Bilkent University

Department of Computer Engineering

Senior Design Project

QUEX

High-Level Design Report

Project Members:

Barış Ardıç 21401578

Emir Acımış 21201233

Umutcan Aşutlu 21301093

Mert Kara 21400976

Atakan Özdemir 21301134

Supervisor: Çiğdem Gündüz Demir

Jury Members: Fazlı Can, Hamdi Dibeklioğlu

Table of Contents

1. Introduction

1.1 Purpose of The System

QUEX is a mobile social media application which aims to find its users trustworthy individuals to help them with the problems encountered in their daily routine. The program provides a platform that connects the user to another user with required skills to try and solve any problem that can occur such as computer hardware or software issues, daily decision-making, travelling or dining.

Even the simplest problems take huge amount of time when the person facing the problem is not familiar with the problem domain. In addition, although there may be lots of solutions described online to the problem, unfamiliarity with the problem domain makes it harder for the person to understand and put the solution into practice. QUEX aims to solve this problem.

The basic principle of the program assumes that there exists a user with a problem, the user notifies the program of that problem and gets assigned to an another user of the program who is close by if that user is considered an expert on the topic based on user background, in app ratings and user history with similar problems. After matching the users, QUEX then allows these users to start a conversation in order to come up with a solution.

We are trying to solve the problem of “expert finding,” therefore, there is a noticeable amount of features regarding expertise validation. The program is designed to fuse certain methods in order to determine which user would be the most appropriate and helpful for which problem. QUEX considers the feedback from users and the user’s preferences in their profiles. The program uses the GPS data of its users to consider how close the potential expert is to the user with the problem. The closest possible expert is generally the best match. However, there also exist situations where the proximity of the expert is irrelevant, so the user can ignore the generic matching

method and push the notification directly to main dashboard under a certain category where the problem can be seen by users without matching. This use cases that benefit from locality can present themselves in a variety of situations.

Consider that you are in a university campus and you are having trouble with your computer. A simple notification to QUEX can quickly refer you to an engineering student t whom may be able to help you instantly. In completely different setting imagine that you are standing in front of the movie theater and trying to decide which movie to see, notifying the program will potentially result in a match with another user who just come out of a movie in that theater. The locality (user GPS data) aspect of the program provides the user with exclusive information that cannot be found on the internet such as how to connect to Bilkent VPN for the first use case example or information about the sound system of the movie theater for the second example.

1.2 Design Goals

1.2.1 Extensibility

QUEX is a nonspecific question and answer platform. As it doesn't focus into a single domain, it is difficult to predict the domain specific usage of the application. Although it is general-purpose, users may mostly use the application for a single or several domains, restaurant and movie recommendation, for example. Depending on the usage therefore, additional functionalities and changes may be added to meet specialized user demands. This brings the extensibility of current features as a concern.

1.2.2 Supportability

QUEX should be easily adapted to new technologies or features via new modules. Object oriented design principles will allow for these type of expansions. New updates will be regularly provided to make sure the application is up to date and innovative.

1.2.3 Usability

QUEX targets users from a variety of domains. In order to be appealing for a broad target group like this, QUEX working mechanisms must be easy to understand without an extensive guide or further instructions. Only some tips should be shown in the

application to make some features easier to understand. Also, a friendly user interface is necessary since the targeted user base is this large.

1.2.4 Availability

QUEX server should always be available for the users. QUEX will be a mobile application which can be accessed very easily and at any time of the day. The application should be free to use, users can download and use it without paying money.

1.2.5 Reliability

Bugs and errors should be eliminated from the final version of the application. QUEX should detect any undesired substrings with its black-list mechanism such as bank account numbers to prevent use case of making profit by the application. Personal data of the users should be preserved well. The Firebase Real Time Database provides security management for our application. Authentications, validations and permissions can be set easily because of our extensive Google Firebase integration.

1.2.6 Portability

QUEX will be made for Android system but it can be ported to different operating systems like iOS, Windows with a little effort. Different ports of the program may be provided after the first release.

1.2.7 Scalability

Increasing number of users is both an expectation and purpose of QUEX. QUEX needs to adapt and handle this increase. More importantly, the question and answers in the database will not be deleted but kept for future reference. This means that the data of question topics will never decrease but increase cumulatively. QUEX needs to handle this growing data and such data should be evaluated regarding its relevance periodically.

1.3 Definitions

API: Application Programming Interface.

Google Firebase: A collection of services and APIs for android programs developed by Google.

MVP: Model View Presenter.

NoSQL: Not Only SQL

Questioner: A user who ask the question.

Expert: A user who is capable of answering the questioner problem.

1.4 Overview

The idea of QUEX came from the problem of expert finding. QUEX aims to solve this problem in a broad range of domains, in 1-to-1 manner, and in real-time if possible. These approaches make QUEX different than other question & answer sites such as Stack Overflow. While pursuing this goal, we value some design goals such as extensibility, usability and scalability. Our choices of design goals are mainly due to our choice of building the application non-domain-specific, as this causes a non-uniform group of people to be targeted. Another important factor in our design goals is the possible large amount of the data involved. There were also other aspects in consideration like performance but this is a desirable element of design for any application therefore this report does not specifically argue elements like performance or responsiveness. The mission of QUEX is to solve the problem of expert finding in a rapid and generic way while considering the end users own immediate vicinity when it has relevance to the possible solutions of the question at hand.

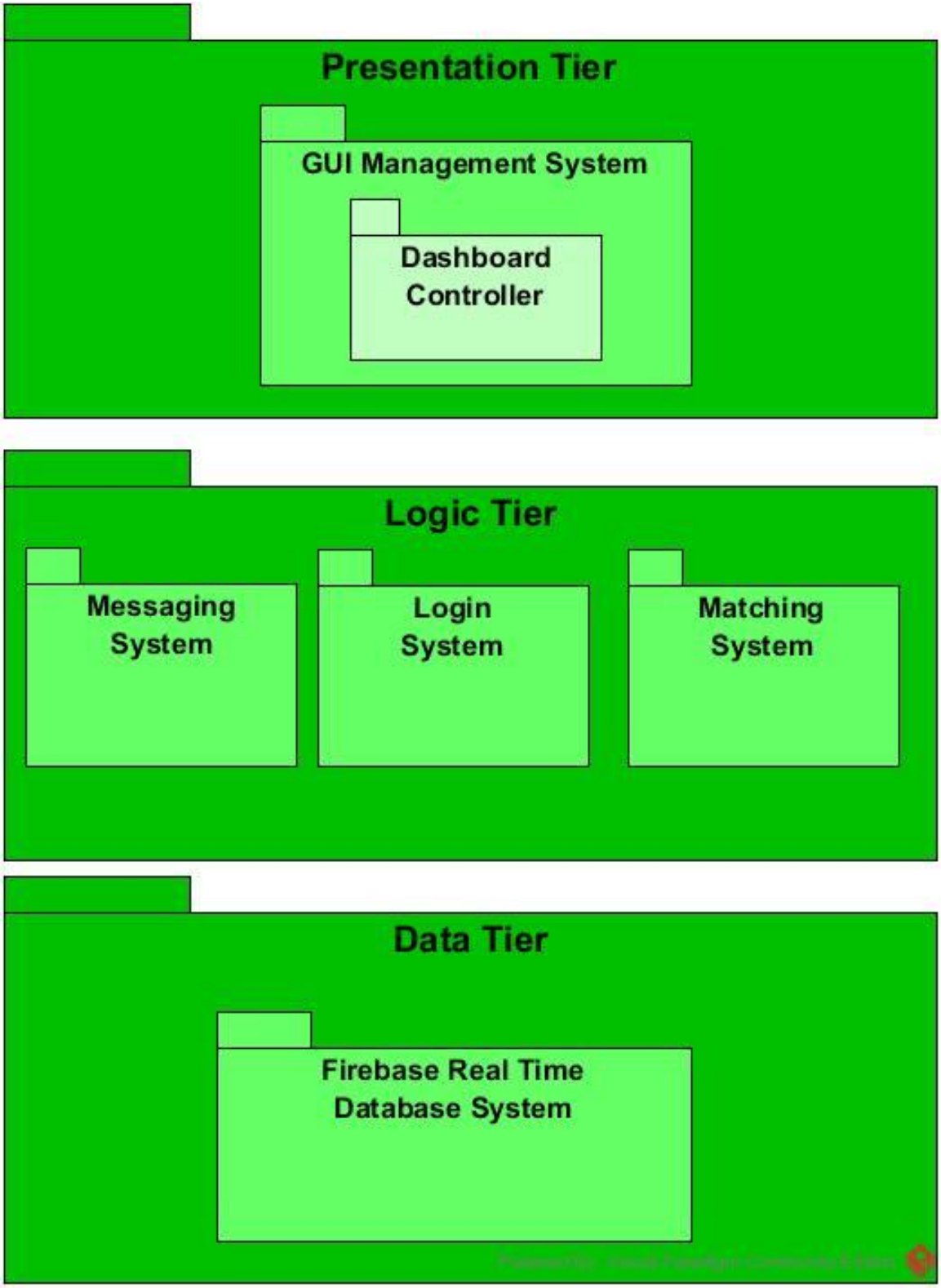
2. Proposed System Architecture

2.1 Overview

QUEX will be an Android application whose main functionality is to connect people who need help and people who can help in real-time, if possible. In addition, previous problems people faced should be kept and updated for future reference for others who could face a similar problem and provide the old solutions to them in a reliable way. Besides these, security is an obvious concern. Some technical necessities in a broad sense are arising in order to provide these functions with an emphasize on concerns and design goals, are to allow users to use their Facebook/Google accounts to log in, match users in a reliable way in terms of the topic and the expertise relation, allow them to communicate and storing the data. Taking all these into account with the described design goals in the previous sections, the following sections will address the system architecture we propose to maintain necessary functionalities while achieving design goals, in the subtitles of system decomposition, hardware/software mapping, data management, access control and security.

2.2 System Decomposition

The derived problem sets to be handled are getting user location, matching users, messaging, storing data and notifying users and the corresponding subsystems are Match Controller, Chat Controller, Firebase Real Time Database and Authentication. We used the 3 tier module view in the system decomposition.



1- 3 tier module view of QUEX

2.2.1 Presentation Tier

The presentation layer includes the subsystem GUI Manager, which includes the subsystem dashboard controller. These subsystems are responsible for presenting the data while allowing a user interface for the end-user of application to interact with the application.

2.2.2 Logic Tier

This layer includes chat, account, match controller and topic subsystems. The bulk of the functionality lies here and the rest of the tiers are either concerned with presenting these features to users or storing data for the needs of the application

2.2.3 Login System

This subsystem is responsible for authenticating our users. It does so by connecting to Firebase Authentication API which provides us with reliable and extendable functionality.

2.2.4 Messaging System

Messaging subsystem consists of Inbox and Chat controller. It is responsible for sending and receiving messages. It connects to the Firebase Cloud Messaging service and also utilizes its capabilities for sending notifications which could be set on real time or ahead of time (these notifications are not from user to user but from the developer s to users).

2.2.5 Match Controller

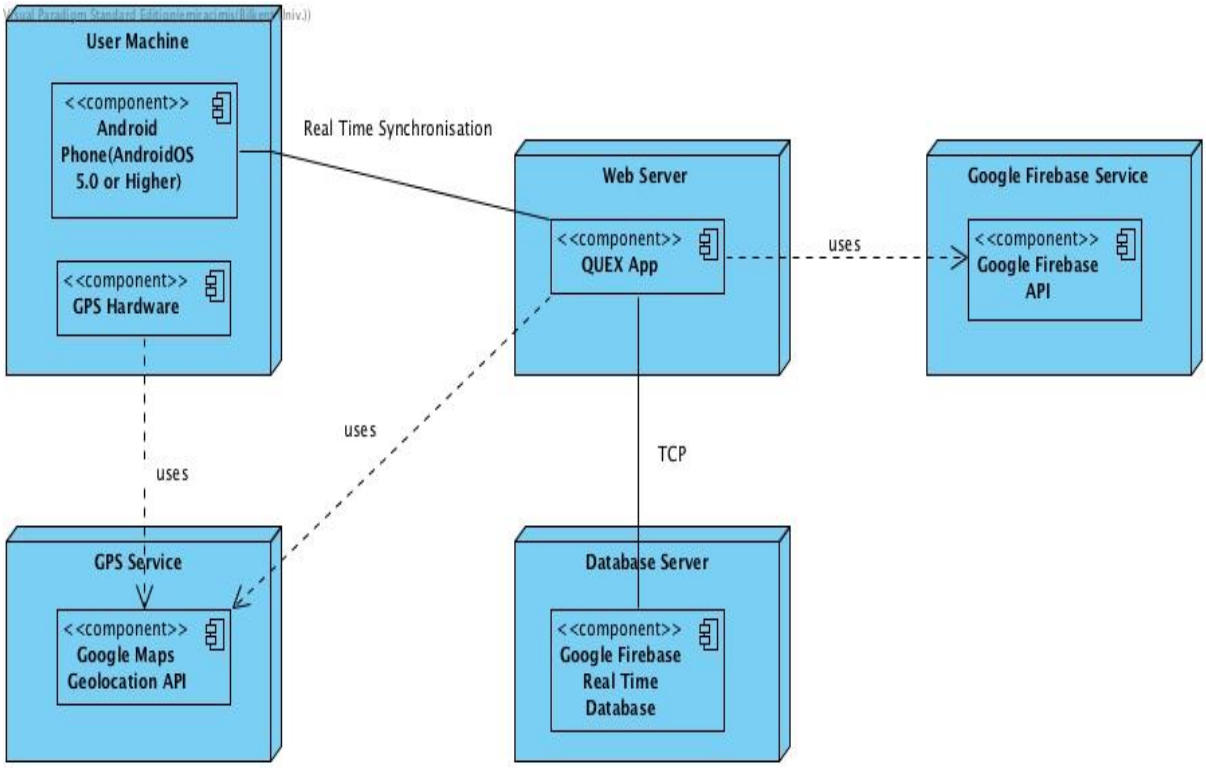
Match controller is responsible for searching a suitable expert for the problem. It is triggered with topic creation by the user who has a problem. If a match has found, it calls the chat subservice to allow the expert and the user to communicate.

2.2.6 Data Tier

Google’s Firebase Real Time Database is used for storage. This is a cloud stored real time NoSQL database. The reason behind selecting this database is its performance acquired from the utilization of data synchronization in real time. Also the cloud aspect provides a certain comfort to the application regarding stored data. With cloud storage managing the amount data stored is much more convenient.

2.3 Hardware/Software mapping

QUEX's presentation and logic tiers will be mapped to the Android device. Subsystems in the logic tier will provide the interface on connecting, manipulating and retrieving data, the application will synchronize with the server when there is new data. The application will require internet connection and the GPS component of the Android phone.



2- Hardware/software mapping of QUEX

2.4 Persistent Data Management

QUEX is an application which aims to instantly match a questioner with an expert. To reach this purpose, server and the application should have a fast connection since efficiency is important. Also, we need to store lots of data such as user profiles, settings, topics, messages etc., Google's Firebase Platform has a cloud stored real time NoSQL

database. This feature complied with all of our needs and allowed the team to focus more on user side of the application.

2.5 Access Control and Security

Users need to sign up with their full name, username, email and password to use features of QUEX. Facebook and Google accounts can also be used to access features of the application. Then, QUEX assigns some expertise for the user by fetching profile information from the chosen platform if anything useful is available. User can select new expertise or remove existing ones in the profile section. User can save his/her account with a registered email address in his/her profile. The Firebase Real Time Database handles security issues which includes permissions, validations and authentications. The application will need network connection to run.

2.6 Global Software Control

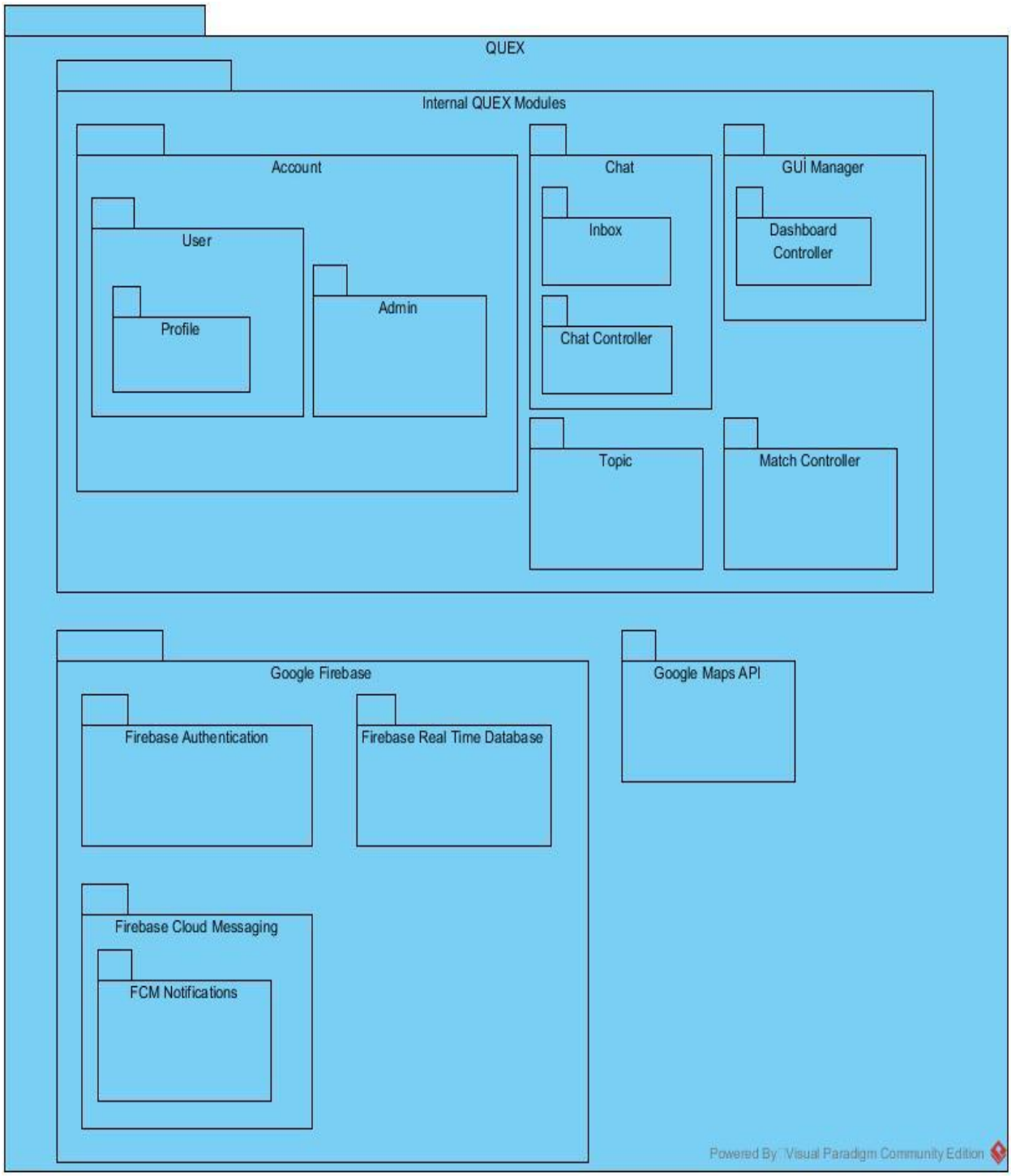
The application is event driven and utilizes the model view presenter architectural style. Each screen displayed to the user is a view and each request created by the user runs the corresponding code segment in the application logic(model). The outputs of the program are updated when there is new information available (synchronization). The presenter segment is responsible from the software control. This segment connects model and view segments together therefore it handles delivery of model outputs to views as inputs or view outputs to model inputs.

2.7 Boundary Conditions

- **Initialization:** Firstly, user needs to download mobile application of QUEX on his/her Android device. When user starts the application for the first time, a sign up screen will be appeared. He/she can use Facebook or Google account to login to the system directly, or user can fill the corresponding fields with proper information (full name, user name, e-mail and password) to create a QUEX account. When he/she enters the true information on login screen, user will be allowed into the application. At this point, user can use QUEX. When the user opens the QUEX application again, he/she does not need login again. He/she will be already logged-in after the first login process.
- **Termination:** Log out option is one of the ways for terminating a user session. When user logs out, QUEX redirects he/she to login page again. Also, because of this is a mobile application for an Android device, users can directly close the program and get back to the main menu of the mobile phone.
- **Failure:** User actions may cause failures in the system. If user does not fill the necessary fields in QUEX when he/she tries to login, the application will show corresponding error messages to the user. In failure on internet connection, the application will not work. If there is a problem in the API's which QUEX utilizes this will be reflected in QUEX.

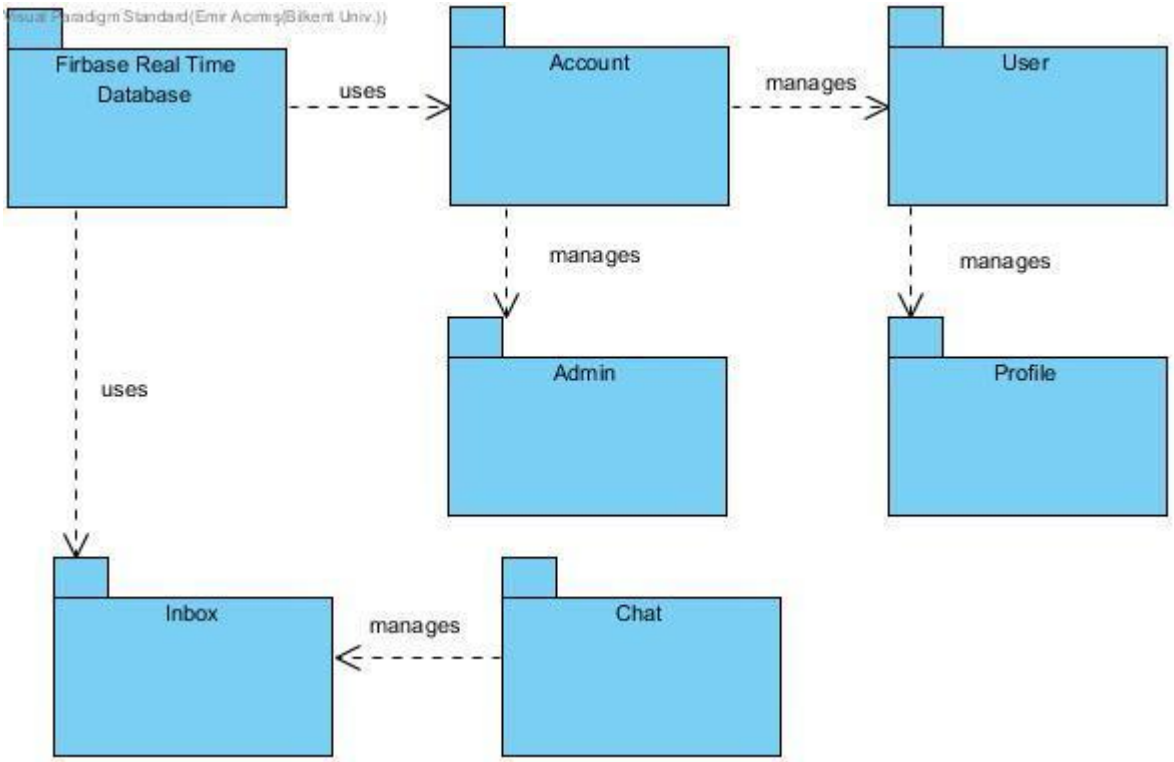
3. Subsystem Services

The subsystem services are decomposed regarding the grouping according to the 3-tier view above section 2.2. A more detailed decomposition of QUEX is given below.



3- Decomposition view of QUEX

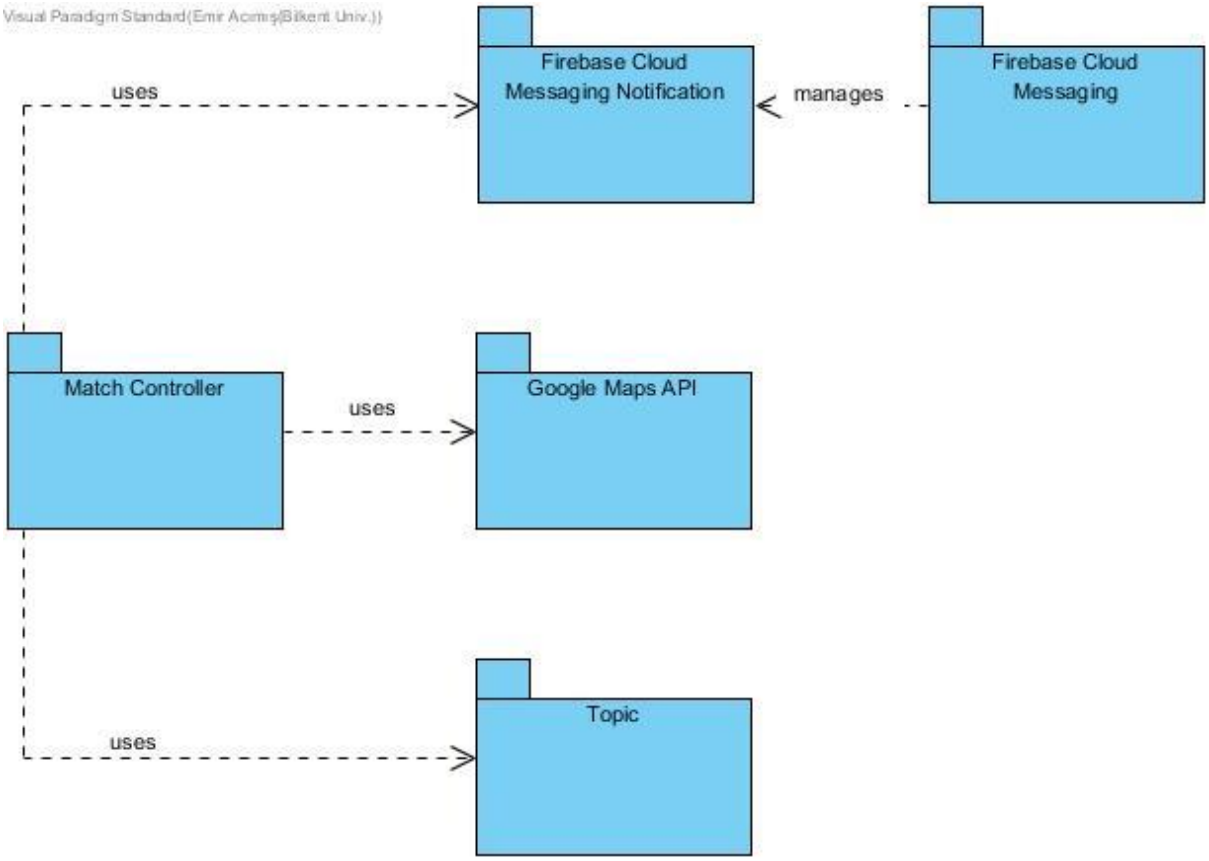
3.1 Subsystem Service for Data Management



4- data management subsystem

Concept	Description of the Solution Domain Concept
Firebase Real time Database[1]	NoSQL cloud store database, part of Firebase Platform
Account	Manages user information, preferences
User	End user account type
Admin	Account type of developers and admins
Profile	Has user information and preferences
Inbox	Collects the questioner chat with experts
Chat	Has the messaging functionality

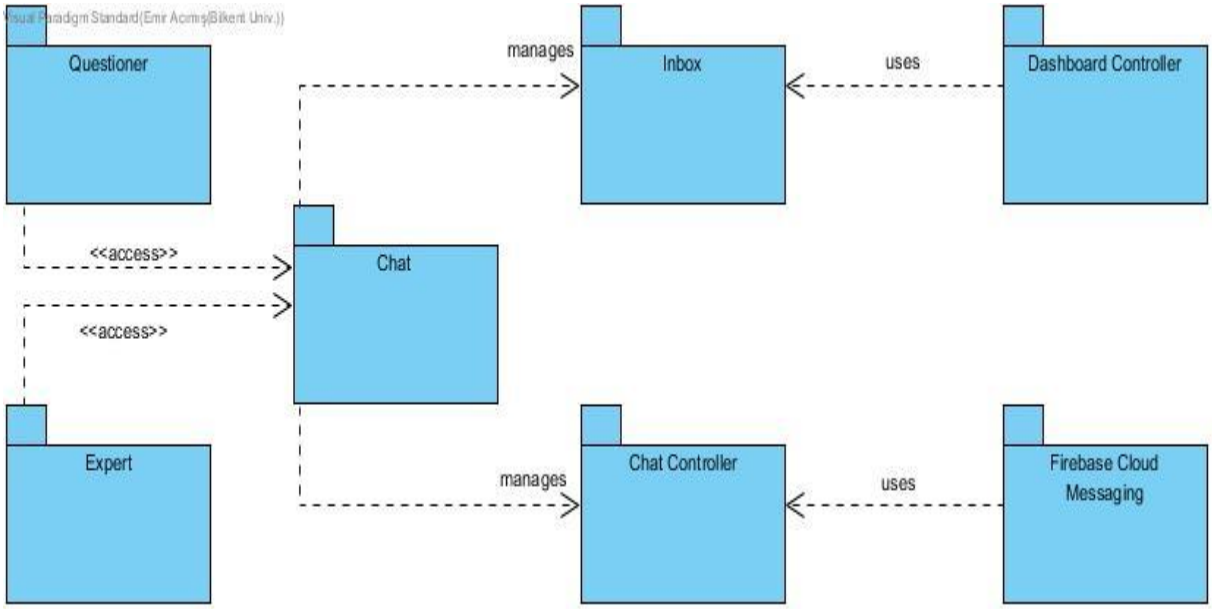
3.2 Subsystem Service for Match Algorithm



5- Matching Subsystem

Concept	Description of the Solution Domain Concept
Match Controller	Controls matching process
Firebase Cloud Messaging Notification[1]	Handles the notifications of the system
Google Maps API[2]	Getting the GPS data of the users
Topic	Questioners problem that write on the program
Firebase Cloud Messaging[1]	Messaging service of the program

3.3 Subsystem Service for Messaging Service



6- Messaging Subsystem

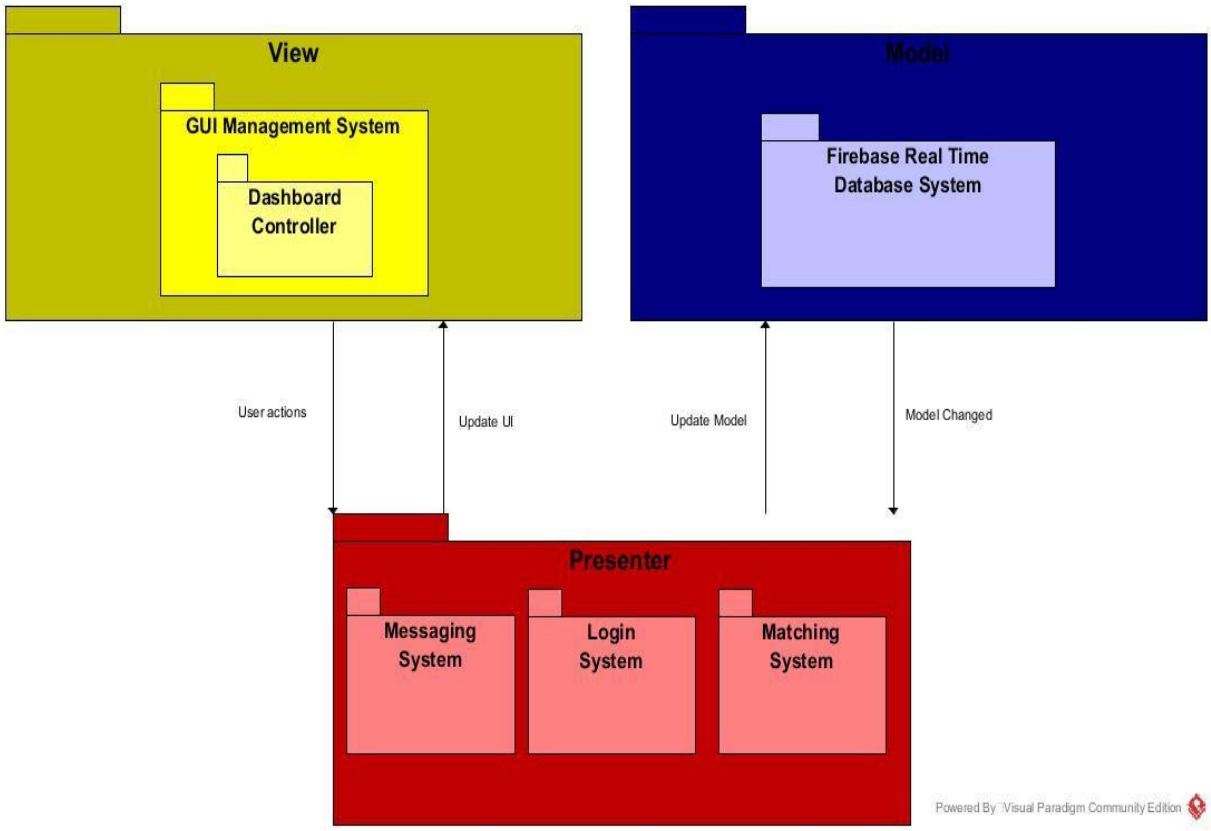
Concept	Description of the Solution Domain Concept
Questioner	The user who write the topic to the system
Expert	The user who matched and answer the topic of the questioner
Chat	Has the messaging functionality
Inbox	Collects the questioner chat with experts
Chat Controller	Controller of the chat functionality
Dashboard Controller	Controller of the dashboard of the program, locally stores data at runtime
Firebase Cloud Messaging[1]	Handles the send and receive messaging between the questioner and expert

3.4 Architectural Style

The MVP architectural style is selected for QUEX application because it has integrated XML modules in it. These modules do not have any logic or decision making in them. They are used for style and user interface design. This allows for XML components to be seen as views. If the XML modules had logic in them, then the decision of architectural style would be moved towards to MVC (Model View Controller) style which allows for communication between view and model components. MVC is still a suitable architecture and it could be interchanged without much effort with the current style. However; the current MVP style is more contemporary for Android development process.

MVP is also preferable for its higher level of decomposition which avoids unnecessary architectural complexity which results with easier implementation process. This design choice also made maintaining the code base easier for the development team.

Model component is responsible from the system logic and system state storage. View component is responsible from the GUI of the system. It handles the events that occurred by the presenter component. Presenter component's major functionality is providing communication of the view and model components, it sends queries to the model and updates the view accordingly



4. References

[1] Firebase. (2017). Firebase. [online] Available at: https://firebase.google.com/ [Accessed 28 Nov. 2017].

[2] Google Developers. (2017). The Google Maps Geolocation API, Google Maps Geolocation API | Google Developers. [online] Available at: https://developers.google.com/maps/documentation/geolocation/intro [Accessed 28 Nov. 2017].

[3] Developer.android.com. (2017). Android Developers. [online] Available at: https://developer.android.com/index.html [Accessed 28 Nov. 2017].

[4] Amazon Web Services, Inc. (2017). What is NoSQL? – Amazon Web Services (AWS). [online] Available at: https://aws.amazon.com/nosql/ [Accessed 28 Nov. 2017].