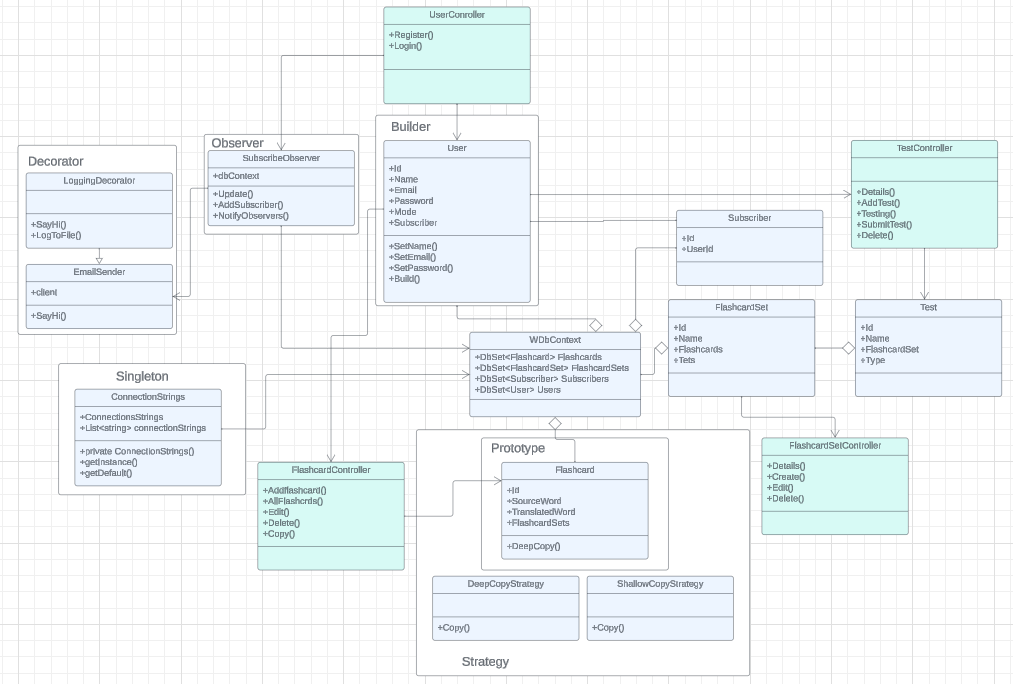
|  |  |
| --- | --- |
| Wydział Informatyki Politechniki Białostockiej  Zaawansowane Techniki Programistyczne | **Data:** 22.01.2024r. |
| Dokumentacja aplikacji  **Grupa**: PS 8   1. Jan Dec 2. Ewa Dobrzycka 3. Dawid Dzienisiewicz | **Prowadzący:**  dr inż. Daniel Reska |

**1.** **Diagram klas**

****

**2.** **Krótki opis każdego wzorca**

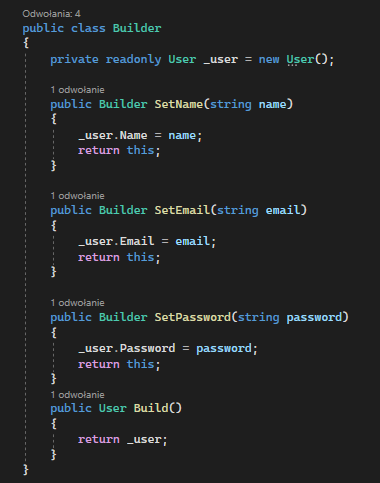
**Budowniczy**

* **Cel użycia:**

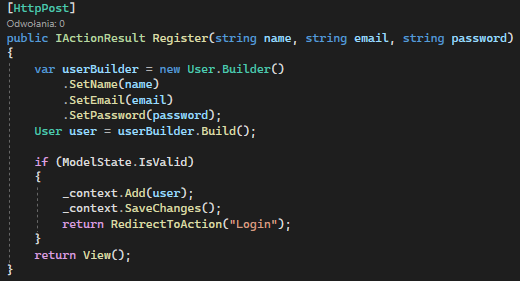
Efektywne i elastyczne tworzenie obiektów *User*, z możliwością kontrolowania procesu ich budowy.

* **Przyporządkowanie klas do wszystkich ról wzorca:**
  + **Product (Produkt)**: Klasa *User* zawierająca właściwości, które mają być ustawione przez Budowniczego, takie jak *Name, Email* czy *Password*.
  + **Builder (Budowniczy):** Klasa wewnętrzna *Builder* w klasie *User* odpowiadająca za definiowanie interfejsu do budowy części składowych obiektu.
  + **Client (Klient):** Klasa *UserController* pełni rolę Klienta, który korzysta z Budowniczego do tworzenia obiektów. W metodzie *Register*, tworzy on nowy obiekt User za pomocą klasy Builder.
* **Lokalizacja wzorca w kodzie:**

Wzorzec jest zaimplementowany w klasie *User*. Konstrukcja obiektu User odbywa się poprzez wywołanie metod *SetName, SetEmail, SetPassword* na instancji klasy *Builder*, a następnie wywołanie *Build* do uzyskania gotowego obiektu *User.*



Użycie wzorca w metodzie Register:



**Singleton**

* **Cel użycia:**

Dzięki zastosowaniu wzorca Singleton w aplikacji, instancja klasy ConnectionStrings, przechowująca nazwy połączeń do baz danych, jest tworzona jednokrotnie. To zapobiega wielokrotnemu tworzeniu tej instancji, co eliminuje potencjalne problemy związane z konfliktami i nadmiernym zużyciem zasobów.

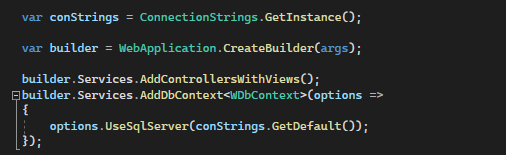
* **Przyporządkowanie klas do wszystkich ról wzorca:**

Klasa *ConnectionStrings* jest właściwym Singletonem. Odpowiada za przechowywanie jednej instancji siebie samej oraz za dostarczanie globalnego punktu dostępu do tej instancji.

* **Lokalizacja wzorca w kodzie:**

****

Użycie:

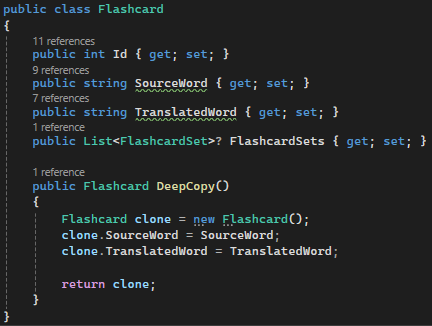
****

**Prototype**

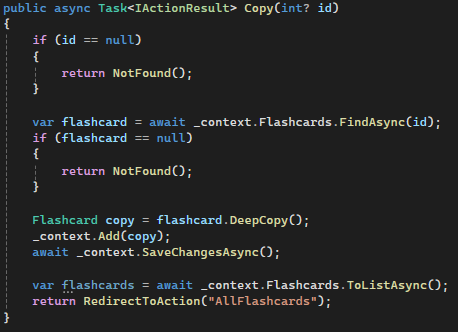
* **Cel użycia:**

Izolacja danych i uniknięcie niepożądanych efektów ubocznych, gdy operujesz na różnych kopii obiektów Flashcard.

* **Przyporządkowanie klas do wszystkich ról wzorca:**
  + **Prototyp (Prototype):** metoda *DeepCopy()* klasy *Flashcard*, która tworzy głęboką kopię obiektu.
  + **Konkretny Prototyp (ConcretePrototype):** klasa *Flashcard* zawierająca metodę *DeepCopy(),* która wykonuje głęboką kopię samej siebie.
  + **Klient (Client):** Metoda *Copy* w klasie *FlashcardController* która wywołuje metodę *DeepCopy()* na instancji *Flashcard*, tworząc w ten sposób kopię.
* **Lokalizacja wzorca w kodzie:**



Użycie:

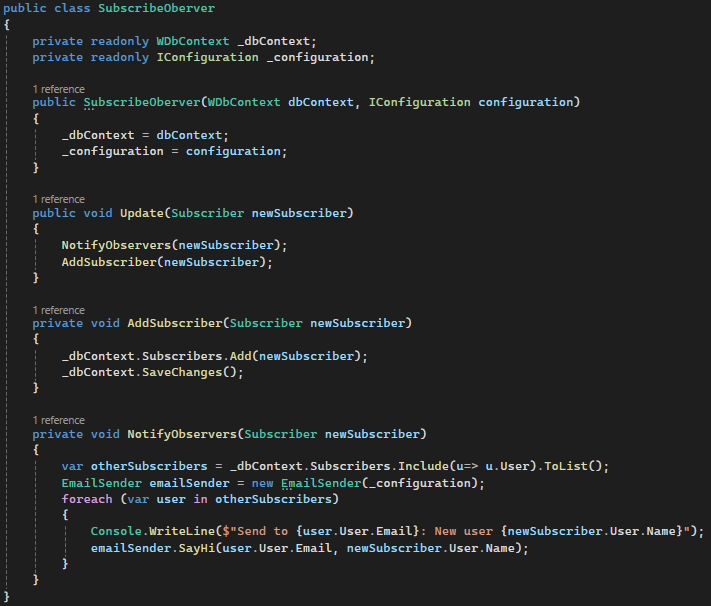


**Observer**

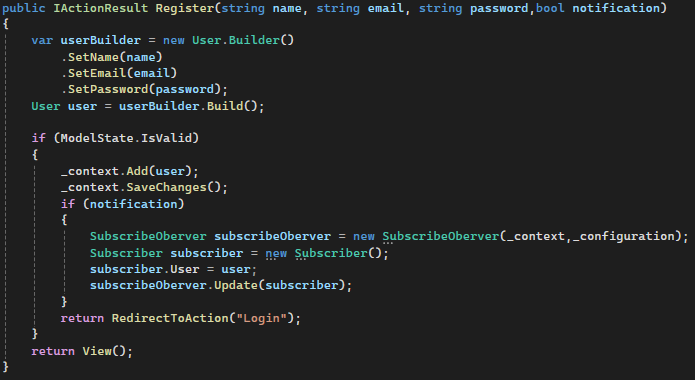
* **Cel użycia:**

Reagowanie na zdarzenie rejestracji użytkownika który zaznaczył opcję w formularzu logowania. Dzięki wykorzystaniu tego wzorca możemy przesyłać informację powitalną dla innych użytkowników.

* **Przyporządkowanie klas do wszystkich ról wzorca:**
  + **Subject(Podmiot):** Klasa “SubscribeOberver” pełni tę funkcję
  + **Observer (Obserwator):** Klasa “SubscribeOberver” pełni tę funkcję
* **Lokalizacja wzorca w kodzie:**

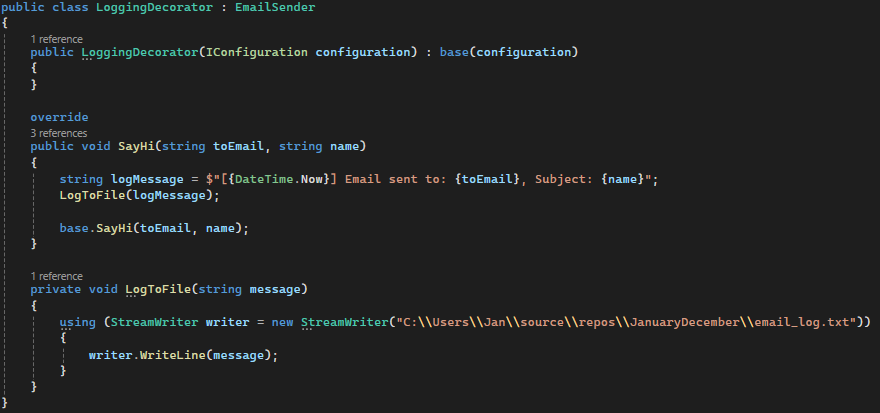


Użycie:

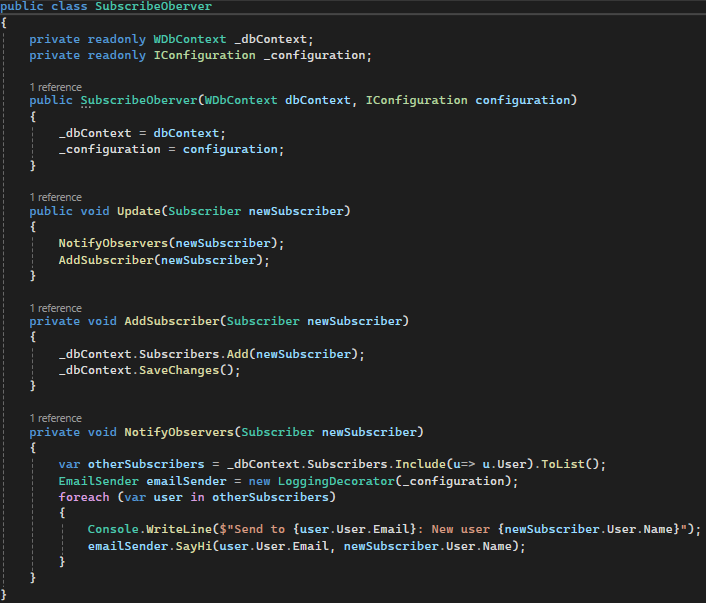


**Decorator**

* **Cel użycia:** Wzorzec dekorator został użyty w celu opakowania EmailSender o dodatkowe funkcję umożliwiającą zapisywanie wysłany e-maili do pliku
* **Przyporządkowanie klas do wszystkich ról wzorca:**
  + **Component (Komponent):** Klasa “EmailSender” pełni tę funkcję
  + **Decorator (Dekorator):** Klasa “LoggingDecorator” pełni tę funkcję
* **Lokalizacja wzorca w kodzie:**

****

Użycie:

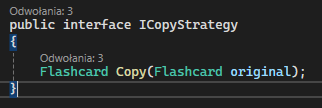
****

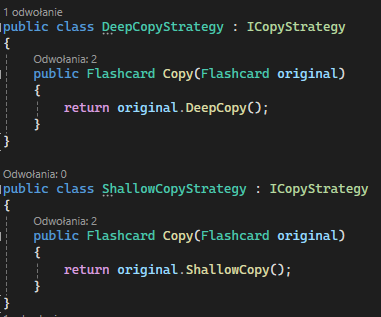
**Strategy**

* **Cel użycia:**

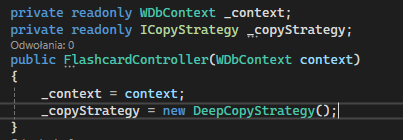
Zapewnienie elastyczności w wyborze metody kopiowania obiektów *Flashcard*. Dzięki temu można łatwo zmienić sposób kopiowania (np. z głębokiej kopii na płytką kopię), bez konieczności modyfikacji kodu klasy *FlashcardController*.

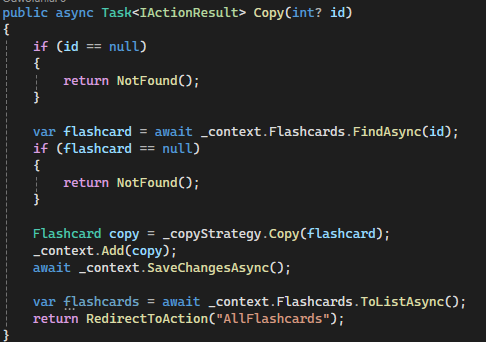
* **Przyporządkowanie klas do wszystkich ról wzorca:**
  + **Context (Kontekst):** *FlashcardController* **-** klasa używająca strategii. Jest odpowiedzialna za przechowywanie referencji do konkretnej strategii *(ICopyStrategy)* i delegowanie jej pracy. Używa strategii do kopiowania obiektów Flashcard.
  + **Strategy Interface (Interfejs Strategii):** *ICopyStrategy* - interfejs, który definiuje rodziny wymiennych algorytmów. Posiada metodę *Copy,* która jest wspólna dla wszystkich strategii kopiowania.
  + **Concrete Strategies (Konkretne Strategie):** *DeepCopyStrategy, ShallowCopyStrategy*- klasy, które implementują interfejs strategii. Każda z nich reprezentuje konkretny algorytm, który można zastosować w kontekście. *DeepCopyStrategy* implementuje głęboką kopię obiektu Flashcard, podczas gdy *ShallowCopyStrategy* implementuje płytką kopię.
* **Lokalizacja wzorca w kodzie:**

****

****

Użycie:

****

****

**3.**  **Opis co najmniej jednego rozwiązania specyficznego dla użytej technologii**

**Użycie Routingu w MVC**

W architekturze MVC, jednym z kluczowych elementów jest system routingu, który zarządza przekierowywaniem żądań od użytkownika do odpowiednich kontrolerów. Routing jest szczególnie istotny w przypadku aplikacji internetowych, gdzie różne adresy URL muszą być skojarzone z konkretnymi funkcjonalnościami.

**Wykorzystane źródło wiedzy na ten temat:**

* <https://www.tutorialsteacher.com/mvc/routing-in-mvc>
* <https://learn.microsoft.com/pl-pl/aspnet/mvc/overview/older-versions-1/controllers-and-routing/asp-net-mvc-routing-overview-cs>

**4.** **Opis podziału pracy w zespole :**

* **Jan Dec**
  + *Answer.cs*
  + *Test.cs*
  + *TestHistory.cs*
  + *TestController.cs*
  + Test
    - *Addtest.cshtml*
    - *Details.cshtml*
    - *Index.cshtml*
    - *Starttest.cshtml*
* **Ewa Dobrzycka**
  + *User.cs*
  + *Flashcard.cs*
  + *FlashcardController.cs*
  + *UserController.cs*
  + Flashcard
    - *AddFlashcard.cshtml*
    - *AllFlashcards.cstml*
    - *Delete.cshtml*
    - *Edit.cshtml*
  + User
    - *Login.cshtml*
    - *Register.cshtml*
* **Dawid Dzienisiewicz**
  + *FlashcardSet.cs*
  + *EmailSender.cs*
  + *SubscribeOberver.cs*
  + *Subscriber.cs*
  + *ConnectionStrings.cs*
  + *FlashcardSetController.cs*
  + FlashcardSet
    - *Create.cshtml*
    - *Delete.cshtml*
    - *Details.cshtml*
    - *Edit.cshtml*
    - *Index.cshtml*

**5.**  **Instrukcja użytkownika**

Każda osoba, aby w pełni korzystać z funkcjonalności systemu, musi się zarejestrować. W procesie rejestracji znajduje się opcja "Say Hi", która po zaznaczeniu powoduje wysłanie e-maila do wszystkich użytkowników o dołączeniu nowej osoby. Po zalogowaniu, użytkownik ma możliwość tworzenia, klonowania, edytowania i usuwania swoich fiszek, definiując słowo i jego tłumaczenie na wybrany język. Można również tworzyć, edytować i usuwać zestawy fiszek. System oferuje także funkcję tworzenia testów bazujących na stworzonych zestawach. Po ukończeniu testu użytkownik otrzymuje informację o liczbie poprawnych odpowiedzi, co pozwala na ocenę wiedzy z danego zestawu słówek.

**6.** **Instrukcja instalacji:**

* **Wymagania Wstępne:**Visual Studio: Upewnij się, że masz zainstalowane Visual Studio, najlepiej najnowszą dostępną wersję.  
  .NET Core: Upewnij się, że zainstalowałeś najnowszą wersję .NET 7.0
* **Krok 1: Utworzenie Projektu ASP.NET MVC**

Otwórz Visual Studio.

Sklonuj repozytorium: [https://github.com/JanuaryDecember/ZTPAPP](https://github.com/JanuaryDecember/ZTPAPP?fbclid=IwAR1pKSknmCfwSBadsEQbN8AOxj_NEbjNGe-WnzLDtwnqT74NQl3MApIohrY)

* **Krok 2: Uruchomienie Aplikacji**

Uruchom aplikację w Visual Studio.