Álvaro Suárez del Cueto

Analysis and Design Document

Student:Alvaro Suarez del Cueto

Group:Erasmus

Revision History

Date	Version	Description	Author
04/04/2018	1.0	Added specifications	Alvaro
25/04/2018	1.1	Iteration 1.2	Álvaro

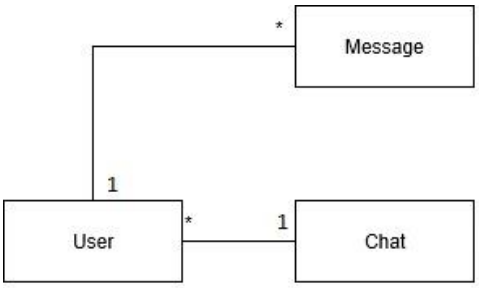
Table of Contents

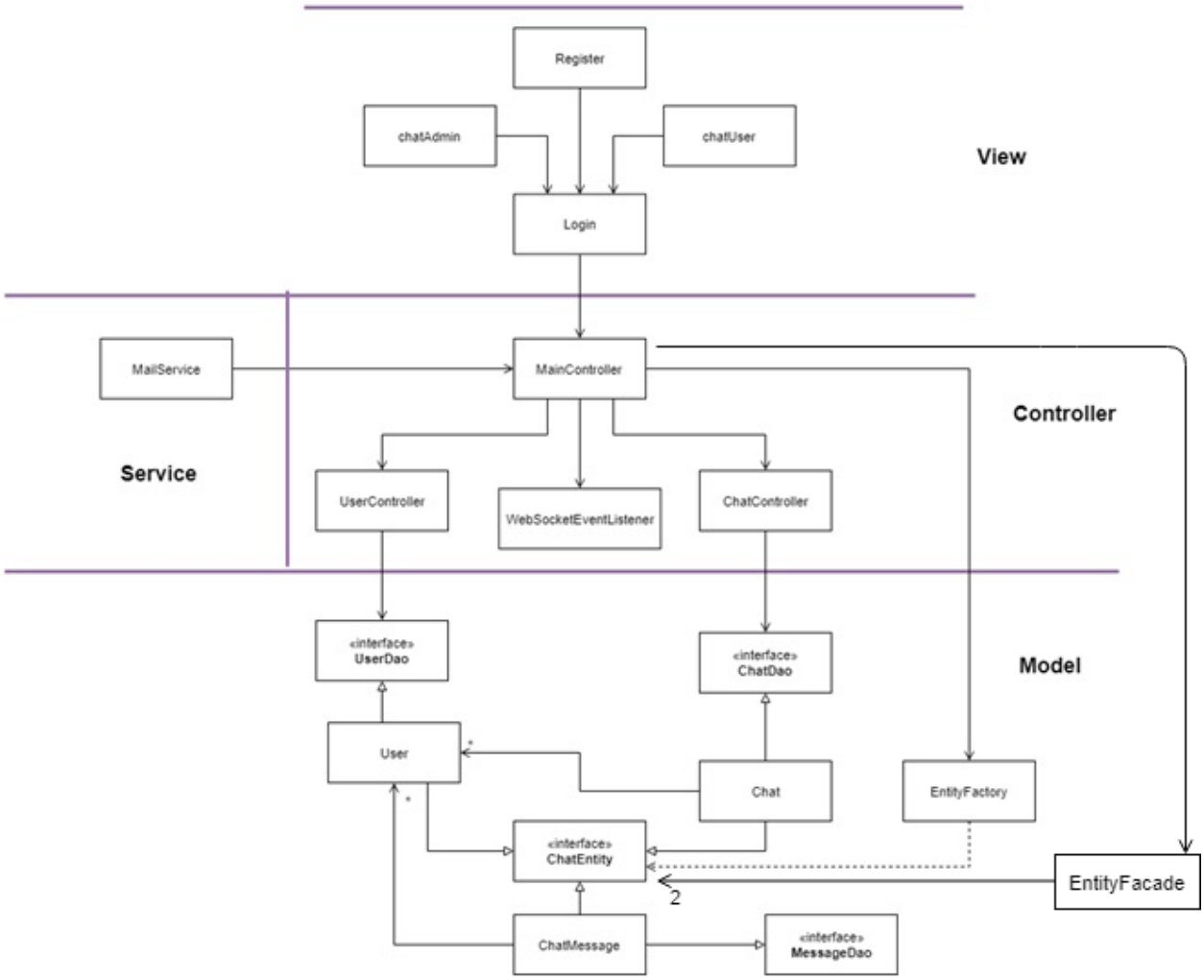
I. Project Specification

Application for tracking suspicious behavior in online chats.

II. Elaboration – Iteration 1.1

1. Domain Model

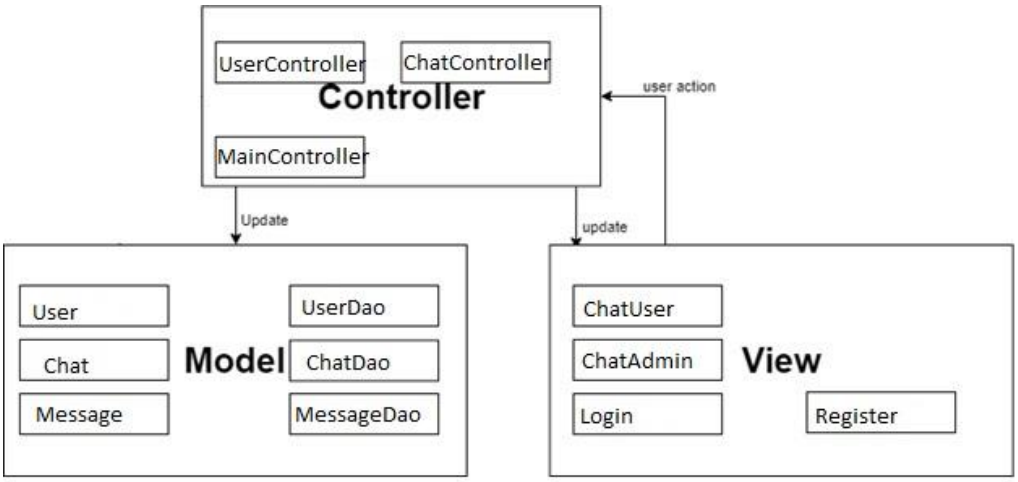




2. Architectural Design

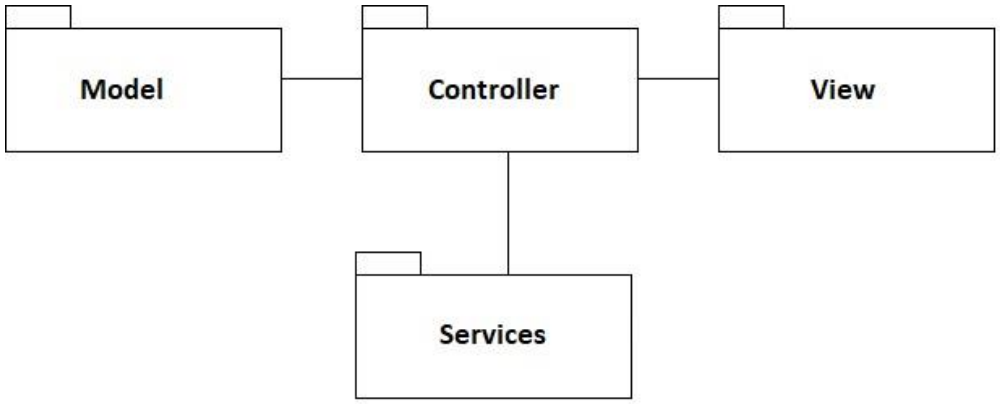
2.1 Conceptual Architecture

The system will follow the Layered + Model View Controller architecture:

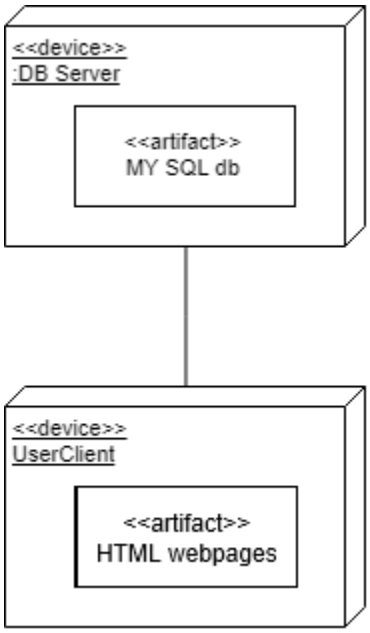


The user interface (View layer) will be created using HTML webpages, the controllers (Controller layer) will allow the webpages to access the data from the model layer (also the controller layer will handle all the logic of the system), the controller layer will access the services from the model layer, the model layer will contain the entities + the repositories.

2.2 Package Design



2.3 Component and Deployment Diagrams

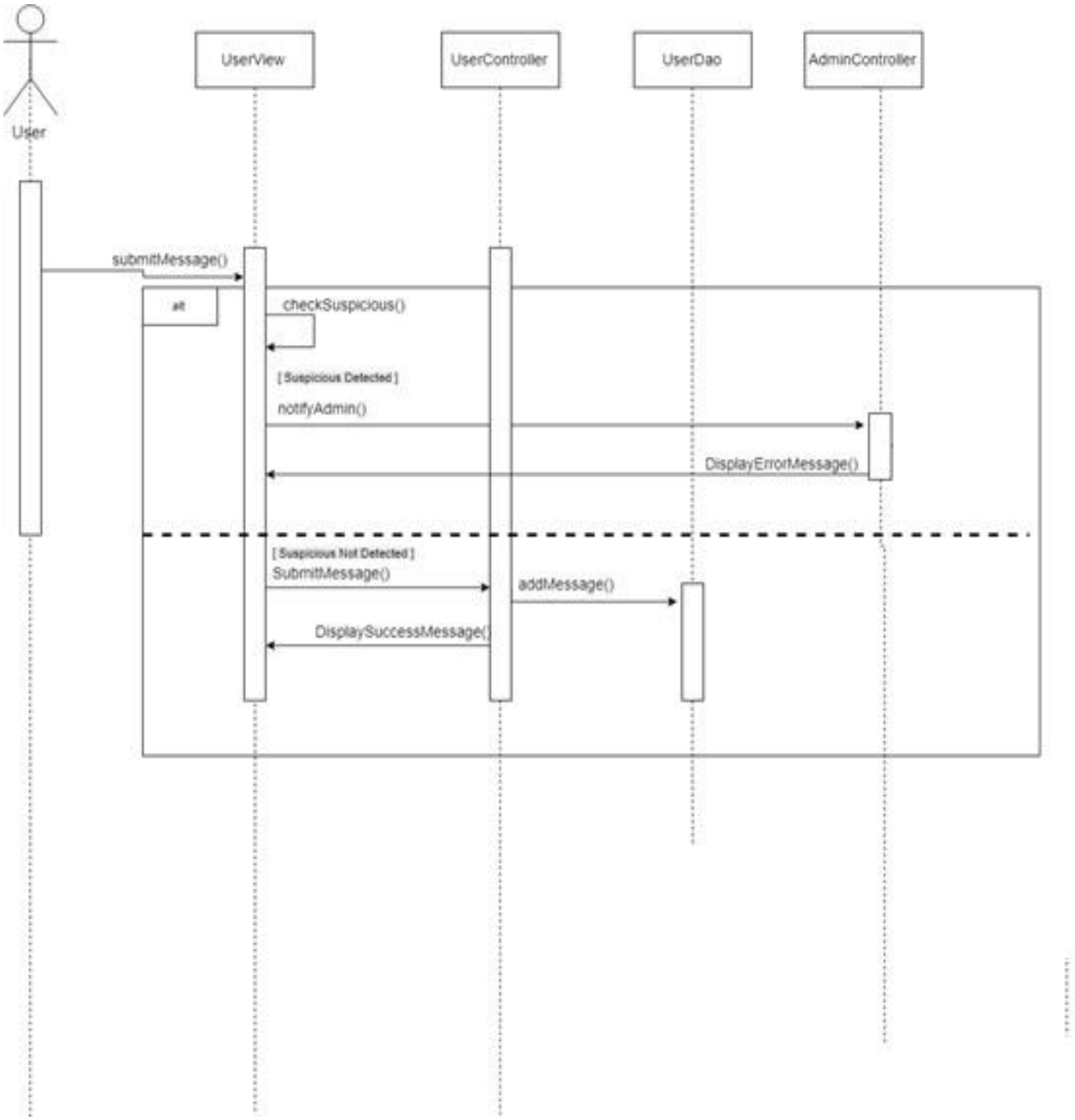


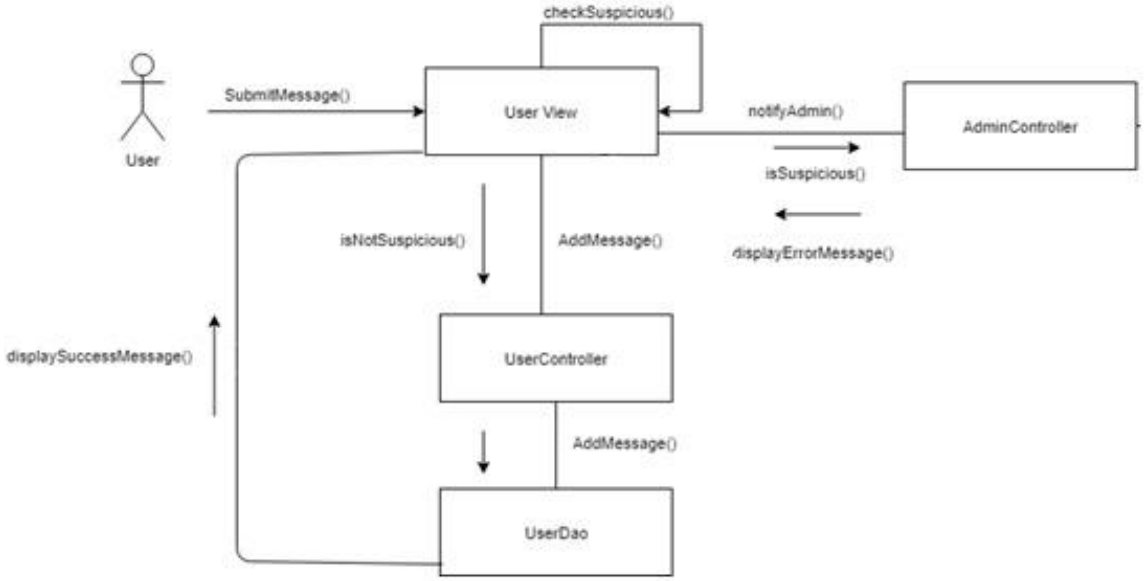
III. Elaboration – Iteration 1.2

1. Design Model

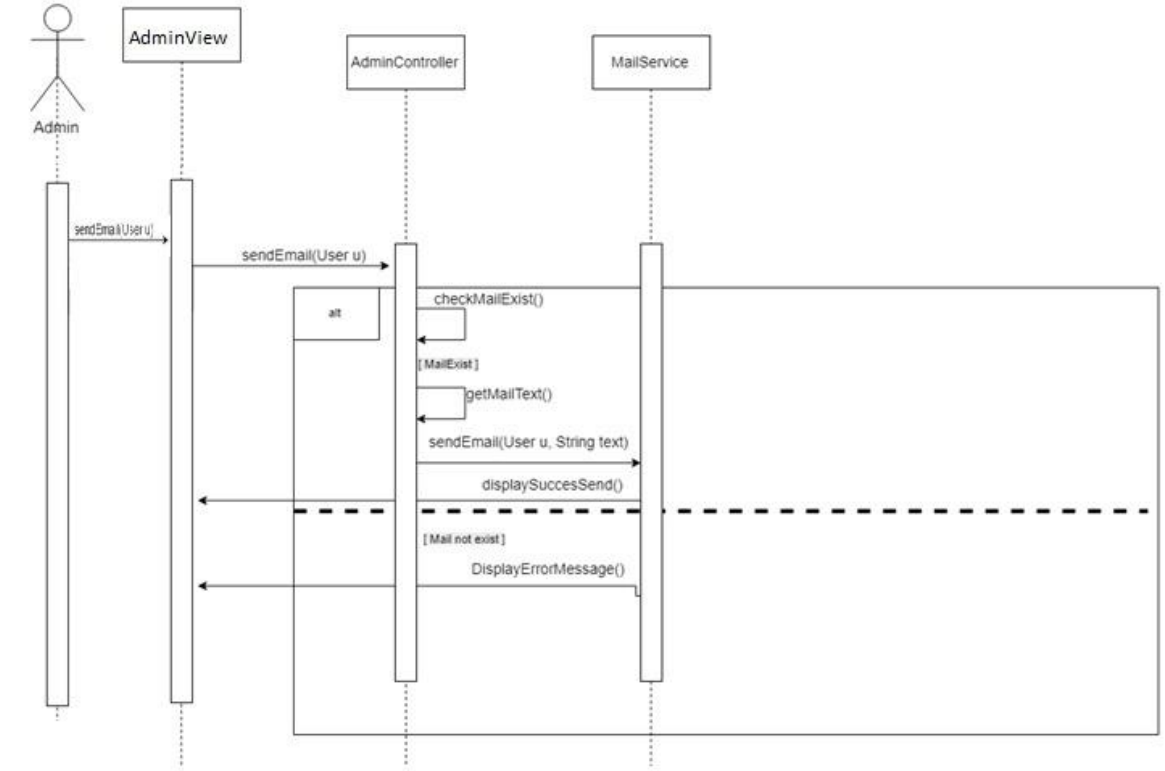
1.1 Dynamic Behavior

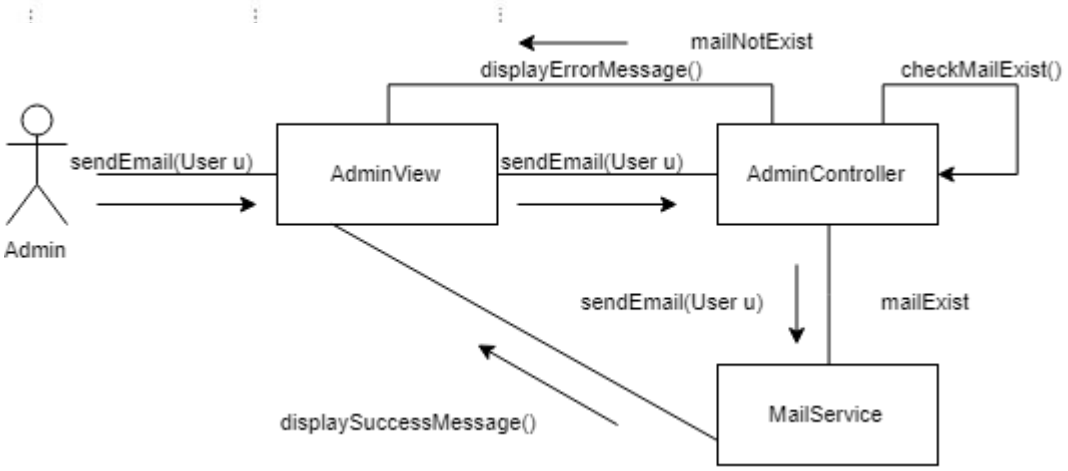
Scenario Submit Message:


User
UserView
UserController
UserDao
AdminController
submitMessage()
alt
checkSuspicious()
[Suspicious Detected]
notifyAdmin()
DisplayErrorMessage()
[Suspicious Not Detected]
SubmitMessage()
addMessage()
DisplaySuccessMessage()

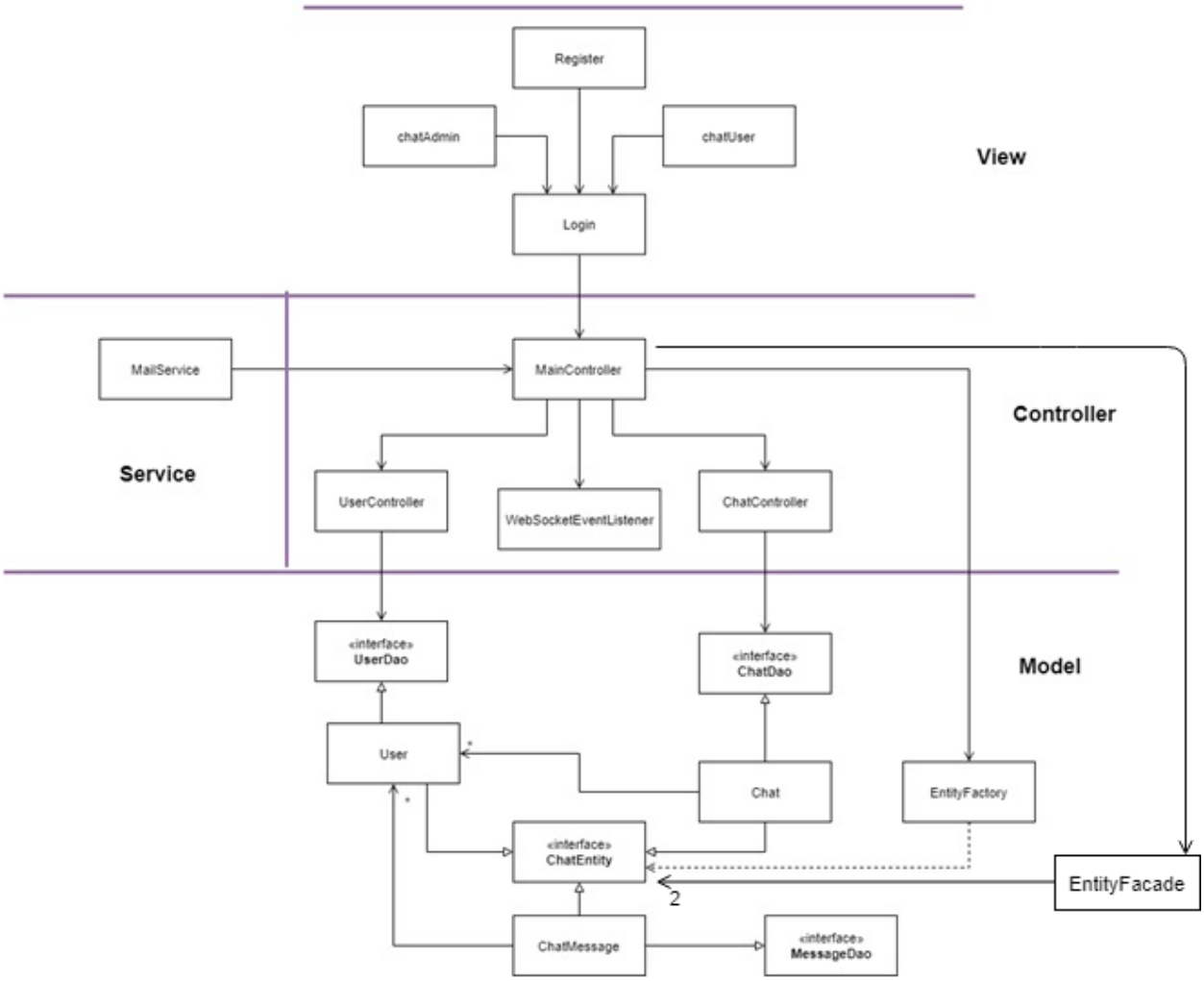


Scenario Send Email:


Admin
AdminView
AdminController
MailService
sendEmail(User u)
sendEmail(User u)
alt
checkMailExist()
[MailExist]
getMailText()
sendEmail(User u, String text)
displaySuccesSend()
[Mail not exist]
DisplayErrorMessage()

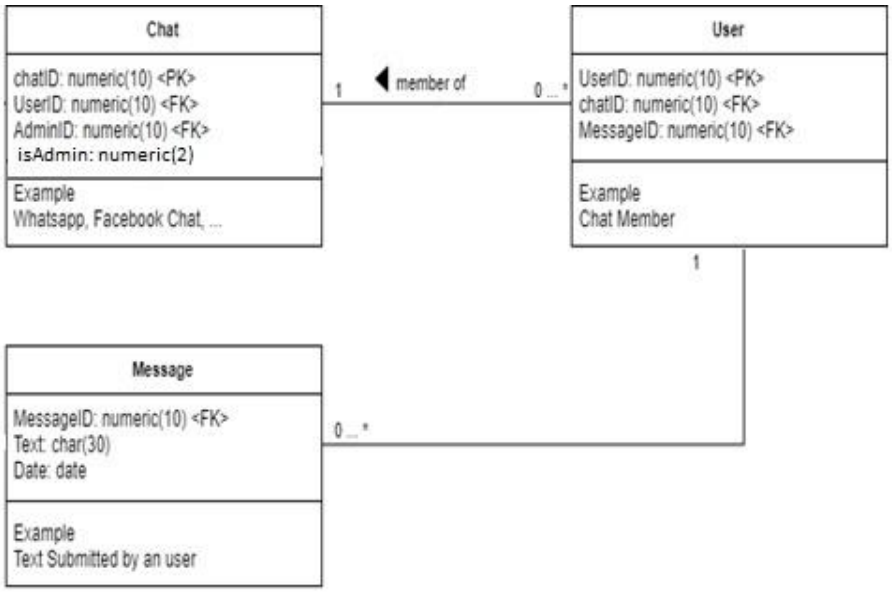

mailNotExist
displayErrorMessage()
checkMailExist()
sendEmail(User u)
AdminView
sendEmail(User u)
AdminController
Admin
sendEmail(User u)
mailExist
displaySuccessMessage()
MailService

1.2 Class Design



I will apply the Factory Design Pattern in order to create the entities, the Dependency Injection for the usability of objects and the façade Design pattern for testing purposes.

2. Data Model



3. Unit Testing

Unit testing will be used.

Example:

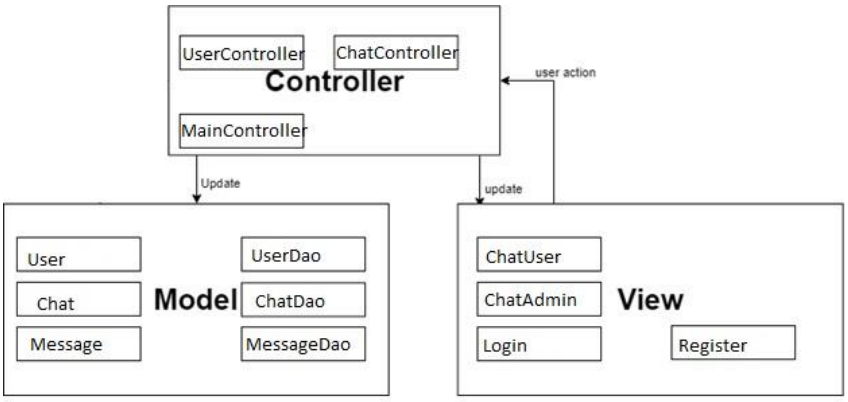
Test Scenario: Validate the login page

Test Case 1: Enter a valid username and password
Test Case 2: create new account
Test Case 3: Enter invalid credentials

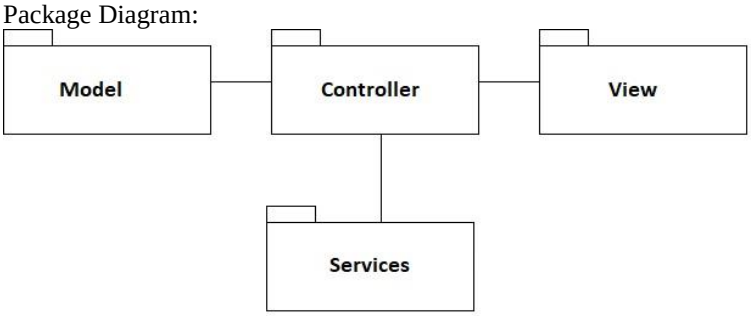
IV. Elaboration – Iteration 2

1. Architectural Design Refinement

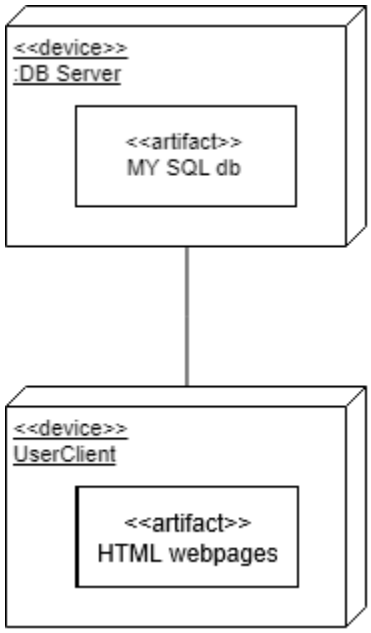
- Conceptual Architecture:



- Package Diagram:

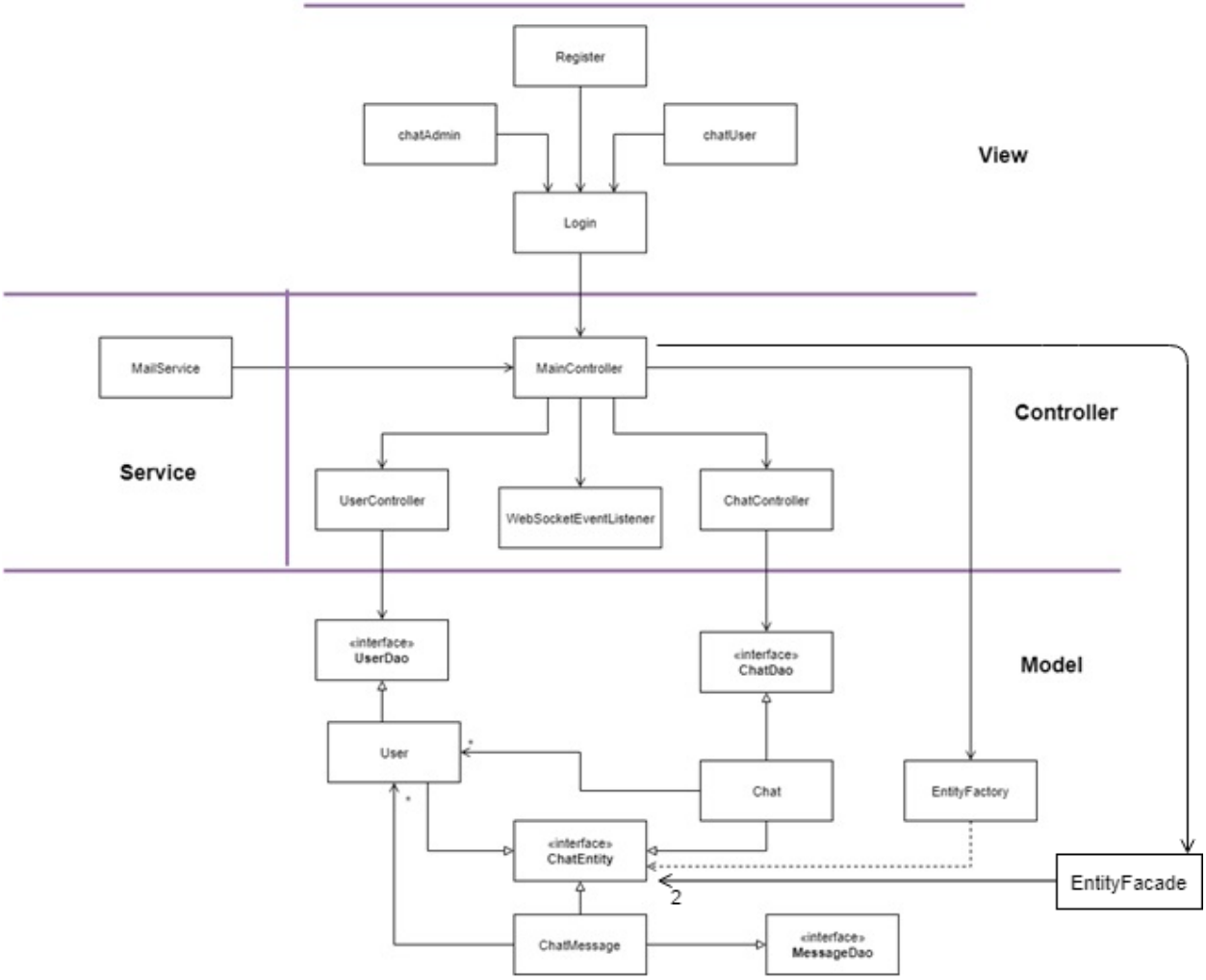


- Deployment:



All the changes were made because of the introduction of the web sockets to my application to handle the chat request allowing them to be refreshed in real time and without user interaction (further explanation below).
The services package that was added contains the mail service to send emails to the users.

2. Design Model Refinement



The major changes are the web socket event Listener which will handle the request of the users and will send them to the topic that I've defined (topic/public) so all the messages there will be handled by the chat, (JavaScript code that automates the process, that is, each time a message is submitted a new paragraph is added to the chat.html webpage)

Also the design patterns added are all used in the main controller. The rest of the classes behavior is the expected one:

- Repositories: database access.
- Entities: define database entities.
- Controllers: handle the logic and update the view.
- Services: mail service which will handle the message exchanging when a user reports another one for example.
- Views: html webpages that the user will interact with.

V. Construction and Transition

1. System Testing

2. Future improvements

The chat could be improved by adding several chat rooms and the users could decide to join a room, and then move between the rooms.

VI. Bibliography