Steam Buddy

Cel projektu

Celem projektu było stworzenie aplikacji, ułatwiającej użytkownikom znalezienie 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 ze 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.
11. Wyszukiwanie graczy o podobnych osiągnięciach jak i czasie gry.
12. System wysyłania próśb o wspólną grę do dopasowanych wcześniej użytkowników (prośby można przyjąć lub odrzucić).
13. Powiadomienia mailowe o otrzymanych prośbach.

14. Powiadomienia mailowe o aktualizacji statusu wysłanych próśb.

15. Udostępnianie linka do profilu Steam użytkownika, jeżeli wyraził on chęć wspólnej gry.

Opis technologiczny

Część backendowa projektu została napisana w PHP z użyciem frameworka Laravel. Frontend powstał z użyciem Tailwind CSS, Reacta i TypeScriptu. Całość została spięta za pomocą biblioteki Inertia.js. Dane aplikacji przechowywane są w bazie danych PostgreSQL. Środowisko developerskie oraz produkcyjne zostało wykonane z użyciem Dockera i Nginx. Do obsługi cache wykorzystana została baza danych Redis.

Wyspecyfikowane 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 PostgreSQL.

4. Cache — W celu redukcji ilości wykonywanych zapytań do bazy, został użyty wbudowany w Laravela mechanizm cache połączony z bazą danych Redis.

5. Dependency Manager — Projekt korzysta z dwóch dependency managerów: Composera dla PHP i npm dla JavaScriptu.

6. HTML — Realizowany przy użyciu biblioteki React.

7. CSS — Aplikacja została ostylowana za pomocą CSS.

8. JavaScript — Frontend został napisany w tym języku, powstały w nim niektóre animacje oraz elementy interaktywne.

9. Routing — Aplikacja korzysta z routingu dostarczonego przez Laravela.

10. ORM — Nasz projekt korzysta z Eloquent ORM w celu ułatwienia pracy z bazą danych.

11. Lokalizacja — Steam Buddy został stworzony w dwóch wersjach językowych, Polskiej i Angielskiej. Wykorzystany został do tego mechanizm lokalizacji połączony z Reactem za pomocą biblioteki *laravel-react-i18n*.

12. 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 mechanizm dostarczany przez Laravela, spięty z usługą MailTrap w środowisku developerskim i PostMark w produkcyjnym.

13. Formularze — Do procesu uwierzytelniania jak i późniejszej aktualizacji profilu użytkownika wykorzystane zostały formularze.

14. Asynchroniczne interakcje — Do pobierania informacji o postępie czasochłonnego procesu pobierania informacji o grach ze Steam wykorzystane zostało JavaScriptowe Fetch API, za którego pomocą wysyłane jest żądanie HTTP pobierające postęp importu danych na podstawie którego jest aktualizowany interfejs użytkownika.

15. Konsumpcja API — Do pobierania danych o grach i osiągnięciach użytkownika wykorzystane zostało zewnętrzne API udostępnione przez Steam.

16. Publikacja API — Aplikacja udostępnia wybrane statystyki serwisu (najpopularniejsze gry, łączna liczba przegranych godzin itp.) za pomocą wystawianego przez nią API.

17. RWD — Aplikacja jest responsywna, dzięki użyciu *media query* i jednostek relatywnych wspiera zarówno urządzenia mobilne, tablety, jak i komputery osobiste.

18. Logger — Dane o pobieranych danych ze Steam i aktualizacje statusów zaproszeń użytkowników są zapisywane w logach aplikacji z wykorzystaniem wbudowanego w Laravela loggera.

19. Deployment — Aplikacja została skonteneryzowana, dzięki czemu ułatwiony został proces deploymentu. Dodatkowo został napisany skrypt GitHub Actions, który buduje automatycznie obrazy projektu i zapisuje je w pliku *image.tar*.

Instrukcja lokalnego i zdalnego uruchomienia systemu

Środowisko developerskie

1. Zainstaluj na swojej ulubionej dystrybucji Linuxa Dockera, 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 aplikację do pracy:

   ```
   make init
   ```

6. Uruchom projekt:

   ```
   make dev
   ```

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

Środowisko produkcyjne

1. Zainstaluj na swojej ulubionej dystrybucji Linuxa Dockera, 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 niepotrzebnie, na aktualizację Laravela do wersji 11, co znacznie utrudniło nam korzystanie z poradników i bibliotek. W przyszłości będziemy ostrożniej podejmować takie decyzje. 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 pracę. Podczas pisania tego projektu zmieniło się również nasze postrzeganie PHP, dał się on poznać jako nowoczesne i wygodne narzędzie. Także wzrosła nasza sprawność w posługiwaniu się systemem wersji git i samym Laravel'em.