SE -2430
Aisultan&Abylaikhan
Introduction

This project introduces a Photo Gallery application developed in Go, focusing on the implementation and demonstration of six software design patterns: Factory, Strategy, Decorator, Facade, Observer, and Adapter. The application allows users to manage and display photos interactively while maintaining a modular and extensible architecture.

Architecture Overview

The system is architected using Go’s struct and interface features. It demonstrates flexible software design by integrating patterns that solve real-world problems such as data creation, format adaptation, extendable filtering, notification management, and interface simplification. Legacy data formats are supported through the Adapter pattern, while filters and notifications exhibit dynamic behaviors.

The main components and their relationships are shown in the UML diagram, illustrating how each pattern interacts within the project.

Implemented Design Patterns

- Factory: Handles the creation of photo objects with essential attributes like title and URL.
- Strategy: Encapsulates filtering logic, allowing the application of different filters (Black & White, Sepia) to gallery photos.
- Decorator: Enables runtime enhancement of photo display, such as adding a watermark.
- Facade: Provides a simplified, unified interface for all gallery operations, including adding and showing photos.
- Observer: Manages notification logic; for example, sending an email when a new photo is uploaded.
- Adapter: Integrates legacy or external photo formats into the gallery, ensuring compatibility and extensibility.

Team Member Contributions

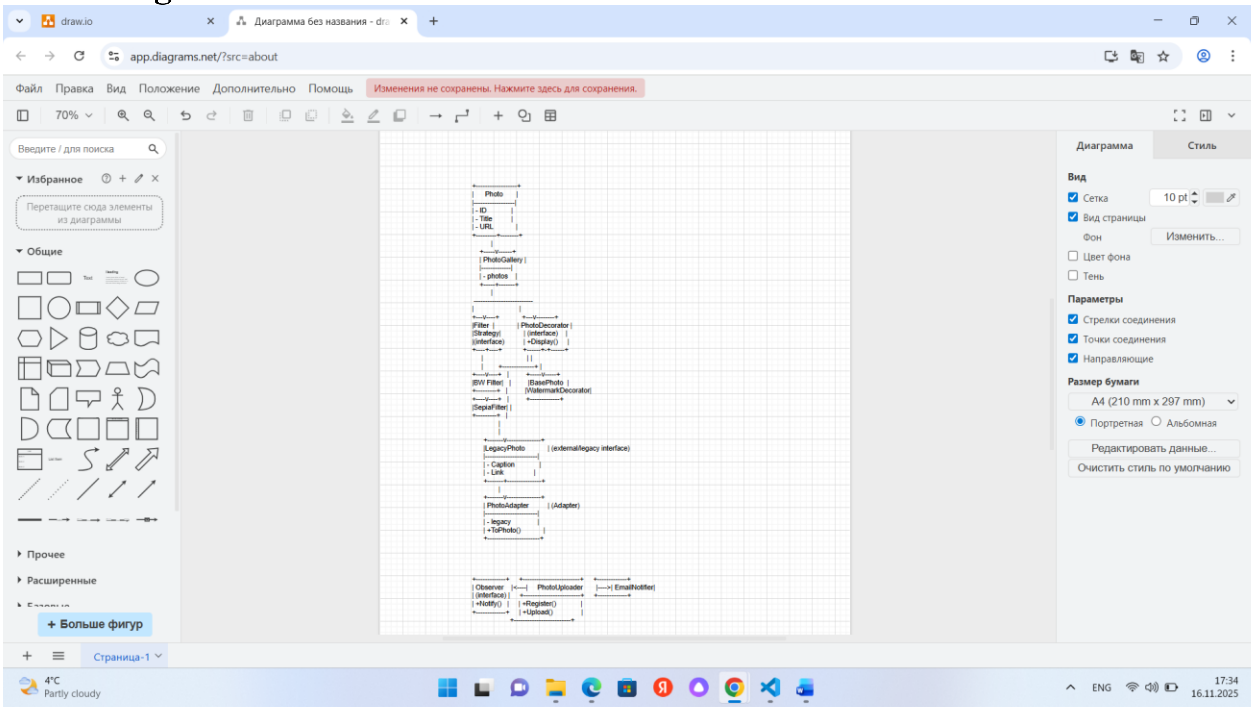
Team Member	**Patterns**
Aisultan	Factory, Decorator, Adapter
Abylaikhan	Strategy, Facade, Observer

- Aisultan: Responsible for photo creation (Factory), decoration (Decorator), and legacy photo integration (Adapter).
- Abylaikhan: Developed filtering (Strategy), gallery management (Facade), and notification system (Observer).

Application Features

- Interactive menu for gallery management in the terminal.
- Addition of both “native” gallery photos and legacy/external photos via the Adapter.
- Support for filters and decorations applied at runtime.
- Instant notifications on photo upload.
- Unified management of gallery content.

UML Diagram



Challenges and Solutions

During development, special consideration was given to integrating patterns in a coordinated manner as required by the project brief. Compatibility issues with legacy data were resolved using the Adapter pattern, and team collaboration was maintained through code modularity and clear documentation.

Conclusion

All six design patterns are fully implemented and operational within a cohesive architecture. The project demonstrates a practical and extensible solution using Go, providing both educational value and foundations for further development.