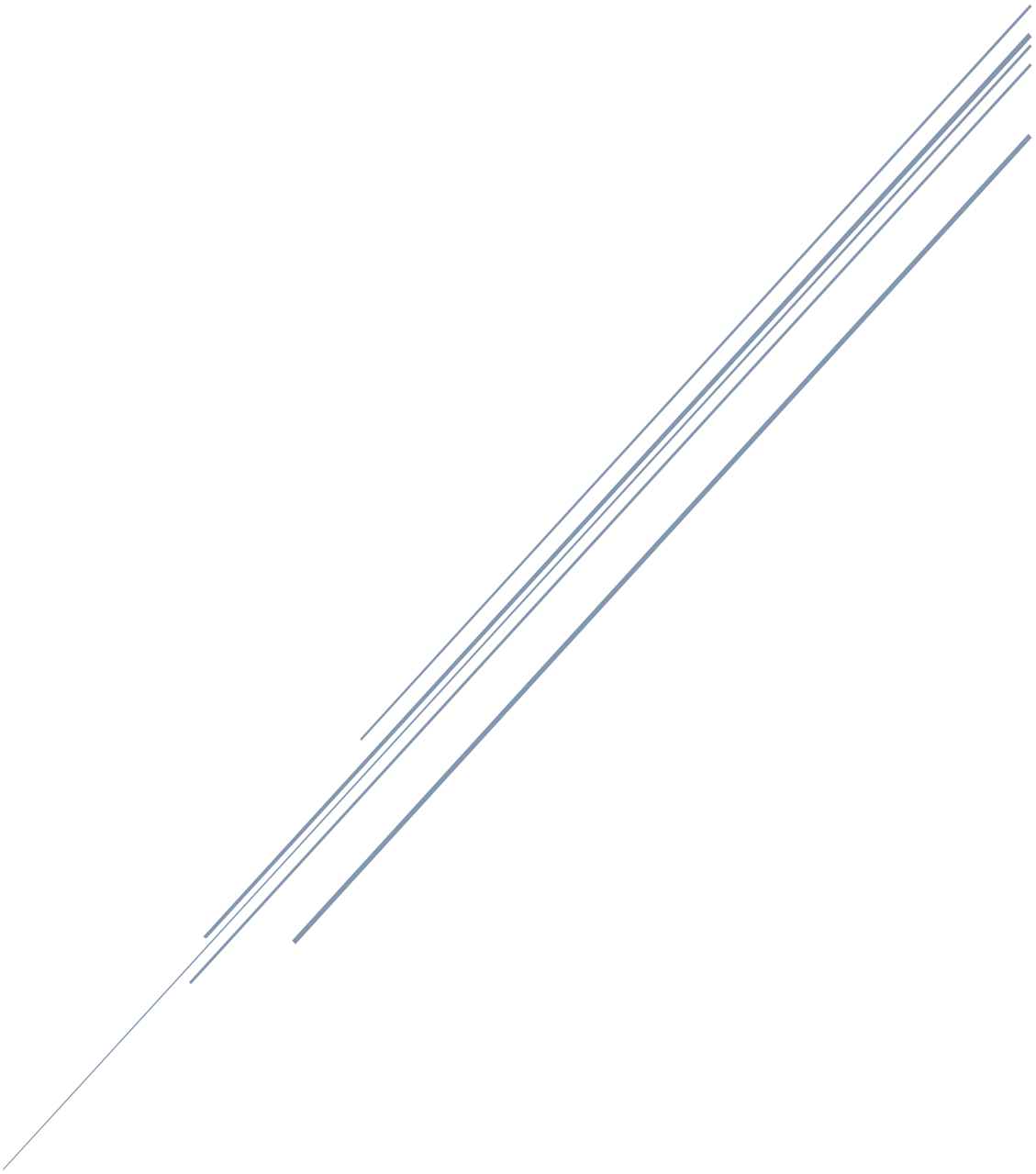
METODOLOGIA, ARCHITEKTURA I WZORCE

Coin Hunter



Uniwersytet Jagielloński, Wydział Matematyki i Informatyki
Inżyniera Oprogramowania

Spis treści

1. Wstęp

Niniejszy dokument przedstawia metodologie, które usprawniały tworzenie aplikacji „Coin Hunter” oraz architekturę i wzorce, z jakich skorzystano. Wszystkie wymienione w niniejszym zestawie rzeczy sprawiły, że projekt „Coin Hunter” dostarczył produkt o tak wysokiej jakości, jak było to tylko możliwe dla jego twórców.

Każda z metodologii, czy też wzorców posiadają opis i powód, dla którego zostały wybrane, czy też wykorzystane.

Symbol ‘$’ następujący po nazwie każdej z wymienionych nazw technologii, narzędzi czy też bibliotek oznacza, że została ona opisana w oddzielnym, specjalnie do tego przeznaczonym pliku o nazwie „Wykorzystane technologie” w folderze „Documentation”.

2. Zestaw

2.1. MVC (Model-View-Controller)

Opis:
Wzorzec architektoniczny służący do organizowania struktury aplikacji posiadających graficzne interfejsy użytkownika. Głównym założeniem MVC jest podzielenie dużej aplikacji na specyficzne obszary, które mają indywidualne zastosowanie. Szczegółowy opis (w języku angielskim) wzorca możemy znaleźć pod adresem „https://blog.logrocket.com/building-structuring-node-js-mvc-application/”.

Powód wykorzystania:
Aplikacja “Coin Hunter” jest aplikacją przeglądarkową. Z racji wykorzystania w jej budowie technologii Node.js[$] postawiono na użycie wzorca MVC, który zapewnia, że kod źródłowy jest bardziej czytelny i lepszej jakości. Dzięki temu może być
w przyszłości łatwiej przejęty przez innych programistów, a także łatwiej go później poprawiać i ulepszać. Kolejnym powodem był fakt, że dzięki MVC łatwo jest zmienić interfejs graficzny, który widzi użytkownik (GUI – Graphical User Interface). Logika biznesowa może pozostać niezmieniona, gdy zmienia się wygląd interfejsu użytkownika, te dwa fragmenty kodu są od siebie oddzielone.

2.2. OOP (Object Oriented Programming)

Opis:
Sposób tworzenia oprogramowania wykorzystujący obiekty — elementy scalające dane i funkcje. Program składa się z obiektów komunikujących się ze sobą
i realizujących różne zadania.
Źródło: https://pl.wikipedia.org/wiki/Programowanie_obiektowe.

Powód wykorzystania:
Głównym powodem, dla którego OOP znalazło zastosowanie w projekcie „Coin Hunter”, jest ułatwienie rozwijania już istniejącego kodu. Początkowo cała aplikacja,
a przede wszystkim gra były tworzone za pomocą języka programowania JavaScript[$], w którym możliwości pisania obiektowego są znacząco ograniczone. W celu zmiany powyższego faktu postanowiono wzbogacić projekt technologią TypeScript[$]. Dzięki właściwościom języka TS (TypeScript) kod stał się łatwiejszy do pisania, modyfikowania oraz utrzymania.

2.3. SOLID

Opis:
Sama nazwa „SOLID” jest akronimem. Akronim ten zawiera zestaw wytycznych. Wytyczne te stosuje się podczas pisania programów w sposób obiektowy. Każda litera w akronimie pochodzi od danej wytycznej. Są to kolejno:
S – Single Responsibility Principle,
O - Open/Closed Principle,
L - Liskov Substitution Principle,
I - Interface Segregation Principle,
D - Dependency Inversion Principle.
Źródło: https://pl.wikipedia.org/wiki/SOLID_(programowanie_obiektowe).

Powód wykorzystania:
Z racji świadomości o istnieniu wytycznych SOLID, nasz zespół już przy wstępnej analizie projektu wiedział, że będzie chciał się do nich stosować, gdyż przy rozsądnym i świadomym ich użyciu zwiększa ona jakość kodu, a tym samym produktu dostarczonego klientowi.

2.4. Kanban

Opis:
Metoda Kanban to zwinna metodyka zarządzania przepływem pracy oraz wizualizacji procesów biznesowych. Metoda ta służy optymalizowaniu pracy i procesów w celu maksymalizacji wydajności, minimalizacji marnotrawstwa dzięki wizualizacji pracy oraz ciągłemu doskonaleniu procesów na podstawie raportów i analizy.
Źródło: https://pl.wikipedia.org/wiki/Kanban.

Powód wykorzystania:
Nasz zespół stał przed wyborem dwóch metod, który ułatwiliby mu pracę nad projektem i zmaksymalizowało efektywność. Ostateczny wybór padł na metodę Kanban, która wydała się nam w naszym projekcie bardziej elastyczna. Na początku drogi tworzenia aplikacji „*Coin Hunter*” nasz zespół podzielił się na trzy grupy, w skład której wchodziły po dwie osoby. Tak też powstała grupa zajmująca się stroną aplikacji i jego serwerem, grupa od gry i testów oraz grupa od dokumentacji. Każda
z grup, miała swoje własne, wewnętrzne cele, które na bieżąco aktualizowała i wykonywała. Dodatkowo pracowaliśmy jako jedna, duża drużyna, sześcioosobowa, która miała wspólną TODO listę. Dzięki tak zorganizowanej pracy każdy wiedział, jakie zadania powinien wykonywać. Zmniejszyło to chaos w procesie wykonawczym i znacząco ułatwiło pracę nad projektem.

2.5. Fabryka abstrakcyjna

Opis
Kreacyjny wzorzec projektowy, którego celem jest dostarczenie interfejsu do tworzenia różnych obiektów jednego typu (tej samej rodziny) bez specyfikowania ich konkretnych klas. Umożliwia jednemu obiektowi tworzenie różnych, powiązanych ze sobą, reprezentacji podobiektów określając ich typy podczas działania programu.
Źródło: https://pl.wikipedia.org/wiki/Fabryka_abstrakcyjna.

Powód wykorzystania:
Wzorzec Fabryka Abstrakcyjna umożliwia bardzo szybkie i nieskomplikowane rozszerzanie aplikacji o kolejne obiekty gry (takie jak Coin czy Enemy). Dodanie kolejnego typu Enemy(Coin) wiąże się jedynie z dodaniem nowej klasy pochodnej do AbstractEntity reprezentującej ten obiekt oraz metody Create w odpowiedniej fabryce.

Fabryka abstrakcyjna jest realizowana w projekcie poprzez abstrakcyjną klasę EntityFactory oraz implementujące ją klasy CoinFactory i EnemyFactory zwracające odpowiednio obiekty Coin i Enemy, które są pochodnymi klasy AbstractEntity.