Secure Backup Software System
Analysis and Design Document

Student: Ciucescu Vlad Andrei
Group: 30432

Revision History

Date	Version	Description	Author
04/Apr/18	1.0	First version of the project analysis and design document	Ciucescu Vlad
19/Apr/18	1.1	Added domain model, updated conceptual architecture, package diagram, deployment diagram, component diagram	Ciucescu Vlad
21/Apr/18	1.2	Added design model, data model and testing methods	Ciucescu Vlad
16/May/18	1.3	Iteration 2 of elaboration Updated domain model, architecture, package diagram, design model and data model.	Ciucescu Vlad

Table of Contents

I. Project Specification

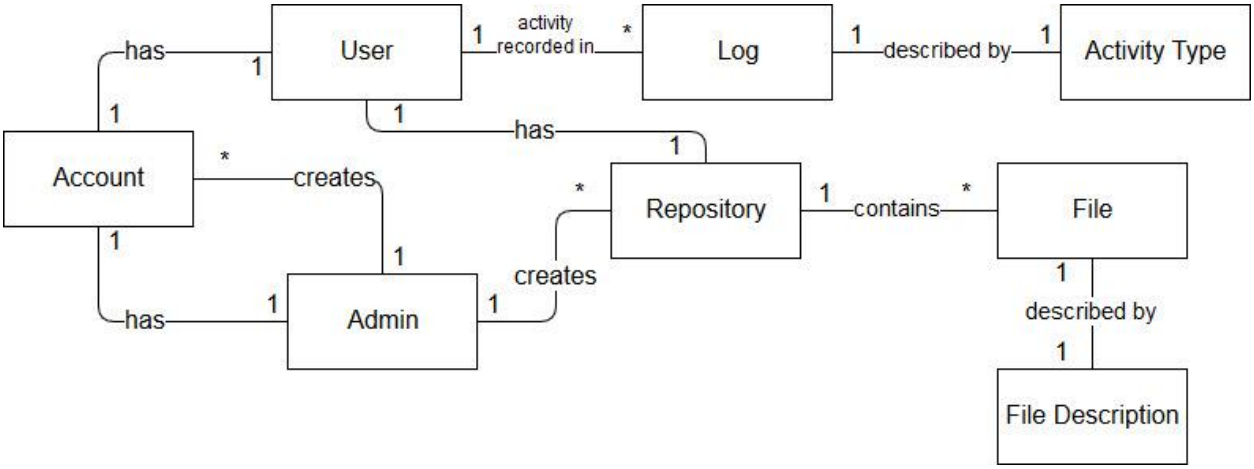
A software backup system should be implemented, which allows users to store files, documents, images and videos through a windows desktop application in a secure manner. A separate folder should be created for each user for them to store their files in. This folder should be only accessible to authorized users. All the details of the users, user activity and file details will be stored in a RDBMS. The actual files will be stored in a NoSql DMBS, such as MongoDB.

There will exist two types of users:

- Administrator – has the responsibility of managing user accounts (creating new accounts, deleting accounts at the request of users, blocking a user etc.)
- Regular User – can use the software to store and retrieve files. Can access their account after an administrator has created it and can request an administrator to close their account.

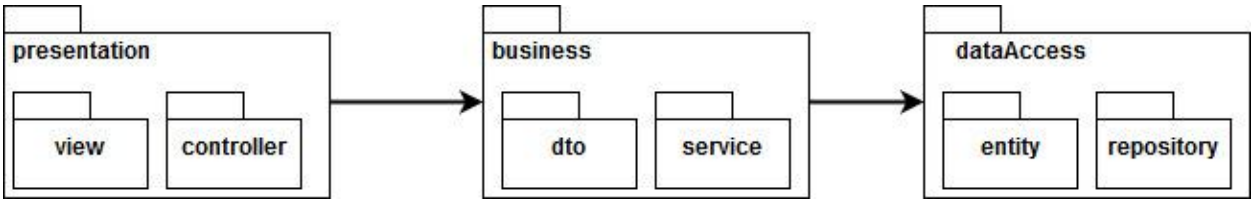
II. Elaboration – Iteration 1.1

1. Domain Model



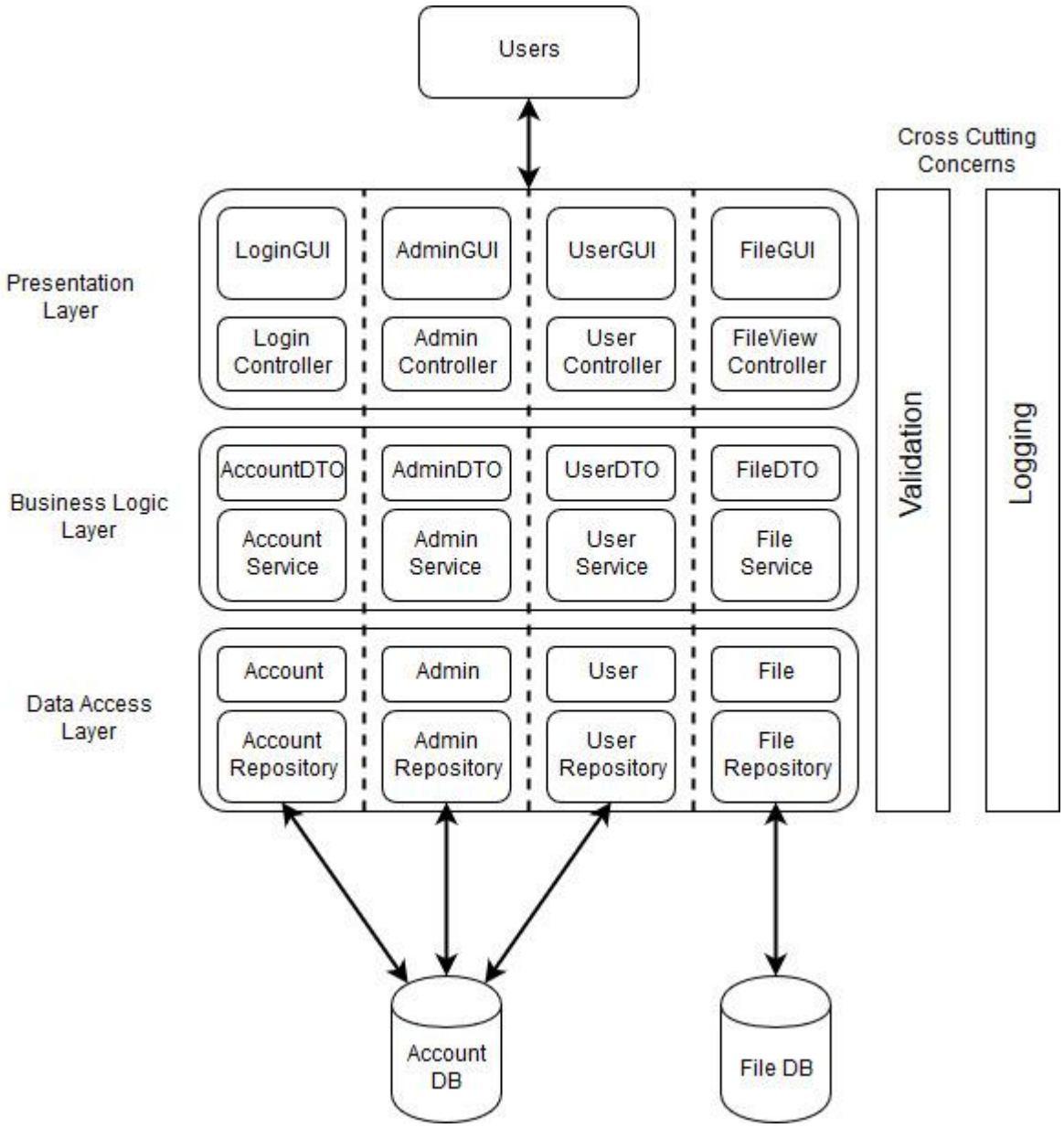
2. Architectural Design

2.1 Package Design



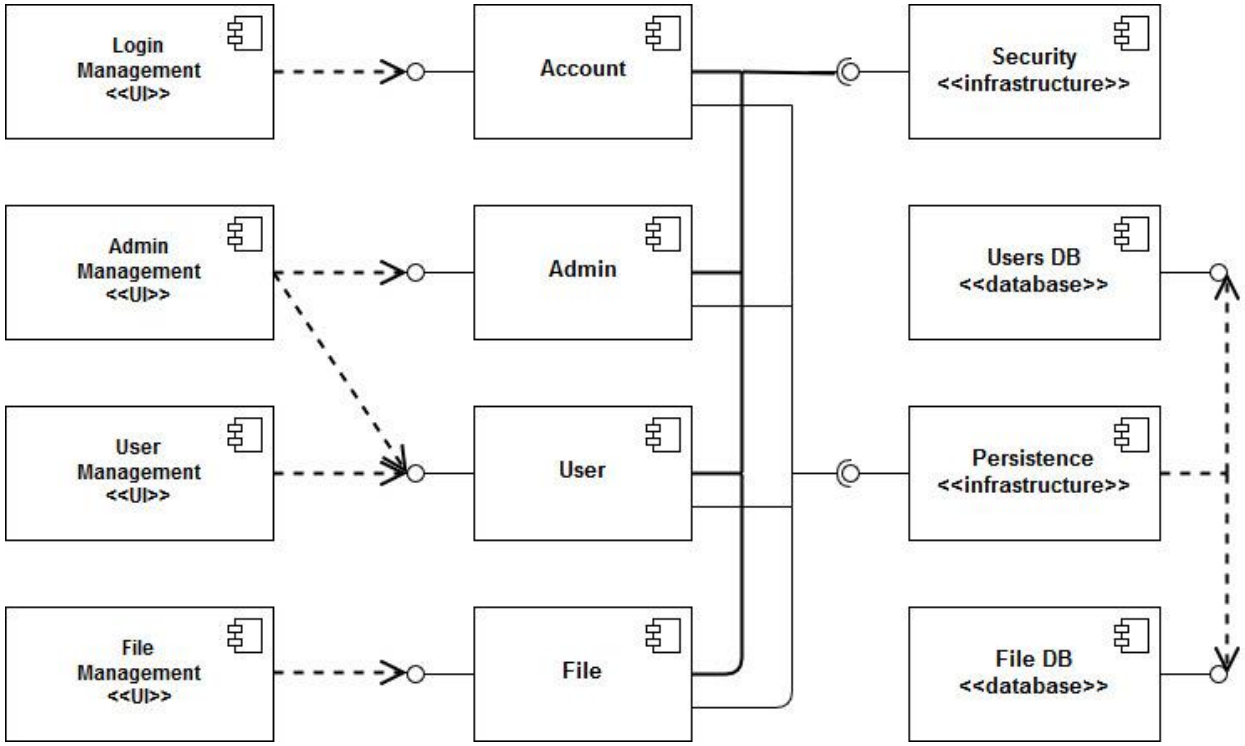
2.2 Conceptual Architecture

The proposed architecture of the application is the layers architecture. This aids in separating the responsibilities, because each layer is designed to perform a specific role within the application. As such, each layer will be cohesive and easily replaceable, should the need arise. This system will be composed of three layers: data access layer, business logic layer and presentation layer.

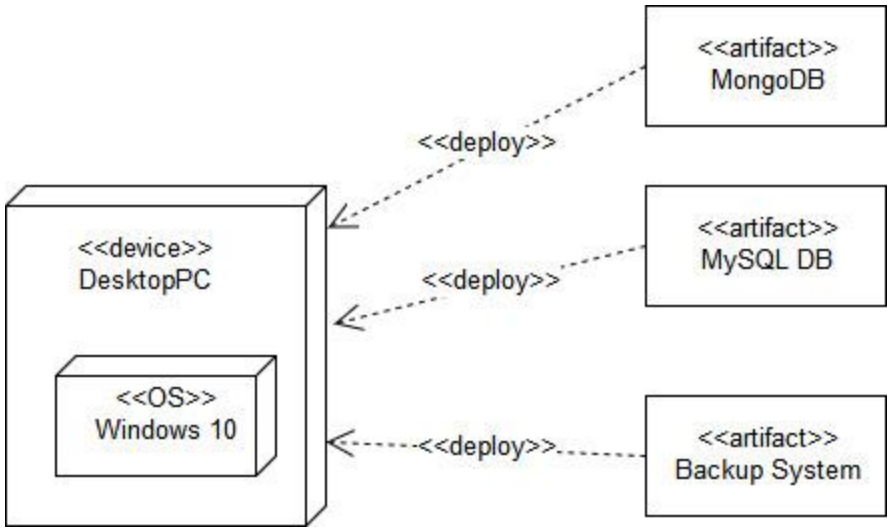


2.3 Component and Deployment Diagrams

Component Diagram



Deployment Diagram

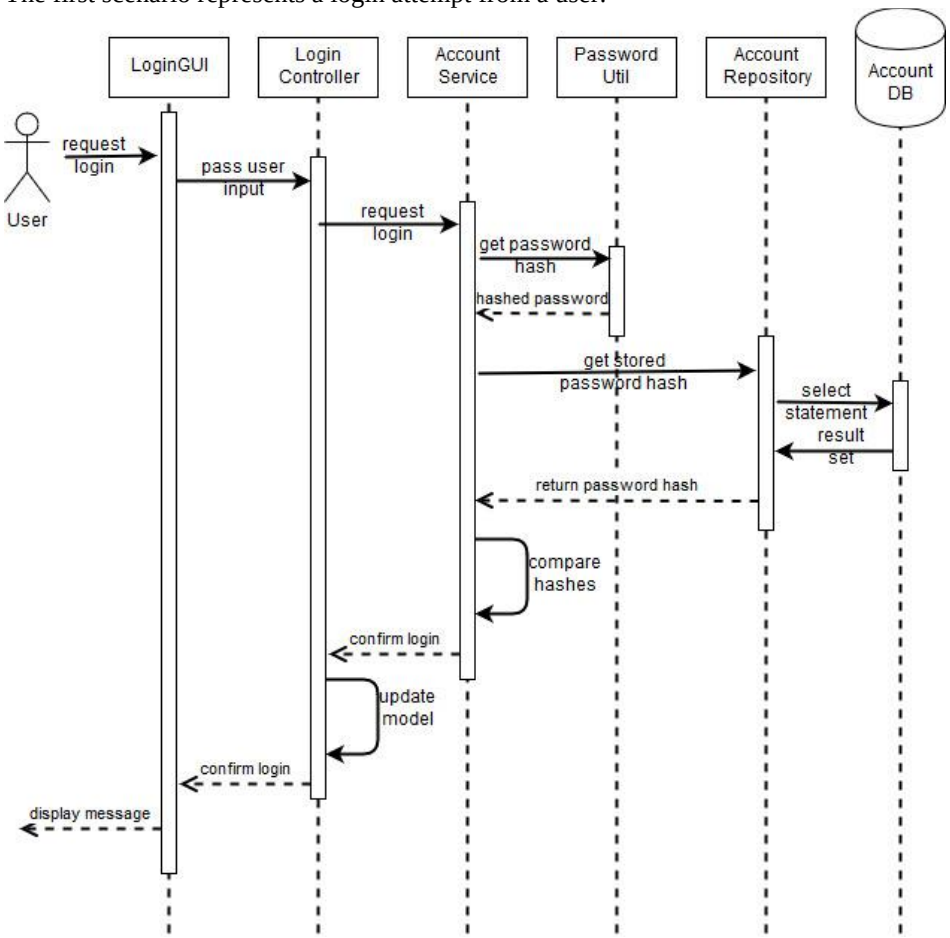


III. Elaboration – Iteration 1.2

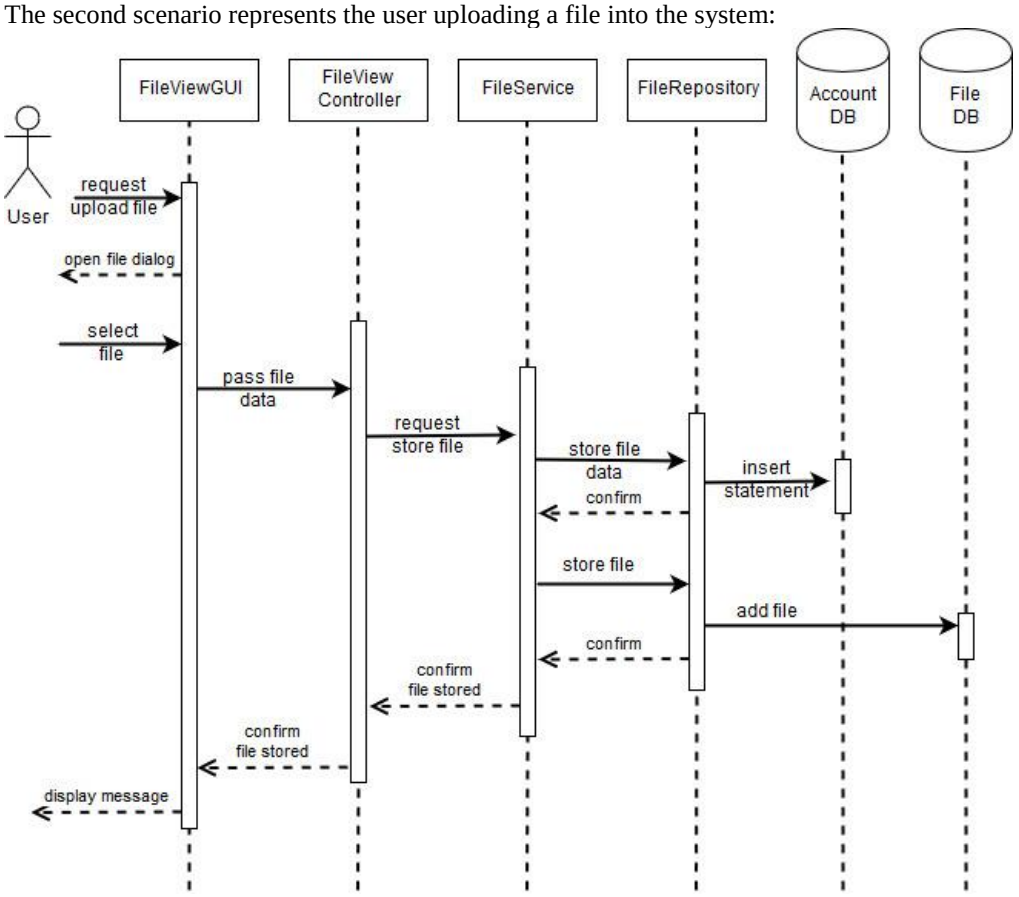
1. Design Model

1.1 Dynamic Behavior

The first scenario represents a login attempt from a user:



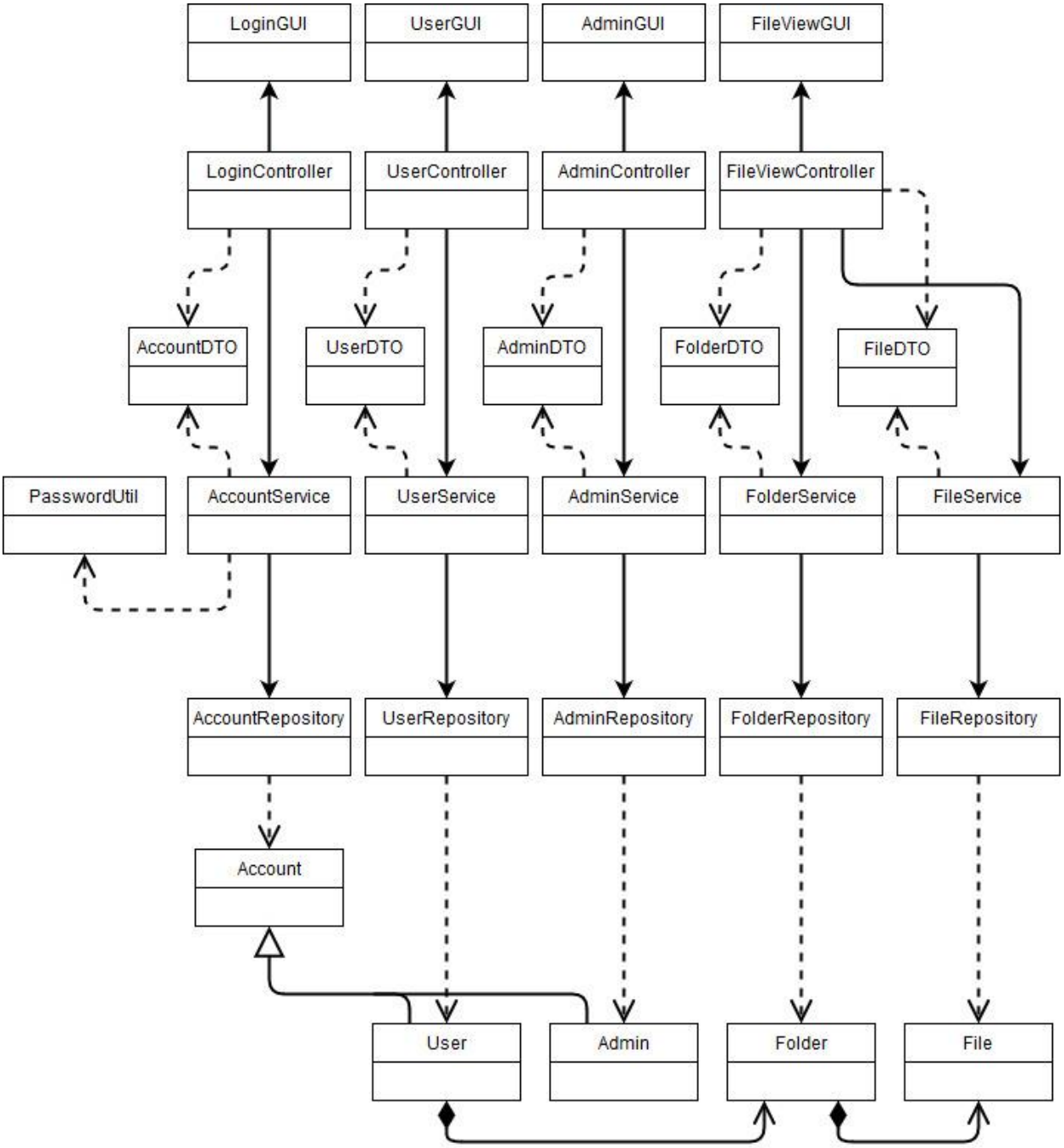
The second scenario represents the user uploading a file into the system:



1.2 Class Design

The following patterns will be applied when implementing the application:

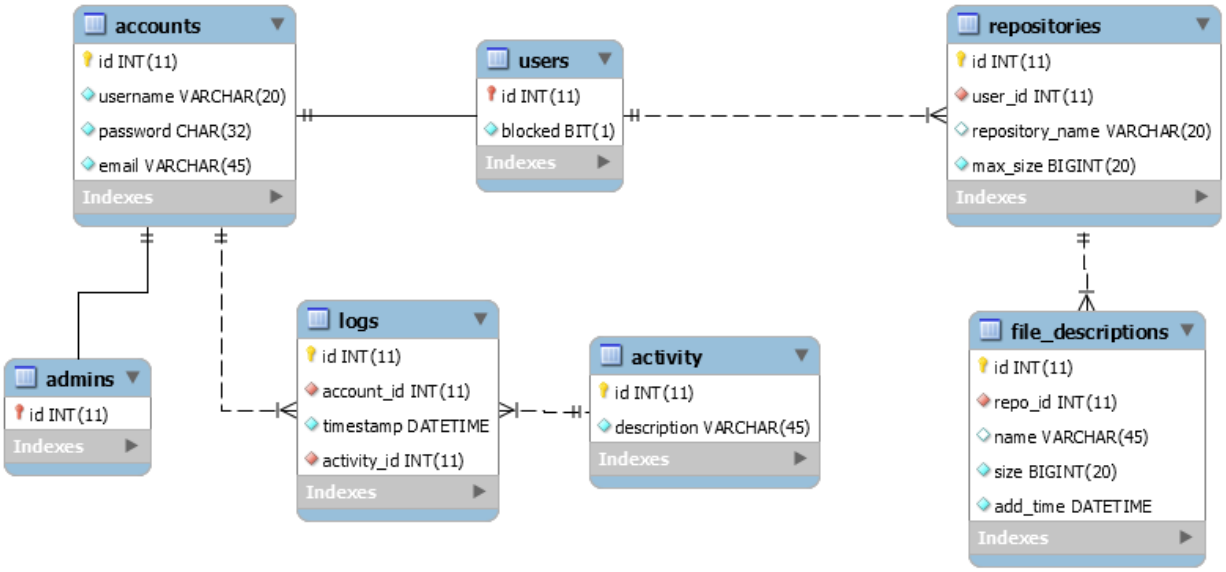
- the Observer pattern in the presentation layer, the GUI classes being the Observable and notifying the controllers of changes
- the Factory pattern is used to create connections to MongoDB
- the Adapter pattern will be used to convert MongoDB documents to Entities and vice-versa
- the Proxy pattern will be used to represent the list of files to the user: the actual files will not be loaded until the user needs them. Instead, a basic description of the file will be provided.



2. Data Model

The account data, activity logs and file descriptions associated to each user will be stored in a relational DBMS, such as MySQL. The actual files will however be kept in a NoSQL DBMS, such as MongoDB.

The data model for the relational database is the following:



3. Unit Testing

The following methods will be used to test the application:

- Unit testing, which involves testing small individual units of code, such as methods. To this end, two testing frameworks will be used: JUnit along with Mockito. Together, these enable the creation of test double objects in automated unit tests.
- Integration testing, which tests multiple software modules working together. It occurs after unit testing.

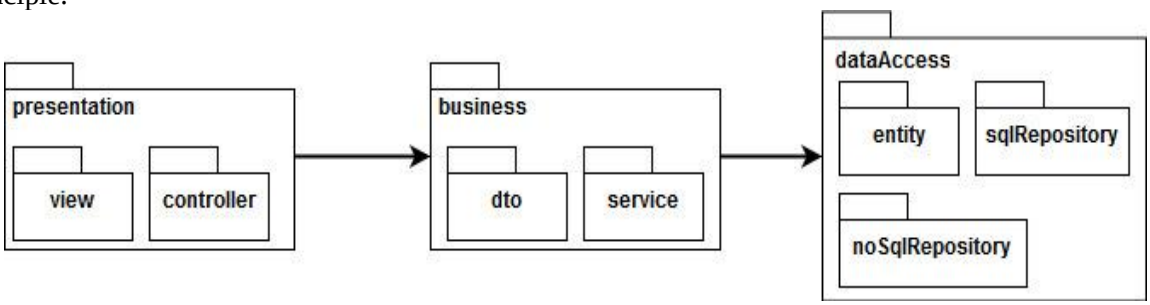
IV. Elaboration – Iteration 2

1. Architectural Design Refinement

1.1 Package Design

The package design has been updated and restructured. The repositories in the data access layer have been split in two sub-packages: sql repositories and nosql repositories.

The packages respect the principles of package design, as there exist no circular dependencies between them and they are separated according to the single responsibility principle.

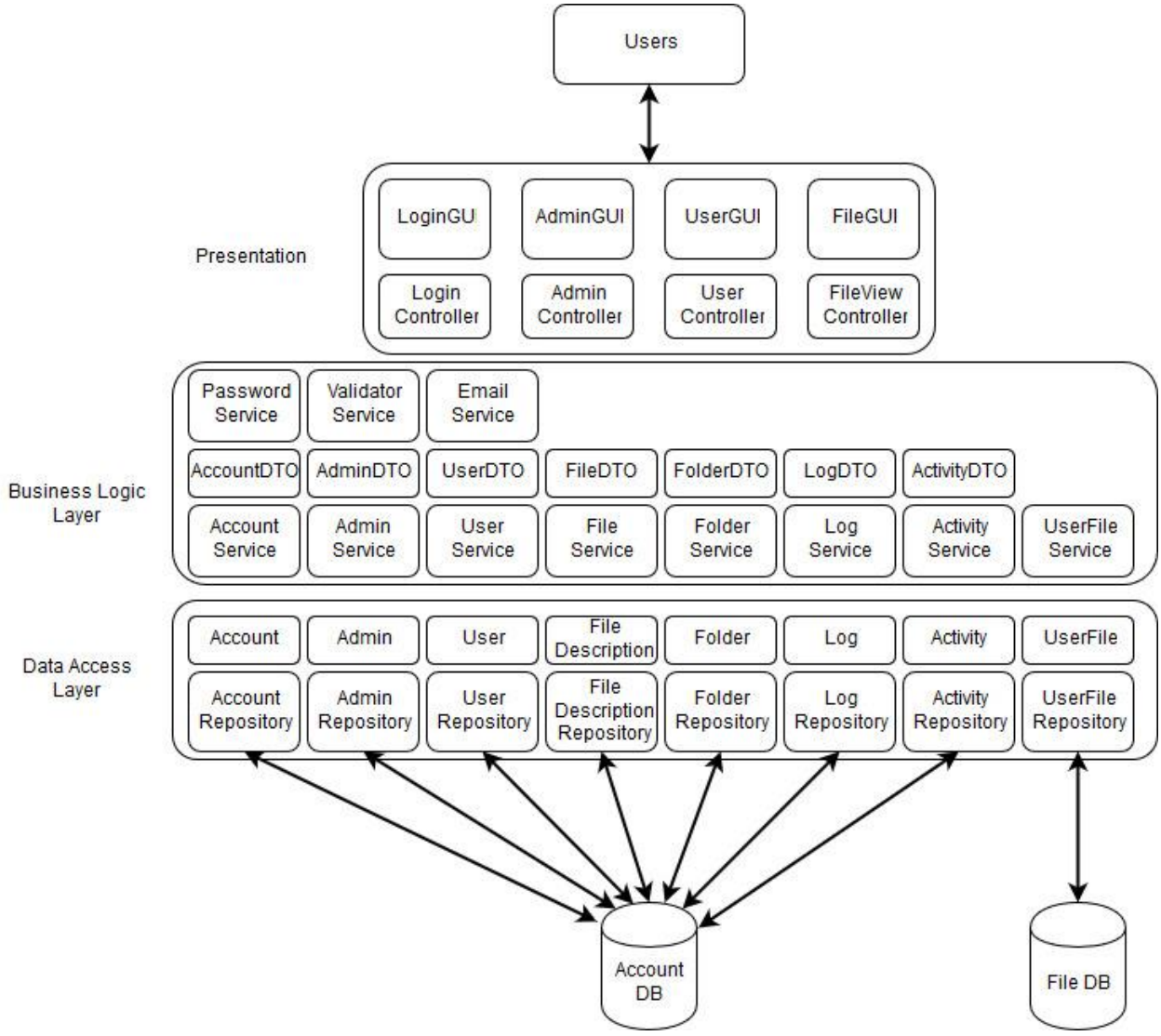


1.2 Conceptual Architecture

The architecture is still based on the layers + MVC pattern, but it has been updated to reflect the additional entities, repositories and services added.

The cross-cutting concerns have been removed, as validation and logging will only be performed in the business layer.

Each entity, DTO, repository and service has been modified to extends or implement a superclass/interface, in order to facilitate writing more generic classes with them.



2. Design Model Refinement

[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]

3. System Testing

Unit testing has been performed both for the data access layers (for the repositories) and for the business layer (for the services). This involves testing individual methods of the respective classes. It has been done with support from the JUnit and Mockito frameworks.

Integration testing has been performed for the data access layer, using JUnit. Both SQL Repositories, as well as NoSQL Repositories have been tested in this way. The tests have been performed in such a way as to leave the database in the same state as before the tests.

4. Future improvements

A series of improvements can be implemented in the application:

- Improve the graphical user interface, to make it prettier and more intuitive.
- Add the possibility for users to have multiple repositories, with an increased capacity.
- Maximum file size should be increased (through add-ons to MongoDB)
- Some sort of preview of selected files should be allowed.
- Improve the password encryption algorithm and add salt.
- Also encrypt usernames.
- Require One Time Password authentication through SMS.

Bibliography

https://msdn.microsoft.com/en-us/library/ee658109.aspx
http://softwaretestingfundamentals.com/unit-testing/
http://www.vogella.com/tutorials/Mockito/article.html
http://www.agilemodeling.com/artifacts/
http://www.oodesign.com/adapter-pattern.html
https://en.wikipedia.org/wiki/Observer_pattern
http://hibernate.org/orm/documentation/5.3/
http://hibernate.org/validator/documentation/
http://mongodb.github.io/mongo-java-driver/3.7/