**Design patterns and diagrams**

Template Number 1 – Factory Methods

**Use of the Pattern:**

The reason for choosing this pattern arises from the recognition that we are dealing with a polymorphic family of classes inheriting from the Form class. The choice to implement this pattern was made to move the responsibility of creation from the Client to a separate class, contributing to code maintainability (e.g., if we want to expand the system to include additional forms or make changes, we only need to modify the factory class). Separating the creation logic from the Client prevents code duplication with identical logic.

**Implementation Approach:**

We created a static class called FormFactory, which contains a static function called CreateForm. This function takes parameters: eFormType, FriendLogic, FindMyMatchFacade, Album, and bool. It creates the relevant Form based on eFormType and returns it.

Sequence Diagram

תמונה שמכילה שולחן

התיאור נוצר באופן אוטומטי

Class Diagram

תמונה שמכילה תרשים, שרטוט, תוכנית, שרטוט טכני

התיאור נוצר באופן אוטומטי

Template Number 2 – Facade

**Use of the Pattern:**

The choice to implement this pattern stems from the fact that MyOptionalMatchesForm and SelectedMatchForm (which serve as the clients) make direct calls to FindMyMatchLogic and UserLogic (which are part of the logical components). Implementing the Facade simplifies the usage of the logical component and abstracts the data, making it more manageable. Additionally, when clients have direct access to each sub-system, managing and maintaining these sub-systems becomes difficult. Implementing the Facade contributes to system maintainability. Wrapping the logical component in the FindMyMatchFacade class enhances the safety of system usage and makes it easier to use, as users interact through a component that encapsulates the data.

**Implementation Approach:**

We created a class named FindMyMatchFacade for the following forms: SelectedMatchForm and MyOptionalMatchesForm (meaning these forms interact with FindMyMatchFacade). The FindMyMatchFacade class calls FindMyMatchLogic and UserLogic to obtain the required data. Therefore, the FindMyMatchFacade class holds fields for FindMyMatchLogic and UserLogic to fetch the necessary data.

Sequence Diagram

תמונה שמכילה טקסט, תרשים, מקביל, תוכנית

התיאור נוצר באופן אוטומטי

Class Diagram

תמונה שמכילה שולחן

התיאור נוצר באופן אוטומטי

Template Number 3 – Singleton

**Use of the Pattern:**

The Singleton pattern is used when you want only one instance of a class, and you want to allow components to access this instance on their initiative. This is particularly useful when there's no difference in state between instances of a class, and multiple instances would lead to unnecessary overhead.

In our case, before implementing the Singleton pattern, there were multiple identical instances of the UserLogic class, causing inefficiency. Singleton allows us to create a single instance of the UserLogic class, which can be accessed directly by various components (as in our system, only one user can be connected at a time, so multiple instances are not needed). This prevents unnecessary overhead.

**Implementation Approach:**

The Singleton pattern was implemented in the UserLogic class as follows:

1. We created a private constructor to prevent anyone from creating instances, ensuring that there's no chance of creating more than one instance (thus taking control away from the UserLogic class).
2. To prevent inheritance, we sealed the class.
3. In the UserLogic class, we hold a static instance of type UserLogic called **s\_Instance**, initialized to null.
4. We implemented a property named **Instance** in the UserLogic class to prevent the creation of more than one instance of the class. It includes a Double-Check Lock for thread safety.

The **Instance** property is only implemented on the first use in the program. It creates an instance of the UserLogic class, which is stored in the static field **s\_Instance**. On subsequent uses of the **Instance** property, the first and only instance created is returned, stored in the static field **s\_Instance**.

Sequence Diagram

תמונה שמכילה טקסט, תרשים, מקביל, שרטוט טכני

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט, תרשים, מקביל, תוכנית

התיאור נוצר באופן אוטומטיClass Diagram

Template Number 4 - [Strategy]

**Use of the Template:**

In the "Find My Match" feature, we have added options to filter by gender or by a list of cities. For example, when filtering by gender, if the user chooses no filtering, they will see matches including both women and men. If they choose filtering (for example: selected "female"), they will see matches including only women. We identified the entry condition to the list of matching users as a point of potential change in the future. Through the Strategy pattern, we enable code reuse and save future duplication of code. Additionally, we make the code open for extensions (adding additional filters).

**Implementation Approach:**

In the logical part, the **FindMyMatchLogic** class holds a List<IMatchStrategy> field through composition to determine how to filter the list of matches. When a user selects to filter by gender, we execute the **FilterByGender** function (located in the **FindMyMatchFacade** class), and it creates an object of the **FilterByGenderStrategy** type and adds it to the **List<IMatchStrategy>**. When the user chooses to filter by a list of cities, we execute the **FilterByCities** function (located in the **FindMyMatchFacade** class), and it creates an object of the **FilterByCitiesStrategy** type and adds it to the **List<IMatchStrategy>**. By using these filters, we create the desired list of matches when needed.

The **IMatchStrategy** is in the role of the Strategy, with the **CouldBeMatch** method. **FilterByGenderStrategy** and **FilterByCitiesStrategy** serve as ConcreteStrategy, implementing the **CouldBeMatch** method and checking if the current user meets the user-defined filtering conditions.

In the UI part, we have added optional filtering to the "FindMyMatchForm." The user decides whether to filter the list of matches by a list of cities or by gender.

This design allows for flexibility and maintainability, enabling new filtering options to be added easily in the future.

Sequence Diagram

תמונה שמכילה טקסט, תרשים, מקביל, מספר

התיאור נוצר באופן אוטומטי

Class Diagram

תמונה שמכילה טקסט, קבלה, צילום מסך, קו

התיאור נוצר באופן אוטומטי

Template Number 5 - [Iterator]

**Use of the Template:**

We used this pattern to allow users to scan the list of friends based on the chosen filtering criteria (gender) without being aware of the data structure implemented in the logic layer. In other words, if we want to change the data structure in the future, we can do so without altering the user interface, contributing to the maintainability of the system.

**Implementation Approach:**

In the logical part, we created a class named **FriendsFilter** that implements the **IEnumerable<FriendLogic>** interface. The **FriendsFilter** class holds the user's list of friends as a field and also holds a pointer to a function that takes a **FriendLogic** and returns a boolean.

Additionally, the **FriendsFilter** class implements the **GetEnumerator** method, returning an **IEnumerator** of **FriendLogic**. The **GetEnumerator** method iterates over each friend in the user's list of friends (**r\_Friends**) and filters each friend based on the chosen criteria (in **FriendsFilter**) using **yield return** (which implements the **MoveNext**, **Reset**, **Current**, **Dispose** functions).

In the UI part, we added a new form called **FriendsFilterForm** where users can select the gender by which they want to filter the list of friends (male, female, or without specifying gender). After making their selection, the **FriendsForm** with the desired list is presented to the user.

This design separates the logic from the UI and allows for future changes in data structure without affecting the user interface.

Sequence Diagram

תמונה שמכילה טקסט, תרשים, מקביל, תוכנית

התיאור נוצר באופן אוטומטי

Class Diagram

תמונה שמכילה טקסט, תרשים, מקביל, מספר

התיאור נוצר באופן אוטומטי

**Template Number 6 - [Template Method]**

**Use of the Template:**

We used the Template Method pattern to avoid code duplication between two classes: **AlbumStatisticsByYearsLogic** and **AlbumStatisticsByMonthsLogic**. The purpose of these classes is to return statistics on the number of photos for each year (from the year Facebook was founded until the current year) and for each month, respectively. We noticed that these classes had a similar implementation of the code, except for specific places we identified as injection points. Therefore, we used the Template Method pattern to prevent the code duplication that might occur and to avoid future maintenance issues. By using the Template Method, we enable the reuse of the algorithm and save potential future code duplication. Additionally, this approach allows the code to be open to extensions, such as adding new statistics (e.g., statistics on the number of photos for years with even digits in their tens place).

**Implementation Approach:**

We created the abstract class **AlbumStatisticsLogic**, which contains the template method **GetStatisticsAboutAmountOfPhotos**. This class also includes abstract methods (such as **CreatedTimeValue**, **JumpsBetweenEachKey**, **MinNumberOfKey**, **MaxNumberOfKey**) that serve as injection points. The classes **AlbumStatisticsByYearsLogic** and **AlbumStatisticsByMonthsLogic** inherit from the **AlbumStatisticsLogic** class and implement the abstract methods (such as **CreatedTimeValue**, **JumpsBetweenEachKey**, **MinNumberOfKey**, **MaxNumberOfKey**).

In the UI part:

* **AlbumStatisticsByYearsForm** holds a field of type **AlbumStatisticsByYearsLogic** and displays the statistics on the number of photos for each year.
* **AlbumStatisticsByMonthsForm** holds a field of type **AlbumStatisticsByMonthsLogic** and displays the statistics on the number of photos for each month.

Sequence Diagram

תמונה שמכילה טקסט, תרשים, מקביל, תוכנית

התיאור נוצר באופן אוטומטי

Class Diagram

תמונה שמכילה טקסט, גופן, קבלה, מספר

התיאור נוצר באופן אוטומטי

To improve the user experience, we implemented asynchronous work in the following places:

1. We implemented this in the PostsForm: In the **onShown** method, after opening the form, we created a thread for the **fetchPosts** function. These actions can be time-consuming, and creating a thread for this function allows the user to see the form and interact with it while the data is being fetched, improving the speed of data presentation to the user.
2. We implemented this in the AlbumsForm: In the **onShown** method, after opening the form, we created threads for functions such as **fetchAlbumsAndCreateObjectsOfAlbumsThatSortedByDifferentCategories**, **fetchAlbums**, and **makeControlsEnabled**. These actions are also time-consuming, and creating threads for each of these functions enables the user to see the form and interact with it while data is being fetched, which improves the speed of data presentation to the user.
3. We implemented this in the MyOptionalMatchesForm: In the **onShown** method, after opening the form, we created threads for functions such as **fetchMyOptionalMatchesAndGetDiffrentListsOfMatches**, **fetchMyOptionalMatches**, and **makeControlsEnabled**. Like the previous cases, these actions take time, and creating threads for each of these functions allows the user to see and interact with the form while data is being fetched, thus improving the speed of data presentation.
4. We implemented this in the SelectedMatchForm: In the **onShown** method, after opening the form, we created threads for the **fetchMyOptionalMatchInfo** function, and within it, we created threads for the **passingThroughThePictures** function. These actions can also be time-consuming, and creating threads for each of them enables the user to see and interact with the form while data is being fetched, improving the speed of data presentation.

Additionally, we utilized Data Binding in several forms:

* In the **MainForm**, we used Data Binding in the **setTheUserInfo** method to display user details such as name, cover photo, birthday, and email.
* In the **FriendsForm**, Data Binding was used in the **fetchFriends** method to display the selected friend's profile picture and birthday.
* In the **LikedPagesForm**, Data Binding was used in the **fetchLikedPages** method to display the cover photo and description of the liked page that was selected.
* In the **GroupsForm**, Data Binding was applied in the **fetchGroups** method to display the cover photo and description of the selected group.