<Social Media Application>

Analysis and Design Document

Student: Socaci Radu Andrei

**Group: 30431**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 18/03/2020 | 1.0 | Project Deliverable 1. Project Specification, Elaboration Iteration I, Construction and Transition sections added. | Socaci Radu Andrei |
| 02/05/2020 | 1.0 | Project Deliverable 3. Data Model and Design Model | Socaci Radu Andrei |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

# Project Specification

*The project is a social media application, with fully featured login/registration, notifications, likes, comments, image uploads, etc. The system will be developed using React on the front-end and Node Express for the REST API. Certain user roles can be created to restrict the access to certain activities, such as deleting a post that does not belong to the currently logged in user, etc. After the user registers a new account, he will be prompted to upload a profile picture and update some bio information. Therefore, he/she will be able to see other posts, like, comment, receive notifications, etc. Firebase will be used as the data store. Basic CRUD operations will be performed at every level for each entity in the system.*

# Elaboration – Iteration 1.1

# Domain Model

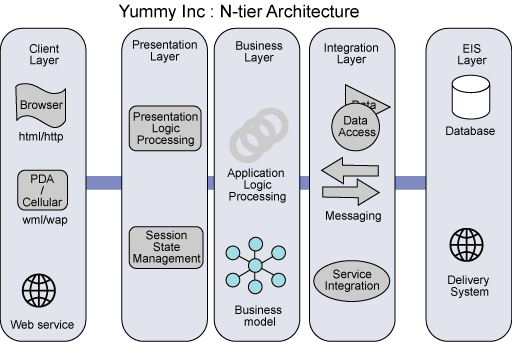
*The domain model of this application can be seen as a visual representation of the conceptual classes in the domain of interest of this application. There will be a class for the users, posts, comments, status of the posts, messages, etc.*

# Architectural Design

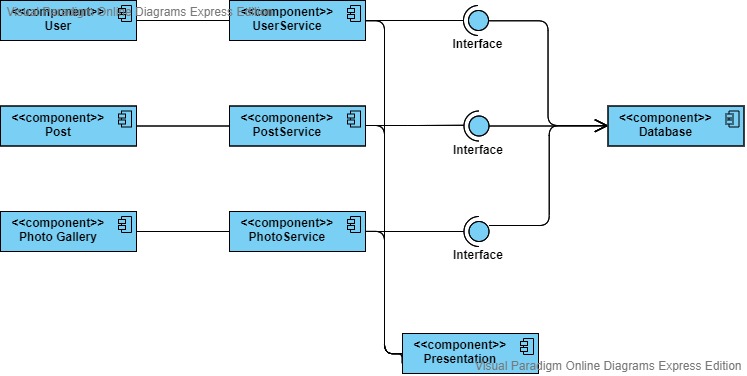
## Conceptual Architecture

*The project will use the client-server architectural pattern, with the server handling all the operations on data and providing the data to the client by means of a REST API. The server will be based on a 3-tier architecture (Data Access, Business Logic, Presentation) and the client (front-end) will be based on the Model View Controller (MVC) pattern.*

## Package Design



## Component and Deployment Diagrams

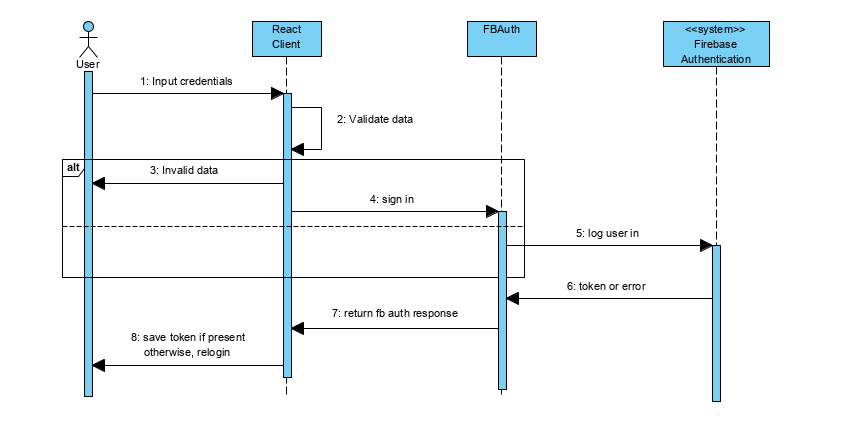


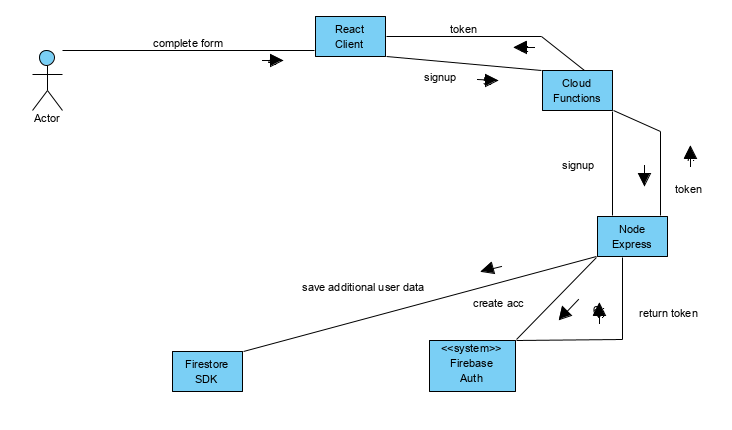
# 

# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

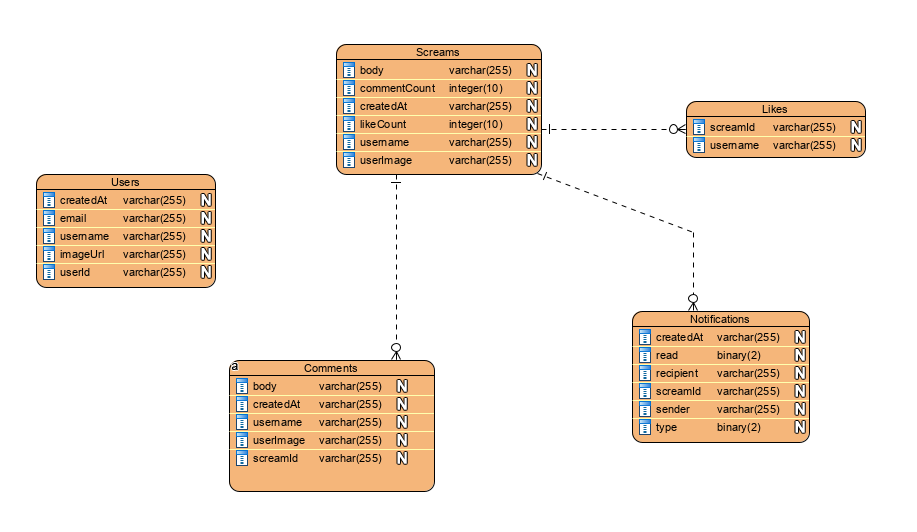




## Class Design

*[Create the UML class diagram; apply GoF patterns and motivate your choice]*

# Data Model



Some data is duplicated across multiple collections to keep the number of requests small. Otherwise, the free firebase plan may not work anymore. The user image itself is stored in a storage bucked in firebase. The database collections keep the path to the image. A logical relation between tables is established using the screamId field (to avoid retrieving a huge data frame whenever the user makes a request).

# Unit Testing

*[Present the used testing methods and the associated test case scenarios.]*

# Elaboration – Iteration 2

# Architectural Design Refinement

*[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]*

# Design Model Refinement

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

# Construction and Transition

# System Testing

*The integration testing units are built taking into consideration the functional requirements and the use case diagram. At unit lever, unit testing will be employed to verify the correctness of certain pieces of code and ensure the application can later be refactored without the risk of breaking.*

*Test cases:*

* *Login – a user should be able to login successfully using his username and password. If he introduces incorrect credentials, he will not be granted access.*
* *Register – a user should be able to create an account. If the validations succeed, the database should be updated with the new account.*
* *Logout – if already logged in, a user should be able to log out at any time.*
* *Create post – if the user is logged in, he should be able to create a new post, which should result in the database being updated.*
* *Delete post – in much the same fashion, the user should also be able to delete a post. There are 2 scenarios: either the user deletes his own post, or an admin permission is required to delete any post.*
* *Add comment – if the user is logged in, he should be able to post comments, unless comments are blocked by the author.*
* *Notification system – the notification system should also be verified. When a post of a certain user gets updated, he should be notified at login.*

# Future improvements

*The application can be extended using a real time chat (using a multithreaded context). Moreover, the notification system could be improved to show notification in real time, without the need to refresh the page (websockets).*

*Support can be added for videos, groups, etc.*

# Bibliography