Serwis cyfrowej dystrybucji gier komputerowych „G-Store”

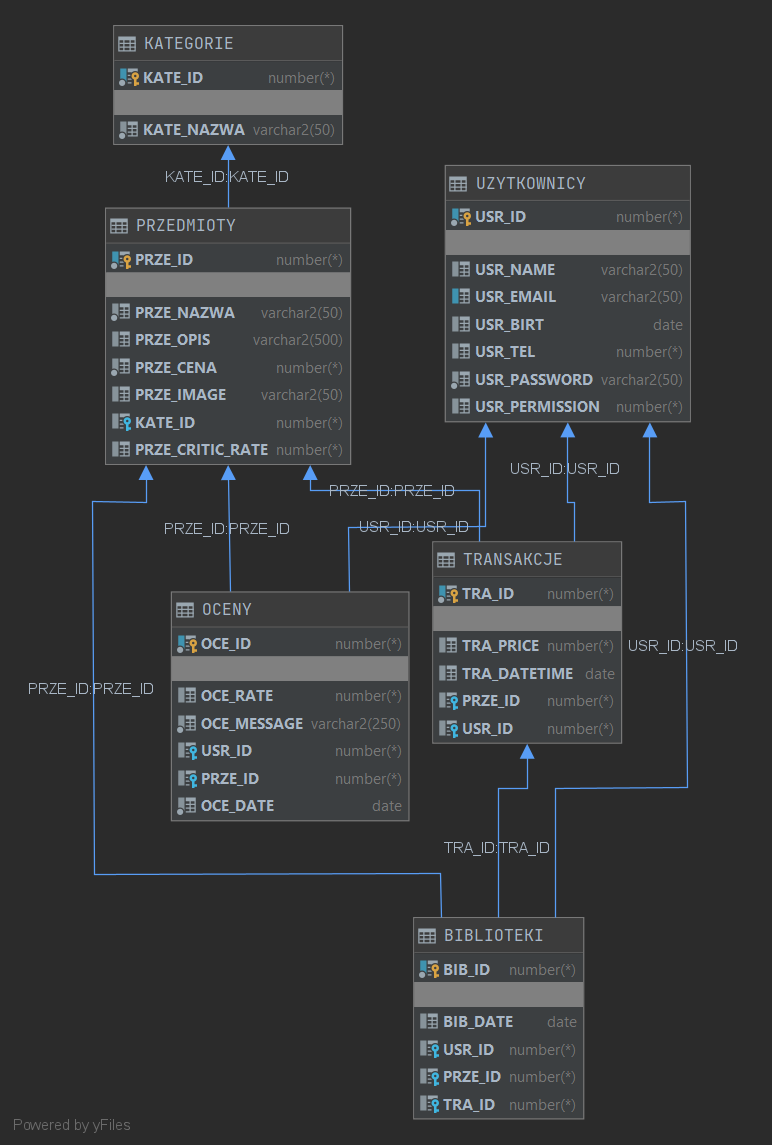
Łukasz Zebzda

Informatyka rok 2

Grupa laboratoryjna 3

1. Założenia projektu:

Projekt zakłada stworzenie aplikacji internetowej umożliwiającej zakup gier komputerowych, które trafią do biblioteki przypisanej dla danego użytkownika. Aplikacja ma również za zadanie dać możliwość wystawiania ocen jak i komentarzy dla poszczególnych produktów, w tym przypadku gier dystrybucji cyforwej. Aplikacja łączy w sobie „backend” stowrzony przy pomocy php oraz bazy danych Oracle, oraz „frontend” stworzony przy pomocy popularnego frameworku Angular.

2. Diagram ERD:

Na powyższym diagramie możemy zaobserwować łącznie 6 tabel.

Krótki opis danych tabel:

* **Przedmioty** – Tabela ta przechowuje wszystkie gry jakie znajdziemy na stronie, znajdziemy tam ich opisy, nazwe, cenę, czy chociażby ocenę renomowanych krytyków.
  + Przedmioty -> Kategorie – jest to relacja wiele do jednego, pozwala ono przypisać do każdej z kategorii wiele tytułów gier.
* **Użytkownicy** – Tabela ta przechowuje użytkowników serwisu. Zawiera takie dane jak dane logowania, oraz co ważne poziom uprawnień danego użytkownika.
* **Transakcje** – Tabela ta zawiera informacje na temat wszelakich zakupów dokonanych w obrębie aplikacji, przykładowe dane to cena dokonanego zakupu z danej chwili, czas dokonania zakupu oraz numer użytkownika i przedmiotu.
  + Transakcje -> Użytkownicy – jest to relacja wiele do jednego, gdyż jeden użytkownik może dokonać wielu transakcji, natomiast każda transakcja może posiadać wyłącznie jednego użytkownika.
  + Transakcje -> Przedmioty – jest to relacja wiele do jednego, gdyż każdy przedmiot może być zakupiony wielokrotnie, natomiast każda transakcja dotyczy tylko jednego egzemplarza.
* **Oceny** – Tabela ta przechowuje zarówno komentarze jak i oceny użytkowników(Nie mylić z oceną krytyków).
  + Oceny -> Przedmioty – jest to relacja wiele do jednego, ponieważ każdy przedmiot może posiadać wiele różnych komentarzy użytkowników, ale każdy komentarz odnosi się wyłącznie do jednego konkretnego przedmiotu.
  + Oceny -> Użytkownicy – jesto to relacja wiele do jednego, ponieważ każdy użytkownik może napisać wiele komentarzy w zależności od posiadanego zasobu gier, natomiast każdy z komentarzy został stworzony wyłącznie przez jednego konretnego użytkownika.
* **Biblioteki** – Tabela ta posiada odrębną od transakcji informację o posiadaniu przez danego użytkownika prawa do danego tytułu. Każdy wpis w tej tabeli zawiera datę dodania danej gry do biblioteki, oraz to kto zakupił, co zakupił, i przy pomocy której transakcji, dane te można w łatwy sposób odczytać przy pomocy podanych w późniejszej części procedury.
  + Biblioteki -> Przedmioty – jest to relacja wiele do jednego, ponieważ każdy przedmiot może znaleźć się w bibliotece wielu osób a więc co za tym idzie w wielu bibliotekach, natomiast każdy wiersz tabeli Biblioteki odnosi się wyłącznie do jednego przedmiotu.
  + Biblioteki -> Użytkownicy – jest to relacja wiele do jednego, ponieważ każdy użytkownik może mieć wiele przedmiotów w swojej osobistej bibliotece, a co za tym idzie wiele rekordów w tabeli Biblioteki, natomiast jak w przypadku przedmiotów każdy rekord tabeli Biblioteka odnosi się tylko i wyłącznie do jednego użytkownika.
  + Biblioteki -> Transakcje – jest do nieco odmienny przypadek, ponieważ występuje tu rodzaj relacji jeden do jednego, a co za tym idzie każdy jeden wpis w tabeli Biblioteki posiada jeden odpowiadający mu wpis w tabeli transakcje.
* **Kategorie** – Jest to ostatnia aczkolwiek nie mniej ważna tabela mająca za zadanie przechowywanie nazw kategorii rozróżnianych w serwisie, jej relacja z tabelą przedmioty została już wcześniej wspomniana.

**3. Wykorzystane Technologie:**

* Oracle Database 18c Enterprise Edition
* Angular CLI 9.1.7/ Angular 9.1.9
* Xampp Version 7.4.2

**4. Procedury / Triggery / Sekwencje**

* **AddComment**(ocena, treść, id\_użytkownika, id\_przedmiotu)
  + Procedura ma za zadanie wstawienie rekordu do tablicy Oceny, jednocześnie sprawdza czy aby przypadkiem dany użytkownik nie skomentował już tego tytułu, w takim przypadku zwraca niestandardowy wyjątek
* **ChangePermission**(kto, komu, do rangi)
  + Procedura ma za zadanie zmienić uprawnienia użytkownika(Docelowo podnieść je o stopień, bądź obniżyć, w zależności od wartości wejściowej)
* **DeleteComment**(id\_komentarza)
  + Procedura usuwa wybrany komentarz
* **Login**( mail, haslo, OUT rekord)
  + Procedura pobiera dane autoryzacji podane przez użytkownika, po czym zwraca dane przy pomocy wartości wyjściowe rekord
* **MakeTransaction**(cena, id\_przedmiotu, id\_uzytkownika, OUT data\_transakcji)
  + Procedura pobiera niezbędne wartości do przerpowadzenia transakcji, pierwszym jej krokiem jest sprawdzenie, czy aby na pewno dany użytkownik nie posiada jeszcze gry o podanym id, w razie wykrycia takie stanu zgłasza niestandardowy wyjątek, w innym wypadku docohdzi do pobrania daty systemowej, oraz wstawienia wartości dla tabel Transakcje, oraz Biblioteka. W razie powodzenia procedura zwraca zadeklarowaną wcześniej wartość wyjściową data\_transakcji
* **Registration**(nick, haslo, email, tel, OUT uzy\_id)
  + Procedura pobiera niezbędne dane, ustala domyślną wartość uprawnień użytkownika jako 1, pobiera date systemową, oraz wstawia wszystkie wymienione powyżej informacje do tabeli Uzytkownicy. Zwraca wartość id nowo wstawionego użytkownika.
* **Show\_Library**(id\_uzytkownika, OUT rekordy)
  + W przypadku tej procedury łączymy dwie tabele przy pomocy INNER JOIN, po to aby wypisać dane wszystkich gier z tabeli przedmioty, gdzie wartości prze\_id są równe dla obydwu tabel, oraz wartość usr\_id jest identyczna dla podanej wartości procedury(id\_uzytkownika), wynikiem tego zapytania jest tablica przedmiotów podana na wyjście(rekordy), którą możemy przypisać bezpośrednio do klasy Items(Jest kompatybilna z tabelą Przedmioty)
* **Show\_Categories**(OUT rekordy)
  + Prosta procedura mająca za zadanie podać wszystkie nazwy z tabeli Kategorie
* **Show\_Items**(OUT rekordy)
  + Prosta procedura podająca na wyjście(rekordy) wszystkie przedmioty zawarte w tabeli przedmioty
* **ShowComments**(id\_przedmiotu, OUT rekordy)
  + Procedura ma na celu znalezienie komentarzy odpowiadających id podanemu na wejściu, oraz zwrócenie ich do wartości rekordy w postaci ustalonej w aplikacji klasy, stąd pojawia się konieczność użycia inner join dla skorzystania połączonych rekordów tabel Oceny, oraz Uzytkownicy
* **ShowUsers**(rekordy)
  + Procedura zwraca do zmiennej rekordy dane wszystkich uzytkownikow, z pominieciem wartosci kolumn password
* Aplikacja posiada również kilka triggerów, mają one na celu automatyczne wprowadzanie nowega id dla każdej z kolumn przez nich obsługiwanych w miejsce wartości null
  + Tworzenie ich opierałem o sekwencję „trzymającą aktualny index dla danej tabeli, następnie przystępowałem do stworzenia triggera o warunkach BEFORE INSERT ON (nazwa tabeli) FOR EACH ROW itd.
  + Z czasem jednak znalazłem nowsze, bardziej adekwatne rozwiązanie które polegało na dodaniu: „NUMBER GENERATED by default on null as IDENTITY”. Dlatego w bazie można spotkać się z obiema formami podejścia do autoincrementowania indexu tabeli. Trzeba jednak pamiętać że to rozwiązanie jest kompatybilne wyłącznie dla wersji Oracle 12c+

**5.Praktyczna implementacja projektu**

* **Backend PHP**

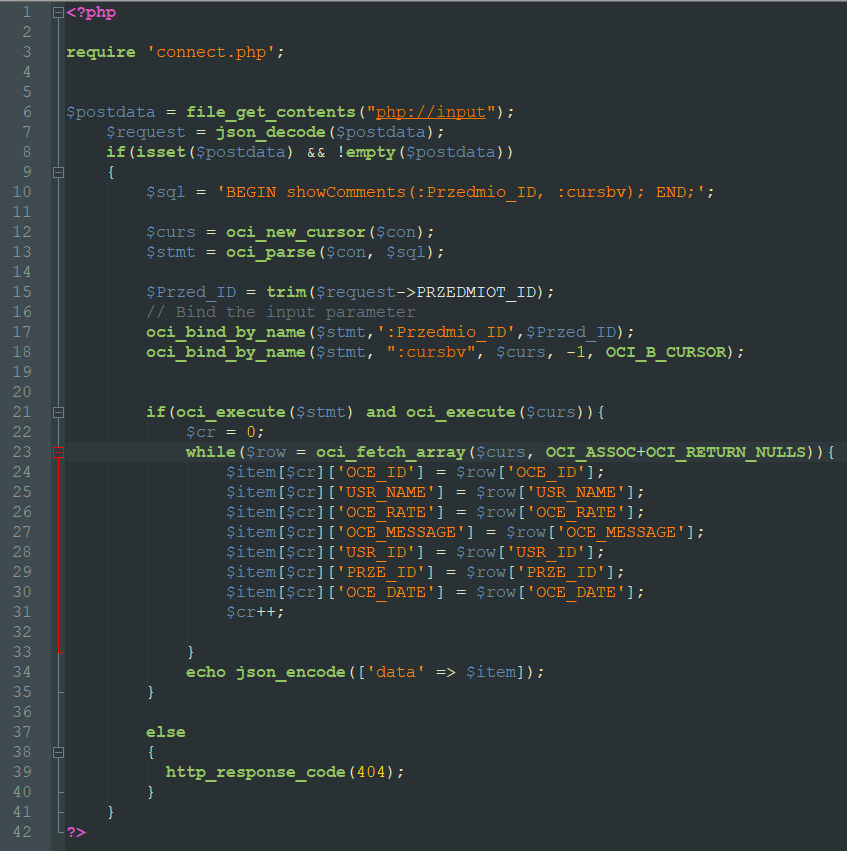
Plik zawierający dane połączenia z bazą

(connect.php)



Pliki zawierające przykładowe mechanizm przesyłania danych pomiędzy Frontendem(Angular) a bazą, przy pomocy PHP

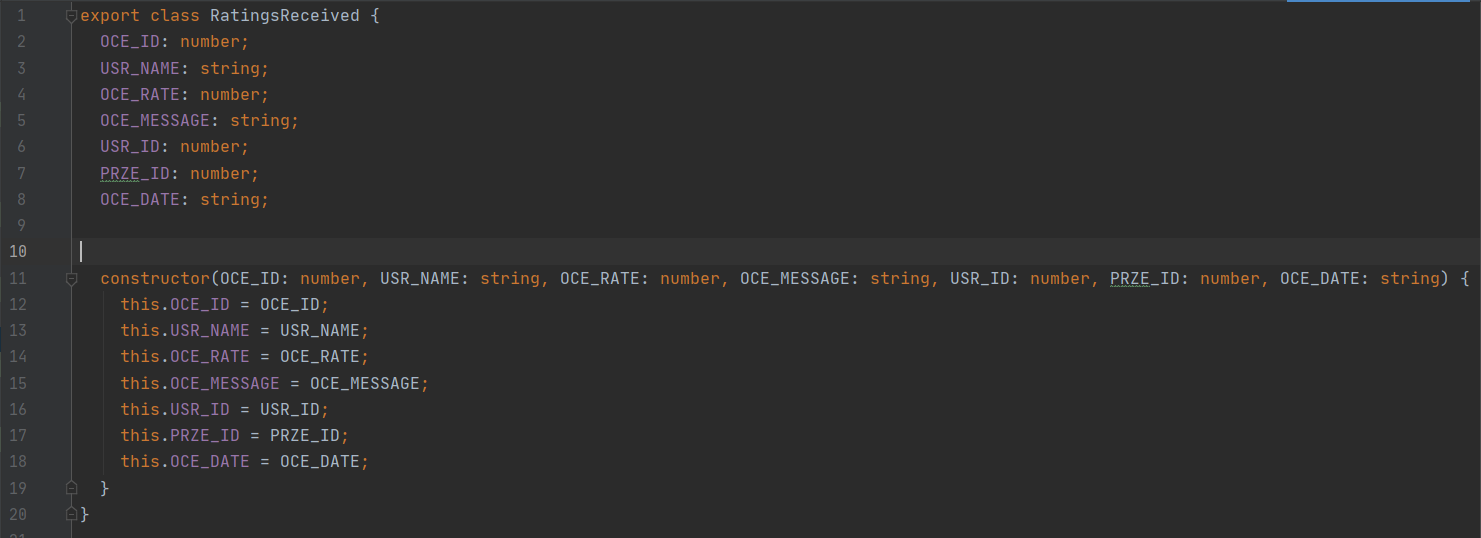
(readComment.php)



Powyższy plik ukazuje wysyłania do bazy przy pomocy procedury, mapowania jej wyników na wartości, a następnie mapowania tych wartości do formatu JSON.



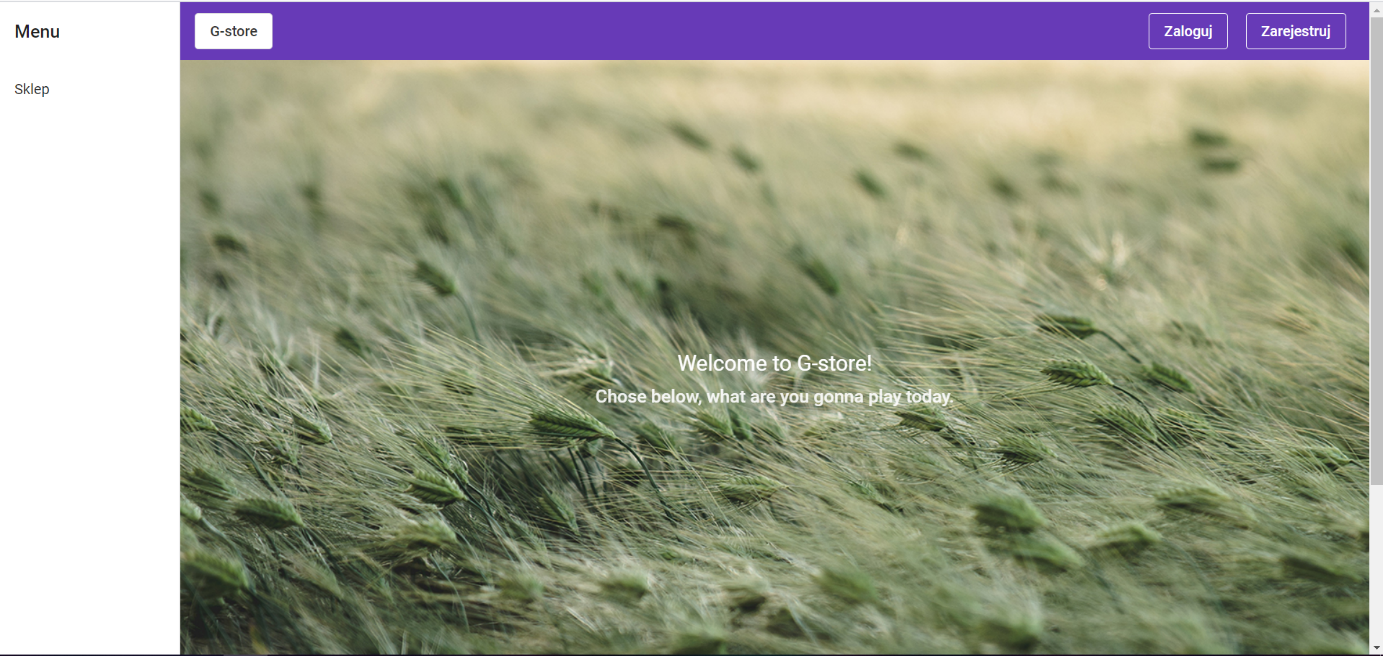
Powyżej obserwujemy przykład procedury wykonującej wstawianie danych do wielu kolumn jednocześnie w prostej i przejrzystej formie.



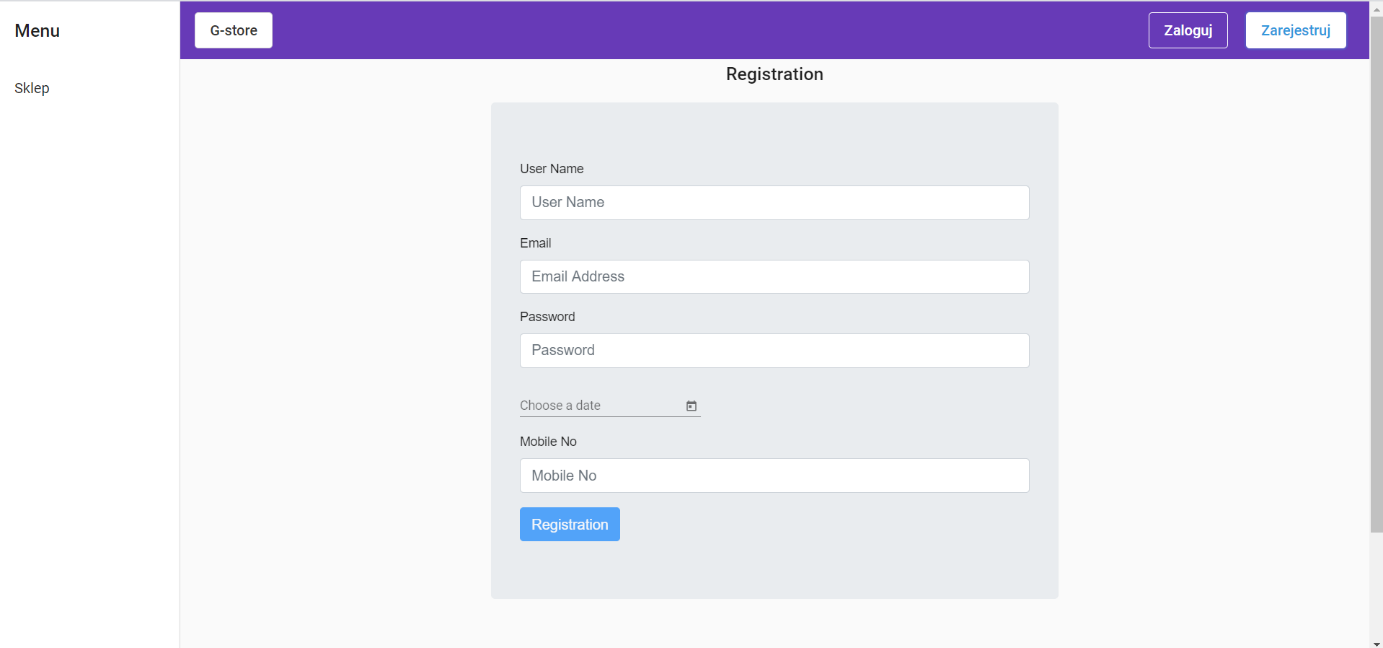
Docelowa Klasa mapowana z obiektu JSON.

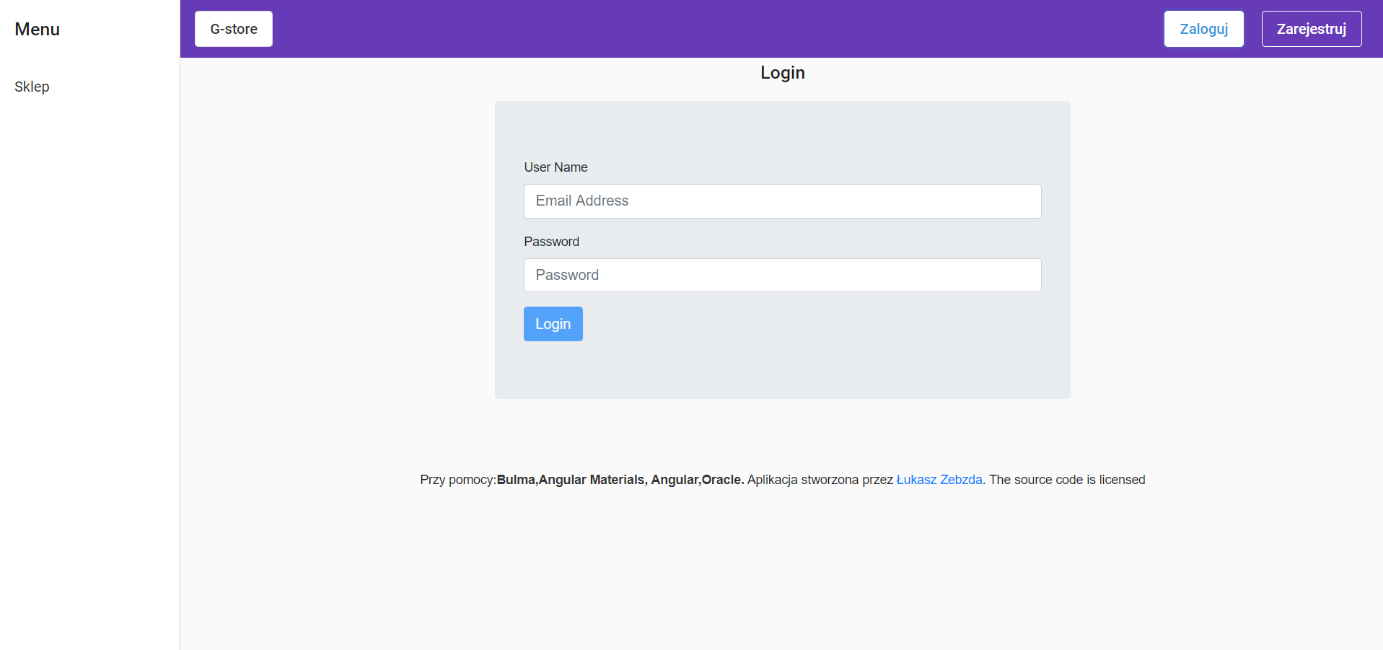
**6.GUI**

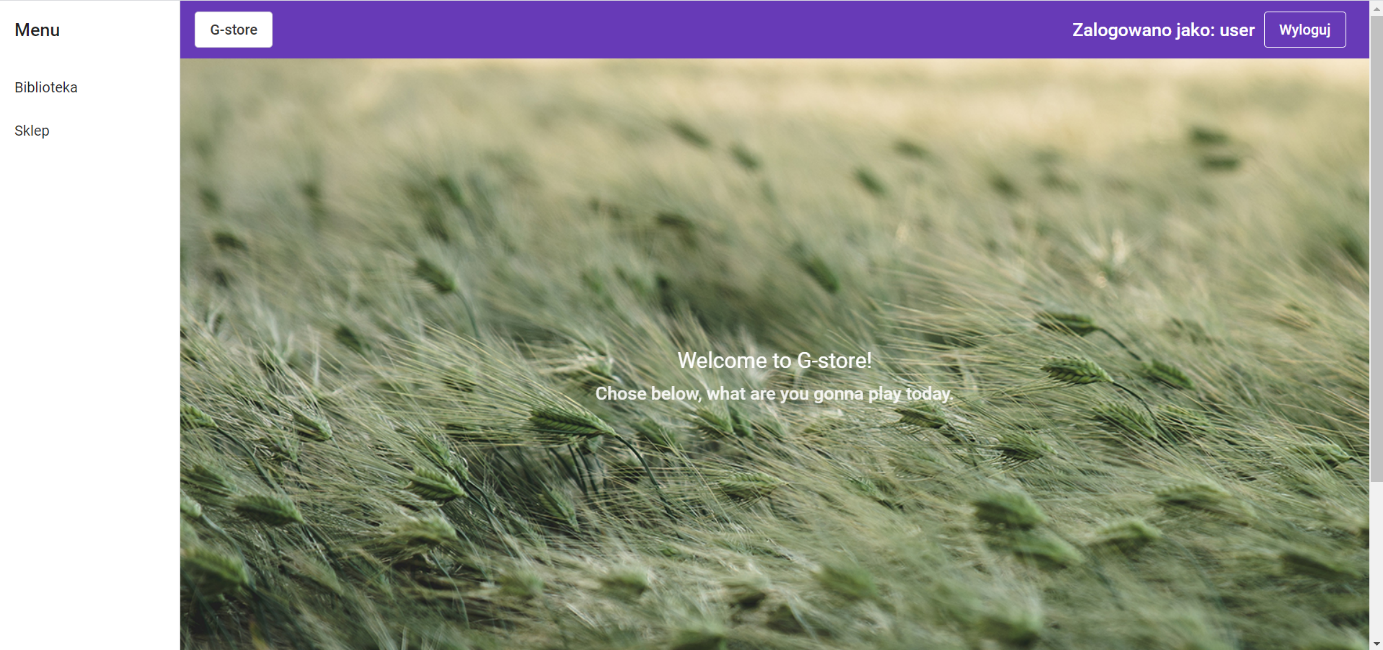
a)Wygląd startowy

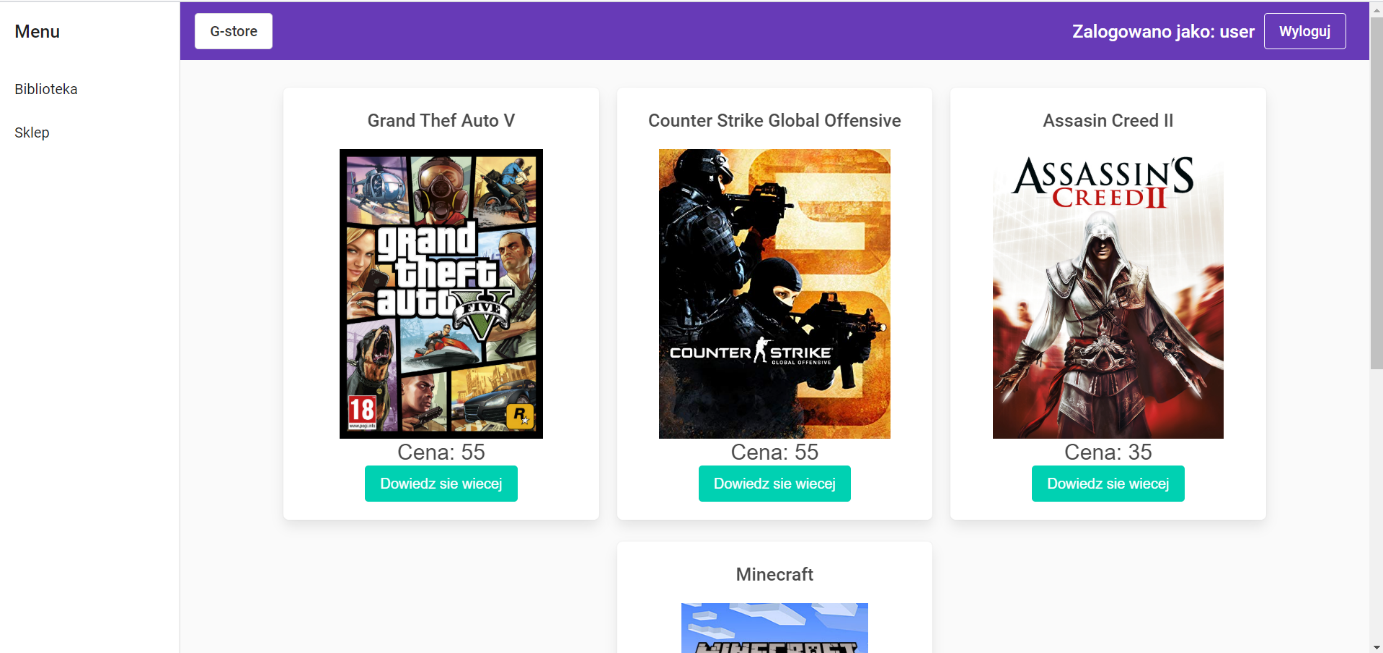


b) Panel rejestracji

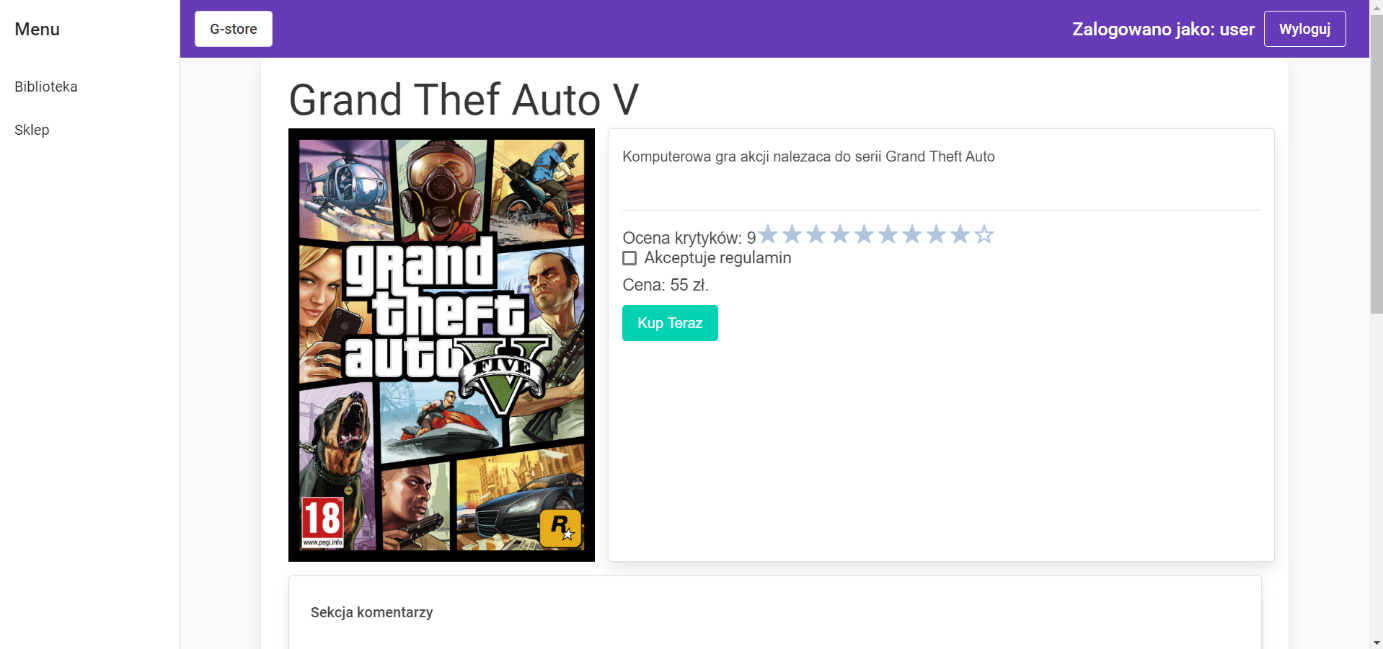


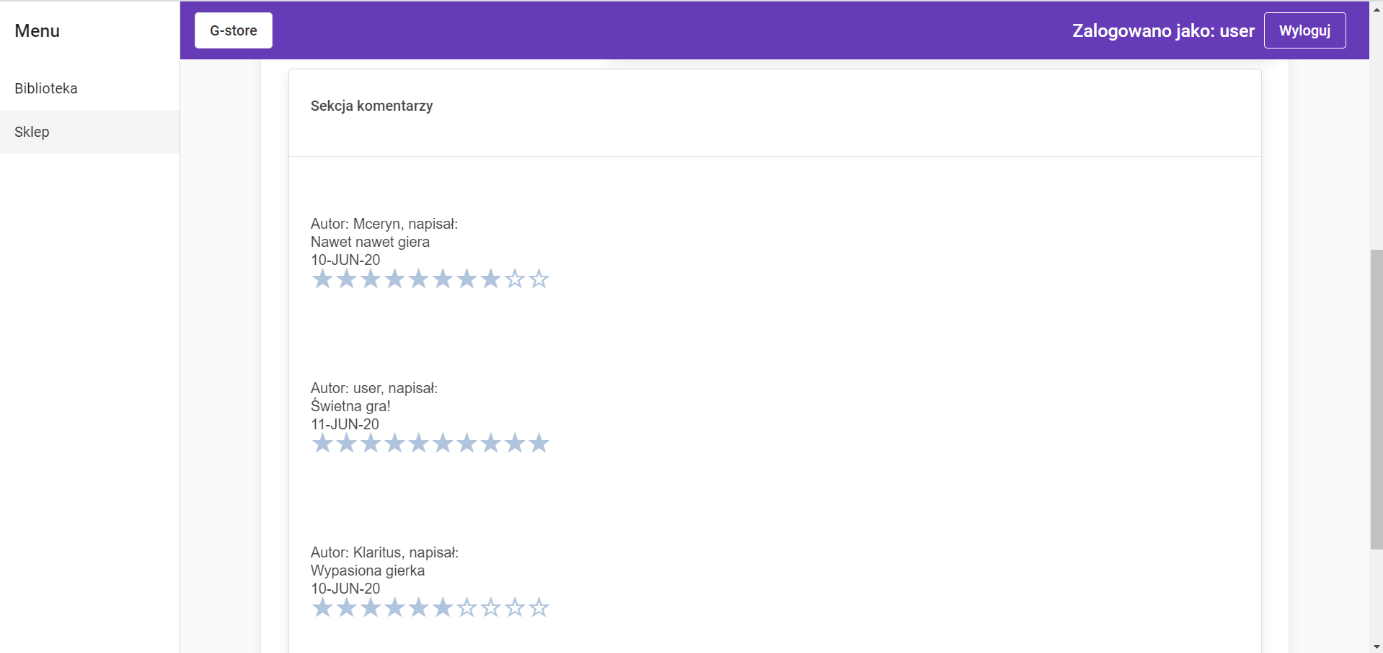
c) Panel logowania

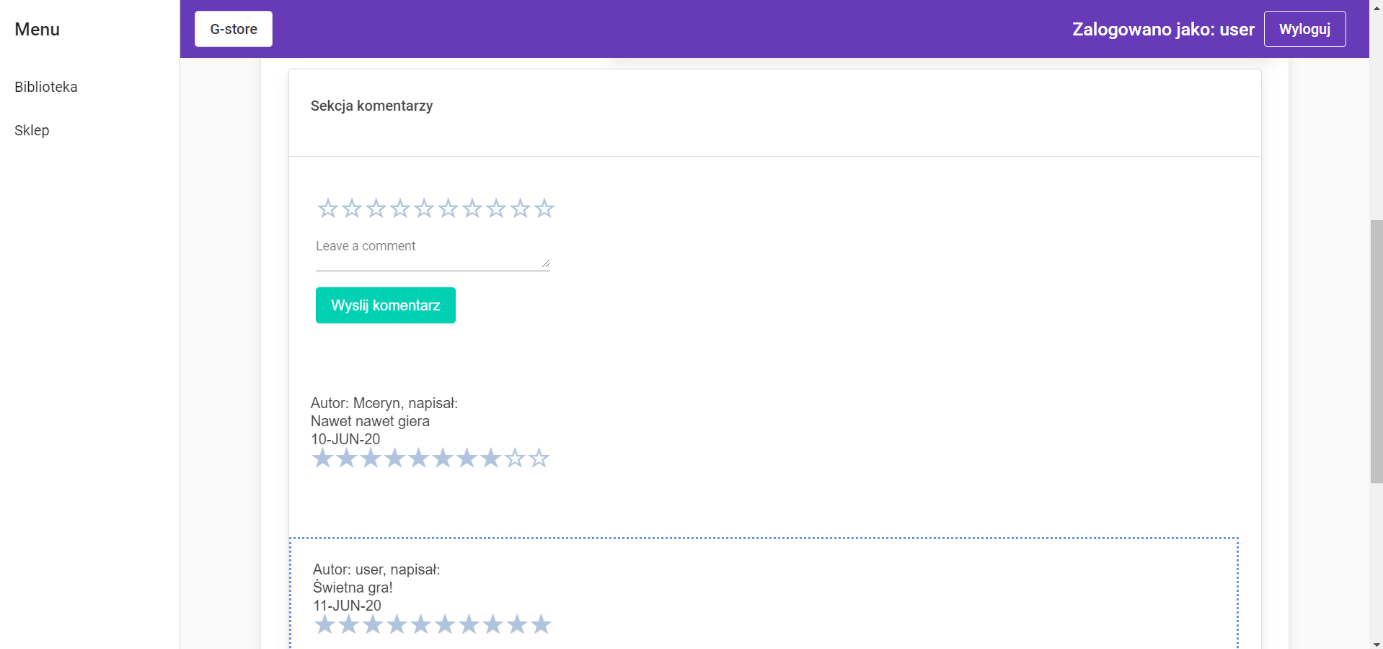
d) Widoczność zalogowania użytkownika

e)Przykładowa zawartość sklepu

f)Przykładowy produkt w sklepie



g)Sekcja komentarzy w sklepie

h) Sekcja komentarzy w bibliotece

**Podsumowanie**

Projekt reprezentuje działająca aplikacja o realnych możliwościach dalszego rozwoju dzięki zstosowaniu procedur dla zmniejszenia redundancji kodu, jak i użycia framewordku Angular.

Aktualna struktura projektu:

* Backend – Katalog ten zawiera pliki php niezbędne do połączenia GUI z bazą
* Docs – Katalog ten zawiera aktualną ostateczną wersję dokumentacji ze schematem, jak i podkatalog zawierający wcześniejszy opis aplikacji oraz jego schemat
* Gui – Katalog ten zawiera czesc Graficzną aplikacji, więcej o postępowaniu z tym katalogiem można przeczytać w instrukcji
* Pass - w tym katalogu znajdują się hasła niezbędne do testowania działania aplikacji
* Sql – tutaj znajduje się niebinarny export bazy danych