Mateusz Kłos
139871

GRA MMO

Technologie obiektowe

Podstawowe cele oraz funkcjonalności

- Możliwość stworzenia swojej postaci o jednej z trzech dostępnych klas
- Zdobywanie doświadczenia i zwiększanie poziomu postaci
- Zdobywanie przedmiotów oraz możliwość ich wyekwipowania
- Statystyki postaci w pełni zależne od klasy, poziomu oraz przedmiotów

Omówienie aplikacji

Projektem jest prosta gra typu MMO. Gra daje nam możliwość zagrania jedną z trzech klas: magiem, wojownikiem lub łucznikiem. Każda klasa posiada oddzielną logikę odnośnie wzmacniania statystyk. Na przykład, mag po zdobyciu poziomu będzie dostawał najwięcej punktów intelektu. Natomiast wojownik, siły. Przedmioty możliwe do wyekwipowania również są podzielone na dane klasy. Bohaterowie mogą wyekwipować tylko przedmioty dla swojej klasy oraz muszą spełniać wymogi poziomowe danego przedmiotu. Na chwilę obecną przedmioty są zdobywane przez bohatera w momencie wbicia poziomu. Trafiają one do jego torby i może on zadecydować czy chce taki przedmiot wyekwipować czy też nie. Poziom jest możliwy do zdobycia poprzez klikanie przycisku.

Strona techniczna

Cały backend został napisany w języku Java korzystając z frameworka Spring. Jest to główna część projektu napisana obiektowo. Aby jednak lepiej zaprezentować działanie aplikacji, został stworzony również bardzo podstawowy frontend, umożliwiający to. Jako, że cały backend to w zasadzie jedno wielkie API, zdecydowałem że bardziej przejrzyste będzie prezentowanie tego za pomocą UI niż używając postmana czy też innych narzędzi tego typu.

Aby uruchomić aplikację trzeba posiadać zainstalowanego docker'a i docker-compose. W rozpakowanym repozytorium (lub pobranym z gita: https://github.com/Klosowsky/mmo_game) używamy następującej komendy:

docker-compose up --build -V --force-recreate

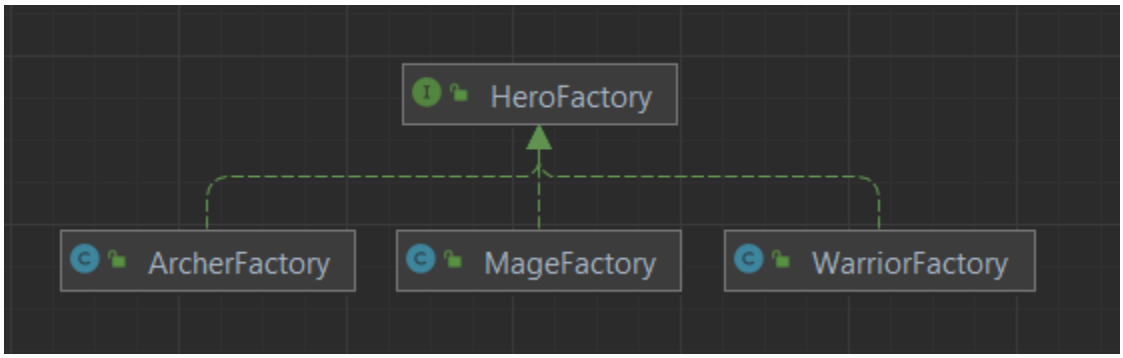
Aplikacja zostanie zbudowana oraz postawiona zostanie baza danych (PostgreSQL) z przykładowymi danymi.

Aby otworzyć UI, musimy wpisać url: *http://localhost:82/*

Wykorzystane wzorce projektowe

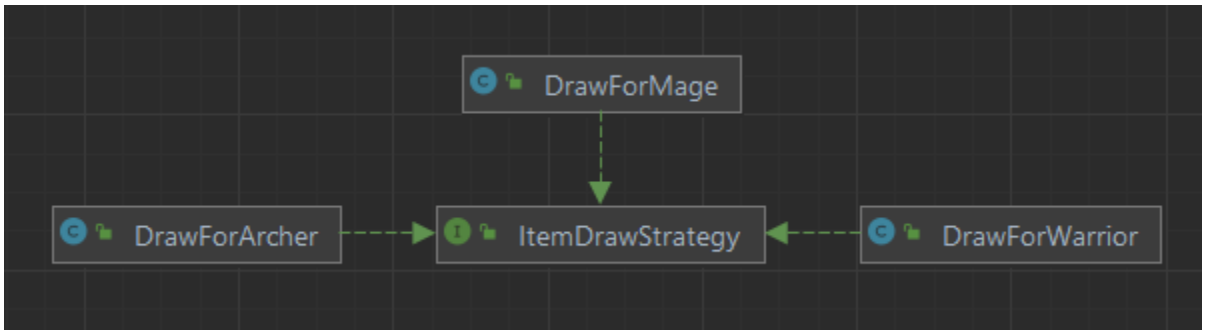
Abstract Factory

Fabryka abstrakcyjna została użyta do tworzenia bohatera. Jako że aplikacja pozwala na utworzenie jednej z trzech klas, fabryka abstrakcyjna idealnie spełnia tu swoje zadanie.



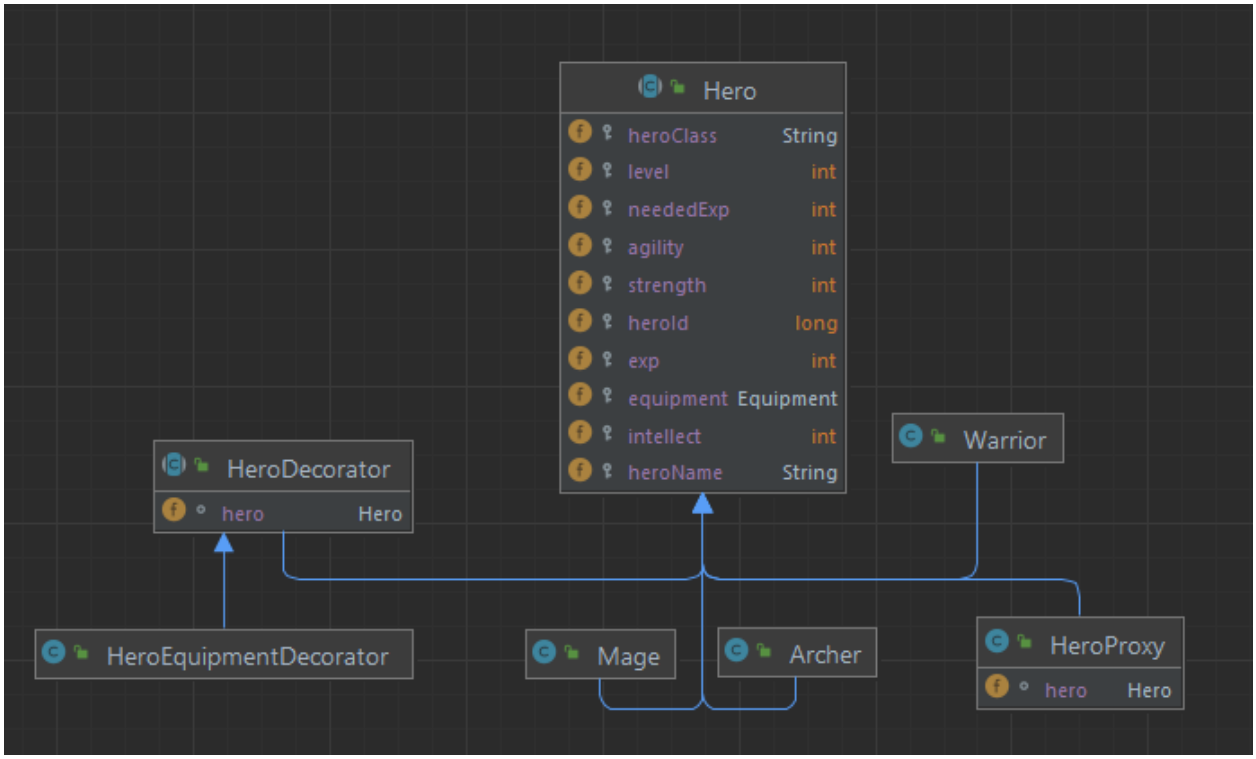
Strategy

Strategia została wykorzystana do zaimplementowania logiki losującej przedmiot dla bohatera. Każdy bohater losuje przedmioty dla swojej klasy.



Decorator

Dekorator jest wykorzystywany do wyświetlania statystyk bohatera z założonym ekwipunkiem. Domyślnie bohater posiada swoje statystyki które są tylko zależne od poziomu i klasy. Użycie dekoratora umożliwia nam pokazanie pełnych statystyk (bohater + ekwipunek).



Proxy

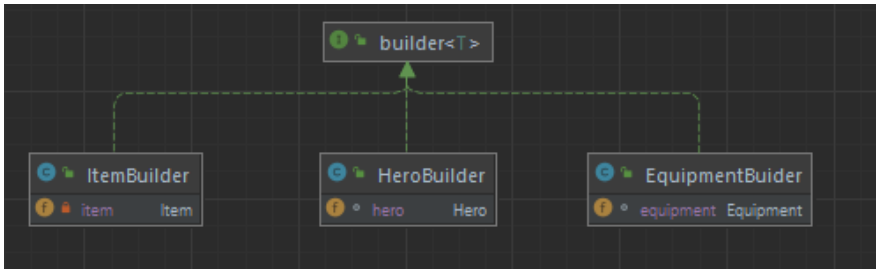
Diagram dla proxy jest widoczny na zrzucie ekranu powyżej. Użyty jest w celach deserializacji JSONa, otrzymanego z requesta. Jako że Hero (główny Entity) jest klasą abstrakcyjną nie możemy stworzyć jego instancji. Pomocny w tym przypadku jest proxy który pośredniczy w tworzeniu odpowiedniego obiektu.

Singleton

Jest wykorzystany do przechowywania rankingu bohaterów. Ranking jest zawsze jeden więc nie ma potrzeby do istnienia dwóch jego instancji.

Builder

Builder wykorzystuje do budowania obiektów bohatera, itemu czy też ekwipunku.



SOLID

Single responsibility

Każda klasa w projekcie odpowiada za pojedynczą, konkretną rzecz.

Open/closed

Zasada mówi że nasz kod powinien być zamknięty na modyfikację oraz otwarty na rozbudowę. W projekcie trzymałem się tej zasady, m.in bardzo często wykorzytując polimorfizm, dziedziczenie oraz wyżej wymienione wzorce projektowe.

Liskov substitution

Zasada podstawienia Liskov mówi o tym, że w miejscu klasy bazowej można użyć dowolnej klasy pochodnej. W projekcie również jest to zachowane a przykładem może być klasa Hero oraz jej subklasy: Archer, Mage, Warrior.

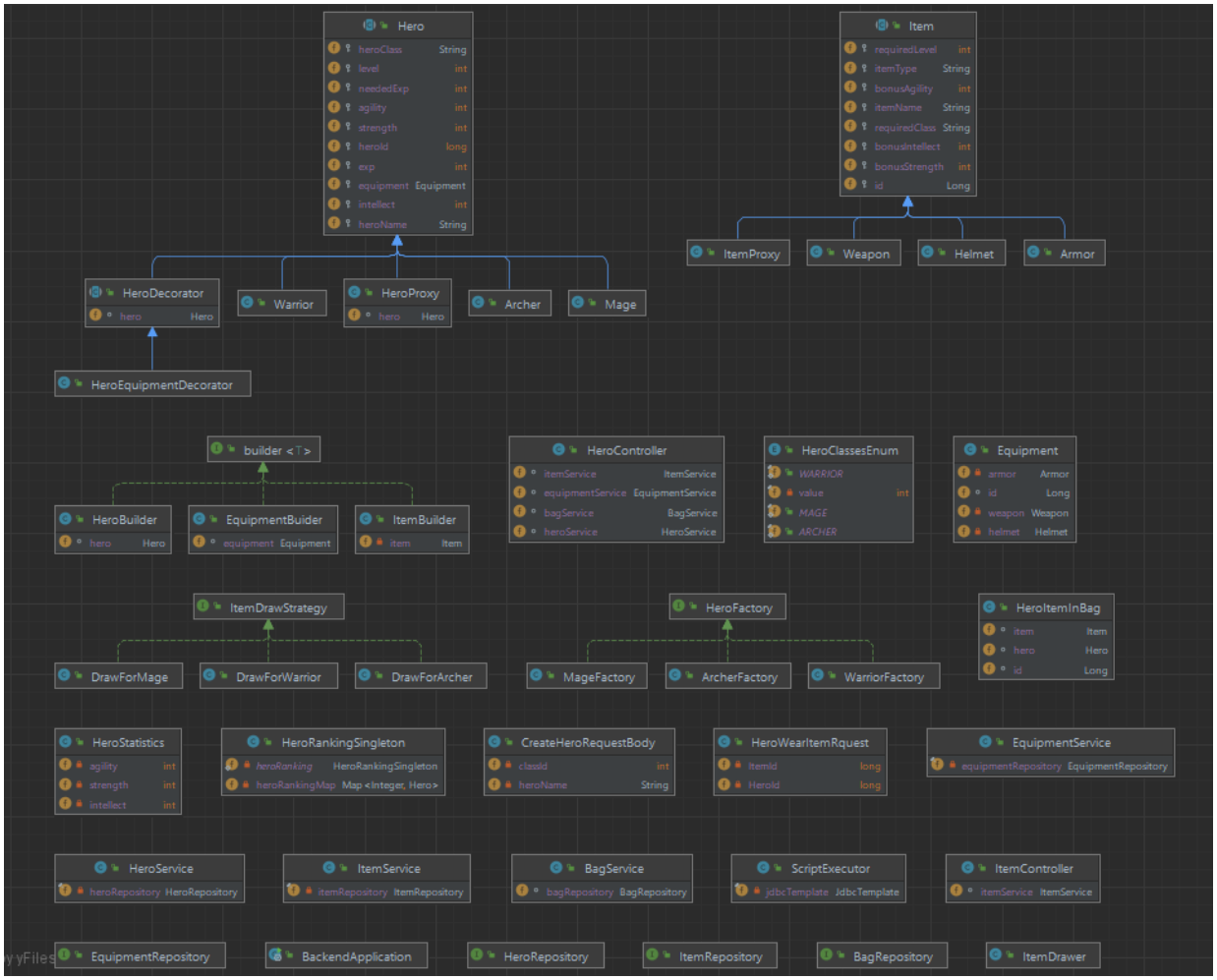
Interface segregation

Interfejsy powinny być takie proste jak tylko się da. W projekcie występuje kilka takich interfejsów a do tworzenia bardziej skomplikowanych abstrakcji użyłem klas abstrakcyjnych. Pozwoliło mi to lepiej opisać konkretny typ.

Dependency Inversion

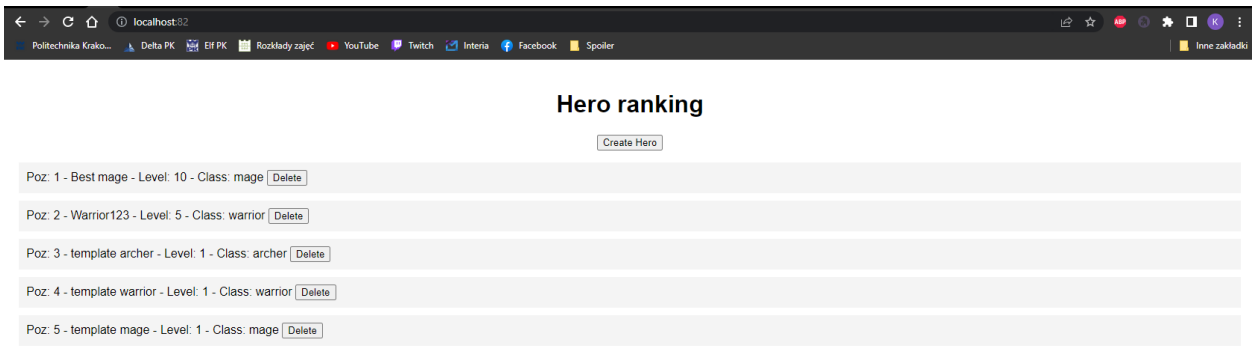
Zasada odwrócenia zależności. W skrócie, zależności powinny zależeć od abstrakcji o ile jest to możliwe. W projekcie jest dużo przykładów a jednymi z nich to klasy Hero czy też Item.

UML

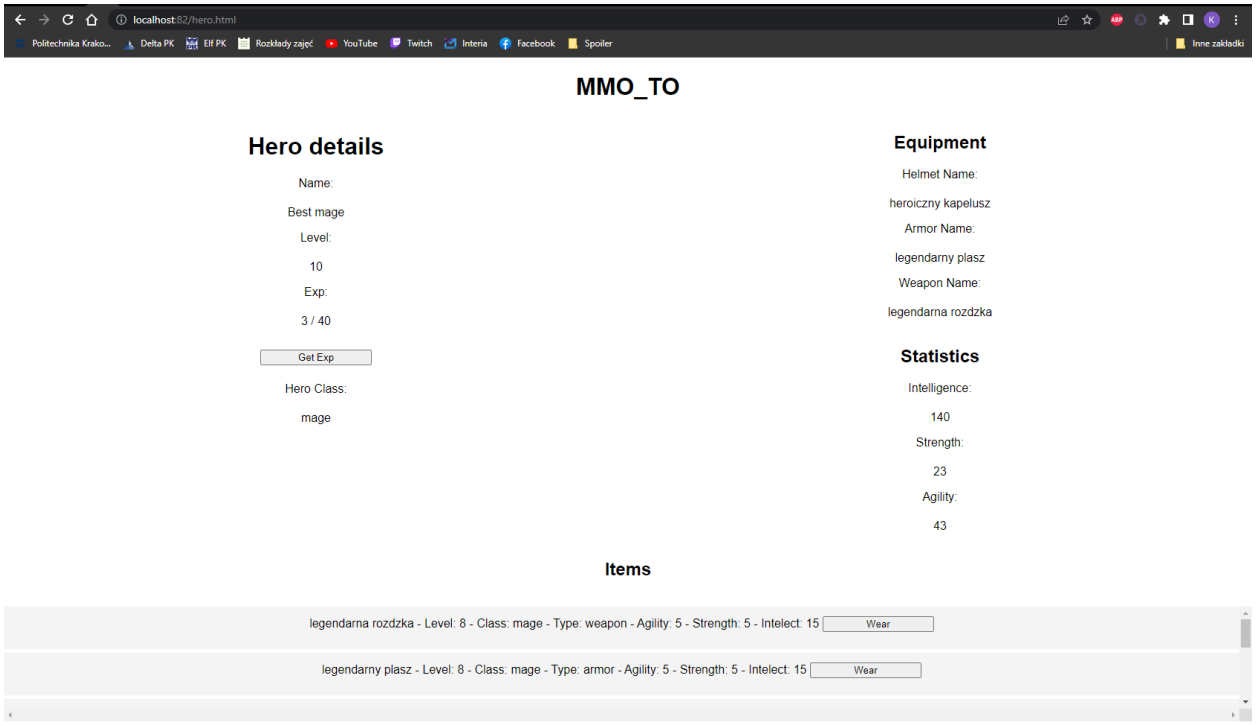


Przykładowe korzystanie z aplikacji

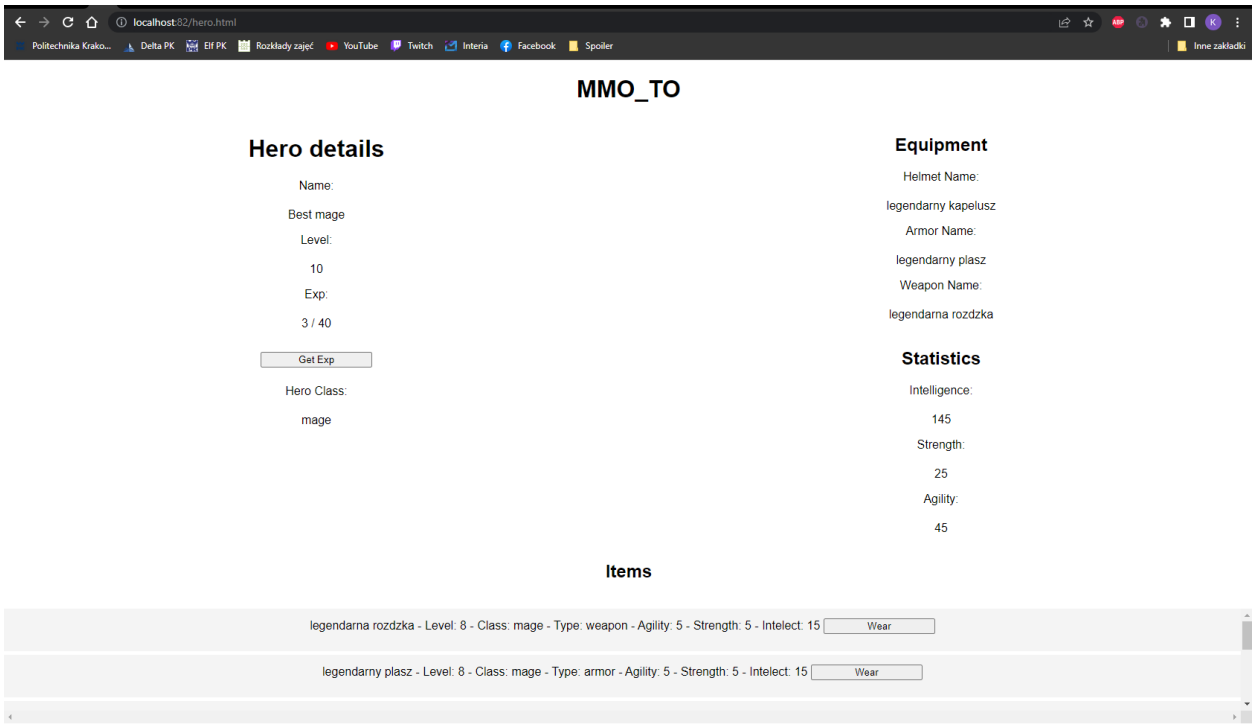
Ranking bohaterów (wg levelu) prezentuje się następująco:



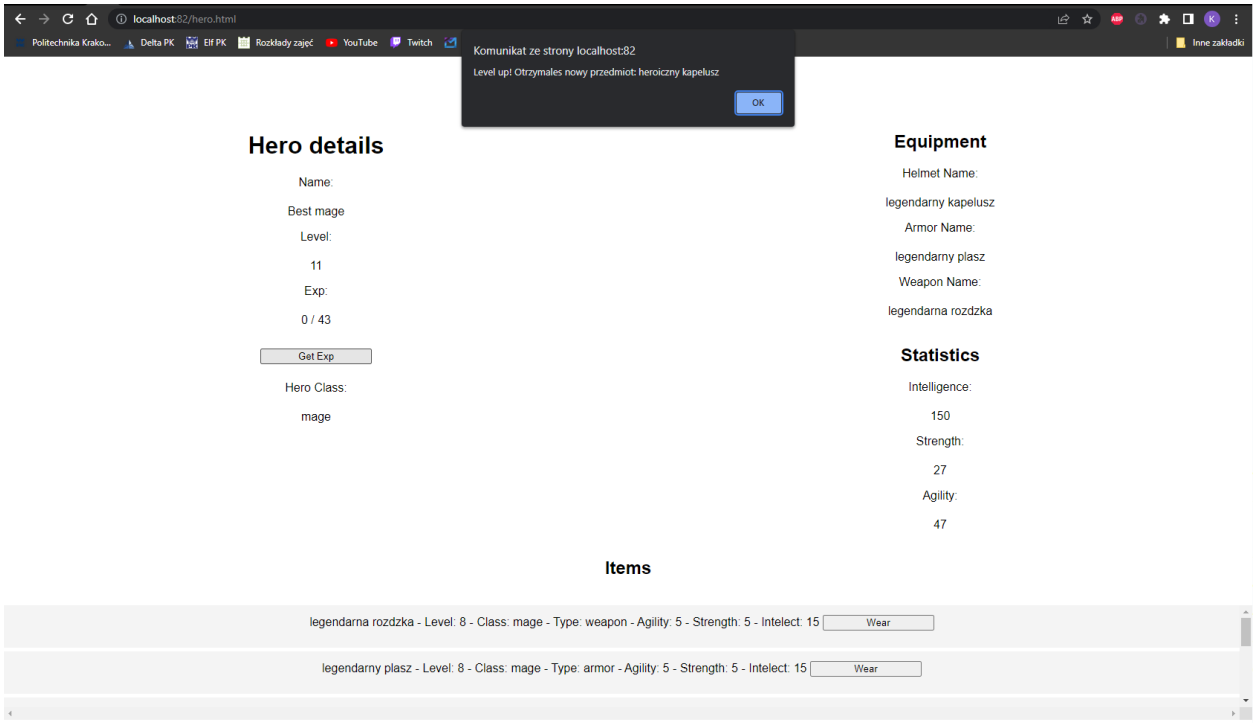
Po wybraniu bohatera widzimy jego szczegóły.



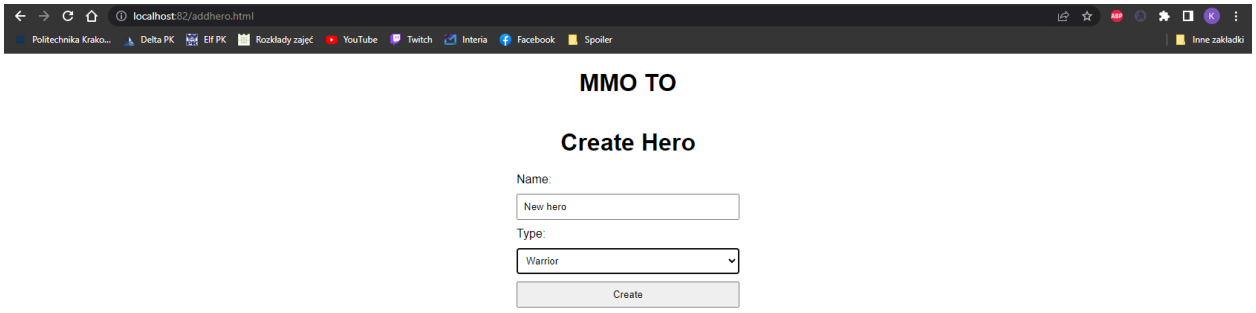
Na tym ekranie możemy np. ubrać inny item z plecaka. Podczas ubierania są sprawdzane wszystkie wymagania (levelowe i klasy) i statystyki są aktualizowane.



Jest również możliwość “expienia”. Działa to na zasadzie clickera. Klikamy Get Exp i po uzyskaniu odpowiedniej ilości punktów doświadczenia, wbijamy poziom - statystyki bohatera są wzmacniane oraz losowany jest przedmiot który trafia do plecaka.



Tworzenie bohatera jest możliwe po kliknięciu przycisku “Create Hero” na stronie domowe. Możemy wybrać imię dla naszego bohatera (musi być unikalne) oraz klasę.



Po utworzeniu, tak prezentuje się “świeży” bohater. Możemy teraz śmiało wbijać poziom i zdobywać przedmioty.