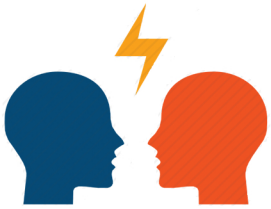
OBJECT-ORIENTED SOFTWARE ENGINEERING



DEBATE IT

FINAL REPORT

GROUP 1B

21301294 **YAĞIZ GANİ**

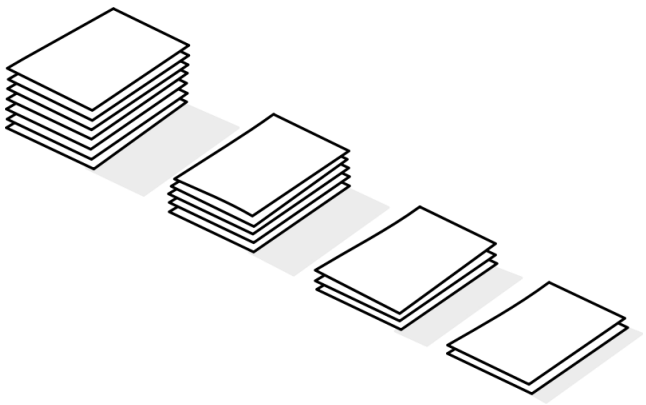
21502938 **ÇAĞATAY SEL**

21200846 **AHMET SARIGÜNEY**

21501109 **YASİN BALCANCI**

SECTION 01

05.05.2018



INTRODUCTION

- **Ultimate analysis**

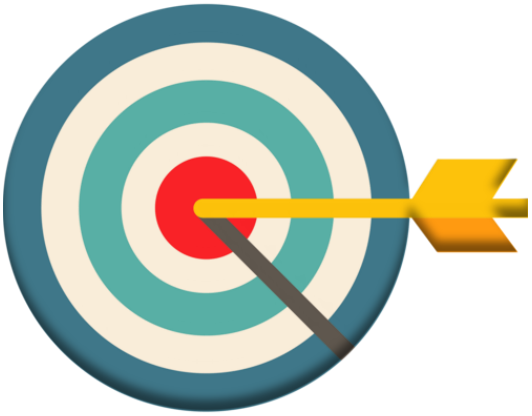
We proudly present the mobile application "Debate it" as an overall product at last. It was accomplished through a long period of time and lots of effort were fairly made in a body. We as a group had some aims and wishes about the fictionalised state of game before everything gets start and finally we came to an end by actualising most of them.

Strictly speaking, "Debate it" was considerably achieved in parallel with our objectives. We went through lots of phases to get to that final point. Thus, a vast amount of progress were made for finalization. Ofcourse it is wrong if we say that any conflicts in opinions did not occur, however, we maintained to be harmonious in respect of teamwork. The most significant phenomenon in that long-running road was to perform cooperation properly. We strongly believe that each of us made a sacrifice in the context of cooperative work.

- **In respect of final state**

  In a nutshell, the current status of our implementation, some clarifications about the design and what kind of experience we gained will be enlightened in this documentation. Besides, there is user's guide part in order to introduce the limits of our program.

  All kinds of architectural material in our data repository embodied and the final outcome in regard to concurrent execution was achieved; nevertheless there are some incomplete points according to our early goals which will be broadly clarified on the implementation part. Even so, we as a group moved forward in line with our persistence. Observing the final state of "Debate it" excited and also motivated us up to the present; therefore, we undoubtfully tried our best.



IMPLEMENTATION

- **Changes in design**

In our initial design, we planned that UserHandler class will be always listening to client for requests. However, we have noticed that this is very CPU consuming and not necessary. Instead of such a design, a new UserHandler thread will be created for each request and when the request is finished, thread will stop executing.

DataReceivable interface will have an additional method to update data receivement progress to UI. This way, user will be able to see that they are waiting for server's response. In our initial design, we planned that all of the UI classes should implement this interface. However, not all of the UI classes interact with server. So, only the necessary UI classes will implement this interface.

While designing the client part, we tried to define the necessary functions of the game on the UI classes. UI class names and their functions are not changed during implementation. However, there are some little changes on the methods. Classes that will connect with the server have a ServerBridge object that will request and get data from the server.

We moved the responsiblity of starting AsyncTasks to update UI from UI classes to ServerBridge. UI classes is only required to override method of DataReceivable interface in order to get updates to their UI from server.

Our initial design which aimed to use fragments to implement UI has changed. All UI classes are now child of Activity class and not Fragment class.

- **Implementation status**

We managed to implement UserHandler and main Server classes. With these two classes we are able to establish communication between multiple clients and the server and send objects through network. We have also implemented ServerBridge class which handles the communication in the client side. This class handles the requests coming from UI classes and forwards them to server and notifies UI classes with responds coming from server machine.

We have also implemented BattleThread and PlayerHandler classes which are two main class in server side that is responsible for multiplayer game experience. These classes allow 4 player to join a game lobby and play the game by using multi threading.

In client side, we have managed to implement some of the UI classes which are LoginActivity, RegisterActivity,MainActivity, BrowseLobbiesActivity, BattleActivity and model classes. We have also created 3 small,medium and large size avatar that users can choose.

In database side, we have managed to implement UserManager and DebateManager classes but ItemManager class is not implemented. ItemManager class merged with UserManager class and some of its functionality is moved to UserHandler class.

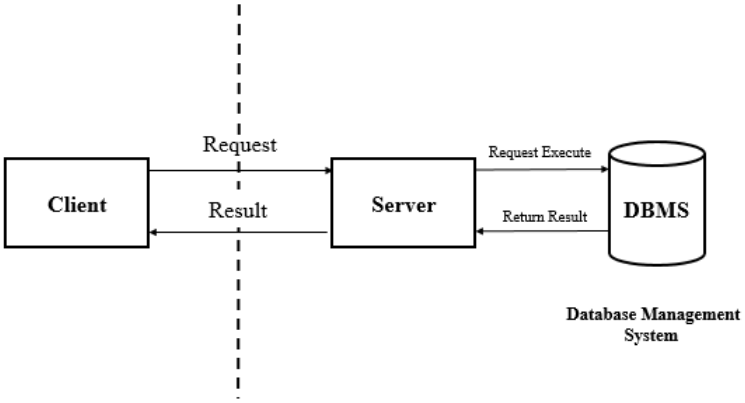
Our current system does not implement voting to past debates and replaying past debates. We also did not implemented point system which allowes users to buy new items. These features are left to be implemented in future due to limited time.

GAINED EXPERIENCE

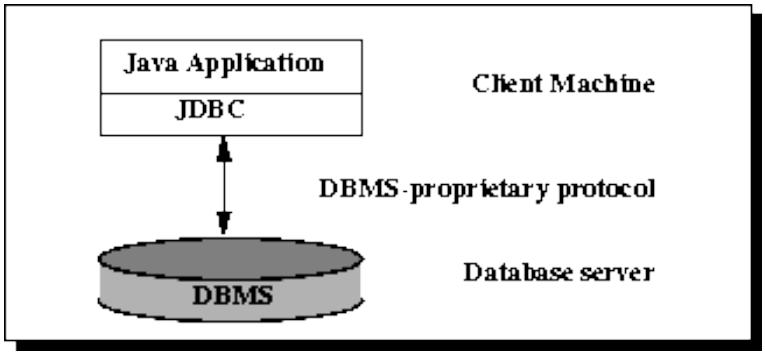
- **Database management system**

Database management issue of the program is handled by using structured query language(SQL). All kinds of datas are easily stored and manipulated in this manner.

In a nutshell, SQL is a language that provides an easy way to communicate with a database. Basically, a sturdy database can be established with the help of a few ordinary commands in SQL. Table objects are used to store all kinds of datas. The reason why we decided to use SQL is related with the complexity of our program. Using notepad or something like that could be another and the simpler way in order to process data set, however, "Debate it" requires much more efficiency in data operations like access and retrieval due to having a large amount of data. In addition to that, they have to be served users simultaneously and thus using an SQL server was considered as the right way to deal with the database management.

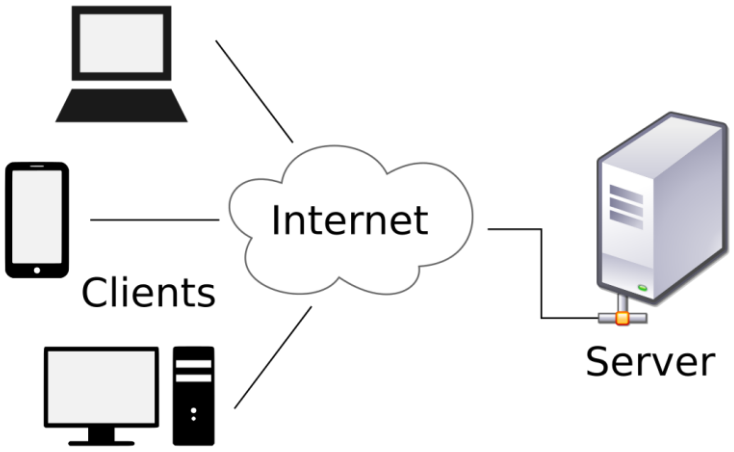


Another issue that we encountered about processing database is integrating our database system with game implementation. We are utilizing Java language to construct the game and SQL to build a stable database system for game. In this context, an application programming interface called "Java Database Connectivity" was needed to handle the interconnection between them. It basically provides our Java program to access the database management system.



- **Client-server architecture**

  To implement a server/client architecture we had to learn about multi-threading so that the server can handle multiple clients. Also, since we are implementing server in java, we learned about the Socket and ServerSocket classes in the java.net package and how to send objects through network by using these classes.



- **User interface**

There are two main parts on UI creation. .xml coding and creation of GUI components and manipulating the created GUI components by using java code.

- ⇨ ActionListener class and its inner components are handled.
- ⇨ Intent class of android for the activity passings is learned. Creating Intent object and directing the object to another activity's java class is handled.
- ⇨ Button creation is handled.
- ⇨ Android'S TextView object and displaying data on TextView objects is handled.
- ⇨ OnClick method of each GUI component (Button, Uri, TextView, Multimedia Object, Intent etc.) is handled.
- ⇨ Listview creation and other types of Lists are handled.
- ⇨ How to show an arrayList of users' information on the ListView and other types of List objects is handled.
- ⇨ Creating Login Interface and activity changes during login activity is handled.
- ⇨ Stopping taking input in certain cases (if it is already taken etc.) is handled.
- ⇨ Because we deal with a server, what to do to not perform before the data is received in order for application to not get confused are what we have learned so far.

USER'S GUIDE

- **System requirements**

  Server could be deployed to any windows machine that has jre.1.8.0_161 or higher version of JRE. Server machines firewall must allow incoming request from port 54134 (this could change) and must be in the same network as client devices. Server is only tested on Windows 10 machine so it recommended that it should be deployed on a Windows 10 machine.

  Client devices must have Android 4.4 (KitKat) or newer versions of android OS. They must be in the same network with server machine.

- **Installation**

  There is no need for installation of the server. It can be distributed as an executable file.

  Client devices can install the application from the distributed .apk file by following the standard android application installation progress.

- **How to debate**

  - ⇨ Select battle to enter.
  - ⇨ When there are four players in a battle, debate starts.
  - ⇨ Choose if you agree or not with the auto-generated idea. You will be assigned to be a participant in the debate or a spectator according to the side you have chosen and the amount of debates you were in.
  - ⇨ If you are a participator, write your arguments according to the idea generated and responses given to it for four rounds each of which will last 60 seconds.
  - ⇨ Select side according to the performances of debaters at the end of the debate.
  - ⇨ You can also see your past debates in main page and view others' in browse tab and vote for their debates too.