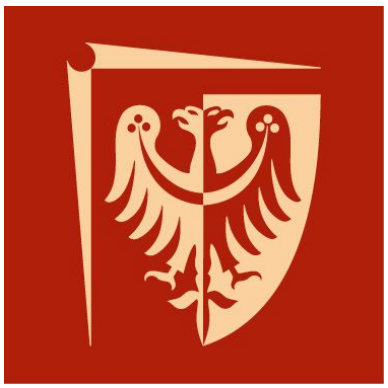


Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Informatyczne systemy automatyki

Bazy danych - Podsumowanie Projektu

Autorzy:
Damian Filipowski id. 272555
Konrad Landzberg id. 272508

Przedmiot:
Bazy danych - projekt

8 stycznia 2025

Spis treści

Spis rysunków

1 Wprowadzenie:

Nasza aplikacja webowa, Game Shop DamKon, to platforma inspirowana Steamem, która umożliwia użytkownikom zakup gier, przeglądanie kolekcji gier posiadanych przez znajomych oraz organizowanie swojej biblioteki gier. Aplikacja oferuje szereg funkcjonalności, takich jak zarządzanie kontem, lista znajomych, integracja portfela, konwersja walut, szczegółowe przeglądanie gier oraz możliwość eksportu historii konta w formie raportów PDF. Wszystkie te elementy zostały pomyślnie zaimplementowane, zapewniając intuicyjne i przyjazne dla użytkownika doświadczenie.

Link do repozytorium projektu: https://github.com/Bergu1/game-shop-project
Link do aplikacji live: http://ec2-52-91-102-211.compute-1.amazonaws.com/login/

2 Wykorzystane technologie:

Ostatecznie udało się wykorzystać wszystkie technologie zadeklarowane w poprzednim etapie.

- Typ aplikacji: Webowa
- Język programowania: Python
- Framework: Django
- Baza danych: PostgreSQL
- Frontend: HTML, CSS
- Dodatkowo: Git, Docker, Github Actions, AWS
- Testy: testy jednostkowe, testy integracyjne

3 Backend:

W projekcie cała dokumentacja backendu została przygotowana w formie kolekcji Postman. Każdy endpoint został dokładnie opisany, uwzględniając adres URL, wymagane dane wejściowe, odpowiedzi serwera oraz metody HTTP. Dokumentacja zawiera szczegółowe opisy poszczególnych widoków, takich jak logowanie, rejestracja, zarządzanie kontem, transakcje w portfelu, a także funkcjonalności związane z biblioteką gier i listą znajomych.

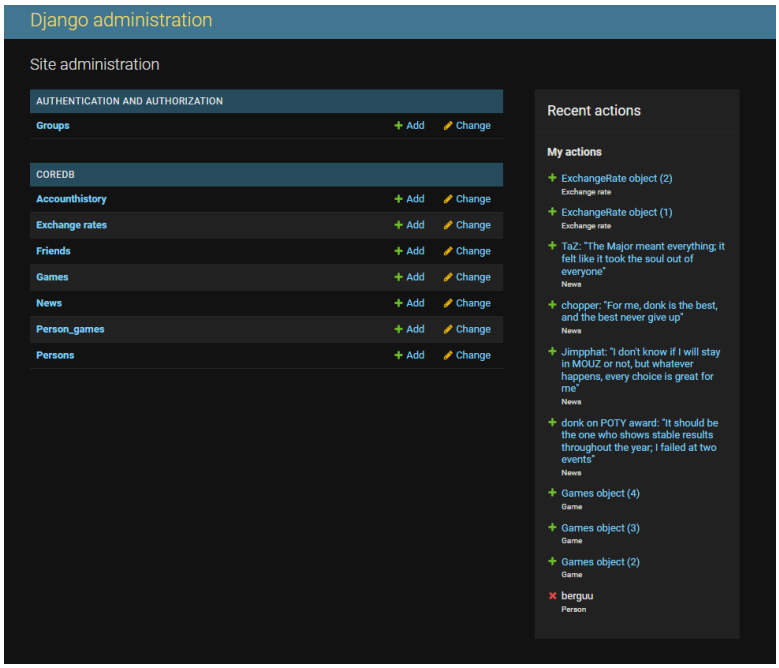
Poniżej znajduje się link do pobrania pliku dokumentacji w formacie JSON:
https://github.com/Bergu1/game-shop-project/blob/main/GameStore˙backend˙documentation.postman˙collection.json

Zawartość dokumentacji:
Ponieważ w przyszłości planowany jest dalszy rozwój projektu, w tym wprowadzenie API, cała dokumentacja backendu została napisana w języku angielskim, aby zachować spójność z ogólnym formatem projektu.

- Opis dla widoków:
  - Logowanie i rejestracja użytkowników.
  - Przeglądanie i zarządzanie biblioteką gier.
  - Dodawanie, usuwanie oraz zarządzanie znajomymi.
  - Historia transakcji i zarządzanie portfelem.
  - Zakup gier i zakup gier jako prezent.

  - Możliwość zarządzania swoim kontem oraz danymi.
- Opis wszystkich metod HTTP (GET, POST) i wymagań (np. tokenów CSRF).
- Szczegółowe opisy logiki widoków, obsługiwanych parametrów oraz przykładów odpowiedzi serwera.
- Integracja z systemem konwersji walut, aby ceny były dynamicznie dostosowywane do preferowanej waluty użytkownika.
- Możliwość zresetowania hasła drogą mailową.

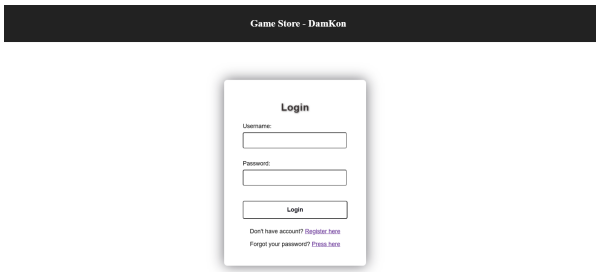
Cały backend został zintegrowany z panelem django-admin aby admin mógł nim zarządzać bez konieczności logowania się na serwer PostgreSQL.



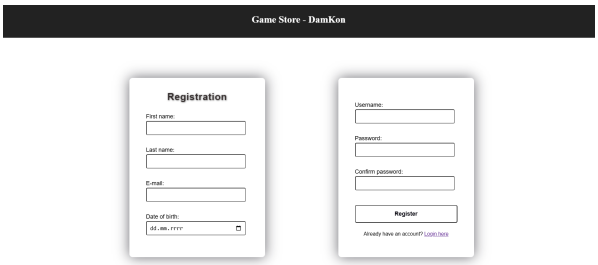
Rysunek 1: Panel administratora django.

4 Frontend:

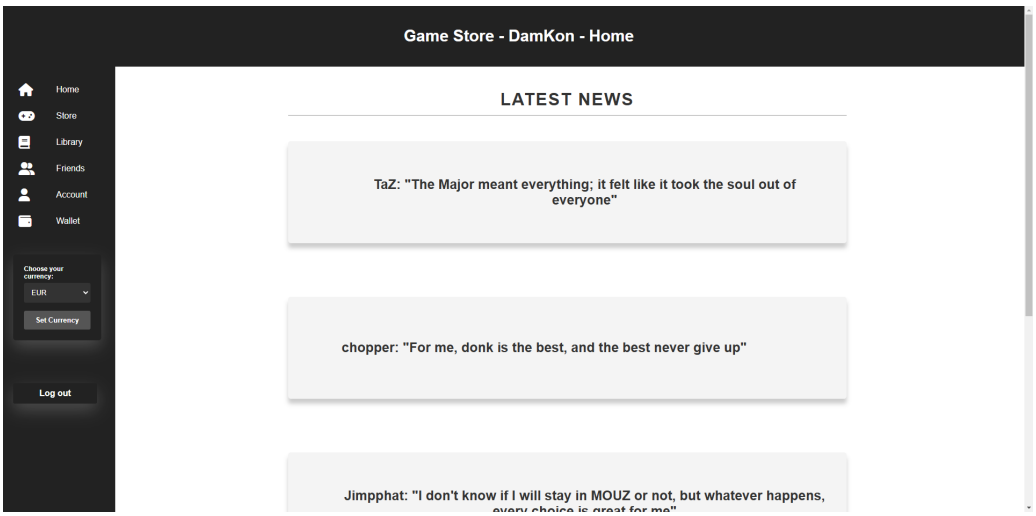
Frontend został wykonany w HTML i CSS, a jego integracja z backendem została zrealizowana za pomocą widoków oraz formularzy. Poniżej znajdują się przykładowe zrzuty ekranu przedstawiające interfejs użytkownika.



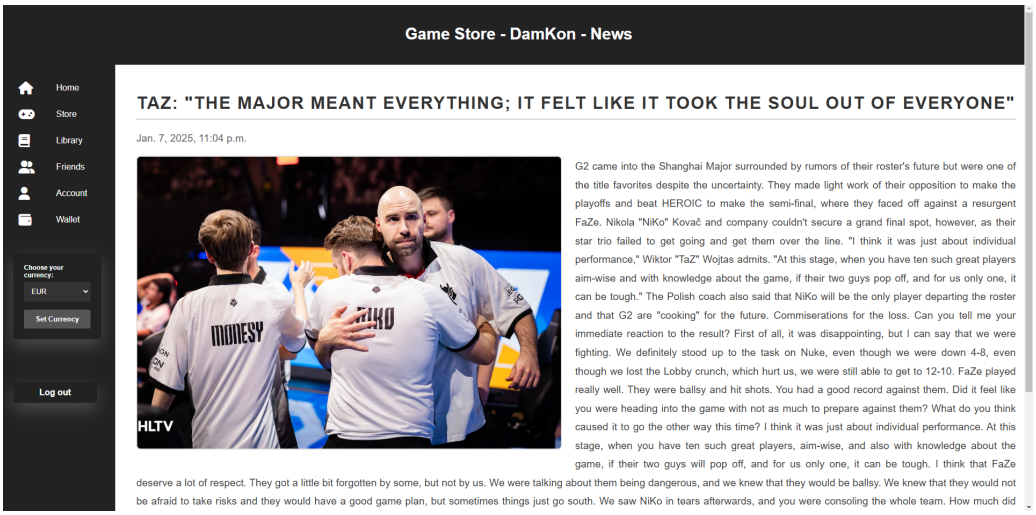
Rysunek 2: UI strony logowania.



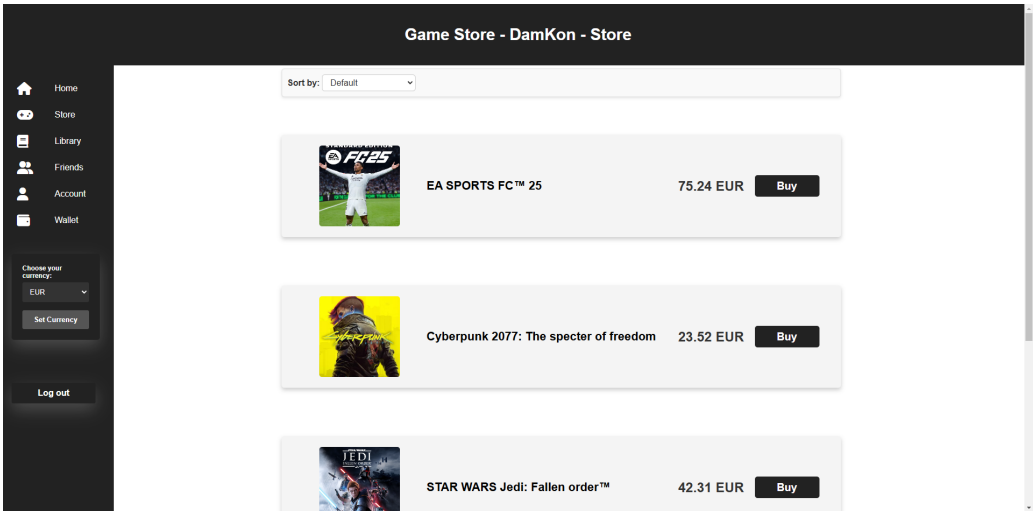
Rysunek 3: UI strony rejestracji.



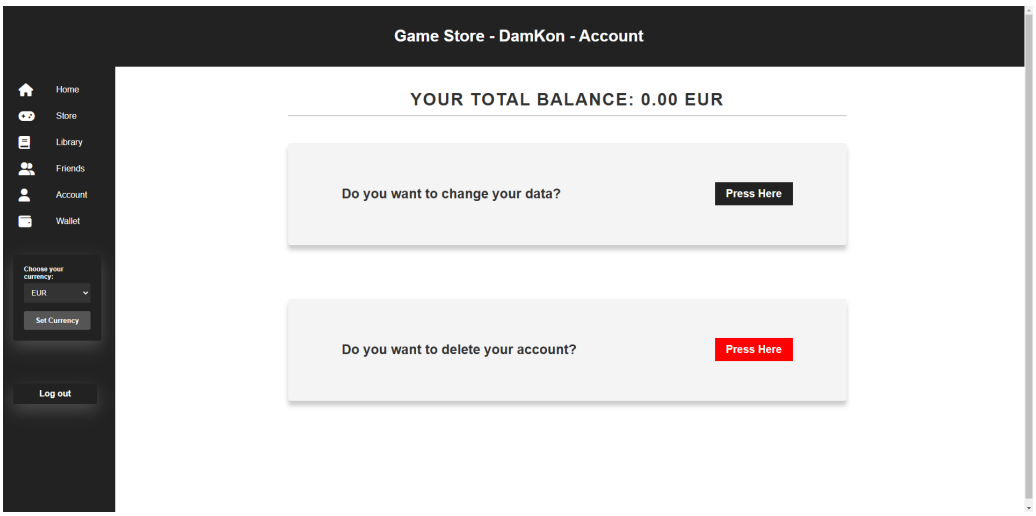
Rysunek 4: UI głównej strony do której przekierowany jest użytkownik po poprawnym logowaniu.



Rysunek 5: UI wybranego newsa.



Rysunek 6: UI sklepu.



Rysunek 7: UI zakładki account.

W celu przetestowania całego frontendu oraz aplikacji zapraszamy na naszą stronę: http://ec2-52-91-102-211.compute-1.amazonaws.com/login/. Dane do konta testowego Login: berguu Hasło: 2003kk, aby uniknąć etapu rejestracji.

5 Testy:

Podczas realizacji projektu przeprowadzono kluczowe testy w celu zapewnienia poprawnego działania aplikacji oraz spełnienia wymagań funkcjonalnych i niefunkcjonalnych. Testy te miały na celu zweryfikowanie stabilności systemu, zgodności z założeniami projektowymi oraz zidentyfikowanie potencjalnych błędów lub obszarów wymagających optymalizacji. Poniżej przedstawiono najważniejsze kategorie testów, które zostały wykonane.

Aplikacja: coredb

- Test test_wait_for_db_ready: Sprawdza, czy komenda wait_for_db działa poprawnie, gdy baza danych jest od razu dostępna.
- Test test_wait_for_db_delay: Weryfikuje, czy komenda wait_for_db obsługuje sytuacje opóźnionego dostępu do bazy danych.

Aplikacja: users

- Test test_create_user_with_email_successful: Sprawdza, czy użytkownik może zostać poprawnie utworzony z podanym adresem e-mail i hasłem.
- Test test_new_user_email_normalized: Weryfikuje, czy adres e-mail nowo utworzonego użytkownika jest normalizowany do małych liter.
- Test test_new_user_without_email_raises_error: Upewnia się, że próba utworzenia użytkownika bez e-maila wywołuje błąd.
- Test test_create_superuser: Sprawdza, czy superużytkownik jest tworzony z odpowiednimi uprawnieniami (is_superuser i is_staff).
- Test test_registration_page_status_code: Weryfikuje, czy strona rejestracji zwraca poprawny kod statusu HTTP.
- Test test_register_user: Sprawdza, czy użytkownik może się poprawnie zarejestrować i zostać zapisany w bazie danych.

- Test test_login_page_status_code: Weryfikuje, czy strona logowania zwraca poprawny kod statusu HTTP.
- Test test_user_login: Upewnia się, że użytkownik może się poprawnie zalogować i uzyskać dostęp do strony głównej.

Aplikacja: store

- Test test_buy_game_success: Sprawdza, czy użytkownik może pomyślnie kupić grę, a jego saldo i historia konta zostają odpowiednio zaktualizowane.
- Test test_buy_game_insufficient_balance: Weryfikuje, czy zakup gry jest blokowany w przypadku niewystarczających środków na koncie użytkownika.
- Test test_buy_game_already_owned: Sprawdza, czy użytkownik nie może ponownie kupić gry, którą już posiada.
- Test test_buy_game_invalid_password: Upewnia się, że zakup gry jest blokowany w przypadku podania nieprawidłowego hasła.
- Test test_buy_game_as_gift_success: Sprawdza, czy użytkownik może pomyślnie kupić grę jako prezent, a odbiorca otrzymuje ją na swoim koncie.
- Test test_buy_game_as_gift_insufficient_balance: Weryfikuje, czy zakup gry jako prezent jest blokowany, gdy użytkownik nie ma wystarczających środków.
- Test test_buy_game_as_gift_invalid_recipient: Sprawdza, czy zakup gry jako prezent jest blokowany w przypadku podania nieistniejącego użytkownika jako odbiorcy.
- Test test_buy_game_as_gift_already_owned: Upewnia się, że użytkownik nie może kupić gry jako prezent dla osoby, która już tę grę posiada.
- Test test_buy_game_as_gift_invalid_password: Sprawdza, czy zakup gry jako prezent jest blokowany w przypadku podania nieprawidłowego hasła.

Aplikacja: resetpassbymail

- Test test_send_email: Sprawdza, czy mail podczas resetowania hasła jest wysyłany do użytkownika w poprawnej formie.

Aplikacja: friends

- Test test_send_friend_request: Sprawdza, czy użytkownik może wysłać zaproszenie do znajomych do innego użytkownika.
- Test test_send_friend_request_to_self: Weryfikuje, czy system blokuje wysłanie zaproszenia do znajomych samemu sobie.
- Test test_send_duplicate_friend_request: Upewnia się, że użytkownik nie może wysłać duplikatu zaproszenia do znajomych.
- Test test_accept_friend_request: Sprawdza, czy użytkownik może zaakceptować zaproszenie do znajomych, zmieniając jego status na accepted.
- Test test_reject_friend_request: Weryfikuje, czy użytkownik może odrzucić zaproszenie do znajomych, zmieniając jego status na rejected.
- Test test_view_friends_list: Sprawdza, czy użytkownik może zobaczyć listę zaakceptowanych znajomych.

- Test test_remove_friend: Upewnia się, że użytkownik może usunąć znajomego ze swojej listy znajomych.
- Test test_remove_friend_not_involved: Weryfikuje, czy użytkownik nie może usunąć relacji znajomości, w której nie uczestniczy.
- Test test_view_friends_games: Sprawdza, czy użytkownik może zobaczyć gry posiadane przez swojego znajomego.

Aplikacja: currency

- Test test_change_currency_to_valid_currency: Sprawdza, czy użytkownik może zmienić walutę na jedną z obsługiwanych (np. USD), a jego wybór zostaje zapisany w bazie danych.
- Test test_change_currency_to_invalid_currency: Weryfikuje, czy próba zmiany waluty na nieobsługiwaną (np. GBP) jest blokowana, a domyślna waluta użytkownika pozostaje niezmieniona.

Aplikacja: account

- Test test_change_first_name: Sprawdza, czy użytkownik może pomyślnie zmienić swoje imię.
- Test test_change_last_name: Weryfikuje, czy użytkownik może pomyślnie zmienić swoje nazwisko.
- Test test_change_email_success: Sprawdza, czy użytkownik może pomyślnie zmienić swój adres e-mail na nowy, unikalny.
- Test test_change_email_duplicate: Upewnia się, że użytkownik nie może zmienić adresu e-mail na już istniejący w systemie.
- Test test_change_username_success: Sprawdza, czy użytkownik może pomyślnie zmienić swoją nazwę użytkownika.
- Test test_change_username_duplicate: Weryfikuje, czy użytkownik nie może zmienić nazwy użytkownika na już istniejącą w systemie.
- Test test_change_password: Sprawdza, czy użytkownik może zmienić swoje hasło i zalogować się ponownie za pomocą nowego hasła.
- Test test_change_date_of_birth: Weryfikuje, czy użytkownik może pomyślnie zmienić swoją datę urodzenia.
- Test test_delete_account_page_loads: Sprawdza, czy strona potwierdzenia usunięcia konta ładuje się poprawnie.
- Test test_delete_account_with_correct_password: Upewnia się, że użytkownik może usunąć swoje konto po podaniu prawidłowego hasła.
- Test test_delete_account_with_incorrect_password: Sprawdza, czy próba usunięcia konta kończy się niepowodzeniem w przypadku podania nieprawidłowego hasła.

Aplikacja: wallet

- Test test_wallet_balance_display: Sprawdza, czy aktualne saldo portfela użytkownika jest poprawnie wyświetlane na stronie portfela.
- Test test_add_funds_success: Upewnia się, że użytkownik może pomyślnie dodać środki do swojego portfela i saldo zostaje zaktualizowane.

- Test test_add_funds_invalid_amount: Weryfikuje, czy dodanie nieprawidłowej kwoty (np. wartości ujemnej lub zerowej) jest blokowane.
- Test test_withdraw_funds_success: Sprawdza, czy użytkownik może pomyślnie wypłacić środki z portfela i saldo zostaje odpowiednio zmniejszone.
- Test test_withdraw_funds_insufficient_balance: Upewnia się, że wypłata środków jest blokowana w przypadku braku wystarczającego salda.
- Test test_transaction_history_display: Sprawdza, czy historia transakcji portfela (dodania i wypłaty środków) jest poprawnie wyświetlana.
- Test test_invalid_currency_transaction: Weryfikuje, czy transakcja w nieobsługiwanej walucie jest blokowana.
- Test test_currency_conversion_on_transaction: Sprawdza, czy transakcja z użyciem różnych walut poprawnie przelicza wartość przed aktualizacją salda.

Wszystkie testy są zintegrowane z GitHub Actions, dzięki czemu są automatycznie uruchamiane przy każdym wgraniu kodu na repozytorium GitHub, a dodatkowo można je lokalnie uruchomić za pomocą komendy python manage.py test.