JavaDoc Documentation for Brain Blitz:

`Leaderboard.java`

```java
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

/**
 * Class representing the leaderboard for the trivia game.
 */
public class Leaderboard {
    private List<UserScore> scores;

    /**
     * Constructor for Leaderboard.
     * Initializes the leaderboard with sample scores.
     */
    public Leaderboard() {
        this.scores = new ArrayList<>();
        // Adding sample scores
        this.scores.add(new UserScore("Bilal Ahmad", 35));
        this.scores.add(new UserScore("Mohammed Bensassi", 30));
        this.scores.add(new UserScore("Ahmad Nasrallah", 25));
    }

    /**
     * Adds a score to the leaderboard.
     *
     * @param username The username of the player.
     * @param score    The score of the player.
     */
    public void addScore(String username, int score) {
        scores.add(new UserScore(username, score));
        scores.sort(Comparator.comparingInt(UserScore::getScore).reversed());
    }

    /**
     * Gets the top scores from the leaderboard.
     *
     * @param n The number of top scores to retrieve.
     * @return A list of the top scores.
```

```
     */
    public List<UserScore> getTopScores(int n) {
        return scores.subList(0, Math.min(n, scores.size()));
    }

    /**
     * Class representing a user's score on the leaderboard.
     */
    public static class UserScore {
        private String username;
        private int score;

        /**
         * Constructor for UserScore.
         *
         * @param username The username of the player.
         * @param score    The score of the player.
         */
        public UserScore(String username, int score) {
            this.username = username;
            this.score = score;
        }

        /**
         * Gets the username of the player.
         *
         * @return The username of the player.
         */
        public String getUsername() {
            return username;
        }

        /**
         * Gets the score of the player.
         *
         * @return The score of the player.
         */
        public int getScore() {
            return score;
        }
    }
}
```

`Question.java`

```java
/**
 * Class representing a trivia question.
 */
public class Question {
    private String questionText;
    private String[] choices;
    private String correctAnswer;

    /**
     * Constructor for Question.
     *
     * @param questionText The text of the question.
     * @param choices      The choices for the question.
     * @param correctAnswer The correct answer to the question.
     */
    public Question(String questionText, String[] choices, String correctAnswer) {
        this.questionText = questionText;
        this.choices = choices;
        this.correctAnswer = correctAnswer;
    }

    /**
     * Gets the text of the question.
     *
     * @return The text of the question.
     */
    public String getQuestionText() {
        return questionText;
    }

    /**
     * Gets the choices for the question.
     *
     * @return An array of choices for the question.
     */
    public String[] getChoices() {
        return choices;
    }

    /**
     * Gets the correct answer to the question.
```

```
     *
     * @return The correct answer to the question.
     */
    public String getCorrectAnswer() {
        return correctAnswer;
    }
}
```

`QuestionBank.java`

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * Class representing a bank of trivia questions.
 */
public class QuestionBank {
    private static List<Question> questions = new ArrayList<>();

    static {
        questions.add(new Question("What is the capital of France?", new String[]{"Paris",
"London", "Berlin", "Madrid"}, "Paris"));
        questions.add(new Question("What is the largest planet in our solar system?", new
String[]{"Earth", "Jupiter", "Mars", "Saturn"}, "Jupiter"));
        questions.add(new Question("Who wrote 'Hamlet'?", new String[]{"Shakespeare",
"Hemingway", "Fitzgerald", "Twain"}, "Shakespeare"));
        // Add more questions as needed
    }

    /**
     * Gets a shuffled list of questions from the question bank.
     *
     * @return A shuffled list of questions.
     */
    public static List<Question> getQuestions() {
        List<Question> shuffledQuestions = new ArrayList<>(questions);
        Collections.shuffle(shuffledQuestions);
        return shuffledQuestions.subList(0, Math.min(10, shuffledQuestions.size())); // Ensure only
10 questions are returned
    }
}
```

```

### `TriviaClient.java`

```java
/**
 * Class representing the trivia client.
 */
public class TriviaClient {
    public static void main(String[] args) {
        // Client code to interact with the server
    }
}
```

`TriviaGame.java`

```java
import java.util.List;

/**
 * Class representing a trivia game.
 */
public class TriviaGame {
    private User user;
    private List<Question> questions;
    private int currentQuestionIndex = 0;
    private int score = 0;

    /**
     * Constructor for TriviaGame.
     *
     * @param user      The user playing the game.
     * @param questions The list of questions for the game.
     */
    public TriviaGame(User user, List<Question> questions) {
        this.user = user;
        this.questions = questions;
    }

    /**
     * Gets the user playing the game.
     *
     * @return The user playing the game.
```
```

```
     */
    public User getUser() {
        return user;
    }

    /**
     * Gets the next question in the game.
     *
     * @return The next question in the game.
     */
    public Question getNextQuestion() {
        if (currentQuestionIndex < questions.size()) {
            return questions.get(currentQuestionIndex++);
        }
        return null;
    }

    /**
     * Verifies the user's answer to the current question.
     *
     * @param userAnswer The user's answer to the current question.
     * @return True if the answer is correct, false otherwise.
     */
    public boolean verifyAnswer(String userAnswer) {
        if (currentQuestionIndex == 0) return false; // no question has been asked yet
        Question currentQuestion = questions.get(currentQuestionIndex - 1);
        boolean isCorrect = currentQuestion.getCorrectAnswer().equals(userAnswer);
        if (isCorrect) {
            score += 5;
        }
        return isCorrect;
    }

    /**
     * Gets the user's current score.
     *
     * @return The user's current score.
     */
    public int getScore() {
        return score;
    }
}
```

`TriviaGUI.java`

```java
/**
 * Class representing the graphical user interface for the trivia game.
 */
public class TriviaGUI {
    public static void main(String[] args) {
        // GUI code to interact with the game
    }
}
```

`TriviaServer.java`

```java
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpServer;
import java.io.*;
import java.net.InetSocketAddress;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Class representing the server for the trivia game.
 */
public class TriviaServer {
    private UserStore userStore = new UserStore();
    private Map<String, TriviaGame> userGames = new HashMap<>();
    private Leaderboard leaderboard = new Leaderboard();

    /**
     * Starts the trivia server.
     *
     * @throws IOException If an I/O error occurs.
     */
    public void startServer() throws IOException {
        HttpServer server = HttpServer.create(new InetSocketAddress(8000), 0);
        server.createContext("/api/login", this::handleLoginRequest);
        server.createContext("/api/register", this::handleRegisterRequest);
        server.createContext("/api/start", this::handleStartGameRequest);
        server.createContext("/api/answer", this::handleAnswerRequest);
```

```
        server.createContext("/api/quest", this::handleQuestionRequest);
        server.createContext("/api/submitScore", this::handleSubmitScoreRequest);
        server.createContext("/api/leaderboard", this::handleLeaderboardRequest);
        server.setExecutor(null); // creates a default executor
        server.start();
        System.out.println("Server started on port 8000");
    }

    private void handleLoginRequest(HttpExchange exchange) throws IOException {
        handlePostRequest(exchange, data -> {
            boolean isAuthenticated = userStore.authenticateUser(data.get("username"),
data.get("password"));

            String response;
            if (isAuthenticated) {
                response = "{\"success\": true}";
                exchange.sendResponseHeaders(200,
response.getBytes(StandardCharsets.UTF_8).length);
            } else {
                response = "{\"success\": false, \"message\": \"Invalid credentials\"}";
                exchange.sendResponseHeaders(403,
response.getBytes(StandardCharsets.UTF_8).length);
            }
            return response;
        });
    }

    private void handleRegisterRequest(HttpExchange exchange) throws IOException {
        handlePostRequest(exchange, data -> {
            boolean isRegistered = userStore.registerUser(data.get("username"),
data.get("password"));

            String response;
            if (isRegistered) {
                response = "{\"success\": true}";
                exchange.sendResponseHeaders(200,
response.getBytes(StandardCharsets.UTF_8).length);
            } else {
                response = "{\"success\": false, \"message\": \"User already exists\"}";
                exchange.sendResponseHeaders(409,
response.getBytes(StandardCharsets.UTF_8).

length);
            }
```

```
            return response;
        });
    }

    private void handleStartGameRequest(HttpExchange exchange) throws IOException {
        handlePostRequest(exchange, data -> {
            String username = data.get("username");
            User user = userStore.getUser(username);
            TriviaGame newGame = new TriviaGame(user, QuestionBank.getQuestions());
            userGames.put(username, newGame);

            String response = "{\"success\": true}";
            exchange.sendResponseHeaders(200,
response.getBytes(StandardCharsets.UTF_8).length);
            return response;
        });
    }

    private void handleAnswerRequest(HttpExchange exchange) throws IOException {
        handlePostRequest(exchange, data -> {
            String username = data.get("username");
            String userAnswer = data.get("answer");

            TriviaGame game = userGames.get(username);
            boolean isCorrect = game.verifyAnswer(userAnswer);

            String response = isCorrect ? "{\"correct\": true}" : "{\"correct\": false}";

            exchange.sendResponseHeaders(200,
response.getBytes(StandardCharsets.UTF_8).length);
            return response;
        });
    }

    private void handleQuestionRequest(HttpExchange exchange) throws IOException {
        handlePostRequest(exchange, data -> {
            String username = data.get("username");
            TriviaGame game = userGames.get(username);
            Question question = game.getNextQuestion();

            String response;
            if (question != null) {
                response = String.format("{\"question\": \"%s\", \"choices\": [\"%s\", \"%s\", \"%s\",
\"%s\"]}",
```

```
                    question.getQuestionText(), question.getChoices()[0], question.getChoices()[1],
question.getChoices()[2], question.getChoices()[3]);
                exchange.sendResponseHeaders(200,
response.getBytes(StandardCharsets.UTF_8).length);
            } else {
                response = "{\"question\": null}";
                exchange.sendResponseHeaders(200,
response.getBytes(StandardCharsets.UTF_8).length);
            }
            return response;
        });
    }

    private void handleSubmitScoreRequest(HttpExchange exchange) throws IOException {
        handlePostRequest(exchange, data -> {
            String username = data.get("username");
            int score = Integer.parseInt(data.get("score"));

            leaderboard.addScore(username, score);

            String response = "{\"success\": true}";
            exchange.sendResponseHeaders(200,
response.getBytes(StandardCharsets.UTF_8).length);
            return response;
        });
    }

    private void handleLeaderboardRequest(HttpExchange exchange) throws IOException {
        List<Leaderboard.UserScore> topScores = leaderboard.getTopScores(10);
        StringBuilder response = new StringBuilder("[");
        for (Leaderboard.UserScore userScore : topScores) {
            response.append(String.format("{\"username\": \"%s\", \"score\": %d},",
userScore.getUsername(), userScore.getScore()));
        }
        if (response.length() > 1) {
            response.setLength(response.length() - 1); // Remove the trailing comma
        }
        response.append("]");
        exchange.sendResponseHeaders(200,
response.toString().getBytes(StandardCharsets.UTF_8).length);

exchange.getResponseBody().write(response.toString().getBytes(StandardCharsets.UTF_8));
        exchange.getResponseBody().close();
    }
```

```
    private void handlePostRequest(HttpExchange exchange, RequestHandler handler) throws
IOException {
        exchange.getResponseHeaders().add("Access-Control-Allow-Origin", "*");
        exchange.getResponseHeaders().add("Access-Control-Allow-Methods", "POST,
OPTIONS");
        exchange.getResponseHeaders().add("Access-Control-Allow-Headers", "Content-Type");

        if ("OPTIONS".equals(exchange.getRequestMethod())) {
            exchange.sendResponseHeaders(204, -1);
            return;
        }

        if ("POST".equals(exchange.getRequestMethod())) {
            InputStreamReader isr = new InputStreamReader(exchange.getRequestBody(),
StandardCharsets.UTF_8);
            BufferedReader br = new BufferedReader(isr);

            String query = br.readLine();
            Map<String, String> data = parseFormData(query);

            String response = handler.handle(data);
            OutputStream os = exchange.getResponseBody();
            os.write(response.getBytes());
            os.close();
        } else {
            exchange.sendResponseHeaders(405, 0);
            exchange.getResponseBody().close();
        }
    }

    private Map<String, String> parseFormData(String formData) {
        Map<String, String> map = new HashMap<>();
        String[] pairs = formData.split("&");
        for (String pair : pairs) {
            String[] keyValue = pair.split("=");
            map.put(keyValue[0], keyValue[1]);
        }
        return map;
    }

    @FunctionalInterface
    private interface RequestHandler {
        String handle(Map<String, String> data) throws IOException;
```

```
    }

    public static void main(String[] args) throws IOException {
        new TriviaServer().startServer();
    }
}
```

`User.java`

```java
/**
 * Class representing a user of the trivia game.
 */
public class User {
    private String username;
    private String password;
    private int score;

    /**
     * Constructor for User.
     *
     * @param username The username of the user.
     * @param password The password of the user.
     */
    public User(String username, String password) {
        this.username = username;
        this.password = password;
        this.score = 0;
    }

    /**
     * Gets the username of the user.
     *
     * @return The username of the user.
     */
    public String getUsername() {
        return username;
    }

    /**
     * Gets the password of the user.
     *
     * @return The password of the user.
```

```
     */
    public String getPassword() {
        return password;
    }

    /**
     * Gets the score of the user.
     *
     * @return The score of the user.
     */
    public int getScore() {
        return score;
    }

    /**
     * Adds points to the user's score.
     *
     * @param points The points to add.
     */
    public void addScore(int points) {
        this.score += points;
    }
}
```

`UserStore.java`

```java
import java.util.HashMap;
import java.util.Map;

/**
 * Class representing a store for user information.
 */
public class UserStore {
    private Map<String, User> users = new HashMap<>();

    /**
     * Registers a new user.
     *
     * @param username The username of the new user.
     * @param password The password of the new user.
     * @return True if the user was successfully registered, false if the username is already taken.
     */
```

```
    public boolean registerUser(String username, String password) {
        if (users.containsKey(username)) {
            return false;
        }
        users.put(username, new User(username, password));
        return true;
    }

    /**
     * Authenticates a user.
     *
     * @param username The username of the user.
     * @param password The password of the user.
     * @return True if the username and password are correct, false otherwise.
     */
    public boolean authenticateUser(String username, String password) {
        User user = users.get(username);
        return user != null && user.getPassword().equals(password);
    }

    /**
     * Gets a user by their username.
     *
     * @param username The username of the user.
     * @return The user with the specified username, or null if no such user exists.
     */
    public User getUser(String username) {
        return users.get(username);
    }
}
```