Facebook Supplementary Application

This is a project I've developed as part of the Design Patterns course.
This application, developed using C# WinForms, extends Facebook's functionality by implementing two additional features. It demonstrates proficiency in Facebook API integration and practices UML-based design patterns.

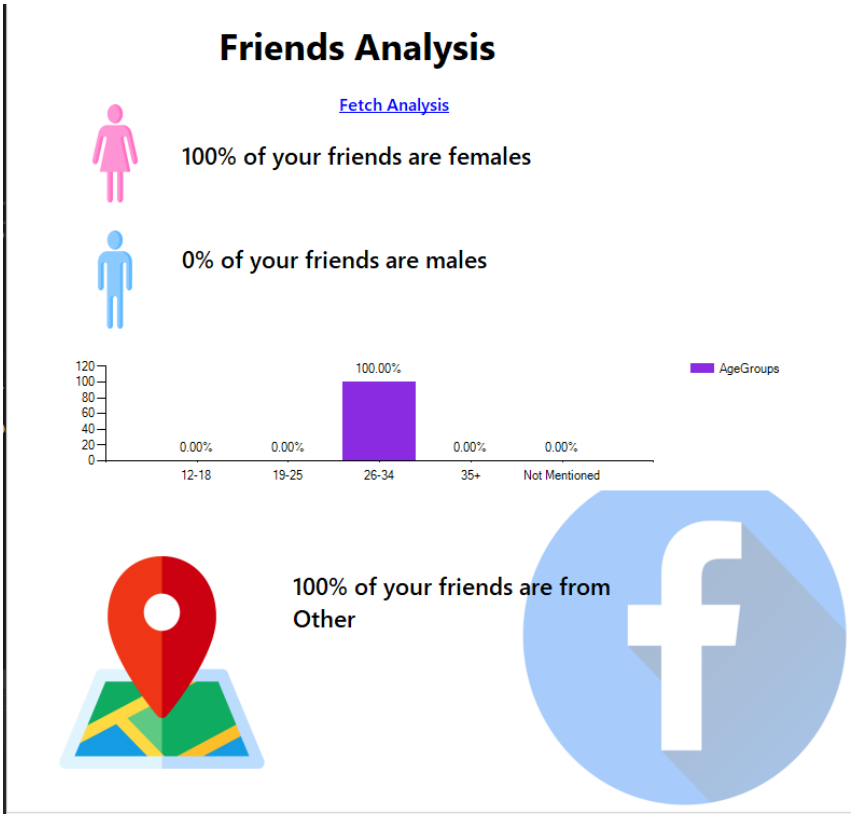
The Two Additional Features:

Friends Analysis:

The Friends Analysis feature allows users to gain insights into their Facebook friends' demographics.
It provides the following information:

- Gender distribution among friends.
- The top three most common hometowns among friends, along with the percentage of friends residing in each.
- Age group distribution among friends: 12-18, 19-25, 26-34, 35+, and those with unlisted ages.

Friends without hometown information are categorized as "Other."
Please note that due to API limitations, hometown information is always displayed as "Other".

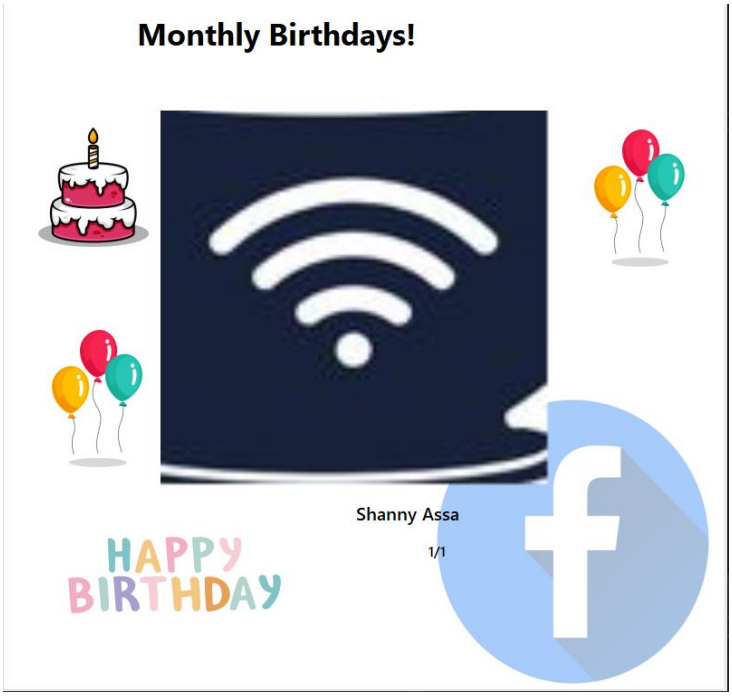


You can explore the logic behind this feature in the following classes within the FacebookAppLogic directory.

Monthly Birthdays:

The Monthly Birthdays feature allows users to view and celebrate their Facebook friends' birthdays for the current month. It presents a list of friends' names and profile pictures, allowing you to navigate through them and view each friend's birthday celebration.

You can explore the logic for this feature in the MonthlyBirthdays class within the FacebookAppLogic directory.



Design Patterns Implemented in the App:

1. Factory Method:

I've implemented the Factory Method pattern for creating different forms presented to the user. These forms inherit from the Form class and form a polymorphic family. This design choice allows easy addition of new forms in the future without altering user code.

2. Façade:

The Façade pattern is used in the Friends Analysis feature. It encapsulates complex logic, providing users with a simple interface to access relevant information while hiding the underlying complexities.

3. Builder:

The Builder pattern is employed in the Posts feature to enable users to filter and display posts based on various criteria. It simplifies the construction of complex Post objects according to user-specified criteria.

4. Iterator:

The Iterator pattern is used in the MonthlyBirthdays class to provide users with a way to iterate through friends celebrating their birthdays in the current month without exposing the underlying data structure.

5. Strategy:

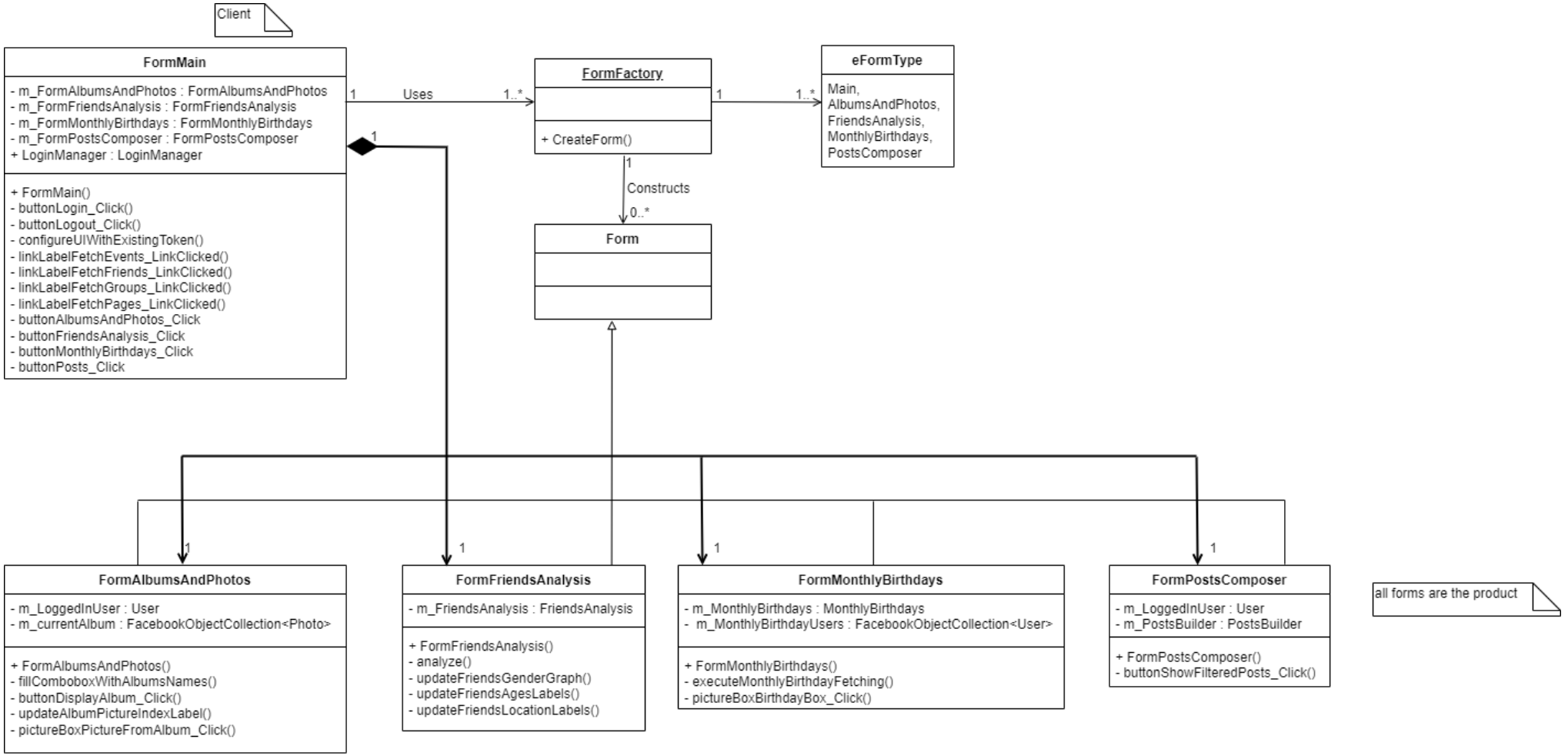
The Strategy pattern is applied to the Online Status Filter feature. Users can filter their friends based on online status. The Strategy pattern allows us to switch filtering algorithms at runtime without modifying the client code.

6. Observer:

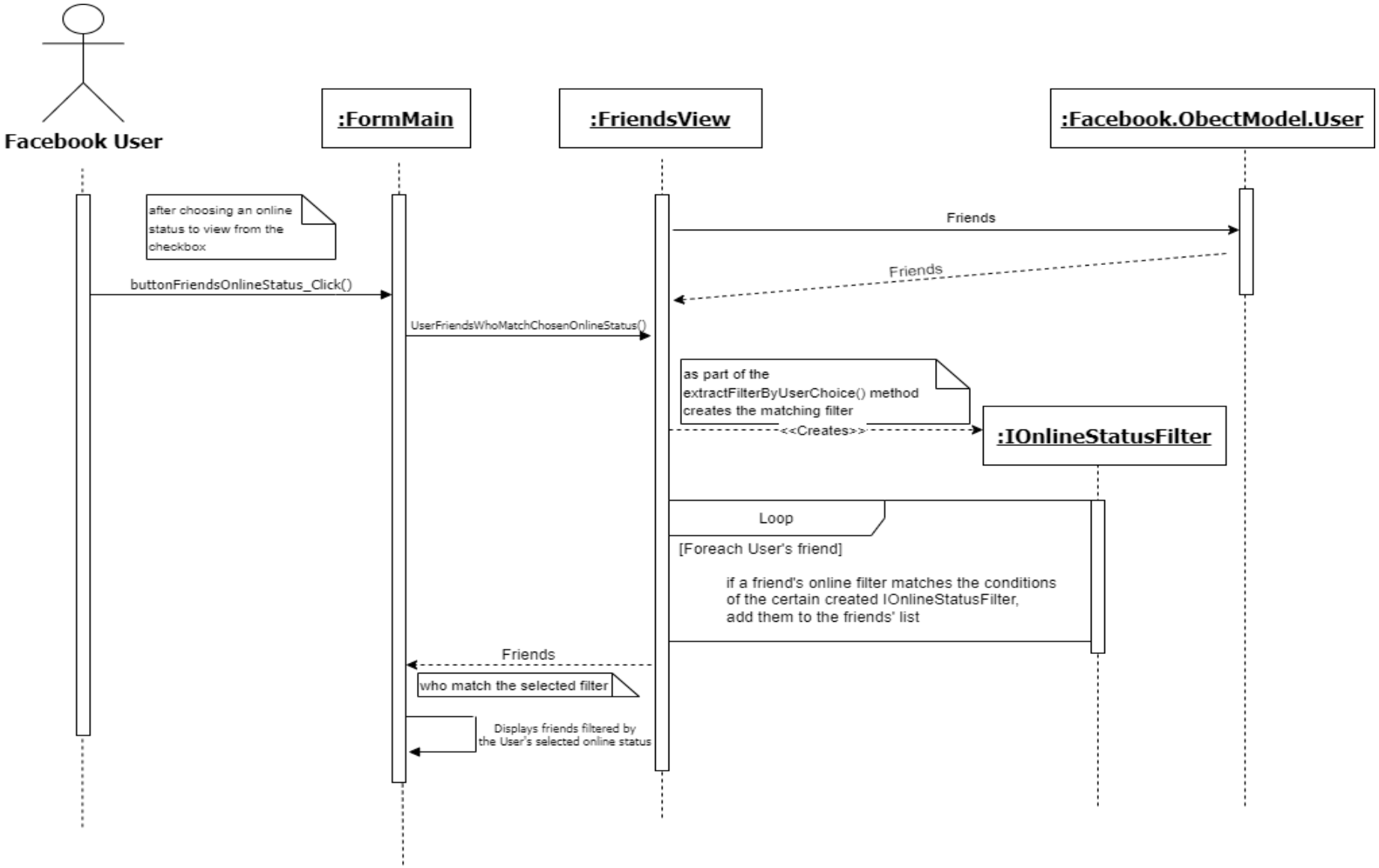
I've employed the Observer pattern to implement a light/dark mode switch. When the user toggles this mode, all registered forms and components adapt to the selected mode. The pattern allows future forms to easily subscribe to mode change events.

The following charts are examples of Class Diagrams and UMLs I have created for the application, illustrating the implementation of various design patterns:

Factory Method Class Diagram:



Strategy Sequence Diagram:



Strategy Class Diagram: