Steam Buddy

Celem projektu było sworznie aplikacji, ułatwiającej użytkownikom znalezienia partnera do wspólnej rozgrywki w gry wideo.

opis funkcjonalny systemu

1. System potrafi pobierać dane gier ze Steam.
2. Aplikacja generuje logi najważniejszych systemów (steam, invite).
3. System generuje statystyki.
4. Rejestracja użytkownika.
5. Łączenie się użytkownika z kontem Steam (aplikacja potrafi uzyskać z api steam_id użytkownika)
6. Pobieranie danych użytkownika z steam.
7. Wyświetlanie biblioteki gier użytkownika.
8. Sortowanie gier po nazwie oraz tagach.
9. Wyświetlanie okładki oraz opisu gry. 10 Wyświetlanie statystyk i osiągnięć gracza w danej grze.
10. Wyszukiwanie graczy o podobnych osiągnięciach jaki i czasie gry.
11. System wysyłania prośbę o wspólną grę do dopasowanych wcześniej użytkowników (prośbę można przyjąć lub odrzucić)
12. Powiadomienia mailowe o otrzymanych prośbach.
13. Powiadomienia mailowe o aktualizacji status wysłanych próśb.
14. Udostępnianie linka do profilu steam użytkownika jeżeli wyraził on chęć wspólnej gry.

Opis technologiczny

Część backendowa projektu została napisana w php z użyciem framework'a laravel 11. Frontend powstał z użyciem tailwind css, react'a i typescriptu. Całość została spięta z sobą za pomocą biblioteki inertia.js. Dane aplikacji przechowywane są w bazie danych PostgreSQL. Środowisko developerskie, oraz produkcyjne zostało wykonane z użyciem dokera i nginx. Do obsługi cache i wykorzystana została baza danych Redis.

wyszczególnione wdrożone zagadnienia kwalifikacyjne

1. Framework MVC — Projekt został przygotowany z użyciem Laravel'a i klasycznego dla niego wzorca MVC.
2. Framework CSS — W celu przyspieszenia developmentu użyliśmy frameworka tailwind.css dostosowanego do potrzeb naszego designu poprzez modyfikacje pliku (tailwind.config.js)
3. Baza danych — Projekt korzysta z bazy danych PostgresDB
4. Cache — W celu redukcji ilości wykonywanych zapytań do bazy, został użyty wbudowany w laravela mechanizm cache połączony z bazą danych Rails.
5. Dependency Manager — Projekt korzysta z dwóch dependency managerów. Composer'a dla php i npm dla javascriptu.
6. HTML — realizowany przy użyciu biblioteki React.
7. CSS — aplikacja została ostylowana za pomocą css.
8. JavaScript — frontent został napisany w tym języku, powstały w nim niektóre animacje oraz elementy interaktywne.
9. Routing — aplikacja korzysta z routingu dostarczoneg przez laravela.
10. ORM — Nasz projekt korzysta ze Eloquent ORM w celu ułatwienia pracy z bazą danych.
11. Lokalizacja — Steam buddy został stworzony w dwóch wersjach językowych, Polskiej iAngielskiej. Wykorzystana została do tego dostarczony wraz z laravelem mechanizm lokalizacji połączony z reactem za pomocą biblioteki laravel-react-i18n. 13.Mailing — system wysyła powiadomienia o zmianie statusu wysłanych zaproszeń jak i o zakończonej synchronizacji ze steam poprzez wiadomości e-mail. Wykorzystany został do tego celu udostępniany przez laravel'a mailing spięty z usługą MailTrap w środowisku developerski i PostMark w produkcyjnym.
12. Formularze — do procesu uwierzytelniania jak i późniejszej aktualizacji profilu użytkownika wykorzystane zostały formularze.
13. asynchroniczne interackje — do pobierania informacji o postępie czasochłonnego procesu pobierani informacji o grach ze Steam wykorzystane zostało javascriptowe FeatchApi, za którego pomocą wysyłane jest żądanie http pobierające postęp importu danych na podstawie którego jest aktualizowany interface użytkownika.
14. Konsumpcja api — do pobierani danych o grach i osiągnięciach użytkownika wykorzystane zostało zewnętrzne api udostępnione przez steam.

15. Publikacja api — aplikacja udostępnię wybrane statystyki serwisu (najpopularniejszych gry, łączna liczba przegranych godzin itp.) za pomocą wystawianego przez niąAPI.
16. RWD — aplikacja jest responsywna, dzięki użyciu media query i jednostek relatywnych wspiera zarówno urządzenia mobilne, tablety, jaki i komputery osobiste.
17. Logger — dane o pobieranych danych ze Steam i aktualizacje statusów zaproszeń użytkowników są zapisywane w logach aplikacji z wykorzystaniem wbudowanego w laravela loggera.
18. Deployment — aplikacja została skonteneryzowana, dzięki czemu ułatwiony został proces deploymentu. Dodatkowo został napisany skrypt GitHub actions, który w buduje automatycznie obrazy projektu i zapisuje je w pliku image.tar.

Instrukcję lokalnego i zdalnego uruchomienia systemu

Środowisko developerskie

1. Zainstaluj na swojej ulubionej dystrybucji linux'a docker'a, docker-compose i make.
2. Sklonuj repozytorium git

   ```
   git clone https://github.com/MRR-Group/steam-buddy.git
   ```

3. Otwórz terminal i przejdź do katalogu, w którym znajduje się plik docker-compose.yml.
4. Stwórz plik .env i dostosuj jego konfigurację.

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

5. Przygotuj aplikacje do pracy

   ```
   make init
   ```

6. Uruchom projekt:

   ```
   make dev
   ```

7. Po zakończeniu procesu uruchamiania projekt będzie dostępny pod adresem http://localhost.

Środowisko produkcje

1. Zainstaluj na swojej ulubionej dystrybucji linux'a docker'a, docker-compose i make.
2. Sklonuj repozytorium git

   ```
   git clone https://github.com/MRR-Group/steam-buddy.git
   git branch -f production origin/production
   git checkout production
   ```

3. Otwórz terminal i przejdź do katalogu, w którym znajduje się plik docker-compose.yml.
4. Stwórz plik .env i dostosuj jego konfigurację.

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

5. Pobierz spakowane obrazy docker projektu

   ```
   gh release download WERSJA
   ```

6. Wczytaj pobrane obrazy

   ```
   docker load < image.tar
   ```

7. Uruchom projekt:

   ```
   make prod
   ```

Wnioski projektowe

Podczas pisania projektu zetknięcie się z tak dużym frameworkiem było dla nas trudne. Praktycznie cały czas, do samego końca posiłkowaliśmy się dokumentacją, która jednak okazała się naprawdę dobrym i przystępnym źródłem wiedzy. Zdecydowaliśmy się, zupełnie nie potrzebnie na aktualizacje laravela do wersji 11, co znacznie utrudniło nam korzystanie z poradników i bibliotek. W przyszłości będziemy ostrożniej podejmować takie decyzję. Praca w środowisku

dokerowym okazała się zaletą i wadą, przyspieszyła ona znacząco postawienie środowiska developerskiego oraz uprościła wprowadzanie w nim zmian, jednak jednocześnie podniosła ona próg wejścia do projektu. Większość z nas nie była zaznajomiona wcześniej, ani z dokerem, ani z linuxem. Dodatkowo 2/3 uczestników pracowało na Windowsie, co jeszcze bardziej utrudniło prace.

Podczas pisania tego projektu zmieniło się również nasze postrzeganie PHP, dał się on poznać jako nowoczesne i wygodne narzędzie. Jaki i również wzrosła nasza sprawność w posługiwaniu się systemem wersji git i samym laravelem.