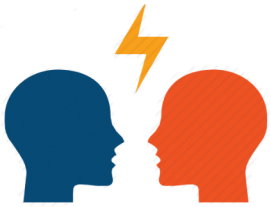
OBJECT-ORIENTED SOFTWARE ENGINEERING



DEBATE IT

DESIGN REPORT

GROUP 1B

21301294 **YAĞIZ GANİ**

21502938 **ÇAĞATAY SEL**

21200846 **AHMET SARIGÜNEY**

21501109 **YASİN BALCANCI**

SECTION 01

03.03.2018

INTRODUCTION **2**
Purpose of the system .....
Design goals.....
SOFTWARE ARCHITECTURE **3**
Subsystem decomposition.....
Hardware/software mapping.....
Persistent data management.....
Access control and security.....
Boundary conditions.....
SUBSYSTEM SERVICES **6**
Database management .....
Server models .....
Client bridge .....
Controller classes .....
Cliend models.....
Views .....
LOW-LEVEL DESIGN **15**
Object design trade-offs.....
Design pattern.....
GLOSSARY & REFERENCES **17**

INTRODUCTION

This section explains why we come up with such a design and the tradeoffs that were made during the design.

- **Purpose of the system**

  Purpose of the Debate It is to provide people an environment where they can practice their debate skills in an enjoyable environment. Our system wills also aims to provide its user with the replays of debates that other users participated in so that even the users that only want to read about the ideas of other people can utilize our application. Debate it also offers a variety of buyable items to users in order to enable them to customize their profiles.

- **Design goals**

  Primary goal of this design is to construct a system that could meet the functional and non-functional requirements that we stated in the analyze report. To meet those requirements we designed a client server based system so that we can support multiplayer gameplay. We also designed a database management interface to make the connection between a MySQL server and clients.

SOFTWARE ARCHITECTURE

- **Subsystem decomposition**

- **Hardware/software mapping**

Our system has 2 different types of hardware. One of them is the server which will be deployed in a windows machine and will be implemented in JAVA language with the latest jdk.Server machine will also have a MySQL database. Database and server connection will be done by using Java Database Connectivity (JDBC) api.

Second hardware in our system is the client devices which are mobile phones that run Android OS. Our system will support Android

KitKat and later android versions. Server and client machines will be in the same local network communicate with Client Bridge and Server Bridge subsystems. These subsystem will utilize Socket Programming in order to establish the connection.

- **Persistent data management**

Data Management is a crucial part for our system. Server will be responsible for storing and managing the data. Users have to register to our system and sign in to use our system. Necessary information such as username and password will be stored in the server machine. Server will also keep other user related informations such as inventory, past debates etc. The server machine will also keep record of all the past debates so that users can view or replay these past debates and keep voting.

Client machines will keep some of information regarding their users. These informations will be user options and their username and password so that users will not need to type their username and password every time they try to log in.

- **Access control and security**

In order to have a functioning application, users must set up the game by .apk file and must be connected to the same LAN with the server. Then the user must login with a pre-registered account, if he has one; otherwise register to the game and login. Because all the objects including User are kept in database, not in client, personal info will be safe and secure against possible attacks.

- **Boundary conditions**
  - **Starting**

The user must have the application that was installed via provided .apk file and must be connected to the same LAN with the server.

  - **Login**

The user must have a ID-password matching, pre-registered account in order to login.

- **Logout**

  The user must be logged in in order to log out. He can log out by tapping the "Log Out" button in the slide-menu.

- **Shutdown**

  The user will be able to quit the game by double-tapping "back" button when he is in main menu.

- **Exception Handling**

  Any possible lethal error due to connection problems with the such as problem at geting the User object will be printed to screen with a proper explaination. Any nonlethal problems will be tried to handle.

SUBSYSTEM SERVICES

- **Database management subsystem**

  - **ItemManager class:** Makes us able to get item objects according to their IDs from the database.

    Item[ ] getBuyableItems(int userID): Takes user ID as parameter and returns the items that that User has not got in its inventory.

    Item[ ] getUserItems(int userID): Takes user ID as parameter and returns the items that has been bought by that User.

    void storeUserItem(int userID, int itemID): Takes user ID and item ID as parameters and records that item to the users inventory.

  - **DebateManager class:** Provides finished debates, which are stored in the database, as Debate objects, according to their debate IDs.

    Debate getDebate(int debateID): Takes debate ID as parameter and returns the unique Debate object that that ID belongs to.

    void storeDebate(Debate dbt): Takes a Debate object as parameter and records it to the database.

    Debate[ ] getUserDebates(int[ ] debateIDs): Takes an array of debate IDs as parameter and returns the array of Debate objects that the IDs correspond to.

  - **UserManager class:** Records the users that sign up to the database also provides the signing in. Checks whether a username is in use or not and provides User objects according to their usernames.

    boolean signUp(String username, String password): Takes username and password as parameters and tries to create a new User object in the database according to that. Returns if it managed it or not.

    User signIn(String username, String password): Takes username and password as parameters and checks whether they match with a User object from the database, if so returns it, otherwise returns null.

boolean checkUsername(String username): Takes a username as parameter and checks whether it matches a User object from database or not and returns corresponding boolean.

User getUserDetails(String username): Takes username as parameter and returns corresponding User object from the database.

- **Server models subsystem**
  - **Player class:** This class represents the user during a debate battle. Player objects contain the state of a user and the decisions they made during a battle.

Constants

public final int SIDE_NEGATIVE = -1

public final int SIDE_POSITIVE = 1

public final int SIDE_SPECTATOR = 0

public final int VOTE_YES = 2

public final int VOTE_NO = 3

Attributes

private int id

private String username

private Avatar selectedAvatar

private Title selectedTitle

private Frame selectedFrame

private int side:

private String[] arguments

private int vote

private int consecutiveGamesPlayed: Number of games that a player selected as debater will be recorded so that side selection algorithm could give priority people who haven't been able to debate for a while.

Methods

Setter/Getter methods.

- **Client bridge subsystem**

  - **BattleThread class:** This class is responsible for the dynamic structure of gameplay. Battles will be held in new thread in the server so that server can support more than one battle to be held at the same time.

Attributes

private Debate currentDebate

private PlayerHandler[] playerHandlers

private int remaningTimeInMinutes

Methods

public void run() : This is the methed that will run when the thread is executed.

private boolean startInitialArgumentStage() : This method will initialize first stage of the game. If one of the debaters leave before the stage is complete, it will return false and the debate will be closed.

private boolean startCounterArgumentStage(): This method will initialize second stage of the game. If one of the debaters leave before the stage is complete, it will return false and the debate will be closed.

private boolean startAnswersStage():This method will initialize third stage of the game. If one of the debaters

leave before the stage is complete, it will return false and the debate will be closed.

private boolean startConclusionStage(): This method will initialize last stage of the game. If one of the debaters leave before the stage is complete, it will return false and the debate will be closed.

private void generateNewDebate(: This method will generate a new debate when the current debate of battle is closed.

private void closeCurrentDebate(): This method will close the debate when conclusion stage is complete and when a debater leaves prior to finish.

private void designateSides(): This method will assign one player to positive side, one player to negative side and remaining two players to spectator side based on their side selections. This method will give priority to players who weren't selected as debater in the past debates so that every user will be able to play as debater.

- **PlayerHandler class:** This class is responsible for listening to players during the battle. It will also inform the player about the current status of the game.

Attributes

Player player

Methods

public String listenPlayer()

public void updateBattleStatus(Debate db, int stage): This method will send the debate object and the current state of the battle to client so that client can update its ui.

public void sendCurTimeToClient()

- **UserHandler class:** This class is responsible for handling the clients request. It will start listening to client when the user opens the app. It will use manager classes to retrieve data from database and will send the data retrieved data to user.

Constants

public final int REQUEST_INVENTORY = 0

public final int REQUEST_PLAYED_DEBATES =1

public final int REQUEST_PAST_DEBATES = 2

public final int REQUEST_INVENTORY=3

public final int REQUEST_BUYABLE_ITEMS = 4

Attributes

private ItemManager itemMan

private DebateManager debateMan

private UserManager userMan

Methods

public void answerUserRequest(int requestID): This method takes a request id as parameter and retrieves the necessary data using the manager. It then send the date to client.

- **Client models subsystem**
  - **User class**

private int id: Represents user ID

private string username: Represents user name

private int points: Represents the points that the user has in his own account

private Avatar selectedAvatar: This attribute will be used to keep the user's current avatar object.

private Title selectedTitle: This attribute will be used to keep the user's current title object.

private Frame selectedFrame: This attribute will be used to keep the user's current frame object.

private int[] pastDebateIDs: This attribute is for holding the ID's of the past debates in the game.

private Inventory myInventory: Inventory object for the record of the inventory of the user.

private int[] votedDebates: It shows the vote history of the user.

---

public void buyItem(int itemId): User can buy items by entering the ID of the item selected.

public void changeAvatar(int itemId): User can change his/her avatar by entering the ID of the item selected.

public void changeTitle(int itemId): User can change his/her title by entering the ID of the item selected.

public void changeFrame(int itemId): User can change his/her Frame by entering the ID of the item selected.

- **Item class:** Item class extends Expression, Avatar, Frame, Title classes.

  int itemID: Represents that each item has a unique ID.

  String itemName: Represents each item has a name.

- **Expression class**

  string image: Represents the image number or code that each expression type has.

- **Avatar class**

  int imageID: Represents the image id that each avatar type has.

- **Title class**

  string titleKey: Represents the type of the title

- **Inventory class**

  item[] myItems: Shows the item id's that the user has on his own inventory

  ---

  Avatar[] getAvatars(): Inventory can access the avatar ids of the user by using the item object. The function is used to seperate avatars from other items.

  Expression[] getExpressions(): Inventory can access the expression ids of the user by using the item object. The function is used to seperate expressions from other items.

Frame[] getFrames(): Inventory can access the frame styles that the user has, by using the item object. The function is used to seperate frames from other items.

Title getTitles(): Inventory can access the titles that the user has, by using the item object. The function is used to seperate titles from other items.

- **Idea class**

  int ideaID: Each idea object has a unique id.

  string statement: Each idea has statements.

  int category: Each idea has a category like philosophy, military, history or health, etc.

- **ServerBridge class:** Represents the client side of the data transfer bridge between the server and the client. Sends requests to the server via predefined unique request IDs and takes data in return. It can also forward data to user interface classes.

  String[ ] request(int requestID, User user): Takes a User object that defines from whom the request come and a request ID as parameters. Each request ID correspond to a unique action in server and a proper reply is provided by the server according to the request.

  void forwardDataToUI(int UIID, String data): Takes the data to be transferred and the ID of the user interface to which the data will be transferred and forwards the data.

- **Controller classes subsytem**

  - **RegisterActivity class**: Displays one text box for the username, one for the password and one for the confirmation of the password, also listens the actions of register page.

    boolean receiveAndUpdateUI(Sting data): Takes a data string and updates the UI. Returns if it updated the UI or not.

  - **LoginActivity class**: Displays one text box for the username and one for the password, also listens the actions of login page.

boolean receiveAndUpdateUI(Sting data): Takes a data string and updates the UI. Returns if it updated the UI or not.

- **MainFragment class**: Displays the selected items, points and last debates of the user, also listens the actions of the main page.

  boolean receiveAndUpdateUI(Sting data): Takes a data string and updates the UI. Returns if it updated the UI or not.

- **InventoryFragment class**: Displays the items of the user and listens the actions of the inventory page.

  boolean receiveAndUpdateUI(Sting data): Takes a data string and updates the UI. Returns if it updated the UI or not.

- **MarketFragment**: Displays all items and listens the actions of the market page.

  boolean receiveAndUpdateUI(Sting data): Takes a data string and updates the UI. Returns if it updated the UI or not.

- **BrowseBattleFragment**: Displays a screen until 4 available users exist on this page.

  boolean receiveAndUpdateUI(Sting data): Takes a data string and updates the UI. Returns if it updated the UI or not.

- **GameplayActivity:** Displays 4 players in the battle; the selected items, expressions and sides of them. Listens to the arguments and display arguments from the past stages. Listens to the final voting,

  boolean receiveAndUpdateUI(Sting data): Takes a data string and updates the UI. Returns if it updated the UI or not.

- **FinishedDebateFragment:** Displays all the finished debates by the time that each debate was voted, the category and the topic of each. Listens to the specific debate selection. Displays options for replaying the debate or seeing conclusion directly. Listens to the choice. Displays the debate accordingly and listens to the vote.

boolean receiveAndUpdateUI(Sting data): Takes a data string and updates the UI. Returns if it updated the UI or not.

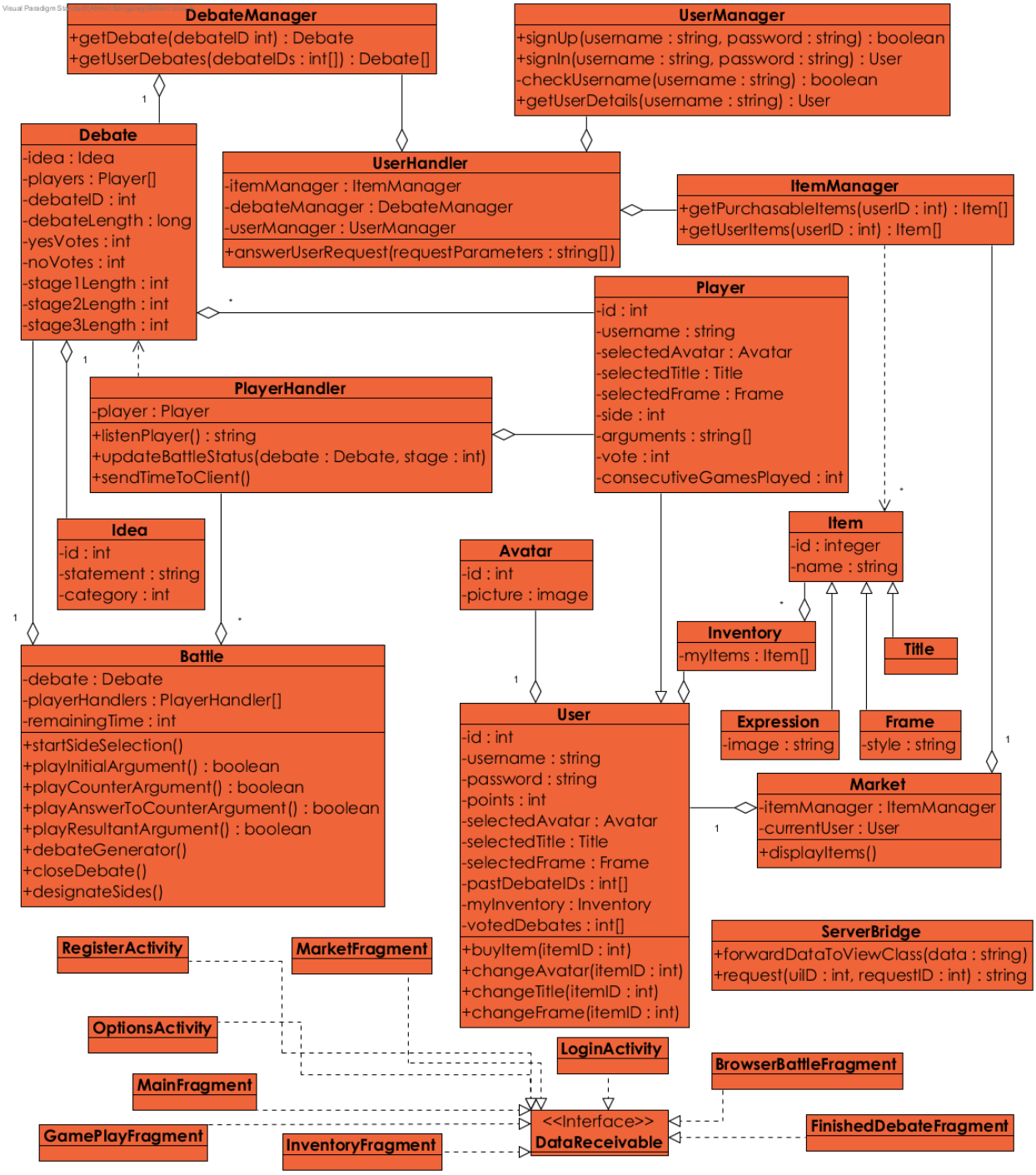
- **OptionsActivity:** Display "Sound on/off", "Set text font" and logout options and listens to the actions of this page.

  boolean receiveAndUpdateUI(Sting data): Takes a data string and updates the UI. Returns if it updated the UI or not.

- **Views subsytem**

This system contains xml files that will be attached to controller classes.

LOW-LEVEL DESIGN



- **Object Design Trade-offs**

  - **Security vs response time**

    Because all the objects will be kept in database, not in the client, there will be some extension at the response time exchange for security impromevents.

- **Design Pattern**

  Design pattern is a structural design pattern which proposes that developers can easily manage a subsystem from a façade class since the communication between outside of this subsystem is performed only by this class.

  Our whole system consists of 2 systems. First one is "Client" and the other one is "Server". Server consists of DBManager subsystem, ServerModels subsystem and ClientBridge Subsystem. Client consists of ClientModels, ControllerClasses and views subsystems. The façade class of ClientModels subsystem is ServerBridge class because it handles the operations of entity objects of our system. The façade class of ClientBridge subsystem is BattleThread class.

GLOSSARY

Idea: A controversial statement which will divide users to 2 sides.

Avatar: A small image which will be shown in the debate screen with the nickname of user.

Frame: A border that will be around user's nickname.

Title: A title which will be shown before user's nickname in his profile and debate screen.

Expression: A certain expression such as "WOW" or "GG" which users will be able to use during debates.

REFERENCES

[1]"The Premier Online Debate Website | Debate.org", Debate.org, 2018. [Online]. Available: http://www.debate.org/. [Accessed: 17- Feb- 2018].

[2]B. Bruegge and A. Dutoit, *Object-oriented software engineering*. Harlow, Essex: Pearson, 2014.