Phase 2: Implementation of Design Patterns

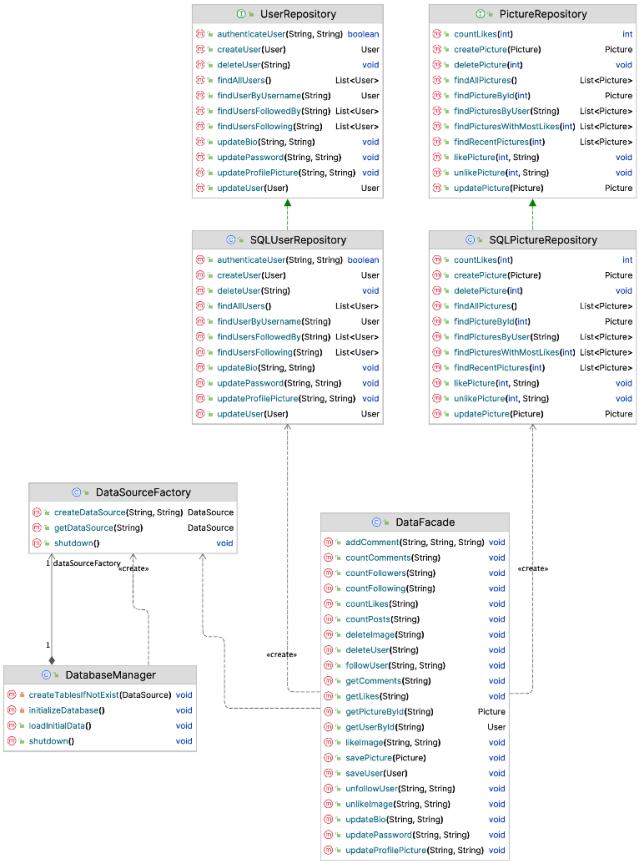
Group Number: 03, Student Names: David Wicker, Antoni Rodawski, Nahdjay Lin.

Thursday, March 28, 2024

Table of contents

1. Design Patterns (8%)

Throughout the project development, three design patterns were utilized, all revolving around the implementation of an SQLite database. These patterns are interconnected, as depicted in the accompanying Class Diagram.



1.1 Repository Design Pattern

We adopted a Repository pattern for each model in our application, including **User** and **Picture**. This pattern involves implementing both a Repository **Interface** and a corresponding Repository **Implementation** for each model. The Repository interfaces define the methods that must be implemented by different implementations. We opted for this pattern because it offers abstraction at the database layer and supports multiple implementations of the Repository. This flexibility enables us to employ various database implementations while preserving a **consistent structure** across the application.

1.2 Facade Design Pattern.

The Facade pattern serves to simplify the interface of a complex subsystem or a collection of interfaces, streamlining their usage for clients. Rather than encapsulating the subsystem, it provides a higher-level interface, enhancing the subsystem's usability. In our implementation, only the **DataFacade** class is exposed to external packages and the rest of the application. The DataFacade then interfaces with the **Repositories** based on the type of action that is being performed.

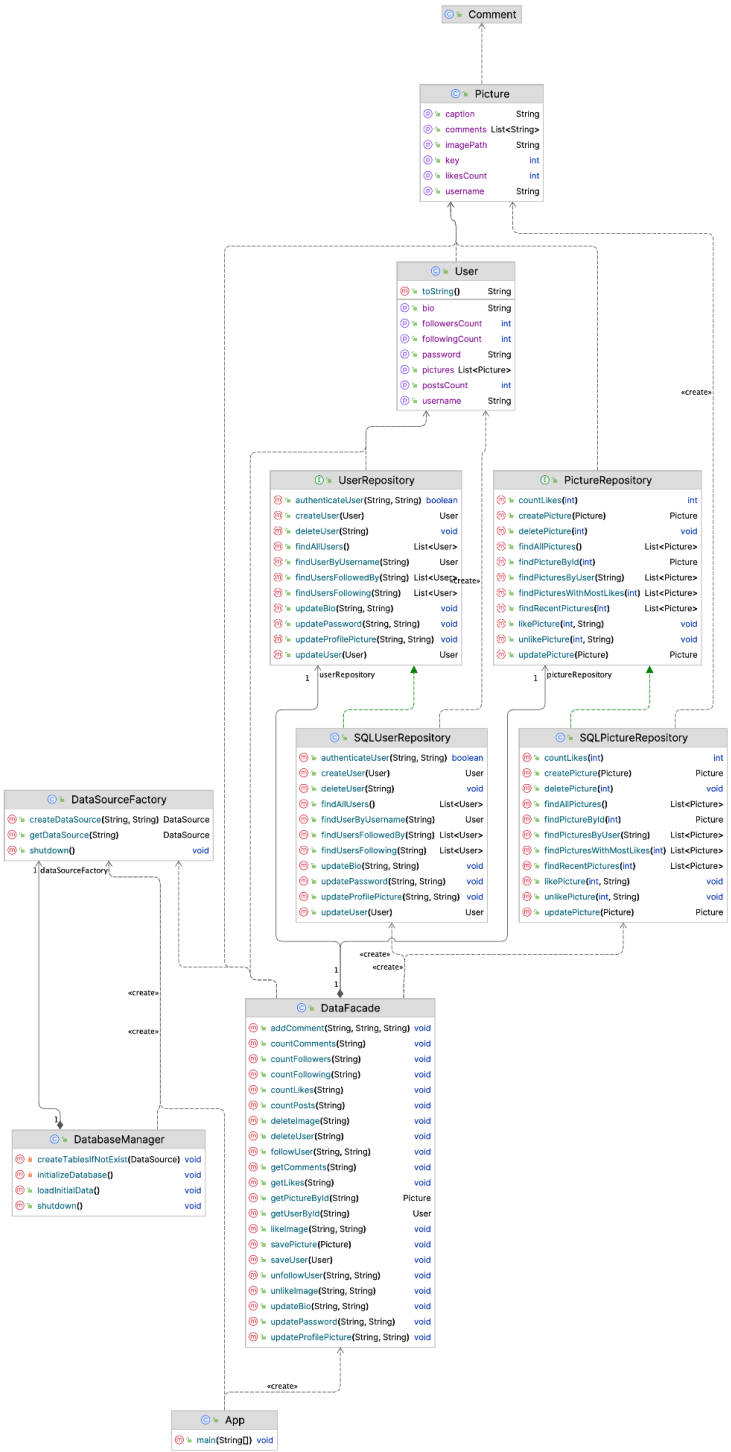
1.2 Factory Design Pattern

The Factory design pattern, a widely used programming design pattern, involves creating objects without specifying the exact class of object that will be created. In our application, we utilize the Factory pattern through the implementation of a **DataSourceFactory**. This factory is responsible for creating **DataSource** instances as required, enabling the establishment of multiple DataSources. These DataSources facilitate connections to different databases, such as test and production databases, as needed. By managing the connection with the Database, the DataSources ensure error-free operation.

2. Final UML Diagram

The second phase of development introduced significant changes, particularly in the backend package and the adoption of an SQLite database to manage the application's data, replacing the earlier text-file system. The implementation of SQL queries also replaces the functionalities previously handled by the "ImageLikesManager" and "UserRelationshipManager" classes.

This diagram emphasizes the implemented modifications and illustrates the dependencies among the involved classes, providing a clear overview of the system's architecture.



3. Application Functionality and User Experience

3.1 Application Functionality (6% judged both by this report and your demo)

The final application focuses on the implementation of an SQLite Database to replace the previous text-file system. The repositories, organized within the database package, replace the 'logic folder' containing classes such as ImagesLikesManager and UserRelationshipManager. These repositories facilitate the storage and retrieval of data related to pictures and users.

To enhance scalability, the application now includes the DataSourceFactory class to handle multiple connections to multiple databases. The DatabaseManager class loads initial data from text files onto the SQLite database.

Additionally, the implementation of Like and Follow features through the SQLite database replaces the old implementations.. This design pattern enhances maintainability and extensibility while abstracting the complexity of database operations.