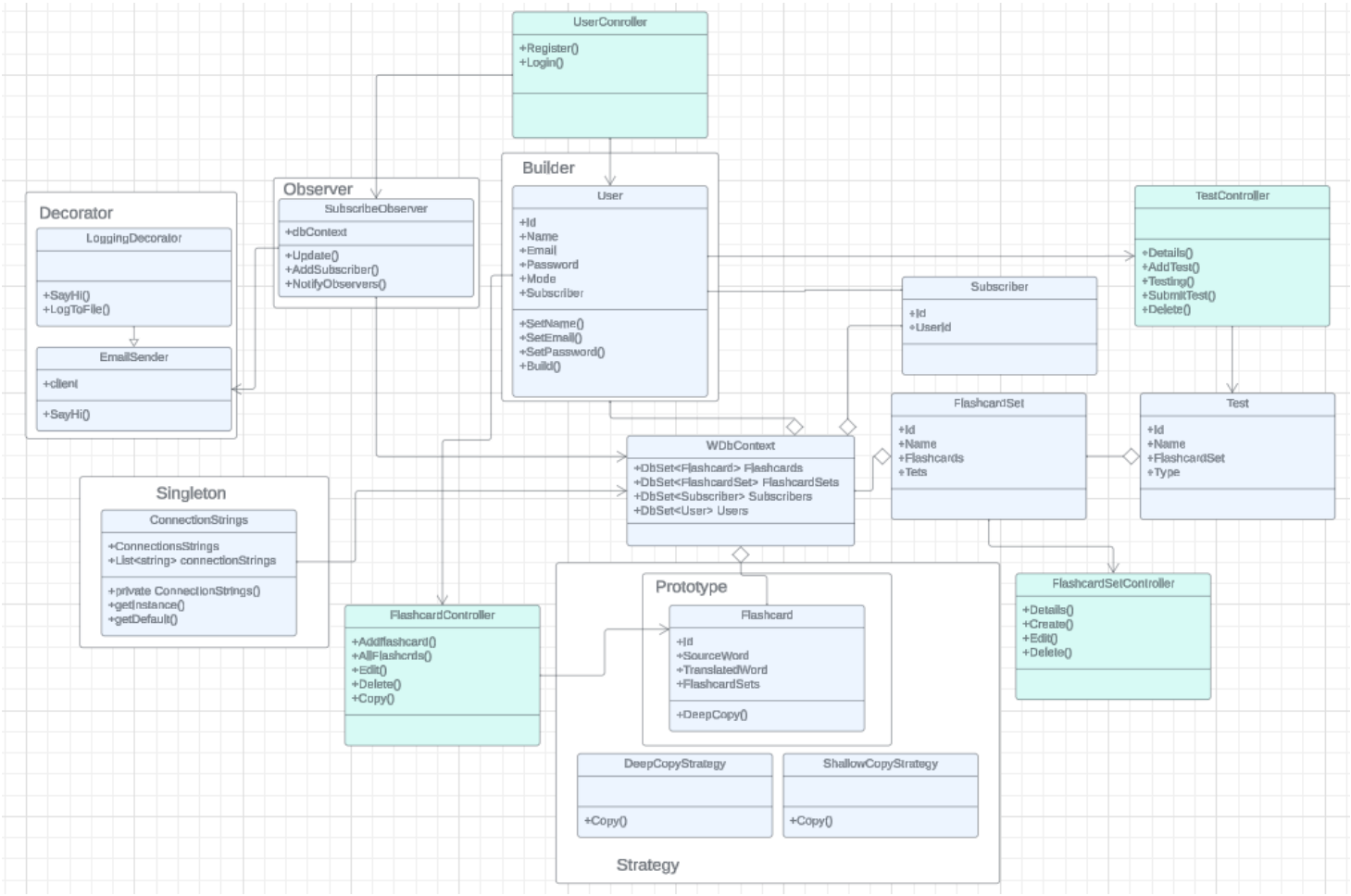
Wydział Informatyki Politechniki Białostockiej Zaawansowane Techniki Programistyczne	**Data:** 22.01.2024r.
Dokumentacja aplikacji **Grupa**: PS 8 1. Jan Dec 2. Ewa Dobrzycka 3. Dawid Dzienisiewicz	**Prowadzący:** dr inż. Daniel Reska

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.*

```
Odwołania: 4
public class Builder
{
    private readonly User _user = new User();

    1 odwołanie
    public Builder SetName(string name)
    {
        _user.Name = name;
        return this;
    }

    1 odwołanie
    public Builder SetEmail(string email)
    {
        _user.Email = email;
        return this;
    }

    1 odwołanie
    public Builder SetPassword(string password)
    {
        _user.Password = password;
        return this;
    }
    1 odwołanie
    public User Build()
    {
        return _user;
    }
}
```

Użycie wzorca w metodzie Register:

```
[HttpPost]
Odwołania: 0
public IActionResult Register(string name, string email, string password)
{
    var userBuilder = new User.Builder()
        .SetName(name)
        .SetEmail(email)
        .SetPassword(password);
    User user = userBuilder.Build();

    if (ModelState.IsValid)
    {
        _context.Add(user);
        _context.SaveChanges();
        return RedirectToAction("Login");
    }
    return View();
}
```

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:**

```
Odwołania: 5
public class ConnectionStrings
{
    private static ConnectionStrings instance;
    private Dictionary<string, string> connectionStringsDic = new Dictionary<string, string>();

    1 odwołanie
    private ConnectionStrings()
    {
        connectionStringsDic.Add("Default", "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=Projekt_ZTP;
    }
    1 odwołanie
    public static ConnectionStrings GetInstance()
    {
        if (instance == null)
        {
            instance = new ConnectionStrings();
        }
        return instance;
    }
    1 odwołanie
    public string GetDefault()
    {
        if (connectionStringsDic.ContainsKey("Default"))
            return connectionStringsDic["Default"];

        return "";
    }
}
```

Użycie:

```
var conStrings = ConnectionStrings.GetInstance();

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<WDbContext>(options =>
{
    options.UseSqlServer(conStrings.GetDefault());
});
```

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:**

```
public class Flashcard
{
    11 references
    public int Id { get; set; }
    9 references
    public string SourceWord { get; set; }
    7 references
    public string TranslatedWord { get; set; }
    1 reference
    public List<FlashcardSet>? FlashcardSets { get; set; }

    1 reference
    public Flashcard DeepCopy()
    {
        Flashcard clone = new Flashcard();
        clone.SourceWord = SourceWord;
        clone.TranslatedWord = TranslatedWord;

        return clone;
    }
}
```

Użycie:

```
public async Task<IActionResult> Copy(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var flashcard = await _context.Flashcards.FindAsync(id);
    if (flashcard == null)
    {
        return NotFound();
    }

    Flashcard copy = flashcard.DeepCopy();
    _context.Add(copy);
    await _context.SaveChangesAsync();

    var flashcards = await _context.Flashcards.ToListAsync();
    return RedirectToAction("AllFlashcards");
}
```

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:**

```
public class SubscribeOberver
{
    private readonly WDbContext _dbContext;
    private readonly IConfiguration _configuration;

    1 reference
    public SubscribeOberver(WDbContext dbContext, IConfiguration configuration)
    {
        _dbContext = dbContext;
        _configuration = configuration;
    }

    1 reference
    public void Update(Subscriber newSubscriber)
    {
        NotifyObservers(newSubscriber);
        AddSubscriber(newSubscriber);
    }

    1 reference
    private void AddSubscriber(Subscriber newSubscriber)
    {
        _dbContext.Subscribers.Add(newSubscriber);
        _dbContext.SaveChanges();
    }

    1 reference
    private void NotifyObservers(Subscriber newSubscriber)
    {
        var otherSubscribers = _dbContext.Subscribers.Include(u=> u.User).ToList();
        EmailSender emailSender = new EmailSender(_configuration);
        foreach (var user in otherSubscribers)
        {
            Console.WriteLine($"Send to {user.User.Email}: New user {newSubscriber.User.Name}");
            emailSender.SayHi(user.User.Email, newSubscriber.User.Name);
        }
    }
}
```

Użycie:

```
public IActionResult Register(string name, string email, string password,bool notification)
{
    var userBuilder = new User.Builder()
        .SetName(name)
        .SetEmail(email)
        .SetPassword(password);
    User user = userBuilder.Build();

    if (ModelState.IsValid)
    {
        _context.Add(user);
        _context.SaveChanges();
        if (notification)
        {
            SubscribeOberver subscribeOberver = new SubscribeOberver(_context,_configuration);
            Subscriber subscriber = new Subscriber();
            subscriber.User = user;
            subscribeOberver.Update(subscriber);
        }
        return RedirectToAction("Login");
    }
    return View();
}
```

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:**

```
public class LoggingDecorator : EmailSender
{
    1 reference
    public LoggingDecorator(IConfiguration configuration) : base(configuration)
    {
    }

    override
    3 references
    public void SayHi(string toEmail, string name)
    {
        string logMessage = $"[{DateTime.Now}] Email sent to: {toEmail}, Subject: {name}";
        LogToFile(logMessage);

        base.SayHi(toEmail, name);
    }

    1 reference
    private void LogToFile(string message)
    {
        using (StreamWriter writer = new StreamWriter("C:\\Users\\Jan\\source\\repos\\JanuaryDecember\\email_log.txt"))
        {
            writer.WriteLine(message);
        }
    }
}
```

Użycie:

```
public class SubscribeOberver
{
    private readonly WDbContext _dbContext;
    private readonly IConfiguration _configuration;

    1 reference
    public SubscribeOberver(WDbContext dbContext, IConfiguration configuration)
    {
        _dbContext = dbContext;
        _configuration = configuration;
    }

    1 reference
    public void Update(Subscriber newSubscriber)
    {
        NotifyObservers(newSubscriber);
        AddSubscriber(newSubscriber);
    }

    1 reference
    private void AddSubscriber(Subscriber newSubscriber)
    {
        _dbContext.Subscribers.Add(newSubscriber);
        _dbContext.SaveChanges();
    }

    1 reference
    private void NotifyObservers(Subscriber newSubscriber)
    {
        var otherSubscribers = _dbContext.Subscribers.Include(u=> u.User).ToList();
        EmailSender emailSender = new LoggingDecorator(_configuration);
        foreach (var user in otherSubscribers)
        {
            Console.WriteLine($"Send to {user.User.Email}: New user {newSubscriber.User.Name}");
            emailSender.SayHi(user.User.Email, newSubscriber.User.Name);
        }
    }
}
```

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:**

```
Odwołania: 3
public interface ICopyStrategy
{
    Odwołania: 3
    Flashcard Copy(Flashcard original);
}
```

```
1 odwołanie
public class DeepCopyStrategy : ICopyStrategy
{
    Odwołania: 2
    public Flashcard Copy(Flashcard original)
    {
        return original.DeepCopy();
    }
}

Odwołania: 0
public class ShallowCopyStrategy : ICopyStrategy
{
    Odwołania: 2
    public Flashcard Copy(Flashcard original)
    {
        return original.ShallowCopy();
    }
}
```

Użycie:

```
private readonly WDbContext _context;
private readonly ICopyStrategy _copyStrategy;
Odwołania: 0
public FlashcardController(WDbContext context)
{
    _context = context;
    _copyStrategy = new DeepCopyStrategy();
}
```

```
public async Task<IActionResult> Copy(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var flashcard = await _context.Flashcards.FindAsync(id);
    if (flashcard == null)
    {
        return NotFound();
    }

    Flashcard copy = _copyStrategy.Copy(flashcard);
    _context.Add(copy);
    await _context.SaveChangesAsync();

    var flashcards = await _context.Flashcards.ToListAsync();
    return RedirectToAction("AllFlashcards");
}
```

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
- **Krok 2: Uruchomienie Aplikacji**
  Uruchom aplikację w Visual Studio.