Collegium Witelona Uczelnia Państwowa w Legnicy
Wydział Nauk Technicznych i Ekonomicznych
Kierunek: Informatyka

Projektowanie i programowanie systemów internetowych I

`Questify`

Krzysztof Kozyra, indeks: 43853
Tomasz Rebizant, indeks: 43877
Oskar Kuklewski, indeks: 43858

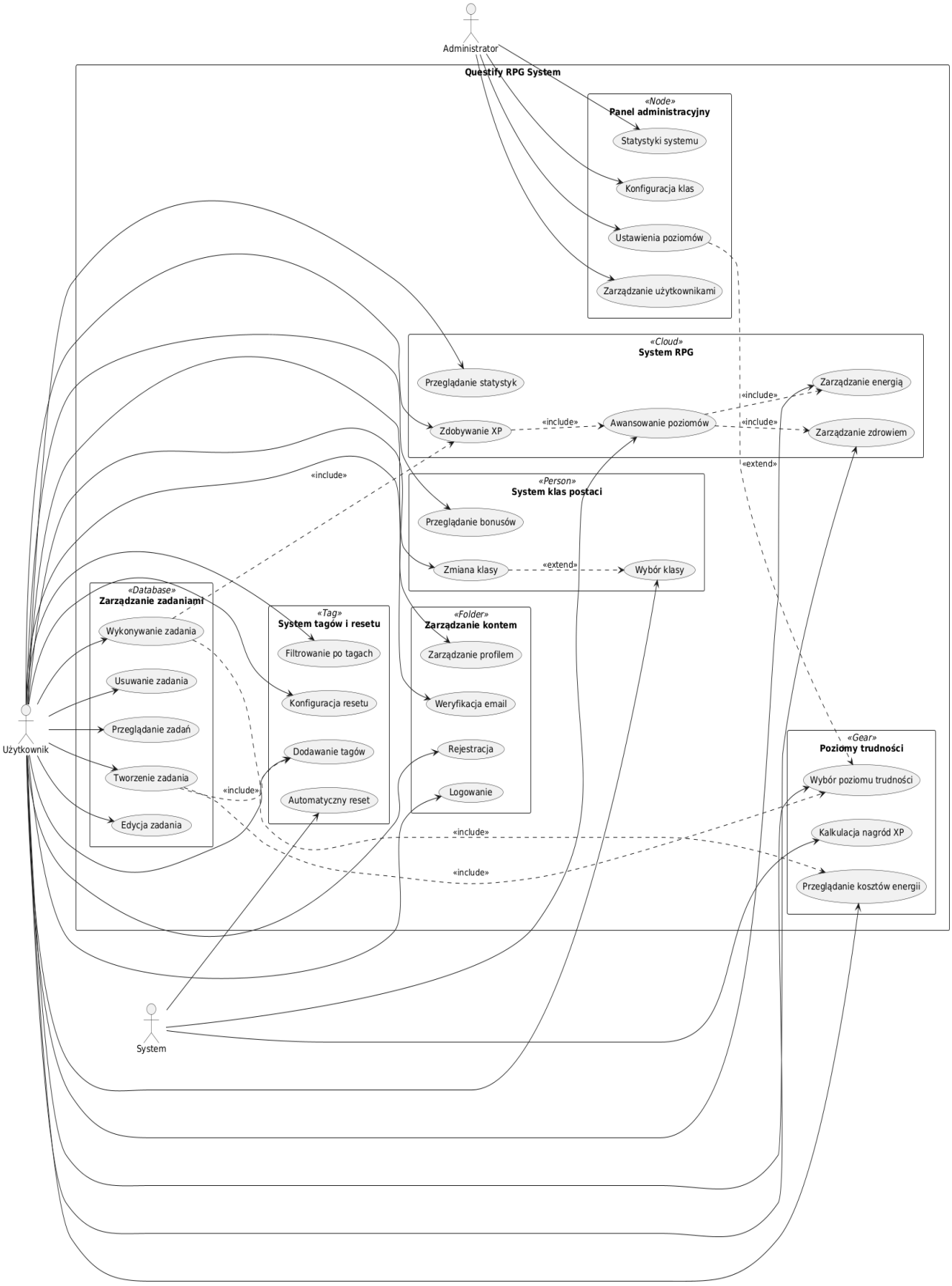
Prowadzący
mgr inż. Krzysztof Rewak

Spis treści

1 Cel projektu

Questify został zaprojektowany jako kompleksowy system zarządzania zadaniami, który łączy w sobie elementy produktywności z mechanikami znanymi z gier RPG. Głównym celem systemu jest stworzenie angażującego środowiska, które motywuje użytkowników do systematycznego wykonywania zadań poprzez:

- Implementację systemu zarządzania zadaniami opartego na trzech głównych typach: nawykach, zadaniach dziennych i zadaniach jednorazowych
- Wprowadzenie systemu motywacyjnego wykorzystującego mechanikę doświadczenia i poziomów
- Stworzenie systemu klas postaci, który dodaje elementy RPG do codziennej organizacji zadań
- Umożliwienie śledzenia postępów i statystyk użytkowników
- Wprowadzenie systemu tagów do efektywnej kategoryzacji zadań



Rysunek 1: Diagram przypadków użycia

1.1 Opis funkcjonalny systemu

Aplikacja Questify pozwala użytkownikom na:

1.1.1 Efektywne zarządzanie czasem poprzez system zadań

Questify oferuje kompleksowy system zarządzania zadaniami podzielony na trzy główne kategorie. Nawyki (Habits) pozwalają na tworzenie powtarzalnych zadań z własnym harmonogramem, co pomaga w budowaniu rutyny. Zadania dzienne (Dailies) są resetowane według ustalonej frekwencji, zachęcając do regularnej aktywności. Zadania do wykonania (Todos) umożliwiają zarządzanie pojedynczymi zadaniami z określonym terminem. Każde zadanie można dostosować pod względem trudności, częstotliwości i nagrody, co pozwala na elastyczne planowanie i organizację czasu.

1.1.2 Motywację do działania poprzez elementy grywalizacji

System motywacyjny Questify wykorzystuje elementy gier RPG, aby zachęcić użytkowników do regularnej aktywności. Każde wykonane zadanie przynosi punkty doświadczenia (XP), które wpływają na poziom postaci. System zdrowia i energii dodaje strategicznego wymiaru do zarządzania zadaniami. Użytkownicy mogą zdobywać nagrody za konsekwentne wykonywanie zadań, a system mnożników doświadczenia zachęca do podejmowania trudniejszych wyzwań. Cytaty motywacyjne i system osiągnięć dodatkowo wspierają motywację do działania.

1.1.3 Śledzenie postępów i osiągnięć

Aplikacja oferuje szczegółowy system śledzenia postępów. Użytkownicy mogą monitorować swoje statystyki, w tym poziom doświadczenia, liczbę wykonanych zadań, serię dni z rzędu (streak) oraz ogólny postęp. Historia aktywności pokazuje wszystkie wykonane zadania i zdobyte nagrody, pozwalając na analizę własnych osiągnięć. System osiągnięć nagradza za określone kamienie milowe, takie jak wykonanie określonej liczby zadań czy utrzymanie serii dni.

1.1.4 Organizację codziennych obowiązków

Dashboard aplikacji zapewnia przejrzysty widok na wszystkie zadania i obowiązki. Zadania są automatycznie kategoryzowane według typu i priorytetu. System tagów pozwala na dodatkową organizację zadań według własnych kategorii. Kalendarz i harmonogram tygodniowy pomagają w planowaniu zadań, a system przypomnień informuje o zbliżających się terminach. Możliwość filtrowania i sortowania zadań ułatwia zarządzanie codziennymi obowiązkami.

1.1.5 Rozwój osobisty poprzez system poziomów i nagród

Questify oferuje system rozwoju postaci inspirowany grami RPG. Użytkownicy mogą wybrać klasę postaci, która wpływa na ich statystyki i możliwości. System poziomów pozwala na stopniowy rozwój, a każdy nowy poziom odblokowuje nowe możliwości i bonusy. Nagrody za wykonane zadania motywują do dalszego rozwoju, a system klas dodaje strategicznego wymiaru do zarządzania zadaniami.

1.1.6 Budowanie nawyków poprzez regularne zadania

System nawyków w Questify jest zaprojektowany, aby pomóc w budowaniu trwałych zmian behawioralnych. Użytkownicy mogą tworzyć nawyki z określoną częstotliwością i śledzić swój postęp. System nagród za konsekwencję zachęca do regularnego wykonywania zadań, a statystyki pokazują długoterminowy postęp. Możliwość ustawienia harmonogramu tygodniowego pomaga w utrzymaniu regularności.

1.1.7 Analizę własnej produktywności

Aplikacja oferuje zaawansowane narzędzia analityczne do śledzenia produktywności. Użytkownicy mogą analizować swoje statystyki wykonania zadań, trendy w produktywności i obszary wymagające poprawy. System raportów generuje podsumowania tygodniowe i miesięczne, pokazując postępy i osiągnięcia. Możliwość porównywania statystyk w czasie pomaga w identyfikacji wzorców i optymalizacji produktywności.

1.1.8 Personalizację doświadczenia użytkownika

Aplikacja oferuje możliwość tworzenia własnych tagów pozwala na indywidualne podejście do organizacji. System ustawień pozwala na dostosowanie aplikacji do własnych potrzeb i preferencji.

1.1.9 Bezpieczne przechowywanie i synchronizację danych

Questify zapewnia bezpieczne przechowywanie danych użytkownika poprzez zaawansowany system autentykacji i szyfrowania. Automatyczna synchronizacja danych między urządzeniami zapewnia spójność informacji. System backupów chroni przed utratą danych, a kontrola dostępu gwarantuje prywatność użytkownika. Regularne aktualizacje zabezpieczeń zapewniają najwyższy poziom bezpieczeństwa danych.

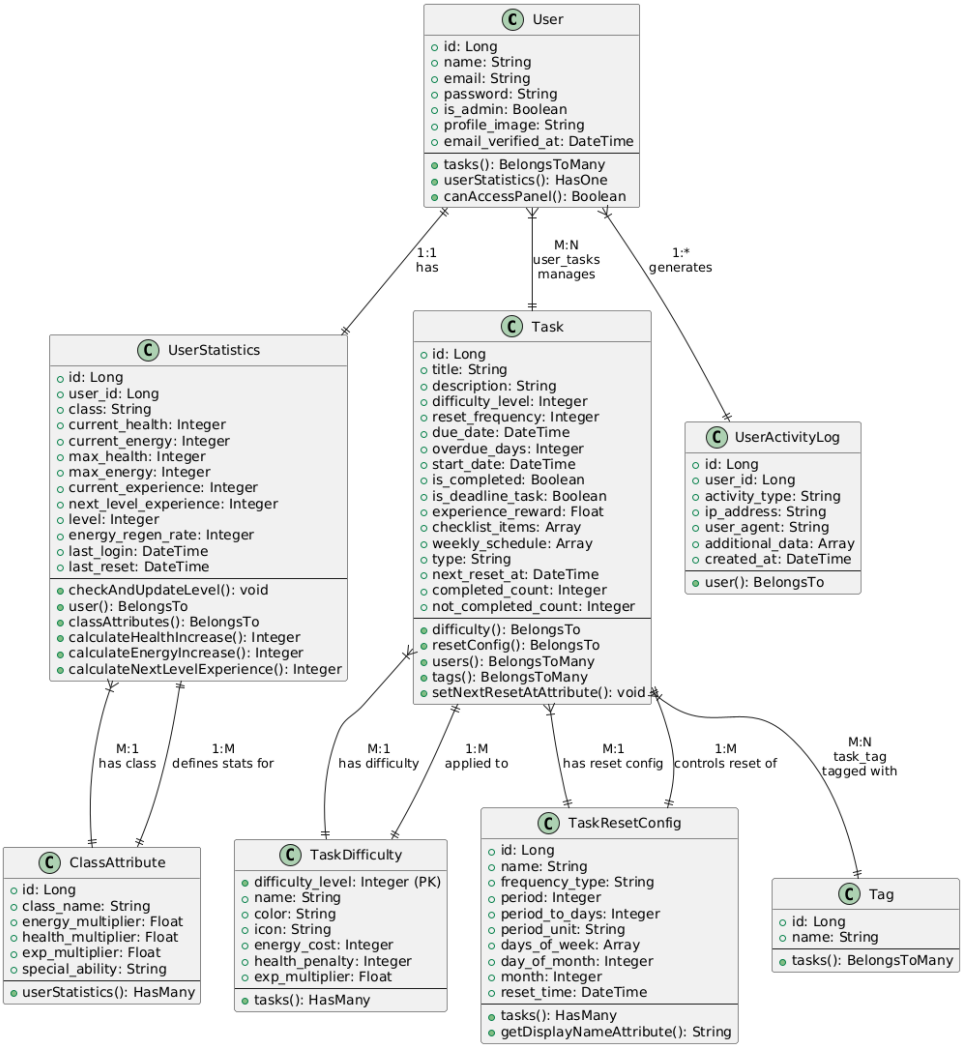
2 Opis technologiczny

Tabela przedstawia zestawienie użytych technologii:

Technologia	Opis wykorzystania
Backend i logika aplikacji	
Laravel (MVC)	Framework backendowy oparty na architekturze Model-View-Controller, odpowiedzialny za logikę serwera i tworzenie API
PHP (8.2)	Język programowania wykorzystywany po stronie serwera
Eloquent ORM	Narzędzie mapowania relacyjno-obiektowego zintegrowane z Laravel
Laravel Sanctum	Mechanizm uwierzytelniania i autoryzacji dla API
Laravel Breeze	Wstępna konfiguracja systemu logowania i rejestracji użytkownika
Frontend i interfejs użytkownika	
Vue.js	Biblioteka JavaScript do budowania dynamicznych komponentów interfejsu (z użyciem Vite i pluginu Vue)
Tailwind CSS	Framework CSS wspomagający tworzenie responsywnych interfejsów
NPM	Menedżer pakietów JavaScript wykorzystywany do zarządzania zależnościami frontendowymi
Axios	Biblioteka do wykonywania asynchronicznych żądań HTTP (np. do API)
Bazy danych i przechowywanie danych	
PostgreSQL	Główna relacyjna baza danych aplikacji
Redis	Wykorzystywany do przechowywania sesji oraz jako cache
JSON	Format do przechowywania plików lokalizacyjnych
Zarządzanie środowiskiem i narzędzia developerskie	
Docker Compose	Narzędzie do definiowania i uruchamiania środowiska wielokontenerowego (np. bazy danych, backend, frontend)
Composer	Menedżer zależności dla PHP
Mailpit	Lokalny serwer SMTP do testowania funkcji e-mailowych

Tabela 1: Zgrupowane technologie użyte w projekcie

Powyższe technologie zostały dobrane z myślą o kompleksowej realizacji funkcjonalności aplikacji webowej. Laravel odpowiada za solidne API oraz logikę serwera, natomiast Vue.js i Tailwind CSS zapewniają nowoczesny, responsywny oraz interaktywny interfejs użytkownika. Docker Compose umożliwia spójną konfigurację środowiska deweloperskiego, co znacząco ułatwia wdrażanie i rozwój aplikacji.



Rysunek 2: Diagram klas

3 Wdrożone zagadnienia kwalifikacyjne

Lp.	Zagadnienie	Opis wdrożenia
1	Framework MVC	Laravel 11 implementuje wzorzec Model-View-Controller. Modele (Task, User), Kontrolery (TaskController), Widoki (Vue.js komponenty).
2	Framework CSS	Tailwind CSS jako główny framework CSS. Komponenty UI w resources/js/Components z responsywnym designem.
3	Baza danych	PostgreSQL z migracjami w `database/migrations`. Tabele: `users`, `tasks`, `user_statistics`, `task_difficulties`, `task_reset_configs`.
4	Cache	Redis do przechowywania danych dashboardu i statystyk użytkownika. Automatyczna synchronizacja co `SYNC_INTERVAL`.
5	Dependency manager	Composer dla PHP (composer.json) i npm dla JavaScript (package.json). Zarządzanie zależnościami frontend i backend.
6	HTML	Struktura HTML generowana przez komponenty Vue.js. Szablony w resources/js/Components i resources/js/Pages.
7	CSS	Stylowanie przez Tailwind CSS. Własne style w komponentach Vue.js. Responsywny design dla różnych urządzeń.
8	JavaScript	Vue.js z TypeScript. Komponenty w resources/js/Components. Interaktywne elementy UI i zarządzanie stanem.
9	Routing	Laravel routing w routes/web.php. Pretty URLs dla zadań, profilu, dashboardu. Inertia.js do integracji frontend-backend.
10	ORM	Eloquent ORM do komunikacji z bazą danych. Relacje w modelach (belongsTo, hasMany, belongsToMany).
11	Uwierzytelnianie	Laravel Sanctum oraz Laravel Breeze do autentykacji/autoryzacji . System logowania, rejestracji i weryfikacji email. Middleware auth i verified.
12	Lokalizacja	System tłumaczeń przez useTranslation composable. Przełącznik języka w LanguageSwitcher.vue. Lokalizacja pobierana z plików /lang/*.json
13	Mailing	Laravel Mail do wysyłania emaili (weryfikacja, reset hasła). Konfiguracja w config/mail.php. MailPit do lokalnego testowania wysyłania e-maili
14	Formularze	Formularze Vue.js (TextInput, Checkbox, Dropdown). Walidacja po stronie klienta i serwera. Wysyłanie danych za pomocą FormData
15	Asynchroniczne interakcje	Axios do asynchronicznych zapytań. Automatyczna synchronizacja dashboardu.
16	Konsumpcja API	Wykorzystanie zewnętrznych API do pobierania cytatów motywacyjnych i ich translacji.
17	Publikacja API	REST API przez Laravel dla zadań, statystyk i profilu użytkownika. Endpointy zabezpieczone przez Sanctum.
18	RWD	Responsywny design przez Tailwind CSS. Breakpointy dla mobile, tablet i desktop. Komponenty dostosowują się do rozmiaru ekranu.
19	Logger	UserActivityLogger do śledzenia działań użytkownika. Logi w storage/logs a także w bazie danych.
20	Deployment	Docker dla konteneryzacji (docker-compose.yaml). CI/CD przez GitHub Actions. Automatyczne testy i deployment. Wdrożenie przez VPS Hetzner

Tabela 2: Zestawienie zagadnień kwalifikacyjnych i ich implementacji w projekcie Questify

4 Struktura projektu

Projekt Questify został zorganizowany zgodnie z architekturą Model-View-Controller (MVC) oraz zasadami framework-ów Laravel i Vue.js. Struktura katalogów zapewnia modularność, czytelność kodu oraz łatwość utrzymania aplikacji.

4.1 Główne katalogi aplikacji

4.1.1 Katalog backend (app/)

Katalog `app/` zawiera główną logikę aplikacji backendowej zorganizowaną w następujące podfoldery:

Katalog	Opis
`Http/Controllers/`	Kontrolery aplikacji obsługujące żądania HTTP
`Models/`	Modele danych reprezentujące encje systemu
`Services/`	Serwisy biznesowe zawierające logikę aplikacji
`Providers/`	Dostawcy usług Laravel
`Console/`	Komendy konsolowe Artisan
`Mail/`	Klasy do obsługi wysyłania emaili
`Filament/`	Konfiguracja panelu administracyjnego

Kluczowe kontrolery:

- `TaskController.php` - zarządzanie zadaniami (CRUD, wykonywanie)
- `DashboardController.php` - logika głównego dashboardu
- `ProfileController.php` - zarządzanie profilem użytkownika
- `Auth/` - kontrolery autentykacji i autoryzacji

Główne modele danych:

- `User.php` - model użytkownika systemu
- `Task.php` - model zadań z różnymi typami
- `UserStatistics.php` - statystyki RPG użytkownika
- `TaskDifficulty.php` - poziomy trudności zadań
- `TaskResetConfig.php` - konfiguracja resetowania zadań
- `ClassAttribute.php` - atrybuty klas postaci
- `Tag.php` - etykiety zadań
- `UserActivityLog.php` - logi aktywności użytkownika

Serwisy biznesowe:

- `TaskService.php` - logika biznesowa zarządzania zadaniami
- `UserActivityLogger.php` - logowanie aktywności użytkowników
- `TranslationService.php` - zarządzanie tłumaczeniami

4.1.2 Katalog frontend (resources/)

Katalog `resources/` zawiera wszystkie zasoby frontendowe aplikacji:

Katalog	Opis
`js/`	Kod JavaScript/TypeScript aplikacji
`css/`	Style CSS i pliki Tailwind
`lang/`	Pliki tłumaczeń w różnych językach
`views/`	Szablony Blade (głównie app.blade.php)

Struktura katalogu JavaScript:

4.1.3 Katalogi systemowe

Katalog	Opis
`database/`	Migracje, seedy i fabryki bazy danych
`routes/`	Definicje tras aplikacji (web.php, api.php)
`config/`	Pliki konfiguracyjne Laravel
`tests/`	Testy jednostkowe i funkcjonalne
`storage/`	Przechowywanie danych, logów i cache
`public/`	Pliki dostępne publicznie (obrazy, CSS, JS)
`vendor/`	Zależności PHP instalowane przez Composer
`node_modules/`	Zależności JavaScript instalowane przez npm

4.2 Przepływ danych w aplikacji

sectionStruktura projektu

Projekt Questify został zorganizowany zgodnie z architekturą Model-View-Controller (MVC) oraz zasadami framework-ów Laravel i Vue.js. Struktura katalogów zapewnia modularność, czytelność kodu oraz łatwość utrzymania aplikacji.

4.3 Główne katalogi aplikacji

4.3.1 Katalog backend (app/)

Katalog `app/` zawiera główną logikę aplikacji backendowej zorganizowaną w następujące podfoldery:

Katalog	Opis
`Http/Controllers/`	Kontrolery aplikacji obsługujące żądania HTTP
`Models/`	Modele danych reprezentujące encje systemu
`Services/`	Serwisy biznesowe zawierające logikę aplikacji
`Providers/`	Dostawcy usług Laravel
`Console/`	Komendy konsolowe Artisan
`Mail/`	Klasy do obsługi wysyłania emaili
`Filament/`	Konfiguracja panelu administracyjnego

Kluczowe kontrolery:

- `TaskController.php` - zarządzanie zadaniami (CRUD, wykonywanie)

- `DashboardController.php` - logika głównego dashboardu
- `ProfileController.php` - zarządzanie profilem użytkownika
- `Auth/` - kontrolery autentykacji i autoryzacji

Główne modele danych:

- `User.php` - model użytkownika systemu
- `Task.php` - model zadań z różnymi typami
- `UserStatistics.php` - statystyki RPG użytkownika
- `TaskDifficulty.php` - poziomy trudności zadań
- `TaskResetConfig.php` - konfiguracja resetowania zadań
- `ClassAttribute.php` - atrybuty klas postaci
- `Tag.php` - etykiety zadań
- `UserActivityLog.php` - logi aktywności użytkownika

Serwisy biznesowe:

- `TaskService.php` - logika biznesowa zarządzania zadaniami
- `UserActivityLogger.php` - logowanie aktywności użytkowników
- `TranslationService.php` - zarządzanie tłumaczeniami

4.3.2 Katalog frontend (resources/)

Katalog `resources/` zawiera wszystkie zasoby frontendowe aplikacji:

Katalog	Opis
`js/`	Kod JavaScript/TypeScript aplikacji
`css/`	Style CSS i pliki Tailwind
`lang/`	Pliki tłumaczeń w różnych językach
`views/`	Szablony Blade (głównie app.blade.php)

Struktura katalogu JavaScript:

```
resources/js/
|-- Components/           # Komponenty Vue.js
|   |-- UI/              # Podstawowe komponenty interfejsu
|   |-- Habits/          # Komponenty dla nawyków
|   |-- Dailies/         # Komponenty dla zadań dziennych
|   |-- Todos/           # Komponenty dla zadań do wykonania
|   +-- Shared/          # Współdzielone komponenty
|-- Pages/               # Strony Inertia.js
|   |-- Dashboard.vue    # Główny dashboard aplikacji
|   |-- Settings.vue     # Strona ustawień
|   +-- Auth/            # Strony autentykacji
|-- Composables/         # Kompozycje Vue.js
|   |-- useTranslation.js # Zarządzanie tłumaczeniami
|   +-- useNotification.js # System powiadomień
+-- app.js               # Główny plik aplikacji
```

4.3.3 Katalogi systemowe

Katalog	Opis
`database/`	Migracje, seedy i fabryki bazy danych
`routes/`	Definicje tras aplikacji (web.php, api.php)
`config/`	Pliki konfiguracyjne Laravel
`tests/`	Testy jednostkowe i funkcjonalne
`storage/`	Przechowywanie danych, logów i cache
`public/`	Pliki dostępne publicznie (obrazy, CSS, JS)
`vendor/`	Zależności PHP instalowane przez Composer
`node_modules/`	Zależności JavaScript instalowane przez npm

4.4 Przepływ danych w aplikacji

4.4.1 Architektura komunikacji

Aplikacja wykorzystuje architekturę full-stack z komunikacją między frontendem a backendem przez Inertia.js. Przepływ danych odbywa się według następującego schematu:

1	**Użytkownik**	2	**Vue.js**	3
Interakcja	Komponenty Vue.js	Inertia.js	Laravel Backend	Odpowiedź

1. **Użytkownik** wykonuje akcję w interfejsie (kliknięcie, formularz)
2. **Komponenty Vue.js** przechwytują zdarzenie i wywołują akcję
3. **Inertia.js** wysyła żądanie HTTP do backendu Laravel
4. **Laravel Backend** przetwarza żądanie i zwraca dane JSON
5. **Frontend** aktualizuje interfejs na podstawie otrzymanych danych

4.4.2 Przepływ obsługi żądań

W aplikacji Laravel żądania HTTP są przetwarzane przez następujące warstwy:

Krok	Warstwa	Odpowiedzialność
1	HTTP Request	Żądanie od klienta
2	Middleware	Autentykacja, autoryzacja, walidacja
3	Router	Kierowanie do odpowiedniego kontrolera
4	Controller	Obsługa żądania, delegacja do serwisów
5	Service	Logika biznesowa aplikacji
6	Model	Dostęp do danych, operacje na bazie
7	Database	Przechowywanie i pobieranie danych
8	HTTP Response	Zwrócenie wyniku do klienta

4.4.3 System zarządzania zadaniami

Proces wykonywania zadania w systemie RPG przebiega następująco:

1. **TaskController** otrzymuje żądanie wykonania zadania
2. **TaskService** sprawdza możliwość wykonania (energia, zdrowie)
3. **Task Model** aktualizuje status zadania w bazie danych
4. **UserStatistics** otrzymuje punkty doświadczenia za zadanie
5. **RPG Logic** sprawdza możliwość awansu na wyższy poziom
6. **Model Events** automatycznie logują aktywność użytkownika
7. **Response** zawiera zaktualizowane statystyki i informacje o nagrodach

4.5 Pliki konfiguracyjne

4.5.1 Konfiguracja główna

Plik	Przeznaczenie
`composer.json`	Zależności PHP i autoloader
`package.json`	Zależności JavaScript i skrypty npm
`.env`	Zmienne środowiskowe aplikacji
`docker-compose.yaml`	Konfiguracja kontenerów Docker

4.5.2 Konfiguracja frontend

Plik	Przeznaczenie
`vite.config.js`	Konfiguracja bundlera Vite
`tailwind.config.js`	Konfiguracja Tailwind CSS
`tsconfig.json`	Konfiguracja TypeScript
`eslint.config.js`	Konfiguracja lintera ESLint

4.6 Architektura modułowa

4.6.1 Separacja odpowiedzialności

Projekt implementuje zasady SOLID poprzez następującą organizację:

- **Models** - odpowiadają za dostęp do danych i relacje
- **Controllers** - obsługują żądania HTTP i delegują logikę
- **Services** - zawierają logikę biznesową aplikacji
- **Middleware** - obsługują cross-cutting concerns (auth, cors)
- **Components** - enkapsulują funkcjonalność UI
- **Composables** - udostępniają wielokrotnego użytku logikę Vue.js

4.7 Struktura bazy danych

4.7.1 Główne tabele

Tabela	Przeznaczenie
`users`	Dane użytkowników systemu
`user_statistics`	Statystyki RPG użytkowników
`tasks`	Zadania w systemie
`task_difficulties`	Poziomy trudności zadań
`task_reset_configs`	Konfiguracje resetowania zadań
`class_attributes`	Atrybuty klas postaci
`tags`	Etykiety do kategoryzacji zadań
`user_activity_logs`	Logi aktywności użytkowników
`user_tasks`	Tabela pivot użytkownik-zadanie
`task_tag`	Tabela pivot zadanie-etykieta

4.8 Zalety struktury projektu

Zastosowana organizacja projektu zapewnia:

1. **Czytelność kodu** - logiczne grupowanie plików według funkcjonalności
2. **Modularność** - łatwe dodawanie nowych funkcji bez wpływu na istniejące
3. **Łatwość utrzymania** - jasne rozgraniczenie odpowiedzialności
4. **Skalowalność** - możliwość rozszerzania aplikacji
5. **Testowalnolość** - struktura ułatwiająca pisanie testów
6. **Współpracę zespołową** - przejrzysta organizacja dla developerów

5 Lokalne uruchomienie systemu

5.1 Wymagania systemowe

Aby uruchomić aplikację Questify w środowisku lokalnym, wymagane są następujące narzędzia i zależności:

- Linux z zainstalowanym Docker (lub Windows z dodatkową konfiguracją)
- Bash shell
- GNU Make w wersji 4.x
- PHP 8.2 z następującymi modułami:
  - xml
  - curl
  - zip
  - cli
- Composer w wersji 2.5.x
- Node.js oraz npm
- Git

Instalacja zależności w systemie Debian/Ubuntu:

```
sudo apt update
sudo apt install -y make php php8.2-cli php8.2-xml php8.2-curl \
    php8.2-zip composer npm git
```

Listing 1: Instalacja wymaganych pakietów

5.2 Konfiguracja środowiska deweloperskiego

5.2.1 Przygotowanie Docker

Dodanie użytkownika do grupy docker w celu uruchamiania kontenerów bez uprawnień sudo:

```
sudo usermod -aG docker $USER
newgrp docker
docker run hello-world
```

Listing 2: Konfiguracja Docker

5.2.2 Klonowanie repozytorium

```
git clone https://github.com/krzysztofkozyra021/Questify.git
cd Questify
```

Listing 3: Pobieranie kodu źródłowego

5.2.3 Inicjalizacja projektu

Opcja 1 - Automatyczna inicjalizacja (Linux):

```
make init
make stop
```

Listing 4: Automatyczna inicjalizacja

Opcja 2 - Manualna konfiguracja:

```
cp .env.example .env
```

Listing 5: Manualna konfiguracja środowiska

5.2.4 Przełączenie na gałąź deweloperską

```
git checkout develop
```

Listing 6: Przełączenie na branch develop

5.2.5 Instalacja zależności

```
npm install
composer install
```

Listing 7: Instalacja zależności frontend i backend

5.2.6 Konfiguracja bazy danych

```
make shell
php artisan migrate:fresh --seed
exit
```

Listing 8: Przygotowanie bazy danych

5.2.7 Uruchomienie serwera deweloperskiego

```
make dev
```

Listing 9: Start serwera deweloperskiego

Po wykonaniu powyższych kroków aplikacja będzie dostępna pod adresem: `http://localhost:63851`

5.3 Dostępne polecenia

5.3.1 Polecenia Make

Polecenie	Opis
`make init`	Inicjalizuje projekt: sprawdza plik env, buduje kontenery, uruchamia je i konfiguruje bazę testową
`make build`	Buduje kontenery Docker
`make run`	Uruchamia kontenery Docker w trybie detached
`make stop`	Zatrzymuje działające kontenery Docker
`make restart`	Restartuje kontenery Docker
`make shell`	Otwiera bash shell w kontenerze aplikacji jako bieżący użytkownik
`make shell-root`	Otwiera bash shell w kontenerze aplikacji jako root
`make test`	Uruchamia testy backend
`make fix`	Sprawdza i naprawia formatowanie plików backend
`make analyse`	Uruchamia analizę Larastan dla plików backend
`make dev`	Uruchamia serwer deweloperski frontend
`make queue`	Uruchamia worker kolejki Laravel
`make create-test-db`	Tworzy bazę danych testową

5.3.2 Polecenia wewnątrz kontenera

Przed wykonaniem poniższych poleceń należy wejść do kontenera aplikacji:

```
make shell
```

Polecenie	Opis
`composer <command>`	Polecenia Composer
`composer test`	Uruchamia testy backend
`composer analyse`	Uruchamia analizę Larastan dla plików backend
`composer cs`	Sprawdza formatowanie plików backend
`composer csf`	Sprawdza i naprawia formatowanie plików backend
`php artisan <command>`	Polecenia Artisan
`npm run dev`	Kompiluje i uruchamia hot-reload dla development
`npm run build`	Kompiluje i minifikuje dla produkcji
`npm run lint`	Sprawdza formatowanie plików frontend
`npm run lintf`	Sprawdza i naprawia formatowanie plików frontend
`npm run tsc`	Uruchamia sprawdzanie TypeScript

5.4 Konfiguracja kontenerów

Aplikacja wykorzystuje następujące kontenery Docker:

Serwis	**Nazwa kontenera**	**Port**
`app`	`Questify-app-dev`	63851
`database`	`Questify-db-dev`	63853
`redis`	`Questify-redis-dev`	63852
`mailpit`	`Questify-mailpit-dev`	63854

5.5 Rozwiązywanie problemów

W przypadku problemów z uruchomieniem aplikacji, zalecane jest sprawdzenie następujących elementów:

1. Upewnienie się, że wszystkie wymagane zależności są zainstalowane
2. Sprawdzenie, czy porty nie są zajęte przez inne aplikacje
3. Weryfikacja uprawnień do Docker (możliwość uruchamiania bez sudo)
4. Sprawdzenie logów kontenerów: `docker logs Questify-app-dev`
5. Restart kontenerów: `make restart`

6 Zdalne uruchomienie systemu

6.1 Wymagania systemowe

- Docker i Docker Compose
- Node.js (wersja LTS)
- PHP 8.1 lub nowszy
- PostgreSQL 16.3
- Redis 7.2.5
- Nginx
- Make (dla automatyzacji procesów)

6.2 Konfiguracja środowiska produkcyjnego

6.2.1 Przygotowanie plików konfiguracyjnych

1. Skopiuj plik `.env.example` do `.env` i dostosuj zmienne środowiskowe:
   - `APP_ENV=production`
   - `APP_DEBUG=false`
   - `DB_USERNAME` - nazwa użytkownika bazy danych
   - `DB_PASSWORD` - hasło do bazy danych
   - `DB_DATABASE` - nazwa bazy danych
   - `DOCKER_APP_HOST_PORT` - port dla aplikacji
   - `DOCKER_DATABASE_HOST_PORT` - port dla bazy danych
   - `DOCKER_REDIS_HOST_PORT` - port dla Redis
   - `DOCKER_MAILPIT_DASHBOARD_HOST_PORT` - port dla Mailpit
2. Dostosuj konfigurację Docker Compose:
   - Zmień porty w `docker-compose.yaml` na odpowiednie dla środowiska produkcyjnego
   - Usuń lub dostosuj konfigurację Xdebug
   - Ustaw odpowiednie limity zasobów dla kontenerów
   - Dostosuj konfigurację sieci i wolumenów

6.3 Proces wdrożenia

6.3.1 Inicjalizacja środowiska

1. Wykonaj inicjalizację środowiska:
   - `make init` - pełna inicjalizacja środowiska
   - `make build` - budowa kontenerów
   - `make run` - uruchomienie kontenerów

6.3.2 Przygotowanie kodu

1. Przygotowanie kodu:
    - Wykonaj `composer install -no-dev -optimize-autoloader`
    - Wykonaj `npm install`
    - Wykonaj `npm run build`
    - Wykonaj `php artisan config:cache`
    - Wykonaj `php artisan route:cache`
    - Wykonaj `php artisan view:cache`
2. Uruchomienie kontenerów:
    - Wykonaj `docker-compose up -d`
    - Poczekaj na uruchomienie wszystkich serwisów
    - Wykonaj migracje bazy danych: `php artisan migrate -force`
3. Konfiguracja Nginx:
    - Dostosuj konfigurację Nginx w `environment/dev/app/nginx.conf`
    - Skonfiguruj SSL/TLS
    - Ustaw odpowiednie nagłówki bezpieczeństwa

6.4 Konfiguracja Nginx

6.4.1 Przykładowa konfiguracja nginx.conf

Poniżej znajduje się przykładowa konfiguracja Nginx dla środowiska produkcyjnego:

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 1024;
    multi_accept on;
}

http {
    # Podstawowe ustawienia
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    server_tokens off;
```

```
# MIME
include /etc/nginx/mime.types;
default_type application/octet-stream;

# Logi
access_log /var/log/nginx/access.log;
error_log /var/log/nginx/error.log;

# Gzip
gzip on;
gzip_vary on;
gzip_proxied any;
gzip_comp_level 6;
gzip_types text/plain text/css text/xml application/json
           application/javascript application/xml+rss
           application/atom+xml image/svg+xml;

# Bezpieczeństwo
add_header X-Frame-Options "SAMEORIGIN";
add_header X-XSS-Protection "1; mode=block";
add_header X-Content-Type-Options "nosniff";
add_header Referrer-Policy "strict-origin-when-cross-origin";
add_header Content-Security-Policy "default-src 'self';
    script-src 'self' 'unsafe-inline' 'unsafe-eval';
    style-src 'self' 'unsafe-inline';
    img-src 'self' data: https:;
    font-src 'self' data: https:;";

# Konfiguracja serwera
server {
    listen 80;
    listen [::]:80;
    server_name example.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    server_name example.com;
    root /application/public;

    # SSL
    ssl_certificate /etc/nginx/ssl/example.com.crt;
    ssl_certificate_key /etc/nginx/ssl/example.com.key;
    ssl_session_timeout 1d;
    ssl_session_cache shared:SSL:50m;
    ssl_session_tickets off;
```

```
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-EC
ssl_prefer_server_ciphers off;

# HSTS
add_header Strict-Transport-Security "max-age=63072000" always;

# Indeks
index index.php;

# Charset
charset utf-8;

# Logi
access_log /var/log/nginx/example.com.access.log;
error_log /var/log/nginx/example.com.error.log;

# Lokalizacja główna
location / {
    try_files $uri $uri/ /index.php?$query_string;
}

# PHP-FPM
location ~ \.php$ {
    fastcgi_pass unix:/var/run/php/php8.1-fpm.sock;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
    include fastcgi_params;
    fastcgi_buffers 16 16k;
    fastcgi_buffer_size 32k;
}

# Statyczne pliki
location ~* \.(jpg|jpeg|png|gif|ico|css|js|svg|woff|woff2|ttf|eot)$ {
    expires 30d;
    add_header Cache-Control "public, no-transform";
}

# Zabezpieczenia
location ~ /\. {
    deny all;
}

location ~ /\.(?!well-known).* {
    deny all;
}

# Limity
```

```
        client_max_body_size 100M;
        client_body_buffer_size 128k;
        client_header_buffer_size 1k;
        large_client_header_buffers 4 4k;
    }
}
```

6.4.2 Ważne elementy konfiguracji

- **SSL/TLS**:
  - Konfiguracja certyfikatów SSL
  - Wymuszenie HTTPS
  - Optymalne ustawienia bezpieczeństwa SSL
  - HSTS dla dodatkowego bezpieczeństwa
- **Wydajność**:
  - Włączone buforowanie
  - Optymalizacja Gzip
  - Konfiguracja worker_processes
  - Buforowanie statycznych plików
- **Bezpieczeństwo**:
  - Nagłówki bezpieczeństwa
  - Content Security Policy
  - Blokada dostępu do ukrytych plików
  - Limity rozmiaru żądań
- **PHP-FPM**:
  - Konfiguracja fastcgi
  - Optymalne bufory
  - Prawidłowa obsługa skryptów PHP

6.5 Zarządzanie środowiskiem

6.5.1 Komendy Make

- Podstawowe operacje:
  - `make init` - pełna inicjalizacja środowiska
  - `make build` - budowa kontenerów
  - `make run` - uruchomienie kontenerów
  - `make stop` - zatrzymanie kontenerów
  - `make restart` - restart kontenerów

- Operacje na kontenerach:
  - `make shell` - wejście do kontenera aplikacji jako użytkownik
  - `make shell-root` - wejście do kontenera aplikacji jako root
  - `make queue` - uruchomienie workera kolejki
  - `make dev` - uruchomienie serwera deweloperskiego
- Operacje na kodzie:
  - `make test` - uruchomienie testów
  - `make fix` - naprawa stylu kodu
  - `make analyse` - analiza kodu

6.6 Monitorowanie i utrzymanie

- Monitorowanie logów:
  - Logi aplikacji: `storage/logs/laravel.log`
  - Logi Nginx: `/var/log/nginx/`
  - Logi PHP-FPM: `/var/log/php-fpm/`
  - Logi kontenerów: `docker-compose logs`
- Zadania cron:
  - Skonfiguruj harmonogram zadań w `app/Console/Kernel.php`
  - Ustaw crona dla `php artisan schedule:run`
  - Monitoruj działanie workera kolejki: `make queue`
- Kopie zapasowe:
  - Regularne kopie bazy danych PostgreSQL
  - Kopie plików z katalogu `storage/app/public`
  - Kopie konfiguracji środowiska

6.7 Bezpieczeństwo

- Ustaw silne hasła dla wszystkich serwisów
- Skonfiguruj firewall
- Włącz HTTPS
- Regularnie aktualizuj zależności
- Monitoruj logi pod kątem podejrzanej aktywności
- Ustaw odpowiednie uprawnienia do plików i katalogów
- Skonfiguruj limity zasobów dla kontenerów
- Włącz tryb produkcyjny w konfiguracji PHP i Laravel

6.8 Rozwiązywanie problemów

- Podstawowe operacje diagnostyczne:
  - Sprawdź status kontenerów: `docker-compose ps`
  - Przejrzyj logi kontenerów: `docker-compose logs`
  - Sprawdź połączenie z bazą danych
  - Weryfikuj uprawnienia do katalogów
  - Sprawdź konfigurację PHP i Nginx
- Zaawansowane operacje:
  - Restart kontenerów: `make restart`
  - Przejście do powłoki kontenera: `make shell`
  - Sprawdzenie statusu workera kolejki
  - Weryfikacja konfiguracji środowiska
- Typowe problemy:
  - Problemy z uprawnieniami do katalogów
  - Konflikty portów
  - Problemy z pamięcią
  - Problemy z połączeniem do bazy danych
  - Problemy z kolejką zadań

7 Wnioski projektowe

7.1 Wybór technologii

Zastosowany stos technologiczny okazał się trafnym wyborem:

- **Laravel 11 + Vue.js 3** - zapewniły stabilne i wydajne środowisko rozwoju
- **Inertia.js** - skutecznie połączył frontend z backendem bez potrzeby osobnego API
- **Tailwind CSS** - przyspieszył stylizację i zapewnił spójny wygląd
- **Docker** - uprościł proces developmentu i zapewnił spójne środowisko

7.2 Funkcjonalności systemu

Zaimplementowane rozwiązania skutecznie realizują założone cele:

- **System RPG** - efektywnie motywuje użytkowników do regularnej aktywności
- **Trzy typy zadań** - zapewniają elastyczność w zarządzaniu czasem
- **Statystyki i poziomy** - skutecznie angażują użytkowników w długoterminową aktywność

7.3 Główne wyzwania i rozwiązania

Wyzwanie	Rozwiązanie
Zarządzanie stanem aplikacji	Vue.js Composables i reaktywne właściwości
Synchronizacja danych w czasie rzeczywistym	System cache i automatyczna synchronizacja
Responsywność na różnych urządzeniach	Tailwind CSS z breakpointami
Niepełna znajomość używanych technologii	Wsparcie przy pisaniu kluczowych funkcjonalności podpowiedziami otrzymanymi od modeli LLM

7.4 Mocne strony projektu

1. Nowoczesna, skalowalna architektura
2. Intuicyjny interfejs użytkownika
3. Efektywny system motywacji przez grywalizację
4. Stabilne środowisko technologiczne

7.5 Obszary do poprawy

1. Rozszerzenie testów automatycznych
2. Optymalizacja wydajności bazy danych
3. Rozwój funkcji społecznościowych

7.6 Rekomendacje na przyszłość

Dalszy rozwój aplikacji powinien skupić się na:

- Rozwoju aplikacji mobilnej
- Integracji z popularnymi kalendarzami
- Implementacji systemu wyzwań grupowych
- Rozszerzeniu analityki i raportowania

7.7 Podsumowanie

Projekt Questify udowodnił skuteczność połączenia nowoczesnych technologii webowych z elementami grywalizacji. Wybrane rozwiązania techniczne zapewniły stabilne środowisko rozwoju, a funkcjonalności RPG skutecznie motywują użytkowników do regularnej aktywności.

Architektura aplikacji umożliwia łatwe rozszerzanie funkcjonalności, co stanowi solidną podstawę do dalszego rozwoju systemu. Implementacja różnych typów zadań w połączeniu

z systemem poziomów i nagród tworzy kompleksowe narzędzie zarządzania czasem, które łączy funkcjonalność z motywacją.

Cały proces zapewnił grupie skuteczny rozwój i doświadczenie które może zostać wykorzystane w potencjalnie przyszłej karierze