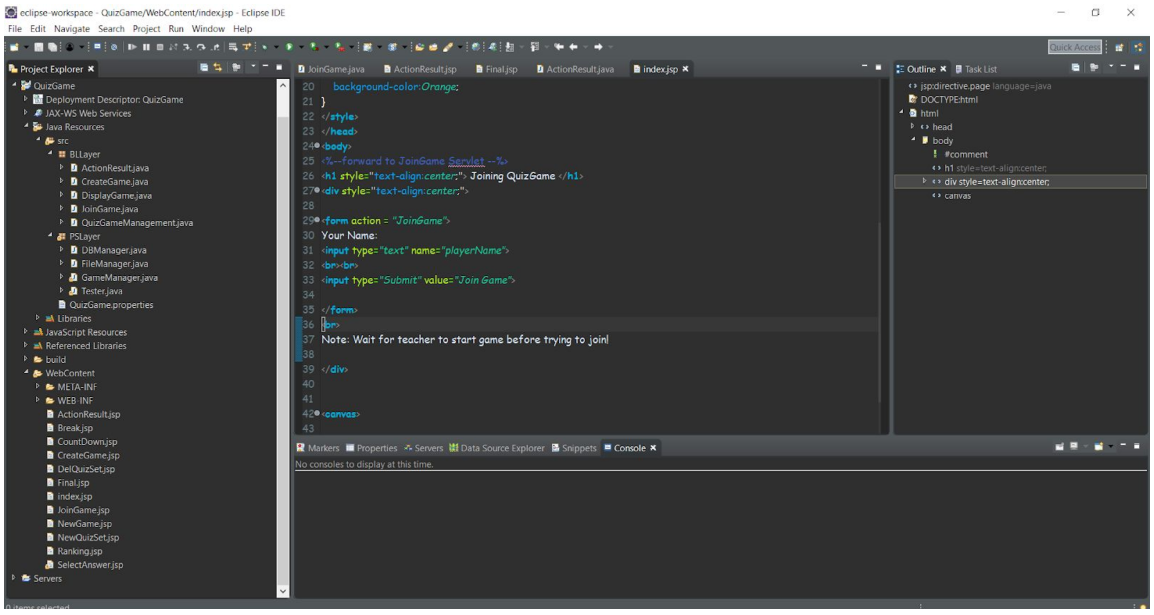
Criterion C: Development

(1021 words)

<u>Main Third Party Tools</u>

Eclipse IDE for Java , Enterprise Edition: This integrated development environment allows simple setup and organization of a web application to run on the local server. It has great support for java, JSP, javascript, among other files types used.

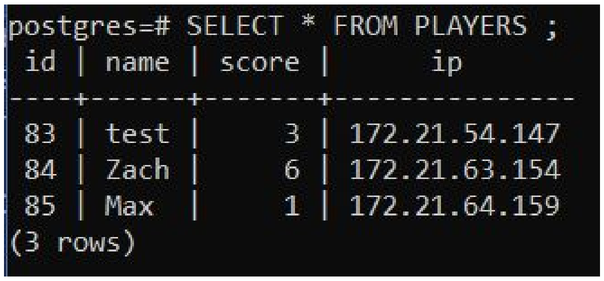


```
QuizGame
  Deployment Descriptor: QuizGame
  JAX-WS Web Services
  Java Resources
    src
      BLLayer
        ActionResult.java
        CreateGame.java
        DisplayGame.java
        JoinGame.java
        QuizGameManagement.java
      PSLayer
        DBManager.java
        FileManager.java
        GameManager.java
        Tester.java
      QuizGame.properties
    Libraries
  JavaScript Resources
  Referenced Libraries
  build
  WebContent
    META-INF
    WEB-INF
    ActionResult.jsp
    Break.jsp
    CountDown.jsp
    CreateGame.jsp
    DelQuizSet.jsp
    Final.jsp
    index.jsp
    JoinGame.jsp
    NewGame.jsp
    NewQuizSet.jsp
    Ranking.jsp
    SelectAnswer.jsp
Servers
```

```
  background-color:Orange;
}
</style>
</head>
<body>
<%--forward to JoinGame Servlet --%>
<h1 style="text-align:center;"> Joining QuizGame </h1>
<div style="text-align:center;">

<form action = "JoinGame">
Your Name:
<input type="text" name="playerName">
<br><br>
<input type="Submit" value="Join Game">

</form>
<br>
Note: Wait for teacher to start game before trying to join!

</div>


<canvas>

```

```
No consoles to display at this time.
```

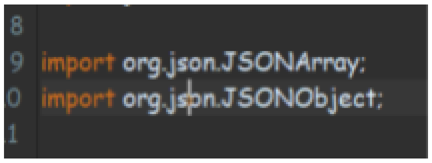
PostgreSQL and JDBC Drivers: A PostgreSQL database allows for player management. JDBC drivers allowed plain old java objects (POJO) used in the persistence layer in the application to communicate directly with PostgreSQL.



```
postgres=# SELECT * FROM PLAYERS ;
 id | name | score |       ip
----+------+-------+---------------
 83 | test |     3 | 172.21.54.147
 84 | Zach |     6 | 172.21.63.154
 85 | Max  |     1 | 172.21.64.159
(3 rows)
```

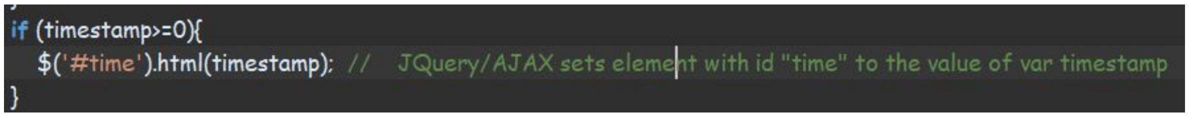
Main Third Party Libraries

Org.json: Essential import to allow for creation, modification, deletion of JSON objects and arrays from files.



```java
import org.json.JSONArray;
import org.json.JSONObject;
```

Jquery.ajax: Allows for easy client-side updating of essential elements such as countdowns without reloading the webpage.

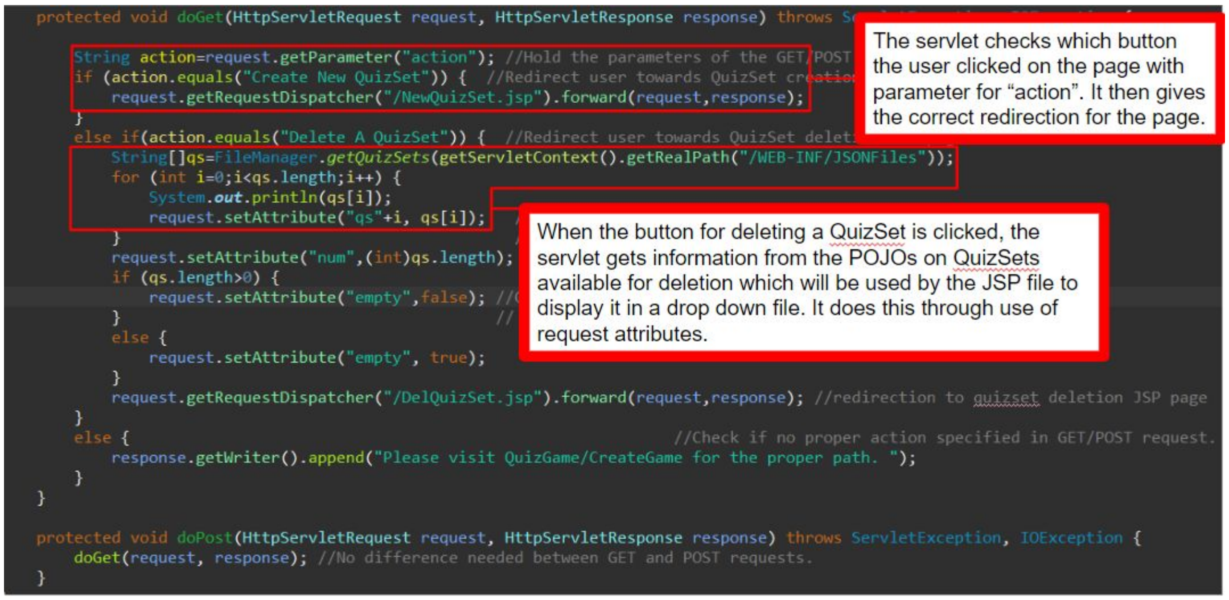


```javascript
if (timestamp>=0){
   $('#time').html(timestamp); //   JQuery/AJAX sets element with id "time" to the value of var timestamp
}
```

Java Servlets:

Basic Interaction: Java servlets enable student-server web interaction and teacher-server web interaction.

Snippet: QuizGameManagement.java



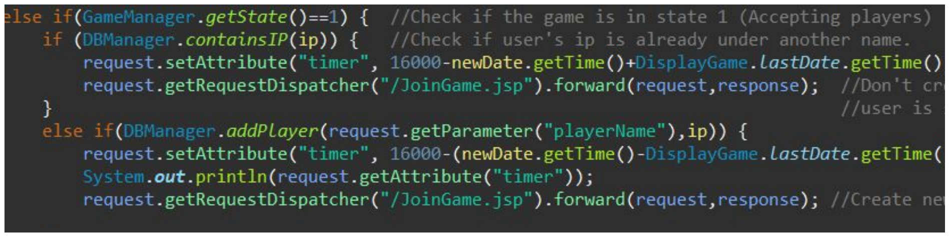
```java
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws Se

        String action=request.getParameter("action"); //Hold the parameters of the GET/POST
        if (action.equals("Create New QuizSet")) {  //Redirect user towards QuizSet creation
            request.getRequestDispatcher("/NewQuizSet.jsp").forward(request,response);
        }
        else if(action.equals("Delete A QuizSet")) {  //Redirect user towards QuizSet delet
            String[]qs=FileManager.getQuizSets(getServletContext().getRealPath("/WEB-INF/JSONFiles"));
            for (int i=0;i<qs.length;i++) {
                System.out.println(qs[i]);
                request.setAttribute("qs"+i, qs[i]);
            }
            request.setAttribute("num",(int)qs.length);
            if (qs.length>0) {
                request.setAttribute("empty",false); //
            }                                        //
            else {
                request.setAttribute("empty", true);
            }
            request.getRequestDispatcher("/DelQuizSet.jsp").forward(request,response); //redirection to quizset deletion JSP page
        }
        else {                                              //Check if no proper action specified in GET/POST request.
            response.getWriter().append("Please visit QuizGame/CreateGame for the proper path. ");
        }
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response); //No difference needed between GET and POST requests.
    }
```

The servlet checks which button the user clicked on the page with parameter for "action". It then gives the correct redirection for the page.

When the button for deleting a QuizSet is clicked, the servlet gets information from the POJOs on QuizSets available for deletion which will be used by the JSP file to display it in a drop down file. It does this through use of request attributes.

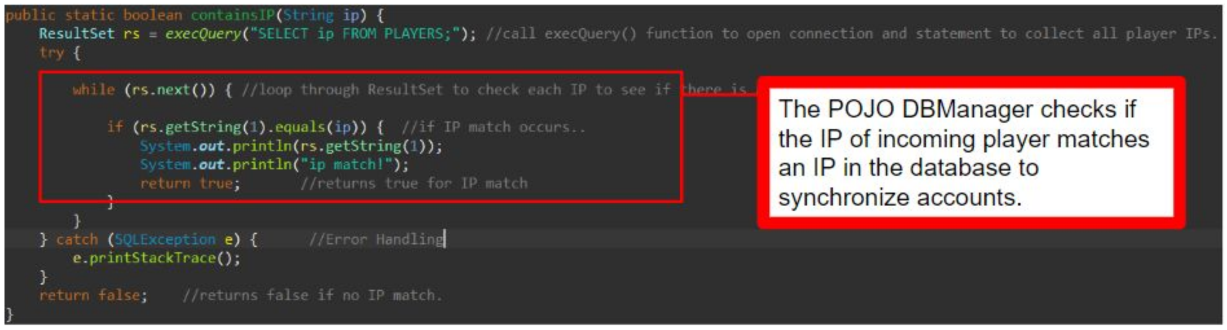
GET and POST request handling is done by this QuizGameManagement.java servlet to decide which JSP page to return to the user in the response.

User Management: In order to prevent users from making multiple game accounts to create an unfair advantage or help users that accidently disconnected, player internet protocol address was tracked and checked through servlets making use of the persistence layer class, DBManager.



```
else if(GameManager.getState()==1) {  //Check if the game is in state 1 (Accepting players)
    if (DBManager.containsIP(ip)) {   //Check if user's ip is already under another name.
        request.setAttribute("timer", 16000-newDate.getTime()+DisplayGame.lastDate.getTime()
        request.getRequestDispatcher("/JoinGame.jsp").forward(request,response);  //Don't cr
    }                                                                             //user is
    else if(DBManager.addPlayer(request.getParameter("playerName"),ip)) {
        request.setAttribute("timer", 16000-(newDate.getTime()-DisplayGame.lastDate.getTime(
        System.out.println(request.getAttribute("timer"));
        request.getRequestDispatcher("/JoinGame.jsp").forward(request,response); //Create ne
```

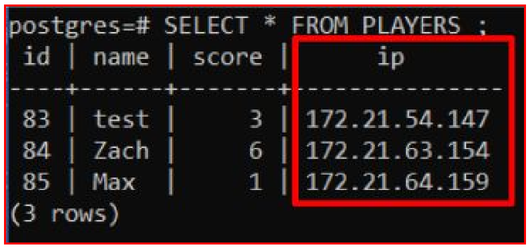
Business Logic needs to know if IP in GET request is in database



```
public static boolean containsIP(String ip) {
    ResultSet rs = execQuery("SELECT ip FROM PLAYERS;"); //call execQuery() function to open connection and statement to collect all player IPs.
    try {

        while (rs.next()) { //loop through ResultSet to check each IP to see if there is

            if (rs.getString(1).equals(ip)) {  //if IP match occurs..
                System.out.println(rs.getString(1));
                System.out.println("ip match!");
                return true;        //returns true for IP match
            }
        }
    } catch (SQLException e) {        //Error Handling
        e.printStackTrace();
    }
    return false;    //returns false if no IP match.
}
```

The POJO DBManager checks if the IP of incoming player matches an IP in the database to synchronize accounts.

Persistence Layer code used to make direct connection to DB to check if IP is in DB.

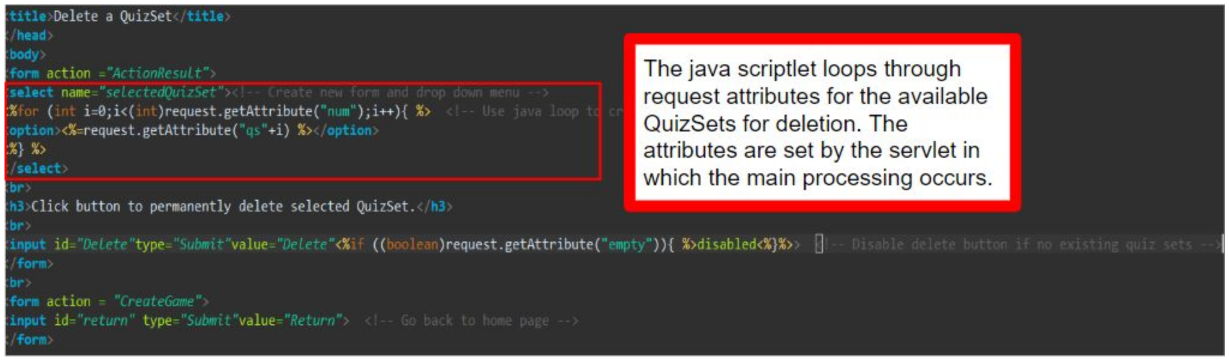


```
postgres=# SELECT * FROM PLAYERS ;
 id | name | score |      ip
----+------+-------+---------------
 83 | test |     3 | 172.21.54.147
 84 | Zach |     6 | 172.21.63.154
 85 | Max  |     1 | 172.21.64.159
(3 rows)
```

JSP Front-End

Scriptlets: Java Server Pages (JSPs) allow for the use of java code(scriptlets) alongside regular html and javascript code. Scriptlets are generally looked down upon for complex dynamic web applications because it is hard to read and can thus make the code hard to maintain. However, for my project, they were useful in small amounts for certain tasks such as modifying the web page based on servlet information through attributes.

Snippet: DelQuizSet.jsp

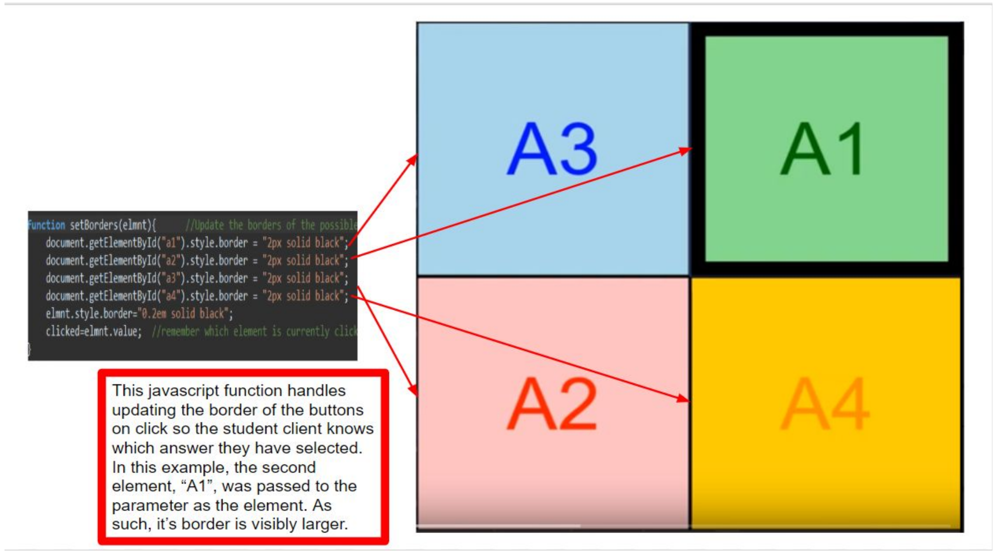


```
title>Delete a QuizSet</title>
/head>
body>
form action ="ActionResult">
select name="selectedQuizSet"><!-- Create new form and drop down menu -->
%for (int i=0;i<(int)request.getAttribute("num");i++){ %>  <!-- Use java loop to cr
option><%=request.getAttribute("qs"+i) %></option>
%} %>
/select>
br>
h3>Click button to permanently delete selected QuizSet.</h3>
br>
input id="Delete"type="Submit"value="Delete"<%if ((boolean)request.getAttribute("empty")){ %>disabled<%}%>> <!-- Disable delete button if no existing quiz sets -->
/form>
br>
form action = "CreateGame">
input id="return" type="Submit"value="Return">  <!-- Go back to home page -->
/form>
```

The java scriptlet loops through request attributes for the available QuizSets for deletion. The attributes are set by the servlet in which the main processing occurs.

JavaScript: Some javascript was also used in the JSP pages for tasks that require updating the front-end without having to reload the page. Jquery.ajax was also used here to make code for the countdowns on various pages.

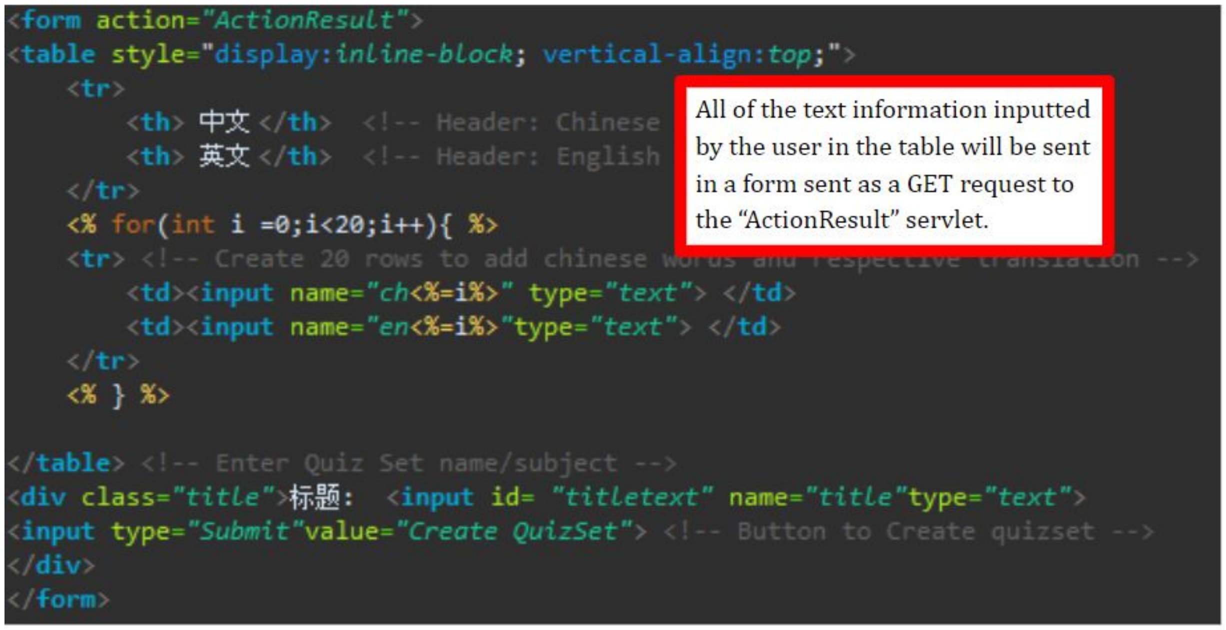
Snippet: SelectAnswer.jsp



```
function setBorders(elmnt){      //Update the borders of the possible
    document.getElementById("a1").style.border = "2px solid black";
    document.getElementById("a2").style.border = "2px solid black";
    document.getElementById("a3").style.border = "2px solid black";
    document.getElementById("a4").style.border = "2px solid black";
    elmnt.style.border="0.2em solid black";
    clicked=elmnt.value;  //remember which element is currently click
}
```

This javascript function handles updating the border of the buttons on click so the student client knows which answer they have selected. In this example, the second element, “A1”, was passed to the parameter as the element. As such, it’s border is visibly larger.

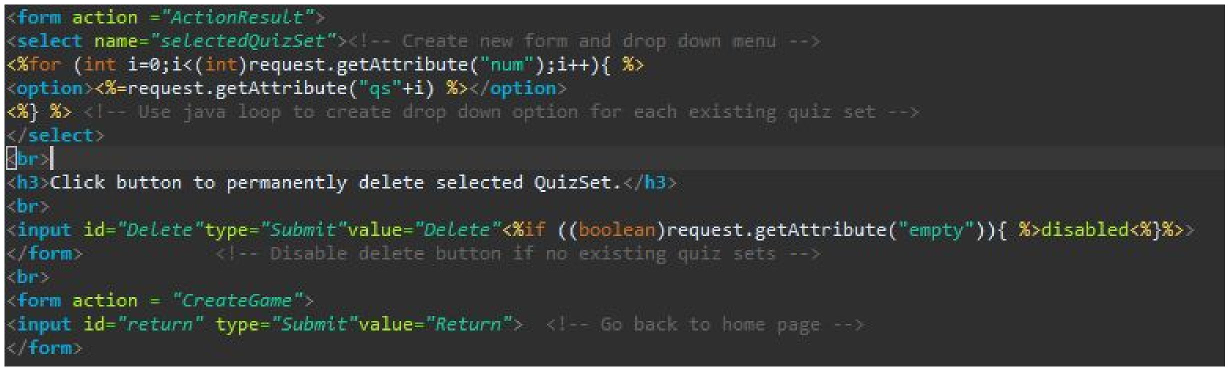
Web Forms: Forms needed to be used to allow for users to send data to be processed by the servlet as GET or POST requests.



```
<form action="ActionResult">
<table style="display:inline-block; vertical-align:top;">
    <tr>
        <th> 中文 </th>  <!-- Header: Chinese
        <th> 英文 </th>  <!-- Header: English
    </tr>
    <% for(int i =0;i<20;i++){ %>
    <tr> <!-- Create 20 rows to add chinese words and respective translation -->
        <td><input name="ch<%=i%>" type="text"> </td>
        <td><input name="en<%=i%>"type="text"> </td>
    </tr>
    <% } %>

</table> <!-- Enter Quiz Set name/subject -->
<div class="title">标题:  <input id= "titletext" name="title"type="text">
<input type="Submit"value="Create QuizSet"> <!-- Button to Create quizset -->
</div>
</form>
```

All of the text information inputted by the user in the table will be sent in a form sent as a GET request to the "ActionResult" servlet.

Featured here are text input elements in the form to input quiz set information and a submit button to create the quiz set. Other featured form elements elsewhere in the application are drop-down selection menus and buttons. This is necessary for the client input into the system for which a response can be generated.

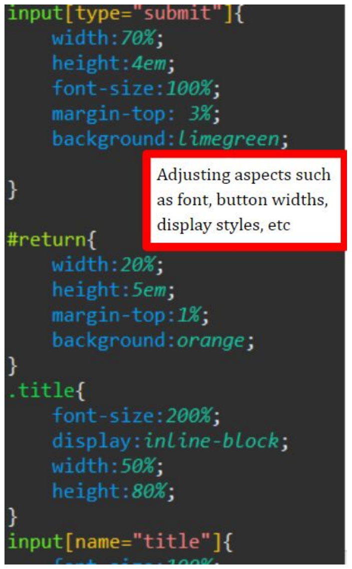


```
<form action ="ActionResult">
<select name="selectedQuizSet"><!-- Create new form and drop down menu -->
<%for (int i=0;i<(int)request.getAttribute("num");i++){ %>
<option><%=request.getAttribute("qs"+i) %></option>
<%} %> <!-- Use java loop to create drop down option for each existing quiz set -->
</select>
<br>
<h3>Click button to permanently delete selected QuizSet.</h3>
<br>
<input id="Delete"type="Submit"value="Delete"<%if ((boolean)request.getAttribute("empty")){ %>disabled<%}%>>
</form>            <!-- Disable delete button if no existing quiz sets -->
<br>
<form action = "CreateGame">
<input id="return" type="Submit"value="Return">  <!-- Go back to home page -->
</form>
```

Quiz set deletion form featuring drop-down menu of available quiz sets to delete.

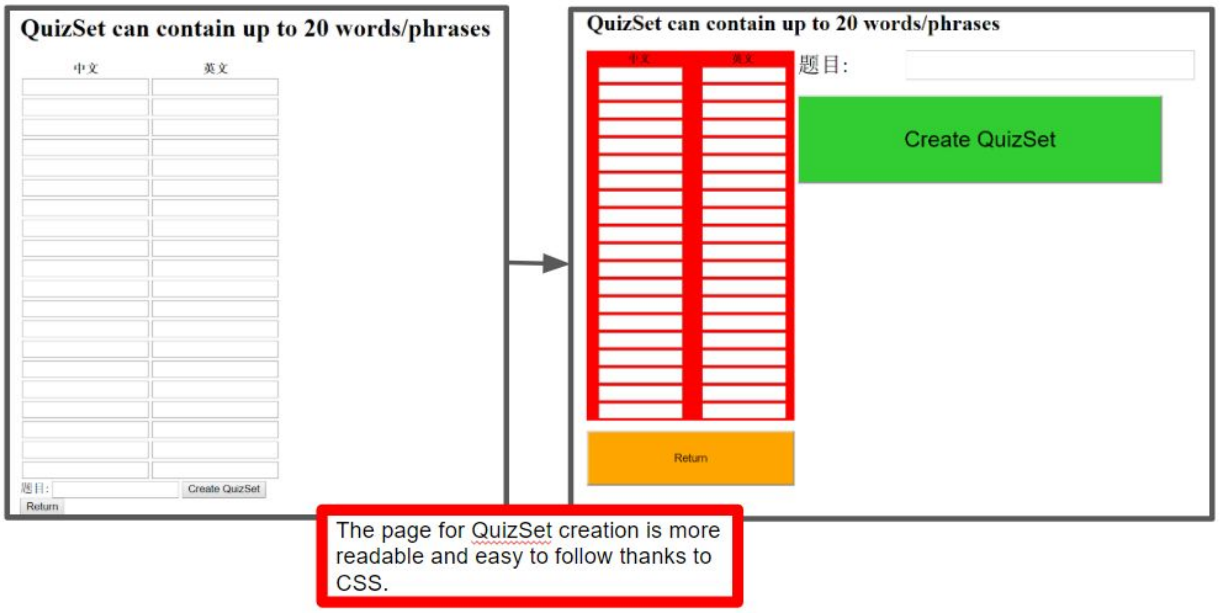
CSS Styling: To make the project appear visually appealing, Cascading Style Sheets (CSS) styles were implemented in all JSP pages for formatting and user accessibility. Without the CSS styles, the pages may look extremely confusing and difficult to navigate without specific instructions.

Snippet: NewQuizSet.jsp



```
input[type="submit"]{
    width:70%;
    height:4em;
    font-size:100%;
    margin-top: 3%;
    background:limegreen;

}

#return{
    width:20%;
    height:5em;
    margin-top:1%;
    background:orange;
}
.title{
    font-size:200%;
    display:inline-block;
    width:50%;
    height:80%;
}
input[name="title"]{
```

Adjusting aspects such as font, button widths, display styles, etc

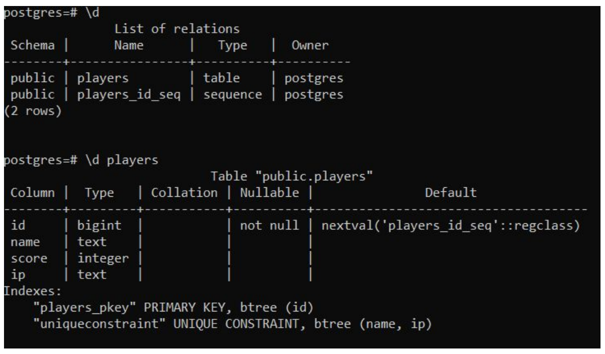


QuizSet can contain up to 20 words/phrases
中文
英文
题目:
Create QuizSet
Return
QuizSet can contain up to 20 words/phrases
中文
英文
题目:
Create QuizSet
Return

The page for QuizSet creation is more readable and easy to follow thanks to CSS.

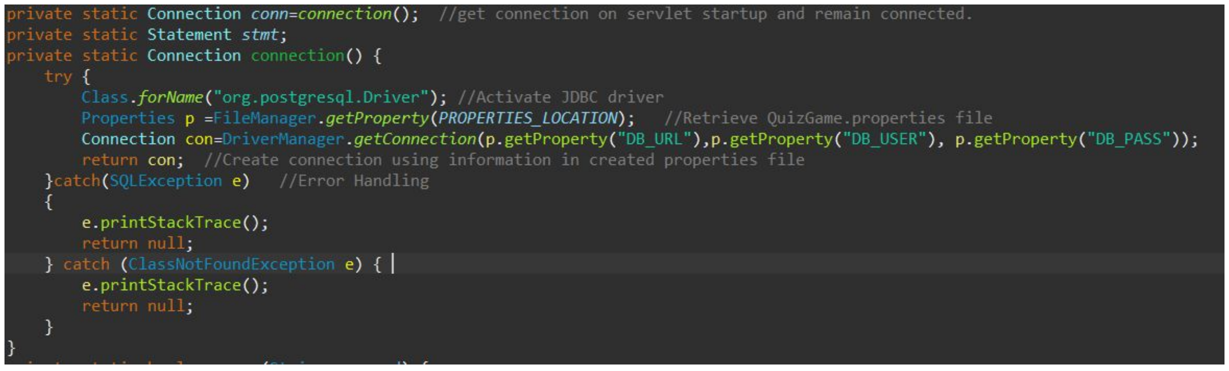
Database Management:

PostgreSQL: For the database technology, PostgreSQL was used as it allowed for simple creations of data tables easily modified by Java objects JDBC drivers. Four columns are used: id, name, score, and ip. Player ID automatically increments itself on player creation through the players_id_seq sequence. Name and IP have “UNIQUE” constraints to ensure there are no duplicate accounts from one device and no two players with same name to avoid confusion.



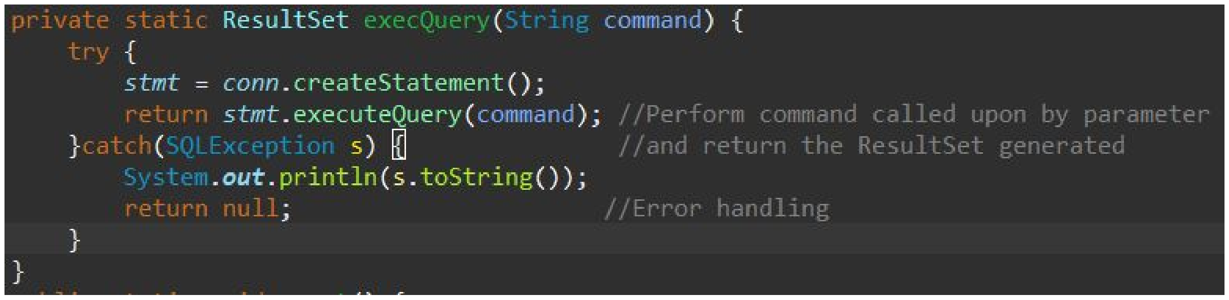
```
postgres=# \d
               List of relations
 Schema |      Name      |   Type   |  Owner
--------+----------------+----------+----------
 public | players        | table    | postgres
 public | players_id_seq | sequence | postgres
(2 rows)


postgres=# \d players
                             Table "public.players"
 Column |  Type   | Collation | Nullable |               Default
--------+---------+-----------+----------+-------------------------------------
 id     | bigint  |           | not null | nextval('players_id_seq'::regclass)
 name   | text    |           |          |
 score  | integer |           |          |
 ip     | text    |           |          |
Indexes:
    "players_pkey" PRIMARY KEY, btree (id)
    "uniqueconstraint" UNIQUE CONSTRAINT, btree (name, ip)
```

Connecting to Database: JDBC drivers had to be installed and called upon to enable connection to the database from DBManagement.java file. It allowed for all forms of operations to be performed on the database based on game actions.



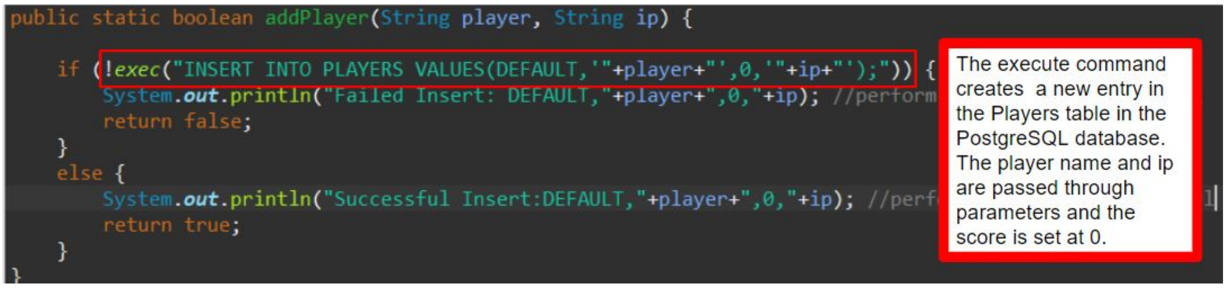
```java
private static Connection conn=connection();  //get connection on servlet startup and remain connected.
private static Statement stmt;
private static Connection connection() {
    try {
        Class.forName("org.postgresql.Driver"); //Activate JDBC driver
        Properties p =FileManager.getProperty(PROPERTIES_LOCATION);   //Retrieve QuizGame.properties file
        Connection con=DriverManager.getConnection(p.getProperty("DB_URL"),p.getProperty("DB_USER"), p.getProperty("DB_PASS"));
        return con;  //Create connection using information in created properties file
    }catch(SQLException e)   //Error Handling
    {
        e.printStackTrace();
        return null;
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
        return null;
    }
}
```

Executing Commands: Functions execQuery() and exec() were created to serve as foundational functions for use by other higher level DB management functions.



```java
private static ResultSet execQuery(String command) {
    try {
        stmt = conn.createStatement();
        return stmt.executeQuery(command); //Perform command called upon by parameter
    }catch(SQLException s) {             //and return the ResultSet generated
        System.out.println(s.toString());
        return null;                    //Error handling
    }
}
```

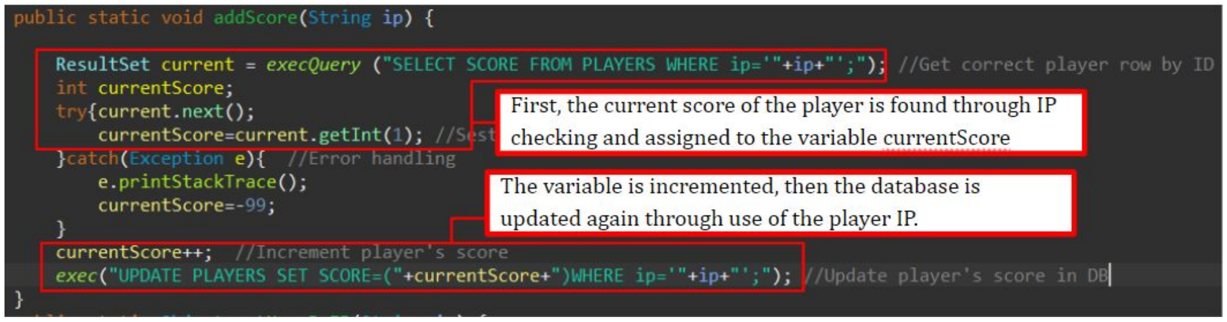
Adding Players: This function making use of the exec() function allows for addition of players into the database with the SQL statement sent as a string parameter.



```
public static boolean addPlayer(String player, String ip) {

    if (!exec("INSERT INTO PLAYERS VALUES(DEFAULT,'"+player+"',0,'"+ip+"');")) {
        System.out.println("Failed Insert: DEFAULT,"+player+",0,"+ip); //perform
        return false;
    }
    else {
        System.out.println("Successful Insert:DEFAULT,"+player+",0,"+ip); //perf
        return true;
    }
}
```

The execute command creates a new entry in the Players table in the PostgreSQL database. The player name and ip are passed through parameters and the score is set at 0.

Updating Players: This function allows for updating the player score after they selected a correct answer.



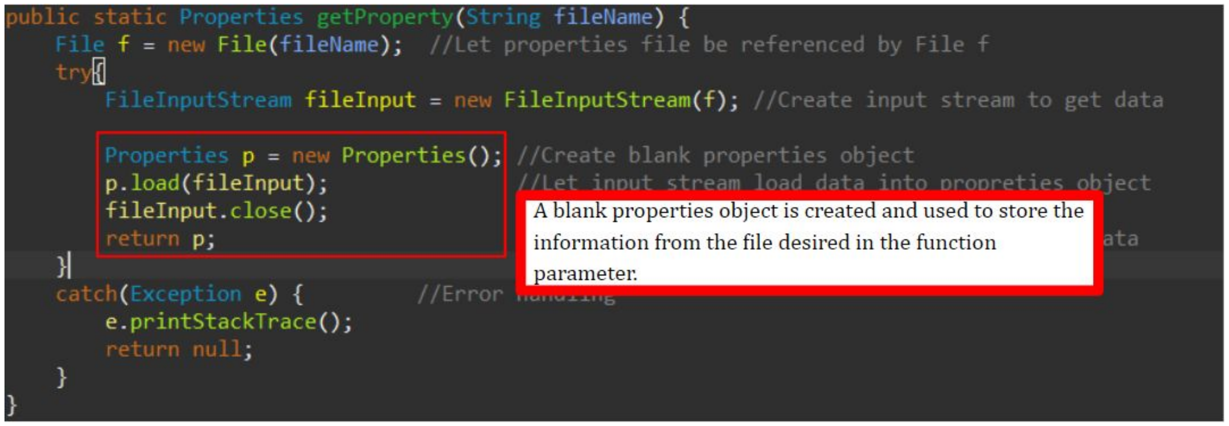
```
public static void addScore(String ip) {

    ResultSet current = execQuery ("SELECT SCORE FROM PLAYERS WHERE ip='"+ip+"';"); //Get correct player row by ID
    int currentScore;
    try{current.next();
        currentScore=current.getInt(1); //Sest
    }catch(Exception e){  //Error handling
        e.printStackTrace();
        currentScore=-99;
    }
    currentScore++;  //Increment player's score
    exec("UPDATE PLAYERS SET SCORE=("+currentScore+")WHERE ip='"+ip+"';"); //Update player's score in DB
}
```

First, the current score of the player is found through IP checking and assigned to the variable currentScore

The variable is incremented, then the database is updated again through use of the player IP.

File Management

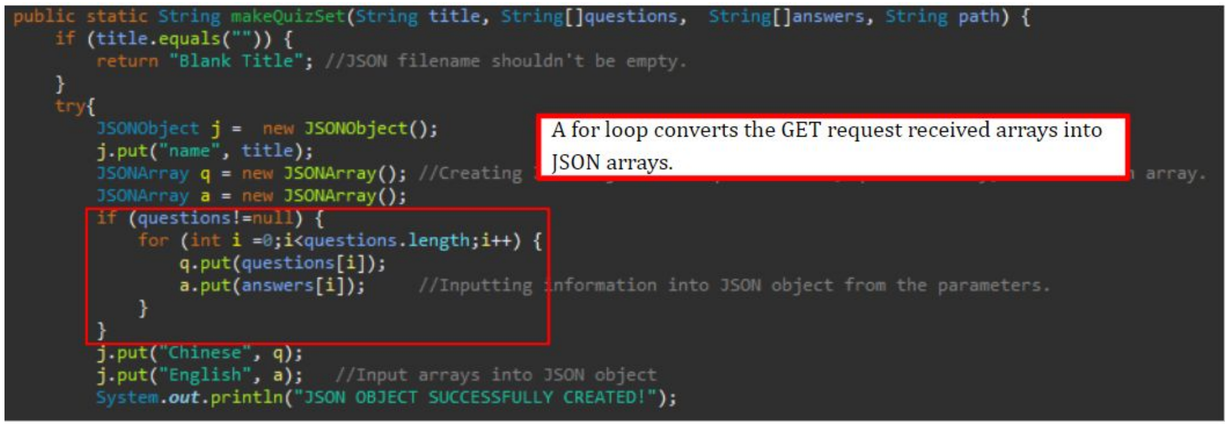
Retrieving Properties Files: Special information is saved in the QuizGame.properties file such as DB connection information and location of JSON objects being saved to. To retrieve the properties to use the values in other areas of the code, the function getProperty(String fileName) was created in FileManager.java.



```java
public static Properties getProperty(String fileName) {
    File f = new File(fileName);  //Let properties file be referenced by File f
    try{
        FileInputStream fileInput = new FileInputStream(f); //Create input stream to get data

        Properties p = new Properties(); //Create blank properties object
        p.load(fileInput);               //Let input stream load data into propreties object
        fileInput.close();
        return p;                                                                    ata
    }
    catch(Exception e) {          //Error
        e.printStackTrace();
        return null;
    }
}
```

A blank properties object is created and used to store the information from the file desired in the function parameter.

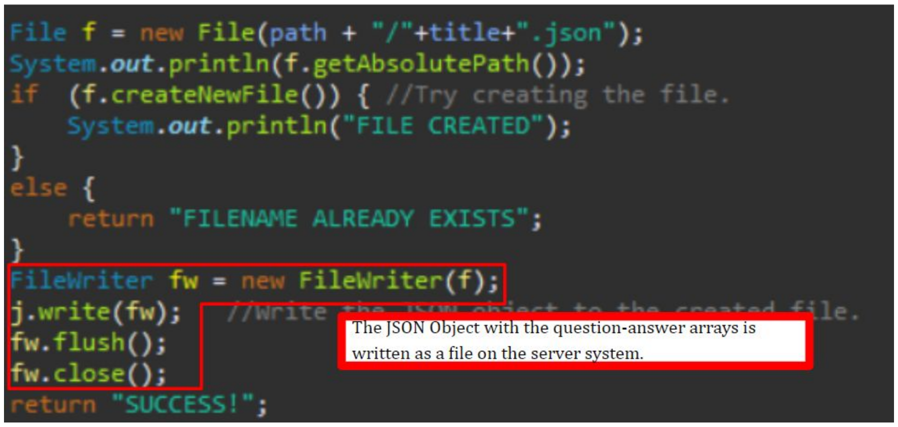
QuizSet Storage: Quiz Sets needed to be saved as stated by success criterion 4. To do so, I used a JSON object. This allows me to straightforwardly store the quiz set data (questions, answers, title).



```java
public static String makeQuizSet(String title, String[]questions,  String[]answers, String path) {
    if (title.equals("")) {
        return "Blank Title"; //JSON filename shouldn't be empty.
    }
    try{
        JSONObject j =  new JSONObject();
        j.put("name", title);
        JSONArray q = new JSONArray(); //Creating                                       array.
        JSONArray a = new JSONArray();
        if (questions!=null) {
            for (int i =0;i<questions.length;i++) {
                q.put(questions[i]);
                a.put(answers[i]);     //Inputting information into JSON object from the parameters.
            }
        }
        j.put("Chinese", q);
        j.put("English", a);   //Input arrays into JSON object
        System.out.println("JSON OBJECT SUCCESSFULLY CREATED!");
```

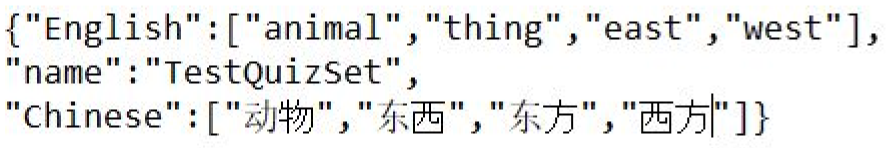
A for loop converts the GET request received arrays into JSON arrays.

Then, I had to write the JSON object to a file saved on the system so the file is not lost even if the server goes down for any reason.



```java
File f = new File(path + "/"+title+".json");
System.out.println(f.getAbsolutePath());
if  (f.createNewFile()) { //Try creating the file.
    System.out.println("FILE CREATED");
}
else {
    return "FILENAME ALREADY EXISTS";
}
FileWriter fw = new FileWriter(f);
j.write(fw);    //Write the JSON object to the created file.
fw.flush();
fw.close();
return "SUCCESS!";
```

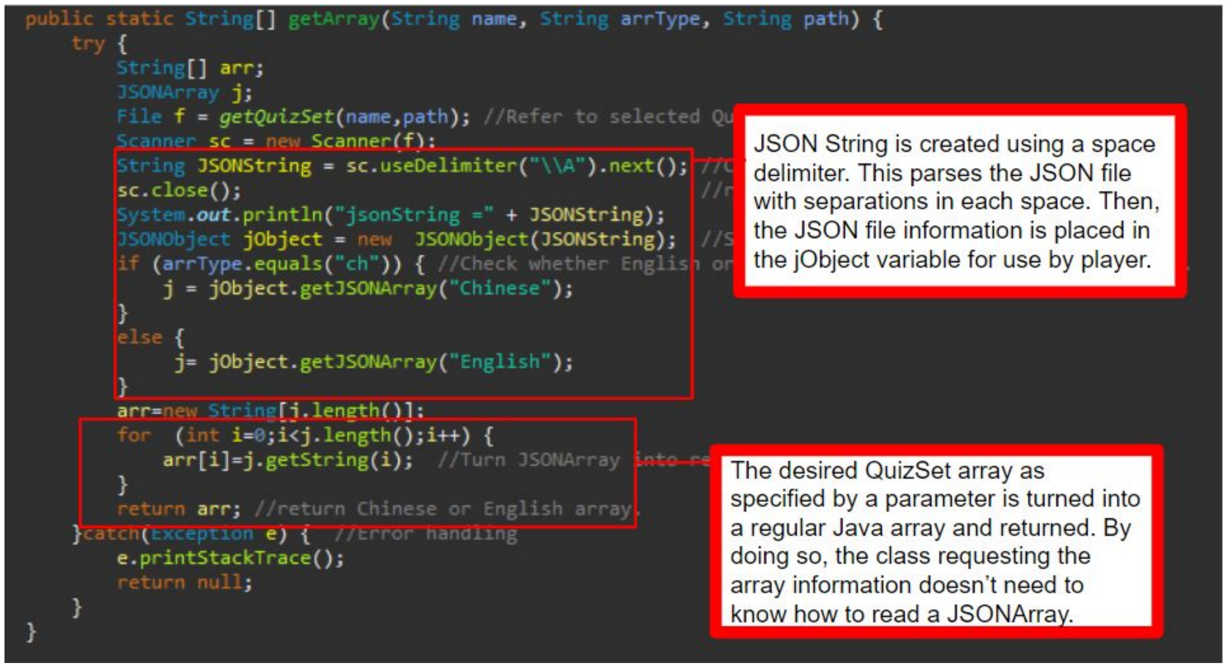
The JSON Object with the question-answer arrays is written as a file on the server system.



```
{"English":["animal","thing","east","west"],
"name":"TestQuizSet",
"Chinese":["动物","东西","东方","西方"]}
```

Example of FileManager.java-created JSON file.

Parsing JSON Objects: For certain parts of the application, I needed to access certain parts of a quiz set JSON file, but not others. To do so, I made a JSON retrieval function to get certain information using a delimiter to be able to turn the file's information into a readable JSON object.



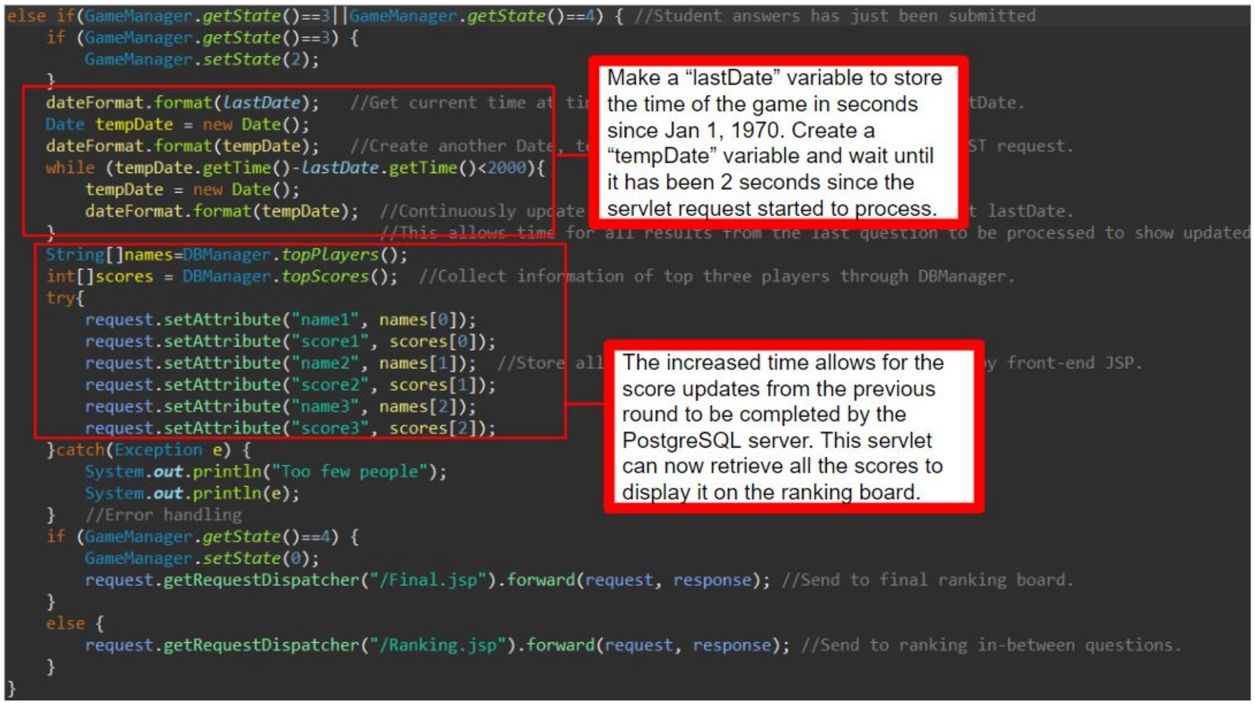
```java
public static String[] getArray(String name, String arrType, String path) {
    try {
        String[] arr;
        JSONArray j;
        File f = getQuizSet(name,path); //Refer to selected Qu
        Scanner sc = new Scanner(f);
        String JSONString = sc.useDelimiter("\\A").next(); //C
        sc.close();                                         //r
        System.out.println("jsonString =" + JSONString);
        JSONObject jObject = new  JSONObject(JSONString);  //S
        if (arrType.equals("ch")) { //Check whether English or
            j = jObject.getJSONArray("Chinese");
        }
        else {
             j= jObject.getJSONArray("English");
        }
        arr=new String[j.length()];
        for  (int i=0;i<j.length();i++) {
            arr[i]=j.getString(i);  //Turn JSONArray into re
        }
        return arr; //return Chinese or English array.
    }catch(Exception e) {  //Error handling
        e.printStackTrace();
        return null;
    }
}
```

JSON String is created using a space delimiter. This parses the JSON file with separations in each space. Then, the JSON file information is placed in the jObject variable for use by player.

The desired QuizSet array as specified by a parameter is turned into a regular Java array and returned. By doing so, the class requesting the array information doesn't need to know how to read a JSONArray.

Synchronization:

One significant challenge faced while programming is keeping student players in sync with changes on teacher client. For example, when the first quiz question is asked by the teacher client, the student clients should automatically move on to the page where they select an answer to the question in a quick manner. To do so, I made use of the Date class.



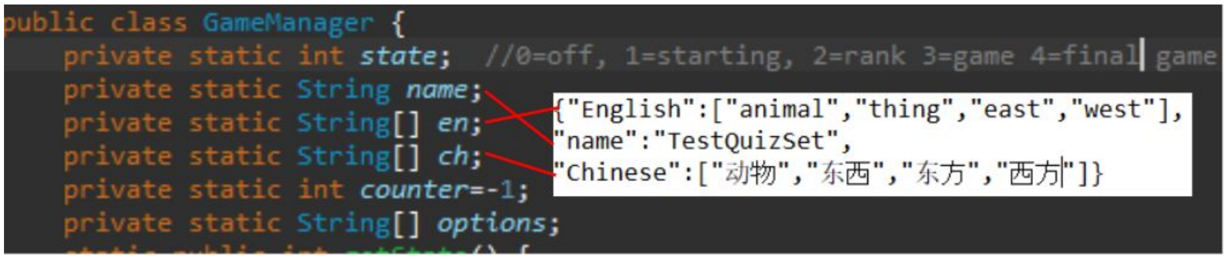
```
else if(GameManager.getState()==3||GameManager.getState()==4) { //Student answers has just been submitted
    if (GameManager.getState()==3) {
        GameManager.setState(2);
    }
    dateFormat.format(lastDate);   //Get current time at ti                                tDate.
    Date tempDate = new Date();
    dateFormat.format(tempDate);   //Create another Date, t                                ST request.
    while (tempDate.getTime()-lastDate.getTime()<2000){
        tempDate = new Date();
        dateFormat.format(tempDate);  //Continuously update                                t lastDate.
    }                                 //This allows time for all results from the last question to be processed to show updated
    String[]names=DBManager.topPlayers();
    int[]scores = DBManager.topScores();  //Collect information of top three players through DBManager.
    try{
        request.setAttribute("name1", names[0]);
        request.setAttribute("score1", scores[0]);
        request.setAttribute("name2", names[1]);  //Store all                                 y front-end JSP.
        request.setAttribute("score2", scores[1]);
        request.setAttribute("name3", names[2]);
        request.setAttribute("score3", scores[2]);
    }catch(Exception e) {
        System.out.println("Too few people");
        System.out.println(e);
    }   //Error handling
    if (GameManager.getState()==4) {
        GameManager.setState(0);
        request.getRequestDispatcher("/Final.jsp").forward(request, response); //Send to final ranking board.
    }
    else {
        request.getRequestDispatcher("/Ranking.jsp").forward(request, response); //Send to ranking in-between questions.
    }
}
```

Make a “lastDate” variable to store the time of the game in seconds since Jan 1, 1970. Create a “tempDate” variable and wait until it has been 2 seconds since the servlet request started to process.

The increased time allows for the score updates from the previous round to be completed by the PostgreSQL server. This servlet can now retrieve all the scores to display it on the ranking board.

Abstraction

The most central class to the application was the GameManager POJO. Often, servlets required information from it such as game states, current game question, and other game information. To make it easy to access from these servlets, the GameManager made use of abstraction in the form of variable encapsulation with only a select few public accessor methods.

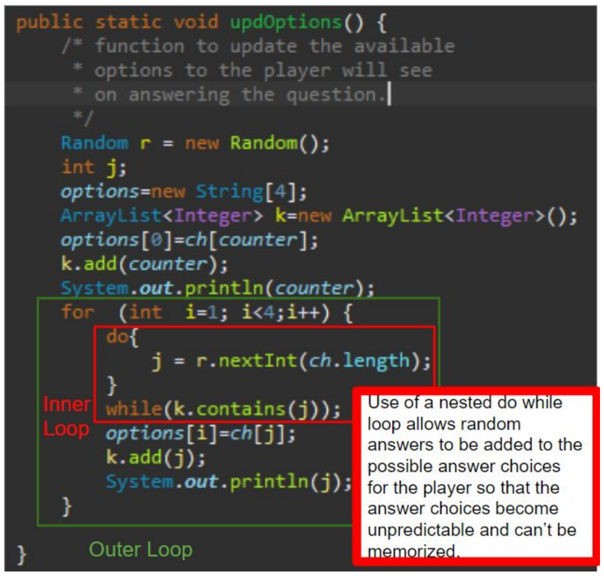


```
public class GameManager {
    private static int state;  //0=off, 1=starting, 2=rank 3=game 4=final game
    private static String name;
    private static String[] en;
    private static String[] ch;
    private static int counter=-1;
    private static String[] options;
```

```
{"English":["animal","thing","east","west"],
"name":"TestQuizSet",
"Chinese":["动物","东西","东方","西方"]}
```

For example, rather than making a servlet access all the arrays to adjust the options array to update it when a new question is asked, the function updOptions() is used.

In this way, the amount of logic clutter inside the servlets will be reduced so it can be performed by the POJO instead. This helps create a clear distinction between the PT (Presentation) layer and the BL (Business Logic) layer.



```java
public static void updOptions() {
    /* function to update the available
     * options to the player will see
     * on answering the question.
     */
    Random r = new Random();
    int j;
    options=new String[4];
    ArrayList<Integer> k=new ArrayList<Integer>();
    options[0]=ch[counter];
    k.add(counter);
    System.out.println(counter);
    for  (int  i=1; i<4;i++) {
        do{
            j = r.nextInt(ch.length);
        }
        while(k.contains(j));
        options[i]=ch[j];
        k.add(j);
        System.out.println(j);
    }
}
```

Use of a nested do while loop allows random answers to be added to the possible answer choices for the player so that the answer choices become unpredictable and can't be memorized.

.

Works Cited:

Garett, James (1999) http://api.jquery.com/jquery.ajax/