Bazy danych

Projekt

Określenie i analiza projektu

Data wykonania: 02.05.2024 r. Kamil Pomykała L03

Tematyka i zakres projektu

Projekt skupia się na stworzenie systemu zarządzania kolekcją gier planszowych. Głównym celem projektu jest stworzenie bazy danych, która umożliwi katalogowanie gier planszowych, mechanik tych gier, recenzji oraz historii wydań.

System pozwala na zarządzanie kolekcją jednej osoby lub grupy domowników. W projekcie zostaną zaimplementowane dwie główne role użytkowników: administratora, który ma pełen dostęp do funkcji systemu, oraz zwykłego użytkownika, który może korzystać z podstawowych funkcji przeglądania bazy gier oraz ma możliwość dodania recenzji. Dzięki takiemu podejściu użytkownicy mogą bez obaw o stratę danych, zarządzać swoją kolekcją gier planszowych.

Projekt nazwałem **„GameVault”**. Nazwa nawiązuje do przechowywania gier w wirtualnym „skarbcu”.

Zagadnienia związane z tematem

- **Katalogowanie gier**: Należy opracować strukturę bazy danych, która pozwoli na przechowywanie informacji o grach planszowych, takich jak nazwa gry, gatunek, liczba graczy, czas trwania rozgrywki, opis (gdzie powinny znaleźć się dodatkowe informacje o grze).
- **Mechaniki gier**: System powinien umożliwiać przypisanie mechanik gry do danej gry planszowej. Mechaniki gier to zestaw reguł, które określają sposób rozgrywki. Każda gra może posiadać wiele mechanik.
- **Recenzje gier**: Użytkownicy systemu mogą dodawać recenzje gier planszowych. Recenzje powinny zawierać ocenę gry oraz opis. Każda gra może posiadać wiele recenzji.
- **Historia wydań**: System powinien przechowywać informacje o wydaniach danej gry planszowej. Historia wydań zawiera informacje o dacie wydania, wydawcy, numerze wydania oraz opisie gdzie można znaleźć dodatkowe informacje o wydaniu. Każda gra może posiadać wiele wydań.
- **Role użytkowników**: System powinien umożliwiać zarządzanie rolami użytkowników. Administrator ma pełen dostęp do funkcji systemu, natomiast zwykły użytkownik ma ograniczony dostęp do funkcji systemu.

Funkcje bazy danych i ich priorytety

1. **Przechowywanie informacji o grach planszowych**
   - Priorytet: ***Wysoki***
   - Opis: Najważniejsza funkcjonalność systemu, która umożliwia przechowywanie informacji o grach planszowych. Bez tej funkcjonalności projekt nie ma za dużego sensu.
2. **Przechowywanie informacji o mechanikach gier**
   - Priorytet: ***Średni***
   - Opis: Funkcjonalność umożliwiająca przypisanie mechanik gry do danej gry planszowej. Dzięki tej funkcjonalności opisy gier stają się bardziej kompleksowe. Każdy użytkownik z dostępem do bazy gier może dowiedzieć się, jakie mechaniki posiada dana gra i podejrzeć je podczas rozgrywki.

3. **Przechowywanie informacji o recenzjach gier**
   - Priorytet: ***Średni***
   - Opis: Funkcjonalność umożliwiająca dodawanie recenzji gier planszowych. Recenzje gier są ważne dla użytkowników, którzy chcą znaleźć informacje zwrotne na temat danej gry. Dzięki tej funkcjonalności użytkownicy mogą dzielić się swoimi opiniami na temat gier planszowych.
4. **Przechowywanie informacji o historii wydań gier**
   - Priorytet: ***Niski***
   - Opis: Funkcjonalność umożliwiająca przechowywanie informacji o wydaniach danej gry planszowej. Ma najniższy priorytet, ponieważ nie jest to funkcjonalność, która jest niezbędna do działania systemu. Jednakże, dzięki tej funkcjonalności użytkownicy mogą lepiej zarządzać swoją kolekcją gier planszowych.

Technologie i narzędzia

Technologie i rodzaj bazy danych

Projekt zostanie zrealizowany w oparciu o bazę danych **PostgreSQL**. Jest to jeden z najpopularniejszych silników bazodanowych, który oferuje zaawansowane funkcje i możliwości. PostgreSQL jest otwartoźródłowym systemem, co dla mnie - zwolennika takich rozwiązań, jest bardzo ważne. W projekcie wykorzystam wersję 16.2 PostgreSQL, która podczas pisania tego dokumentu jest najnowszą wersją stabilną. Użyję tego obrazu Dockerowego.

Do stworzenia systemu ORM wykorzystam język programowania **Rust**. Oprócz tego wykorzystam takie biblioteki jak:

- **Diesel**: ORM dla języka Rust, który umożliwia łatwe zarządzanie bazą danych PostgreSQL.
- **Clap**: Biblioteka do obsługi argumentów wiersza poleceń. Umożliwia łatwe tworzenie aplikacji konsolowych w języku Rust.
- **Serde i Serde_json**: Biblioteka do serializacji i deserializacji danych w języku Rust. Umożliwia łatwe przekształcanie danych do formatu JSON.
- **Inne biblioteki**, które mogą okazać się przydatne podczas implementacji systemu.

W projekcie wykorzystam wersję 1.77.2 języka Rust. Nie jest to najnowsza wersja języka, ale jest to wersja, która jest stabilna i sprawdzona. Program będzie kompilowany do wersji wykonywalnych pod systemy Windows i Linux więc nie jest potrzebny Docker. Podczas implementacji systemu będę korzystał z najnowszych wersji bibliotek, które są dostępne w repozytorium Cargo.

Narzędzia

Przygotowanie dokumentacji projektu zostanie zrealizowane w języku **Typst**. Typst to alternatywne narzędzie do LaTeX, które umożliwia tworzenie dokumentacji w sposób tekstowy.

Do stworzenia diagramów ERD użyję narzędzia **dbdiagram.io**. Jest to narzędzie online, które wykorzystywałem już w przeszłości. W szczególności podoba mi się fakt, że diagramy tworzy się w sposób tekstowy. Dzięki temu można skupić się na projektowaniu bazy danych, a nie na rysowaniu diagramów.

Lokalne i produkcyjne środowisko projektu zostanie zrealizowane w oparciu o kontenery **Docker** i narzędzie **Docker Compose**. Docker umożliwia łatwe zarządzanie środowiskami deweloperskimi oraz produkcyjnymi. Repozytoria Dockerowe zawierają gotowe obrazy PostgreSQL i Rust, które można zmodyfikować według własnych potrzeb. Docker Compose umożliwia zarządzanie wieloma kontenerami jednocześnie za pomocą jednego pliku konfiguracyjnego `docker-compose.yml`.

Do zarządzania bazą danych użyję narzędzia **DataGrip** firmy **JetBrains**. Jest to zaawansowane narzędzie do zarządzania bazą danych, które oferuje wiele funkcji ułatwiających pracę z bazą danych. JetBrains oferuje darmową licencję dla studentów, co jest dodatkowym atutem tego narzędzia.

Do implementacji systemu wykorzystam środowisko programistyczne **RustRover**. RustRover to nowe IDE firmy JetBrains, które oferuje wiele funkcji ułatwiających pracę z językiem Rust. RustRover oferuje integrację z Cargo, co umożliwia łatwe zarządzanie zależnościami w projekcie.

Wdrożenie projektu końcowego zostanie zrealizowane na platformie **Railway**. Railway umożliwia na łatwe wdrożenie kontenerów Docker na serwerach w chmurze. Ich darmowy plan wystarczy na potrzeby projektu.

Kontrola wersji projektu zostanie zrealizowana za pomocą narzędzia **Git** i platformy **GitHub**. Git umożliwia zarządzanie kodem źródłowym projektu, a GitHub umożliwia przechowywanie kodu źródłowego w chmurze. Repozytorium projektu będzie dostępne publicznie pod tym linkiem.