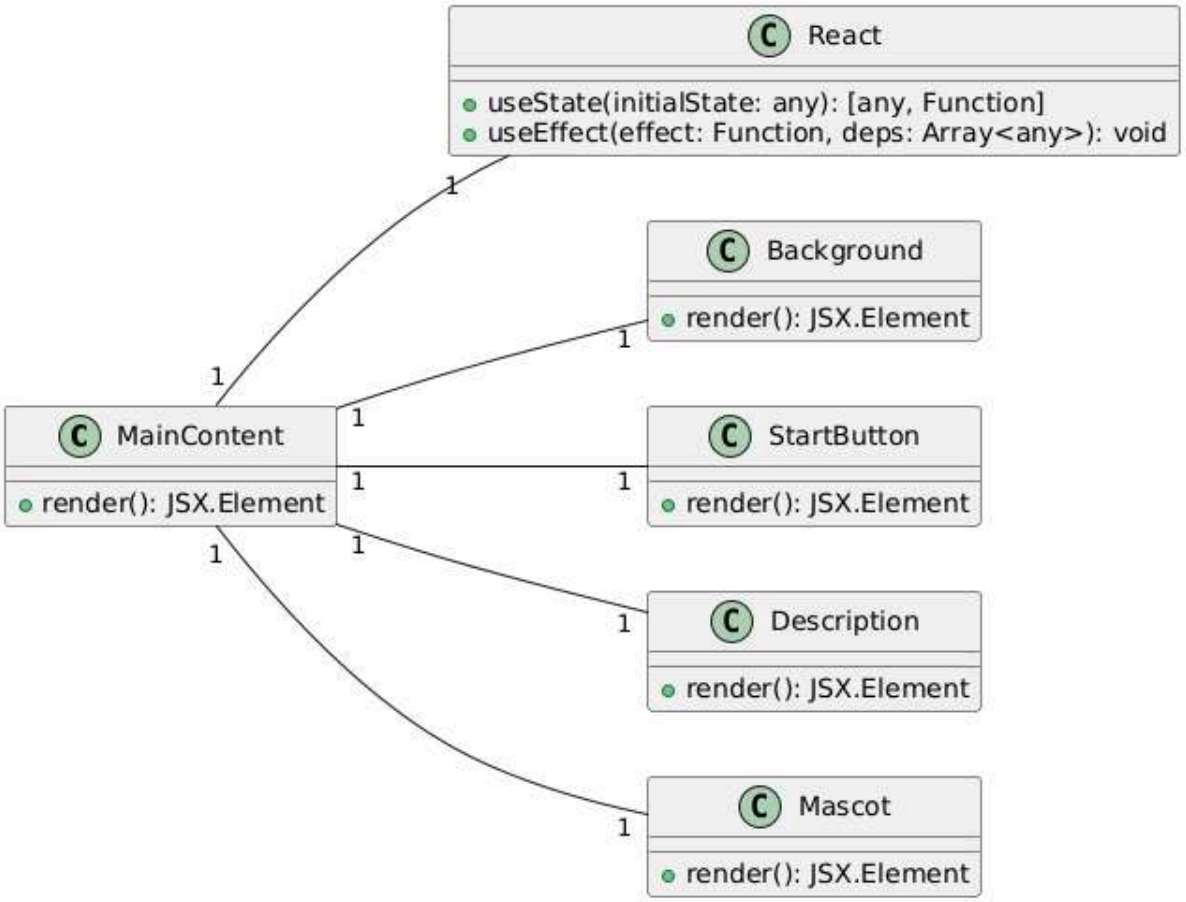
Rysunek 3.57 Diagram klas dla pliku HeartShop.tsx (źródło: opracowanie własne)



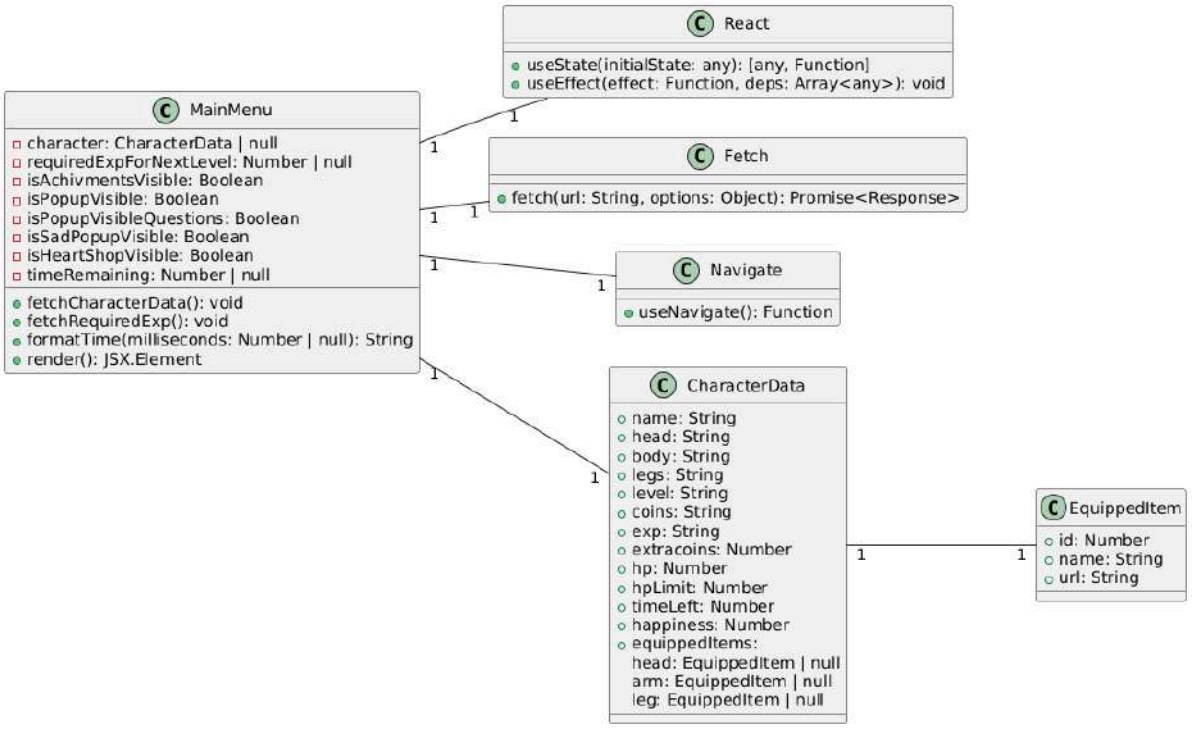
Rysunek 3.58 Diagram klas dla pliku ItemShop.tsx (źródło: opracowanie własne)



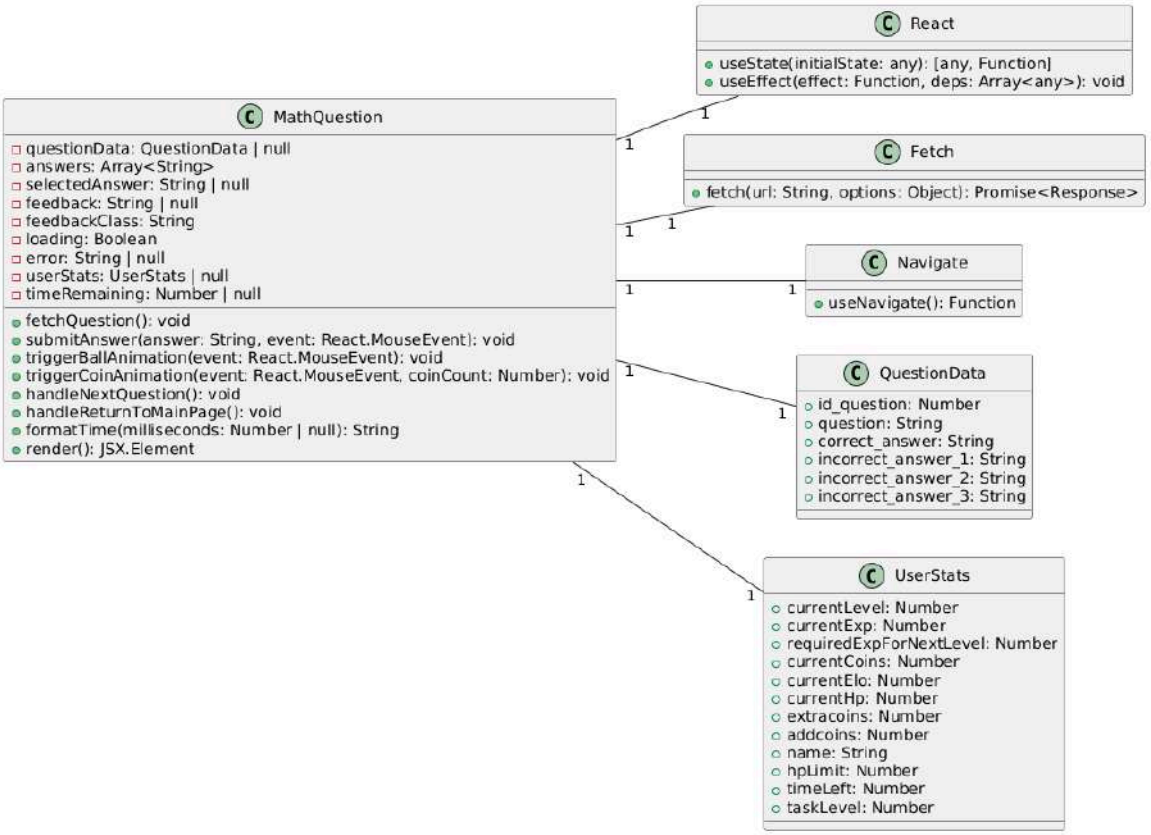
Rysunek 3.59 Diagram klas dla pliku Login.tsx (źródło: opracowanie własne)



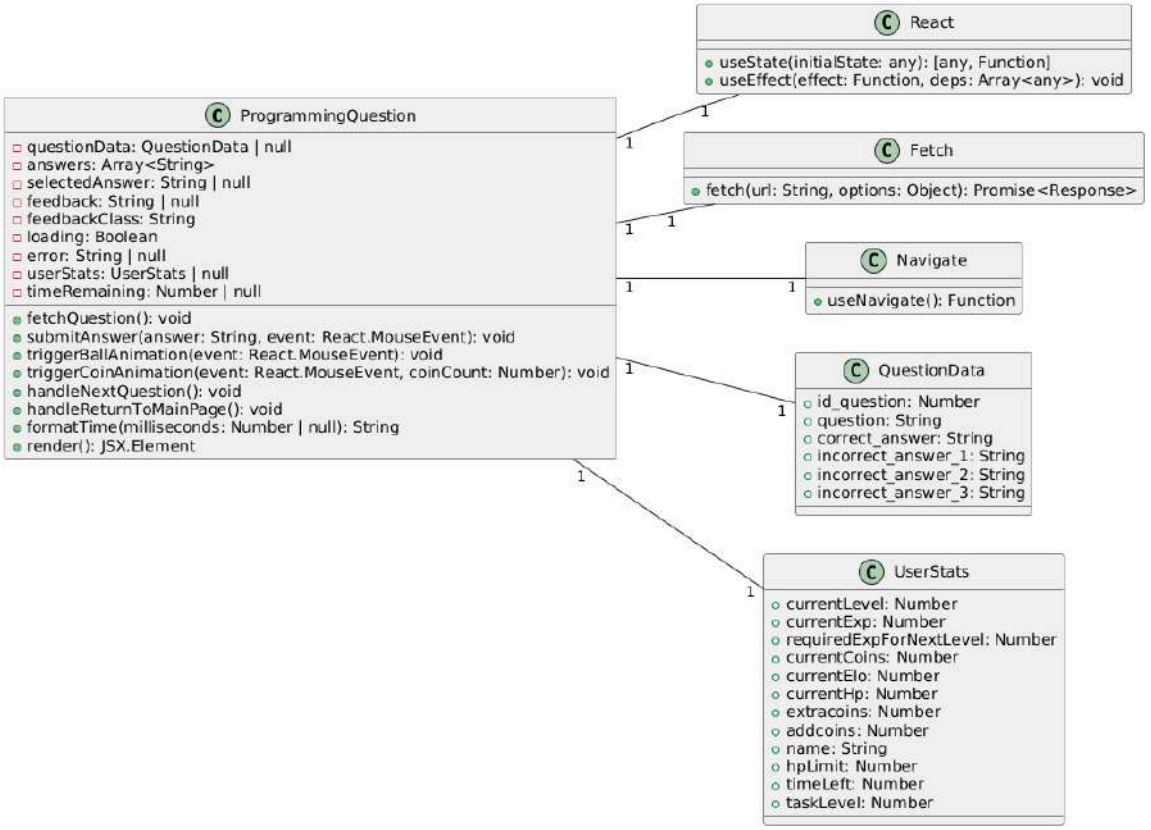
Rysunek 3.60 Diagram klas dla pliku MainContent.tsx (źródło: opracowanie własne)



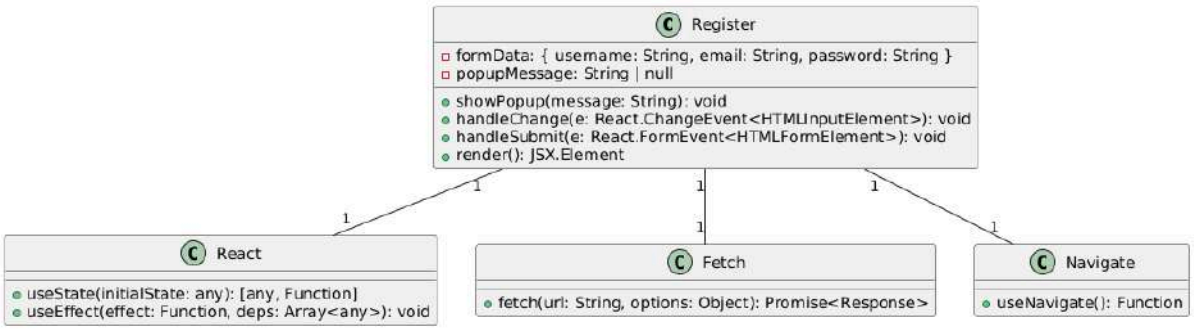
Rysunek 3.61 Diagram klas dla pliku MainMenu.tsx (źródło: opracowanie własne)



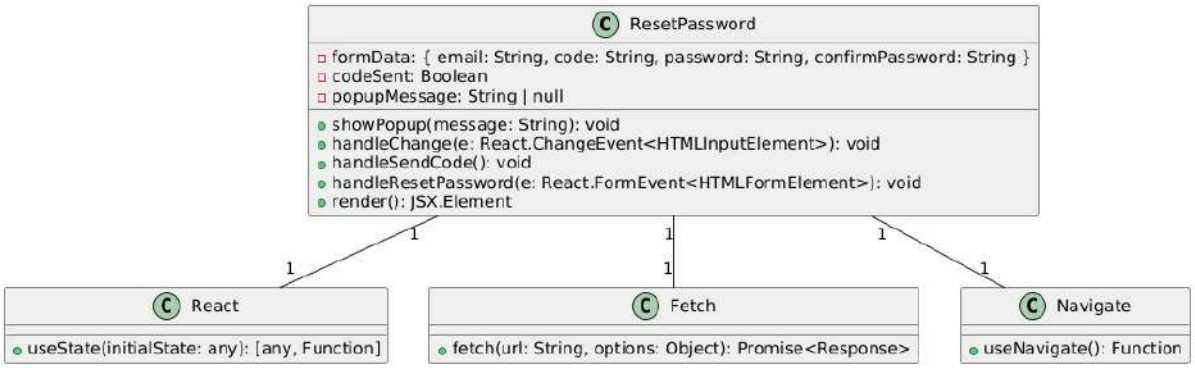
Rysunek 3.62 Diagram klas dla pliku MathQuestions.tsx (źródło: opracowanie własne)



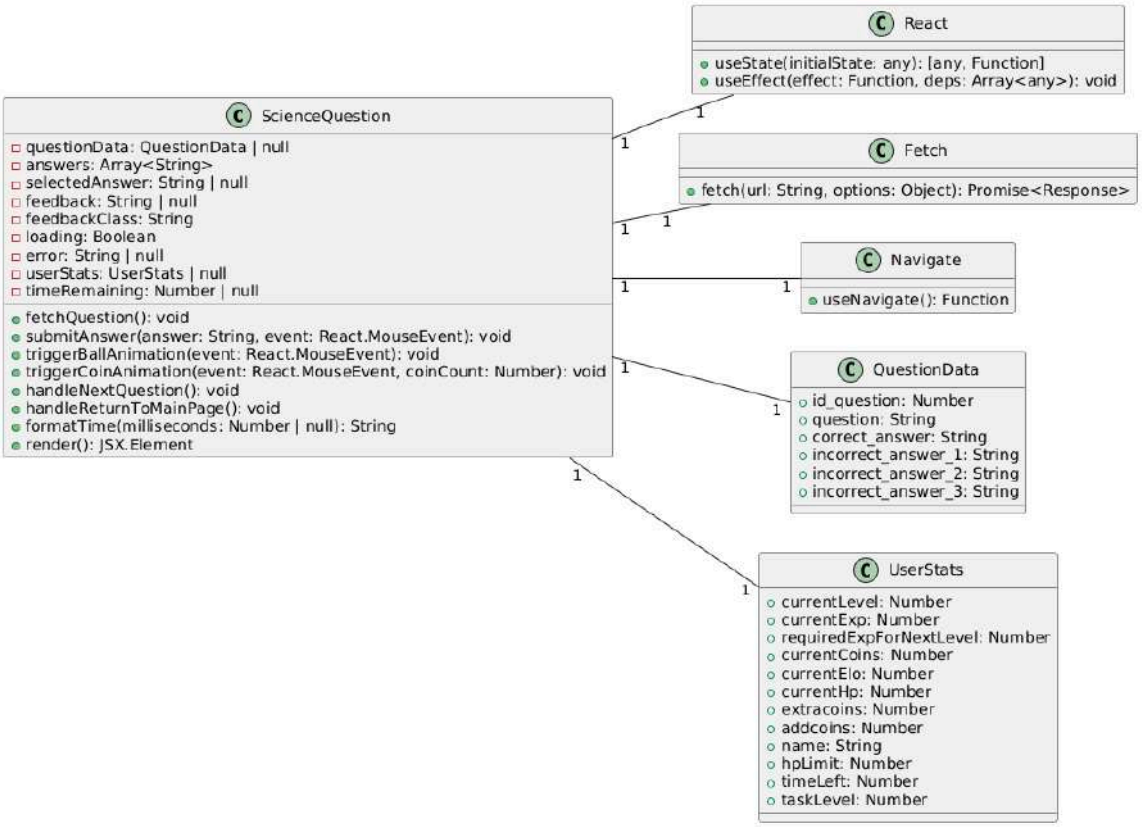
Rysunek 3.63 Diagram klas dla pliku ProgramingQuestions.tsx (źródło: opracowanie własne)



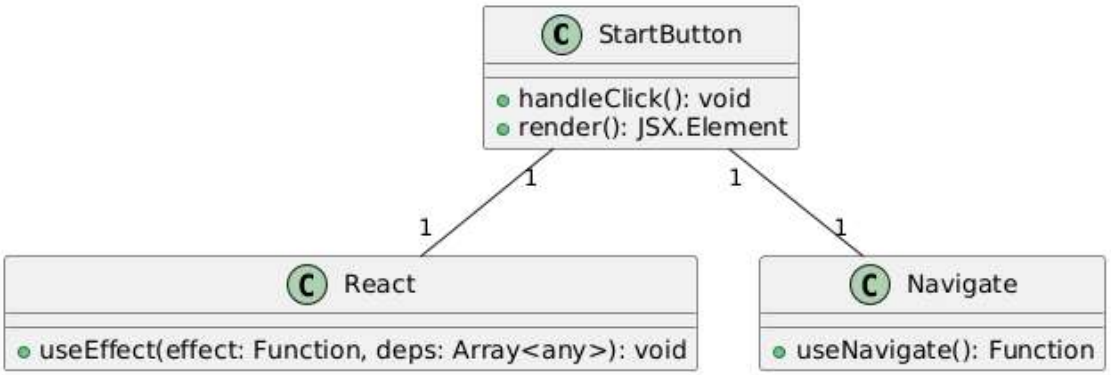
Rysunek 3.64 Diagram klas dla pliku Register.tsx (źródło: opracowanie własne)



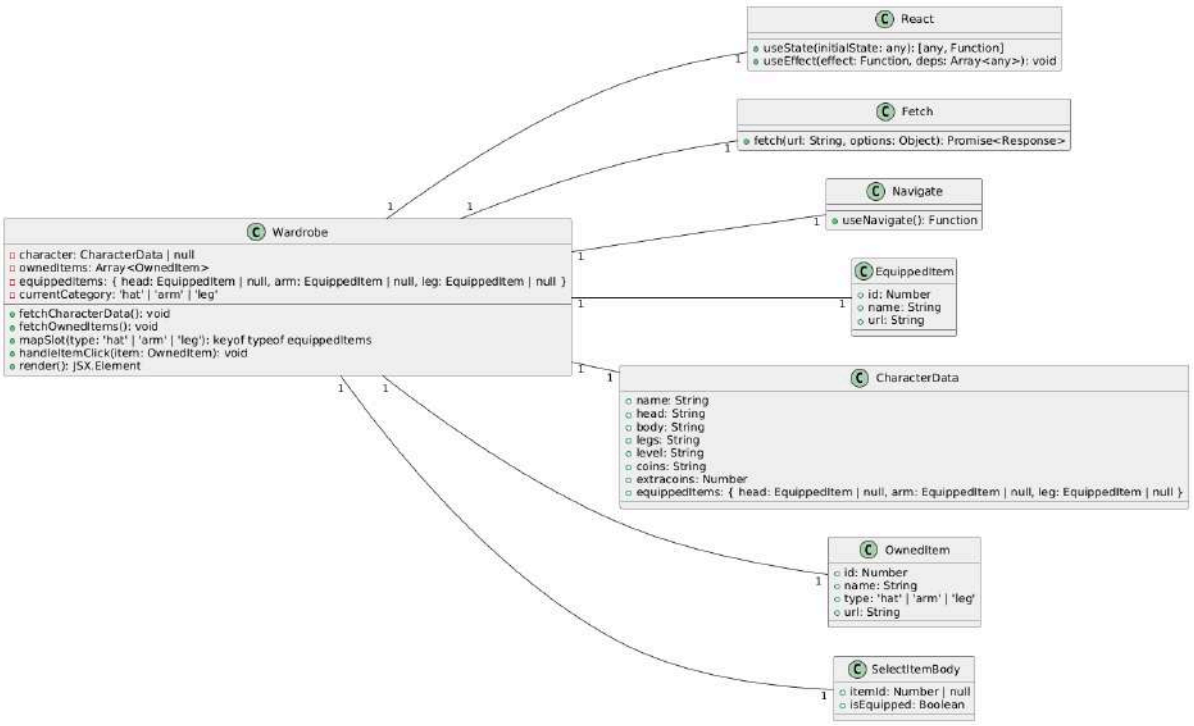
Rysunek 3.65 Diagram klas dla pliku ResetPassword.tsx (źródło: opracowanie własne)



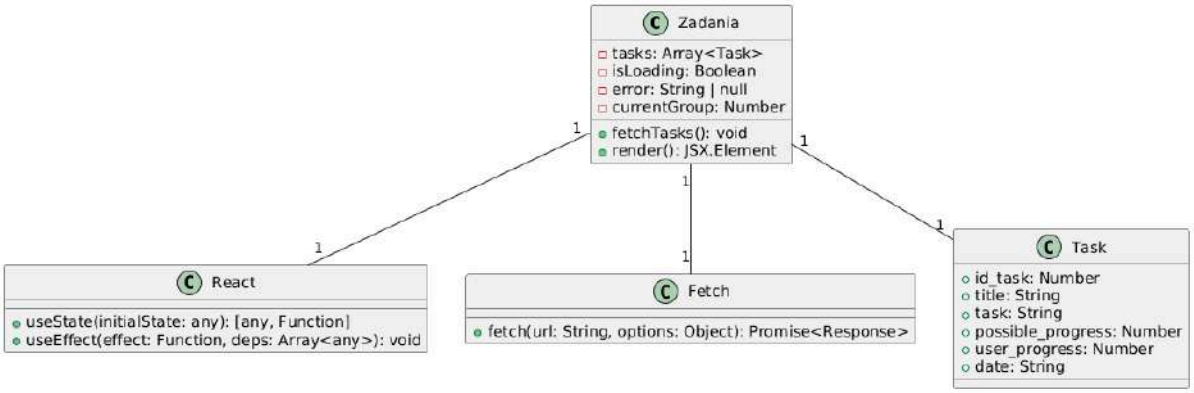
Rysunek 3.66 Diagram klas dla pliku ScienceQuestions.tsx (źródło: opracowanie własne)



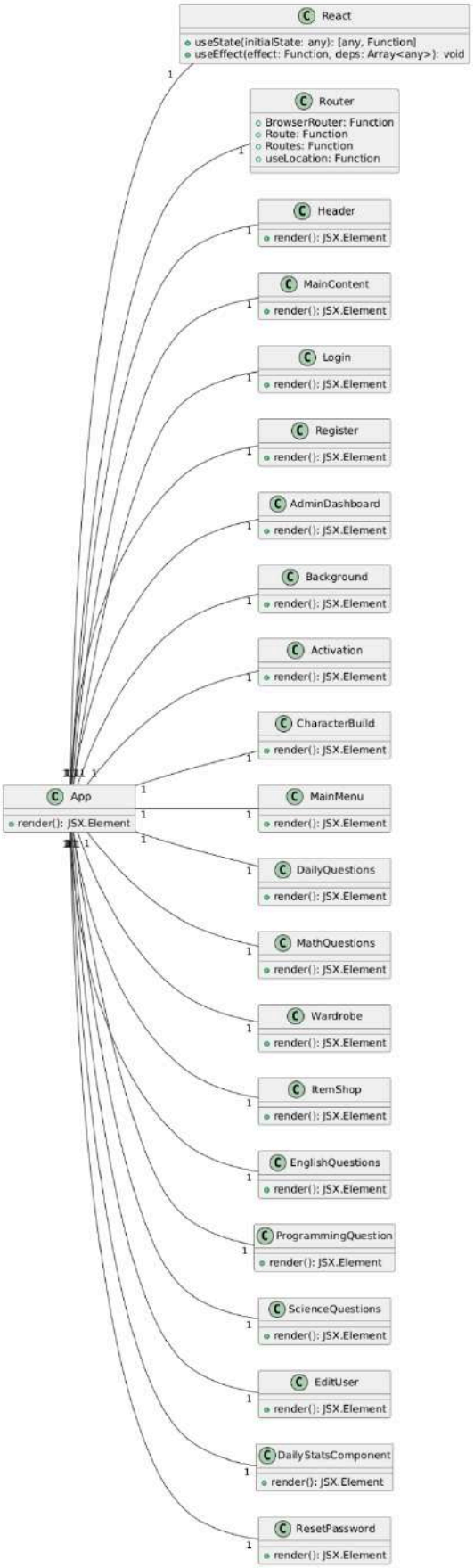
Rysunek 3.67 Diagram klas dla pliku StartButton.tsx (źródło: opracowanie własne)



Rysunek 3.68 Diagram klas dla pliku Wardrobe.tsx (źródło: opracowanie własne)

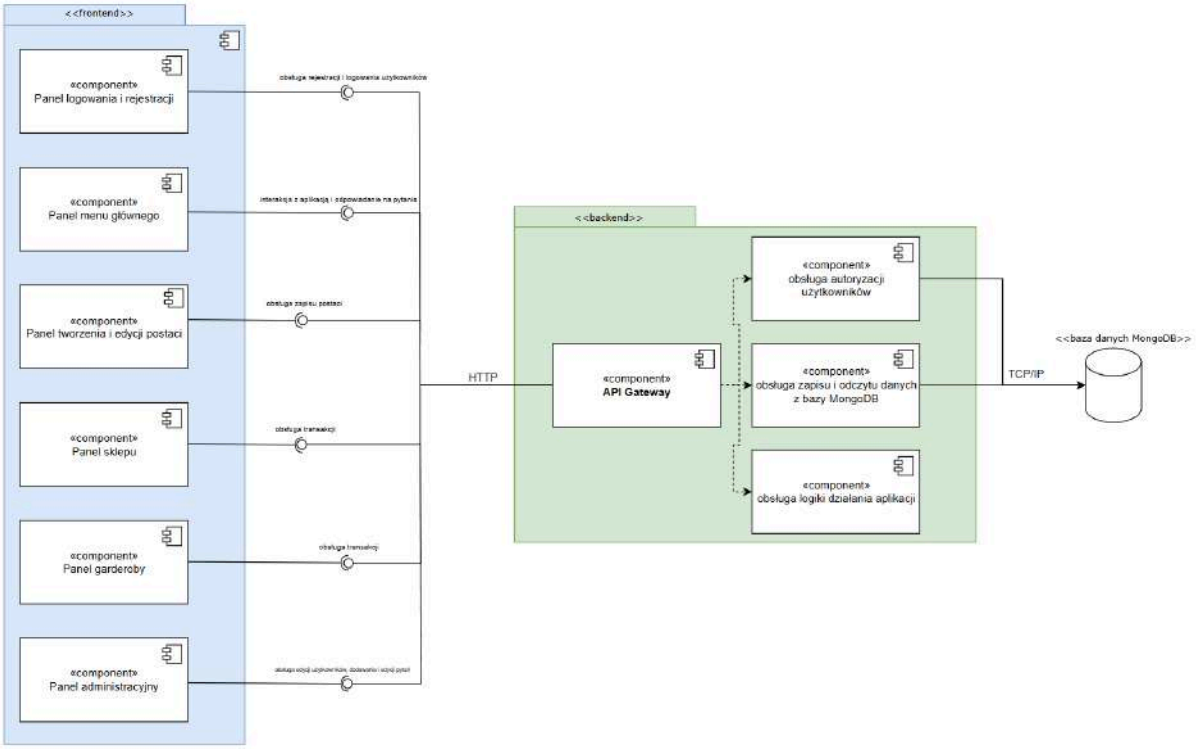


Rysunek 3.69 Diagram klas dla pliku Zadania.tsx (źródło: opracowanie własne)



Rysunek 3.70 Diagram klas dla pliku App.tsx (źródło: opracowanie własne)

3.10 Diagram komponentów



Rysunek 3.71 Diagram komponentów (źródło: opracowanie własne)

Diagram przedstawia architekturę komponentową aplikacji, podzieloną na trzy główne obszary: frontend, backend i bazę danych.

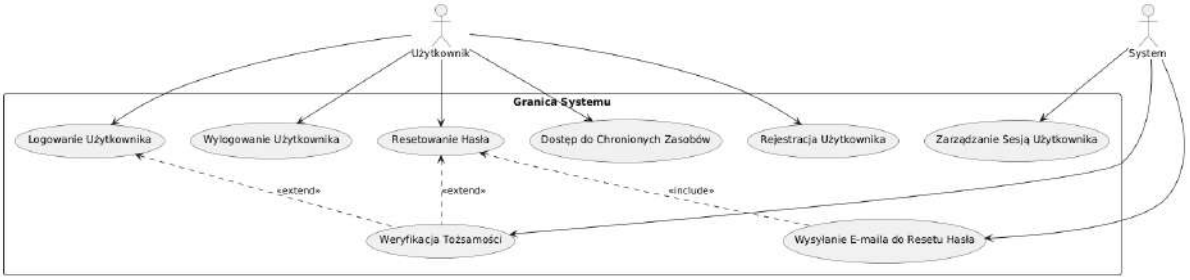
Frontend, składa się z kilku komponentów odpowiadających za różne funkcjonalności aplikacji. Panel logowania i rejestracji obsługuje procesy uwierzytelniania użytkowników. Panel menu głównego pełni funkcję nawigacyjną, umożliwiając dostęp do pozostałych modułów. Panel tworzenia i edycji postaci pozwala użytkownikom na zarządzanie postaciami w aplikacji. Panel sklepu obsługuje transakcje zakupowe, a panel garderoby prawdopodobnie odpowiada za przechowywanie ekwipunku. Panel administracyjny umożliwia edycję użytkowników oraz zarządzanie danymi aplikacji. Wszystkie komponenty frontendowe komunikują się z backendem poprzez protokół HTTP.

Backend, pełni rolę warstwy logicznej i pośredniczącej między frontendem a bazą danych. Centralnym komponentem tej warstwy jest API Gateway, który zarządza ruchem i dystrybuuje żądania do odpowiednich usług. Obsługa autoryzacji użytkowników odpowiada za logowanie, rejestrację i weryfikację tożsamości. Komponent odpowiedzialny za zapis i odczyt danych pośredniczy w operacjach na bazie MongoDB, natomiast obsługa logiki działania aplikacji realizuje zasady biznesowe i operacje aplikacyjne. Komunikacja między komponentami backendu odbywa się wewnętrznie, a API Gateway pełni funkcję głównego punktu dostępu.

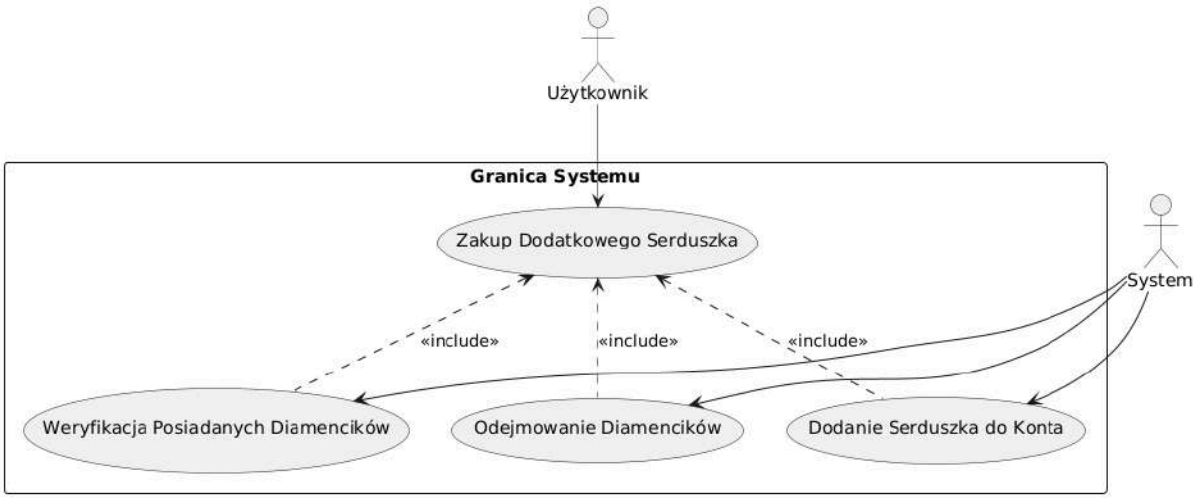
Baza danych, oznaczona jako MongoDB, przechowuje informacje niezbędne do działania aplikacji. Komunikacja backendu z bazą odbywa się za pomocą protokołu TCP/IP, co pozwala na wydajny zapis i odczyt danych. Cała architektura przedstawia

system modularny, w którym frontend zapewnia interfejs użytkownika, backend odpowiada za przetwarzanie logiki, a baza danych gromadzi informacje aplikacyjne.

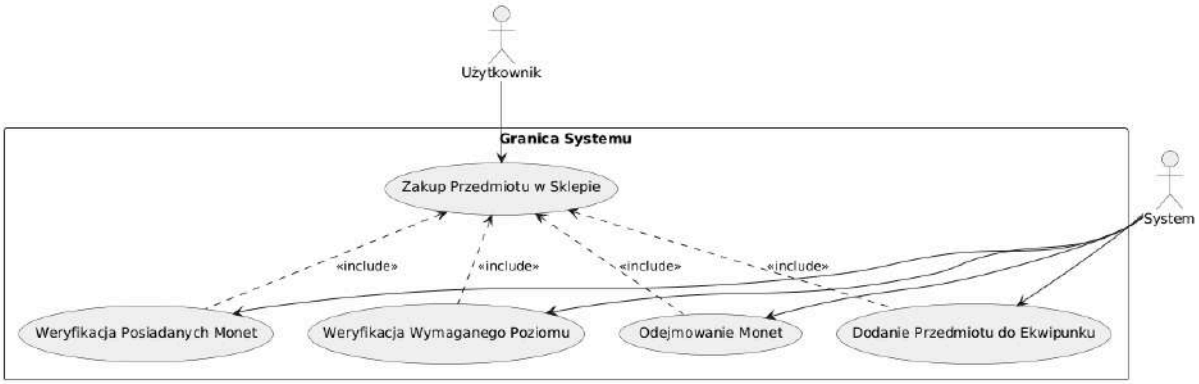
3.11 Diagramy przypadków użycia



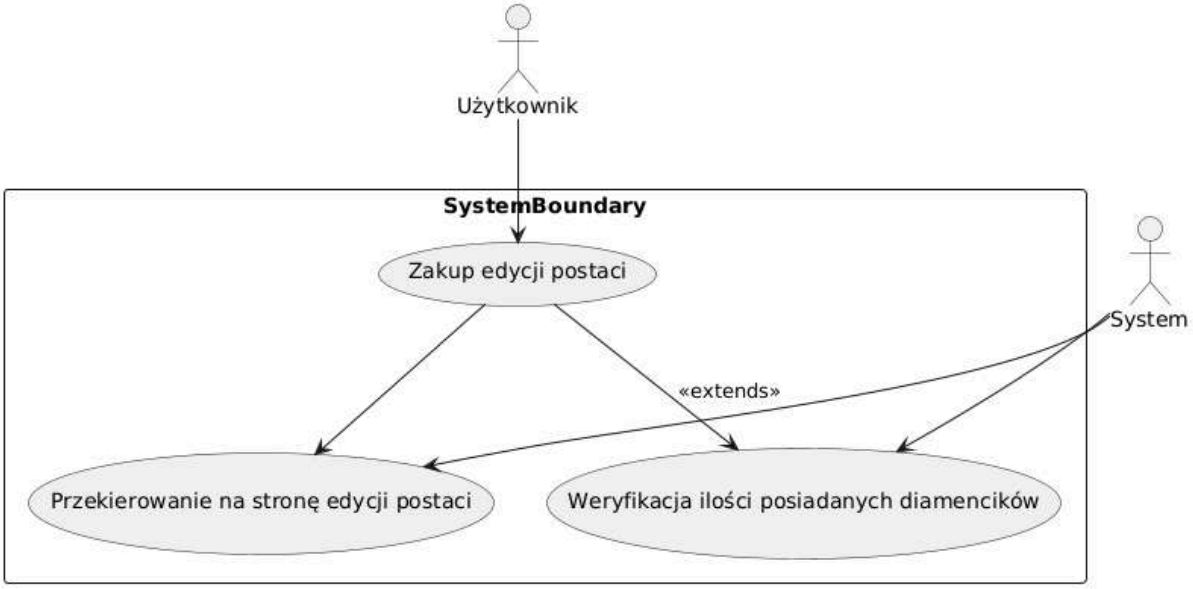
Rysunek 3.72 Diagram przypadków użycia — panel logowania i rejestracji (źródło: opracowanie własne)



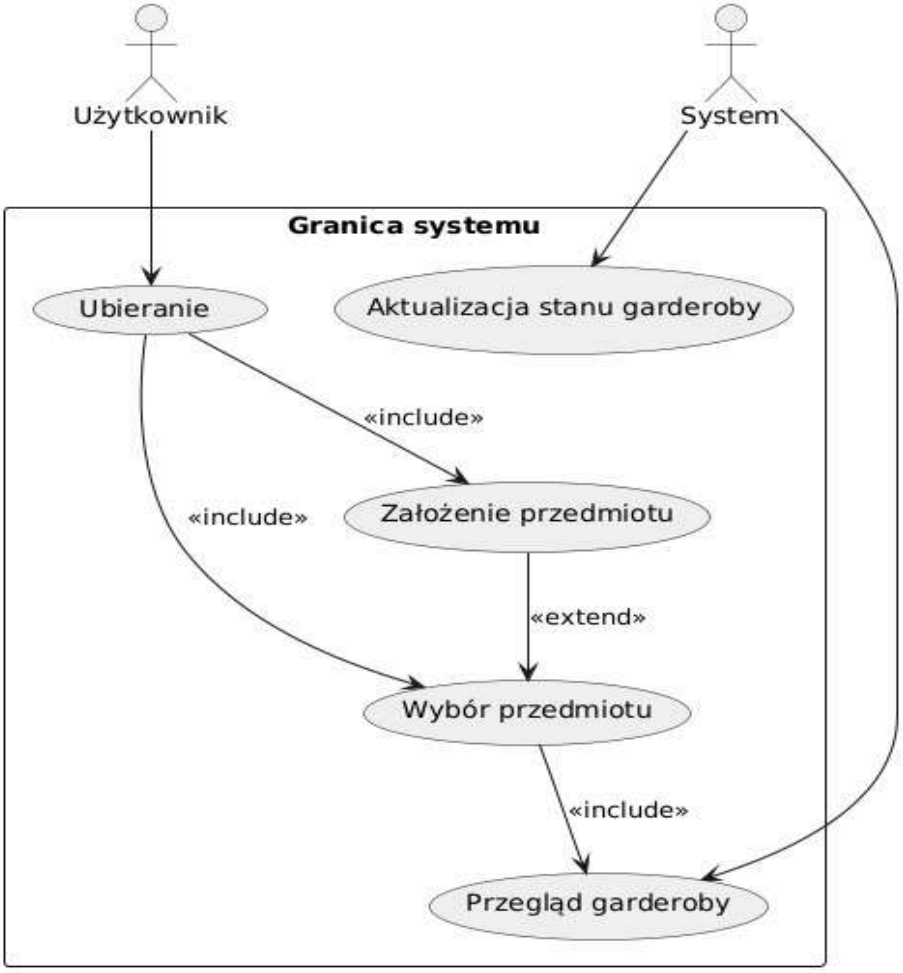
Rysunek 3.73 Diagram przypadków użycia — zakup dodatkowego serduszka (źródło: opracowanie własne)



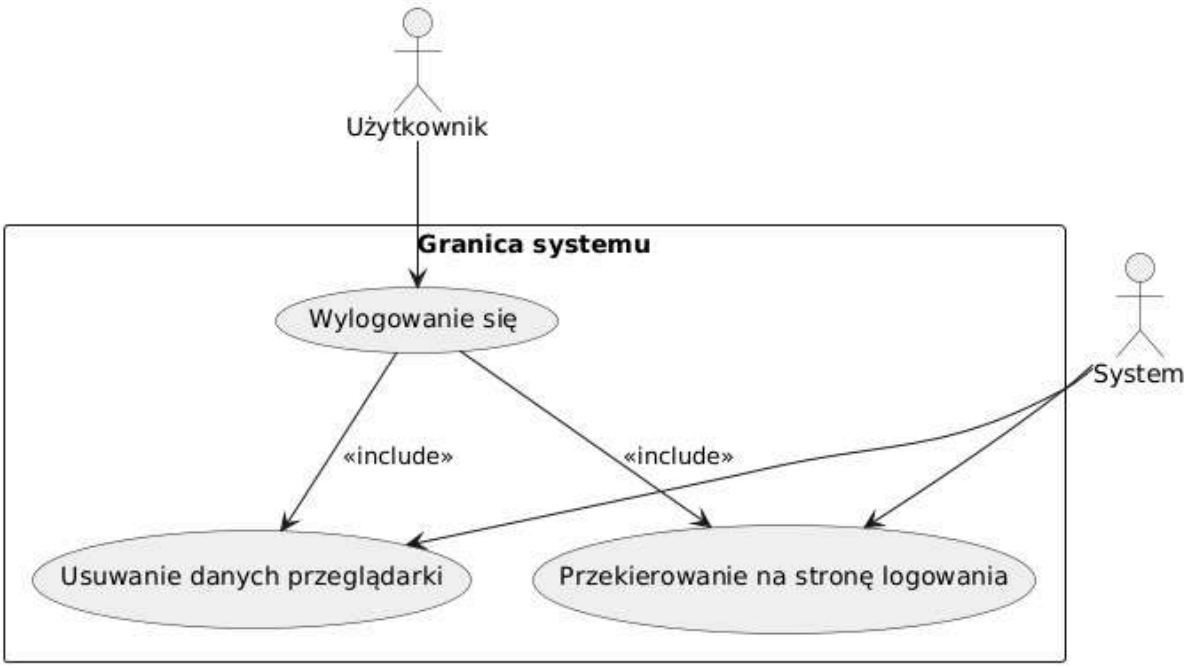
Rysunek 3.74 Diagram przypadków użycia — zakup przedmiotu w sklepie (źródło: opracowanie własne)



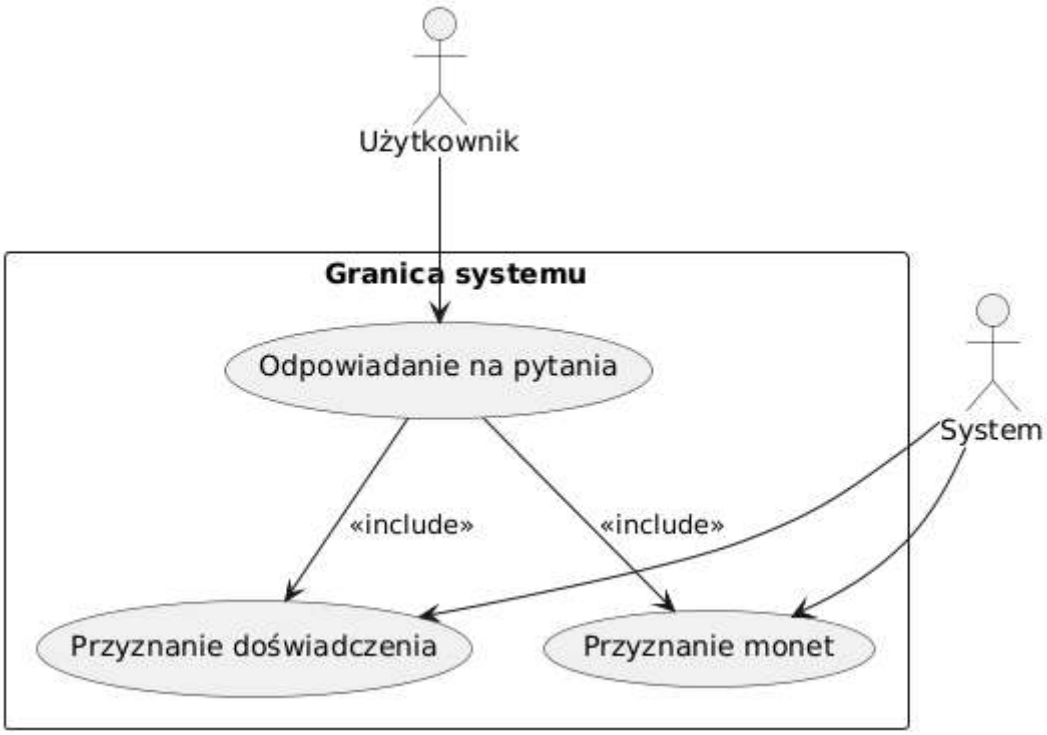
Rysunek 3.75 Diagram przypadków użycia — zakup edycji postaci (źródło: opracowanie własne)



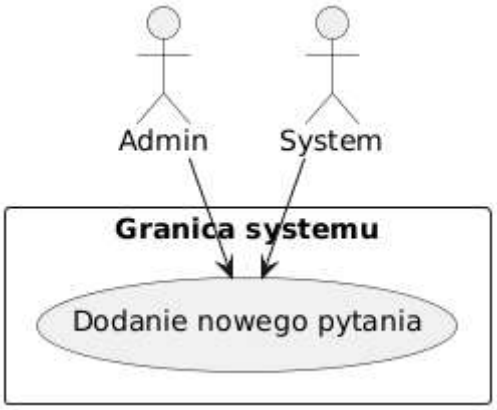
Rysunek 3.76 Diagram przypadków użycia — założenie przedmiotu z garderoby (źródło: opracowanie własne)



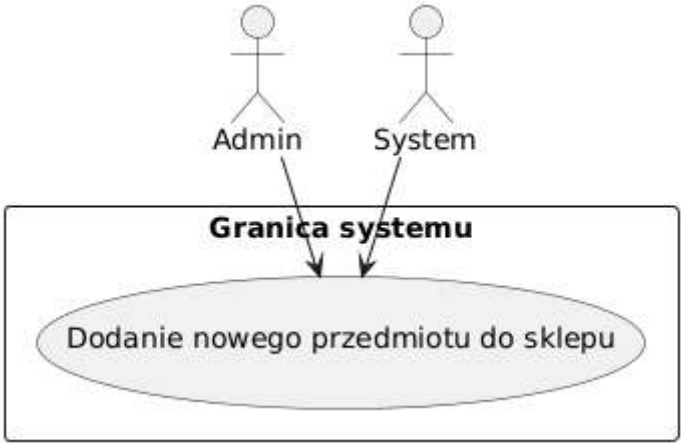
Rysunek 3.77 Diagram przypadków użycia — wylogowanie (źródło: opracowanie własne)



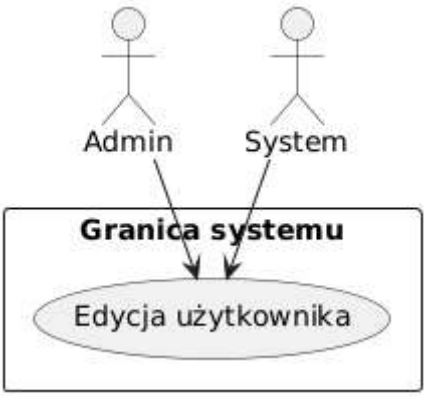
Rysunek 3.78 Diagram przypadków użycia — odpowiadanie na pytania (źródło: opracowanie własne)



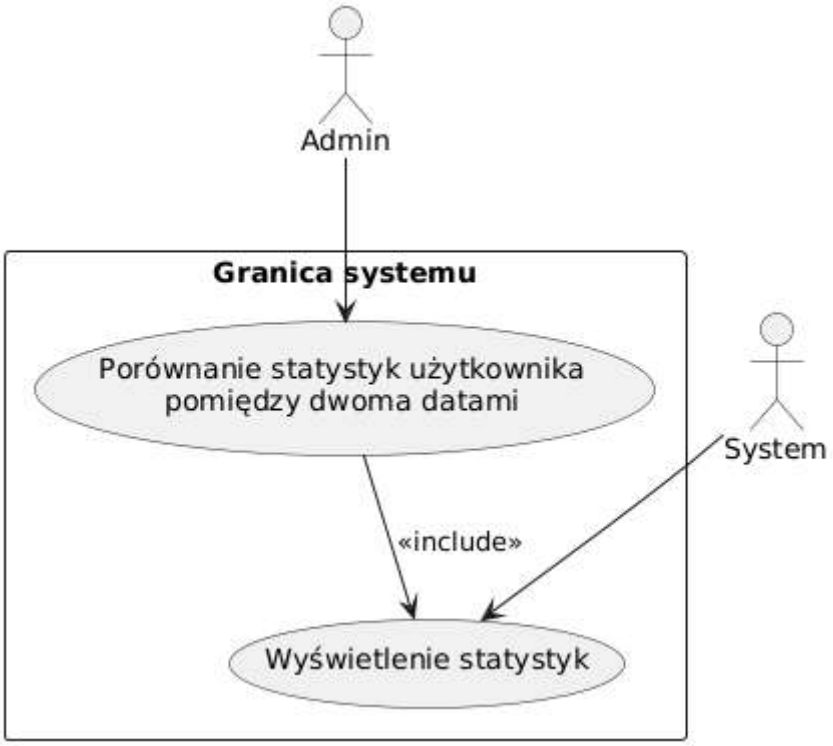
Rysunek 3.79 Diagram przypadków użycia — dodanie nowego pytania (źródło: opracowanie własne)



Rysunek 3.80 Diagram przypadków użycia — dodanie nowego pytania (źródło: opracowanie własne)



Rysunek 3.81 Diagram przypadków użycia — edycja użytkownika (źródło: opracowanie własne)



Rysunek 3.82 Diagram przypadków użycia — edycja użytkownika (źródło: opracowanie własne)

Wszystkie diagramy przedstawiają interakcje między użytkownikami (w tym administratorem) a systemem, pokazując, jak poszczególne funkcjonalności są realizowane w obrębie wyraźnie określonych granic systemu. Każdy diagram obrazuje sekwencję działań, gdzie jedna strona inicjuje operację, a druga — system — odpowiada za wykonanie określonych kroków, takich jak modyfikacja stanu danych, przekierowywanie, przyznawanie nagród czy prezentacja informacji. W ten sposób modele te ułatwiają zrozumienie przepływu procesów oraz podziału odpowiedzialności między aktorami i systemem.

4. Opis realizacji projektu

4.1 Wprowadzenie

Realizacja projektu SQULIO odbywała się według zaplanowanych uprzednio etapów, zaczynając od projektowania koncepcyjnego, implementacji, testowania, po finalne wdrożenie. Wykorzystana przez zespół metodyka Kanban pozwoliła na dynamiczne zarządzanie zadaniami i dostosowywanie harmonogramu do zmieniających się wymagań projektu. Dodatkowym wsparciem przy monitorowaniu postępu prac był opracowany dla projektu diagram Gantta.

4.2 Etapy tworzenia projektu

Przed rozpoczęciem realizacji projektu, został on podzielony na kilka kluczowych etapów obejmujących aspekty techniczne i organizacyjne.

4.2.1 Projektowanie stron i podstron

Pierwszy etap polegał na stworzeniu koncepcji i struktury aplikacji. W jego ramach zostały opracowane:

- Główne sekcje aplikacji,
- Schemat nawigacyjny,
- Projekt strony logowania i rejestracji,
- Projekt panelu użytkownika i panelu administracyjnego,
- Koncepcję personalizacji postaci.

Te działania pozwoliły na określenie kluczowych wymagań funkcjonalnych i stanowiły podstawę do kolejnych etapów implementacji.

4.2.2 Backend development — podstawy

W kolejnym kroku rozpoczęte zostały prace nad backendem aplikacji.
Skonfigurowano środowisko i przygotowano podstawowe funkcjonalności, takie jak:

- Konfiguracja serwera Express.js,
- Konfiguracja bazy danych MongoDB,
- Implementacja systemu rejestracji i logowania użytkowników,
- Tworzenie panelu administratora do zarządzania zawartością aplikacji.

Dzięki temu backend był gotowy do integracji z warstwą frontendową.

4.2.3 Frontend development — podstawy

Równolegle do prac nad backendem rozpoczęta została implementacja frontendu. Składało się na nią:

- Inicjalizacja projektu w React.js,
- Konfiguracja środowiska deweloperskiego,
- Tworzenie interfejsu użytkownika dla strony logowania i rejestracji,
- Pierwsze elementy systemu zarządzania postaciami graczy.

Umożliwiło to dalszą integrację z backendem oraz dodawanie kolejnych funkcji.

4.2.4 Backend development — główne funkcjonalności

Po przygotowaniu podstawowej struktury backendu rozpoczęto wdrażanie kluczowych mechanizmów gry, takich jak:

- System punktacji i poziomów użytkowników,
- Obsługa nagród za wykonanie codziennych zadań,

- Przyznawanie punktów i nagród za poprawne rozwiązania quizów.

4.2.5 Frontend development — panel użytkownika

W tym etapie rozwinięty został interfejs użytkownika i jego interakcje z backendem. Podczas tego etapu wdrożono:

- Ekran główny panelu użytkownika,
- Interakcję użytkownika z quizami,
- Integrację z systemem postaci i jej personalizacji,
- Obsługę komunikacji użytkownika z backendem.

Była to istotna część pracy zapewniająca intuicyjną i przyjazną obsługę aplikacji przez użytkownika końcowego.

4.2.6 Backend development — rozwinięcia

Po wdrożeniu kluczowych mechanizmów gry, backend został rozbudowany o dodatkowe funkcjonalności takie jak:

- System waluty w grze,
- Możliwość zakupu przedmiotów kosmetycznych dla postaci gracza,
- Mechanizm dodawania nowych pytań i poziomów przez administratorów.

Aplikacja stała się bardziej elastyczna i rozbudowana pod aspektem zarządzania.

4.2.7 Frontend development — panel administratorski

Jednym z końcowych etapów było wdrożenie panelu administracyjnego po stronie frontendu, umożliwiając administratorom na wygodne zarządzanie treściami na stronie. Obejmowało to:

- Stworzenie interfejsu UI dla panelu administratora,
- Integrację panelu administratora z backendem,
- Implementację funkcji administracyjnych, takich jak zarządzanie pytaniami i użytkownikami.

Panel administracyjny pozwolił na kontrolowanie i łatwe rozwijanie aplikacji w przyszłości.

4.2.8 Testowanie i poprawki

Po zakończeniu głównych prac deweloperskich zespół przystąpił do testowania aplikacji. Proces ten składał się z elementów takich jak:

- Testy funkcjonalne,
- Identyfikacja błędów i ich naprawa,
- Optymalizacja wydajności aplikacji,
- Testy manualne.

Wykryte podczas tego etapu błędy były na bieżąco poprawiane, co pozwoliło na zwiększenie stabilności aplikacji przed wdrożeniem.

4.2.9 Finalne dopracowanie i wdrożenie

Końcowym etapem projektu było wdrożenie aplikacji do środowiska produkcyjnego. W jego ramach wykonano:

- Wdrożenie i konfiguracja aplikacji w środowisku produkcyjnym,
- Finalne testy i sprawdzenie działania aplikacji w warunkach rzeczywistych,
- Monitorowanie działania aplikacji po wdrożeniu.

Dzięki temu aplikacja była gotowa do użytku przez użytkowników końcowych.

5. Wdrożenie i eksploatacja

5.1. Wdrożenie w procesie rozwoju oprogramowania

5.1.1. Wdrożenie aplikacji

Wymagania:

- zainstalowany i uruchomiony Proxmox 8.3.2
- zainstalowany Node.js v18.19.0
- zainstalowany npm: 9.2.0
- zainstalowana baza danych MongoDB v8.0.3
- skonfigurowany dostęp do VPN (Tailscale)

Kopiowanie i konfiguracja plików:

Aplikacja działa bez systemu kontroli wersji, co oznacza, że wszystkie pliki są zarządzane ręcznie bez użycia Git. Wszelkie zmiany w kodzie należy wprowadzać bezpośrednio na serwerze. Brak systemu kontroli wersji jest uzasadniony przez prostotę zarządzania kodem w zamkniętym środowisku oraz codzienne kopie zapasowe kontenera, które pozwalają na szybkie odzyskiwanie danych w przypadku awarii.

Instalowanie zależności:

Aby zainstalować zależności aplikacji, należy wykonać komendę:

npm install

Uruchomienie aplikacji lokalnie:

Uruchomienie aplikacji na serwerze odbywa się poprzez PM2, który zapewnia automatyczne restartowanie w przypadku błędów dzięki opcji -watch:

/home/squlio/frontend$ npm run build

/home/squlio/backend$ pm2 start server.js -watch

Aby zarządzać aplikacją i monitorować jej stan, można użyć komend PM2:

pm2 list

pm2 logs

pm2 restart all

Aplikacja działa na porcie 443, dzięki czemu jest dostępna tylko przez HTTPS, a cały ruch z HTTP (port 80) przekierowany jest na HTTPS.

5.1.2. Konfiguracja serwera

Środowisko uruchomieniowe: Aplikacja jest wdrożona w kontenerze działającym na serwerze Proxmox. Dostęp do serwera odbywa się wyłącznie za pośrednictwem VPN (Tailscale), co uniemożliwia nieautoryzowany dostęp. Firewall jest aktywny na poziomie serwera, routera oraz VPN, co dodatkowo zabezpiecza system.

Dostęp do bazy danych: Baza danych MongoDB działa w tym samym kontenerze co aplikacja. Dostęp do niej jest możliwy wyłącznie poprzez VPN (Tailscale) oraz narzędzia MongoDB Compass lub wiersz poleceń.

5.1.3. Kopie zapasowe i monitoring

Mechanizmy tworzenia kopii zapasowych: Codziennie o godzinie 7:30 oraz 22:30 wykonywana jest automatyczna kopia zapasowa całego kontenera przy użyciu narzędzi Proxmox. Dzięki temu możliwe jest szybkie odtworzenie systemu w przypadku awarii.

Monitoring działania kontenera: W przypadku awarii kontenera lub zaniku prądu, proxmox automatycznie uruchomi ponownie kontener. Dzięki temu zmniejszany jest czas niedostępności aplikacji.

Monitoring i utrzymanie: Do monitorowania i zarządzania aplikacją wykorzystano PM2, który automatycznie restartuje aplikację w razie błędów oraz dostarcza szczegółowe informacje o jej wydajności i zużyciu zasobów.

5.2. Wdrożenie produkcyjne

Wymagania:

- Działający serwer z Proxmox.
- Skonfigurowany dostęp do VPN (Tailscale).
- Firewall aktywny na poziomie serwera, routera oraz VPN.
- Zainstalowany Node.js oraz MongoDB.

Uruchomienie aplikacji na środowisku produkcyjnym: Po wdrożeniu aplikacji na serwerze, uruchomienie jej odbywa się poprzez PM2:

/home/squlio/frontend$ npm run build

/home/squlio/backend$ pm2 start server.js -watch

W przypadku restartu serwera lub błędów, PM2 automatycznie przywraca działanie aplikacji.

Zabezpieczenia:

- Aplikacja dostępna jest jedynie na porcie 443 (HTTPS).
- Zdalne logowanie na konto root jest wyłączone.
- Połączenia z bazą danych możliwe są wyłącznie przez VPN (Tailscale).
- Firewall zabezpiecza ruch na poziomie serwera, routera i VPN.

Dzięki takiemu podejściu system zapewnia wysoki poziom bezpieczeństwa, wydajności oraz niezawodności.

5.3 Działanie frontendu

5.3.1 Wybór technologii

React

Do implementacji warstwy frontendowej aplikacji wybraliśmy bibliotekę React, która jest jedną z najpopularniejszych technologii do tworzenia interaktywnych interfejsów użytkownika. Wybór ten został podyktowany kilkoma kluczowymi czynnikami:

- Modularność i komponentowa architektura — React pozwala na podział aplikacji na niezależne, wielokrotnego użytku komponenty, co znacząco ułatwia zarządzanie kodem, jego rozwój oraz testowanie.
- Virtual DOM — React wykorzystuje wirtualny model DOM, co przyspiesza proces renderowania i optymalizuje działanie aplikacji. Zmiany są porównywane z poprzednim stanem, a aktualizowane są tylko te fragmenty interfejsu, które rzeczywiście tego wymagają.
- Deklaratywne podejście — React umożliwia opisywanie UI w sposób deklaratywny, co sprawia, że kod jest czytelniejszy i łatwiejszy w utrzymaniu.
- Bogaty ekosystem — Dzięki dużej społeczności oraz wsparciu firm takich jak Facebook (obecnie Meta), React posiada szeroki wybór gotowych bibliotek i narzędzi, które ułatwiają rozwój aplikacji.
- Wsparcie dla TypeScript — W celu zapewnienia większej stabilności i minimalizacji błędów zdecydowaliśmy się na wykorzystanie TypeScript, który dodaje statyczne typowanie do JavaScriptu, co ułatwia utrzymanie kodu na dłuższą metę.

- Obsługa nawigacji i routingu — W naszej aplikacji React współpracuje z biblioteką React Router, co umożliwia dynamiczne zarządzanie trasami w aplikacji jedno- i wielostronicowej (SPA/MPA). Dzięki temu użytkownicy mogą płynnie przechodzić między różnymi sekcjami aplikacji bez konieczności pełnego przeładowania strony.
- Łatwa integracja z backendem — React pozwala na łatwą komunikację z backendem za pomocą REST API, co jest kluczowe dla naszej aplikacji, gdzie dane użytkownika, postępy w grze oraz waluty muszą być synchronizowane między frontendem a serwerem.

Nasza aplikacja korzysta z wersji React 18.3.1, co zapewnia najnowsze funkcje oraz lepszą wydajność i optymalizację w działaniu aplikacji.

Vite

Do budowy i zarządzania aplikacją frontendową wybraliśmy Vite, nowoczesne narzędzie do bundlingu i zarządzania modułami. Vite to rozwiązanie, które znacząco przyspiesza proces developmentu w porównaniu do tradycyjnych bundlerów, takich jak Webpack.
Zalety Vite:

- Błyskawiczne uruchamianie projektu — Vite działa znacznie szybciej niż Webpack dzięki wykorzystaniu natywnych modułów ES (ESM) oraz pre-kompilacji kodu.
- Hot Module Replacement (HMR) — pozwala na natychmiastowe odświeżanie komponentów po wprowadzeniu zmian w kodzie, bez konieczności ponownego uruchamiania aplikacji, co znacząco usprawnia pracę deweloperów.
- Wsparcie dla TypeScript — Vite natywnie obsługuje TypeScript, dzięki czemu nie wymaga dodatkowej konfiguracji, co pozwala skupić się na pisaniu kodu zamiast dostosowywaniu narzędzi.
- Lekkość i optymalizacja — generuje zoptymalizowane pliki wynikowe, które ładują się szybciej w przeglądarce użytkownika, co poprawia ogólną wydajność aplikacji.
- Łatwa integracja z React — Vite został zaprojektowany z myślą o frameworkach takich jak React, Vue czy Svelte, co sprawia, że jest łatwy w użyciu i oferuje domyślną konfigurację dostosowaną do tych technologii.
- Efektywne zarządzanie zależnościami — w przeciwieństwie do Webpacka, który przetwarza cały kod przed uruchomieniem aplikacji, Vite działa na zasadzie „on-demand”, co oznacza, że przetwarza tylko te pliki, które są faktycznie używane.

Nasza aplikacja korzysta z wersji Vite 5.4.8, co zapewnia nowoczesne wsparcie i optymalizację procesu budowania aplikacji.
Komunikacja z backendem
Frontend naszej aplikacji komunikuje się z backendem za pomocą REST API, co zapewnia wydajne zarządzanie danymi użytkownika oraz jego postępami w grze. Backend został zaimplementowany w Node.js, co jest naturalnym wyborem w ekosystemie JavaScript, pozwalającym na efektywne zarządzanie żądaniami i szybkie operacje na bazie danych.
Struktura komunikacji:

- Pobieranie danych użytkownika — Aplikacja frontendowa pobiera dane zalogowanego użytkownika, takie jak ilość zgromadzonej waluty, poziom doświadczenia oraz zakupione przedmioty.
- Aktualizacja postępów — Po udzieleniu odpowiedzi na pytanie lub wykonaniu zadania, dane użytkownika są wysyłane do backendu w celu zapisania nowych wyników.
- Obsługa autoryzacji — Komunikacja z serwerem wymaga uwierzytelnienia użytkownika, co odbywa się za pomocą tokenów JWT (JSON Web Token) lub sesji.

- Płynna synchronizacja danych — Dzięki wykorzystaniu asynchronicznych zapytań HTTP (fetch, axios), aplikacja jest w stanie dynamicznie aktualizować interfejs użytkownika bez konieczności przeładowania strony.

5.3.2 Budowanie aplikacji

Po zakończeniu procesu developmentu aplikacja musi zostać przygotowana do wdrożenia. W tym celu korzystamy z polecenia:
npm run build
Polecenie to powoduje uruchomienie procesu budowania aplikacji w Vite, co skutkuje:

- Minifikacją kodu — Wszystkie pliki JavaScript i CSS są minimalizowane, co zmniejsza ich rozmiar i przyspiesza ładowanie strony.
- Tree Shaking — Usuwane są nieużywane fragmenty kodu, co prowadzi do dalszej optymalizacji.
- Generowaniem zoptymalizowanych plików statycznych — Aplikacja jest kompilowana do statycznych plików HTML, CSS i JavaScript, które mogą być hostowane na serwerze lub w usłudze CDN.
- Obsługą cache — Pliki wynikowe otrzymują unikalne hashe, co pozwala na lepsze zarządzanie pamięcią podręczną przeglądarki i unikanie problemów z odświeżaniem strony.

W pliku package.json nasza aplikacja zawiera następujące skrypty:

```
"scripts": {
  "dev": "vite",
  "build": "tsc -b && vite build",
  "lint": "eslint .",
  "preview": "vite preview"
},
```

Rysunek 5.1 Skrypty w pliku package.json (źródło: opracowanie własne)

- dev — Uruchamia środowisko deweloperskie.
- build — Buduje aplikację, najpierw kompilując TypeScript, a następnie generując zoptymalizowane pliki.
- lint — Sprawdza kod pod kątem błędów i zgodności ze standardami ESLint.
- preview — Uruchamia podgląd zbudowanej aplikacji.

Wygenerowane pliki trafiają do katalogu dist/, który może zostać bezpośrednio wdrożony na serwer produkcyjny lub do chmury (np. Vercel, Netlify, AWS S3). Proces ten zapewnia wysoką wydajność i optymalizację działania aplikacji.

5.3.3 Struktura frontendu

```
|-- README.md
|-- cypress
|   |-- e2e
|   |   `-- answer_question.cy.js
|   |-- integration
|   `-- screenshots
|-- cypress.config.js
|-- eslint.config.js
|-- index.html
|-- package-lock.json
|-- package.json
|-- src
|   |-- App.css
|   |-- App.tsx
|   |-- assets
|   |   `-- react.svg
|   |-- components
|   |   |-- \ Activation.tsx
|   |   |-- Achivments.css
|   |   |-- Achivments.tsx
|   |   |-- Activation.css
|   |   |-- AdminDashboard.css
|   |   |-- AdminDashboard.tsx
|   |   |-- Background.css
|   |   |-- Background.tsx
|   |   |-- CharacterBuild.css
|   |   |-- CharacterBuild.tsx
|   |   |-- DailyQuestions.css
|   |   |-- DailyQuestions.tsx
|   |   |-- DailyStats.css
|   |   |-- DailyStats.tsx
|   |   |-- Description.css
|   |   |-- Description.tsx
|   |   |-- EditUser.tsx
|   |   |-- EnglishQuestions.css
|   |   |-- EnglishQuestions.tsx
|   |   |-- Header.css
|   |   |-- Header.tsx
|   |   |-- HeartShop.css
|   |   |-- HeartShop.tsx
|   |   |-- ItemShop.css
|   |   |-- ItemShop.tsx
|   |   |-- Login.css
|   |   |-- Login.tsx
|   |   |-- MainContent.css
|   |   |-- MainContent.tsx
|   |   |-- MainMenu.css
|   |   |-- MainMenu.tsx
|   |   |-- Mascot.css
|   |   |-- Mascot.tsx
|   |   |-- MathQuestions.css
|   |   |-- MathQuestions.tsx
|   |   |-- ProgrammingQuestions.css
|   |   |-- ProgrammingQuestions.tsx
|   |   |-- Register.css
|   |   |-- Register.tsx
|   |   |-- ResetPassword.css
|   |   |-- ResetPassword.tsx
|   |   |-- ScienceQuestions.css
|   |   |-- ScienceQuestions.tsx
|   |   |-- StartButton.css
|   |   |-- StartButton.tsx
|   |   |-- Wardrobe.css
|   |   |-- Wardrobe.tsx
|   |   |-- Zadania.css
|   |   `-- Zadania.tsx
|   |-- index.css
|   |-- main.tsx
|   |-- styles
|   `-- vite-env.d.ts
|-- tsconfig.app.json
|-- tsconfig.app.tsbuildinfo
|-- tsconfig.json
|-- tsconfig.node.json
|-- tsconfig.node.tsbuildinfo
`-- vite.config.ts
```

Rysunek 5.2 Struktura frontendu (źródło: opracowanie własne)

5.4 Działanie backendu

5.4.1 Technologia backendowa: Node.js

Node.js to środowisko uruchomieniowe JavaScript oparte na silniku V8 firmy Google, które umożliwia wykonywanie kodu JavaScript poza przeglądarką. Jego architektura opiera się na modelu zdarzeniowym z nieblokującym wejściem/wyjściem, co pozwala na obsługę dużej liczby jednoczesnych połączeń przy zachowaniu wysokiej wydajności. Dzięki tym cechom, Node.js jest często wykorzystywany w aplikacjach czasu rzeczywistego, takich jak czaty czy systemy powiadomień.

Framework backendowy: Express.js

Express.js to minimalistyczny i elastyczny framework dla Node.js, który upraszcza proces tworzenia aplikacji webowych i API. Zapewnia zestaw narzędzi i funkcji, takich jak zarządzanie trasami, middleware oraz obsługę szablonów, co przyspiesza rozwój aplikacji. Express.js jest często określany jako standard de facto wśród frameworków serwerowych dla Node.js.

Przykład zastosowania:

Połączenie Node.js i Express.js umożliwia tworzenie wydajnych i skalowalnych aplikacji webowych. Przykładowo, w pracy "Comparison of Node.Js and Spring Boot in Web Development"[4] porównano wydajność aplikacji opartych na tych dwóch technologiach, wskazując na zalety asynchronicznej obsługi żądań w Node.js z wykorzystaniem Express.js.

5.4.2 Biblioteki i moduły używane w aplikacji

Aplikacja korzysta z szerokiego zestawu bibliotek Node.js, które usprawniają jej działanie i zwiększają bezpieczeństwo.

Obsługa serwera i routingu

- express — główny framework backendowy, zarządzający trasami i middleware.
- path — moduł do zarządzania ścieżkami plików na serwerze.
- fs — moduł umożliwiający operacje na systemie plików (np. wczytywanie certyfikatów SSL).

Obsługa baz danych

- mongodb — natywny sterownik do łączenia się z MongoDB.
- mongoose — warstwa ORM umożliwiająca modelowanie danych i obsługę operacji CRUD.
- connect-mongo — umożliwia przechowywanie sesji użytkowników w MongoDB.

Obsługa sesji i ciasteczek

- express-session — obsługuje sesje użytkowników, przechowując je w bazie danych.
- cookie-parser — umożliwia przetwarzanie ciasteczek HTTP.

Middleware dla bezpieczeństwa

- helmet — zabezpiecza aplikację przed popularnymi atakami (np. XSS, clickjacking).
- express-rate-limit — ogranicza liczbę żądań HTTP, chroniąc przed atakami brute-force.
- dotenv — umożliwia przechowywanie zmiennych środowiskowych w pliku .env, co zwiększa bezpieczeństwo aplikacji.

Obsługa uwierzytelniania i autoryzacji

- bcrypt — służy do szyfrowania haseł użytkowników.
- jsonwebtoken — obsługuje autoryzację użytkowników za pomocą tokenów JWT.

Obsługa wiadomości e-mail

- nodemailer — biblioteka do wysyłania e-maili, wykorzystywana do powiadomień i odzyskiwania hasła.

Obsługa czasu i stref czasowych

- moment-timezone — obsługuje zarządzanie czasem, np. konwersję dat w różnych strefach czasowych.

Harmonogramowanie zadań (cron jobs)

- node-cron — umożliwia wykonywanie zadań cyklicznych, np. codzienne generowanie quizów, aktualizację statystyk użytkowników czy resetowanie postępu.

Walidacja danych

- express-validator — zestaw funkcji do walidacji i sanitizacji danych wejściowych użytkownika.

```
backend > {} package.json > ...
1   {
2     "name": "backend",
3     "version": "1.0.0",
4     "description": "",
5     "main": "index.js",
      ▷Debuguj
6     "scripts": {
7       "test": "echo \"Error: no test specified\" && exit 1"
8     },
9     "keywords": [],
10    "author": "",
11    "license": "ISC",
12    "dependencies": {
13      "bcrypt": "^5.1.1",
14      "body-parser": "^1.20.3",
15      "connect-flash": "^0.1.1",
16      "connect-mongo": "^5.1.0",
17      "cookie-parser": "^1.4.7",
18      "dotenv": "^16.4.5",
19      "express": "^4.21.1",
20      "express-rate-limit": "^7.5.0",
21      "express-session": "^1.18.1",
22      "express-validator": "^7.2.0",
23      "http-proxy": "^1.18.1",
24      "jsonwebtoken": "^9.0.2",
25      "moment-timezone": "^0.5.46",
26      "mongodb": "^6.9.0",
27      "mongoose": "^8.7.2",
28      "node-cron": "^3.0.3",
29      "nodemailer": "^6.9.15",
30      "pug": "^3.0.3",
31      "save": "^2.9.0"
32    }
33  }
```

Rysunek 5.3 Plik package.json (źródło: opracowanie własne)

```
.
|-- app.js
|-- config
|   |-- connection.js
|   `-- database.js
|-- controllers
|   |-- AchievementsController.js
|   |-- ActivateUserController.js
|   |-- AdminController.js
|   |-- CharacterController.js
|   |-- DailyQuestionsController.js
|   |-- DailyStatsController.js
|   |-- EnglishQuestlionsController.js
|   |-- GetCharacterController.js
|   |-- HeartShopController.js
|   |-- ItemShopController.js
|   |-- LoginController.js
|   |-- MathQuestionsController.js
|   |-- ProgramingQuestlionsController.js
|   |-- RegisterController.js
|   |-- ResetPasswordController.js
|   |-- ScienceQuestlionsController.js
|   |-- TasksController.js
|   |-- UserStatsController.js
|   `-- WardrobeController.js
|-- cron
|   |-- dailyQuizCron.js
|   |-- dailyStatistics.js
|   |-- restoreHp.js
|   `-- userDataCron.js
|-- ecosystem.config.js
|-- emailservices
|   `-- emailservice.js
|-- logs
|   `-- attack_logs.txt
|-- middlewares
|   |-- attackDetection.js
|   |-- authenticateUser.js
|   |-- checkEditableCharacter.js
|   `-- tasks.js
|-- models
|   |-- Achievements.js
|   |-- DailyQuestions.js
|   |-- DailyQuiz.js
|   |-- DailyStatistics.js
|   |-- EnglishQuestions.js
|   |-- ItemShop.js
|   |-- MathsQuestions.js
|   |-- ProgramingQuestions.js
|   |-- ScienceQuestions.js
|   |-- Tasks.js
|   `-- User.js
|-- mysql-apt-config_0.8.30-1_all.deb
|-- mysql-apt-config_0.8.33-1_all.deb
|-- package-lock.json
|-- package.json
|-- routes
|   `-- index.js
|-- server.js
|-- ssl_keys
|   |-- squlio.crt
|   `-- squlio.key
`-- utils
    |-- codeValidation.js
    |-- emailValidation.js
    |-- passwordValidation.js
    `-- usernameValidation.js
```

Rysunek 5.4 Struktura backendu aplikacji (źródło: opracowanie własne)

5.4.3 Połączenie z bazą danych:

Połączenie aplikacji backendowej z bazą MongoDB zostało zaimplementowane przy użyciu biblioteki Mongoose, co zapewnia większą elastyczność i bezpieczeństwo operacji na danych. W kodzie aplikacji wykorzystano asynchroniczną funkcję connectDB(), która nawiązuje połączenie z bazą danych i obsługuje potencjalne błędy.

plik database.js

```
backend > config > JS database.js > connectDB
1   const mongoose = require('mongoose');
2   const config = require('./connection');
3
4   const connectDB = async () => {
5     try {
6       await mongoose.connect(config.mongoURI, {
7       });
8       console.log('Polaczono z baza danych MongoDB (po zmianie serwera)');
9     } catch (error) {
10      console.error('Blad polaczenia z baza danych MongoDB:', error);
11      process.exit(1);
12    }
13  };
14
15  module.exports = connectDB;
16
```

Rysunek 5.5 Plik database.js (źródło: opracowanie własne)

W powyższym kodzie zastosowano następujące mechanizmy:

- Asynchroniczność — użycie async/await zapewnia nieblokującą obsługę połączenia, co pozwala na utrzymanie wysokiej wydajności aplikacji.
- Obsługa wyjątków — w przypadku błędu podczas połączenia, aplikacja zwraca odpowiedni komunikat (console.error), a następnie wymusza zakończenie procesu (process.exit(1)).
- Konfiguracja połączenia — adres serwera bazy danych MongoDB jest przechowywany w osobnym pliku konfiguracyjnym (connection.js), co ułatwia zarządzanie danymi dostępowymi i zwiększa bezpieczeństwo aplikacji.

Konfiguracja połączenia

W celu zapewnienia modularności kodu, dane konfiguracyjne dotyczące połączenia z bazą danych zostały umieszczone w osobnym pliku connection.js.

```
backend > config > JS connection.js > ...
1   module.exports = {
2       mongoURI: 'mongodb://localhost:27017'
3   };
4
```

Rysunek 5.6 plik connection.js (źródło: opracowanie własne)

Takie podejście pozwala na:

- Łatwą zmianę adresu bazy danych — np. w środowisku produkcyjnym można zastąpić localhost adresem instancji bazy danych w chmurze.
- Lepszą organizację kodu — separacja logiki aplikacyjnej od konfiguracji.
- Bezpieczeństwo — w przypadku bardziej rozbudowanych systemów, dane dostępowe mogą być przechowywane w zmiennych środowiskowych (process.env.MONGO_URI), co zabezpiecza je przed nieautoryzowanym dostępem.

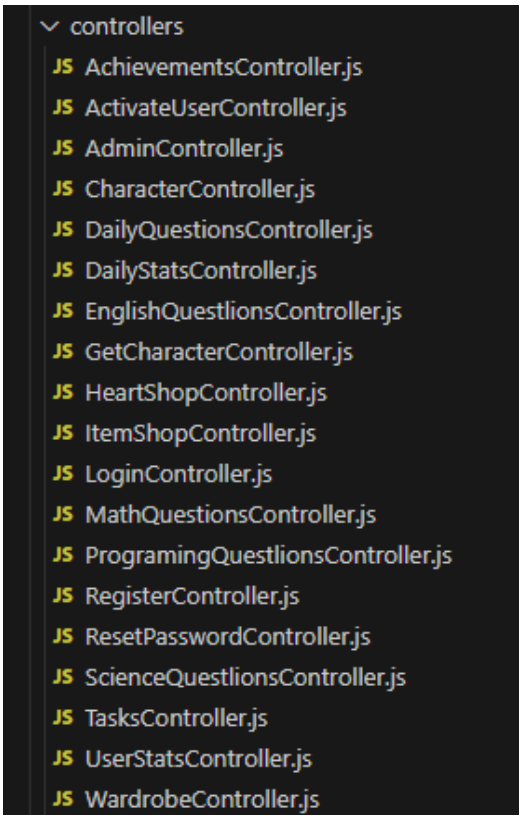
5.4.4 Kontrolery

W architekturze aplikacji backendowych kontrolery pełnią kluczową rolę w zarządzaniu logiką biznesową oraz obsłudze żądań użytkowników. W tym projekcie stanowią warstwę odpowiedzialną za przetwarzanie żądań HTTP, komunikację z warstwą modeli oraz zwracanie odpowiedzi do użytkownika

Funkcja kontrolerów w aplikacji

Główne zadania kontrolerów obejmują:

- Obsługa żądań HTTP — kontrolery odbierają żądania od użytkowników (np. GET, POST, PUT, DELETE) i odpowiednio je przetwarzają.
- Interakcja z modelami danych — kontrolery komunikują się z modelem bazy danych, aby pobierać, modyfikować lub usuwać dane.
- Zarządzanie logiką — zawierają operacje związane z walidacją, autoryzacją użytkowników oraz innymi aspektami przetwarzania danych.
- Zwracanie odpowiedzi do klienta — po przetworzeniu żądania kontroler zwraca odpowiedź w odpowiednim formacie.



Rysunek 5.7 Lista kontrolerów w aplikacji (źródło: opracowanie własne)

5.4.5 Pliki cron

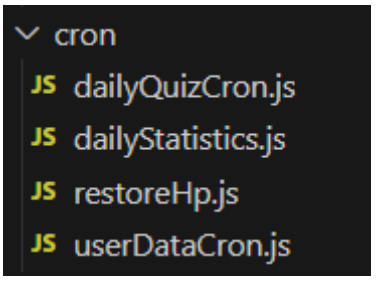
W nowoczesnych aplikacjach backendowych często pojawia się potrzeba wykonywania cyklicznych operacji, takich jak aktualizowanie statystyk, zarządzanie użytkownikami czy generowanie nowych danych. W tym celu stosuje się zadania cron, które umożliwiają automatyczne wykonywanie określonych operacji w zdefiniowanych interwałach czasowych. W aplikacji backendowej wykorzystano bibliotekę node-cron, która pozwala na harmonogramowanie zadań w środowisku Node.js.

Rola zadań cron w aplikacji
Pliki cron w aplikacji pełnią kluczową rolę w:

- Automatyzacji procesów — zapewniają regularne wykonywanie zadań bez konieczności interakcji użytkownika.
- Optymalizacji wydajności — eliminują potrzebę ciągłego sprawdzania warunków w czasie rzeczywistym, co zmniejsza obciążenie serwera.
- Zapewnieniu spójności danych — umożliwiają aktualizację i synchronizację danych w określonych odstępach czasu.

Dzięki zastosowaniu plików cron, możliwe jest realizowanie zadań takich jak:

- Generowanie codziennego quizu — codziennie o północy system tworzy nowy zestaw pytań dla użytkowników.
- Aktualizacja dziennych statystyk — obliczanie liczby aktywnych użytkowników, średniego poziomu ELO i innych statystyk.
- Odnawianie punktów życia (HP) użytkowników — co godzinę system odnawia punkty HP użytkowników, jeśli spełniają określone warunki.
- Resetowanie postępu użytkowników — o północy system resetuje dzienny postęp użytkowników, umożliwiając rozpoczęcie nowej sesji.



Rysunek 5.8 Pliki cron w aplikacji (źródło: opracowanie własne)

5.4.6 Automatyczna wysyłka wiadomości e-mail w aplikacji backendowej

Współczesne aplikacje internetowe często wymagają funkcjonalności związanych z automatycznym wysyłaniem wiadomości e-mail, np. w celu weryfikacji konta, resetowania hasła, czy powiadamiania użytkowników o ważnych zdarzeniach. W aplikacji backendowej wykorzystano w tym celu bibliotekę Nodemailer, która umożliwia wysyłanie wiadomości e-mail z poziomu środowiska Node.js. Dzięki temu możliwe jest bezpośrednie integrowanie funkcji e-mail z logiką aplikacji, co usprawnia komunikację między systemem a użytkownikami.

5.4.7 Implementacja mechanizmu wysyłki wiadomości

Do realizacji wysyłki e-mail wykorzystano bibliotekę Nodemailer, która stanowi jedno z najpopularniejszych rozwiązań dla aplikacji opartych na Node.js. Biblioteka ta

obsługuje różne protokoły e-mail, w tym SMTP (Simple Mail Transfer Protocol), co umożliwia integrację z popularnymi dostawcami usług pocztowych.

Konfiguracja połączenia SMTP

Kod konfiguracji klienta e-mailowego w aplikacji wygląda następująco:

```
const nodemailer = require('nodemailer');

const transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: {
      user: 'squlio.noreply@gmail.com',
      pass: 'cpfs jvfq vscf qxry'
    }
});
```

Rysunek 5.9 Pliki cron w aplikacji (źródło: opracowanie własne)

Biblioteka Nodemailer — zapewnia obsługę wysyłki e-mail z poziomu serwera.
nodemailer.createTransport() — tworzy instancję klienta SMTP.
service: 'gmail' — określa dostawcę usługi e-mail (w tym przypadku Gmail).
Dane uwierzytelniające (auth) — logowanie do konta e-mailowego odbywa się poprzez podanie adresu użytkownika oraz hasła aplikacyjnego (wygenerowanego w ustawieniach konta Gmail).
W rzeczywistej aplikacji, aby zwiększyć bezpieczeństwo, dane uwierzytelniające powinny być przechowywane w zmiennych środowiskowych, zamiast być zapisane bezpośrednio w kodzie.

Wysyłanie wiadomości e-mail

Po skonfigurowaniu klienta e-mailowego można skorzystać z funkcji sendEmail(), która umożliwia wysyłkę wiadomości do użytkowników:

```
function sendEmail(to, subject, text) {
    const mailOptions = {
      from: 'squlio.noreply@gmail.com',
      to,
      subject,
      text
    };

    transporter.sendMail(mailOptions, (error, info) => {
      if (error) {
        console.error('Błąd podczas wysyłania emaila:', error);
      } else {
        console.log('Email został wysłany:', info.response);
      }
    });
}

module.exports = { sendEmail };
```

Rysunek 5.10 Konfiguracja wysyłki e-maili (źródło: opracowanie własne)

5.4.8 Middleware

Middleware, czyli oprogramowanie pośredniczące, stanowi istotny element architektury aplikacji backendowych. Jest to warstwa kodu, która pośredniczy pomiędzy żądaniami HTTP a ich obsługą przez serwer, umożliwiając modyfikację żądań i odpowiedzi przed ich ostatecznym przetworzeniem. W aplikacji backendowej middleware pełni kluczową rolę w zakresie autoryzacji, bezpieczeństwa, ograniczania liczby żądań oraz walidacji danych wejściowych[8].

Definicja i funkcje middleware

Middleware można określić jako funkcję, która posiada dostęp do obiektu żądania (request), odpowiedzi (response) oraz kolejnej funkcji w cyklu przetwarzania żądania, zwanej next(). Może ono:

- Modyfikować obiekt żądania lub odpowiedzi — np. dodając do niego dodatkowe informacje,
- Przekazywać kontrolę do kolejnego middleware lub kontrolera — umożliwiając sekwencyjne przetwarzanie żądań,
- Zatrzymywać przetwarzanie żądania — w przypadku wykrycia błędów lub nieautoryzowanego dostępu,
- Obsługiwać błędy — umożliwiając scentralizowaną obsługę wyjątków.

Najważniejszymi middleware w aplikacji są zawarte w plikach attackDetection.js oraz authenticateUser.js.

Plik odpowiada za następujące rzeczy:

Middleware loginLimiter wykorzystuje bibliotekę express-rate-limit do ograniczenia liczby prób logowania w określonym czasie. Jeśli użytkownik przekroczy limit 10 prób logowania w ciągu 15 minut, jego adres IP zostanie zablokowany na 15 minut, a użytkownik otrzyma komunikat o zbyt wielu próbach logowania.

Blokowanie i odblokowywanie adresów IP:

Funkcja blockIP dodaje adres IP do listy zablokowanych adresów, uniemożliwiając dalsze próby logowania z tego adresu.
Funkcja unblockIP usuwa adres IP z listy zablokowanych adresów, umożliwiając ponowne próby logowania z tego adresu.
Sprawdzanie, czy adres IP jest zablokowany:

Middleware isBlocked sprawdza, czy adres IP klienta znajduje się na liście zablokowanych adresów. Jeśli adres IP jest zablokowany, użytkownik otrzymuje komunikat o zablokowaniu dostępu.
Logowanie podejrzanej aktywności:

Funkcja logSuspiciousActivity rejestruje podejrzaną aktywność, zapisując informacje o adresie IP, powodzie podejrzenia oraz ewentualnym adresie e-mail użytkownika. Funkcja ta może również blokować adres IP, jeśli powód podejrzenia nie jest związany z zbyt wieloma próbami logowania.
Wykrywanie prób SQL/NoSQL Injection:

Middleware detectInjection analizuje dane wejściowe w żądaniach HTTP pod kątem wzorców charakterystycznych dla ataków SQL/NoSQL Injection. Jeśli wykryje podejrzane dane, rejestruje podejrzaną aktywność i zwraca odpowiedni komunikat o błędzie.

Powiadamianie administratora o podejrzanej aktywności:

Funkcja notifyAdmin wysyła e-mail do administratora z informacjami o podejrzanej aktywności, w tym adresie IP, powodzie podejrzenia, nagłówkach żądania oraz informacjach o połączeniu.

Powiadamianie użytkownika o podejrzanej aktywności:

Funkcja notifyUser wysyła e-mail do użytkownika z informacjami o podejrzanej aktywności na jego koncie, w tym adresie IP i powodzie podejrzenia.

Middleware authenticateUser.js pełni kluczową rolę w zabezpieczaniu aplikacji poprzez zapewnienie, że tylko uwierzytelnieni użytkownicy mają dostęp do chronionych zasobów.

Jego znaczenie można opisać w następujący sposób:

Weryfikacja tożsamości użytkownika:

Middleware authenticateUser wykorzystuje tokeny JWT (JSON Web Tokens) do weryfikacji tożsamości użytkowników. Tokeny JWT są bezpiecznym sposobem przekazywania informacji między klientem a serwerem, ponieważ są podpisane cyfrowo. Middleware sprawdza, czy token JWT jest obecny w nagłówkach żądania HTTP i czy jest poprawny.

Ochrona zasobów:

Middleware authenticateUser chroni zasoby aplikacji, zapewniając, że tylko uwierzytelnieni użytkownicy mają do nich dostęp. Jeśli token JWT jest nieobecny lub nieprawidłowy, middleware zwraca odpowiedni komunikat o błędzie (401 Unauthorized lub 403 Forbidden), uniemożliwiając dostęp do chronionych zasobów.

```
backend > middlewares > JS authenticateUser.js > ...
1   const jwt = require('jsonwebtoken');
2
3   const authenticateUser = (req, res, next) => {
4       const token = req.headers.authorization?.split(' ')[1];
5       if (!token) return res.status(401).json({ message: 'Brak autoryzacji8' });
6
7       jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
8           if (err) return res.status(403).json({ message: 'Token nieprawidłowy' });
9
10          req.user = decoded;
11          next();
12      });
13  };
14
15  module.exports = authenticateUser;
```

Rysunek 5.11 Zawartość pliku authenticateUser.js (źródło: opracowanie własne)

5.4.9 Modele

Modele danych odgrywają kluczową rolę w organizacji, przechowywaniu i manipulacji danymi przechowywanymi w bazie danych. W analizowanej aplikacji backendowej wykorzystano bazę danych MongoDB, która operuje na dokumentach w formacie BSON (Binary JSON). Do zarządzania danymi zastosowano bibliotekę

Mongoose, która pełni funkcję Object Data Modeling (ODM) dla MongoDB, ułatwiając definiowanie struktur danych oraz operowanie na nich.

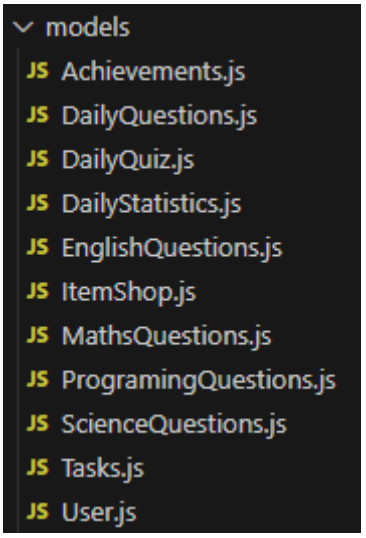
Definicja i rola modeli w aplikacji

Model w kontekście aplikacji backendowej jest abstrakcyjnym odwzorowaniem struktury danych w bazie, umożliwiającym ich przechowywanie, pobieranie i modyfikowanie. Główne funkcje modeli obejmują:

- Reprezentację struktury danych — modele definiują pola, ich typy oraz relacje między danymi,
- Obsługę operacji CRUD (Create, Read, Update, Delete) — ułatwiają interakcję z bazą danych,
- Walidację danych — zapewniają zgodność zapisanych danych z wymaganiami aplikacji,
- Zarządzanie relacjami — mimo że MongoDB jest bazą nierelacyjną, modele mogą odwoływać się do innych dokumentów poprzez identyfikatory.

Struktura modeli w aplikacji

W aplikacji backendowej modele są umieszczone w katalogu models/, co pozwala na modularną organizację kodu i ułatwia jego zarządzanie.



Rysunek 5.12 Zawartość katalogu /models/ (źródło: opracowanie własne)

Struktura poszczególnych modeli

Ze względu na kluczową rolę, jaką modele odgrywają w funkcjonowaniu aplikacji, podjęto decyzję o przedstawieniu ich pełnej struktury

```
d > models > JS Achievements.js > ...
const mongoose = require('mongoose');

const AchievementsShema = new mongoose.Schema({
    task: { type: String, required: true },
    levels: [
    {
      1: { type: Number, required: true, default: 50},
      2: { type: Number, required: true, default: 100},
      3: { type: Number, required: true, default: 200},
      4: { type: Number, required: true, default: 500},
      5: { type: Number, required: true, default: 1000},
    }
  ]
});

const Achievements = mongoose.model('Achievements', AchievementsShema);

module.exports = Achievements;
```

Rysunek 5.13 Model Achievements.js (źródło: opracowanie własne)

```
d > models > JS DailyQuestions.js > ...
const mongoose = require('mongoose');

const DailyQuestionsSchema = new mongoose.Schema({
  question: { type: String, required: true, unique: true },
  correct_answer: { type: String, required: true, unique: false},
  incorrect_answer_1: { type: String, required: true, unique: false},
  incorrect_answer_2: { type: String, required: true, unique: false},
  incorrect_answer_3: { type: String, required: true, unique: false},
  createdAt: { type: Date, default: Date.now }
});

const DailyQuestions = mongoose.model('DailyQuestions', DailyQuestionsSchema);

module.exports = DailyQuestions;
```

Rysunek 5.14 Model DailyQuestions.js (źródło: opracowanie własne)

```
> models > JS DailyQuiz.js > ...
const mongoose = require('mongoose');

const DailyQuizSchema = new mongoose.Schema({
  date: { type: Date, required: true, unique: true },
  questions: [
    {
      id_question: { type: Number, required: true},
      question: { type: String, required: true },
      correct_answer: { type: String, required: true },
      incorrect_answer_1: { type: String, required: true },
      incorrect_answer_2: { type: String, required: true },
      incorrect_answer_3: { type: String, required: true }
    }
  ]
});

const DailyQuiz = mongoose.model('DailyQuiz', DailyQuizSchema);

module.exports = DailyQuiz;
```

Rysunek 5.15 Model DailyQuiz.js (źródło: opracowanie własne)

```
> models > JS DailyStatistics.js > ...
const mongoose = require('mongoose');

const dailyStatisticsSchema = new mongoose.Schema({
  date: { type: Date, required: true, unique: true },
  totalUsers: { type: Number, required: true, default: 0 },
  completedUsers: { type: Number, required: true, default: 0 },
  completionRate: { type: Number, required: true, default: 0 },
  avgEloScores: {
    english: { type: Number, required: true, default: 1000 },
    maths: { type: Number, required: true, default: 1000 },
    science: { type: Number, required: true, default: 1000 },
    programming: { type: Number, required: true, default: 1000 }
  },
  totalCorrectAnswers: {
    english: { type: Number, required: true, default: 0 },
    maths: { type: Number, required: true, default: 0 },
    science: { type: Number, required: true, default: 0 },
    programming: { type: Number, required: true, default: 0 }
  },
  avgLevel: { type: Number, required: true, default: 1 },
  avgCoins: { type: Number, required: true, default: 0 }
}, { timestamps: true });

const DailyStatistics = mongoose.model('DailyStatistics', dailyStatisticsSchema);

module.exports = DailyStatistics;
```

Rysunek 5.16 Model DailyStatistics.js (źródło: opracowanie własne)

```
> models > JS EnglishQuestions.js > ...
const mongoose = require('mongoose');

const EnglishQuestionSchema = new mongoose.Schema({
  level: { type: Number, required: true},
  questions: [
    {
      date: { type: Date, required: true },
      id_question: { type: Number, required: true},
      question: { type: String, required: true },
      correct_answer: { type: String, required: true },
      incorrect_answer_1: { type: String, required: true },
      incorrect_answer_2: { type: String, required: true },
      incorrect_answer_3: { type: String, required: true }
    }
  ]
});

const EnglishQuestion = mongoose.model('EnglishQuestion', EnglishQuestionSchema);

module.exports = EnglishQuestion;
```

Rysunek 5.17 Model EnglishQuestions.js (źródło: opracowanie własne)

```
> models > JS ItemShop.js > ...
const mongoose = require('mongoose');

const ItemShopSchema = new mongoose.Schema({
    id: { type: Number, required: true, unique: true },
    name: { type: String, required: true, unique: true },
    type: { type: String, required: true, unique: false },
    url: { type: String, required: true, unique: true },
    requiredLevel: { type: Number, required: true, unique: false },
    cost: { type: Number, required: true, unique: false },
});

const ItemShop = mongoose.model('ItemShop', ItemShopSchema);

module.exports = ItemShop;
```

Rysunek 5.18 Model ItemShop.js (źródło: opracowanie własne)

```
d > models > JS MathsQuestions.js > ...
const mongoose = require('mongoose');

const MathsQuestionSchema = new mongoose.Schema({
  level: { type: Number, required: true},
  questions: [
    {
      date: { type: Date, required: true },
      id_question: { type: Number, required: true},
      question: { type: String, required: true },
      correct_answer: { type: String, required: true },
      incorrect_answer_1: { type: String, required: true },
      incorrect_answer_2: { type: String, required: true },
      incorrect_answer_3: { type: String, required: true }
    }
  ]
});

const MathsQuestion = mongoose.model('MathsQuestion', MathsQuestionSchema);

module.exports = MathsQuestion;
```

Rysunek 5.19 Model MathsQuestions.js (źródło: opracowanie własne)

```
d > models > JS ProgramingQuestions.js > ...
const mongoose = require('mongoose');

const ProgramingQuestionSchema = new mongoose.Schema({
  level: { type: Number, required: true},
  questions: [
    {
      date: { type: Date, required: true },
      id_question: { type: Number, required: true},
      question: { type: String, required: true },
      correct_answer: { type: String, required: true },
      incorrect_answer_1: { type: String, required: true },
      incorrect_answer_2: { type: String, required: true },
      incorrect_answer_3: { type: String, required: true }
    }
  ]
});

const ProgramingQuestion = mongoose.model('ProgramingQuestion', ProgramingQuestionSchema);

module.exports = ProgramingQuestion;
```

Rysunek 5.20 Model ProgramingQuestions.js (źródło: opracowanie własne)

```
> models > JS ScienceQuestions.js > ...
const mongoose = require('mongoose');

const ScienceQuestionSchema = new mongoose.Schema({
  level: { type: Number, required: true},
  questions: [
    {
      date: { type: Date, required: true },
      id_question: { type: Number, required: true},
      question: { type: String, required: true },
      correct_answer: { type: String, required: true },
      incorrect_answer_1: { type: String, required: true },
      incorrect_answer_2: { type: String, required: true },
      incorrect_answer_3: { type: String, required: true }
    }
  ]
});

const ScienceQuestion = mongoose.model('ScienceQuestion', ScienceQuestionSchema);

module.exports = ScienceQuestion;
```

Rysunek 5.21 Model ScienceQuestions.js (źródło: opracowanie własne)

```
> models > JS Tasks.js > ...
const mongoose = require('mongoose');

const tasksSchema = new mongoose.Schema({
    level: {type: Number, required: true},
    id_task: { type: Number, required: true, unique: true },
    title: { type: String, required: true, unique: true  },
    task: { type: String, required: true },
    possible_progress: { type: Number, required: true},
    date: { type: Date, required: true },
});

const Tasks = mongoose.model('Tasks', tasksSchema);

module.exports = Tasks;
```

Rysunek 5.22 Model Tasks.js (źródło: opracowanie własne)

```
> models > JS User.js > ...
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: false },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  type: {type: String, required: false},
  activationKey: { type: String, required: true, unique: false },
  userStatus: { type: String, required: true, unique: false },
  head: { type: String, required: false, unique: false },
  body: { type: String, required: false, unique: false },
  legs: { type: String, required: false, unique: false },
  name: { type: String, required: false, unique: false },
  createdAt: { type: Date, default: Date.now },
  lastLogin: { type: Date },
  answeredQuestions: { type: [Number], default: [] },
  completedAllQuestionsToday: {type: Boolean,  default: false },
  lastCompletedDate: { type: Date, default: null },
  englishQuestionElo: { type: Number, default: 1000, required: true},
  mathsQuestionElo: { type: Number, default: 1000, required: true},
  scienceQuestionElo: { type: Number, default: 1000, required: true},
  programmingQuestionElo: { type: Number, default: 1000, required: true},
  correct_programmingQuestion: { type: Number, default: 0, required: true},
  correct_mathsQuestion: { type: Number, default: 0, required: true},
  correct_englishQuestion: { type: Number, default: 0, required: true},
  correct_scienceQuestion: { type: Number, default: 0, required: true},
  level: {type: Number, required: true, unique: false, default: 1},
  coins: {type: Number, required: true, unique: false, default: 0},
  extracoins: {type: Number, required: true, unique: false, default: 0},
  happines: {type: Number, required: true, unique: false, default: 3},
  exp: {type: Number, required: true, unique: false, default: 0},
  hp: {type: Number, required: true, unique: false, default: 5, min: 0},
  hpLimit: {type: Number, required: true, unique: false, default: 5, min: 0},
  lastHpLoss: { type: Date, default: null },
  items: { type: [Number], default: [] },
  headSkin: {type: String, required: false, unique: false },
  legSkin: {type: String, required: false, unique: false },
  armSkin: {type: String, required: false, unique: false },
  extracoinsgiven: {type: [Number], default: []},
  editableCharacter: {type: Boolean,  default: false },
  tasks_progress: [
    {
      id_1: {type: Number, required: true, unique: false, default: 0, max: 4},
      id_2: {type: Number, required: true, unique: false, default: 0, max: 1},
      id_3: {type: Number, required: true, unique: false, default: 0, max: 1},
      id_4: {type: Number, required: true, unique: false, default: 0, max: 1},
      id_5: {type: Number, required: true, unique: false, default: 0, max: 1},
      id_6: {type: Number, required: true, unique: false, default: 0, max: 8},
      id_7: {type: Number, required: true, unique: false, default: 0, max: 2},
      id_8: {type: Number, required: true, unique: false, default: 0, max: 1},
      id_9: {type: Number, required: true, unique: false, default: 0, max: 2},
      id_10: {type: Number, required: true, unique: false, default: 0, max: 2},
      id_11: {type: Number, required: true, unique: false, default: 0, max: 40},
      id_12: {type: Number, required: true, unique: false, default: 0, max: 3},
      id_13: {type: Number, required: true, unique: false, default: 0, max: 1},
      id_14: {type: Number, required: true, unique: false, default: 0, max: 3},
      id_15: {type: Number, required: true, unique: false, default: 0, max: 1},
      id_16: {type: Number, required: true, unique: false, default: 0, max: 80},
      id_17: {type: Number, required: true, unique: false, default: 0, max: 40},
      id_18: {type: Number, required: true, unique: false, default: 0, max: 3},
      id_19: {type: Number, required: true, unique: false, default: 0, max: 1000},
      id_20: {type: Number, required: true, unique: false, default: 0, max: 1},
      id_21: {type: Number, required: true, unique: false, default: 0, max: 400},
      id_22: {type: Number, required: true, unique: false, default: 0, max: 1},
      id_23: {type: Number, required: true, unique: false, default: 0, max: 20},
      id_24: {type: Number, required: true, unique: false, default: 0, max: 1},
      id_25: {type: Number, required: true, unique: false, default: 0, max: 4},
    }
  ]
});

const User = mongoose.model('User', userSchema);
```

Rysunek 5.23 Model User.js (źródło: opracowanie własne)

5.4.10 Organizacja tras (routes)

W architekturze aplikacji backendowej istotnym elementem są trasy (routes), które określają sposób, w jaki serwer przetwarza i obsługuje żądania HTTP od klientów, takich jak aplikacja frontendowa lub inne systemy integrujące się poprzez API. W analizowanej aplikacji backendowej wykorzystano framework Express.js, który dostarcza mechanizmy do definiowania tras oraz ich obsługi poprzez odpowiednie kontrolery.

Definicja i funkcja tras w aplikacji

Trasy pełnią kluczową funkcję w aplikacji, ponieważ:

- Obsługują żądania HTTP — określają, jakie operacje są wykonywane w odpowiedzi na zapytania użytkowników (np. pobieranie danych, dodawanie nowych wpisów).
- Mapują żądania na odpowiednie kontrolery — każde żądanie trafia do właściwego modułu, który przetwarza je zgodnie z logiką aplikacji.
- Zapewniają strukturę i modularność — organizacja tras w dedykowanym pliku pozwala na czytelne zarządzanie logiką aplikacji.
- Wspierają mechanizmy bezpieczeństwa — umożliwiają stosowanie warstw pośredniczących (middleware) do walidacji danych, uwierzytelniania oraz ochrony przed atakami.

Struktura tras w aplikacji

W aplikacji backendowej trasy zostały zdefiniowane w pliku routes/index.js, który grupuje poszczególne punkty końcowe API i przypisuje je do odpowiednich kontrolerów.

Importowanie wymaganych modułów

Na początku pliku importowane są niezbędne moduły:

```
d > routes > JS index.js > ...
const express = require('express');
const router = express.Router();
```

Rysunek 5.24 Import modułów w pliku routes/index.js (źródło: opracowanie własne)

express.Router() — obiekt routera służy do definiowania tras, co pozwala na modularność i lepszą organizację kodu.

Następnie importowane są kontrolery, które obsługują logikę aplikacji:

```
const RegisterController = require('../controllers/RegisterController')
const LoginController = require('../controllers/LoginController')
const AdminController = require('../controllers/AdminController')
const ActivateUser = require('../controllers/ActivateUserController')
const CharacterController = require('../controllers/CharacterController')
const GetCharacterController = require('../controllers/GetCharacterController')
const DailyQuestionsController = require('../controllers/DailyQuestionsController');
const MathQuestionsController = require('../controllers/MathQuestionsController');
const ItemShopController = require('../controllers/ItemShopController')
const WardrobeController = require('../controllers/WardrobeController')
const ScienceQuestionsController = require('../controllers/ScienceQuestlionsController');
const EnglishQuestionsController = require('../controllers/EnglishQuestlionsController');
const ProgramingQuestionsController = require('../controllers/ProgramingQuestlionsController');
const UserStatsController = require('../controllers/UserStatsController');
const TasksController = require('../controllers/TasksController')
const AchievementsController  = require('../controllers/AchievementsController')
const DailyStatsController = require('../controllers/DailyStatsController')
const ResetPasswordController = require('../controllers/ResetPasswordController')
const BuyHeartController = require('../controllers/HeartShopController')
```

Rysunek 5.25 Import kontrolerów routes/index.js (źródło: opracowanie własne)

Importowane są również middleware, które pełnią funkcje zabezpieczające i kontrolujące dostęp:

```
const authenticateUser = require('../middlewares/authenticateUser');
const { loginLimiter, detectInjection, blockIP, isBlocked } = require('../middlewares/attackDetection');
const checkEditableCharacter = require('../middlewares/checkEditableCharacter');
```

Rysunek 5.26 Import middleware routes/index.js (źródło: opracowanie własne)

authenticateUser — sprawdza, czy użytkownik jest zalogowany, korzystając z mechanizmu JWT.
loginLimiter — ogranicza liczbę nieudanych prób logowania, chroniąc przed atakami brute-force.
detectInjection — wykrywa potencjalne ataki SQL/NoSQL Injection

Definiowanie tras w aplikacji
Każda trasa składa się z:

- Metody HTTP — definiuje typ żądania (GET, POST, PUT, DELETE).
- Ścieżki URL — określa adres zasobu (/api/register, /api/daily-question).
- Middleware (opcjonalnie) — funkcje sprawdzające np. autoryzację użytkownika.
- Funkcji kontrolera — odpowiada za przetwarzanie żądania i zwrócenie odpowiedzi.

```
router.post('/api/register', RegisterController.Register)
router.post('/api/login', isBlocked, loginLimiter, detectInjection, LoginController.Login)
router.post('/api/activation', authenticateUser, ActivateUser.ActivateUser)
router.post('/api/saveCharacter', authenticateUser, checkEditableCharacter, CharacterController.Savecharacter)
router.get('/api/character', authenticateUser, GetCharacterController.getCharacter)
router.post('/CreateDailyQuestion', AdminController.CreateDailyQuestion)
router.post('/CreateMathsQuestion', AdminController.CreateMathsQuestion)
router.post('/CreateProgrammingQuestion', AdminController.CreateProgrammingQuestion);
router.post('/CreateScienceQuestion', AdminController.CreateScienceQuestion);
router.post('/CreateEnglishQuestion', AdminController.CreateEnglishQuestion);
router.get('/api/daily-question', authenticateUser, DailyQuestionsController.getDailyQuestion);
router.post('/api/submit-answer', authenticateUser, DailyQuestionsController.submitAnswer);
router.get('/api/math-questions/:level', authenticateUser, MathQuestionsController.getMathQuestion);
router.post('/api/submit-math-answer', authenticateUser, MathQuestionsController.submitMathAnswer);
router.get('/api/science-questions/:level', authenticateUser, ScienceQuestionsController.getScienceQuestion);
router.post('/api/submit-science-answer', authenticateUser, ScienceQuestionsController.submitScienceAnswer);
router.get('/api/english-questions/:level', authenticateUser, EnglishQuestionsController.getEnglishQuestion);
router.post('/api/submit-english-answer', authenticateUser, EnglishQuestionsController.submitEnglishAnswer);
router.get('/api/programing-questions/:level', authenticateUser, ProgramingQuestionsController.getProgramingQuestion);
router.post('/api/submit-programing-answer', authenticateUser, ProgramingQuestionsController.submitProgramingAnswer);
router.post('/CreateShopItem', AdminController.CreateShopItem);
router.get('/api/shop/items', authenticateUser, ItemShopController.getShopItems);
router.post('/api/shop/buy-item', authenticateUser, ItemShopController.BuyItem);
router.post('/api/buy-edit-character', authenticateUser, ItemShopController.BuyEditCharacter);
router.get('/api/wardrobe/items', authenticateUser, WardrobeController.getWardrobeItems);
router.post('/api/select-item', authenticateUser, WardrobeController.selectItem);
router.get('/api/userstats', authenticateUser, UserStatsController.getUserStats);
router.get('/api/get-tasks', authenticateUser, TasksController.getTasks);
router.get('/api/get-achievements', authenticateUser, AchievementsController.getAchievements);
router.post('/get-all-users', GetCharacterController.getAllUsers);
router.get('/get-user-data/:id', GetCharacterController.getAllUserData);
router.put('/update-user/:id', GetCharacterController.updateUserData);
router.get('/api/stats/:date', DailyStatsController.getDailyStats);
router.get('/api/stats/compare/:date1/:date2', DailyStatsController.compareDailyStats);
router.post('/api/send-reset-code', ResetPasswordController.sendResetCode);
router.post('/api/reset-password', ResetPasswordController.resetPassword);
router.post('/api/buy-heart', authenticateUser, BuyHeartController.BuyHeart);
router.post('/api/get-heart-shop', authenticateUser, BuyHeartController.GetHeartShop);
```

Rysunek 5.27 Trasy API routes/index.js (źródło: opracowanie własne)

5.4.11 Funkcja i implementacja folderu utils

W architekturze aplikacji backendowej folder utils pełni istotną rolę, gromadząc zestaw funkcji pomocniczych, które usprawniają przetwarzanie danych wejściowych oraz zwiększają bezpieczeństwo aplikacji. Głównym zadaniem modułów znajdujących się w tym folderze jest walidacja danych użytkownika, co minimalizuje ryzyko błędów i potencjalnych ataków na system.

Rola folderu utils w aplikacji

Folder utils zawiera funkcje, które są wykorzystywane w różnych częściach aplikacji w celu:

- Walidacji danych wejściowych — sprawdzanie poprawności kodów weryfikacyjnych, adresów e-mail, haseł oraz nazw użytkowników.
- Zapewnienia spójności danych — standaryzacja formatów danych wprowadzanych przez użytkowników.
- Zabezpieczenia przed atakami — ochrona przed SQL Injection, NoSQL Injection oraz innymi technikami manipulacji danymi.
- Dzięki wydzieleniu funkcji walidacyjnych do osobnego modułu możliwe jest ponowne użycie kodu, co ułatwia zarządzanie aplikacją oraz jej skalowalność.

Struktura folderu utils

Folder utils zawiera cztery moduły odpowiedzialne za walidację różnych typów danych użytkownika:

- codeValidation.js — walidacja kodów weryfikacyjnych.
- emailValidation.js — sprawdzanie poprawności adresów e-mail.
- passwordValidation.js — kontrola poprawności haseł użytkowników.
- usernameValidation.js — walidacja nazw użytkowników.

Walidacja kodów weryfikacyjnych jest przeprowadzana według następujących warunków:

- Składa się wyłącznie z sześciu cyfr.
- Nie zawiera znaków mogących wskazywać na próbę ataku SQL Injection lub NoSQL Injection.

Walidacja adresów e-mail jest przeprowadzana według następujących warunków:

- Sprawdzenie poprawności formatu adresu.
- Ochronę przed atakami SQL Injection i NoSQL Injection.
- Odrzucenie jednorazowych adresów e-mail z domen tymczasowych.

Walidacja haseł jest przeprowadzana według następujących warunków:

- Minimalną długość hasła (8 znaków).
- Maksymalną długość (64 znaki).
- Obecność co najmniej jednej cyfry.
- Ochronę przed SQL Injection i NoSQL Injection.

Walidacja nazw jest przeprowadzana według następujących warunków:

- Długość od 3 do 20 znaków.
- Dozwolone znaki: litery, cyfry, _, -.
- Brak znaków mogących sugerować atak SQL Injection lub NoSQL Injection.
- Brak nazw systemowych, takich jak „admin” czy „root”.

5.4.12 Plik głównego - app.js

Plik główny aplikacji backendowej pełni fundamentalną rolę w inicjalizacji i konfiguracji całego systemu. Jego zadaniem jest inicjalizacja serwera, konfiguracja obsługi żądań HTTP, zabezpieczenie aplikacji oraz integracja z bazą danych. W aplikacji backendowej wykorzystano Express.js, który pełni funkcję głównego frameworka serwera. Dodatkowo, zaimplementowano szereg mechanizmów zwiększających bezpieczeństwo i optymalizujących działanie systemu.

Struktura pliku i inicjalizacja aplikacji

Plik rozpoczyna się od importu niezbędnych modułów:

```
> JS app.js > ...
const express = require('express');
const https = require('https');
const fs = require('fs');
const path = require('path');
const bodyParser = require('body-parser');
const cookieParser = require('cookie-parser');
const flash = require('connect-flash');
const session = require('express-session');
const routes = require('./routes/index');
const mongoose = require('mongoose');
const MongoStore = require('connect-mongo');
const nodemailer = require('nodemailer');
const connectDB = require('./config/database');
const moment = require('moment-timezone');
const helmet = require('helmet');
require('dotenv').config();
```

Rysunek 5.28 Import modułów w pliku app.js (źródło: opracowanie własne)

Następnie tworzona jest instancja aplikacji:

```
const app = express();
connectDB();
```

Rysunek 5.29 Tworzenie instancji aplikacji w pliku app.js (źródło: opracowanie własne)

Konfiguracja zabezpieczeń aplikacji:
W celu zwiększenia bezpieczeństwa aplikacji zastosowano bibliotekę Helmet, która dodaje dodatkowe nagłówki HTTP zabezpieczające przed popularnymi atakami.

```
app.use(helmet.hsts({
    maxAge: 31536000,
    includeSubDomains: true,
    preload: true,
  }));
```

Rysunek 5.30 Konfiguracja zabezpieczeń w pliku app.js (źródło: opracowanie własne)

Obsługa sesji użytkowników:
System sesji jest zarządzany przez express-session i przechowywany w bazie MongoDB

```
app.use(session({
  secret: process.env.SESSION_SECRET || 'squlio',
  resave: false,
  saveUninitialized: false,
  store: MongoStore.create({
      client: mongoose.connection.getClient(),
      collectionName: 'sessions',
      ttl: 14 * 24 * 60 * 60,
  }),
  cookie: {
      secure: true,
      httpOnly: true,
      maxAge: 24 * 60 * 60 * 1000
  }
}));
```

Rysunek 5.31 Obsługa sesji użytkowników w pliku app.js (źródło: opracowanie własne)

- *store: MongoStore.create({...})* — przechowuje sesje w MongoDB, co zwiększa trwałość danych.
- *secure: true* — wymusza korzystanie z HTTPS dla ciasteczek sesyjnych.
- *httpOnly: true* — zapobiega dostępowi do sesji przez JavaScript, co chroni przed atakami XSS.

Definiowanie tras i obsługa frontendowej aplikacji
Po zakończeniu konfiguracji aplikacja implementuje obsługę tras API oraz integrację z interfejsem użytkownika opartym na React

```
app.use('/', routes);
```

Rysunek 5.32 Konfiguracja tras API w pliku app.js (źródło: opracowanie własne)

Następnie udostępniane są statyczne pliki aplikacji frontendowej:

```
app.use(express.static(path.join(__dirname, '../frontend/dist')));

app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, '../frontend/dist', 'index.html'));
});
```

Rysunek 5.33 Konfiguracja statycznych plików aplikacji frontendowej w pliku app.js (źródło: opracowanie własne)

5.4.13 Konfiguracja i uruchomienie serwera HTTPS w aplikacji backendowej (server.js)

Plik ten pełni kluczową rolę w uruchamianiu serwera HTTPS, który obsługuje żądania przychodzące do aplikacji backendowej. W przeciwieństwie do standardowego serwera HTTP, implementacja HTTPS zapewnia szyfrowaną komunikację między klientem a serwerem, co znacząco podnosi poziom bezpieczeństwa aplikacji.

Struktura i działanie pliku
Plik ten składa się z trzech głównych sekcji:

- Importowanie niezbędnych modułów
- Konfiguracja certyfikatów SSL
- Inicjalizacja serwera HTTPS i jego uruchomienie

Importowanie modułów:

- *https* — moduł Node.js umożliwiający obsługę protokołu HTTPS.
- *fs* — moduł systemu plików (File System), który jest wykorzystywany do wczytania certyfikatów SSL.
- *app* — import głównej instancji aplikacji backendowej z pliku app.js.

```
const https = require('https');
const fs = require('fs');
const app = require('./app');
```

Rysunek 5.34 Importowanie modułów w pliku server.js (źródło: opracowanie własne)

Konfiguracja certyfikatów SSL:

- *key* — klucz prywatny serwera (squlio.key), który jest wykorzystywany do nawiązywania szyfrowanego połączenia.
- *cert* — certyfikat SSL (squlio.crt), który potwierdza tożsamość serwera i pozwala na bezpieczną wymianę danych.

```
const options = {
  key: fs.readFileSync('./ssl_keys/squlio.key'),
  cert: fs.readFileSync('./ssl_keys/squlio.crt'),
};
```

Rysunek 5.35 Konfiguracja certyfikatów SSL w pliku server.js (źródło: opracowanie własne)

Tworzenie i uruchamianie serwera HTTPS:

- *https.createServer(options, app)* — tworzy serwer HTTPS, który wykorzystuje wcześniej załadowane certyfikaty SSL (options) oraz obsługuje żądania przekazywane do aplikacji (app).
- *listen(app.get('port'))* — uruchamia serwer na porcie zadeklarowanym w aplikacji (app.get('port')), który domyślnie wynosi 443 — standardowy port dla HTTPS.

```
https.createServer(options, app).listen(app.get('port'), () => {
  console.log(`Serwer HTTPS działa na porcie ${app.get('port')}`);
});
```

Rysunek 5.36 Tworzenie i uruchamianie serwera HTTPS w pliku server.js (źródło: opracowanie własne)

```
0|server | Serwer HTTPS działa na porcie 443
```

Rysunek 5.37 Informacja zwrotna wyświetlana po uruchomieniu serwera (źródło: opracowanie własne)

Znaczenie implementacji serwera HTTPS

Zastosowanie HTTPS w aplikacji backendowej ma kluczowe znaczenie dla bezpieczeństwa i integralności danych przesyłanych między klientem a serwerem. Dzięki szyfrowaniu ruchu:

- Chronione są dane użytkowników — wszelkie loginy, hasła oraz wrażliwe informacje są przesyłane w sposób zaszyfrowany.
- Zapobiega się atakom MITM (Man-in-the-Middle) — uniemożliwia przechwytywanie i manipulowanie danymi przez osoby trzecie.
- Spełniane są wymagania przeglądarek i standardów bezpieczeństwa — wiele przeglądarek blokuje niezabezpieczone połączenia HTTP, wymagając wdrożenia HTTPS.
- Podnosi się wiarygodność aplikacji — certyfikaty SSL potwierdzają autentyczność serwera, co zwiększa zaufanie użytkowników.

5.5 Baza danych MongoDB

W aplikacji backendowej wykorzystano MongoDB jako główny system zarządzania bazą danych. MongoDB to nierelacyjna baza danych typu NoSQL, która operuje na dynamicznych dokumentach w formacie BSON (Binary JSON), umożliwiając elastyczne przechowywanie oraz szybkie przetwarzanie danych. Dzięki swojej wydajności, skalowalności oraz odporności na zmiany schematu danych, MongoDB jest szeroko stosowane w nowoczesnych aplikacjach webowych i mobilnych.

5.5.1 Struktura kolekcji i sposób przechowywania danych

Każda kolekcja w MongoDB przechowuje dokumenty w formacie BSON, które są odpowiednikiem rekordów w relacyjnych bazach danych. Przykładowe dokumenty mogą wyglądać następująco:

Kolekcja users:


```
_id: ObjectId('67bf9ac749ddf0cd7c931ba1')
username : "Dejvid"
email : "dawid.rynkiewicz1@wp.pl"
password : "$2b$10$QF/c2X.1wBP8J9xj8.djR..8EgjH7DHlrxOdQYpN.C.9GPH/j954S"
type : "gracz"
activationKey : "997258"
userStatus : "enabled"
head : "/head/happy/h1_orange.png"
▸ answeredQuestions : Array (empty)
completedAllQuestionsToday : false
lastCompletedDate : 2025-03-03T23:00:00.000+00:00
englishQuestionElo : 4100
mathsQuestionElo : 9400
scienceQuestionElo : 4600
programmingQuestionElo : 15500
correct_programmingQuestion : 162
correct_mathsQuestion : 101
correct_englishQuestion : 56
correct_scienceQuestion : 54
level : 13
coins : 330
extracoins : 7
happines : 3
exp : 21440
hp : 20
hpLimit : 20
lastHpLoss : 2025-03-03T06:26:00.269+00:00
▸ items : Array (24)
createdAt : 2025-02-26T22:50:47.154+00:00
▸ tasks_progress : Array (1)
__v : 33
body : "/body/b2_blue.png"
legs : "/legs/l2_orange.png"
name : "Dejvid"
headSkin : "17"
armSkin : "13"
▸ extracoinsgiven : Array (4)
lastLogin : 2025-03-03T23:20:20.808+00:00
legSkin : "25"
editableCharacter : false
▲ Hide 15 fields
```


Rysunek 5.38 Przykładowy użytkownik w kolekcji users (źródło: opracowanie własne)

Kolekcja dailystatistics:


```
_id: ObjectId('67a9682b63438eda661d5c9d')
date : 2025-02-09T23:00:00.000+00:00
__v : 0
avgCoins : 7910.4
avgEloScores : Object
  english : 980
  maths : 1020
  science : 1000
  programming : 2560
avgLevel : 4.2
completedUsers : 0
completionRate : 0
createdAt : 2025-02-10T02:44:59.069+00:00
totalCorrectAnswers : Object
  programming : 75
  maths : 0
  english : 0
  science : 0
totalUsers : 5
updatedAt : 2025-02-10T22:55:00.489+00:00
```

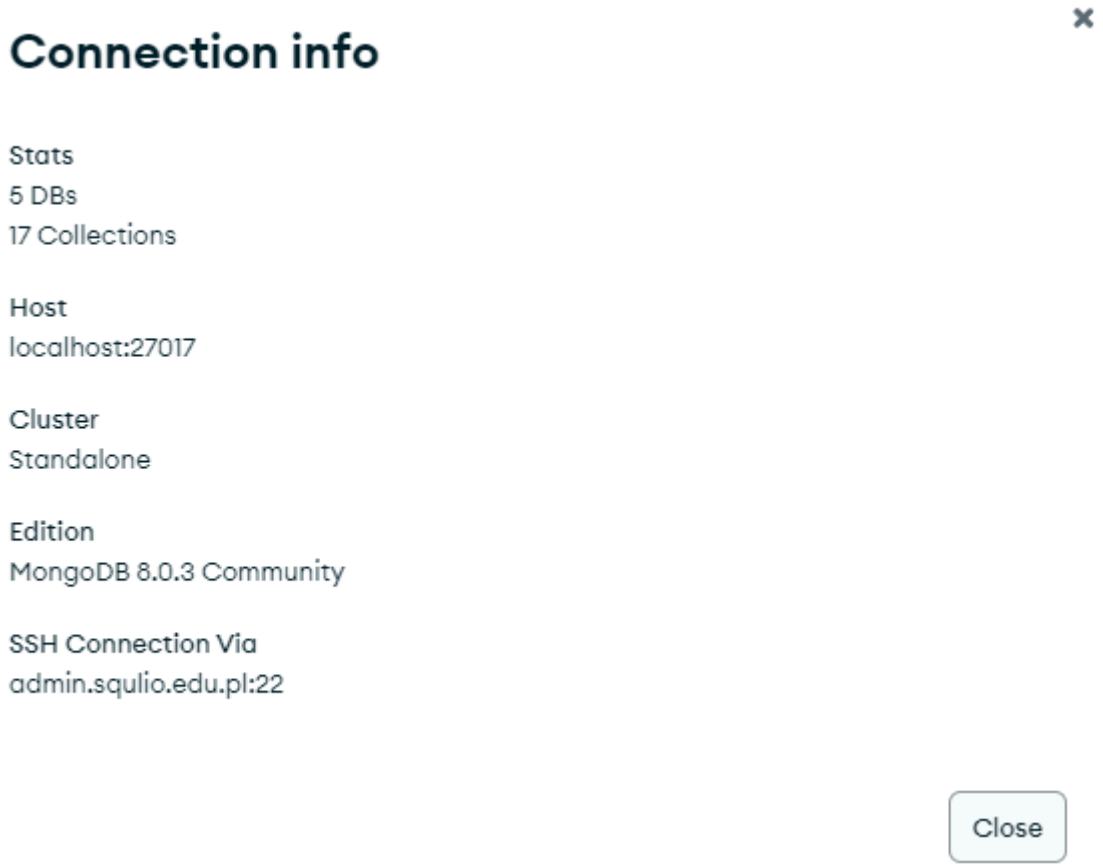
```
_id: ObjectId('67aa84f063438eda661d5c9e')
date : 2025-02-10T23:00:00.000+00:00
__v : 0
avgCoins : 7910.4
avgEloScores : Object
avgLevel : 4.2
completedUsers : 0
completionRate : 0
createdAt : 2025-02-10T23:00:00.770+00:00
totalCorrectAnswers : Object
totalUsers : 5
updatedAt : 2025-02-11T22:55:00.262+00:00
```

```
_id: ObjectId('67abd67063438eda661d5c9f')
date : 2025-02-11T23:00:00.000+00:00
__v : 0
avgCoins : 7910.4
avgEloScores : Object
avgLevel : 4.2
```


Rysunek 5.39 Wygląd kolekcji dailystatistics (źródło: opracowanie własne)

5.5.2 Konfiguracja

MongoDB w aplikacji backendowej działa w środowisku lokalnym (localhost:27017) i funkcjonuje jako instancja standalone, co oznacza, że nie jest częścią klastra replikacyjnego ani rozproszonego systemu shardingowego. Jest to typowa konfiguracja stosowana w środowiskach deweloperskich oraz mniejszych aplikacjach, gdzie nie ma potrzeby zapewnienia wysokiej dostępności poprzez mechanizmy redundancji.



Rysunek 5.40 Informacje o połączeniu bazy danych (źródło: opracowanie własne)

5.2.3 Korzyści wynikające z wykorzystania MongoDB

Wybór MongoDB jako bazy danych dla aplikacji wynika z jej licznych zalet:

- Elastyczna struktura danych — brak sztywnego schematu umożliwia łatwą rozbudowę aplikacji.
- Wysoka wydajność — indeksowanie oraz optymalizacja zapytań zapewniają szybki dostęp do danych.
- Łatwa skalowalność — MongoDB wspiera replikację oraz sharding, co pozwala na rozproszone przechowywanie danych.
- Wsparcie dla danych złożonych — możliwość przechowywania tablic i zagnieżdżonych obiektów bez potrzeby normalizacji.
- Bezpośrednia integracja z aplikacją Node.js — biblioteka Mongoose upraszcza interakcję z bazą.

5.6 Struktura Serwera

Aplikacja została uruchomiona w kontenerze na serwerze lokalnym, działającym w środowisku Proxmox. Kontener ten zawiera wszystkie niezbędne komponenty aplikacji, w tym silnik Node.js, framework Express.js oraz bazę danych MongoDB. Takie podejście zapewnia wysoką izolację i ułatwia zarządzanie zasobami oraz bezpieczeństwem systemu.

Dostęp do serwera został zabezpieczony wielowarstwowo. Bezpośrednia komunikacja z serwerem możliwa jest jedynie poprzez połączenie VPN które przydzielane jest każdorazowo przez administratora systemu, co znacząco ogranicza potencjalne

sposoby ataku. Tylko osoby posiadające dostęp do VPN mogą łączyć się z serwerem, co eliminuje ryzyko nieautoryzowanego dostępu z sieci publicznej. Dodatkowo, na poziomie routera zostało skonfigurowane przekierowanie portów, umożliwiający dostęp do aplikacji jedynie przez protokół HTTPS na porcie 443. Oznacza to, że użytkownicy mogą jedynie korzystać z interfejsu aplikacji, nie mając dostępu do jej wewnętrznych zasobów.

Bezpieczeństwo systemu zostało wzmocnione również poprzez odpowiednie zabezpieczenia samego serwera. Zdalne logowanie na konto root zostało całkowicie wyłączone, co eliminuje możliwość przejęcia pełnej kontroli nad serwerem w przypadku potencjalnego ataku. Dodatkowo, aktywne zapory sieciowe (firewalle) funkcjonują na trzech poziomach: na serwerze, routerze oraz w obrębie połączenia VPN, co stanowi solidną barierę przed nieautoryzowanym ruchem sieciowym.

Dzięki wykorzystaniu narzędzia PM2 aplikacja działa w sposób ciągły i jest automatycznie restartowana w przypadku wystąpienia błędów, co minimalizuje ryzyko długotrwałego przestoju systemu. PM2 pozwala również na monitorowanie zużycia zasobów przez aplikację, co umożliwia optymalizację jej działania i szybkie reagowanie na ewentualne problemy wydajnościowe.

Kopia zapasowa całego kontenera jest wykonywana automatycznie dwa razy dziennie — o godzinie 7:30 oraz 22:30 — za pomocą mechanizmów Proxmox. Takie rozwiązanie zapewnia ochronę przed utratą danych i pozwala na szybkie przywrócenie systemu do pełnej funkcjonalności w przypadku awarii.

Dostęp do bazy danych MongoDB również odbywa się wyłącznie przez połączenie VPN, co zapewnia jej dodatkową warstwę ochrony. Administratorzy mogą zarządzać danymi za pomocą narzędzia MongoDB Compass, jednak tylko po wcześniejszym uzyskaniu dostępu przez VPN. Dzięki temu baza danych jest skutecznie zabezpieczona przed próbami nieautoryzowanego dostępu z zewnątrz.

Takie podejście do architektury systemu zapewnia wysoką izolację, bezpieczeństwo i odporność na ataki. Dzięki wielowarstwowym zabezpieczeniom oraz redundancji danych aplikacja jest odporna na awarie i potencjalne zagrożenia, co gwarantuje jej stabilne działanie oraz ochronę przechowywanych informacji.

5.7 Instrukcja użytkowania

Instrukcja użytkowania platformy SQULIO dla użytkownika końcowego obejmuje opis sposobu rejestracji, logowania, personalizacji konta, korzystania z narzędzi edukacyjnych oraz sklepu i garderoby. Krok po kroku zostanie przedstawione tutaj jak w pełni wykorzystać potencjał aplikacji.

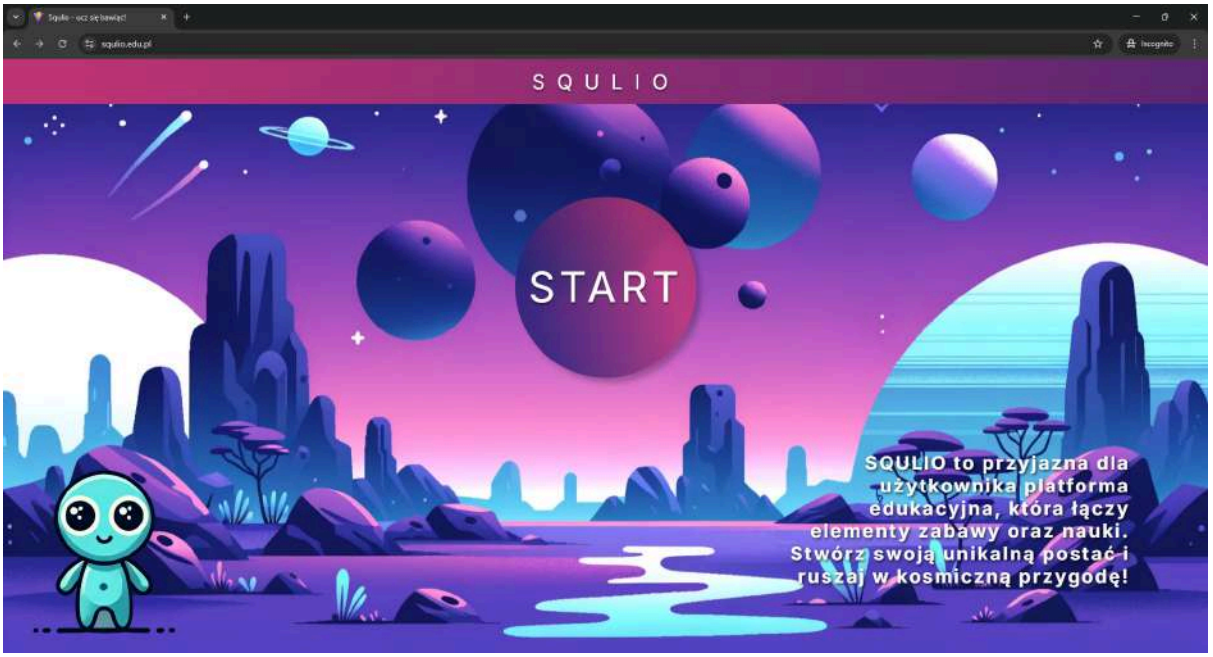
Podstawowe informacje potrzebne do uruchomienia strony.

Aby zacząć korzystać z platformy, należy upewnić się, że urządzenie jest połączone z internetem oraz posiada zainstalowaną aktualną wersję przeglądarki internetowej obsługującej nowoczesne technologie webowe, taką jak: Google Chrome, Mozilla Firefox, Microsoft Edge czy Safari. Po jej uruchomieniu należy przejść pod adres:

https://squlio.edu.pl/

Strona startowa

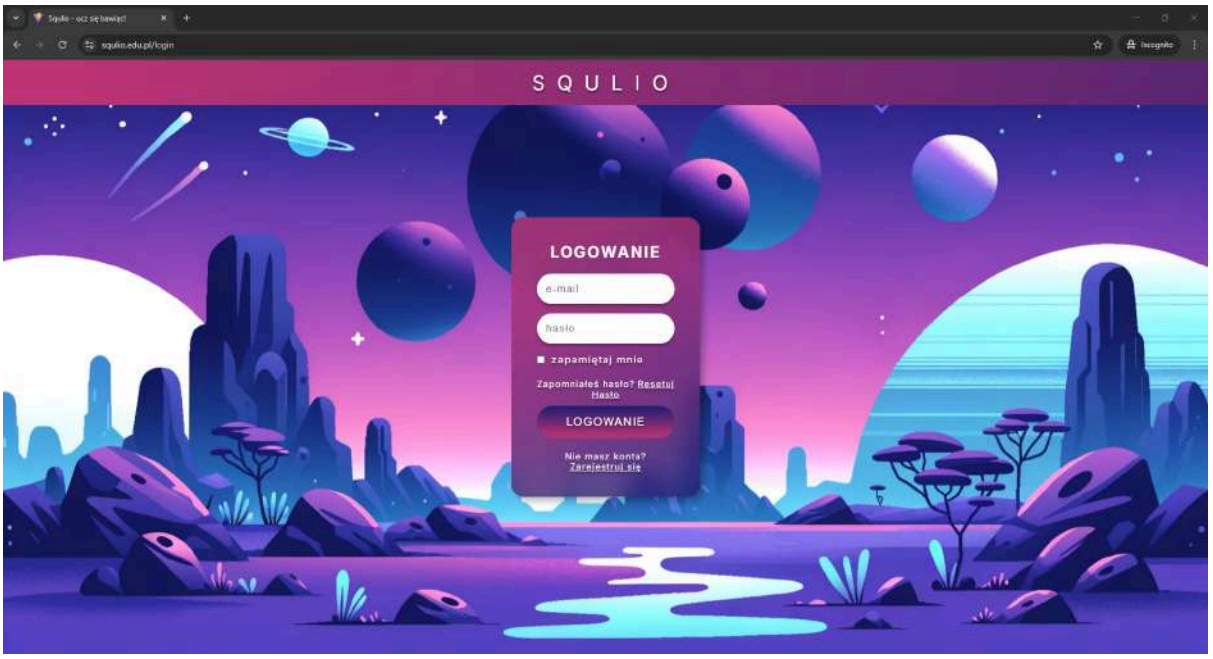
Wyświetlona strona pozwala na zapoznanie się z krótką informacją na temat projektu oraz posiada przycisk “START”, który przeniesie użytkownika do strony logowania.



Rysunek 5.41 Strona startowa platformy SQULIO (źródło: opracowanie własne)

Strona logowania

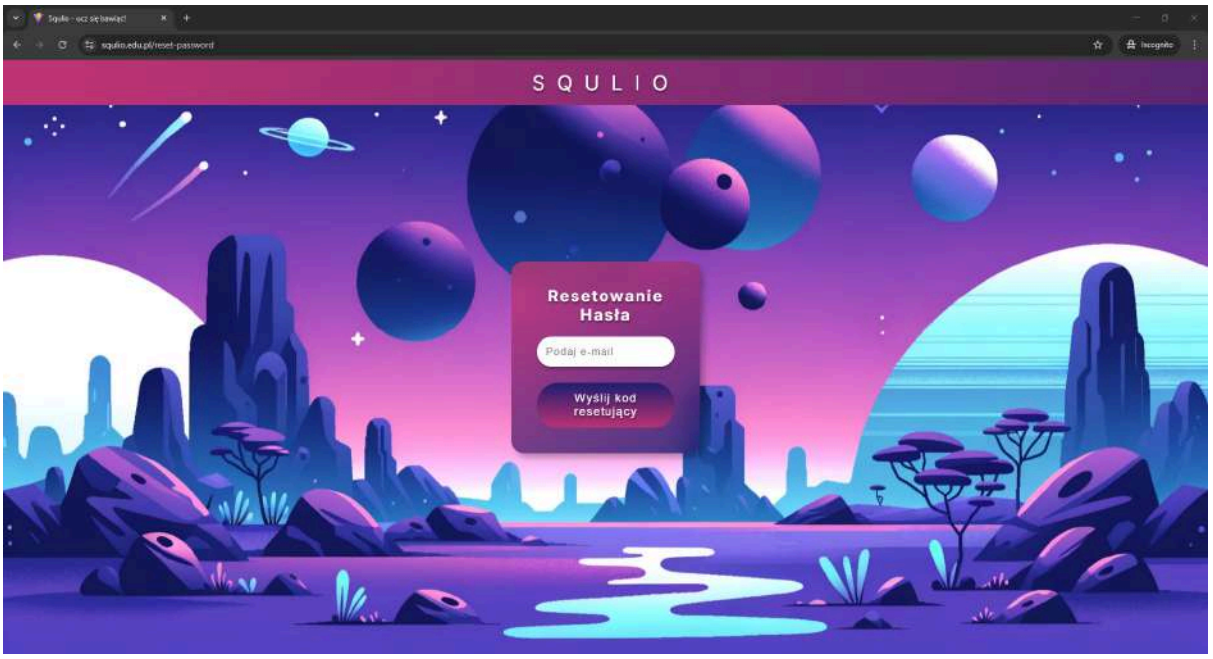
Ekran logowania umożliwia zalogowanie się wcześniej już zarejestrowanym użytkownikom. Posiada pola: "e-mail" i "hasło", w których należy podać dane do logowania oraz przycisk wyboru "zapamiętaj mnie", dzięki któremu użytkownik nie będzie musiał logować się po ponownym połączeniu z tej samej przeglądarki. Poniżej znajduje się przycisk resetowania hasła, pozwalający na zresetowanie danych logowania w przypadku ich zapomnienia oraz duży przycisk "LOGOWANIE" umożliwiający przejście do platformy edukacyjnej po wpisaniu odpowiednich danych. Jeśli na tym etapie użytkownik nie posiada konta, powinien skorzystać z opcji "Zarejestruj się" znajdującej się w dolnej części panelu logowania.



Rysunek 5.42 Strona logowania (źródło: opracowanie własne)

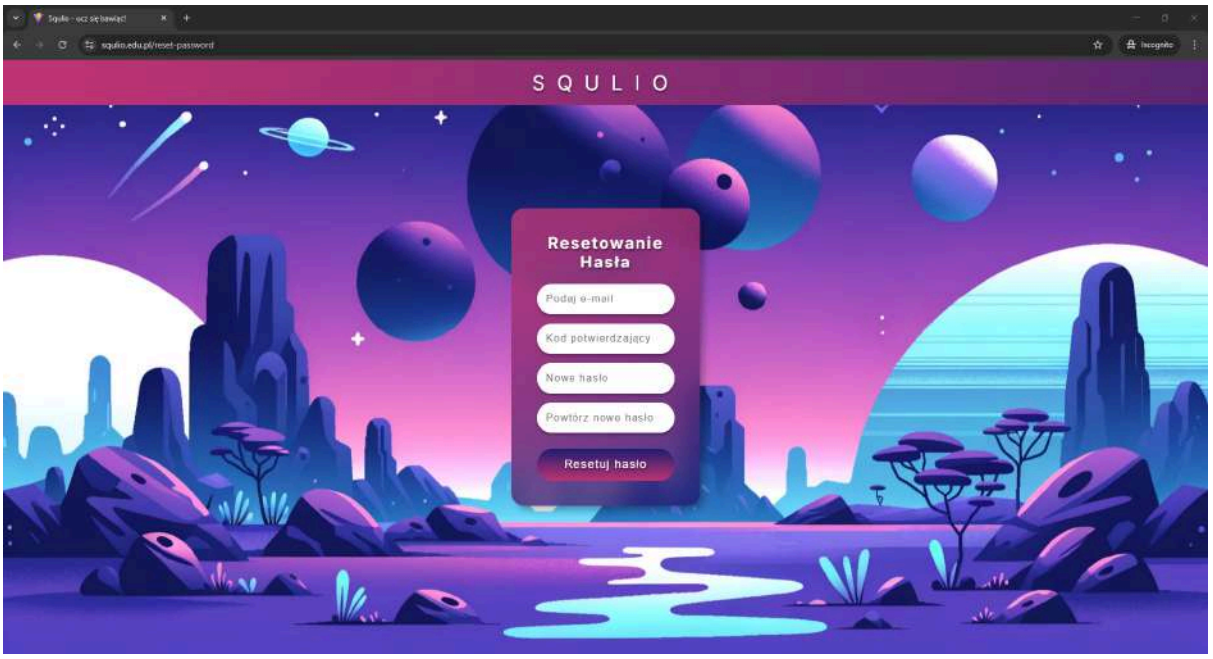
Strona resetowania hasła

Jeśli podczas logowania użytkownik zapomni hasła, możliwe jest jego zresetowanie. W tym celu na stronie resetowania hasła należy w panelu na środku uzupełnić adres e-mail przypisany do konta i użyć przycisku "Wyślij kod resetujący".



Rysunek 5.43 Strona resetowania hasła część I (źródło: opracowanie własne)

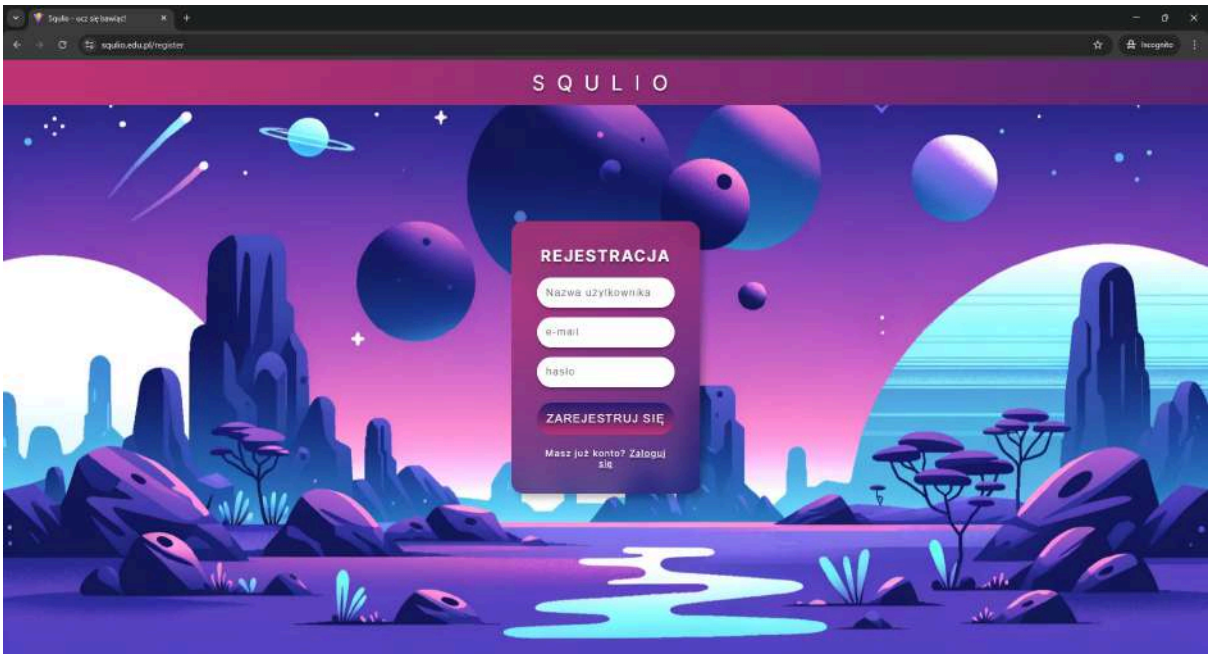
Po wykonaniu tej czynności kod potrzebny do resetu hasła zostanie wysłany na podany adres e-mail, a użytkownik zostanie przekierowany do kolejnego etapu, w którym oprócz adresu e-mail przypisanego do konta i kodu potwierdzającego musi podać dwa razy nowe hasło, które chce używać do logowania. Następnie należy kliknąć "Resetuj hasło".



Rysunek 5.44 Strona resetowania hasła część II (źródło: opracowanie własne)

Strona rejestracji

Ekran rejestracji jest analogiczną do logowania stroną pozwalającą na stworzenie nowego konta użytkownikowi, który jeszcze takowego nie posiada. Zawiera pola "Nazwa użytkownika", "e-mail" i "hasło", które są niezbędne do utworzenia nowego konta. Należy w tym miejscu je podać oraz kliknąć przycisk "ZAREJESTRUJ SIĘ", który przekieruje użytkownika do platformy. W dolnej części panelu znajduje się opcja "Zaloguj się" umożliwiająca powrót do panelu logowania.



Rysunek 5.45 Strona rejestracji (źródło: opracowanie własne)

Strona aktywacji

Jeśli użytkownik założył nowe konto korzystając z narzędzia rejestracji, zostanie przekierowany jeszcze na ekran aktywacji konta, na którym musi podać kod aktywacyjny wysłany na maila podanego podczas tworzenia konta. Po jego wpisaniu należy kliknąć przycisk "AKTYWUJ", aby przejść do platformy.

Rysunek 5.46 Strona aktywacji (źródło: opracowanie własne)

Tworzenie postaci

Po utworzeniu nowego konta i jego aktywacji należy stworzyć swoją postać i nadać jej nazwę. Należy wykorzystać do tego pasek nazwy z napisem "Ustaw nazwę", gdzie ustala się nazwę postaci oraz strzałki znajdujące się w środkowej części ekranu, aby dostosować początkowy wygląd postaci. Po wykonaniu tych kroków należy zapisać postać korzystając z przycisku w dolnej części ekranu.

Rysunek 5.47 Strona tworzenia postaci (źródło: opracowanie własne)

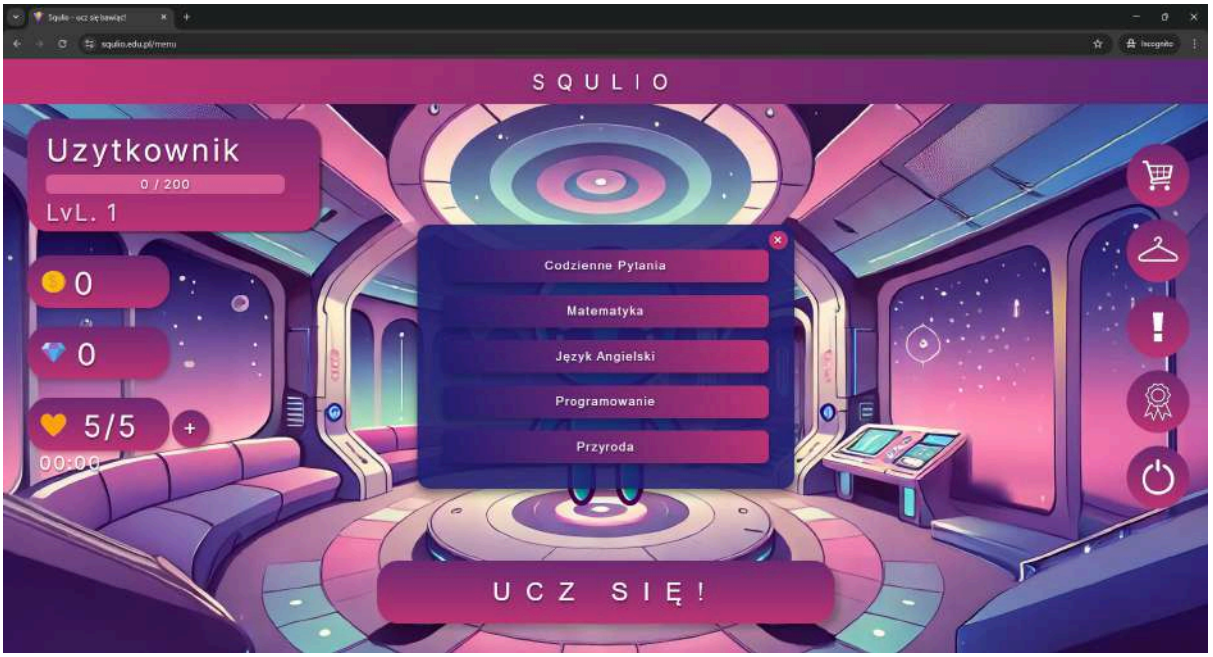
Menu główne

Menu główne jest centrum dowodzenia platformy SQULIO. Pozwala na wykonywanie zadań, przejście do innych elementów aplikacji, takich jak sklep czy garderoba oraz na sprawdzenie aktualnych statystyk. W środkowej części ekranu znajduje się postać użytkownika, pod nią przycisk "UCZ SIĘ", pozwalający na przejście do panelu z zadaniami. Po lewej stronie kolejno od góry znajduje się: panel z nazwą użytkownika, aktualnym poziomem i postępem punktów do zdobycia kolejnego, obecny stan konta monet, obecny stan konta dodatkowej waluty, stan żyć (odbierane są za błędną odpowiedź w zadaniach), przycisk zakupu dodatkowych żyć i licznik czasu pozostałego do odnowienia życia. Po prawej stronie ekranu znajdują się kolejno od góry, przyciski przenoszące do: sklepu, garderoby, zadań, osiągnięć oraz do wylogowania się z platformy.

Rysunek 5.48 Strona menu głównego (źródło: opracowanie własne)

Panel pytań

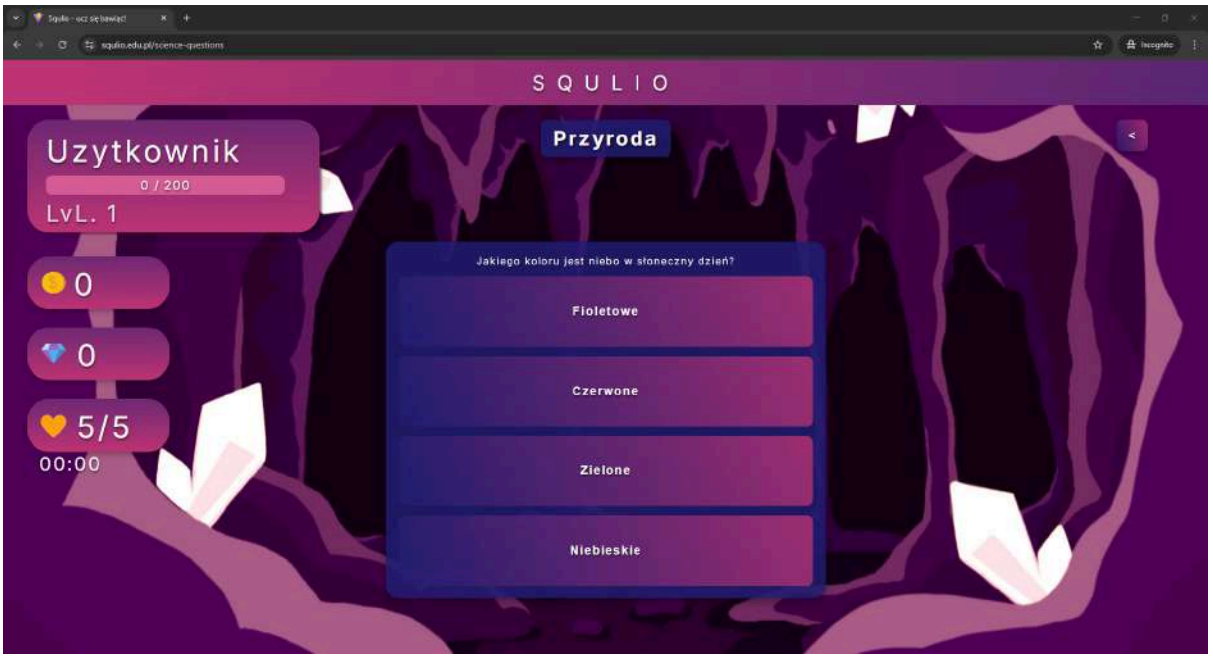
Po kliknięciu przycisku "UCZ SIĘ" pojawia się panel umożliwiający wybór tematyki quizów. Dzielą się one na dwie kategorie: "Codzienne Pytania", których celem jest utrzymywanie postaci użytkownika szczęśliwą oraz ich codzienne wykonywania oraz pytania szczegółowe z poszczególnych kategorii, które pozwalają na zdobycie dodatkowych punktów i poziomu.



Rysunek 5.49 Panel pytań (źródło: opracowanie własne)

Strona pytań

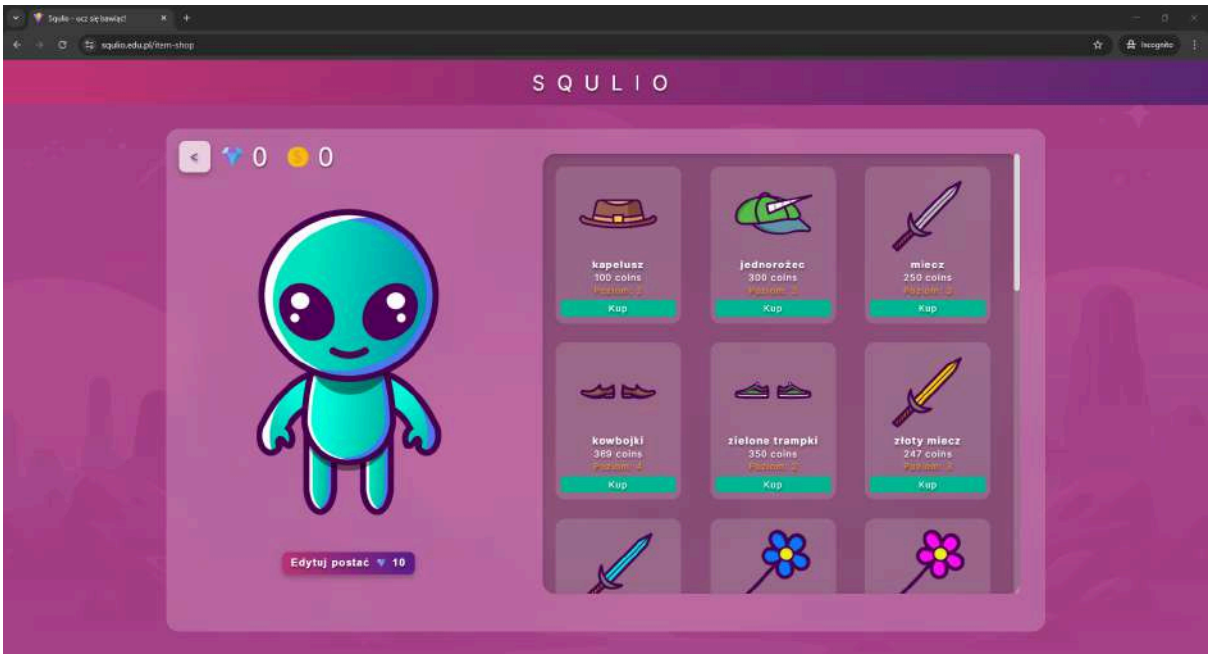
Po przejściu do wybranej przez użytkownika kategorii, przeniesiony zostaje on na stronę pytań, gdzie w centralnej części ekranu znajduje się panel z pytaniem oraz możliwymi odpowiedziami, nad nim kategoria, w której obecnie się znajduje, a po lewej stronie statystyki. Aby udzielić odpowiedzi na pytanie należy kliknąć jedną z podanych pod pytaniem opcji. Aplikacja poinformuje użytkownika o jej poprawności, wyświetli ewentualną dobrą odpowiedź i pozwoli przejść do kolejnego pytania. Wraz z kolejnymi poprawnymi odpowiedziami, poziom trudności rośnie co sygnalizowane jest przez zmianę tła strony. Przycisk wstecz umożliwiający powrót do menu głównego znajduje się w prawym górnym rogu ekranu.



Rysunek 5.50 Strona pytań (źródło: opracowanie własne)

Strona sklepu

W sklepie użytkownik może udekorować swoją postać wieloma akcesoriami dostępnymi do zakupu za zdobyte monety i poziom. W tym celu należy skorzystać z przesuwanego panelu znajdującego się po prawej stronie wybierając tam interesujący przedmiot oraz klikając przycisk "Kup". Należy jednak zwrócić uwagę na cenę przedmiotu wyrażoną w walucie "coins" oraz poziom potrzebny do jego zakupu. Obie te informacje znajdują się nad przyciskiem "Kup". Po lewej stronie od panelu sklepu znajduje się obecny wygląd postaci, nad nim stan konta użytkownika, a pod przycisk umożliwiający zakup edycji wyglądu postaci (zmiana wyglądu głowy, tułowia oraz nóg). Przenosi on do podstrony "Tworzenie postaci". Powrót do menu głównego jest możliwy klikając przycisk "wstecz" w lewym górnym rogu panelu.



Rysunek 5.51 Strona sklepu (źródło: opracowanie własne)

Strona garderoby

Zakupione w sklepie przedmioty można zobaczyć oraz założyć w garderobie. Po lewej stronie od postaci znajdują się kategorie takie jak: czapki, przedmioty do ręki i buty, które klikając dostosowują typ przedmiotów jakie wyświetlają się w panelu po drugiej stronie postaci. W tym też panelu klikając na wybrany item, zostaje on założony na postać, a ponowne jego kliknięcie powoduje jego zdjęcie.

Rysunek 5.52 Strona garderoby (źródło: opracowanie własne)

Panel zadań

Wybierając przycisk zadań, wyświetlony zostaje przewijany panel zawierający informacje o dostępnych do wykonania zadaniach, za które użytkownik może otrzymać dodatkową walutę (diamenty). Cyfry rzymskie po lewej stronie panelu pozwalają na wybranie poziomu trudności zadań, które wyświetlają się w panelu. Wyjście możliwe jest korzystając z przycisku "X" w prawym górnym rogu.

Rysunek 5.53 Panel zadań (źródło: opracowanie własne)

Panel osiągnięć

Wybierając przycisk osiągnięć, wyświetlony zostaje przewijany panel zawierający informacje o dostępnych osiągnięciach i o postępie w ich wykonywaniu. Symbole po prawej i lewej stronie każdego z osiągnięć oznaczają jego poziom, który wraz z kompletowaniem kolejnych z nich rośnie.

Rysunek 5.54 Panel osiągnięć (źródło: opracowanie własne)

Panel zakupu żyć

Jeśli podczas wykonywania zadań użytkownik popełni 5 błędów i stan żyć wyniesie 0/5, nie będzie miał on możliwości kontynuowania nauki do czasu odnowienia stanu żyć. Możliwe jest jednak zakupienie dodatkowych żyć, aby maksymalną wartością nie było 5, a więcej. W tym celu po kliknięciu "+" obok stanu żyć należy użyć przycisku "KUP". Aktualna cena dodatkowego serduszka wyświetlona jest nad przyciskiem i wraz z zakupem kolejnych - rośnie.

Rysunek 5.55 Panel zakupu żyć (źródło: opracowanie własne)

6. Testowanie oprogramowania

6.1 Wstęp do testów

Testowanie aplikacji to kluczowy etap procesu jej tworzenia. Ma na celu zapewnienie niezawodności, jakości i zgodności z założeniami projektowymi. Wybierając odpowiednie metody testowania jesteśmy w stanie zidentyfikować błędy na różnych etapach projektu i szybko je wyeliminować. Podczas tworzenia oprogramowania edukacyjnego kierowanego w głównej mierze do dzieci, niezwykle istotne jest dostosowanie strategii testowania do specyfiki aplikacji. Chcąc utrzymać wysoką jakość zespół projektowy podjął decyzję, aby kompleksowo podejść do testowania, wliczając w nie zarówno weryfikację poszczególnych funkcji, jak i ocenę działania całego oprogramowania. Spośród dostępnych możliwości wybrane zostały testy manualne oraz automatyczne wykonywane za pomocą narzędzia Postman z wykorzystaniem skryptów JavaScript.

Testy manualne umożliwiły ocenę interakcji użytkownika z interfejsem oraz zweryfikowanie, czy funkcje aplikacji działają zgodnie z oczekiwaniami. Natomiast testy API, podczas których zostało wykorzystane narzędzie Postman, pozwoliły na dokładną analizę działania konkretnych endpointów. Każdy z nich był testowany pod kątem zgodności zwracanych danych z założeniami funkcjonalnymi aplikacji, co umożliwiło wykrycie ewentualnych rozbieżności. Rezultaty testów odegrały istotną rolę w zapewnieniu wysokiej jakości aplikacji i stabilności jej działania. Umożliwiły zespołowi wczesne wykrycie oraz eliminację błędów. Z kolei analiza wyników dostarczyła rzeczowych informacji na temat poprawności implementacji oraz efektywności komunikacji pomiędzy poszczególnymi komponentami. Kolejne sekcje rozdziału będą skupione na szczegółowym opisaniu zastosowanych strategii testowych oraz ich wpływu na ostateczną wersję aplikacji.

6.2 Testy Postman

Testowanie API zostało wykonane przy użyciu narzędzia Postman, umożliwiającego wysyłanie żądań HTTP do back-endu aplikacji oraz badanie otrzymanych odpowiedzi. Poddana analizom została integralność połączeń między front-endem a back-endem, oceniając czy żądania wysyłane przez użytkownika są poprawnie przetwarzane przez serwer i czy zwracane odpowiedzi umożliwiają prawidłowe działanie aplikacji. Podczas testów nie została pominięta również weryfikacja reakcji systemu na rozmaite scenariusze.

Testowanie API objęło kilka istotnych kategorii:

- Testy funkcjonalne - sprawdzające, czy API zwraca prawidłowe odpowiedzi i jego działanie jest zgodne z wymaganiami.
- Testy walidacyjne - badające poprawność danych i ich zgodność ze specyfikacją.
- Testy wydajnościowe - koncentrują się na szybkości reakcji serwera oraz czasie w jakim przetwarzane są żądania.
- Testy integracyjne - analizują współpracę API z pozostałymi komponentami, takimi jak bazy danych.

Zastosowanie skryptów JavaScript w Postmanie, umożliwiło zautomatyzowanie części testów, co znacząco przyspieszyło proces weryfikacji i zwiększyło jego efektywność. Poniżej przedstawiona zostanie szczegółowa analiza wyników oraz ich wpływ na finalną wersję aplikacji.

6.2.1 Testy API za pomocą aplikacji PostMan

Testowane API:

```
router.post('/api/register', RegisterController.Register)
```

Przesłane dane w teście:

```
{
    "username": "testUser",
    "email": "test@example.com",
    "password": "Test1234"
}
```

Zwrócone wartości:

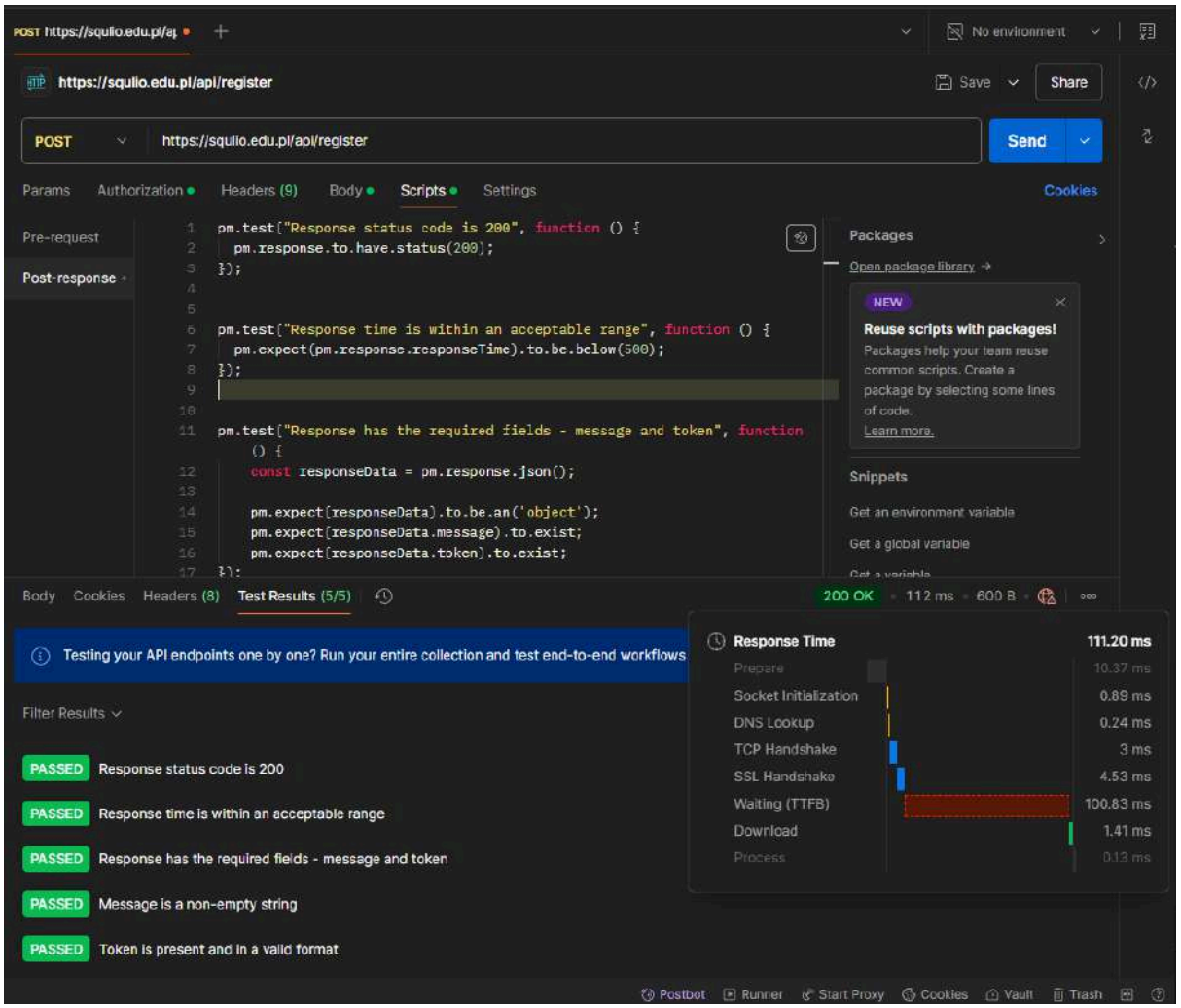
```
{
    "message": "Rejestracja zakończona sukcesem. Sprawdź swoją skrzynkę e-mail, aby
aktywować konto.",
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2N2M2YTQwODlhNTFkYmQyNGRmMTJhN
mMiLCJpYXQiOjE3NDEwNzEzNjgsImV4cCI6MTc0MTExNDU2OH0.Q3sHcUrEQ2VvjATVVsVElMZIP1nimDi5
ct9tksdGd3A"
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});


pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});


pm.test("Response has the required fields - message and token", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.message).to.exist;
    pm.expect(responseData.token).to.exist;
});


pm.test("Message is a non-empty string", function () {
  const responseData = pm.response.json();
```

```
pm.expect(responseData.message).to.exist.and.to.be.a('string').and.to.have.lengthOf
.at.least(1, "Message should not be empty");
});


pm.test("Token is present and in a valid format", function () {
    const responseData = pm.response.json();


pm.expect(responseData).to.have.property('token').that.is.a('string').and.not.empty
;
});
```

Wynik testu:



Rysunek 6.1 Wynik testu API: /api/register (źródło: opracowanie własne)

Testowane API:

```
router.post('/api/login', isBlocked, loginLimiter, detectInjection,
LoginController.Login)
```

Przesłane dane w teście:

```
{
    "email": "wojtek.pestka@onet.pl",
    "password": "admin123",
    "rememberMe": true
}
```

Zwrócone wartości:

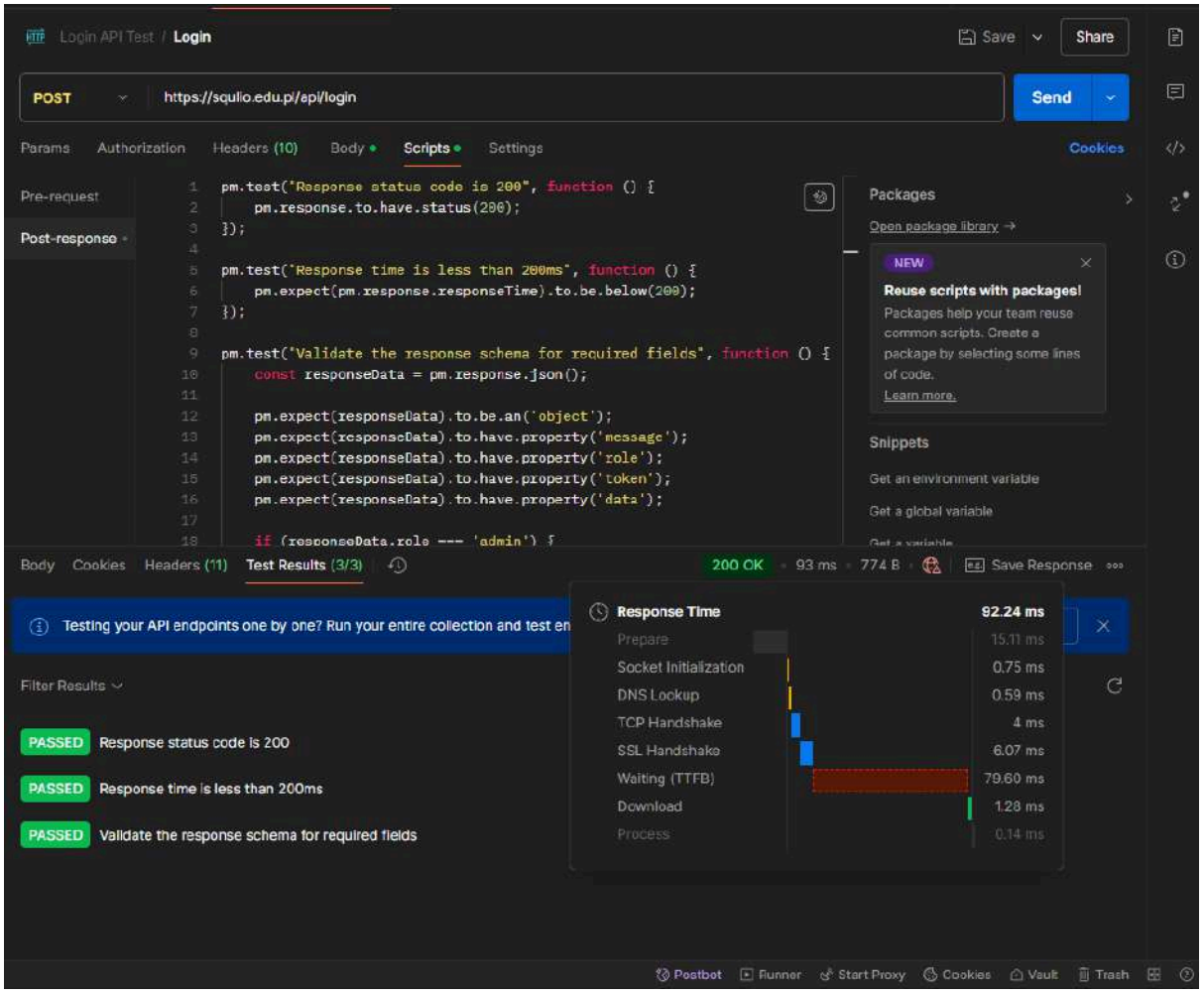

```
{
    "message": "Zalogowano pomyślnie",
    "role": "user",
    "view": "menu",
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2N2M1OWE5NjcxZTJjMmRmM2Q1YmYzM
WUiLCJyb2xlIjoiZ3JhY3oiLCJpYXQiOjE3NDEwNjg1ODgsImV4cCI6MTc0MTY3MzM4OH0.mVmZWC8LvfFB
Dq9BDPCzW040xT-Erq2gkjuAu_-6hgA",
    "data": {
        "name": "wojtek",
        "head": "/head/sad/h2_orange.png",
        "body": "/body/b3_blue.png",
        "legs": "/legs/l1_orange.png"
    }
}
```


Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Validate the response schema for required fields", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData).to.have.property('message');
    pm.expect(responseData).to.have.property('role');
    pm.expect(responseData).to.have.property('token');
    pm.expect(responseData).to.have.property('data');
```

```
    if (responseData.role === 'admin') {
        pm.expect(responseData.data).to.have.property('username');
        pm.expect(responseData.data).to.have.property('users');
        pm.expect(responseData.data).to.have.property('dailyquestions');
    } else if (responseData.role === 'user') {
        if (responseData.view === 'activation') {
            pm.expect(responseData.data).to.have.property('username');
        } else if (responseData.view === 'createcharacter') {
            pm.expect(responseData.data).to.have.property('username');
        } else if (responseData.view === 'menu') {
            pm.expect(responseData.data).to.have.property('name');
            pm.expect(responseData.data).to.have.property('head');
            pm.expect(responseData.data).to.have.property('body');
            pm.expect(responseData.data).to.have.property('legs');
        }
    }
});
```

Wyniki testu:



Rysunek 6.2 Wynik testu API: /api/login (źródło: opracowanie własne)

Testowane API:

```
router.post('/api/activation', authenticateUser,
ActivateUser.ActivateUser)
```

Przesłane dane w teście:

```
{
    "activationKey": "269694"
}
```

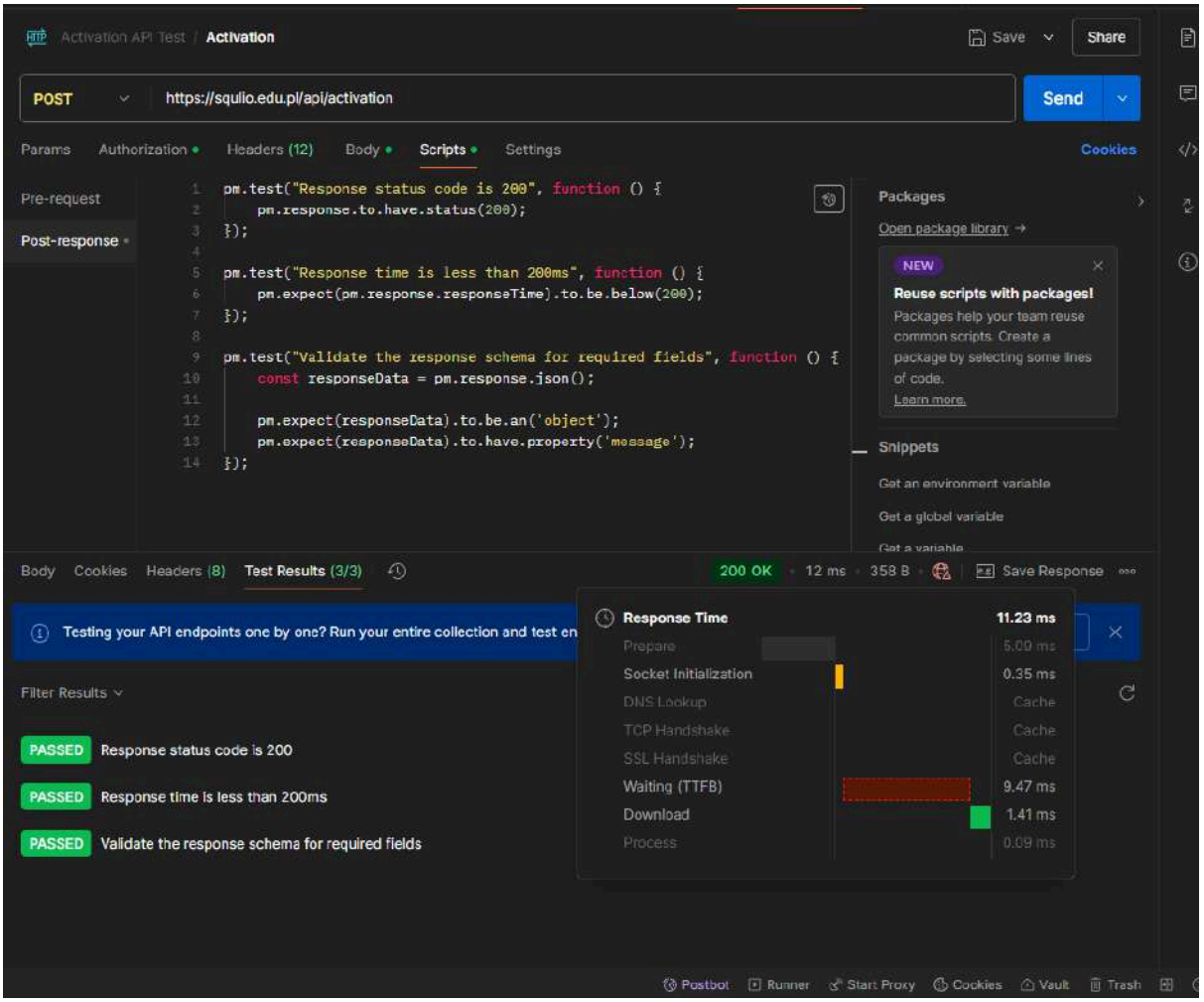
Zwrócone wartości:

```
{
    "message": "Konto zostało aktywowane pomyślnie"
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Validate the response schema for required fields", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData).to.have.property('message');
});
```

Wynik testu:



Rysunek 6.3 Wynik testu API: /api/activation (źródło: opracowanie własne)

Testowane API:

```
router.post('/api/saveCharacter', authenticateUser,
checkEditableCharacter, CharacterController.Savecharacter)
```

Przesłane dane w teście:

```
{
    "name": "wojtek",
    "head": "/head/sad/h2_orange.png",
    "body": "/body/b3_blue.png",
    "legs": "/legs/l1_orange.png"
```

Zwrócone wartości:

```
{
    "updatedUser": {
        "_id": "67c59a9671e2c2df3d5bf31e",
        "username": "wojtek",
```

```
        "email": "wojtek.pestka@onet.pl",
        "password": "$2b$10$v8D6ASbz9ZgVuT/iW45j1OL6gKzVddnnV4jfzMqBf1JTNNT7tNIAm",
        "type": "gracz",
        "activationKey": "269694",
        "userStatus": "enabled",
        "head": "/head/sad/h2_orange.png",
        "answeredQuestions": [],
        "completedAllQuestionsToday": false,
        "lastCompletedDate": "2025-02-20T00:00:00.000Z",
        "englishQuestionElo": 1000,
        "mathsQuestionElo": 800,
        "scienceQuestionElo": 1000,
        "programmingQuestionElo": 5000,
        "correct_programmingQuestion": 600,
        "correct_mathsQuestion": 0,
        "correct_englishQuestion": 0,
        "correct_scienceQuestion": 0,
        "level": 10,
        "coins": 2959,
        "extracoins": 0,
        "happines": 1,
        "exp": 100,
        "hp": 5,
        "hpLimit": 5,
        "lastHpLoss": "2025-03-04T02:27:00.173Z",
        "items": [
            3,
            8,
            5,
            1
        ],
        "extracoinsgiven": [],
        "editableCharacter": false,
        "createdAt": "2025-03-03T12:03:34.581Z",
        "tasks_progress": [
            {
                "id_2": 0,
                "id_5": 0,
                "id_13": 0,
                "id_18": 0,
                "id_20": 0,
                "id_22": 0,
                "id_24": 0,
                "_id": "67c6a0629a51dbd24df129fe",
                "id_25": 4,
```

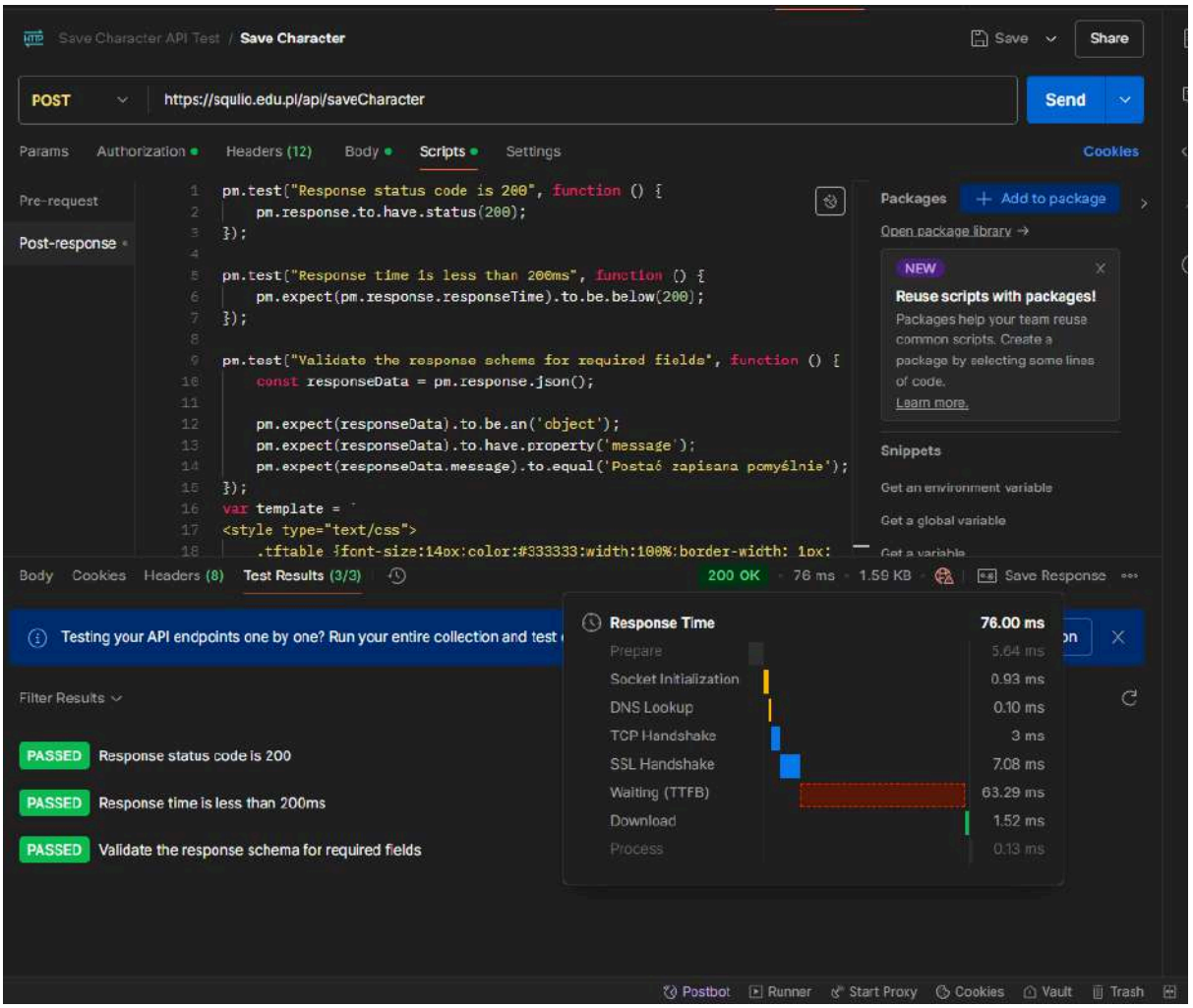
```
                "id_19": 10,
                "id_1": 1,
                "id_11": 2,
                "id_12": 0,
                "id_16": 2,
                "id_17": 0,
                "id_21": 2,
                "id_23": 0,
                "id_6": 2,
                "id_7": 2,
                "id_10": 2,
                "id_4": 1,
                "id_14": 3,
                "id_3": 1,
                "id_8": 1,
                "id_9": 1,
                "id_15": 1
            }
        ],
        "__v": 4,
        "body": "/body/b3_blue.png",
        "legs": "/legs/l1_orange.png",
        "name": "wojtek",
        "headSkin": "1",
        "armSkin": "8",
        "legSkin": "5",
        "lastLogin": "2025-03-04T06:24:05.893Z"
    },
    "message": "Postać zapisana pomyślnie"
}
```


Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Validate the response schema for required fields", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData).to.have.property('message');
    pm.expect(responseData.message).to.equal('Postać zapisana pomyślnie');
```

```
});
```

Wyniki testu:



Rysunek 6.4 Wynik testu API: /api/saveCharacter (źródło: opracowanie własne)

Testowane API:

```
router.get('/api/character', authenticateUser,
GetCharacterController.getCharacter)
```

Przesłane dane w teście:
Jest to zapytanie get więc nie wysyłamy żadnych danych na serwer

Zwrócone wartości:

```
{
    "name": "wojtek",
    "head": "/head/sad/h2_orange.png",
    "body": "/body/b3_blue.png",
    "legs": "/legs/l1_orange.png",
    "level": 10,
    "coins": 2959,
```

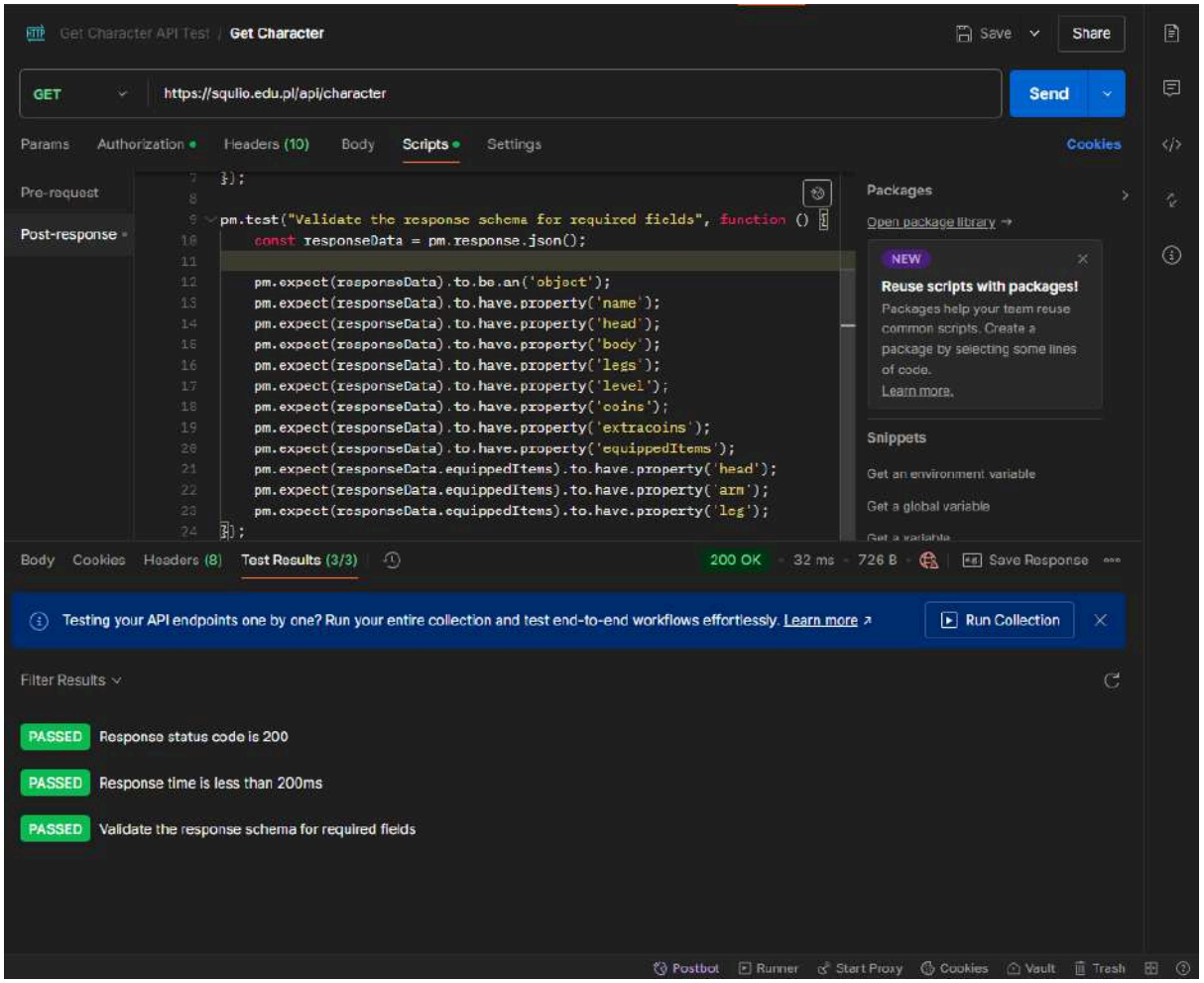
```
    "exp": 100,
    "extracoins": 0,
    "hp": 5,
    "hpLimit": 5,
    "timeLeft": 0,
    "happiness": 1,
    "equippedItems": {
        "head": {
            "id": 1,
            "name": "kapelusz",
            "url": "/Skins/Hats/hat.png"
        },
        "arm": {
            "id": 8,
            "name": "diamentowy miecz",
            "url": "/Skins/Arms/item3.png"
        },
        "leg": {
            "id": 5,
            "name": "kowbojki",
            "url": "/Skins/Legs/shoes1.png"
        }
    }
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Validate the response schema for required fields", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData).to.have.property('name');
    pm.expect(responseData).to.have.property('head');
    pm.expect(responseData).to.have.property('body');
    pm.expect(responseData).to.have.property('legs');
    pm.expect(responseData).to.have.property('level');
    pm.expect(responseData).to.have.property('coins');
    pm.expect(responseData).to.have.property('extracoins');
    pm.expect(responseData).to.have.property('equippedItems');
```

```
    pm.expect(responseData.equippedItems).to.have.property('head');
    pm.expect(responseData.equippedItems).to.have.property('arm');
    pm.expect(responseData.equippedItems).to.have.property('leg');
});
```

Wynik testu:



Rysunek 6.5 Wynik testu API: /api/character (źródło: opracowanie własne)

Testowane API:

```
router.post('/CreateDailyQuestion',
AdminController.CreateDailyQuestion)
```

Przesłane dane w teście:

```
{
    "question": "What is the capital of France?",
    "correct_answer": "Paris",
    "incorrect_answer_1": "London",
    "incorrect_answer_2": "Berlin",
    "incorrect_answer_3": "Madrid"
}
```

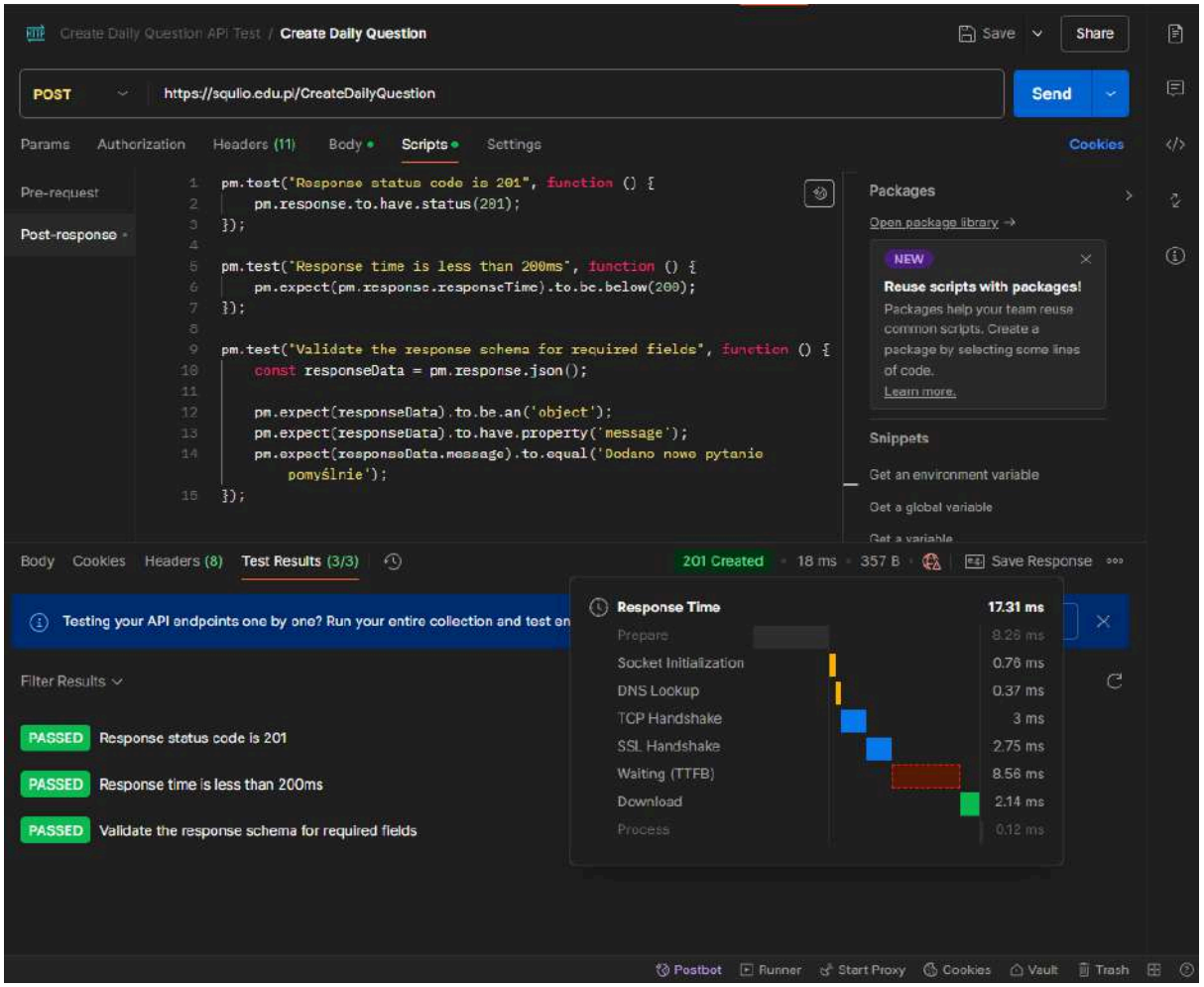
Zwrócone wartości:

```
{
    "message": "Dodano nowe pytanie pomyślnie"
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 201", function () {
    pm.response.to.have.status(201);
});

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Validate the response schema for required fields", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData).to.have.property('message');
    pm.expect(responseData.message).to.equal('Dodano nowe pytanie pomyślnie');
});
```

Wynik testu:



Rysunek 6.6 Wynik testu API: /api/CreateDailyQuestion (źródło: opracowanie własne)

Testowane API:

```
router.post('/CreateMathsQuestion',
AdminController.CreateMathsQuestion)
router.post('/CreateProgrammingQuestion',
AdminController.CreateProgrammingQuestion);
router.post('/CreateScienceQuestion',
AdminController.CreateScienceQuestion);
router.post('/CreateEnglishQuestion',
AdminController.CreateEnglishQuestion);
```

Z uwagi na dużą podobność do wcześniej testowanego API, zdecydowano się pominąć testowanie niektórych endpointów. Konkretnie, API odpowiadające za tworzenie pytań z matematyki, programowania, nauk ścisłych oraz języka angielskiego zostały pominięte, ponieważ ich funkcjonalności są niemal identyczne, co sprawia, że testowanie ich oddzielnie nie wprowadziłoby nowych informacji. Skupiono się na testach, które różnią się istotnie w zakresie implementacji i logiki działania.

Testowane API:

```
router.get('/api/daily-question', authenticateUser,
DailyQuestionsController.getDailyQuestion);
```

Przesłane dane w teście:
Żądanie typu GET służy wyłącznie do pobierania danych, dlatego nie wymaga przesyłania żadnych informacji na serwer.

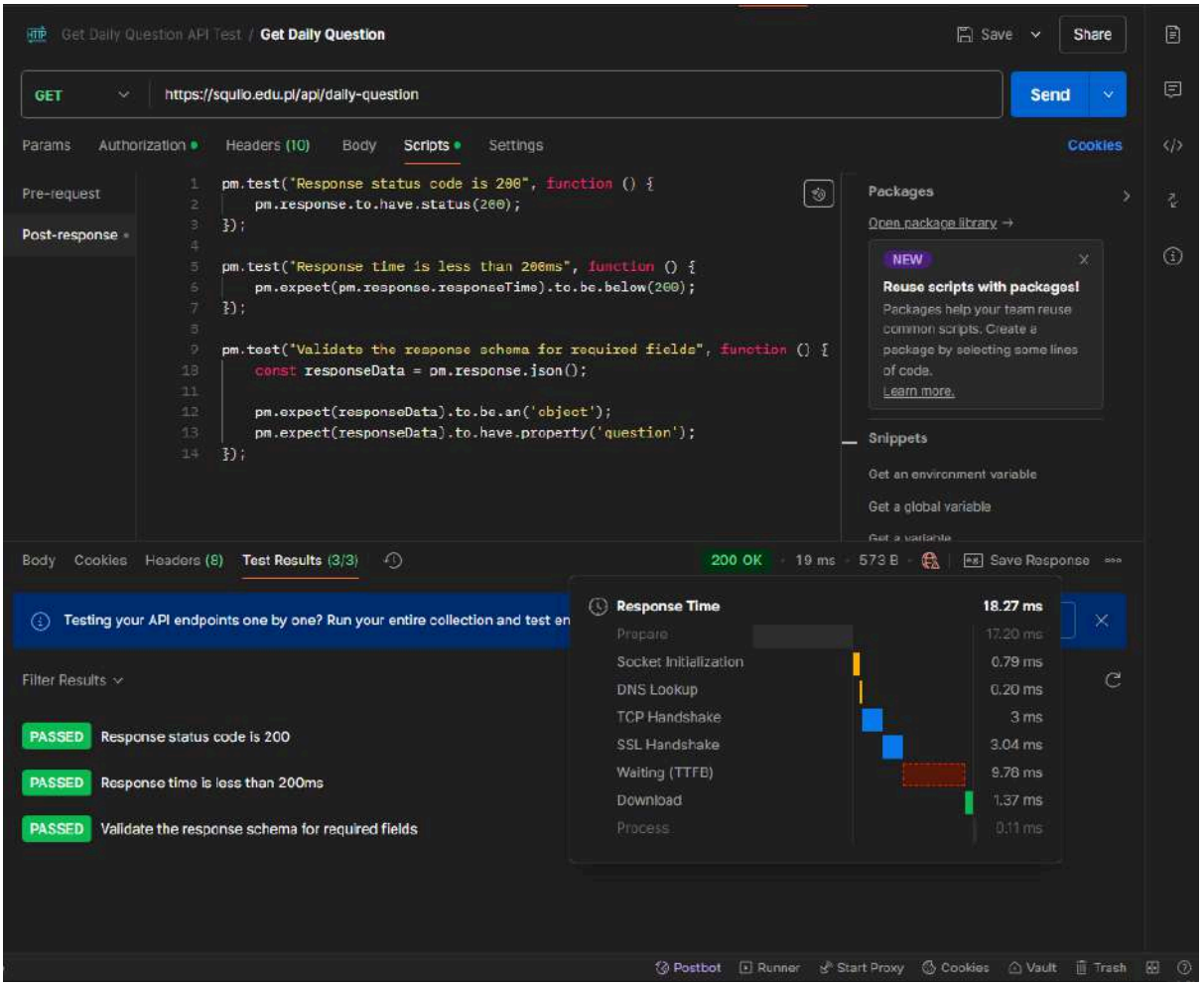
Zwrócone wartości:

```
{
    "id_question": 3,
    "question": "Kto jest autorem powieści „Krzyżacy”?",
    "correct_answer": "Henryk Sienkiewicz",
    "incorrect_answer_1": "Adam Mickiewicz",
    "incorrect_answer_2": "Juliusz Słowacki",
    "incorrect_answer_3": "Bolesław Prus",
    "_id": "67c634703ad3be7db778effd"
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Validate the response schema for required fields", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData).to.have.property('question');
});
```

Wynik testu:



Rysunek 6.7 Wynik testu API: /api/daily-question (źródło: opracowanie własne)

Testowane API:

```
router.post('/api/submit-answer', authenticateUser,
DailyQuestionsController.submitAnswer);
```

Przesłane dane w teście:

```
{
    "questionId": 3,
    "answer": "Henryk Sienkiewicz"
}
```

Zwrócone wartości:

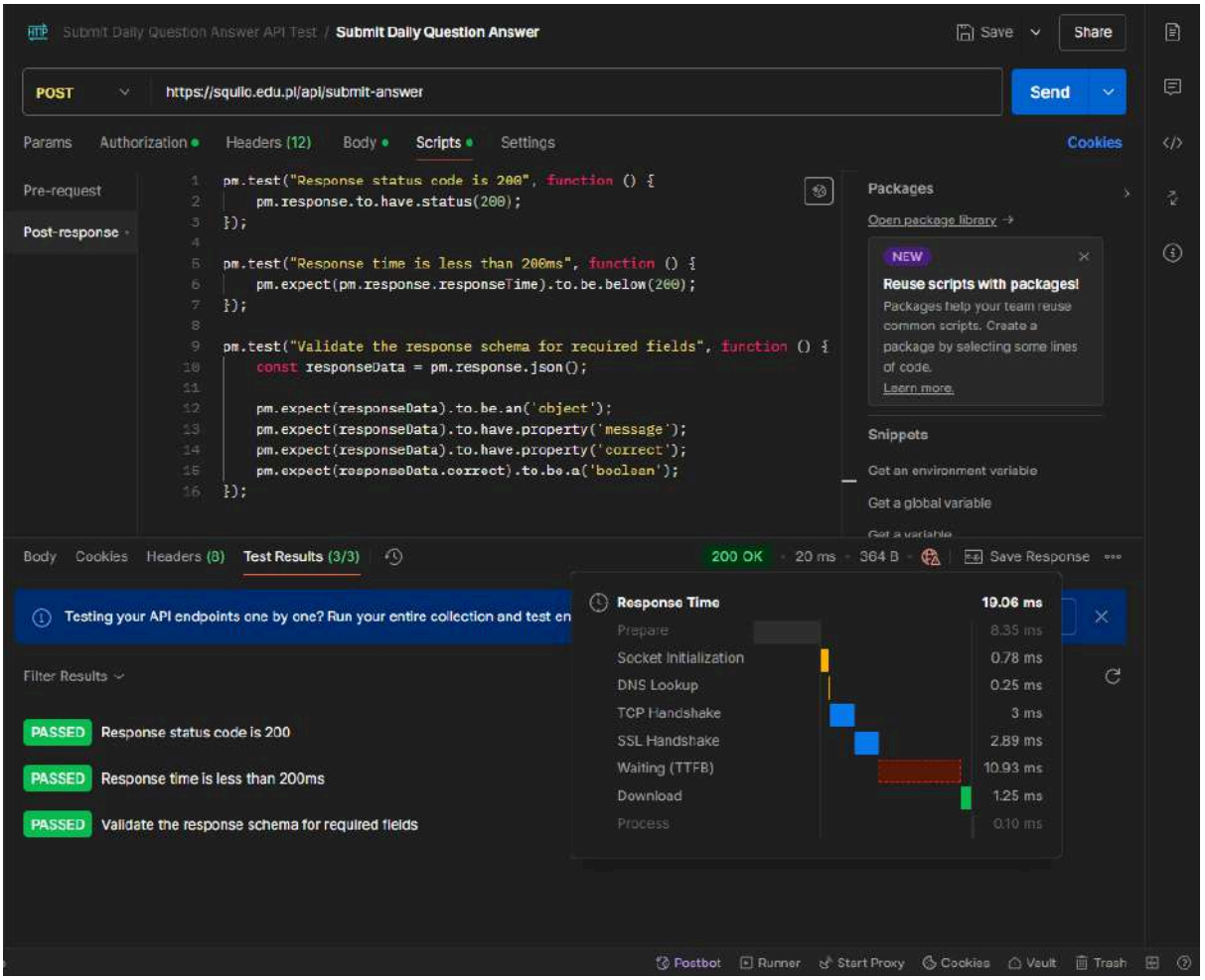
```
{
    "correct": true,
    "message": "Brawo! Poprawna odpowiedź!"
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
```

```
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Validate the response schema for required fields", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData).to.have.property('message');
    pm.expect(responseData).to.have.property('correct');
    pm.expect(responseData.correct).to.be.a('boolean');
});
```

Wynik testu:



Rysunek 6.8 Wynik testu API: /api/submit-answer (źródło: opracowanie własne)

Testowane API:

```
router.get('/api/math-questions/:level', authenticateUser,
MathQuestionsController.getMathQuestion);
```

Przesłane dane w teście:
Jest to zapytanie get więc nie wysyłamy żadnych danych na serwer

Zwrócone wartości:

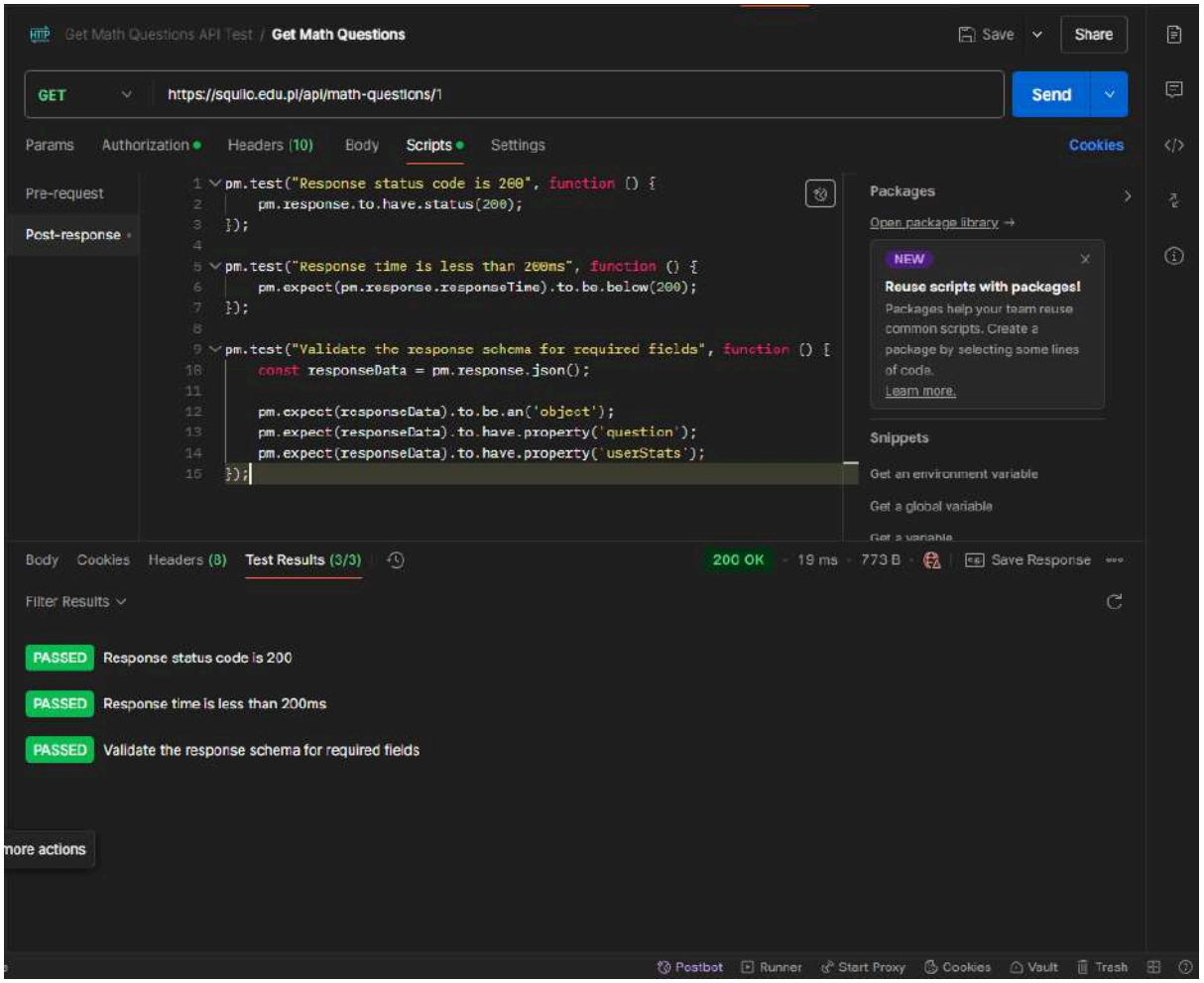

```
{
    "question": {
        "date": "2024-12-15T00:00:00.000Z",
        "id_question": 8,
        "question": "Co jest większe: 5 czy 3?",
        "correct_answer": "5",
        "incorrect_answer_1": "3",
        "incorrect_answer_2": "Są równe",
        "incorrect_answer_3": "Nie można określić",
        "_id": "675f5d39963e22a86230ad81"
    },
    "userStats": {
        "currentLevel": 10,
        "currentExp": 100,
        "requiredExpForNextLevel": 7688,
        "currentCoins": 2959,
        "currentElo": 800,
        "currentHp": 5,
        "hpLimit": 5,
        "extracoins": 0,
        "name": "wojtek",
        "timeLeft": 0,
        "taskLevel": 1
    }
}
```


Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Validate the response schema for required fields", function () {
    const responseData = pm.response.json();
```

```
    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData).to.have.property('question');
    pm.expect(responseData).to.have.property('userStats');
});
```

Wynik testu:



Rysunek 6.9 Wynik testu API: /api/math-questions/:level (źródło: opracowanie własne)

Testowane API:

```
router.post('/api/submit-math-answer', authenticateUser,
MathQuestionsController.submitMathAnswer);
```

Przesłane dane w teście:

```
{
    "questionId": 8,
    "answer": "5"
}
```

Zwrócone wartości:

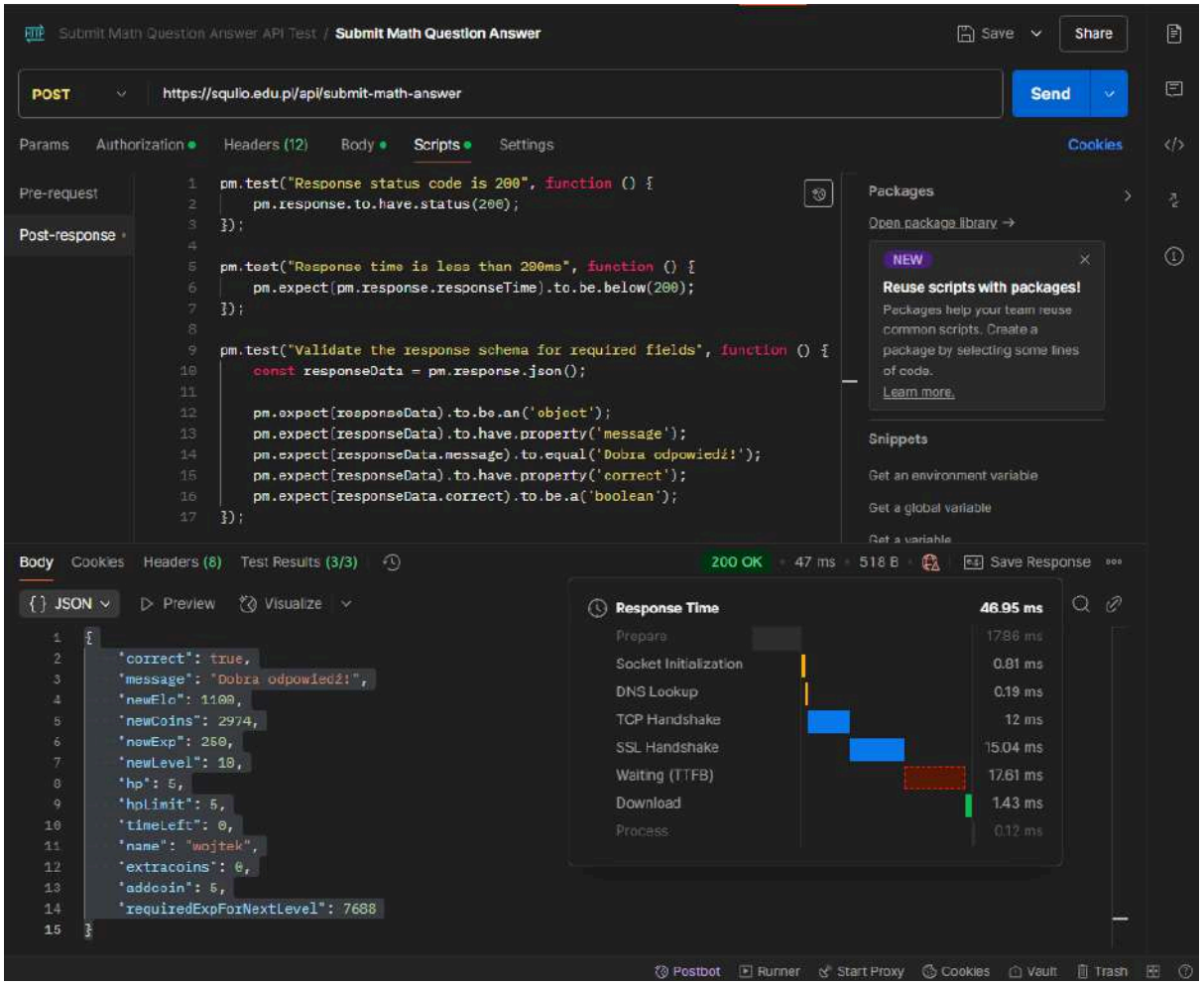
```
{
    "correct": true,
```

```
    "message": "Dobra odpowiedź!",
    "newElo": 1100,
    "newCoins": 2974,
    "newExp": 250,
    "newLevel": 10,
    "hp": 5,
    "hpLimit": 5,
    "timeLeft": 0,
    "name": "wojtek",
    "extracoins": 0,
    "addcoin": 5,
    "requiredExpForNextLevel": 7688
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Validate the response schema for required fields", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData).to.have.property('message');
    pm.expect(responseData.message).to.equal('Dobra odpowiedź!');
    pm.expect(responseData).to.have.property('correct');
    pm.expect(responseData.correct).to.be.a('boolean');
});
```

Wynik testu:



Rysunek 6.10 Wynik testu API: /api/submit-math-answer (źródło: opracowanie własne)

W przeprowadzonych testach skupiono się na sprawdzeniu poprawności działania API obsługującego pytania matematyczne, w tym ich pobieranie oraz przesyłanie odpowiedzi. Wyniki testów potwierdziły prawidłowe działanie endpointów oraz zgodność zwracanych danych ze strukturą odpowiedzi.

Pozostałe API odpowiedzialne za obsługę pytań z innych dziedzin, tj.:

router.get('/api/science-questions/:level') oraz router.post('/api/submit-science-answer'),

router.get('/api/english-questions/:level') oraz router.post('/api/submit-english-answer'),

router.get('/api/programing-questions/:level') oraz router.post('/api/submit-programing-answer'),

nie zostały poddane oddzielnym testom, ponieważ ich struktura i działanie są analogiczne do przetestowanych endpointów matematycznych. Każde z wymienionych API służy do pobierania pytań z określonego poziomu trudności oraz do przesyłania odpowiedzi na nie, co sugeruje wysokie prawdopodobieństwo ich poprawnego działania, o ile podstawowy mechanizm testowany na pytaniach matematycznych działa prawidłowo.

Przeprowadzanie dodatkowych testów dla każdej z wymienionych dziedzin byłoby redundantne i nie wnosiłoby nowych informacji o funkcjonalności systemu.

- Podjęta decyzja o pominięciu testów dla pozostałych API wynika z następujących przesłanek:
  -
- Jednolita struktura API — wszystkie endpointy są zbudowane według tego samego schematu i wykorzystują podobne mechanizmy autoryzacji oraz przetwarzania danych.
  -
- Spójność zwracanych danych — w testach matematycznych zwrócone dane zawierały wszystkie wymagane pola, co wskazuje na ich prawidłowe generowanie.
  -
- Efektywność procesu testowania — testowanie każdego z tych API w oddzielny sposób nie wnosiłoby dodatkowej wartości, a jedynie wydłużyłoby czas testów bez istotnej korzyści analitycznej.

W związku z powyższym uznano, że testy przeprowadzone na API dotyczącym pytań matematycznych są wystarczające do oceny poprawności implementacji całego systemu obsługi pytań w aplikacji.

Testowane API:

```
router.post('/CreateShopItem', AdminController.CreateShopItem);
```

Przesłane dane w teście:

```
{
    "id": 29,
    "type": "leg",
    "url": "/Skins/Legs/shoes7473.png",
    "requiredLevel": 10,
    "cost": 500,
    "name": "Magic Swordddddd"
}
```

Zwrócone wartości:

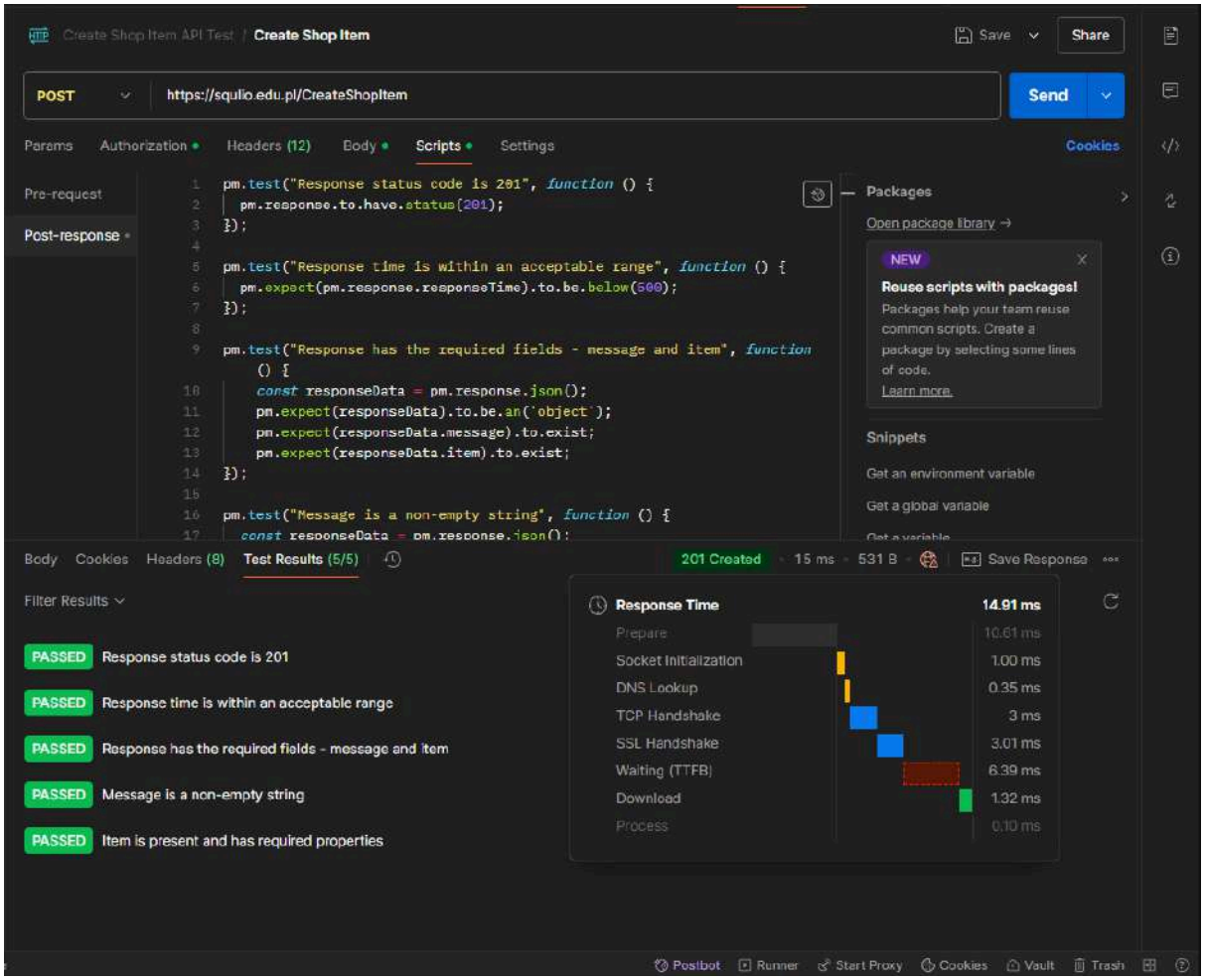
```
{
    "message": "Dodano nowy przedmiot do sklepu pomyślnie",
    "item": {
        "id": 29,
        "name": "Magic Swordddddd",
        "type": "leg",
        "url": "/Skins/Legs/shoes7473.png",
        "requiredLevel": 10,
        "cost": 500,
        "_id": "67c77fba8f1ee8e132456ec4",
        "__v": 0
```

```
    }
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 201", function () {
  pm.response.to.have.status(201);
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});

pm.test("Response has the required fields - message and item", function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.message).to.exist;
    pm.expect(responseData.item).to.exist;
});

pm.test("Message is a non-empty string", function () {
  const responseData = pm.response.json();

pm.expect(responseData.message).to.exist.and.to.be.a('string').and.to.have.lengthOf
.at.least(1, "Message should not be empty");
});

pm.test("Item is present and has required properties", function () {
    const responseData = pm.response.json();
    pm.expect(responseData.item).to.be.an('object');

pm.expect(responseData.item).to.have.property('type').that.is.a('string').and.not.e
mpty;

pm.expect(responseData.item).to.have.property('url').that.is.a('string').and.not.em
pty;

pm.expect(responseData.item).to.have.property('requiredLevel').that.is.a('number');
    pm.expect(responseData.item).to.have.property('cost').that.is.a('number');

pm.expect(responseData.item).to.have.property('name').that.is.a('string').and.not.e
mpty;
});
```

Wynik testu:



Rysunek 6.11 Wynik testu API: /CreateShopItem (źródło: opracowanie własne)

Testowane API:

```
router.get('/api/shop/items', authenticateUser,
ItemShopController.getShopItems);
```

Przesłane dane w teście:
Żądanie typu GET służy wyłącznie do pobierania danych, dlatego nie wymaga przesyłania żadnych informacji na serwer.

Zwrócone wartości:

```
[
    {
        "_id": "6738146331c5db182492befe",
        "id": 2,
        "type": "hat",
        "url": "/Skins/Hats/hat1.png",
        "requiredLevel": 3,
        "cost": 300,
        "__v": 0,
```

```
        "name": "jednorożec"
    },
    {
        "_id": "6738160131c5db182492bf0a",
        "id": 6,
        "type": "leg",
        "url": "/Skins/Legs/shoes2.png",
        "requiredLevel": 2,
        "cost": 350,
        "__v": 0,
        "name": "zielone trampki"
    },
```

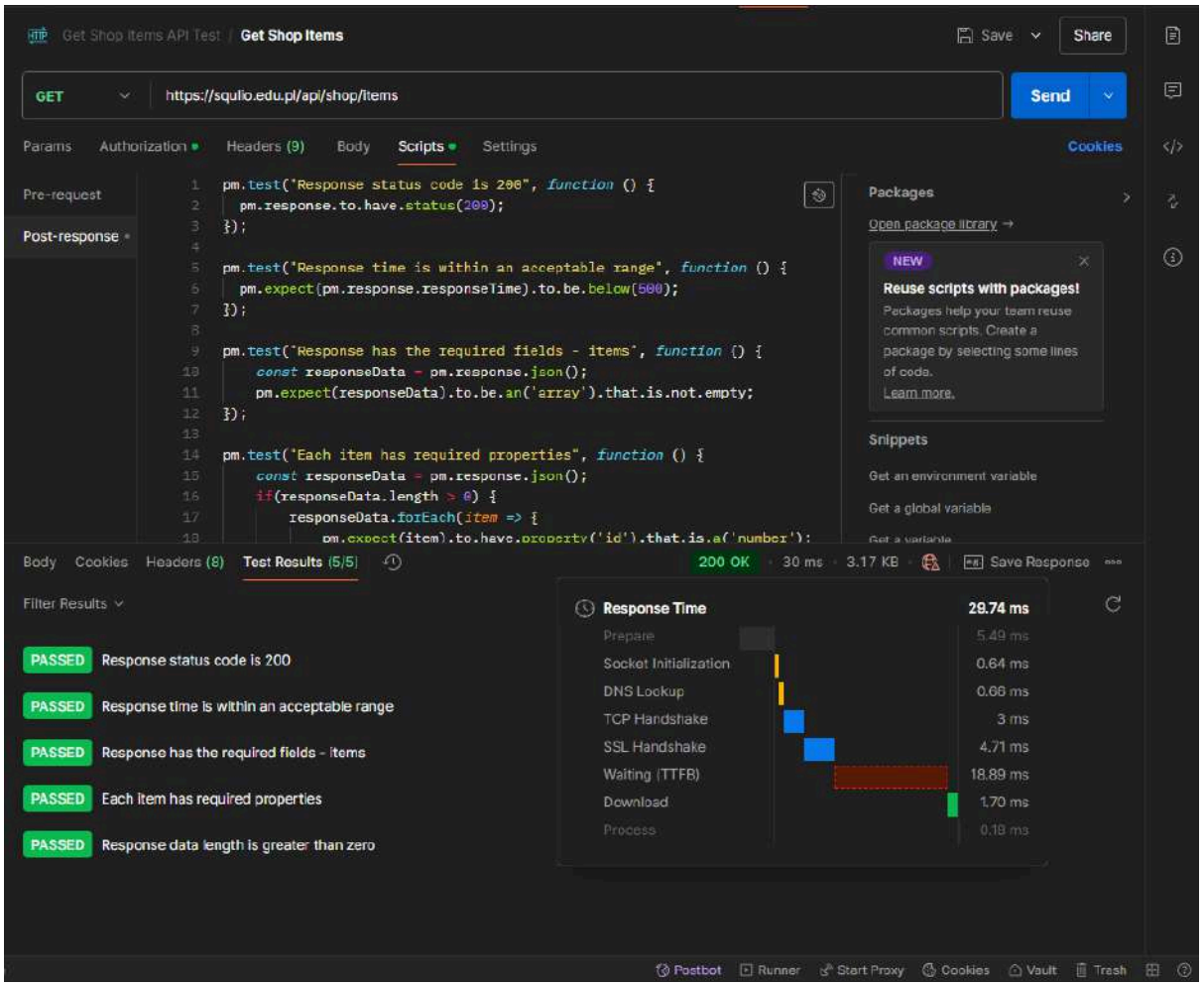

Wszystkie dostępne przedmioty znajdujące się w sklepie zostały poprawnie zwrócone. Ze względu na obszerność odpowiedzi, nie przedstawiono ich w całości.

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});

pm.test("Response has the required fields - items", function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('array').that.is.not.empty;
});

pm.test("Each item has required properties", function () {
    const responseData = pm.response.json();
    if(responseData.length > 0) {
        responseData.forEach(item => {
            pm.expect(item).to.have.property('id').that.is.a('number');

pm.expect(item).to.have.property('type').that.is.a('string').and.not.empty;

pm.expect(item).to.have.property('url').that.is.a('string').and.not.empty;
            pm.expect(item).to.have.property('requiredLevel').that.is.a('number');
            pm.expect(item).to.have.property('cost').that.is.a('number');

pm.expect(item).to.have.property('name').that.is.a('string').and.not.empty;
        });
    }
});
```

```
pm.test("Response data length is greater than zero", function () {
    const responseData = pm.response.json();
    pm.expect(responseData.length).to.be.greaterThan(0, "Response data should not
be empty");
});
```

Wynik testu:



Rysunek 6.12 Wynik testu API: /api/shop/items (źródło: opracowanie własne)

Testowane API:

```
router.post('/api/shop/buy-item', authenticateUser,
ItemShopController.BuyItem);
```

Przesłane dane w teście:

```
{
    "itemId": 1
}
```

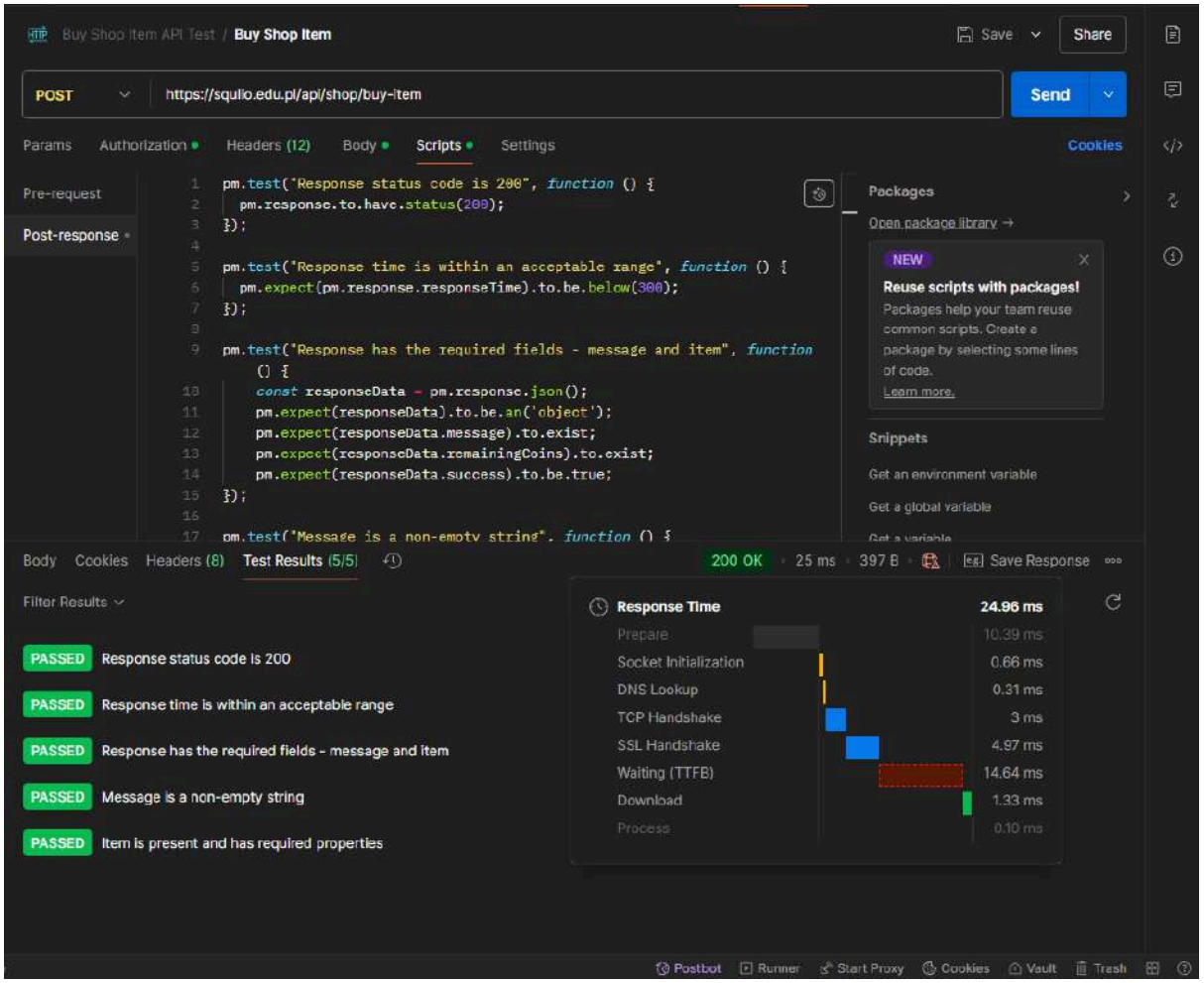
Zwrócone wartości:

```
{
    "message": "Pomyślnie wybrano przedmiot: kapelusz",
    "remainingCoins": 2874,
    "success": true
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(300);
});

pm.test("Response has the required fields - message and item", function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.message).to.exist;
    pm.expect(responseData.remainingCoins).to.exist;
    pm.expect(responseData.success).to.be.true;
});

pm.test("Message is a non-empty string", function () {
  const responseData = pm.response.json();

pm.expect(responseData.message).to.exist.and.to.be.a('string').and.to.have.lengthOf
.at.least(1, "Message should not be empty");
});

pm.test("Item is present and has required properties", function () {
    const responseData = pm.response.json();
    pm.expect(responseData.remainingCoins).to.be.a('number');
});
```

Wynik testu:



Rysunek 6.13 Wynik testu API: /api/shop/buy-item (źródło: opracowanie własne)

Testowane API:

```
router.post('/api/buy-edit-character', authenticateUser,
ItemShopController.BuyEditCharacter);
```

Przesłane dane w teście:
Żadne dane w tym żądaniu nie są przesyłane

Zwrócone wartości:

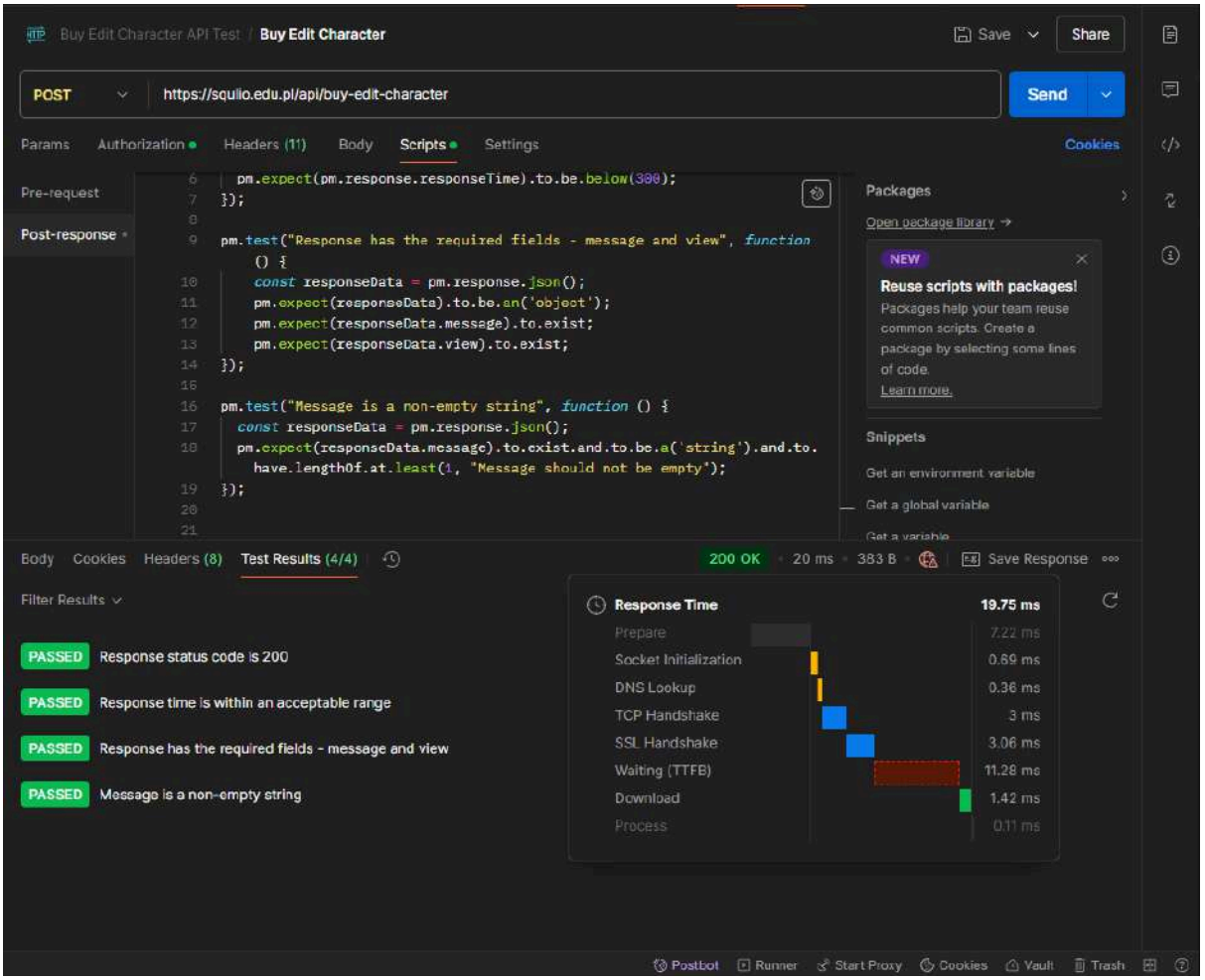
```
{
    "message": "Pomyślnie zakupiono edycję postaci",
    "view": "createcharacter"
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});
```

```
pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(300);
});

pm.test("Response has the required fields - message and view", function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.message).to.exist;
    pm.expect(responseData.view).to.exist;
});

pm.test("Message is a non-empty string", function () {
  const responseData = pm.response.json();

pm.expect(responseData.message).to.exist.and.to.be.a('string').and.to.have.lengthOf
.at.least(1, "Message should not be empty");
});
```

Wynik testu:



Rysunek 6.14 Wynik testu API: /api/buy-edit-character (źródło: opracowanie własne)

Testowane API:

```
router.get('/api/wardrobe/items', authenticateUser,
WardrobeController.getWardrobeItems);
```

Przesłane dane w teście:
Żądanie typu GET służy wyłącznie do pobierania danych, dlatego nie wymaga przesyłania żadnych informacji na serwer.

Zwrócone wartości:

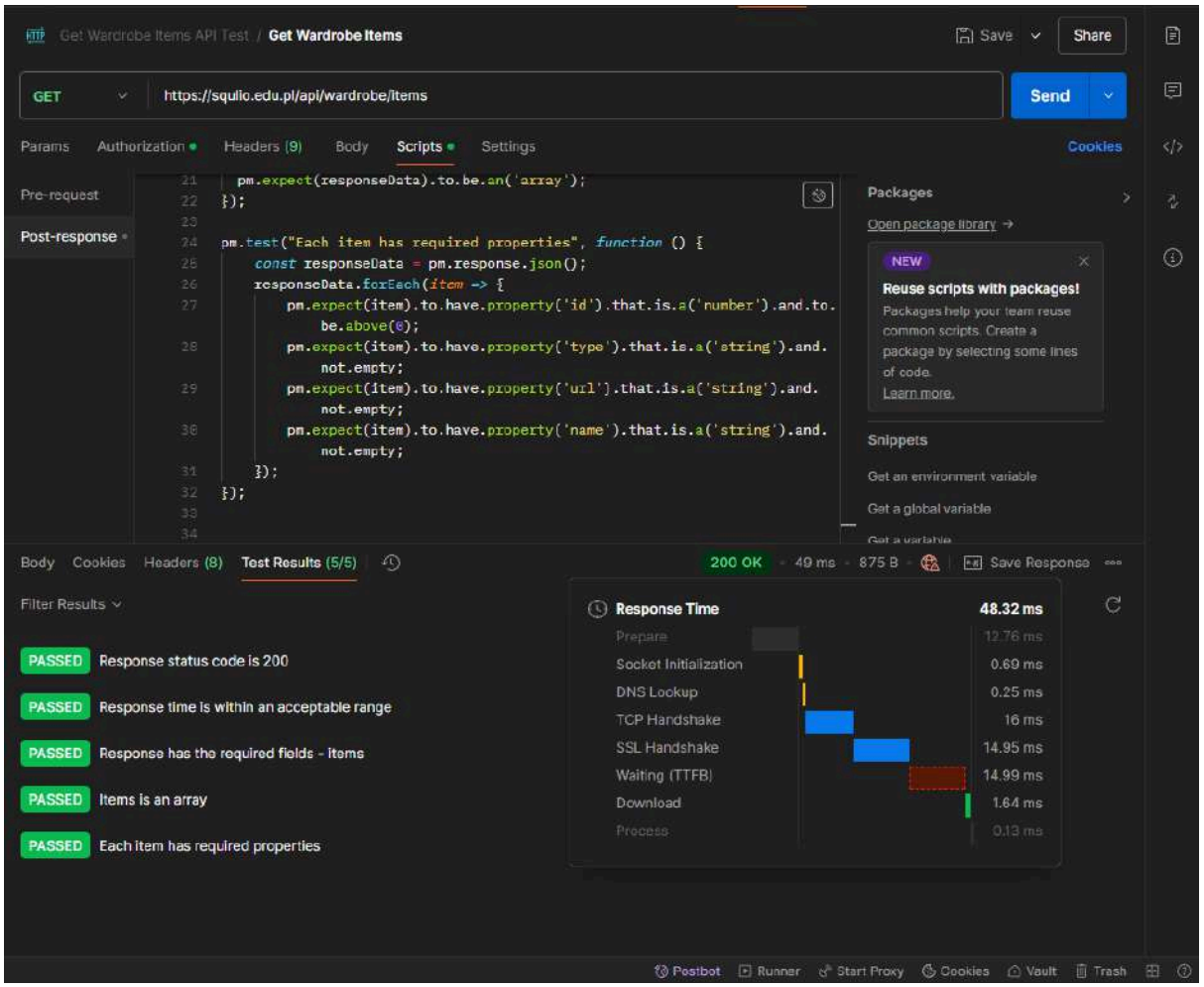
```
[
    {
        "_id": "673812a431c5db182492befb",
        "id": 1,
        "type": "hat",
        "url": "/Skins/Hats/hat.png",
        "requiredLevel": 2,
        "cost": 100,
        "__v": 0,
        "name": "kapelusz"
```

```
    },
    {
        "_id": "673814b731c5db182492bf01",
        "id": 3,
        "type": "arm",
        "url": "/Skins/Arms/item1.png",
        "requiredLevel": 3,
        "cost": 250,
        "__v": 0,
        "name": "miecz"
    },
]
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});

pm.test("Response has the required fields - items", function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('array');
    pm.expect(responseData[0]).to.have.property('id');
    pm.expect(responseData[0]).to.have.property('type');
    pm.expect(responseData[0]).to.have.property('url');
    pm.expect(responseData[0]).to.have.property('name');
});

pm.test("Items is an array", function () {
  const responseData = pm.response.json();
  pm.expect(responseData).to.be.an('array');
});

pm.test("Each item has required properties", function () {
    const responseData = pm.response.json();
    responseData.forEach(item => {

pm.expect(item).to.have.property('id').that.is.a('number').and.to.be.above(0);
        pm.expect(item).to.have.property('type').that.is.a('string').and.not.empty;
        pm.expect(item).to.have.property('url').that.is.a('string').and.not.empty;
        pm.expect(item).to.have.property('name').that.is.a('string').and.not.empty;
    });
});
```

Wynik testu:



Rysunek 6.15 Wynik testu API: /api/wardrobe/items (źródło: opracowanie własne)

Testowane API:

```
router.post('/api/select-item', authenticateUser,
WardrobeController.selectItem);
```

Przesłane dane w teście:

```
{
    "itemId": 1,
    "isEquipped": true
}
```

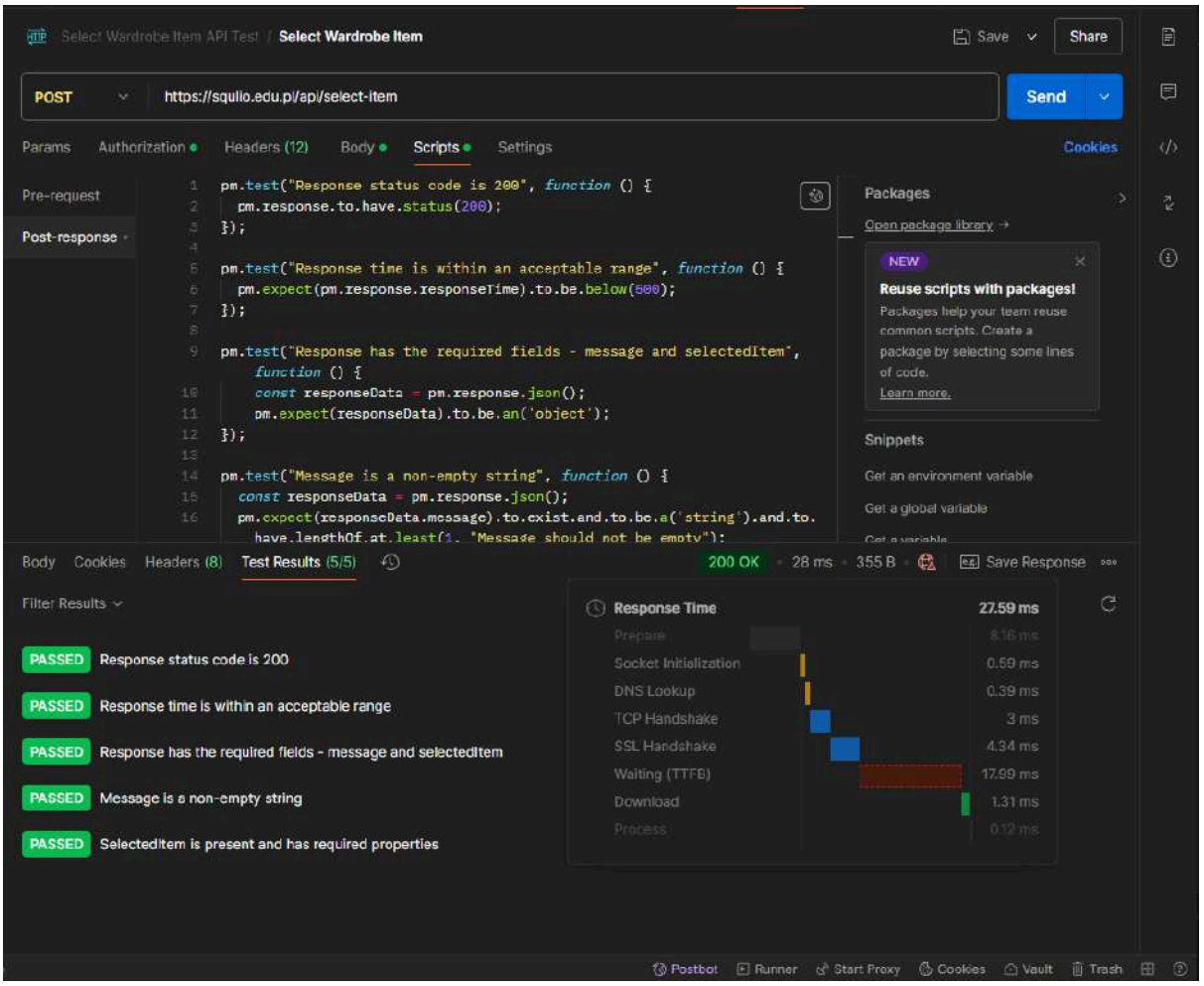
Zwrócone wartości:

```
{
    "message": "Pomyślnie wybrano przedmiot: hat"
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});

pm.test("Response has the required fields - message and selectedItem", function ()
{
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');
});

pm.test("Message is a non-empty string", function () {
  const responseData = pm.response.json();

pm.expect(responseData.message).to.exist.and.to.be.a('string').and.to.have.lengthOf
.at.least(1, "Message should not be empty");
});

pm.test("SelectedItem is present and has required properties", function () {
    const responseData = pm.response.json();


    if (responseData.selectedItem) {
        pm.expect(responseData.selectedItem).to.be.an('object');

pm.expect(responseData.selectedItem).to.have.property('id').that.is.a('string').and
.not.empty;

pm.expect(responseData.selectedItem).to.have.property('type').that.is.a('string').a
nd.not.empty;

pm.expect(responseData.selectedItem).to.have.property('url').that.is.a('string').an
d.not.empty;

pm.expect(responseData.selectedItem).to.have.property('name').that.is.a('string').a
nd.not.empty;
    } else {
        console.error("selectedItem is missing from the response.");
    }
});
```

Wynik testu:



Rysunek 6.16 Wynik testu API: /api/select-item (źródło: opracowanie własne)

Testowane API:

```
router.get('/api/userstats', authenticateUser,
UserStatsController.getUserStats);
```

Przesłane dane w teście:
Żądanie typu GET służy wyłącznie do pobierania danych, dlatego nie wymaga przesyłania żadnych informacji na serwer.

Zwrócone wartości:

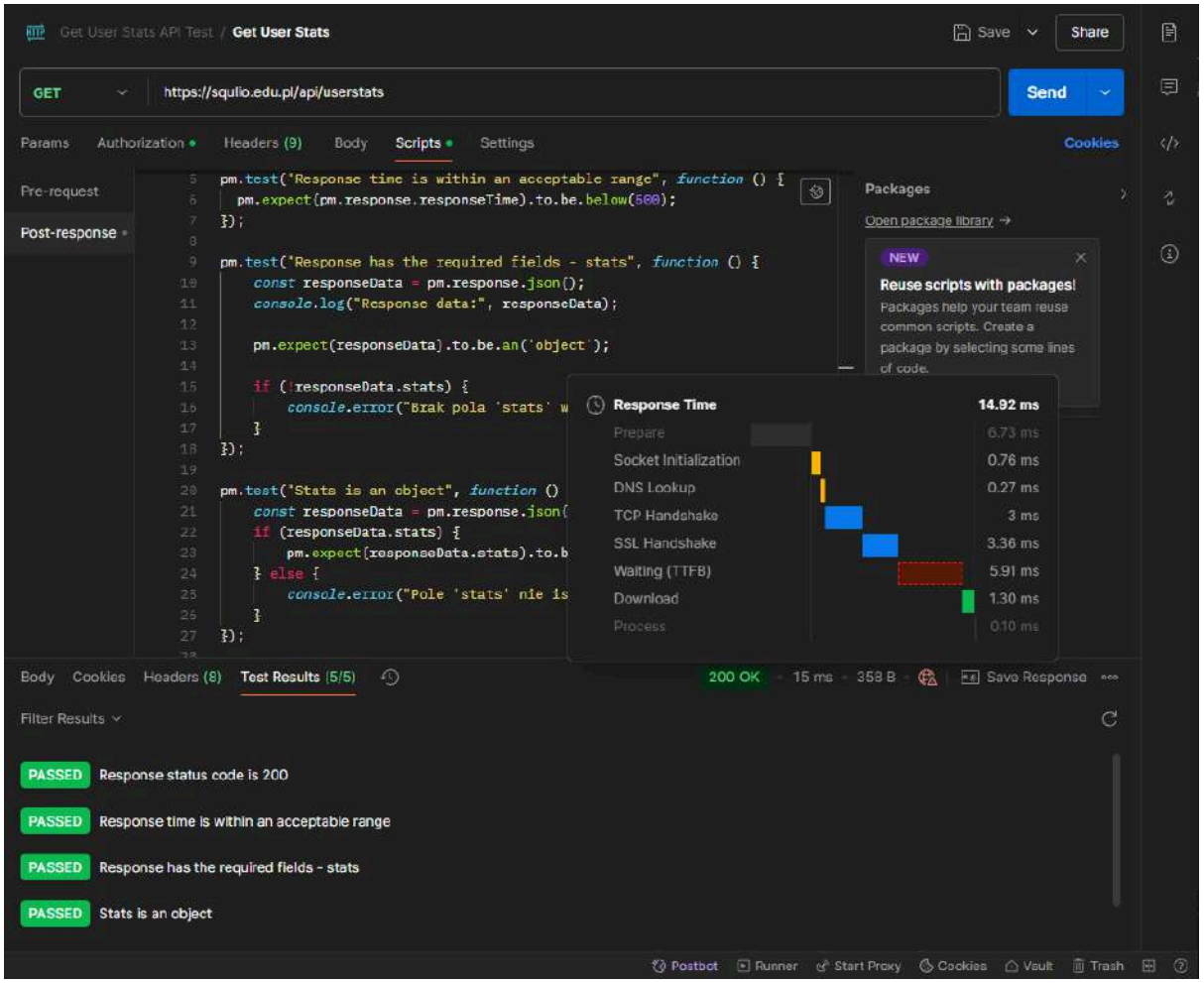
```
{
    "currentLevel": 10,
    "requiredExpForNextLevel": 7688
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
```

```
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});

pm.test("Response has the required fields - stats", function () {
    const responseData = pm.response.json();
    console.log("Response data:", responseData);

    pm.expect(responseData).to.be.an('object');

    if (!responseData.stats) {
        console.error("Brak pola 'stats' w odpowiedzi API!");
    }
});

pm.test("Stats is an object", function () {
    const responseData = pm.response.json();
    if (responseData.stats) {
        pm.expect(responseData.stats).to.be.an('object');
    } else {
        console.error("Pole 'stats' nie istnieje, test pominięty.");
    }
});

pm.test("Stats object has required properties", function () {
    const responseData = pm.response.json();
    if (responseData.stats) {

pm.expect(responseData.stats).to.have.property('level').that.is.a('number');

pm.expect(responseData.stats).to.have.property('experience').that.is.a('number');

pm.expect(responseData.stats).to.have.property('coins').that.is.a('number');
    } else {
        console.error("Pole 'stats' nie istnieje, test pominięty.");
    }
});
```

Wyniki testów:



Rysunek 6.17 Wynik testu API: /api/userstats (źródło: opracowanie własne)

Testowane API:

```
router.get('/api/get-tasks', authenticateUser,
TasksController.getTasks);
```

Przesłane dane w teście:
Żądanie typu GET służy wyłącznie do pobierania danych, dlatego nie wymaga przesyłania żadnych informacji na serwer.

Zwrócone wartości:

```
{
    "message": "Pomyślnie pobrano zadania użytkownika",
    "tasks": [
        {
            "id_task": 1,
            "title": "Początki bywają trudne.",
            "task": "Wykonaj poprawnie po jednym zadaniu z każdej kategorii",
            "possible_progress": 4,
            "user_progress": 2,
```

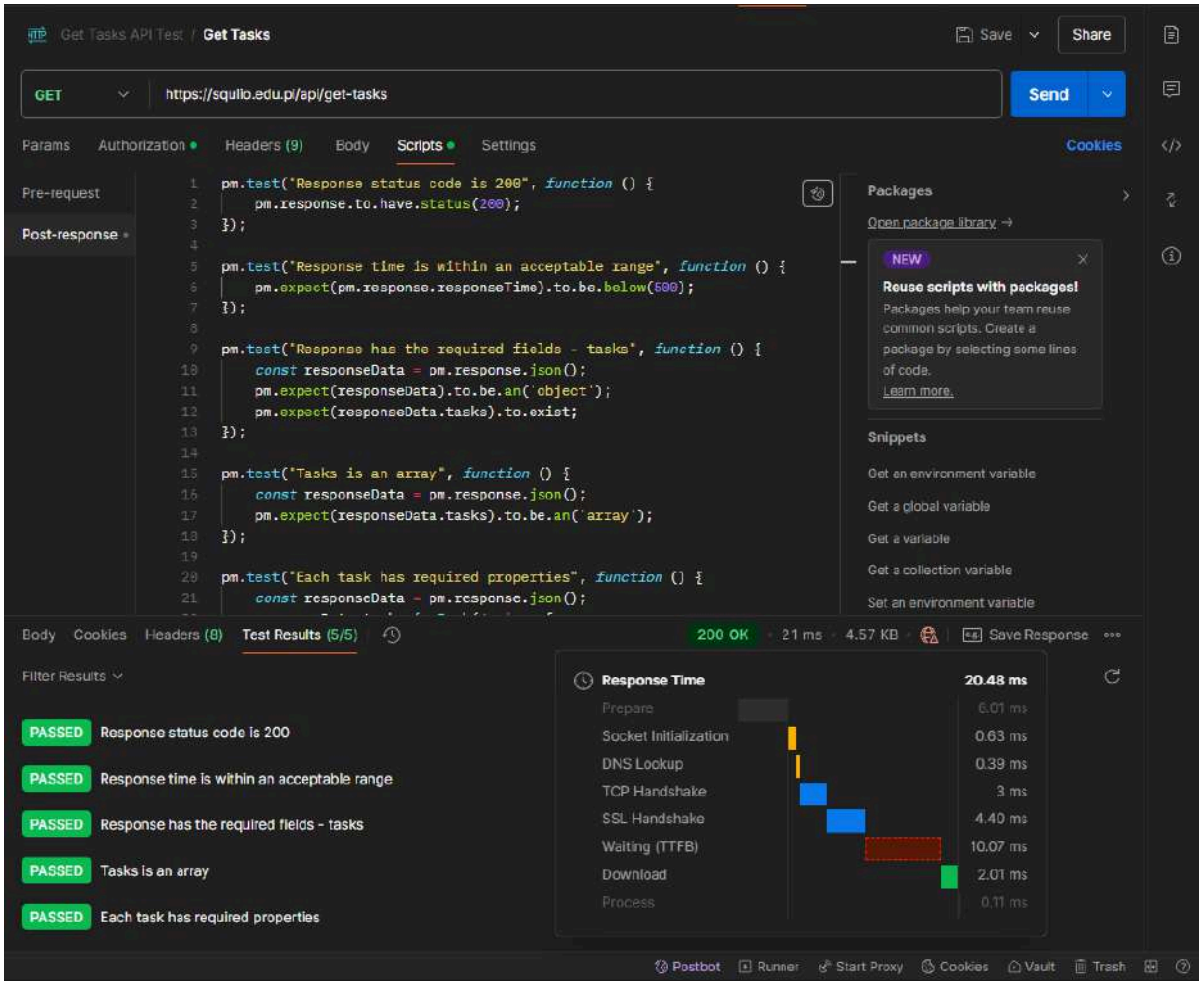
```
            "date": "2025-01-01T10:00:00.000Z"
        },
        {
            "id_task": 2,
            "title": "W stronę gwiazd.",
            "task": "Wyjdz z podziemi w wybranej kategorii",
            "possible_progress": 1,
            "user_progress": 0,
            "date": "2025-02-15T12:30:00.000Z"
        }
```

Wszystkie dostępne zadania znajdujące się w bazie danych zostały poprawnie zwrócone. Ze względu na obszerność odpowiedzi, nie przedstawiono ich w całości.

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is within an acceptable range", function () {
    pm.expect(pm.response.responseTime).to.be.below(500);
});

pm.test("Response has the required fields - tasks", function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.tasks).to.exist;
});

pm.test("Tasks is an array", function () {
    const responseData = pm.response.json();
    pm.expect(responseData.tasks).to.be.an('array');
});

pm.test("Each task has required properties", function () {
    const responseData = pm.response.json();
    responseData.tasks.forEach(task => {

pm.expect(task).to.have.property('id_task').that.is.a('number').and.above(0);

pm.expect(task).to.have.property('title').that.is.a('string').and.not.empty;
        pm.expect(task).to.have.property('task').that.is.a('string').and.not.empty;
    });
});
```

Wyniki testu:



Rysunek 6.18 Wynik testu API: /api/get-tasks (źródło: opracowanie własne)

Testowane API:

```
router.get('/api/get-achievements', authenticateUser,
AchievementsController.getAchievements);
```

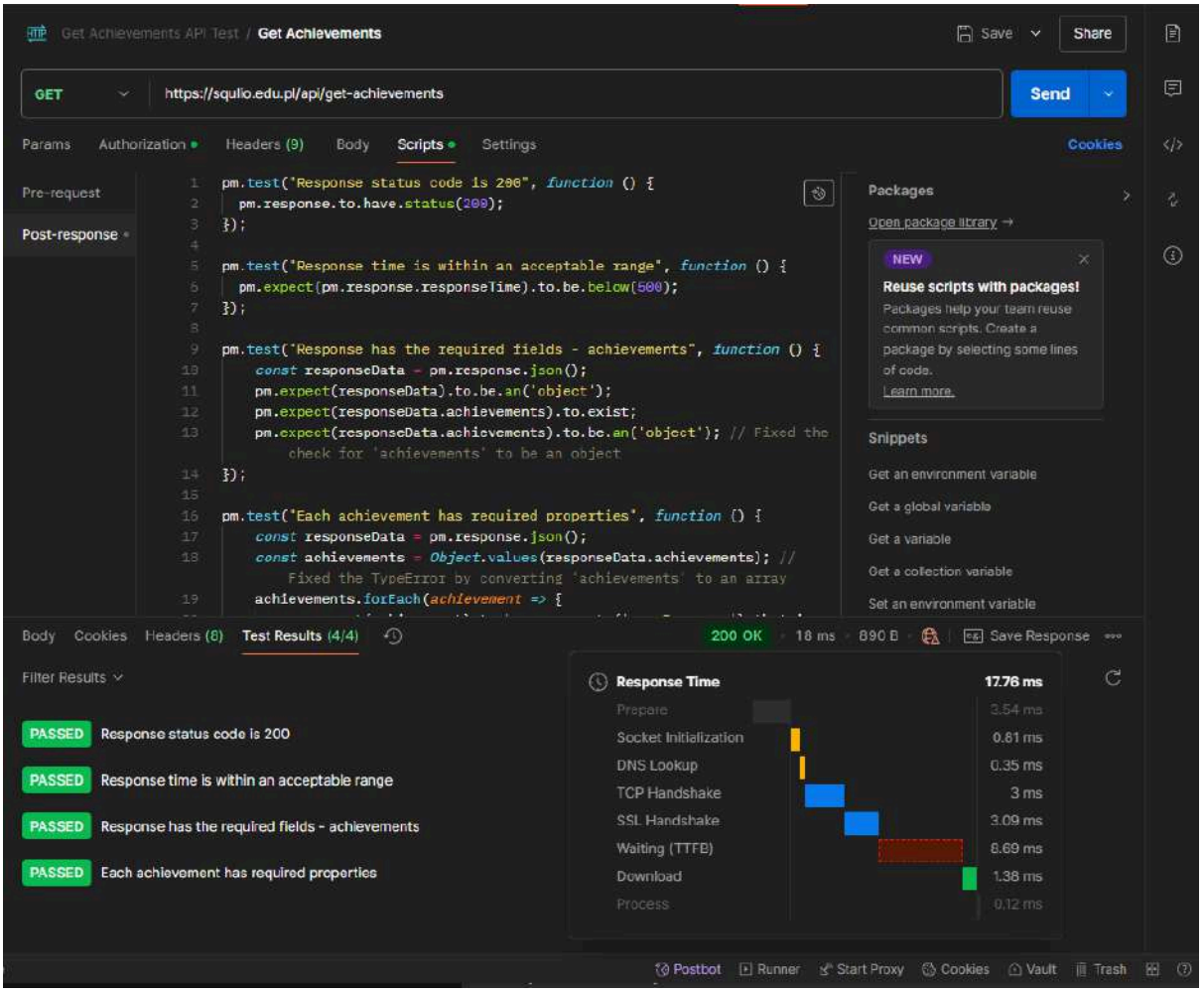
Przesłane dane w teście:
Żądanie typu GET służy wyłącznie do pobierania danych, dlatego nie wymaga przesyłania żadnych informacji na serwer.

Zwrócone wartości:

```
{
    "message": "Pomyślnie pobrano osiągnięcia użytkownika",
    "achievements": {
        "Poprawne odpowiedzi z Matematyki": {
            "userProgress": 3,
            "currentLevel": 0,
            "nextLevelRequirement": 50,
            "progressToNextLevel": 47
        },
```

```
        "Poprawne odpowiedzi z Języka Angielskiego": {
            "userProgress": 0,
            "currentLevel": 0,
            "nextLevelRequirement": 50,
            "progressToNextLevel": 50
        },
        "Poprawne odpowiedzi z Programowania": {
            "userProgress": 600,
            "currentLevel": 4,
            "nextLevelRequirement": 1000,
            "progressToNextLevel": 400
        },
        "Poprawne odpowiedzi z Przyrody": {
            "userProgress": 0,
            "currentLevel": 0,
            "nextLevelRequirement": 50,
            "progressToNextLevel": 50
        }
    }
}
```

Wyniki testu:



Rysunek 6.19 Wynik testu API: /api/get-achievements (źródło: opracowanie własne)

Testowane API:

```
router.post('/get-all-users', GetCharacterController.getAllUsers);
```

Przesłane dane w teście:
W tym żądaniu nie są wysyłane żadne dane

Zwrócone wartości:


```
[
    {
        "_id": "67bf971237b9cd96df1eb97a",
        "email": "mcieslak@edu.cdv.pl",
        "name": "mcieslak"
    },
    {
        "_id": "67bf9ac749ddf0cd7c931ba1",
        "email": "dawid.rynkiewicz1@wp.pl",
        "name": "Dejvid"
    },
```

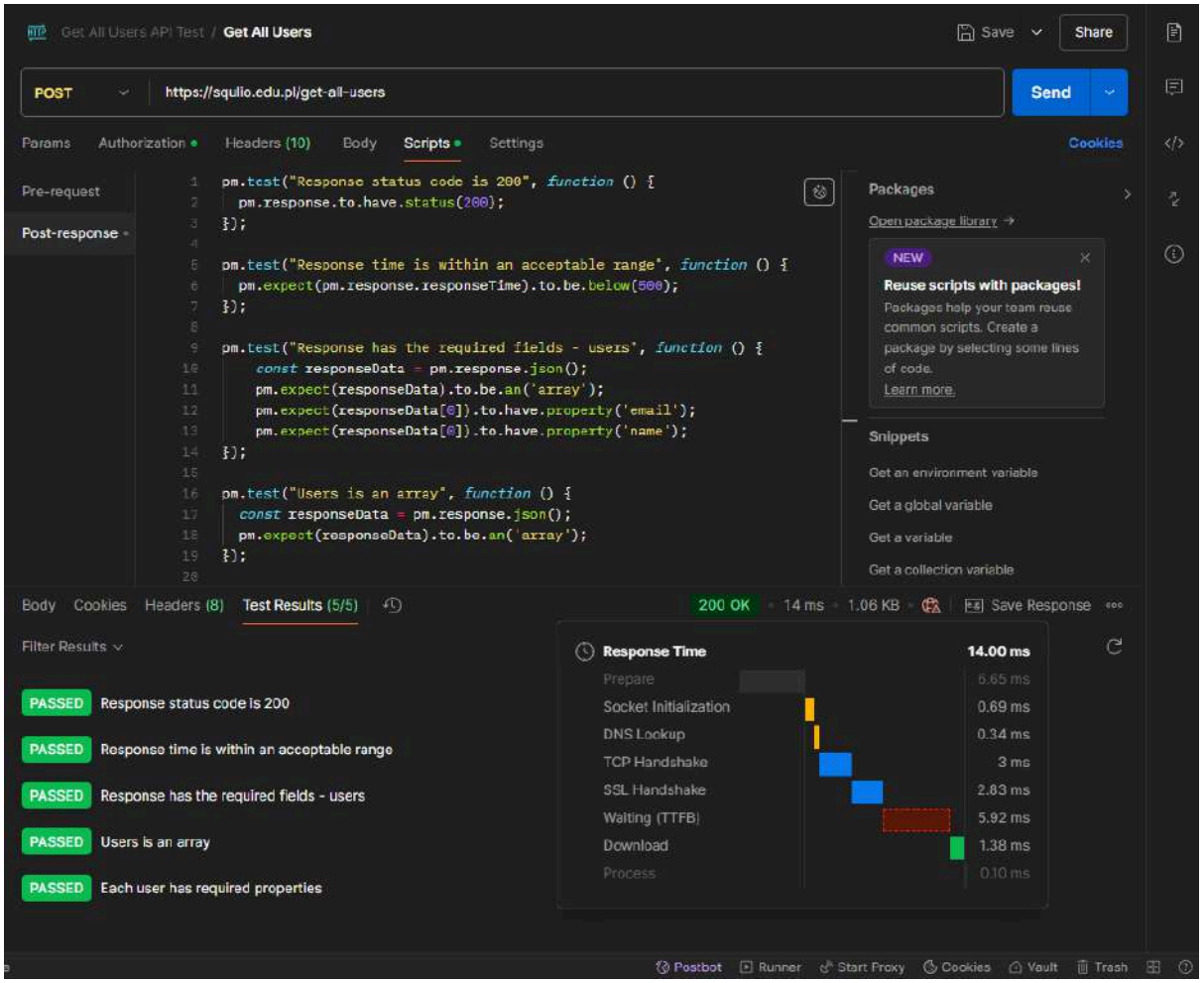
```
    {
        "_id": "67bf9e6b49ddf0cd7c932c84",
        "email": "mikolajek518@gmail.com",
        "name": "mikolajek518"
    },
    {
        "_id": "67bfaaebe8da34c5efde725d",
        "email": "zelaznypl@gmail.com",
        "name": "zelaznypl"
    },
    {
        "_id": "67bfc364e8da34c5efde7472",
        "email": "marcinratajczyk18@gmail.com",
        "name": "Moros_King"
    },
    {
        "_id": "67c21ff63da9faea13d7617a",
        "email": "dawid.sekulaa@gmail.com",
        "name": "JDD"
    },
    {
        "_id": "67c3d7a791ff9864edf2a3ae",
        "email": "dawid.sekula.hixon@gmail.com",
        "name": "12345"
    },
    {
        "_id": "67c59a9671e2c2df3d5bf31e",
        "email": "wojtek.pestka@onet.pl",
        "name": "wojtek"
    },
    {
        "_id": "67c6e8fb8f1ee8e132455b7e",
        "email": "dominikpiszczek2001@gmail.com",
        "name": "test2"
    }
]
```


Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});
```

```
pm.test("Response has the required fields - users", function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('array');
    pm.expect(responseData[0]).to.have.property('email');
    pm.expect(responseData[0]).to.have.property('name');
});

pm.test("Users is an array", function () {
  const responseData = pm.response.json();
  pm.expect(responseData).to.be.an('array');
});

pm.test("Each user has required properties", function () {
    const responseData = pm.response.json();
    responseData.forEach(user => {
        pm.expect(user).to.have.property('_id').that.is.a('string').and.not.empty;

pm.expect(user).to.have.property('email').that.is.a('string').and.not.empty;
    });
});
```

Wynik testu:



Rysunek 6.20 Wynik testu API: /get-all-users (źródło: opracowanie własne)

Testowane API:

```
router.get('/get-user-data/:id',
GetCharacterController.getAllUserData);
```

Przesłane dane w teście:
Żądanie typu GET służy wyłącznie do pobierania danych, dlatego nie wymaga przesyłania żadnych informacji na serwer.

Zwrócone wartości:


```
{
    "_id": "67bf971237b9cd96df1eb97a",
    "username": "mcieslak",
    "email": "mcieslak@edu.cdv.pl",
    "type": "gracz",
    "activationKey": "771352",
    "userStatus": "enabled",
    "head": "/head/neutral/h1_orange.png",
    "answeredQuestions": [],
```

```
    "completedAllQuestionsToday": false,
    "lastCompletedDate": "2025-03-02T23:00:00.000Z",
    "englishQuestionElo": 1000,
    "mathsQuestionElo": 1000,
    "scienceQuestionElo": 1000,
    "programmingQuestionElo": 1000,
    "correct_programmingQuestion": 0,
    "correct_mathsQuestion": 0,
    "correct_englishQuestion": 0,
    "correct_scienceQuestion": 0,
    "level": 14,
    "coins": 23415,
    "extracoins": 1,
    "happines": 2,
    "exp": 0,
    "hp": 5,
    "hpLimit": 5,
    "lastHpLoss": null,
    "items": [
        1,
        2,
        3,
        5,
        6,
        7,
        8,
        9,
        10,
        11,
        12,
        13,
        14,
        15,
        16,
        17,
        18,
        19,
        20,
        21,
        22,
        23,
        24,
        25
    ],
    "createdAt": "2025-02-26T22:34:58.802Z",
```

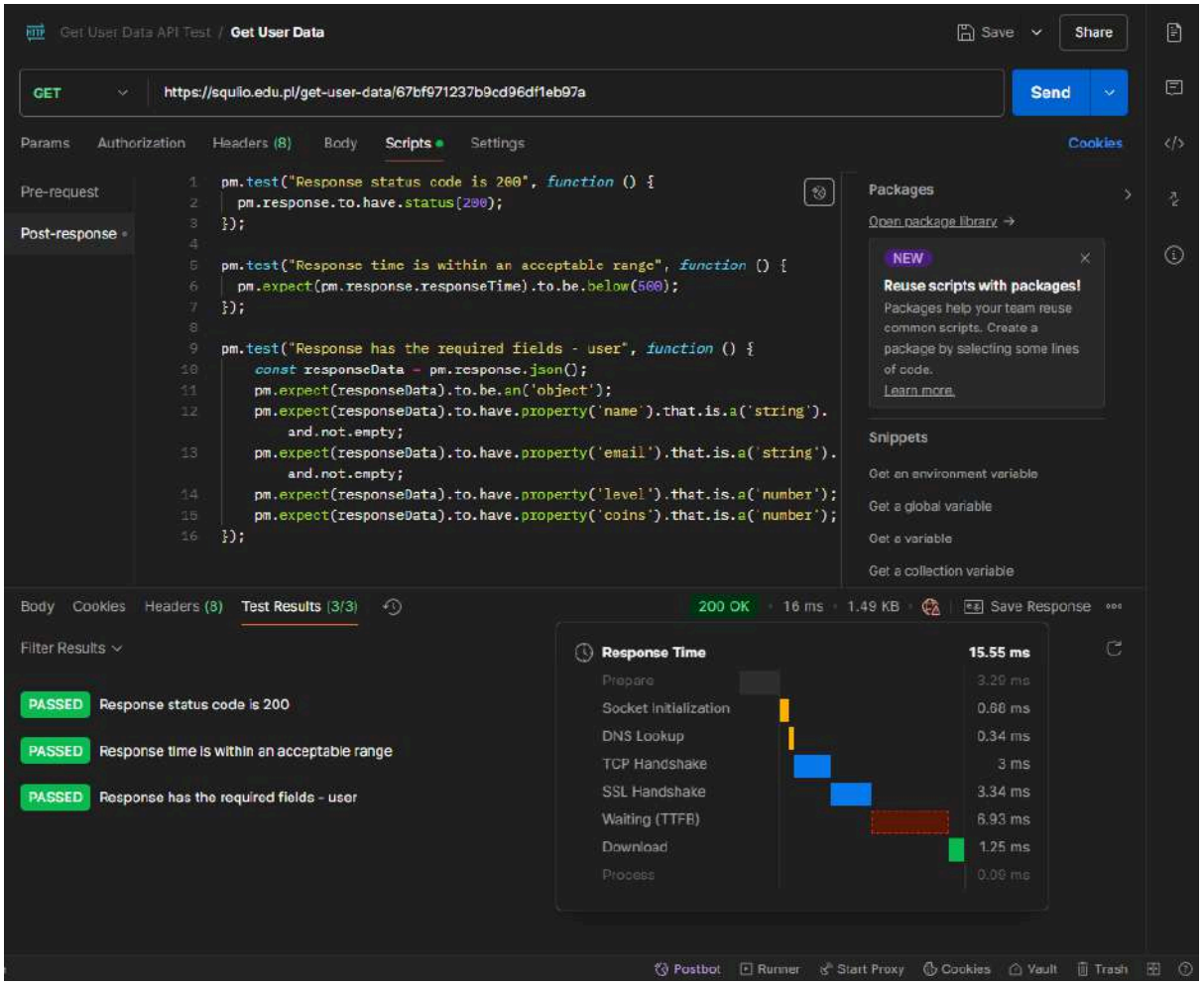
```
    "tasks_progress": [
        {
            "id_1": 0,
            "id_2": 0,
            "id_6": 0,
            "id_7": 0,
            "id_8": 0,
            "id_9": 0,
            "id_11": 0,
            "id_12": 0,
            "id_13": 0,
            "id_15": 0,
            "id_16": 0,
            "id_17": 0,
            "id_19": 0,
            "id_20": 0,
            "id_21": 0,
            "id_22": 0,
            "id_23": 0,
            "_id": "67c7b47cb3ff49331b728770",
            "id_25": 0,
            "id_10": 2,
            "id_4": 1,
            "id_24": 1,
            "id_18": 0,
            "id_14": 0,
            "id_3": 1,
            "id_5": 1
        }
    ],
    "__v": 25,
    "body": "/body/b1_orange.png",
    "legs": "/legs/l1_orange.png",
    "name": "mcieslak",
    "headSkin": "16",
    "lastLogin": "2025-03-03T01:17:38.691Z",
    "legSkin": null,
    "extracoinsgiven": [],
    "editableCharacter": false
}
```


Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});
```

```
pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});

pm.test("Response has the required fields - user", function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');

pm.expect(responseData).to.have.property('name').that.is.a('string').and.not.empty;

pm.expect(responseData).to.have.property('email').that.is.a('string').and.not.empty
;
    pm.expect(responseData).to.have.property('level').that.is.a('number');
    pm.expect(responseData).to.have.property('coins').that.is.a('number');
});
```

Wynik testu:



Rysunek 6.21 Wynik testu API: /get-user-data/:id (źródło: opracowanie własne)

Testowane API:

```
router.put('/update-user/:id', GetCharacterController.updateUserData);
```

Przesłane dane w teście:

```
{
    "name": "Updated Name",
    "email": "updated.email@example.com"
}
```

Zwrócone wartości:


```
{
    "message": "Dane użytkownika zostały zaktualizowane.",
    "user": {
        "_id": "67bf971237b9cd96df1eb97a",
        "username": "mcieslak",
        "email": "updated.email@example.com",
        "type": "gracz",
        "activationKey": "771352",
        "userStatus": "enabled",
        "head": "/head/neutral/h1_orange.png",
        "answeredQuestions": [],
        "completedAllQuestionsToday": false,
        "lastCompletedDate": "2025-03-02T23:00:00.000Z",
        "englishQuestionElo": 1000,
        "mathsQuestionElo": 1000,
        "scienceQuestionElo": 1000,
        "programmingQuestionElo": 1000,
        "correct_programmingQuestion": 0,
        "correct_mathsQuestion": 0,
        "correct_englishQuestion": 0,
        "correct_scienceQuestion": 0,
        "level": 14,
        "coins": 23415,
        "extracoins": 1,
        "happines": 2,
        "exp": 0,
        "hp": 5,
        "hpLimit": 5,
        "lastHpLoss": null,
        "items": [
            1,
            2,
            3,
            5,
            6,
            7,
```

```
            8,
            9,
            10,
            11,
            12,
            13,
            14,
            15,
            16,
            17,
            18,
            19,
            20,
            21,
            22,
            23,
            24,
            25
        ],
        "createdAt": "2025-02-26T22:34:58.802Z",
        "tasks_progress": [
            {
                "id_1": 0,
                "id_2": 0,
                "id_6": 0,
                "id_7": 0,
                "id_8": 0,
                "id_9": 0,
                "id_11": 0,
                "id_12": 0,
                "id_13": 0,
                "id_15": 0,
                "id_16": 0,
                "id_17": 0,
                "id_19": 0,
                "id_20": 0,
                "id_21": 0,
                "id_22": 0,
                "id_23": 0,
                "_id": "67c7b605b3ff49331b72879a",
                "id_25": 0,
                "id_10": 2,
                "id_4": 1,
                "id_24": 1,
                "id_18": 0,
```

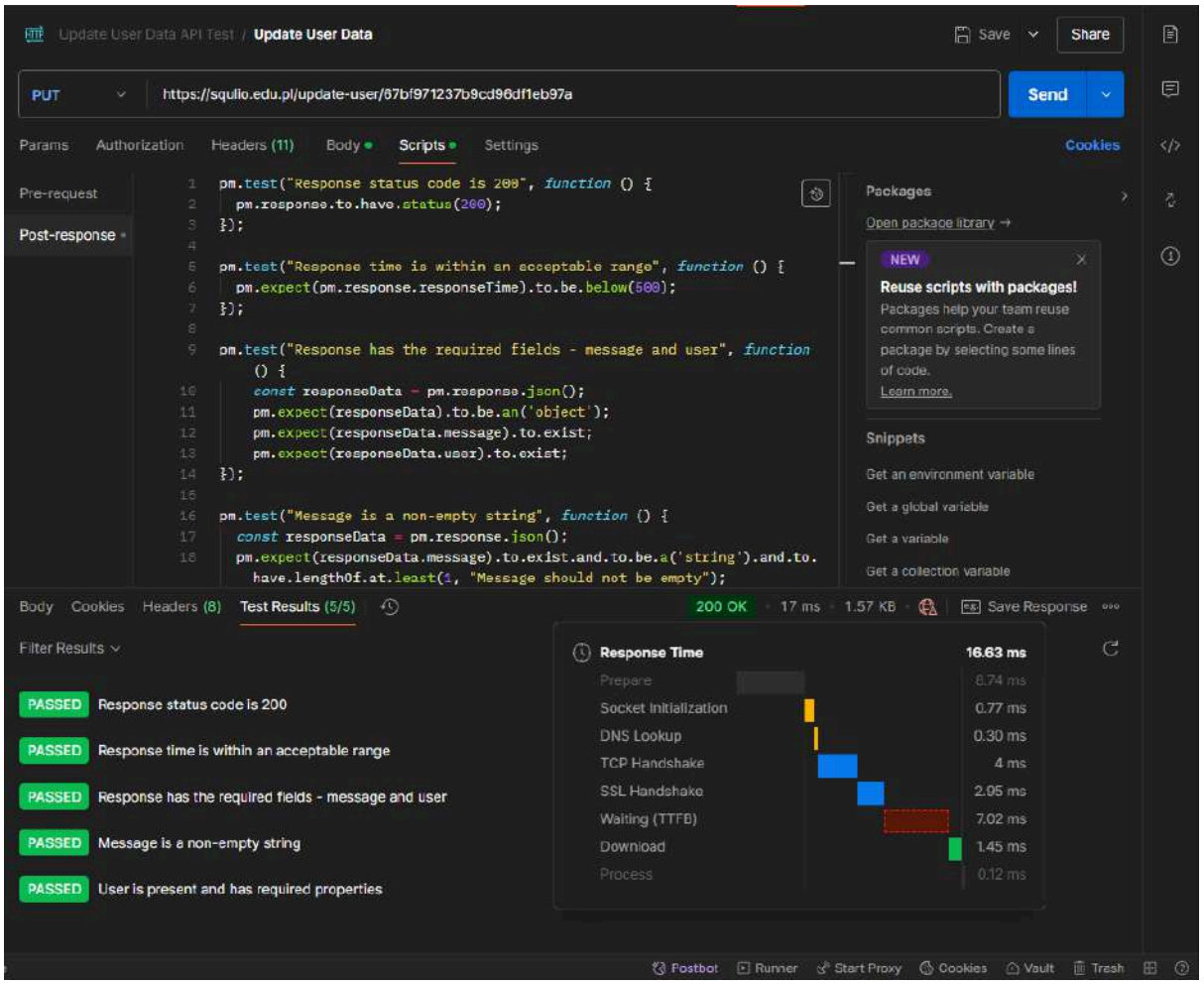
```
                "id_14": 0,
                "id_3": 1,
                "id_5": 1
            }
        ],
        "__v": 25,
        "body": "/body/b1_orange.png",
        "legs": "/legs/l1_orange.png",
        "name": "Updated Name",
        "headSkin": "16",
        "lastLogin": "2025-03-03T01:17:38.691Z",
        "legSkin": null,
        "extracoinsgiven": [],
        "editableCharacter": false
    }
}
```


Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});

pm.test("Response has the required fields - message and user", function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.message).to.exist;
    pm.expect(responseData.user).to.exist;
});

pm.test("Message is a non-empty string", function () {
  const responseData = pm.response.json();

pm.expect(responseData.message).to.exist.and.to.be.a('string').and.to.have.lengthOf
.at.least(1, "Message should not be empty");
});

pm.test("User is present and has required properties", function () {
    const responseData = pm.response.json();
    pm.expect(responseData.user).to.be.an('object');

pm.expect(responseData.user).to.have.property('_id').that.is.a('string').and.not.em
pty;
```

```
pm.expect(responseData.user).to.have.property('name').that.is.a('string').and.not.e
mpty;

pm.expect(responseData.user).to.have.property('email').that.is.a('string').and.not.
empty;
});
```

Wyniki testu:



Rysunek 6.22 Wynik testu API: /update-user/:id (źródło: opracowanie własne)

Testowane API:

```
router.get('/api/stats/:date', DailyStatsController.getDailyStats);
```

Przesłane dane w teście:
Żądanie typu GET służy wyłącznie do pobierania danych, dlatego nie wymaga przesyłania żadnych informacji na serwer.

Zwrócone wartości:

```
{
    "avgEloScores": {
```

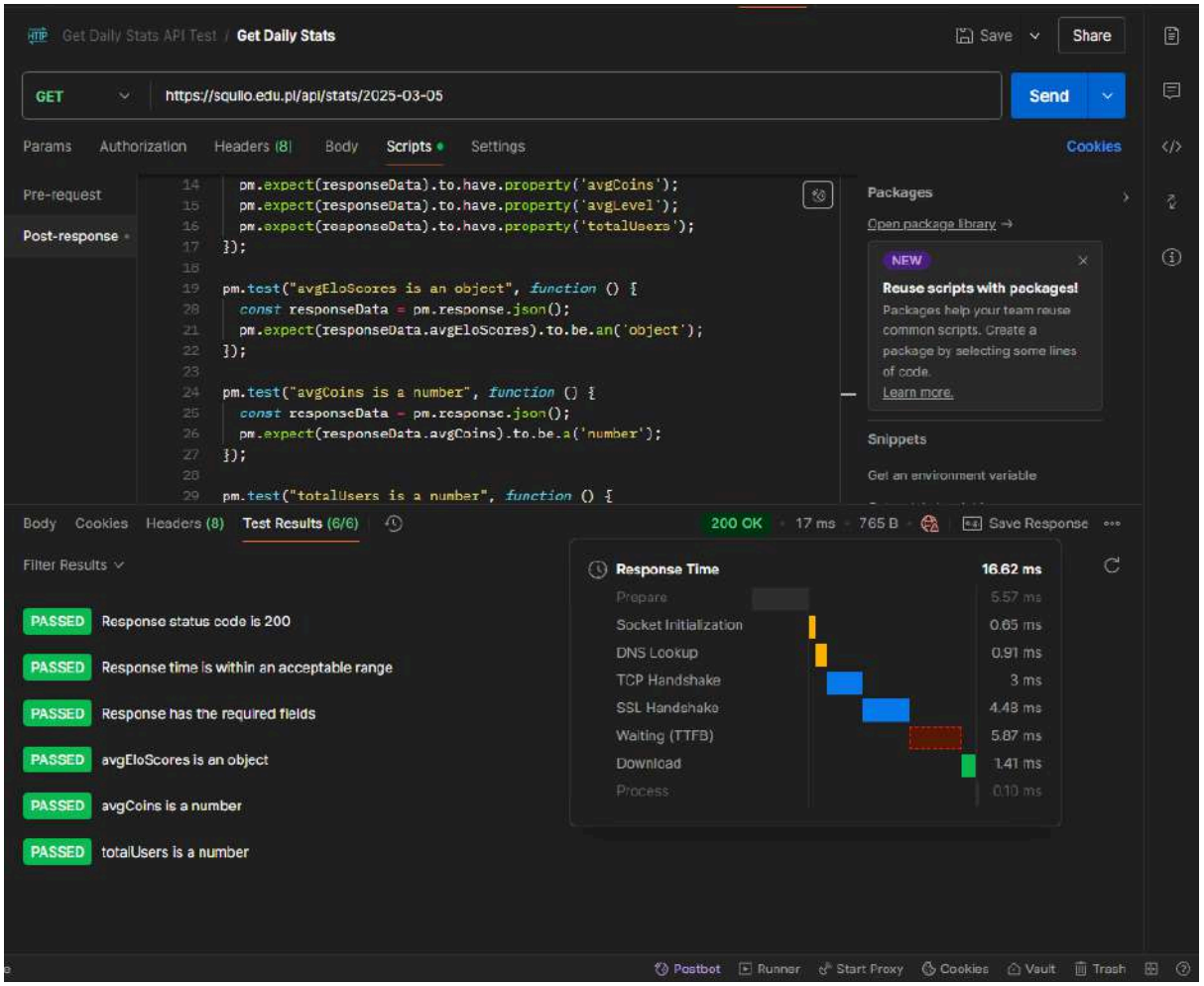
```
        "english": 1740,
        "maths": 2193.3333333333335,
        "science": 1800,
        "programming": 2793.3333333333335
    },
    "totalCorrectAnswers": {
        "programming": 768,
        "maths": 138,
        "english": 56,
        "science": 58
    },
    "_id": "67c785f063438eda661d5cb7",
    "date": "2025-03-05T00:00:00.000Z",
    "__v": 0,
    "avgCoins": 2148.266666666667,
    "avgLevel": 3.8,
    "completedUsers": 1,
    "completionRate": 6.666666666666667,
    "createdAt": "2025-03-04T23:00:00.826Z",
    "totalUsers": 15,
    "updatedAt": "2025-03-04T23:00:00.826Z"
}
```


Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});

pm.test("Response has the required fields", function () {
  const responseData = pm.response.json();
  pm.expect(responseData).to.be.an('object');
  pm.expect(responseData).to.have.property('avgEloScores');
  pm.expect(responseData).to.have.property('totalCorrectAnswers');
  pm.expect(responseData).to.have.property('avgCoins');
  pm.expect(responseData).to.have.property('avgLevel');
  pm.expect(responseData).to.have.property('totalUsers');
});

pm.test("avgEloScores is an object", function () {
  const responseData = pm.response.json();
  pm.expect(responseData.avgEloScores).to.be.an('object');
});
```

```
pm.test("avgCoins is a number", function () {
  const responseData = pm.response.json();
  pm.expect(responseData.avgCoins).to.be.a('number');
});

pm.test("totalUsers is a number", function () {
  const responseData = pm.response.json();
  pm.expect(responseData.totalUsers).to.be.a('number');
});
```

Wyniki testu:



Rysunek 6.23 Wynik testu API: /api/stats/:date (źródło: opracowanie własne)

Testowane API:

```
router.get('/api/stats/compare/:date1/:date2',
DailyStatsController.compareDailyStats);
```

Przesłane dane w teście:
Żądanie typu GET służy wyłącznie do pobierania danych, dlatego nie wymaga przesyłania żadnych informacji na serwer.

Zwrócone wartości:

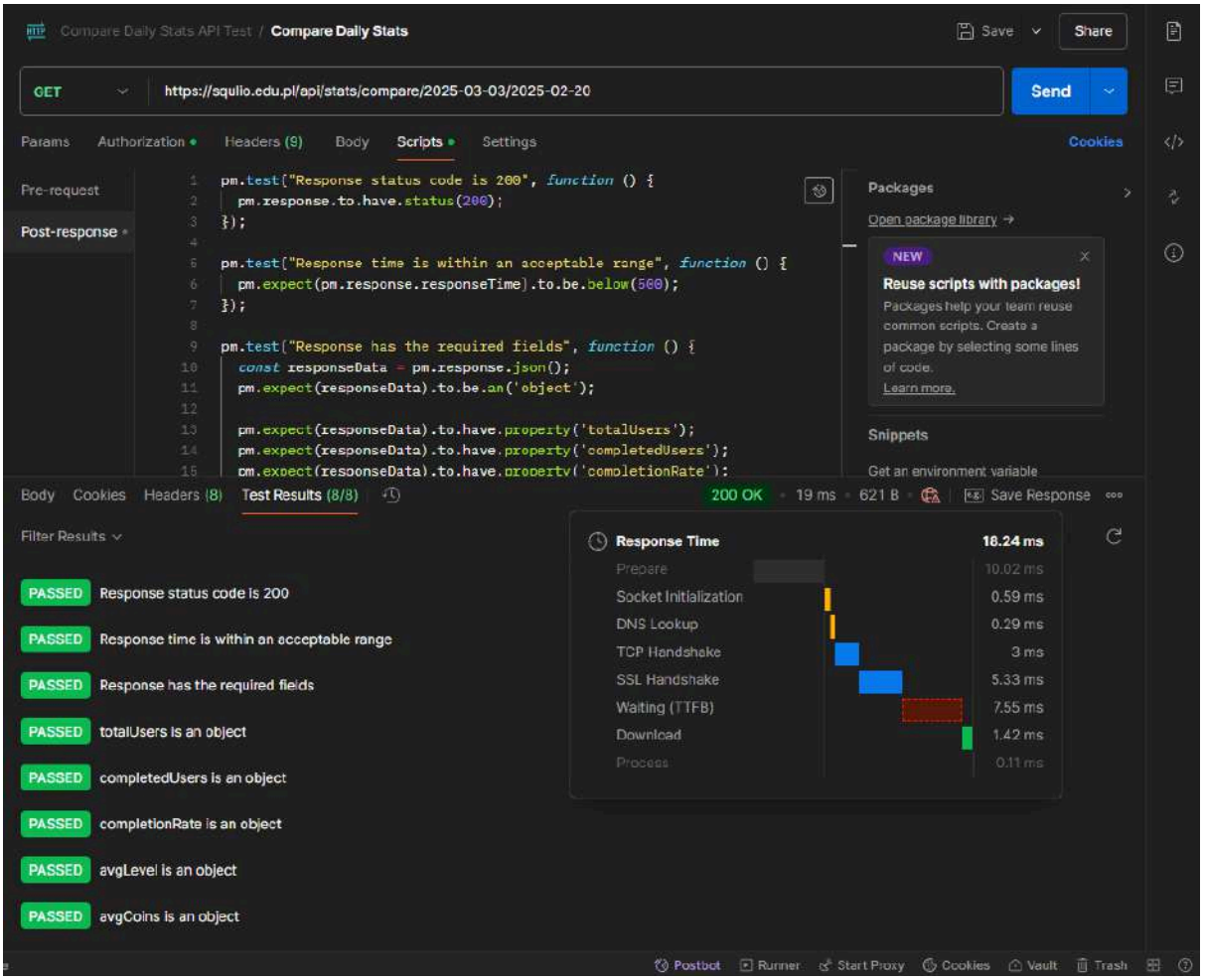
```
{
    "totalUsers": {
        "2025-03-03": 9,
        "2025-02-20": 6
    },
    "completedUsers": {
        "2025-03-03": 3,
        "2025-02-20": 2
    },
    "completionRate": {
        "2025-03-03": 33.33333333333333,
        "2025-02-20": 33.33333333333333
    },
    "avgLevel": {
        "2025-03-03": 4.333333333333333,
        "2025-02-20": 5
    },
    "avgCoins": {
        "2025-03-03": 2683.3333333333335,
        "2025-02-20": 6724.333333333333
    }
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});

pm.test("Response has the required fields", function () {
  const responseData = pm.response.json();
  pm.expect(responseData).to.be.an('object');

  pm.expect(responseData).to.have.property('totalUsers');
  pm.expect(responseData).to.have.property('completedUsers');
  pm.expect(responseData).to.have.property('completionRate');
  pm.expect(responseData).to.have.property('avgLevel');
  pm.expect(responseData).to.have.property('avgCoins');
```

```
});

pm.test("totalUsers is an object", function () {
  const responseData = pm.response.json();
  pm.expect(responseData.totalUsers).to.be.an('object');
});

pm.test("completedUsers is an object", function () {
  const responseData = pm.response.json();
  pm.expect(responseData.completedUsers).to.be.an('object');
});

pm.test("completionRate is an object", function () {
  const responseData = pm.response.json();
  pm.expect(responseData.completionRate).to.be.an('object');
});

pm.test("avgLevel is an object", function () {
  const responseData = pm.response.json();
  pm.expect(responseData.avgLevel).to.be.an('object');
});

pm.test("avgCoins is an object", function () {
  const responseData = pm.response.json();
  pm.expect(responseData.avgCoins).to.be.an('object');
});
```

Wynik testu:



Rysunek 6.24 Wynik testu API: /api/stats/compare/:date1/:date2 (źródło: opracowanie własne)

Testowane API:

```
router.post('/api/send-reset-code',
ResetPasswordController.sendResetCode);
```

Przesłane dane w teście:

```
{
    "email": "wojtek.pestka@onet.pl"
}
```

Zwrócone wartości:

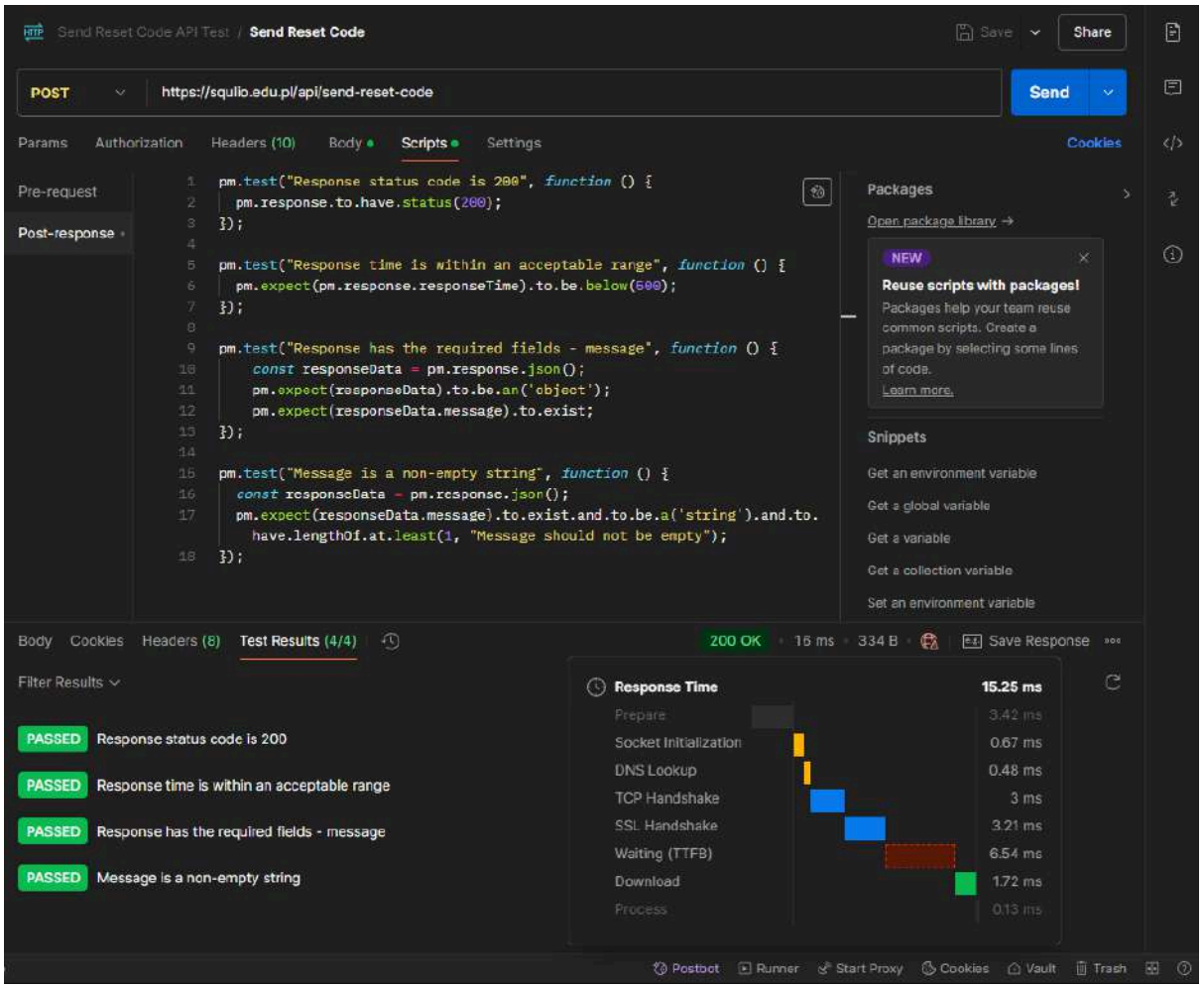
```
{
    "message": "Kod wysłany"
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
```

```
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});

pm.test("Response has the required fields - message", function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.message).to.exist;
});

pm.test("Message is a non-empty string", function () {
  const responseData = pm.response.json();

pm.expect(responseData.message).to.exist.and.to.be.a('string').and.to.have.lengthOf
.at.least(1, "Message should not be empty");
});
```

Wynik testu:



Rysunek 6.25 Wynik testu API: /api/send-reset-code (źródło: opracowanie własne)

Testowane API:

```
router.post('/api/reset-password',
ResetPasswordController.resetPassword);
```

Przesłane dane w teście:

```
{
    "email": "wojtek.pestka@onet.pl",
    "code": 276893,
    "password": "admin123"
}
```

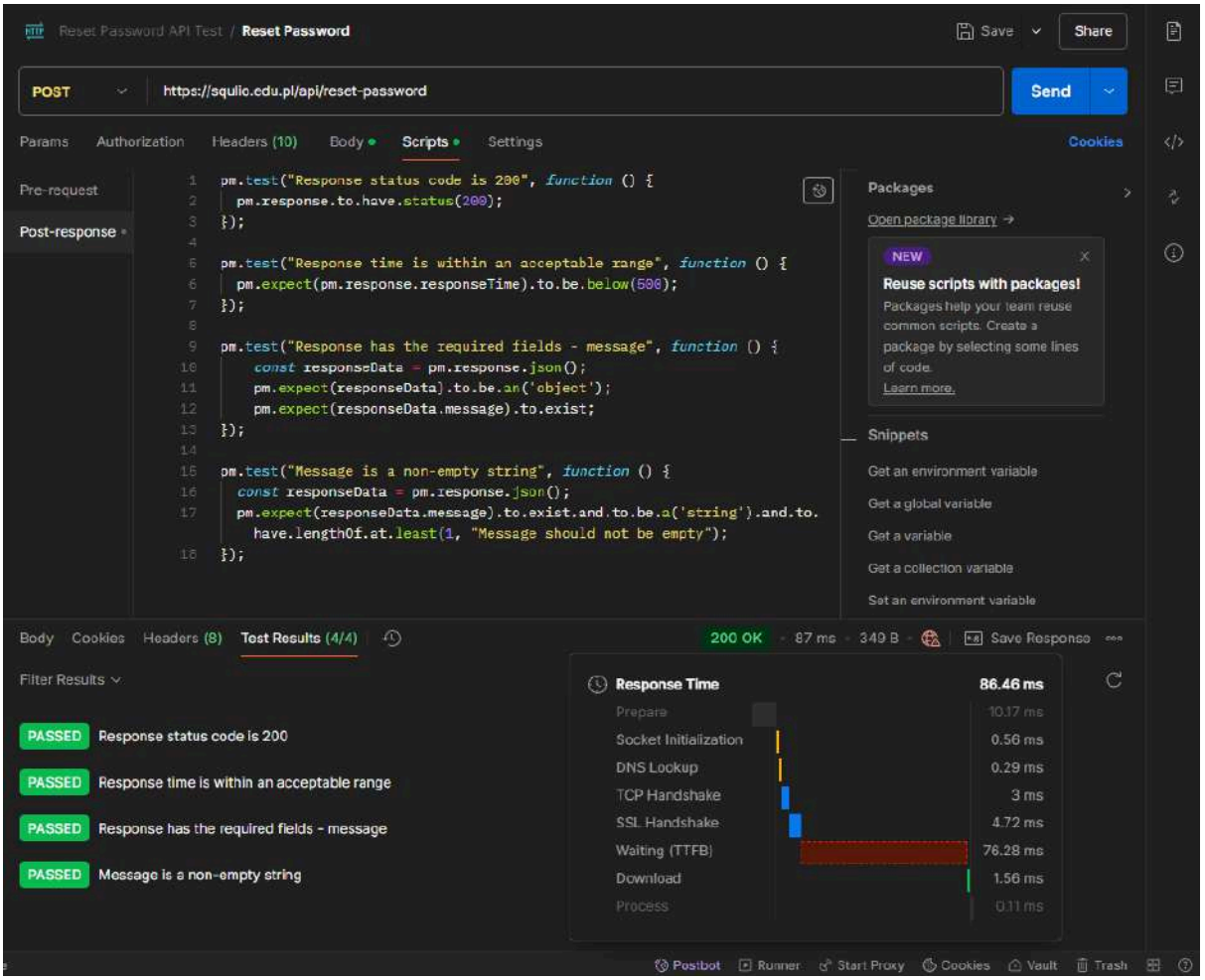
Zwrócone wartości:

```
{
    "message": "Hasło zostało zresetowane"
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});

pm.test("Response has the required fields - message", function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.message).to.exist;
});

pm.test("Message is a non-empty string", function () {
  const responseData = pm.response.json();

pm.expect(responseData.message).to.exist.and.to.be.a('string').and.to.have.lengthOf
.at.least(1, "Message should not be empty");
});
```

Wyniki testu:



Rysunek 6.26 Wynik testu API: /api/reset-password (źródło: opracowanie własne)

Testowane API:

```
router.post('/api/buy-heart', authenticateUser,
BuyHeartController.BuyHeart);
```

Przesłane dane w teście:
W tym żądaniu nie są wymagane żadne dane do przesłania

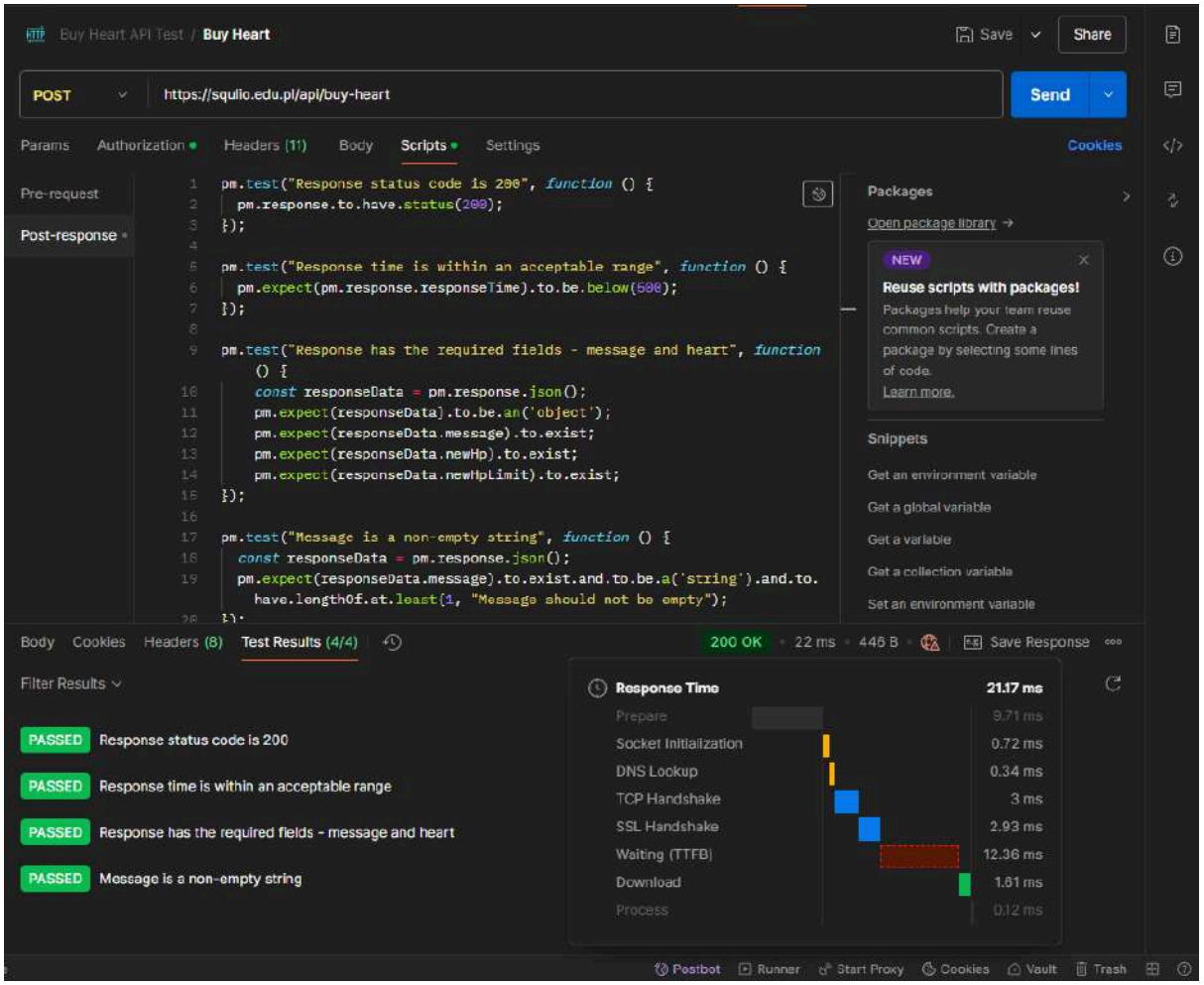
Zwrócone wartości:

```
{
    "message": "Zakupiono 1 serduszko! Twój maksymalny limit HP wzrósł.",
    "remainingExtraCoins": 40,
    "newHp": 9,
    "newHpLimit": 9,
    "success": true
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(500);
});

pm.test("Response has the required fields - message and heart", function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.message).to.exist;
    pm.expect(responseData.newHp).to.exist;
    pm.expect(responseData.newHpLimit).to.exist;
});

pm.test("Message is a non-empty string", function () {
  const responseData = pm.response.json();

pm.expect(responseData.message).to.exist.and.to.be.a('string').and.to.have.lengthOf
.at.least(1, "Message should not be empty");
});
```

Wynik testu:



Rysunek 6.27 Wynik testu API: /api/buy-heart (źródło: opracowanie własne)

Testowane API:

```
router.post('/api/get-heart-shop', authenticateUser,
BuyHeartController.GetHeartShop);
```

Przesłane dane w teście:
W tym żądaniu nie są wymagane żadne dane do przesłania

Zwrócone wartości:

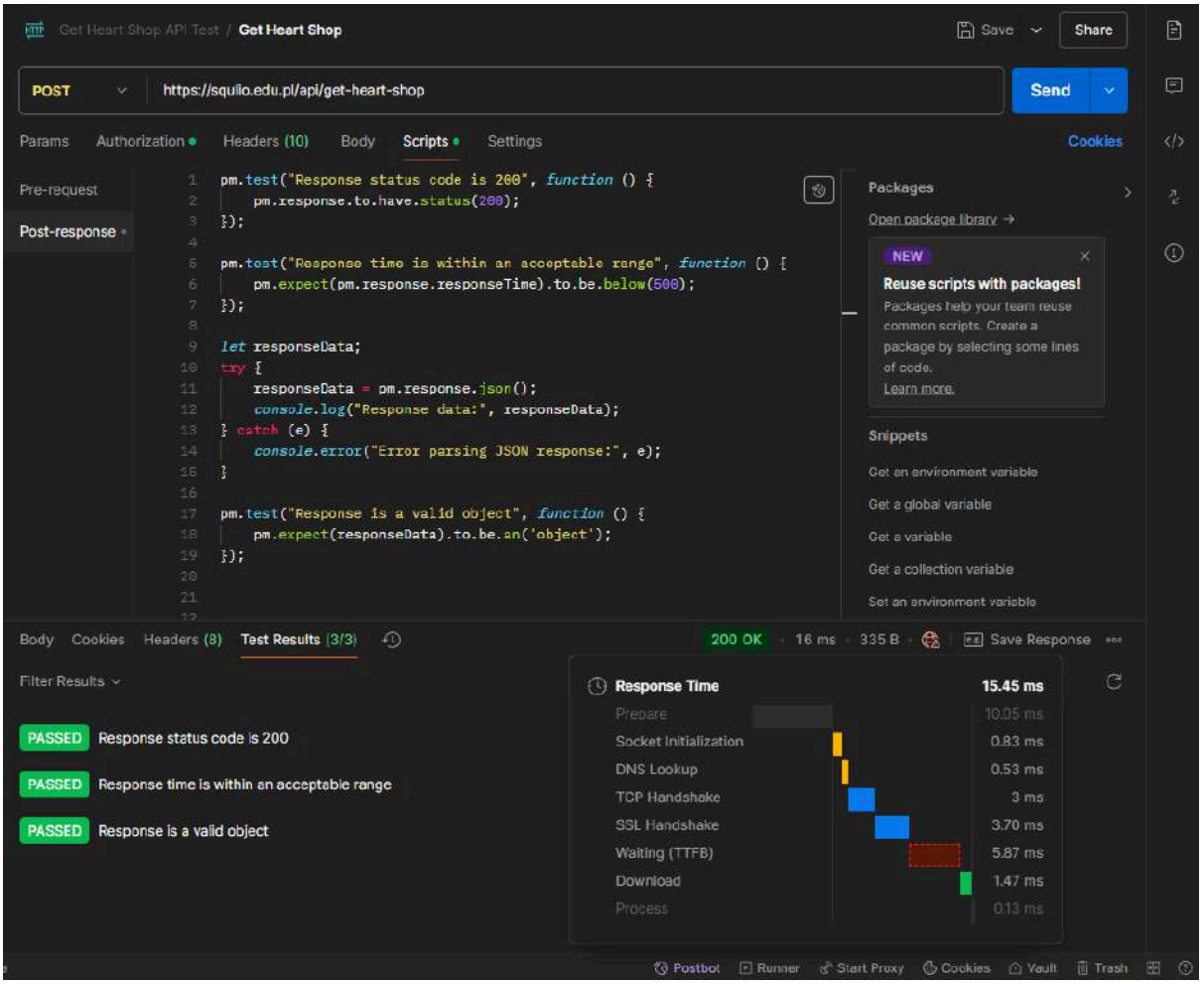
```
{
    "hpLimit": 9,
    "heartCost": 5
}
```

Skrypt użyty do testu:

```
pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});
```

```
pm.test("Response time is within an acceptable range", function () {
    pm.expect(pm.response.responseTime).to.be.below(500);
});

let responseData;
try {
    responseData = pm.response.json();
    console.log("Response data:", responseData);
} catch (e) {
    console.error("Error parsing JSON response:", e);
}

pm.test("Response is a valid object", function () {
    pm.expect(responseData).to.be.an('object');
});
```

Wyniki testu: 8



Rysunek 6.28 Wynik testu API: /api/get-heart-shop (źródło: opracowanie własne)

6.2 Podsumowanie testów Postman

6.2.1 Wprowadzenie

W trakcie testowania systemu poprzez aplikację Postman sprawdzona została poprawność i wydajność API wykorzystanego do realizacji projektu. Testy obejmowały endpointy, które odpowiedzialne były za logowanie, rejestrację, aktywację konta, zarządzanie postacią przez użytkownika oraz obsługę pytań, statystyk i sklepu.

6.2.2 Przeprowadzone testy

Przetestowano następujące funkcjonalności API:

- Rejestracja użytkownika — sprawdzenie procesu rejestracji i zwracanych wartości,
- Logowanie użytkownika — weryfikacja autoryzacji użytkownika i generowania tokenów,
- Aktywacja konta — testowanie procesu aktywacji konta (unikalny klucz),
- Zarządzanie postacią — zapis i pobieranie informacji o postaci,
- Obsługa pytań i odpowiedzi — pobieranie dziennych pytań, przesyłanie odpowiedzi oraz obliczanie wyników,
- Obsługa sklepu — pobieranie przedmiotów ze sklepu, zakup przedmiotów oraz zarządzanie ekwipunkiem,
- Obsługa statystyk — pobieranie i porównywanie statystyk użytkowników,
- Resetowanie hasła — testowanie kodu resetującego oraz zmiany hasła,
- Zakup dodatkowych funkcjonalności — sprawdzanie zakupu dodatkowych serduszek zwiększających limit HP.

6.2.3 Pominięte testy

Ze względu na występujące analogiczne endpointy, zdecydowano się pominąć następujące API:

- **Endpointy odpowiedzialne za tworzenie pytań z różnych kategorii** (matematyka, programowanie, nauki ścisłe, język angielski) — ich struktura była analogiczna względem struktury pytań dziennych.
- **Pobieranie i przesyłanie odpowiedzi do pytań z różnych dziedzin** — testy przeprowadzono tylko dla pytań matematycznych, co uznano za wystarczające dla oceny całego systemu.

6.2.4 Wyniki testów

- **Najkrótszy czas odpowiedzi**: poniżej 20 ms (w testach logowania i aktywacji konta).
- **Najdłuższy czas odpowiedzi**: około 100 ms (w testach rejestracji i pobierania statystyk użytkowników).
- **Poprawność zwracanych danych**: wszystkie testowane API zwracały poprawne i kompletne dane, zgodnie z oczekiwaniami.
- **Spójność struktur JSON**: weryfikacja pól w odpowiedziach potwierdziła ich zgodność z założeniami systemu.
- **Stabilność API**: żadne z testowanych endpointów nie zwróciło błędu ani nie wykazywało niestabilności.

6.2.5 Wnioski

Testy przeprowadzone w ramach weryfikacji kluczowych endpointów wykazały poprawne działanie systemu. Wszystkie z przetestowanych API zwracały odpowiedzi w

akceptowalnym czasie. W trakcie testów została pominięta część endpointów o strukturze analogicznej względem już przetestowanych, co pozwoliło zoptymalizować proces. Otrzymane po testach wyniki potwierdzają stabilność i zgodność z wymaganiami funkcjonalnymi API.

6.3 Testy manualne

Podstawowy rodzaj testów, przeprowadzony przez jednego z członków zespołu, polegający na weryfikowaniu działania strony z perspektywy użytkownika końcowego. Analizowana była interakcja z interfejsem, sprawdzano poprawność działania konkretnych funkcji. Dodatkowo oceniona została intuicyjność poszczególnych elementów aplikacji. Testy te umożliwiły rozpoznanie problemów związanych z wyświetlaniem, nawigacją i logiką działania, które nie zawsze można było przewidzieć na etapie implementacji.

Manualne testowanie aplikacji ma istotne zalety, ponieważ nie wymaga dodatkowego kodu, ale również jest zdecydowanie bardziej elastyczne od innych metod testowania. Umożliwia dostosowywanie przypadków testowych do warunków, co pozwala na szybkie rozpoznanie błędów bez konieczności modyfikacji skryptów testowych.

W przypadku aplikacji SQULIO testy manualne dotyczyły poszczególnych funkcjonalności, takich jak rejestracja, logowanie, obsługa quizów, zarządzanie postacią użytkownika czy mechanizmy nagradzania. Sprawdzone zostało również działanie systemu zakupów oraz ekwipunku, weryfikując czy użytkownik ma możliwość poprawnego zakupu i przypisywania przedmiotów dodatkowych do swojej postaci.

Tabela 6.1 Testy manualne (źródło: opracowanie własne)

ID testu	Opis	Spodziewany rezultat	Wynik
1	Rejestracja użytkownika		
1	Użytkownik uzupełnia nazwę, email oraz hasło	Wprowadzone dane pomyślnie przechodzą walidacje a na podany adres e-mail wysyłany jest kod aktywacyjny	pozytywny
1	Użytkownik uzupełnia kod aktywacyjny	Wprowadzony kod jest poprawny a konto zostaje założone	pozytywny
1	Użytkownik zostaje przekierowany na stronę tworzenia postaci	Przekierowanie przebiegło pomyślnie	pozytywny
2	Logowanie użytkownika		
2	Użytkownik uzupełnia e-mail oraz hasło	Wprowadzane dane pomyślnie przechodzą weryfikację a użytkownik przekierowywany jest na stronę główną	pozytywny
2	Użytkownik zostaje	Przekierowanie przebiegło	pozytywny

	przekierowany na stronę główną aplikacji	pomyślnie	
3	Logowanie na błędne dane		
3	Użytkownik uzupełnia e-mail, ale podaje niepoprawne hasło	Wprowadzane dane nie przechodzą weryfikacji, użytkownik otrzymuje informację o błędnych danych	pozytywny
3	Użytkownik po raz 10 wprowadza niepoprawne hasło dla tego samego adresu e-mail	Aplikacja wykrywa zbyt wiele prób logowania na dany adres e-mail i blokuje możliwość logowania rozpoznając użytkownika po publicznym adresie IP na 15 minut. Jednocześnie na adres e-mail którego dotyczyły nieudane próby logowania zostaje wysłana informacja o wielu próbach dostania się na konto	pozytywny
4	Tworzenie postaci po pomyślnie przeprowadzonej rejestracji		
4	Użytkownik po stworzeniu nowego konta przekierowany zostaje na stronę tworzenia swojej postaci	Przekierowanie przebiegło pomyślnie	pozytywny
4	Użytkownik wybiera nazwę oraz wygląd swojej postaci	Wprowadzona nazwa pomyślnie przechodzi weryfikacje i wraz z wybranym wyglądem zostaje przypisana do użytkownika	pozytywny
4	Użytkownik zostaje przekierowany na stronę główną aplikacji	Przekierowanie przebiegło pomyślnie	pozytywny
5	Poprawne odpowiadanie na pytania		
5	Użytkownik wybiera na stronie głównej przycisk “UCZ SIĘ!”	Pokazywane jest menu do wyboru kategorii pytań	pozytywny
5	Użytkownik wybiera kategorie pytań i zostaje przekierowany do strony wyświetlającej pytania z wybranej kategorii dopasowane do poziomu umiejętności użytkownika	Przekierowanie przebiegło pomyślnie i wyświetlone zostało pierwsze pytanie	pozytywny
5	Użytkownik wybiera poprawną odpowiedź z 4 dostępnych	Wyświetlony zostaje komunikat o poprawnej	pozytywny

		odpowiedzi i przycisk do przejścia do następnego pytania a użytkownik otrzymuje doświadczenie i monety których ilość uzależniona jest od poziomu pytania	
5	Użytkownik wybiera przycisk wyjścia	Następuje przekierowanie na stronę główną	pozytywny
6	Błędne odpowiadanie na pytania		
6	Użytkownik wybiera na stronie głównej przycisk "UCZ SIĘ!"	Pokazywane jest menu do wyboru kategorii pytań	pozytywny
6	Użytkownik wybiera kategorie pytań i zostaje przekierowany do strony wyświetlającej pytania z wybranej kategorii dopasowane do poziomu umiejętności użytkownika	Przekierowanie przebiegło pomyślnie i wyświetlone zostało pierwsze pytanie	pozytywny
6	Użytkownik wybiera błędną odpowiedź z 4 dostępnych	Wyświetlony zostaje komunikat o błędnej odpowiedzi i przycisk do przejścia do następnego pytania a użytkownik traci punkt zdrowia	pozytywny
6	Użytkownik wybiera przycisk wyjścia	Następuje przekierowanie na stronę główną	pozytywny
7	Utrata wszystkich punktów zdrowia		
7	Użytkownik wybiera błędną odpowiedź w trakcie wykonywania pytań posiadając 1 punkt zdrowia	Użytkownik traci ostatni punkt zdrowia, otrzymuje komunikat o zablokowaniu możliwości wykonywania pytań. Blokowane jest odpowiadanie na pytania z każdej kategorii poza "Codzienne Pytania"	pozytywny
7	Użytkownik wybiera przycisk wyjścia	Następuje przekierowanie na stronę główną	pozytywny
8	Zakup przedmiotu kosmetycznego		
8	Użytkownik wybiera na stronie głównej przycisk sklepu	Następuje przekierowanie na stronę sklepu	pozytywny
8	Użytkownik wybiera przedmiot który chce zakupić oraz wybiera widoczny przy nim przycisk	Saldo oraz poziom użytkownika zostają sprawdzone i jeśli są	pozytywny

	"Kup"	wystarczające, przedmiot przyznawany jest użytkownikowi a jego cena odejmowana jest od jego salda	
8	Użytkownik wybiera przycisk wyjścia	Następuje przekierowanie na stronę główną	pozytywny
9	Założenie na postać przedmiotu kosmetycznego		
9	Użytkownik wybiera na stronie głównej przycisk szatni	Następuje przekierowanie na stronę szatni	pozytywny
9	Użytkownik wybiera przedmiot który chce założyć oraz klika na niego	Przedmiot zostaje założony na postać użytkownika a także następuje zapis do bazy danych dzięki czemu postać, do czasu zdjęcia przedmiotu, będzie miała go przypisanego	pozytywny
9	Użytkownik wybiera przycisk wyjścia	Następuje przekierowanie na stronę główną	pozytywny
10	Zdjęcie z postaci przedmiotu kosmetycznego		
10	Użytkownik wybiera na stronie głównej przycisk szatni	Następuje przekierowanie na stronę szatni	pozytywny
10	Użytkownik wybiera przedmiot który chce zdjąć oraz klika na niego	Przedmiot zostaje zdjęty z postaci użytkownika a także następuje zapis do bazy danych	pozytywny
10	Użytkownik wybiera przycisk wyjścia	Następuje przekierowanie na stronę główną	pozytywny
11	Wyświetlenie dostępnych zadań		
11	Użytkownik wybiera na stronie głównej przycisk zadań	Pokazywane jest okienko z dostępnymi zadaniami oraz ich stopniem postępu	pozytywny
11	Użytkownik wybiera inne poziomy zadań	Pokazywane są zadania z innych poziomów doświadczenia wraz z ich postępem	pozytywny
11	Użytkownik wybiera przycisk wyjścia	Następuje przekierowanie na stronę główną	pozytywny
12	Wykonanie zadań z danego poziomu w celu uzyskania "Diamencika"		

12	Użytkownik wykonuje zadania w danym poziomie doświadczenia	Postęp w zadaniach zostaje zapisany i wyświetlony użytkownikowi	pozytywny
12	Wszystkie zadania zostają w pełni wykonane w danym poziomie doświadczenia	Użytkownik otrzymuje diamencik który jest drugą walutą w aplikacji	pozytywny
13	Wyświetlenie dostępnych osiągnięć		
13	Użytkownik wybiera na stronie głównej przycisk osiągnięć	Pokazywane jest okienko z dostępnymi osiągnięciami oraz ich stopniem postępu	pozytywny
13	Użytkownik wybiera przycisk wyjścia	Następuje przekierowanie na stronę główną	pozytywny
14	Wylogowanie z aplikacji		
14	Użytkownik wybiera na stronie głównej przycisk wylogowania	Następuje usunięcie informacji sesyjnych użytkownika oraz przekierowanie na stronę logowania	pozytywny

6.4 Testy E2E

Zmiany wprowadzone na serwerze w celu uruchomienia testów:

- *sudo apt-get install -y libgtk-3-0 libgbm-dev libnotify-dev libgconf-2-4 libnss3 libxss1 libasound2 libxtst6 xauth xvfb*
- *Xvfb :99 -screen 0 1024x768x24 &*
- *export DISPLAY=:99*

W celu przeprowadzenia testów e2e zostało użyte narzędzie cypress

W frontend została zainstalowana usługa npm install cypress --save-dev oraz stworzona odpowiednia struktura:

```
frontend/cypress
|-- e2e
|   `-- answer_question.cy.js
|-- integration
`-- screenshots
```

Główny plik konfiguracyjny:

```
JS cypress.config.js ×
frontend > JS cypress.config.js > default
1   import { defineConfig } from 'cypress';
2
3   export default defineConfig({
4     e2e: {
5       baseUrl: 'https://squlio.edu.pl',
6       setupNodeEvents(on, config) {
7       },
8       supportFile: false,
9       specPattern: 'cypress/e2e/**/*.cy.{js,jsx,ts,tsx}',
10    },
11  });
```

Rysunek 6.29 Główny plik konfiguracyjny cypress (źródło: opracowanie własne)

Przykładowy skrypt użyty do testu:

Jego zadaniem jest weryfikacja poprawności procesu logowania, przechodzenia do różnych kategorii pytań oraz interakcji z interfejsem poprzez wybór odpowiedzi i nawigację.

```
JS answer_question.cy.js ×
frontend > cypress > e2e > JS answer_question.cy.js > describe('Odpowiadanie na pytania z różnych dziedzin') callback
1   describe('Odpowiadanie na pytania z różnych dziedzin', () => {
2     beforeEach(() => {
3       cy.visit('/login');
4       cy.get('input[name="email"]').type('wojtek.pestka@onet.pl');
5       cy.get('input[name="password"]').type('admin123');
6       cy.get('button[type="submit"]').click();
7       cy.wait(5000);
8       cy.url().should('include', '/menu');
9
10    });
11
12    const answerSingleQuestion = (categoryButtonSelector, categoryUrl) => {
13      cy.get('.learn-button').click();
14      cy.get(categoryButtonSelector).click();
15
16      cy.url().should('include', categoryUrl);
17
18
19      cy.get('.question-box').should('be.visible');
20
21      cy.get('.answers button').first().click();
22
23      cy.get('.feedback').should('be.visible');
24
25      cy.get('.back-button').click();
26    };
27
28    it('Powinno odpowiedzieć na jedno pytanie programistyczne', () => {
29      answerSingleQuestion('.programming-questions-button', '/programming-questions');
30    });
31
32    it('Powinno odpowiedzieć na jedno pytanie matematyczne', () => {
33      answerSingleQuestion('.math-questions-button', '/math-questions');
34    });
35
36    it('Powinno odpowiedzieć na jedno pytanie z nauk ścisłych', () => {
37      answerSingleQuestion('.science-questions-button', '/science-questions');
38    });
39
40    it('Powinno odpowiedzieć na jedno pytanie z angielskiego', () => {
41      answerSingleQuestion('.english-questions-button', '/english-questions');
42    });
43
44    afterEach(() => {
45      cy.get('.logout-icon').click();
46      cy.url().should('include', '/login');
47    });
48  });
```

Rysunek 6.30 Skrypt testu (źródło: opracowanie własne)

Wynik testu:

```
==========================================================================================

  (Run Starting)

  ┌────────────────────────────────────────────────────────────────────────────────────┐
  │ Cypress:        14.1.0                                                             │
  │ Browser:        Electron 130 (headless)                                            │
  │ Node Version:   v18.19.0 (/usr/bin/node)                                           │
  │ Specs:          1 found (answer_question.cy.js)                                    │
  │ Searched:       cypress/e2e/**/*.cy.{js,jsx,ts,tsx}                                │
  └────────────────────────────────────────────────────────────────────────────────────┘


────────────────────────────────────────────────────────────────────────────────────────

  Running:  answer_question.cy.js                                                   (1 of 1)


  Odpowiadanie na pytania z różnych dziedzin
    ✓ Powinno odpowiedzieć na jedno pytanie programistyczne (6450ms)
    ✓ Powinno odpowiedzieć na jedno pytanie matematyczne (6173ms)
    ✓ Powinno odpowiedzieć na jedno pytanie z nauk ścisłych (6159ms)
    ✓ Powinno odpowiedzieć na jedno pytanie z angielskiego (6125ms)


  4 passing (26s)


  (Results)

  ┌────────────────────────────────────────────────────────────────────────────────────┐
  │ Tests:        4                                                                    │
  │ Passing:      4                                                                    │
  │ Failing:      0                                                                    │
  │ Pending:      0                                                                    │
  │ Skipped:      0                                                                    │
  │ Screenshots:  0                                                                    │
  │ Video:        false                                                                │
  │ Duration:     25 seconds                                                           │
  │ Spec Ran:     answer_question.cy.js                                                │
  └────────────────────────────────────────────────────────────────────────────────────┘


==========================================================================================

  (Run Finished)


       Spec                                              Tests  Passing  Failing  Pending  Skipped
  ┌────────────────────────────────────────────────────────────────────────────────────────────┐
  │ ✓  answer_question.cy.js                   00:25       4        4        -        -        - │
  └────────────────────────────────────────────────────────────────────────────────────────────┘
    ✓  All specs passed!                       00:25       4        4        -        -        -
```

Rysunek 6.31 Wynik testu (źródło: opracowanie własne)

Skrypt wykonuje się stosunkowo długo (26s) ze względu na celowe opóźnienie wprowadzone za pomocą polecenia cy.wait(5000);, które powoduje 5-sekundową pauzę po zalogowaniu użytkownika. Jest to konieczny zabieg wynikający z mechanizmów zabezpieczeń aplikacji, które wykrywają podejrzane aktywności, takie jak zbyt szybkie i powtarzalne żądania logowania, mogące sugerować atak typu brute-force lub automatyczne boty. Dzięki temu opóźnieniu test symuluje bardziej naturalne zachowanie

użytkownika, co pozwala uniknąć blokad systemowych i umożliwia poprawną weryfikację funkcjonalności aplikacji.

6.4 Podsumowanie testowania

Proces testowania aplikacji pozwolił nie tylko na poznanie i eliminację błędów, ale również na upewnienie się, że system jest niezawodny, zoptymalizowany i odporny na nieprzewidziane sytuacje. Testy manualne umożliwiły symulację rzeczywistego użytkowania aplikacji przez przyszłego odbiorcę, natomiast testy przeprowadzone z pomocą narzędzia Postman zweryfikowały komunikację między poszczególnymi komponentami systemu oraz poprawność obsługi danych.

Wnioski z przeprowadzonych testów świadczą, że testowanie we wcześniejszych etapach rozwoju oprogramowania nie tylko umożliwia szybszą naprawę błędów, ale przede wszystkim efektywne planowanie rozwoju systemu. Wykonany przez zespół proces ujawnił znaczenie regularnej weryfikacji funkcjonalności wdrażanych do aplikacji oraz dostosowywania strategii testowania do zmieniających się wymagań projektu. Pozwoliło to na utrzymanie stabilności aplikacji i zapewnienie przyszłym użytkownikom bezproblemowego dostępu i działania. W przyszłym rozwoju kluczowym powinno być wprowadzanie większej automatyzacji, testów na większej ilości urządzeń i długoterminowe monitorowanie błędów po wdrożeniu.

Podsumowanie

W powyższej pracy inżynierskiej zajęliśmy się projektowaniem i implementacją internetowej gry przygodowej, której celem jest wspieranie edukacji wczesnoszkolnej poprzez elementy grywalizacji oraz interaktywną formę nauki. Przed rozpoczęciem prac nad samą aplikacją przeprowadziliśmy szeroką analizę dostępnych na rynku gier edukacyjnych, ich mechanizmów oraz wpływu na proces przyswajania wiedzy przez dzieci. Omówiliśmy także różne podejścia dydaktyczne stosowane w grach, identyfikując te, które najlepiej sprawdzają się w nauczaniu najmłodszych użytkowników.

Podstawowym celem naszej pracy było stworzenie aplikacji, która nie tylko wspomaga proces edukacyjny, ale także angażuje użytkownika poprzez zastosowanie różnorodnych mechanizmów grywalizacyjnych, takich jak system nagród, personalizacja postaci oraz zdobywanie osiągnięć. Przyjęliśmy konkretne założenia projektowe, a także podzieliliśmy zadania pomiędzy członków zespołu, aby zapewnić efektywność pracy. Do organizacji procesu deweloperskiego wykorzystaliśmy metodykę Kanban oraz diagram Gantta, co pozwoliło nam na skuteczne zarządzanie zadaniami i harmonogramem projektu.

W trakcie realizacji projektu zajęliśmy się zarówno warstwą wizualną, jak i funkcjonalną aplikacji. Zaimplementowaliśmy system rejestracji i logowania użytkowników, moduł personalizacji postaci, a także mechanizmy edukacyjne w postaci interaktywnych quizów i wyzwań. Ponadto, stworzyliśmy system nagród, który motywuje użytkowników do regularnej nauki poprzez zdobywanie punktów, osiągnięć oraz odblokowywanie nowych elementów w grze. Przeprowadziliśmy także szczegółową analizę struktury aplikacji, przygotowując diagramy przypadków użycia oraz diagramy klas, które przedstawiają organizację systemu oraz jego kluczowe funkcjonalności.

Ważnym etapem realizacji projektu było testowanie stworzonego rozwiązania. Przeprowadziliśmy zarówno testy manualne, jak i automatyczne z wykorzystaniem narzędzia Postman, co pozwoliło nam wykryć i wyeliminować błędy związane z działaniem systemu. Dzięki temu mogliśmy zoptymalizować wydajność aplikacji oraz zapewnić jej stabilność na różnych urządzeniach i przeglądarkach.

Podczas realizacji pracy napotkaliśmy kilka wyzwań i problemów, które wymagały od nas dodatkowego zaangażowania i poszukiwania alternatywnych rozwiązań. Jednym z głównych wyzwań był brak doświadczenia w zakresie projektowania graficznego wśród członków zespołu. Dodatkowo, integracja poszczególnych modułów aplikacji oraz optymalizacja wydajności systemu na różnych urządzeniach okazały się bardziej złożonymi zadaniami, niż początkowo zakładaliśmy. Niemniej jednak, dzięki ścisłej współpracy oraz profesjonalnemu podejściu do rozwiązywania problemów, udało nam się skutecznie pokonać napotkane trudności i doprowadzić projekt do pomyślnego zakończenia.

Badania potwierdziły naszą tezę, że wykorzystanie elementów grywalizacji w edukacji może znacząco zwiększyć zaangażowanie uczniów oraz poprawić skuteczność nauki. Stworzona przez nas aplikacja łączy interaktywność z edukacyjnym charakterem, oferując dzieciom atrakcyjną alternatywę dla tradycyjnych metod nauczania. Dodatkowo, zaimplementowane rozwiązania pozwalają na dostosowanie poziomu trudności do umiejętności użytkownika, co sprawia, że aplikacja może być wykorzystywana zarówno w domu, jak i w środowisku szkolnym jako uzupełnienie standardowych metod dydaktycznych.

Następnymi krokami byłby dalszy rozwój projektu poprzez dodanie trybu wieloosobowego, co umożliwi uczniom rywalizację oraz współpracę w czasie rzeczywistym. Kolejnym etapem będzie wzbogacenie aplikacji o nowe treści edukacyjne,

co pozwoli na poszerzenie zakresu tematycznego gry i dostosowanie jej do różnych grup wiekowych. Planujemy także wdrożenie bardziej zaawansowanych mechanizmów analizy postępów użytkowników, co umożliwi nauczycielom i rodzicom monitorowanie wyników i dostosowywanie poziomu trudności do indywidualnych potrzeb dziecka.

Podsumowując, nasza praca inżynierska dowodzi, że odpowiednio zaprojektowane gry edukacyjne mogą stanowić skuteczne narzędzie wspomagające proces nauki. Dzięki połączeniu elementów rozrywkowych z dydaktycznymi stworzyliśmy aplikację, która nie tylko angażuje użytkownika, ale także ułatwia przyswajanie wiedzy. Wierzymy, że dalszy rozwój projektu przyczyni się do jeszcze większej efektywności w edukacji dzieci oraz poszerzy możliwości wykorzystania gier jako nowoczesnych narzędzi dydaktycznych.

Bibliografia

1. Clark, R. E.; Mayer, R. E. — E-Learning and the Science of Instruction: Proven Guidelines for Consumers and Designers of Multimedia Learning, https://www.scirp.org/reference/referencespapers?referenceid=3480061 (dostęp: 03.03.2025)

2. Deterding S.; Dixon D.; Khaled R.; Nacke L. — From Game Design Elements to Gamefulness: Defining Gamification, https://www.researchgate.net/publication/230854710_From_Game_Design_Elements_to_Gamefulness_Defining_Gamification (dostęp: 03.03.2025)

3. Kondracka-Szala M.; Uniwersytet Wrocławski — Gry w rozwijaniu wybranych kompetencji kluczowych dzieci — przegląd literatury przedmiotu, https://repozytorium.uni.wroc.pl/Content/89115/19_Kondracka_Szala_Marta_Gry_w_rozwijaniu_wybranych_kompetencji_kluczowych_dzieci.pdf (dostęp: 03.03.2025)

4. Novac O. C.; Ghiurău D.; Novac M. C.; Gordan C. M.; Oproescu M.; Bujdoso G., Comparison of Node.Js and Spring Boot in Web Development, https://ieeexplore.ieee.org/document/10194025 (dostęp: 05.03.2025)

5. Papert S. — The Children's Machine RETHINKING SCHOOL IN THE AGE OF THE COMPUTER, https://lcl.media.mit.edu/resources/readings/childrens-machine.pdf (dostęp: 03.03.2025)

6. Witoszek-Kubicka A.; Makowiec M. — Grywalizacja w edukacji i biznesie, https://www.researchgate.net/profile/Aleksandra_Witoszek2/publication/344332232_Grywalizacja_w_edukacji_i_biznesie/links/5f68e384458515b7cf46836d/Grywalizacja-w-edukacji-i-biznesie.pdf (dostęp: 03.03.2025)

7. Kanbantool, Czym jest metoda Kanban i jak może ci pomóc?, https://kanbantool.com/pl/metoda-kanban (dostęp: 03.03.2025)

8. Middleware – co to jest?, https://double-digital.pl/baza-wiedzy/middleware-co-to-jest (dostęp: 05.03.2025)

Spis rysunków

Rysunek 3.1 Diagram Gantta (źródło: opracowanie własne)
Strona 12

Rysunek 3.2 Diagram akcji - system logowania (źródło: opracowanie własne)
Strona 21

Rysunek 3.3 Diagram akcji - system rejestracji (źródło: opracowanie własne)
Strona 22

Rysunek 3.4 Diagram akcji - system resetowania hasła (źródło: opracowanie własne)
Strona 23

Rysunek 3.5 Diagram akcji - system aktywacji konta (źródło: opracowanie własne)
Strona 24

Rysunek 3.6 Diagram akcji - system tworzenia postaci (źródło: opracowanie własne)
Strona 25

Rysunek 3.7 Diagram akcji - zakup serduszek (źródło: opracowanie własne)
Strona 26

Rysunek 3.8 Diagram akcji - zakup przedmiotu w sklepie (źródło: opracowanie własne)
Strona 27

Rysunek 3.9 Diagram akcji - zakup edycji postaci (źródło: opracowanie własne)
Strona 28

Rysunek 3.10 Diagram akcji - ubranie przedmiotu (źródło: opracowanie własne)
Strona 29

Rysunek 3.11 Diagram akcji - wylogowanie się (źródło: opracowanie własne)
Strona 30

Rysunek 3.12 Diagram akcji - odpowiadanie na pytanie (źródło: opracowanie własne)
Strona 30

Rysunek 3.13 Diagram akcji - dodaj nowe pytanie (źródło: opracowanie własne)
Strona 31

Rysunek 3.14 Diagram przypadku - dodaj nowy przedmiot do sklepu (źródło: opracowanie własne)
Strona 31

Rysunek 3.15 Diagram klas dla pliku AchievementsController.js (źródło: opracowanie własne)
Strona 32

Rysunek 3.16 Diagram klas dla pliku ActivateUserController.js (źródło: opracowanie własne)
Strona 33

Rysunek 3.17 Diagram klas dla pliku AdminController.js (źródło: opracowanie własne)
Strona 33

Rysunek 3.18 Diagram klas dla pliku CharacterController.js (źródło: opracowanie własne)
Strona 34

Rysunek 3.19 Diagram klas dla pliku DailyQuestionController.js (źródło: opracowanie własne)
Strona 34

Rysunek 3.20 Diagram klas dla pliku DailyStatsController.js (źródło: opracowanie własne)
Strona 35

Rysunek 3.21 Diagram klas dla pliku EnglishQuestionController.js
Strona 35

Rysunek 3.22 Diagram klas dla pliku GetCharacterController.js (źródło: opracowanie własne)
Strona 36

Rysunek 3.23 Diagram klas dla pliku HeartShopController.js (źródło: opracowanie własne)
Strona 36

Rysunek 3.24 Diagram klas dla pliku ItemShopController.js (źródło: opracowanie własne)
Strona 37

Rysunek 3.25 Diagram klas dla pliku LoginController.js (źródło: opracowanie własne)
Strona 37

Rysunek 3.26 Diagram klas dla pliku MathQuestionsController.js (źródło: opracowanie własne)
Strona 38

Rysunek 3.27 Diagram klas dla pliku ProgramingQuestionsController.js (źródło: opracowanie własne)
Strona 39

Rysunek 3.28 Diagram klas dla pliku RegisterController.js (źródło: opracowanie własne)
Strona 39

Rysunek 3.29 Diagram klas dla pliku ResetPasswordController.js (źródło: opracowanie własne)
Strona 39

Rysunek 3.30 Diagram klas dla pliku ScienceQuestionsController.js (źródło: opracowanie własne)
Strona 40

Rysunek 3.31 Diagram klas dla pliku TasksController.js (źródło: opracowanie własne)
Strona 41

Rysunek 3.32 Diagram klas dla pliku UserStatsController.js (źródło: opracowanie własne)
Strona 42

Rysunek 3.33 Diagram klas dla pliku WardrobeController.js (źródło: opracowanie własne)
Strona 42

Rysunek 3.34 Diagram klas dla pliku dailyQuizCron.js (źródło: opracowanie własne)
Strona 43

Rysunek 3.35 Diagram klas dla pliku dailyStatistics.js (źródło: opracowanie własne)
Strona 44

Rysunek 3.36 Diagram klas dla pliku restoreHp.js (źródło: opracowanie własne)
Strona 45

Rysunek 3.37 Diagram klas dla pliku userDataCron.js (źródło: opracowanie własne)
Strona 45

Rysunek 3.38 Diagram klas dla pliku emailservice.js (źródło: opracowanie własne)
Strona 46

Rysunek 3.39 Diagram klas dla pliku attackDetection.js (źródło: opracowanie własne)
Strona 46

Rysunek 3.40 Diagram klas dla pliku authenticateUser.js (źródło: opracowanie własne)
Strona 46

Rysunek 3.41 Diagram klas dla pliku checkEditableCharacter.js (źródło: opracowanie własne)
Strona 47

Rysunek 3.42 Diagram klas dla pliku tasks.js (źródło: opracowanie własne)
Strona 48

Rysunek 3.43 Diagram klas dla pliku codeValidation.js (źródło: opracowanie własne)
Strona 49

Rysunek 3.44 Diagram klas dla pliku emailValidation.js (źródło: opracowanie własne)
Strona 49

Rysunek 3.45 Diagram klas dla pliku passwordValidation.js (źródło: opracowanie własne)
Strona 49

Rysunek 3.46 Diagram klas dla pliku usernameValidation.js (źródło: opracowanie własne)
Strona 50

Rysunek 3.47 Diagram klas dla pliku server.js (źródło: opracowanie własne)
Strona 50

Rysunek 3.48 Diagram klas dla pliku connection.js i database.js (źródło: opracowanie własne)
Strona 50

Rysunek 3.49 Diagram klas dla pliku app.js (źródło: opracowanie własne)
Strona 51

Rysunek 3.50 Diagram klas dla pliku Activation.tsx (źródło: opracowanie własne)
Strona 52

Rysunek 3.51 Diagram klas dla pliku Achivments.tsx (źródło: opracowanie własne)
Strona 52

Rysunek 3.52 Diagram klas dla pliku AdminDashBoard.tsx (źródło: opracowanie własne)
Strona 52

Rysunek 3.53 Diagram klas dla pliku CharacterBuild.tsx (źródło: opracowanie własne)
Strona 53

Rysunek 3.54 Diagram klas dla pliku DailyQuestions.tsx (źródło: opracowanie własne)
Strona 53

Rysunek 3.55 Diagram klas dla pliku DailyStats.tsx (źródło: opracowanie własne)
Strona 53

Rysunek 3.56 Diagram klas dla pliku EnglishQuestions.tsx (źródło: opracowanie własne)
Strona 54

Rysunek 3.57 Diagram klas dla pliku HeartShop.tsx (źródło: opracowanie własne)
Strona 54

Rysunek 3.58 Diagram klas dla pliku ItemShop.tsx (źródło: opracowanie własne)
Strona 55

Rysunek 3.59 Diagram klas dla pliku Login.tsx (źródło: opracowanie własne)
Strona 55

Rysunek 3.60 Diagram klas dla pliku MainContent.tsx (źródło: opracowanie własne)
Strona 56

Rysunek 3.61 Diagram klas dla pliku MainMenu.tsx (źródło: opracowanie własne)
Strona 56

Rysunek 3.62 Diagram klas dla pliku MathQuestions.tsx (źródło: opracowanie własne)
Strona 57

Rysunek 3.63 Diagram klas dla pliku ProgramingQuestions.tsx (źródło: opracowanie własne)
Strona 57

Rysunek 3.64 Diagram klas dla pliku Register.tsx (źródło: opracowanie własne)
Strona 58

Rysunek 3.65 Diagram klas dla pliku ResetPassword.tsx (źródło: opracowanie własne)
Strona 58

Rysunek 3.66 Diagram klas dla pliku ScienceQuestions.tsx (źródło: opracowanie własne)
Strona 58

Rysunek 3.67 Diagram klas dla pliku StartButton.tsx (źródło: opracowanie własne)
Strona 59

Rysunek 3.68 Diagram klas dla pliku Wardrobe.tsx (źródło: opracowanie własne)
Strona 59

Rysunek 3.69 Diagram klas dla pliku Zadania.tsx (źródło: opracowanie własne)
Strona 59

Rysunek 3.70 Diagram klas dla pliku App.tsx (źródło: opracowanie własne)
Strona 60

Rysunek 3.71 Diagram komponentów(źródło: opracowanie własne)
Strona 61

Rysunek 3.72 Diagram przypadków użycia - panel logowania i rejestracji (źródło: opracowanie własne)
Strona 62

Rysunek 3.73 Diagram przypadków użycia - zakup dodatkowego serduszka (źródło: opracowanie własne)
Strona 62

Rysunek 3.74 Diagram przypadków użycia - zakup przedmiotu w sklepie (źródło: opracowanie własne)
Strona 62

Rysunek 3.75 Diagram przypadków użycia - zakup edycji postaci (źródło: opracowanie własne)
Strona 63

Rysunek 3.76 Diagram przypadków użycia - założenie przedmiotu z garderoby (źródło: opracowanie własne)
Strona 63

Rysunek 3.77 Diagram przypadków użycia - wylogowanie (źródło: opracowanie własne)
Strona 64

Rysunek 3.78 Diagram przypadków użycia - odpowiadanie na pytania (źródło: opracowanie własne)
Strona 64

Rysunek 3.79 Diagram przypadków użycia - dodanie nowego pytania (źródło: opracowanie własne)
Strona 65

Rysunek 3.80 Diagram przypadków użycia - dodanie nowego pytania (źródło: opracowanie własne)
Strona 65

Rysunek 3.81 Diagram przypadków użycia - edycja użytkownika (źródło: opracowanie własne)
Strona 65

Rysunek 3.82 Diagram przypadków użycia - edycja użytkownika (źródło: opracowanie własne)
Strona 66

Rysunek 5.1 Skrypty w pliku package.json (źródło: opracowanie własne)
Strona 72

Rysunek 5.2 Struktura frontendu (źródło: opracowanie własne)
Strona 73

Rysunek 5.3 Plik package.json (źródło: opracowanie własne)
Strona 75

Rysunek 5.4 Struktura backendu aplikacji (źródło: opracowanie własne)
Strona 76

Rysunek 5.5 Plik database.js (źródło: opracowanie własne)
Strona 77

Rysunek 5.6 plik connection.js (źródło: opracowanie własne)
Strona 77

Rysunek 5.7 Lista kontrolerów w aplikacji (źródło: opracowanie własne)
Strona 78

Rysunek 5.8 Pliki cron w aplikacji (źródło: opracowanie własne)
Strona 79

Rysunek 5.9 Pliki cron w aplikacji (źródło: opracowanie własne)
Strona 80

Rysunek 5.10 Konfiguracja wysyłki e-maili (źródło: opracowanie własne)
Strona 80

Rysunek 5.11 Zawartość pliku authenticateUser.js (źródło: opracowanie własne)
Strona 82

Rysunek 5.12 Zawartość katalogu /models/ (źródło: opracowanie własne)
Strona 83

Rysunek 5.13 Model Achievements.js (źródło: opracowanie własne)
Strona 84

Rysunek 5.14 Model DailyQuestions.js (źródło: opracowanie własne)
Strona 84

Rysunek 5.15 Model DailyQuiz.js (źródło: opracowanie własne)
Strona 85

Rysunek 5.16 Model DailyStatistics.js (źródło: opracowanie własne)
Strona 85

Rysunek 5.17 Model EnglishQuestions.js (źródło: opracowanie własne)
Strona 86

Rysunek 5.18 Model ItemShop.js (źródło: opracowanie własne)
Strona 86

Rysunek 5.19 Model MathsQuestions.js (źródło: opracowanie własne)
Strona 87

Rysunek 5.20 Model ProgramingQuestions.js (źródło: opracowanie własne)
Strona 87

Rysunek 5.21 Model ScienceQuestions.js (źródło: opracowanie własne)
Strona 88

Rysunek 5.22 Model Tasks.js (źródło: opracowanie własne)
Strona 88

Rysunek 5.23 Model User.js (źródło: opracowanie własne)
Strona 89

Rysunek 5.24 Import modułów w pliku routes/index.js (źródło: opracowanie własne)
Strona 90

Rysunek 5.25 Import kontrolerów routes/index.js (źródło: opracowanie własne)
Strona 91

Rysunek 5.26 Import middleware routes/index.js (źródło: opracowanie własne)
Strona 91

Rysunek 5.27 Trasy API routes/index.js (źródło: opracowanie własne)
Strona 92

Rysunek 5.28 Import modułów w pliku app.js (źródło: opracowanie własne)
Strona 94

Rysunek 5.29 Tworzenie instancji aplikacji w pliku app.js (źródło: opracowanie własne)
Strona 94

Rysunek 5.30 Konfiguracja zabezpieczeń w pliku app.js (źródło: opracowanie własne)
Strona 94

Rysunek 5.31 Obsługa sesji użytkowników w pliku app.js (źródło: opracowanie własne)
Strona 95

Rysunek 5.32 Konfiguracja tras API w pliku app.js (źródło: opracowanie własne)
Strona 95

Rysunek 5.33 Konfiguracja statycznych plików aplikacji frontendowej w pliku app.js (źródło: opracowanie własne)
Strona 95

Rysunek 5.34 Importowanie modułów w pliku server.js (źródło: opracowanie własne)
Strona 96

Rysunek 5.35 Konfiguracja certyfikatów SSL w pliku server.js (źródło: opracowanie własne)
Strona 96

Rysunek 5.36 Tworzenie i uruchamianie serwera HTTPS w pliku server.js (źródło: opracowanie własne)
Strona 96

Rysunek 5.37 Informacja zwrotna wyświetlana po uruchomieniu serwera (źródło: opracowanie własne)
Strona 97

Rysunek 5.38 Przykładowy użytkownik w kolekcji users (źródło: opracowanie własne)
Strona 98

Rysunek 5.39 Wygląd kolekcji dailystatistics (źródło: opracowanie własne)
Strona 99

Rysunek 5.40 Informacje o połączeniu bazy danych (źródło: opracowanie własne)
Strona 100

Rysunek 5.41 Strona startowa platformy SQULIO (źródło: opracowanie własne)
Strona 102

Rysunek 5.42 Strona logowania (źródło: opracowanie własne)
Strona 102

Rysunek 5.43 Strona resetowania hasła część I (źródło: opracowanie własne)
Strona 103

Rysunek 5.44 Strona resetowania hasła część II (źródło: opracowanie własne)
Strona 103

Rysunek 5.45 Strona rejestracji (źródło: opracowanie własne)
Strona 104

Rysunek 5.46 Strona aktywacji (źródło: opracowanie własne)
Strona 105

Rysunek 5.47 Strona tworzenia postaci (źródło: opracowanie własne)
Strona 105

Rysunek 5.48 Strona menu głównego (źródło: opracowanie własne)
Strona 106

Rysunek 5.49 Panel pytań (źródło: opracowanie własne)
Strona 107

Rysunek 5.50 Strona pytań (źródło: opracowanie własne)
Strona 107

Rysunek 5.51 Strona sklepu (źródło: opracowanie własne)
Strona 108

Rysunek 5.52 Strona garderoby (źródło: opracowanie własne)
Strona 109

Rysunek 5.53 Panel zadań (źródło: opracowanie własne)
Strona 109

Rysunek 5.54 Panel osiągnięć (źródło: opracowanie własne)
Strona 110

Rysunek 5.55 Panel zakupu żyć (źródło: opracowanie własne)
Strona 111

Rysunek 6.1 Wynik testu API: /api/register (źródło: opracowanie własne)
Strona 114

Rysunek 6.2 Wynik testu API: /api/login (źródło: opracowanie własne)
Strona 116

Rysunek 6.3 Wynik testu API: /api/activation (źródło: opracowanie własne)
Strona 118

Rysunek 6.4 Wynik testu API: /api/saveCharacter (źródło: opracowanie własne)
Strona 121

Rysunek 6.5 Wynik testu API: /api/character (źródło: opracowanie własne)
Strona 123

Rysunek 6.6 Wynik testu API: /api/CreateDailyQuestion (źródło: opracowanie własne)
Strona 125

Rysunek 6.7 Wynik testu API: /api/daily-question (źródło: opracowanie własne)
Strona 127

Rysunek 6.8 Wynik testu API: /api/submit-answer (źródło: opracowanie własne)
Strona 128

Rysunek 6.9 Wynik testu API: /api/math-questions/:level (źródło: opracowanie własne)
Strona 130

Rysunek 6.10 Wynik testu API: /api/submit-math-answer (źródło: opracowanie własne)
Strona 132

Rysunek 6.11 Wynik testu API: /CreateShopItem (źródło: opracowanie własne)
Strona 135

Rysunek 6.12 Wynik testu API: /api/shop/items (źródło: opracowanie własne)
Strona 137

Rysunek 6.13 Wynik testu API: /api/shop/buy-item (źródło: opracowanie własne)
Strona 139

Rysunek 6.14 Wynik testu API: /api/buy-edit-character (źródło: opracowanie własne)
Strona 141

Rysunek 6.15 Wynik testu API: /api/wardrobe/items (źródło: opracowanie własne)
Strona 143

Rysunek 6.16 Wynik testu API: /api/select-item (źródło: opracowanie własne)
Strona 145

Rysunek 6.17 Wynik testu API: /api/userstats (źródło: opracowanie własne)
Strona 147

Rysunek 6.18 Wynik testu API: /api/get-tasks (źródło: opracowanie własne)
Strona 149

Rysunek 6.19 Wynik testu API: /api/get-achievements (źródło: opracowanie własne)
Strona 151

Rysunek 6.20 Wynik testu API: /get-all-users (źródło: opracowanie własne)
Strona 154

Rysunek 6.21 Wynik testu API: /get-user-data/:id (źródło: opracowanie własne)
Strona 157

Rysunek 6.22 Wynik testu API: /update-user/:id (źródło: opracowanie własne)
Strona 161

Rysunek 6.23 Wynik testu API: /api/stats/:date (źródło: opracowanie własne)
Strona 163

Rysunek 6.24 Wynik testu API: /api/stats/compare/:date1/:date2 (źródło: opracowanie własne)
Strona 166

Rysunek 6.25 Wynik testu API: /api/send-reset-code (źródło: opracowanie własne)
Strona 167

Rysunek 6.26 Wynik testu API: /api/reset-password (źródło: opracowanie własne)
Strona 169

Rysunek 6.27 Wynik testu API: /api/buy-heart (źródło: opracowanie własne)
Strona 171

Rysunek 6.28 Wynik testu API: /api/get-heart-shop (źródło: opracowanie własne)
Strona 172

Rysunek 6.29 Główny plik konfiguracyjny cypress (źródło: opracowanie własne)
Strona 179

Rysunek 6.30 Skrypt testu (źródło: opracowanie własne)
Strona 180

Rysunek 6.31 Wynik testu (źródło: opracowanie własne)
Strona 181

Spis tabel

Tabela 2.1 Podział prac w dziedzinie: Zarządzanie (źródło: opracowanie własne)
Strona 9

Tabela 2.2 Podział prac w dziedzinie: Frontend (źródło: opracowanie własne)
Strona 10

Tabela 2.3 Podział prac w dziedzinie: Infrastruktura (źródło: opracowanie własne)
Strona 10

Tabela 2.4 Podział prac w dziedzinie: Testowanie (źródło: opracowanie własne)
Strona 10

Tabela 2.5 Podział prac w opracowaniu pracy inżynierskiej (źródło: opracowanie własne)
Strona 10

Tabela 3.1 Tabela wymagań funkcjonalnych - logowanie (źródło: opracowanie własne)
Strona 12

Tabela 3.2 Tabela wymagań funkcjonalnych - rejestracja (źródło: opracowanie własne)
Strona 13

Tabela 3.3 Tabela wymagań funkcjonalnych - aktywacja konta (źródło: opracowanie własne)
Strona 14

Tabela 3.4 Tabela wymagań funkcjonalnych - zapis postaci (źródło: opracowanie własne)
Strona 14

Tabela 3.5 Tabela wymagań funkcjonalnych - resetowanie hasła (źródło: opracowanie własne)
Strona 15

Tabela 3.6 Tabela wymagań funkcjonalnych - zakup nowego serduszka (źródło: opracowanie własne)
Strona 15

Tabela 3.7 Tabela wymagań funkcjonalnych - zakup przedmiotu w sklepie (źródło: opracowanie własne)
Strona 16

Tabela 3.8 Tabela wymagań funkcjonalnych - zakup edycji postaci (źródło: opracowanie własne)
Strona 17

Tabela 3.9 Tabela wymagań funkcjonalnych - ubranie postaci (źródło: opracowanie własne)
Strona 18

Tabela 3.10 Tabela wymagań funkcjonalnych - wylogowanie się (źródło: opracowanie własne)
Strona 18

Tabela 3.11 Tabela wymagań funkcjonalnych - odpowiadanie na pytania (źródło: opracowanie własne)
Strona 19

Tabela 3.12 Tabela wymagań funkcjonalnych - dodaj nowe funkcje (źródło: opracowanie własne)
Strona 19

Tabela 3.13 Tabela wymagań funkcjonalnych - dodaj nowy przedmiot do sklepu (źródło: opracowanie własne)
Strona 20

Tabela 6.1 Testy manualne (źródło: opracowanie własne)
Strona 174

Streszczenie

W naszej pracy inżynierskiej zajęliśmy się zaprojektowaniem i stworzeniem internetowej gry przygodowej wspomagającej edukację wczesnoszkolną. W pierwszym rozdziale dokonaliśmy przeglądu aktualnego stanu wiedzy na temat gier edukacyjnych, omawiając mechanizmy dydaktyczne, wpływ gier na rozwój dzieci oraz analizując dostępne na rynku rozwiązania i wykorzystywane technologie. Następnie określiliśmy cel naszej pracy, którym było stworzenie interaktywnej aplikacji edukacyjnej z elementami grywalizacji, a także sprecyzowaliśmy założenia projektowe i podział obowiązków w naszym zespole.

W kolejnych etapach projektu zastosowaliśmy metodykę Kanban oraz diagram Gantta, co pozwoliło nam efektywnie zarządzać zadaniami i harmonogramem pracy. Proces realizacji obejmował projektowanie interfejsu, implementację backendu i frontendu oraz testowanie aplikacji. W pracy szczegółowo opisaliśmy wymagania funkcjonalne naszego systemu, uwzględniając moduły rejestracji, personalizacji postaci, system nagród oraz quizy edukacyjne.

Przygotowaliśmy również diagramy klas i przypadków użycia, które obrazują strukturę stworzonej przez nas aplikacji. W końcowej części pracy skupiliśmy się na testowaniu — przeprowadziliśmy zarówno testy manualne, jak i automatyczne przy użyciu Postmana, co pozwoliło nam wykryć i wyeliminować błędy oraz potwierdzić stabilność systemu.

Podsumowując, nasza praca potwierdza, że wykorzystanie gier edukacyjnych może skutecznie zwiększać motywację dzieci do nauki i poprawiać ich wyniki. Stworzona przez nas aplikacja internetowa SQULIO łączy elementy grywalizacji i interaktywnego nauczania, dzięki czemu stanowi atrakcyjne i skuteczne narzędzie edukacyjne.

Abstract

In our engineering thesis, we focused on designing and developing an online adventure game to support early school education. In the first chapter, we reviewed the current state of knowledge on educational games, discussing didactic mechanisms, the impact of games on children's development, and analyzing available market solutions and applied technologies. We then defined the goal of our work, which was to create an interactive educational application incorporating gamification elements, as well as specifying project assumptions and the division of responsibilities within our team.

In the following project stages, we implemented the Kanban methodology and Gantt charts, allowing us to effectively manage tasks and project timelines. The development process included interface design, backend and frontend implementation, and application testing. In our thesis, we provided a detailed description of the functional requirements of our system, including registration modules, character customization, reward systems, and educational quizzes.

We also prepared class and use case diagrams to illustrate the structure of the application we created. In the final part of our work, we focused on testing — conducting both manual and automated tests using Postman, which enabled us to detect and eliminate errors and confirm the system's stability.

In conclusion, our study confirms that educational games can effectively increase children's motivation to learn and improve their academic performance. The web application we developed, SQULIO, successfully combines gamification and interactive learning, making it an engaging and effective educational tool.