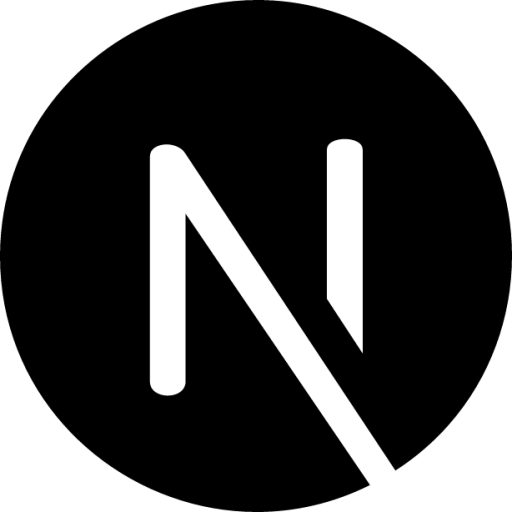
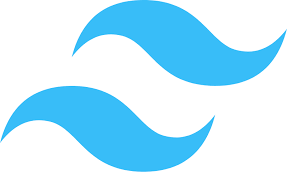
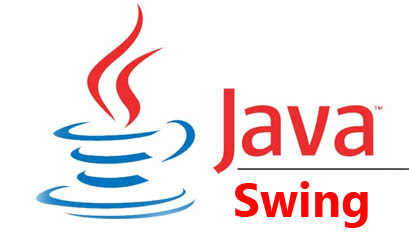
OOSD Project

Online Casino

Temur Rustamov (C00280204)

Filip Melka (C00288344)

April 2024

Table of Contents

[Description 3](#_Toc165408886)

[Web Application 3](#_Toc165408887)

[Desktop App 4](#_Toc165408888)

[Requirements 6](#_Toc165408889)

[Database Tables 7](#_Toc165408890)

[Mystery box table 7](#_Toc165408891)

[Payments table 7](#_Toc165408892)

[Results table 7](#_Toc165408893)

[Tokens table 8](#_Toc165408894)

[Users table 8](#_Toc165408895)

[ER Diagram 9](#_Toc165408896)

[Interesting Source Code Snippets 10](#_Toc165408897)

[Token Authentication 10](#_Toc165408898)

[Hashing Passwords 11](#_Toc165408899)

[Random profile images using DiceBear 12](#_Toc165408900)

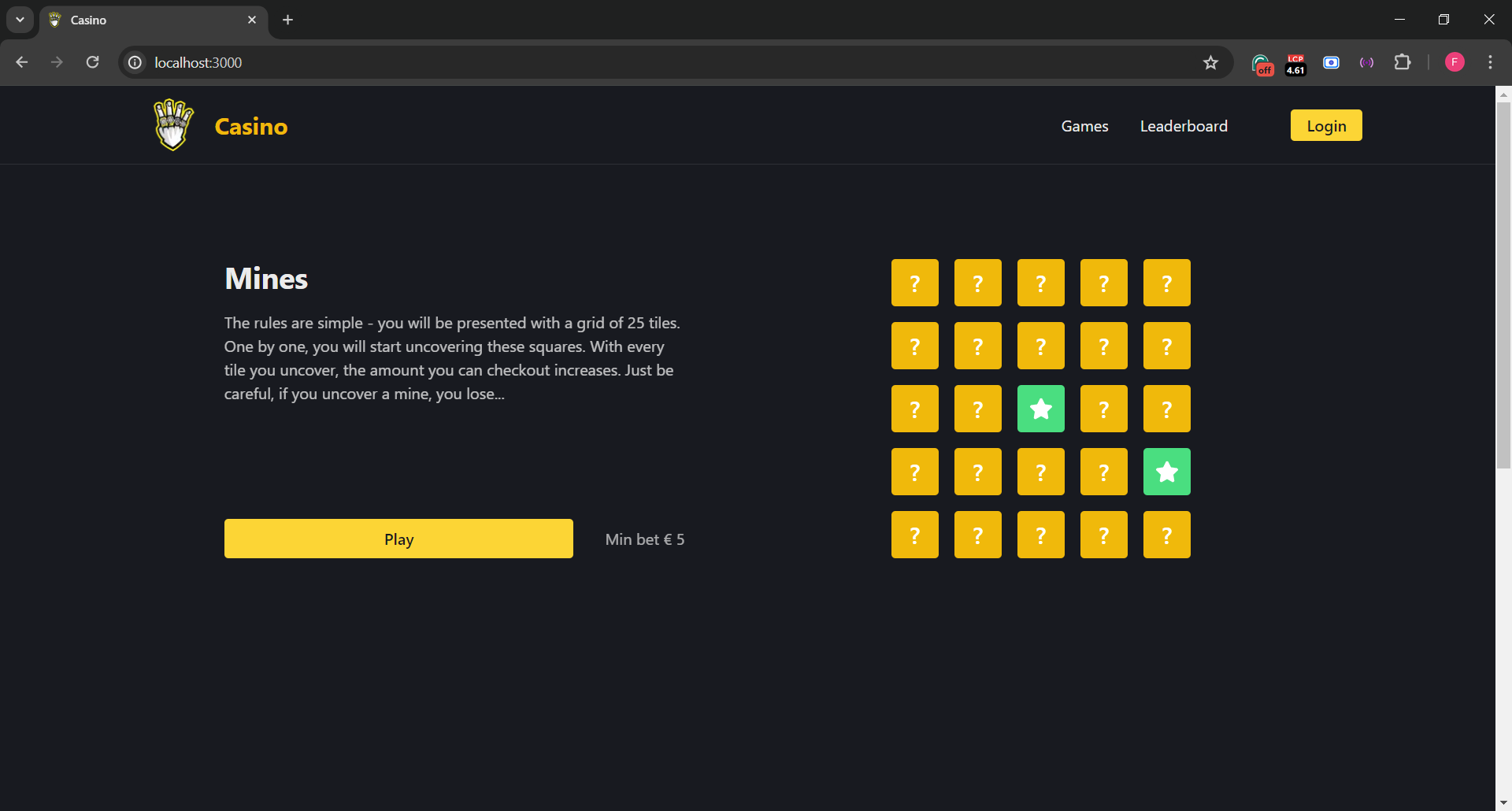
[Tests 13](#_Toc165408901)

# Description

Our system is a thrilling online gambling platform that offers an immersive and exciting gaming experience.

Because most users prefer web applications (instead of having to download an app) we decided to split our project into two parts – a web application and a desktop application.

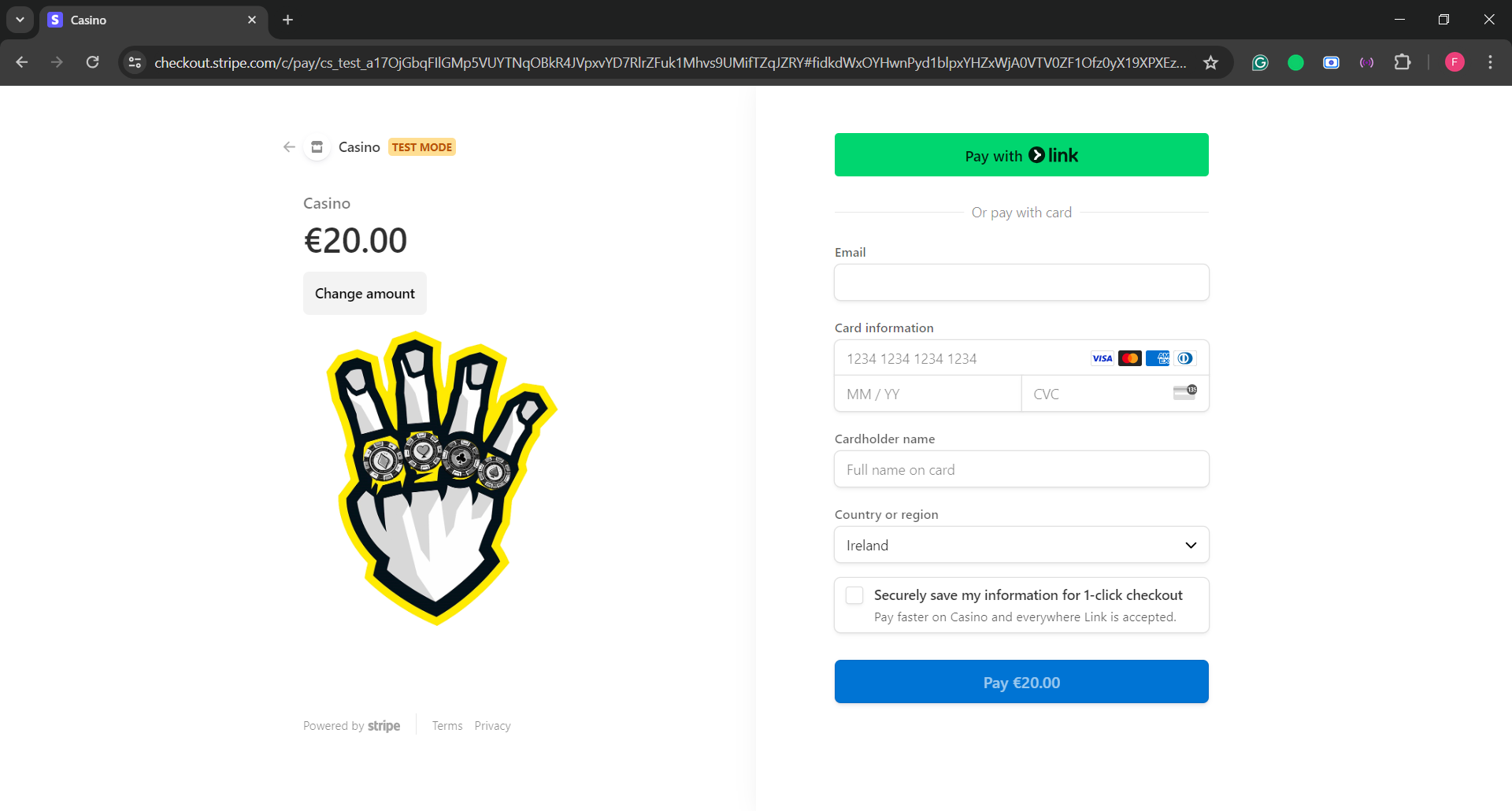
## Web Application



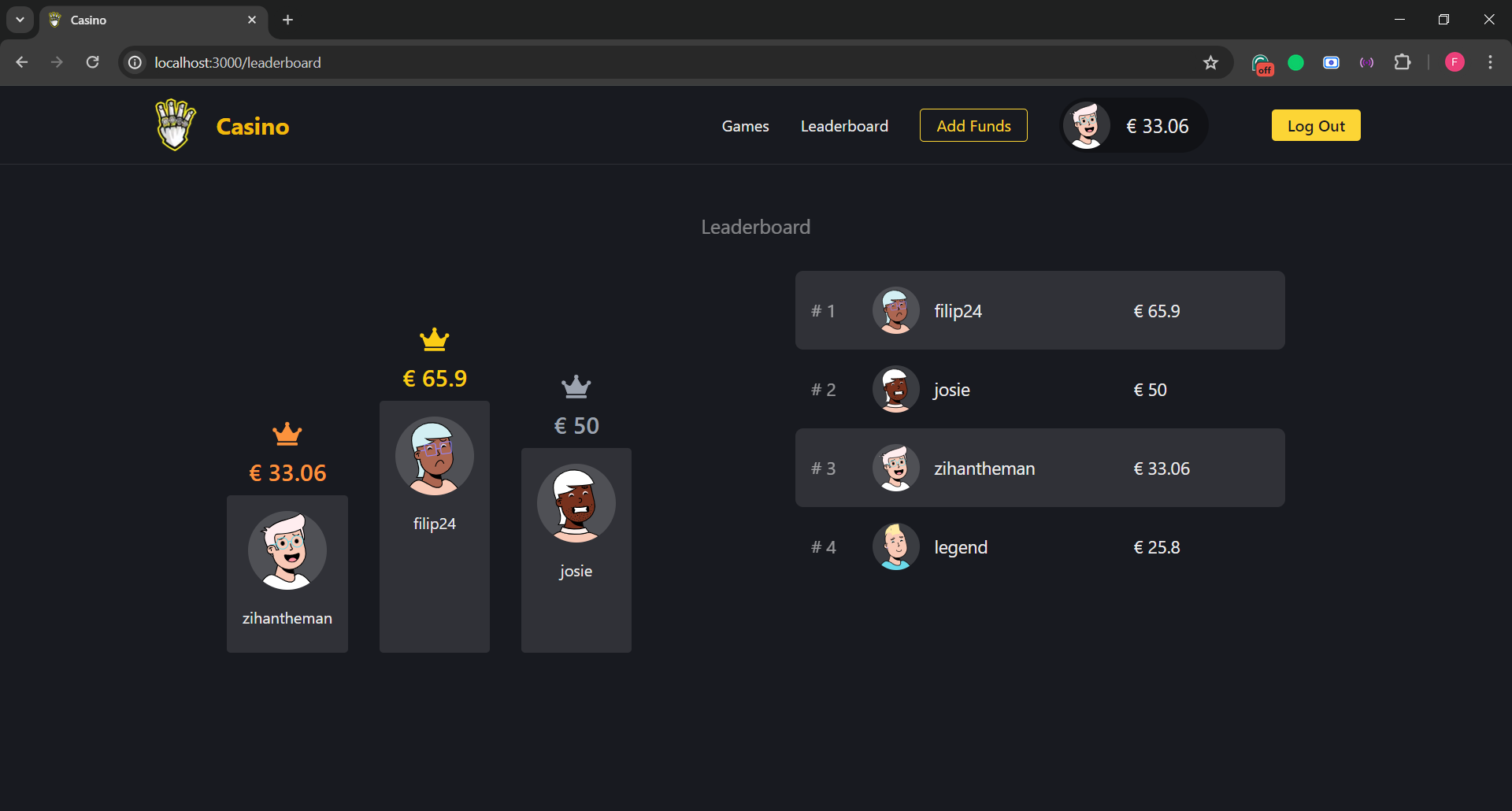
The web application is the foundation of our system. It was created using Next JS. Through this application a user can:

* Sign up
* Play games (currently only Mines)
* Deposit funds
* View leaderboard

Depositing funds is done through Stripe, as it is the most secure and user-friendly way of processing payments.



The leaderboard displays all users, ranked by the amount of money in their account.

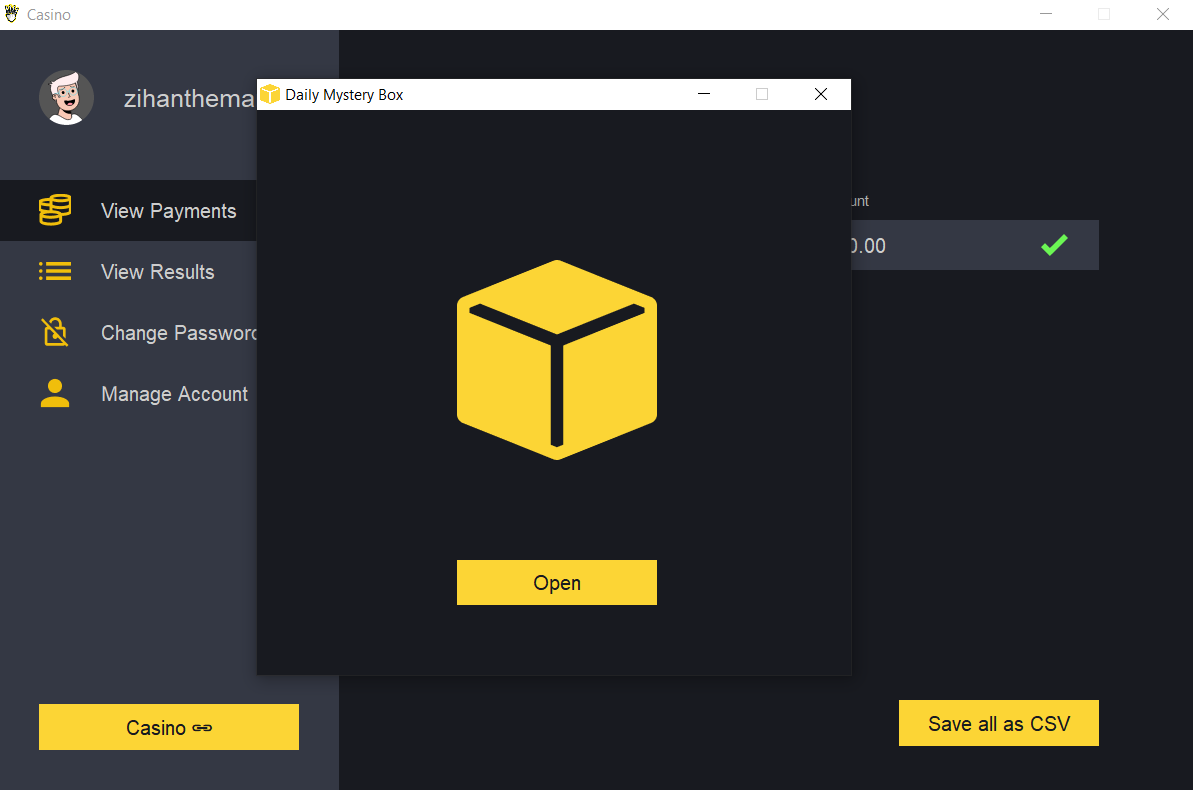


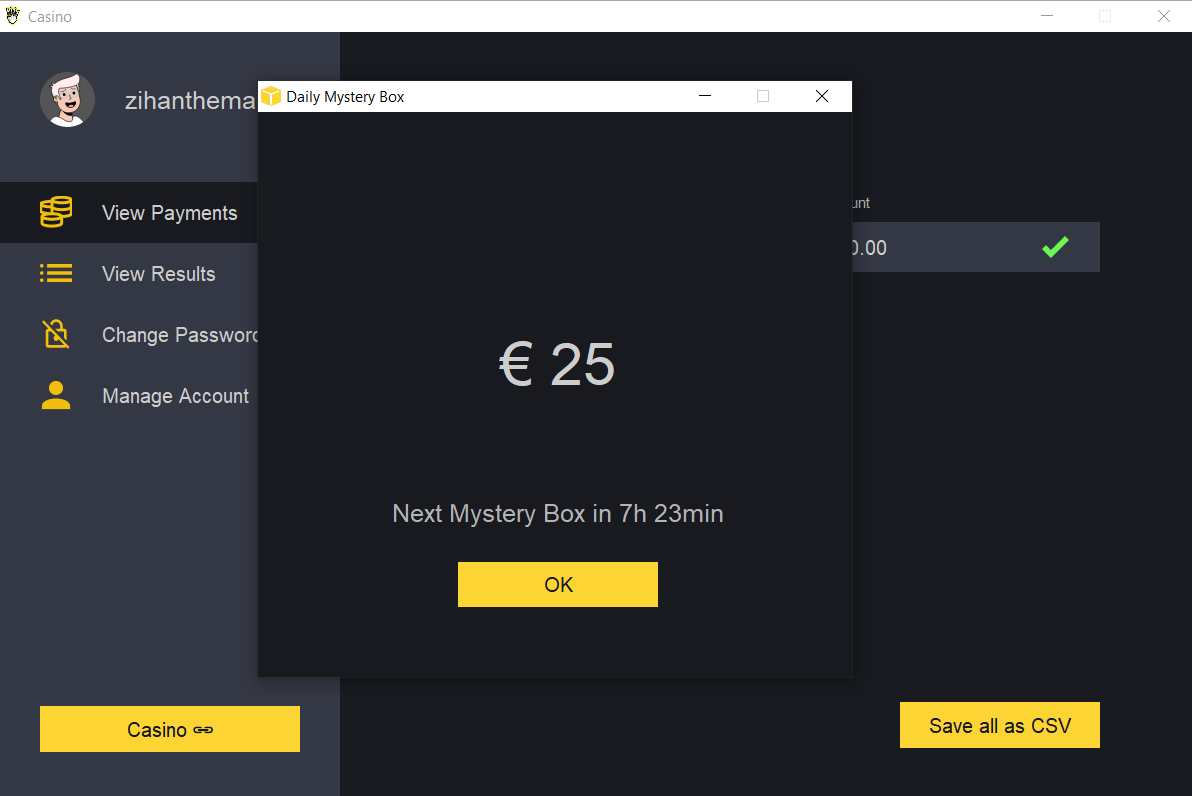
## Desktop App

The desktop app serves as an addition to the web app. It was created using Java Swing library and adds additional features for managing user’s account. It allows users to:

* View recent payments
* Save all payments (as CSV file)
* View recent game results
* Save all game results (as CSV file)
* Change password
* View account details
* Delete account

In addition, to motivate users to download our app, we added daily Mystery Box. When the user opens the app for the first time in a day, a Mystery Box window pops up. After the user clicks on the ‘Open’ button, a random prize is shown, and this amount is then added to user’s balance.





The information about whether the daily Mystery Box has been claimed is stored in the users table in the ‘nextMysteryBox’ column (which stores the date when next Mystery Box is available).



# Requirements

1. User Registration and Authentication

* The system should provide a user registration mechanism for new players to create accounts
* Users should be able to log in securely using their credentials

1. Game Selection and Play

* Users should be able to select and play games of their choice from the available options
* Games should be interactive, engaging, and user-friendly
* The system should provide realistic game simulations with accurate odds and outcomes

1. Account Management

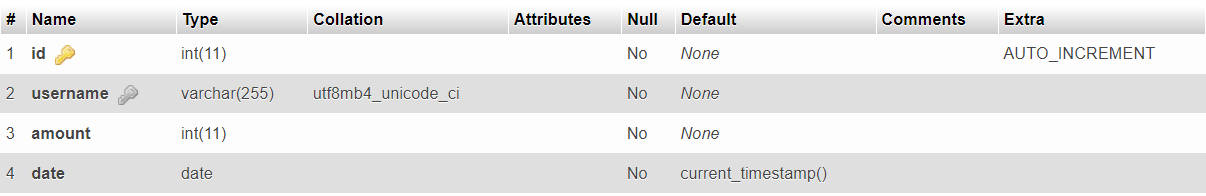
* The system should allow users to deposit funds into their accounts securely
* Users should have access to a dashboard where they can see their user details
* Users should have access to all of their payments and results records
* Users should be able to change their password

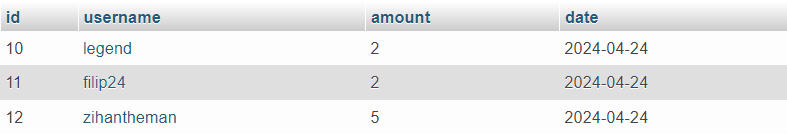
1. Bonuses and Promotions

* The system should offer bonuses and rewards to attract new players and retain existing ones

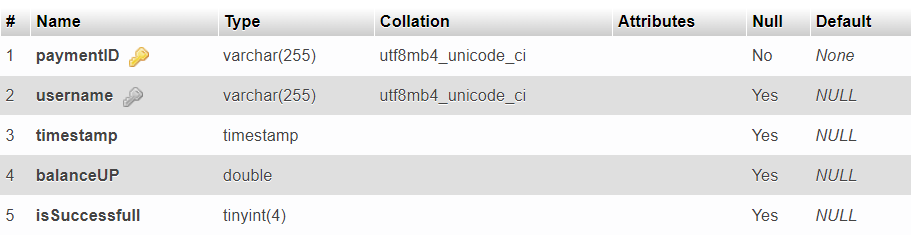
# Database Tables

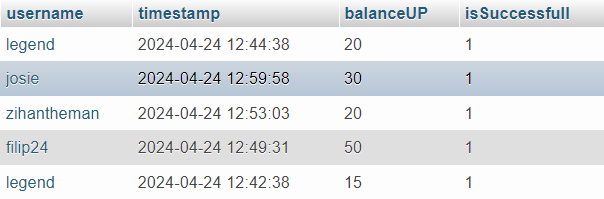
## Mystery box table



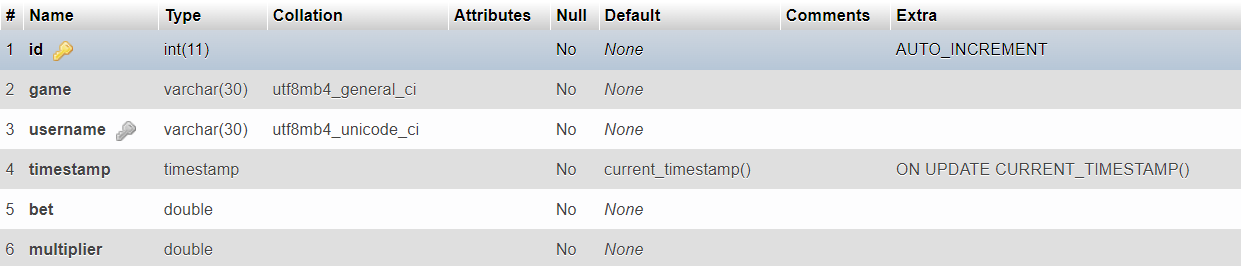


## Payments table



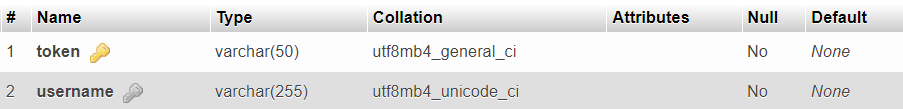


## Results table

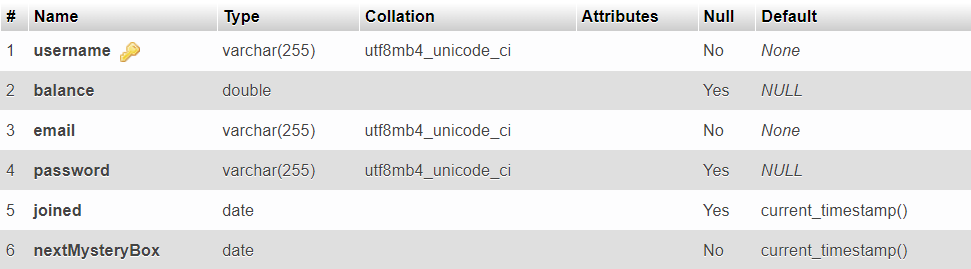


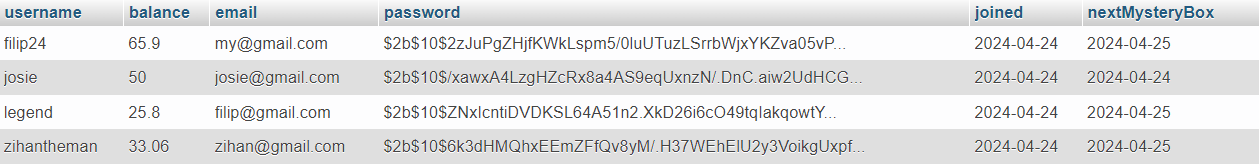


## Tokens table

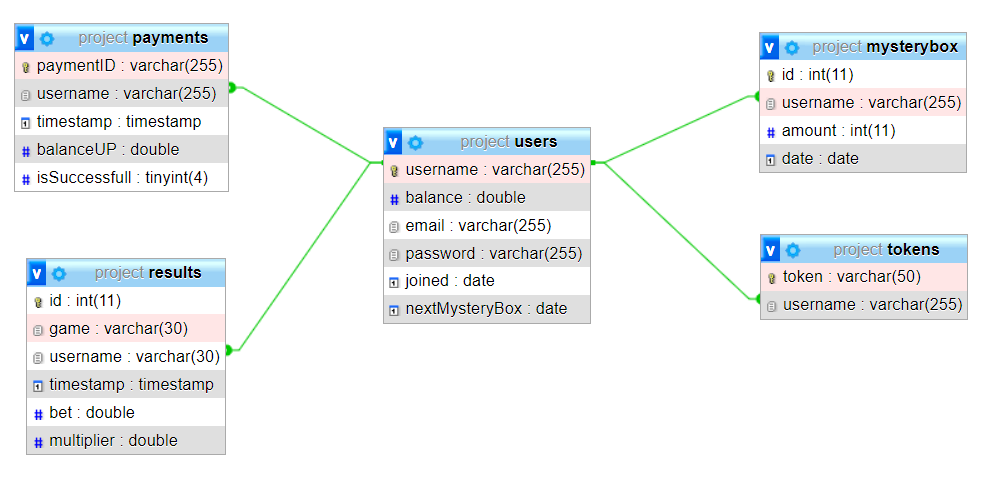


## Users table





# ER Diagram



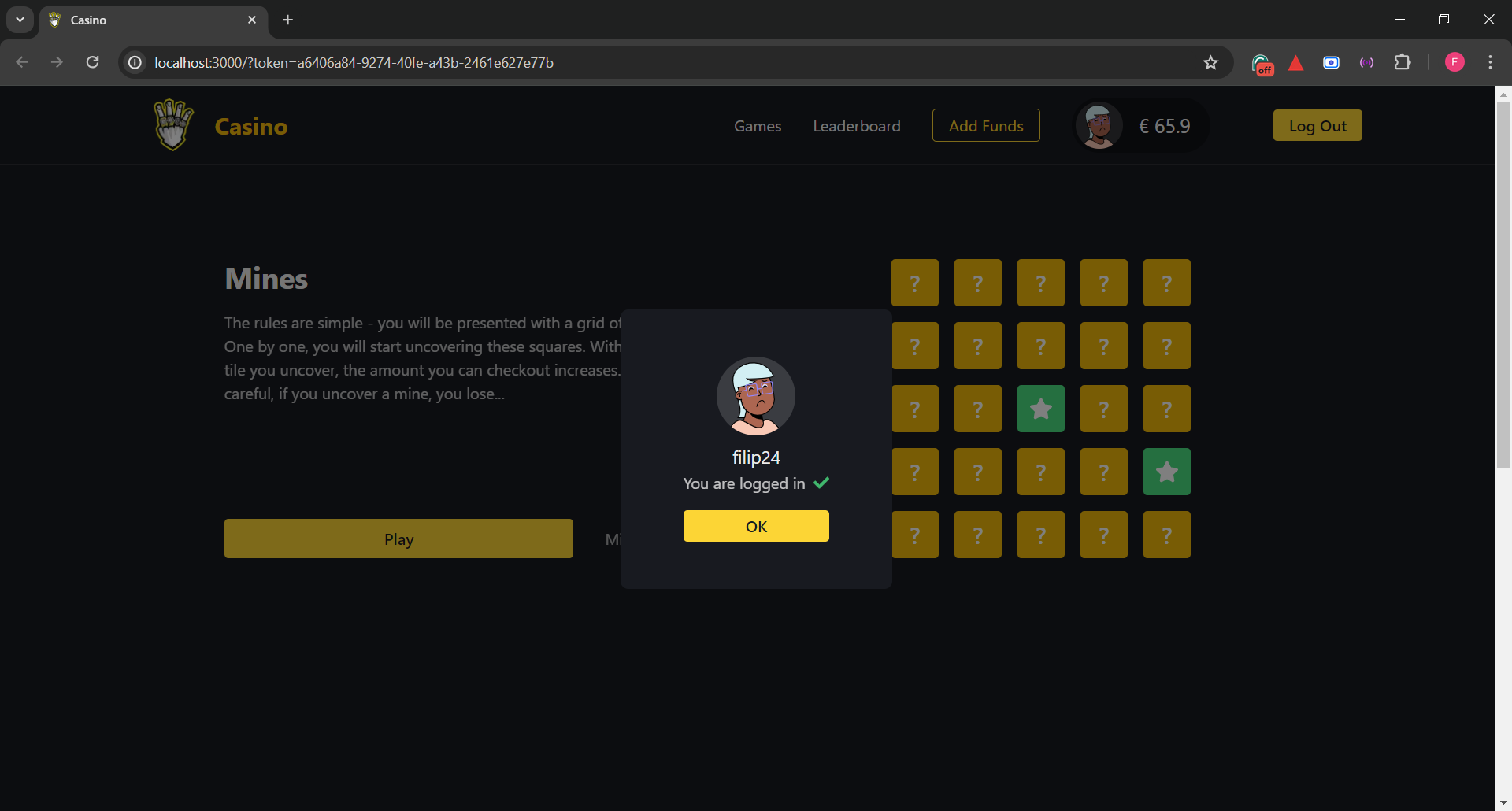
# Interesting Source Code Snippets

## Token Authentication

The desktop application provides a ‘magic’ link to the web app. When the link button is pressed, a new random token is generated and stored in the token table in the database (alongside with the username). This token is then included in the URL.



When the desktop app is opened, it searches the URL for the token parameter. If it finds one, it checks the database if there is a record with this token and if so, it retrieves the username and logs the user in without having to type in the username and password. Also, it deletes the record from the database.



Below is the code for the link button (which extends our ‘Button’ class):

public class LinkButton extends Button implements ActionListener {

    Button btn;

    public void actionPerformed(ActionEvent ev) {

        btn.setIsEnabled(false);

        btn.setBackground(Colors.PRIMARY\_DARK);

        btn.setForeground(Colors.BG);

        /\* create and store auth Token \*/

        String token = UUID.randomUUID().toString();

        System.out.println(token.length());

        try {

            /\* store the auth Token in the 'tokens' table \*/

            Connection connection = DriverManager.getConnection(DB.URL, DB.USER, DB.PASSWORD);

            PreparedStatement pstat = connection.prepareStatement("INSERT INTO tokens (token, username) VALUES (?,?)");

            pstat.setString(1, token);

            pstat.setString(2, User.getUsername());

            pstat.executeUpdate();

            /\* open up the website \*/

            URI uri = new URI("http://localhost:3000?token=" + token);

            Desktop desktop = Desktop.isDesktopSupported() ? Desktop.getDesktop() : null;

            desktop.browse(uri);

        } catch (Exception e) {

            e.printStackTrace();

        } finally {

            btn.setIsEnabled(true);

            btn.setBackground(Colors.PRIMARY);

        }

    }

    public LinkButton(String title) {

        super(title);

        btn = this;

        super.setForeground(Colors.BG);

        super.setBackground(Colors.PRIMARY);

        super.setFont(Fonts.DEFAULT);

        super.addActionListener(this);

    }

}

## Hashing Passwords

The user has the option to change his password. Because all passwords stored in the database are hashed, a hashing algorithm had to be used. For this, Bcrypt was chosen as it is a good algorithm for password storage. Unfortunately, we had difficulties implementing Bcrypt in Java. Because of that, we created an API endpoint in our NextJS application, which takes care of hashing a plain text password and returns its hashed version. In our Java code we send an HTTP request to this endpoint to get back the hashed password, which is then used to update the user table in the database.

/\* get the plain text password from the input field \*/

String newPassword = passwordInputs[1].getPassword();

URL obj = new URL("http://localhost:3000/api/hash-password?password=" + newPassword);

HttpURLConnection con = (HttpURLConnection) obj.openConnection();

con.setRequestMethod("GET"); /\* GET request \*/

con.setRequestProperty("User-Agent", "Mozilla/5.0");

int responseCode = con.getResponseCode();

if (responseCode == HttpURLConnection.HTTP\_OK) {

   BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));

   StringBuffer response = new StringBuffer();

   String inputLine;

   while ((inputLine = in.readLine()) != null) {

       response.append(inputLine);

   }

   in.close();

   /\* hashed password used to update user table \*/

   String hashedPassword = response.toString().replaceAll("\"", "");

}

Similarly, when logging into the desktop app, a NextJS API endpoint is used for comparing the hashed and plain text passwords.

## Random profile images using [DiceBear](https://www.dicebear.com/)

Due to time constraint, we were trying to simplify things as much as possible. We wanted each user to have his own profile image, without having to handle image inputs. Our solution uses [DiceBear](https://www.dicebear.com/), which provides random avatar generators in many distinct styles. In addition, you can provide a seed which is used for generating the image. In our case, we are using the username (which is unique) as the seed. Because of this, we don’t need to store the avatar images in our database, as we are getting the images from a URL.

public class Avatar extends JLabel {

    ImageIcon icon;

    public Avatar() {

        super(User.getUsername());

        try {

            /\* using 'Dicebear' to get random profile image based on the username \*/

            URL url = new URL("https://api.dicebear.com/8.x/micah/png?radius=50&backgroundColor=555555&size=55&seed="

                    + User.getUsername());

            BufferedImage image = ImageIO.read(url);

            icon = new ImageIcon(image);

            this.setIcon(icon);

            this.setForeground(Colors.TEXT);

            this.setFont(Fonts.AVATAR);

            this.setIconTextGap(30);

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}

# Tests

package test;

public class TestCases {

    @RepeatedTest(10)

    @DisplayName("Ensure mystery boxes don't exceed prize of €50 and isn't less than €2")

    public void testMysteryBox() {

        MysteryBox mysteryBox = new MysteryBox();

        assertTrue("The prize doesn't exceed €50", mysteryBox.open() <= 50);

        assertTrue("The prize isn't less than €2", mysteryBox.open() >= 2);

    }

    @Test

    @DisplayName("Test Payment class")

    public void testPayment() {

        Payment payment = new Payment("29/04/2023", "16:23", "20");

        assertAll("Payment tests",

                () -> assertEquals("29/04/2023", payment.getDate()),

                () -> assertEquals("16:23", payment.getTime()),

                () -> assertEquals("€ 20", payment.getAmount())

        );

    }

    @Test

    @DisplayName("Test Result class")

    public void testResult() {

        Result result = new Result("29/04/2023", "5", "1.25", "mines");

        assertAll("Result tests",

                () -> assertEquals("29/04/2023", result.getDate()),

                () -> assertEquals("€ 5", result.getBet()),

                () -> assertEquals("1.25", result.getMultiplier()),

                () -> assertTrue(result.getIsWin())

        );

    }

}

