SWE 4x	**Übung zu Softwareentwicklung mit modernen Plattformen 4**	**SS 2016, Übung 9**

Abgabetermin: MI in der KW 25

☒ **Gr. 1**, Dr. E. Pitzer	**Name** Robert Wurm	**Aufwand in h** 19
☐ **Gr. 2**, DI F. Gruber-Leitner		
	Punkte ________ **Kurzzeichen Tutor / Übungsleiter**	______ / ______

Ausbaustufe 2: EURO-Bet-DB (24 Punkte)

Nachdem der Betreiber der Wettplattform von der Präsentation Ihrer beiden ersten Ausbaustufen restlos begeistert war, steht einem Ausbau bis hin zur Online-Plattform nichts mehr im Wege.

Dafür ist es im nächsten Schritt notwendig, die Daten in einer relationalen Datenbank zu speichern. Weiters beschließen Sie aufgrund Ihres nunmehrigen Wissens über die Vorzüge von RMI, Ihr Programm zu einer Client-/Server-Anwendung auszubauen.

Im Detail sollte Ihre Anwendung folgende Anforderungen erfüllen:

- Überlegen Sie sich eine geeignete Repräsentation für die Daten in Ihrem Programm, also Klassen für die Speicherung der Benutzer, Spielpaarungen, Spielergebnisse und Tipps. Diese Klassen werden häufig als Domänenmodell bezeichnet. Berücksichtigen Sie nicht nur die Daten, die vom Verwaltungstools benötigt werden, sondern auch die Daten für die Online-Wettplattform.
- Implementieren Sie einen RMI-Server, der die vom Verwaltungstool erfassten Daten zentral speichert bzw. diesen mit den zentral gespeicherten Daten versorgt. Stellen Sie sicher, dass die Serverkomponente parallel beliebig viele Clients mit Daten versorgen kann.
- Bauen Sie die in Übung 6 entwickelte JavaFX-Anwendung zu einem RMI-Client aus, der mit der Serverkomponente kommuniziert.
- Da der Zugriff auf die Server-Anwendung mitunter langsam sein kann, müssen Sie Vorkehrungen dafür treffen, dass die Kommunikation mit der Server-Anwendung und die Aktualisierung der Benutzeroberfläche entkoppelt sind.
- Stellen Sie Klassen zum Zugriff auf die Datenbank auf Basis von JDBC zur Verfügung. Bereiten Sie auch bereits Methoden für das Hinzufügen und Laden von Tipps vor. Achten Sie darauf, dass die Datenbankzugriffschicht möglichst weitgehend von den anderen Komponenten der Anwendung getrennt ist. Stellen Sie dazu die gesamte Funktionalität der Datenbankzugriffsschicht über Interfaces zur Verfügung.
- Entwickeln Sie eine Testsuite, mit der Sie die Datenzugriffsschicht unabhängig von den anderen Systemkomponenten testen können.

Überprüfen Sie abschließend, ob Ihr Programm folgende Anforderungen erfüllt:

- Die Eingabe von Benutzern, Mannschaften, Spielpaarungen und Spielergebnissen ist möglich. Dabei werden die Daten mittels RMI zur Serveranwendung übertragen und dort mittels JDBC in eine Datenbank geschrieben. Es hängt von der Gestaltung Ihrer Benutzeroberfläche ab, an welchen Stellen Sie das Speichern der Daten anstoßen. Sorgen Sie aber dafür, dass bei unbeabsichtigter Beendigung des Programms, nicht allzu viele Daten verlorengehen.
- Bei Programmstart oder entsprechender Benutzerinteraktion („Daten aktualisieren“) werden die Daten vom RMI-Server (neu) geladen.
- Mehrere Instanzen des Verwaltungstools können gleichzeitig laufen. Änderungen des Datenbestands sind nach entsprechender Benutzerinteraktion auch in anderen Instanzen sichtbar.

- Das Eintragen und Laden von Tipps ist seitens der Datenbank und der Serveranwendung vorbereitet (es gibt aber im Moment keine Benutzeroberfläche zur Abgabe von Tipps).

Inhaltsverzeichnis

1 Euro-Bet-DB

1.1 Lösungsidee

Um die Daten zu repräsentieren habe ich meine Models erweitert. Die Eigenschaften der Models habe ich nun als Properties umgesetzt. Dies ermöglicht mir, die Models direkt an JavaFX Komponenten, wie zum Beispiel die Listview, zu binden.
Auf der Clientseite habe ich einen Singelton 'EuroBetDbFactory' erstellt. Dieser Singelton ist für die Kommunikation für den Server verantwortlich. Ich habe mich für diese Factory-Implementierung entschieden, da dies in der Praxis sehr oft so umgesetzt wird und einige Vorteile bringt (Backend kann leicht ausgetauscht werden, Event-Listeners können aufgesplittet werden, Server Kommunikation muss nur einmal implementiert werden, usw..). Die Unit-Tests habe ich ebenso gleich für meine Factory geschrieben.
Sind in der UI nicht die Bedingungen erfüllt, sodass ein Update oder das Erstellen eines Objekts wie zum Beispiel User ausgeführt werden darf, so ist der Speichern-Button nicht aktiviert. Hier habe ich das Enabled-Property der Buttons an Verknüpfungen der Model-Properties gebunden.

Die Implemtierung für den Server ist in dem Packet 'swe4.rmi.server' zu finden. Unter Quellcode habe ich nur noch die Klassen angeführt, die sich gegenüber der vorgehenden Übung geändert haben bzw. neu hinzugefügt worden sind, da die Dokumentation ohnehin schon sehr viele Seiten hat.

1.2 Quellcode

1.2.1 swe4.util

Util.java

```
package swe4.util;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Util {

        static long startTime = System.currentTimeMillis();

        public static Object currTime() {
                return (System.currentTimeMillis() - startTime) / 1000.0;
        }


        public static String getHostPortArg(String[] args) {
                if (args == null ||  args.length == 0) {
                        System.out.println("WARNING: host[:port] missing (default is
                            ↪ localhost:1099)");
                    return "localhost:1099";
                }
                return args[0];
        }

        public static int getPort(String host_port) {
                int port = 1099;
            int colonIdx = host_port.lastIndexOf(':');
            if (colonIdx >= 0)
              port = Integer.valueOf(host_port.substring(colonIdx + 1));
            return port;
        }

        public static void unsafeSleep(int millis) {
            try {
              Thread.sleep(millis);
            } catch (InterruptedException ex) {}
        }

        public static void unsafeWait(Object obj) {
            try {
              obj.wait();
            } catch (InterruptedException e) { }
        }

        public static String readline() {
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in)
                ↪ );
            try {
              return reader.readLine();
            } catch (IOException e) { }
            return null;
        }

}
```

1.2.2 swe4.models

Game.java

```java
package swe4.models;

import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;

import javafx.beans.property.BooleanProperty;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.SimpleBooleanProperty;
import javafx.beans.property.SimpleIntegerProperty;

public class Game implements Externalizable {

        private Team teamA;
        private Team teamB;

        private IntegerProperty id;
        private BooleanProperty gameFinished;
        private IntegerProperty goalsTeamA;
        private IntegerProperty goalsTeamB;


        public Game() {
                this.id = new SimpleIntegerProperty(-1);
                this.gameFinished = new SimpleBooleanProperty(false);
                this.goalsTeamA = new SimpleIntegerProperty(-1);
                this.goalsTeamB = new SimpleIntegerProperty(-1);
        }

        public Game(Team teamA, Team teamB) {
                this();

                this.setTeamA(teamA);
                this.setTeamB(teamB);
        }

        public Game(Integer id, Team teamA, Team teamB, Boolean gameFinished, Integer
            ↪ goalsTeamA, Integer goalsTeamB) {
                this(teamA, teamB);

                this.id.set(id);
                this.gameFinished.set(gameFinished);
                this.goalsTeamA.set(goalsTeamA);
                this.goalsTeamB.set(goalsTeamB);
        }


        public void clone(Game game) {
                if(game != null) {
                        this.setId(game.getId());
                        this.setTeamA(game.getTeamA());
                        this.setTeamB(game.getTeamB());
                        this.setGameFinished(game.getGameFinished());
                        this.setGoalsTeamA(game.getGoalsTeamA());
                        this.setGoalsTeamB(game.getGoalsTeamB());

                } else {
                        this.setId(-1);
                        this.setTeamA(new Team());
                        this.setTeamB(new Team());
                        this.setGameFinished(false);
                        this.setGoalsTeamA(-1);
                        this.setGoalsTeamB(-1);
                }
        }


        public Team getTeamA() {
                return teamA;
        }
        public void setTeamA(Team teamA) {
                this.teamA = teamA;
        }
        public Team getTeamB() {
                return teamB;
        }
        public void setTeamB(Team teamB) {
                this.teamB = teamB;
        }


        public IntegerProperty getIdProperty() {
                return id;
        }

        public final Integer getId() {
                return id.get();
        }

        public final void setId(Integer id) {
                this.id.set(id);
        }



        public IntegerProperty getGoalsTeamAProperty() {
                return goalsTeamA;
        }

        public final Integer getGoalsTeamA() {
                return goalsTeamA.get();
        }

        public final void setGoalsTeamA(Integer goals) {
                this.goalsTeamA.set(goals);
```

```java
    }



    public IntegerProperty getGoalsTeamBProperty() {
        return goalsTeamB;
    }

    public final Integer getGoalsTeamB() {
        return goalsTeamB.get();
    }

    public final void setGoalsTeamB(Integer goals) {
        this.goalsTeamB.set(goals);
    }




    public BooleanProperty getGameFinishedProperty() {
        return gameFinished;
    }

    public final Boolean getGameFinished() {
        return gameFinished.get();
    }

    public final void setGameFinished(Boolean finished) {
        this.gameFinished.set(finished);
    }




    public String getResult() {
        if(getGameFinished() && goalsTeamA != null && goalsTeamB != null) {
            return getGoalsTeamA().toString() + " : " + getGoalsTeamB().toString();
        }
        return "- : - ";
    }


    @Override
    public String toString() {
        return "id: " + this.getId() + " "
                + this.teamA.getName() + " - " + this.teamB.getName().toString()
                + " " + this.getGoalsTeamA() + ":" + this.getGoalsTeamB()
                + " finished: " + this.getGameFinished();
    }

```

```java


        @Override
        public void writeExternal(ObjectOutput out) throws IOException {
                out.writeObject(getTeamA());
                out.writeObject(getTeamB());
                out.writeInt(getId());
                out.writeBoolean(getGameFinished());
                out.writeInt(getGoalsTeamA());
                out.writeInt(getGoalsTeamB());
        }

        @Override
        public void readExternal(ObjectInput in) throws IOException,
            ↪ ClassNotFoundException {
                setTeamA((Team) in.readObject());
                setTeamB((Team) in.readObject());
                setId(in.readInt());
                setGameFinished(in.readBoolean());
                setGoalsTeamA(in.readInt());
                setGoalsTeamB(in.readInt());
        }

}
```

Team.java

```java
package swe4.models;

import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class Team implements Externalizable {

        private IntegerProperty id;
        private StringProperty name;
        private StringProperty group;


        public Team() {
                this.id = new SimpleIntegerProperty(-1);
                this.name = new SimpleStringProperty("");
                this.group = new SimpleStringProperty("");
        }
```

```

		public Team(Integer id, String name, String group) {
			this();

			this.id.set(id);
			this.name.set(name);
			this.group.set(group);
		}


		public void clone(Team team) {
			if(team != null) {
				this.setId(team.getId());
				this.setName(team.getName());
				this.setGroup(team.getGroup());
			} else {
				this.setId(-1);
				this.setName("");
				this.setGroup("");
			}
		}



		public IntegerProperty getIdProperty() {
			return id;
		}

		public final Integer getId() {
			return id.get();
		}

		public final void setId(Integer id) {
			this.id.set(id);
		}



		public StringProperty getNameProperty() {
			return name;
		}

    public final String getName() {
        return name.get();
    }

    public final void setName(String name) {
        this.name.set(name);
    }



    public StringProperty getGroupProperty() {
```

```
                return group;
        }

    public final String getGroup() {
        return group.get();
    }

    public final void setGroup(String group) {
        this.group.set(group);
    }


    @Override
        public String toString() {

                return "id:" + id.getValue() +
                                " name: " + name.getValue() +
                                " group: " + group.getValue();
        }



        @Override
        public void writeExternal(ObjectOutput out) throws IOException {
                out.writeInt(getId());
                out.writeObject(getName());
                out.writeObject(getGroup());
        }

        @Override
        public void readExternal(ObjectInput in) throws IOException,
            ↪ ClassNotFoundException {
                setId(in.readInt());
                setName((String) in.readObject());
                setGroup((String) in.readObject());
        }
}
```

User.java

```
package swe4.models;

import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import javafx.beans.property.BooleanProperty;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.SimpleBooleanProperty;
import javafx.beans.property.SimpleIntegerProperty;
```

```java
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class User implements Externalizable {


	private IntegerProperty id;
	private BooleanProperty active;
	private StringProperty name;
	private StringProperty password;


	public User() {
		this.id = new SimpleIntegerProperty(-1);
		this.active = new SimpleBooleanProperty(false);
		this.name = new SimpleStringProperty("");
		this.password = new SimpleStringProperty("");
	}

	public User(Integer id, Boolean active, String name, String password) {
		this();

		this.id.set(id);
		this.active.set(active);
		this.name.set(name);
		this.password.set(password);
	}


	public void clone(User user) {
		if(user != null) {
			this.setId(user.getId());
			this.setActive(user.getActive());
			this.setName(user.getName());
			this.setPassword(user.getPassword());
		} else {
			this.setId(-1);
			this.setActive(false);
			this.setName("");
			this.setPassword("");
		}
	}

	public IntegerProperty getIdProperty() {
		return id;
	}

	public final Integer getId() {
		return id.get();
	}

	public final void setId(Integer id) {
		this.id.set(id);
```

```java
        }



        public BooleanProperty getActiveProperty() {
                return active;
        }

        public final Boolean getActive() {
                return active.get();
        }

        public final void setActive(Boolean active) {
                this.active.set(active);
        }



        public StringProperty getNameProperty() {
                return name;
        }

    public final String getName() {
        return name.get();
    }

    public final void setName(String name) {
        this.name.set(name);
    }




    public StringProperty getPasswordProperty() {
                return password;
        }

    public final String getPassword() {
        return password.get();
    }

    public final void setPassword(String password) {
        this.password.set(password);
    }


        @Override
        public String toString() {

                return "id:" + id.getValue() +
                                " name: " + name.getValue() +
                                " password: " + password.getValue() +
                                " active: " + active.getValue();
        }

        @Override
        public void writeExternal(ObjectOutput out) throws IOException {
                out.writeInt(getId());
                out.writeBoolean(getActive());
                out.writeObject(getName());
                out.writeObject(getPassword());
        }

        @Override
        public void readExternal(ObjectInput in) throws IOException,
            ↪ ClassNotFoundException {
                setId(in.readInt());
                setActive(in.readBoolean());
                setName((String) in.readObject());
                setPassword((String) in.readObject());
        }
}
```

Bet.java

```
package swe4.models;

import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;

import javafx.beans.property.BooleanProperty;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.SimpleBooleanProperty;
import javafx.beans.property.SimpleIntegerProperty;

public class Bet implements Externalizable {

        private User user;
        private Team teamA;
        private Team teamB;
        private IntegerProperty id;
        private IntegerProperty goalsTeamA;
        private IntegerProperty goalsTeamB;


        public Bet() {
                this.id = new SimpleIntegerProperty(-1);
                this.goalsTeamA = new SimpleIntegerProperty(-1);
                this.goalsTeamB = new SimpleIntegerProperty(-1);
        }

```

```
	public Bet(User user,Team teamA, Team teamB) {
		this();

		this.setUser(user);
		this.setTeamA(teamA);
		this.setTeamB(teamB);
	}

	public Bet(Integer id, User user,Team teamA, Team teamB, Integer goalsTeamA,
	    ↪ Integer goalsTeamB) {
		this(user, teamA, teamB);

		this.id.set(id);
		this.goalsTeamA.set(goalsTeamA);
		this.goalsTeamB.set(goalsTeamB);
	}


	public void clone(Bet game) {
		if(game != null) {
			this.setId(game.getId());
			this.setTeamA(game.getTeamA());
			this.setTeamB(game.getTeamB());
			this.setGoalsTeamA(game.getGoalsTeamA());
			this.setGoalsTeamB(game.getGoalsTeamB());

		} else {
			this.setId(-1);
			this.setTeamA(new Team());
			this.setTeamB(new Team());
			this.setGoalsTeamA(-1);
			this.setGoalsTeamB(-1);
		}
	}

	public User getUser() {
		return user;
	}

	public void setUser(User user) {
		this.user = user;
	}

	public Team getTeamA() {
		return teamA;
	}
	public void setTeamA(Team teamA) {
		this.teamA = teamA;
	}
	public Team getTeamB() {
		return teamB;
	}
	public void setTeamB(Team teamB) {
        this.teamB = teamB;
    }


    public IntegerProperty getIdProperty() {
        return id;
    }

    public final Integer getId() {
        return id.get();
    }

    public final void setId(Integer id) {
        this.id.set(id);
    }



    public IntegerProperty getGoalsTeamAProperty() {
        return goalsTeamA;
    }

    public final Integer getGoalsTeamA() {
        return goalsTeamA.get();
    }

    public final void setGoalsTeamA(Integer goals) {
        this.goalsTeamA.set(goals);
    }



    public IntegerProperty getGoalsTeamBProperty() {
        return goalsTeamB;
    }

    public final Integer getGoalsTeamB() {
        return goalsTeamB.get();
    }

    public final void setGoalsTeamB(Integer goals) {
        this.goalsTeamB.set(goals);
    }



    @Override
    public String toString() {
        return "id: " + this.getId() + " "
                + this.teamA.getName() + " - " + this.teamB.getName().
                  ↪ toString()
                + " " + this.getGoalsTeamA() + ":" + this.getGoalsTeamB
                  ↪ ()
```

```
                                + " user: " + this.getUser();
        }



        @Override
        public void writeExternal(ObjectOutput out) throws IOException {
                out.writeObject(getUser());
                out.writeObject(getTeamA());
                out.writeObject(getTeamB());
                out.writeInt(getId());
                out.writeInt(getGoalsTeamA());
                out.writeInt(getGoalsTeamB());
        }

        @Override
        public void readExternal(ObjectInput in) throws IOException,
            ↪ ClassNotFoundException {
                setUser((User) in.readObject());
                setTeamA((Team) in.readObject());
                setTeamB((Team) in.readObject());
                setId(in.readInt());
                setGoalsTeamA(in.readInt());
                setGoalsTeamB(in.readInt());
        }
}
```

1.2.3 swe4.gui.panels

GameDetailPanel.java

```
package swe4.gui.panels;

import java.util.ArrayList;
import java.util.List;
import javafx.beans.binding.Bindings;
import javafx.beans.binding.BooleanBinding;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import swe4.gui.components.PanelComponent;
import swe4.models.Game;
import swe4.models.Team;
import swe4.rmi.client.EuroBetDbFactory;

public class GameDetailPanel extends PanelComponent {
```

```java


        private ComboBox<String> teamABox;
        private ComboBox<String> teamBBox;

        private BooleanBinding disableSaveCondition;
        private Game editedGame = new Game();
        private List<Team> allTeams;

        public GameDetailPanel() {
                super("Details");


                createDetailFields();

                disableSaveCondition = Bindings.or(
                                teamABox.getSelectionModel().selectedItemProperty().
                                    ↪ isEqualTo(new Team().getNameProperty()),
                                teamBBox.getSelectionModel().selectedItemProperty().
                                    ↪ isEqualTo(new Team().getNameProperty())
                );

                addHeaderButtonRight("game-save-button", "/save.png",
                    ↪ disableSaveCondition, event -> {
                        // save or update game

                        Integer teamASelectionIndex =  teamABox.getSelectionModel().
                            ↪ getSelectedIndex();
                        editedGame.setTeamA(allTeams.get(teamASelectionIndex));


                        Integer teamBSelectionIndex =  teamBBox.getSelectionModel().
                            ↪ getSelectedIndex();
                        editedGame.setTeamB(allTeams.get(teamBSelectionIndex));


                        if(editedGame.getId() != -1) {
                                // update game

                                EuroBetDbFactory.getInstance().updateGame(editedGame);


                                System.out.println("Existing game updated: " +
                                    ↪ editedGame.toString());
                        } else {
                                // save new game

                                Integer newId = EuroBetDbFactory.getInstance().addGame(
                                    ↪ editedGame);
                                editedGame.setId(newId);
                                System.out.println("New game saved: " + editedGame.
                                    ↪ toString());

                        }

                });



        }

        private void createDetailFields() {

                VBox details = new VBox();
                details.setSpacing(50);
                details.setId("game-detail-container");


                allTeams = EuroBetDbFactory.getInstance().getTeams();
                List<String> teamNames = new ArrayList<>();
                for(Team t: allTeams) {
                        teamNames.add(t.getName());
                }



                // team a
                VBox groupA = new VBox();
                groupA.setSpacing(3);
                Label groupATitle = new Label("Team A:");

        ObservableList<String> optionsA = FXCollections.observableList(teamNames);


                teamABox = new ComboBox<String>(optionsA);


                teamABox.setMinWidth(340);
                groupA.getChildren().addAll(groupATitle, teamABox);
                VBox.setMargin(groupA, new Insets(60, 30, 0, 30));


                // team b
                VBox groupB = new VBox();
                groupB.setSpacing(3);
                Label groupBTitle = new Label("Team B:");
                ObservableList<String> optionsB = FXCollections.observableList(
                    ↪ teamNames);


                teamBBox = new ComboBox<>(optionsB);
                teamBBox.setMinWidth(340);
                groupB.getChildren().addAll(groupBTitle, teamBBox);
                VBox.setMargin(groupB, new Insets(0, 30, 0, 30));


                details.getChildren().addAll(
                                groupA, groupB
                );

                this.createNewDetail();

                addContent(details);

        }

        public void setDetails(Game game) {
                editedGame.clone(game);
                teamABox.getSelectionModel().select(game.getTeamA().getName());
                teamBBox.getSelectionModel().select(game.getTeamB().getName());
        }

        public void createNewDetail() {
                editedGame.clone(new Game());
                editedGame.setTeamA(new Team());
                editedGame.setTeamB(new Team());
                teamABox.getSelectionModel().select("");
                teamBBox.getSelectionModel().select("");
        }
}
```

GamesPanel.java

```
package swe4.gui.panels;

import java.util.ArrayList;
import java.util.List;

import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.control.ListCell;
import javafx.scene.control.ListView;
import javafx.scene.layout.Border;
import javafx.scene.layout.BorderStroke;
import javafx.scene.layout.BorderStrokeStyle;
import javafx.scene.paint.Color;
import javafx.util.Callback;
import swe4.gui.components.PanelComponent;
import swe4.models.Game;
import swe4.rmi.client.EuroBetDbFactory;
import swe4.rmi.client.GameAddedListener;
import swe4.rmi.client.GameUpdatedListener;


```

```
public class GamesPanel extends PanelComponent{

        private static final Border LIST_BORDER = new Border(
                        new BorderStroke(Color.TRANSPARENT, BorderStrokeStyle.NONE,
                            ↪ null, null)
                );
        private ListView<Game> gamesListView;

        private List<GamesPanelListener> listeners = new ArrayList<GamesPanelListener
            ↪ >();

        public GamesPanel() {
                super("All Games");


                addHeaderButtonLeft("games-add-button", "/plus.png", event -> {

                        // call all added listeners
            for (GamesPanelListener l : listeners) {
                l.onAddNewGame();
            }
                });


                createGamesList();


                EuroBetDbFactory.getInstance().addGamesAddedListener(new
                    ↪ GameAddedListener() {

                        @Override
                        public void onGameAdded(Game game) {
                                gamesListView = null;
                                createGamesList();
                        }
                });

                EuroBetDbFactory.getInstance().addGamesUpdatedListener(new
                    ↪ GameUpdatedListener() {

                        @Override
                        public void onGameUpdated(Game game) {
                                gamesListView = null;
                                createGamesList();
                        }
                });
        }


        private void createGamesList() {
                gamesListView = new ListView<Game>();
                gamesListView.setBorder(LIST_BORDER);

```

```java

                List<Game> games = EuroBetDbFactory.getInstance().getGames();
                ObservableList<Game> items = FXCollections.observableList(games);
        gamesListView.setItems(items);

        gamesListView.setCellFactory(new Callback<ListView<Game>, ListCell<Game>>(){

            @Override
            public ListCell<Game> call(ListView<Game> p) {

                ListCell<Game> cell = new ListCell<Game>(){

                    @Override
                    protected void updateItem(Game t, boolean bln) {
                        super.updateItem(t, bln);
                        if (t != null) {
                                setText(t.getTeamA().getName() + " - " + t.getTeamB().
                                    ↪ getName());
                        }
                    }
                };
                return cell;
            }
        });

        gamesListView.getSelectionModel().selectedItemProperty()
                .addListener(new ChangeListener<Game>() {

                @Override
                public void changed(ObservableValue<? extends Game> observable,
                                Game oldValue, Game newValue) {

                // call all added listeners
                for (GamesPanelListener l : listeners) {
                    l.onGameSelected(newValue);
                }
                }
        });

        addContent(gamesListView);
        }


        public void addListener(GamesPanelListener listener) {
                listeners.add(listener);
    }


}
```

GamesPanelListener.java

```java
package swe4.gui.panels;

import swe4.models.Game;

public interface GamesPanelListener {
    void onGameSelected(Game game);
    void onAddNewGame();
}
```

ResultDetailPanel.java

```java
package swe4.gui.panels;

import java.util.ArrayList;
import java.util.List;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import swe4.gui.NumberTextField;
import swe4.gui.components.PanelComponent;
import swe4.models.Game;
import swe4.rmi.client.EuroBetDbFactory;

public class ResultDetailPanel extends PanelComponent {


        private NumberTextField goalsTeamABox;
        private NumberTextField goalsTeamBBox;
        private ComboBox<String> gameBox;
        private Game editedGame = new Game();
        private List<Game> allGames;

        public ResultDetailPanel() {
                super("Details");

                addHeaderButtonRight("result-save-button", "/save.png", null, event ->
                    ↪ {
                        // save or update game

                        Integer gameSelectionIndex =  gameBox.getSelectionModel().
                            ↪ getSelectedIndex();

                        Game selectedGame = allGames.get(gameSelectionIndex);

                        editedGame.setTeamA(selectedGame.getTeamA());
                        editedGame.setTeamB(selectedGame.getTeamB());

                    editedGame.setGoalsTeamA(Integer.parseInt(goalsTeamABox.getText
                        ↪ ()));
                    editedGame.setGoalsTeamB(Integer.parseInt(goalsTeamBBox.getText
                        ↪ ()));
                    editedGame.setGameFinished(true);

                    if(editedGame.getId() != -1) {
                            // update game

                            EuroBetDbFactory.getInstance().updateGame(editedGame);


                            System.out.println("Existing game updated: " +
                                ↪ editedGame.toString());
                    } else {
                            // save new game

                            Integer newId = EuroBetDbFactory.getInstance().addGame(
                                ↪ editedGame);
                            editedGame.setId(newId);
                            System.out.println("New game saved: " + editedGame.
                                ↪ toString());

                    }
            });

            createDetailFields();
    }

    private void createDetailFields() {

            VBox details = new VBox();
            details.setSpacing(50);
            details.setId("result-detail-container");



            allGames = EuroBetDbFactory.getInstance().getGames();
            List<String> gameNames = new ArrayList<>();
            for(Game t: allGames) {
                    gameNames.add(t.getTeamA().getName() + " - " + t.getTeamB().
                        ↪ getName());
            }

    // games
    VBox game = new VBox();
            game.setSpacing(3);
            Label gameTitle = new Label("Game:");

            ObservableList<String> optionsB = FXCollections.observableList(
                ↪ gameNames);

            gameBox = new ComboBox<String>(optionsB);
            gameBox.setMinWidth(340);
            game.getChildren().addAll(gameTitle, gameBox);
            VBox.setMargin(game, new Insets(30, 30, 0, 30));


            // goals team a
    VBox goalsA = new VBox();
    goalsA.setSpacing(3);
            Label goalsATitle = new Label("Goals Team A:");
            goalsTeamABox = new NumberTextField ();
            goalsA.getChildren().addAll(goalsATitle, goalsTeamABox);
            VBox.setMargin(goalsA, new Insets(0, 30, 0, 30));


            // goals team b
    VBox goalsB = new VBox();
    goalsB.setSpacing(3);
            Label goalsBTitle = new Label("Goals Team B:");
            goalsTeamBBox = new NumberTextField ();
            goalsA.getChildren().addAll(goalsBTitle, goalsTeamBBox);
            VBox.setMargin(goalsB, new Insets(0, 30, 0, 30));

            details.getChildren().addAll(
                            game,goalsA, goalsB
            );

            addContent(details);

    }

    public void setDetails(Game game) {

            editedGame.clone(game);

            gameBox.getSelectionModel().select(game.getTeamA().getName() + " - " +
                ↪ game.getTeamB().getName());


            if(game.getGameFinished()) {
                    goalsTeamABox.setText(game.getGoalsTeamA().toString());
                    goalsTeamBBox.setText(game.getGoalsTeamB().toString());
            } else {
                    goalsTeamABox.setText("");
                    goalsTeamBBox.setText("");
            }
    }

}
```

ResultsPanel.java

```java
package swe4.gui.panels;

import java.util.ArrayList;
import java.util.List;

import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.control.ListCell;
import javafx.scene.control.ListView;
import javafx.scene.layout.Border;
import javafx.scene.layout.BorderStroke;
import javafx.scene.layout.BorderStrokeStyle;
import javafx.scene.paint.Color;
import javafx.util.Callback;
import swe4.gui.components.PanelComponent;
import swe4.models.Game;
import swe4.rmi.client.EuroBetDbFactory;
import swe4.rmi.client.GameAddedListener;
import swe4.rmi.client.GameUpdatedListener;


public class ResultsPanel extends PanelComponent{

        private static final Border LIST_BORDER = new Border(
                        new BorderStroke(Color.TRANSPARENT, BorderStrokeStyle.NONE,
                            ↪ null, null)
                );

        private  ListView<Game> gamesListView;

        private List<ResultsPanelListener> listeners = new ArrayList<
            ↪ ResultsPanelListener>();

        public ResultsPanel() {
                super("All Results");


                createGamesList();



                EuroBetDbFactory.getInstance().addGamesAddedListener(new
                    ↪ GameAddedListener() {

                        @Override
                        public void onGameAdded(Game game) {
                                gamesListView = null;
                                createGamesList();
                        }
                });

                EuroBetDbFactory.getInstance().addGamesUpdatedListener(new GameUpdatedListener() {

                        @Override
                        public void onGameUpdated(Game game) {
                                gamesListView = null;
                                createGamesList();
                        }
                });

        }


        private void createGamesList() {
                gamesListView = new ListView<Game>();
                gamesListView.setBorder(LIST_BORDER);

        List<Game> games = EuroBetDbFactory.getInstance().getGames();


        ObservableList<Game> items = FXCollections.observableList(games);



        gamesListView.setItems(items);


        gamesListView.setCellFactory(new Callback<ListView<Game>, ListCell<Game>>(){

            @Override
            public ListCell<Game> call(ListView<Game> p) {

                ListCell<Game> cell = new ListCell<Game>(){

                    @Override
                    protected void updateItem(Game g, boolean bln) {
                        super.updateItem(g, bln);
                        if (g != null) {
                                setText(g.getResult() + "    " + g.getTeamA().getName() + " - " + g.getTeamB().getName());
                        }
                    }
                };
                return cell;
            }
        });

        gamesListView.getSelectionModel().selectedItemProperty()
                .addListener(new ChangeListener<Game>() {

                @Override
                public void changed(ObservableValue<? extends Game> observable,
                                Game oldValue, Game newValue) {

                // call all added listeners
                for (ResultsPanelListener l : listeners) {
                    l.onResultSelected(newValue);
                }
                }
        });

        addContent(gamesListView);
        }


        public void addListener(ResultsPanelListener listener) {
                listeners.add(listener);
    }


}
```

ResultsPanelListener.java

```
package swe4.gui.panels;

import swe4.models.Game;

public interface ResultsPanelListener {
    void onResultSelected(Game game);
}
```

TeamDetailPanel.java

```
package swe4.gui.panels;

import java.util.List;

import javafx.beans.binding.Bindings;
import javafx.beans.binding.BooleanBinding;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.VBox;
import swe4.gui.components.PanelComponent;
```

```java
import swe4.models.Team;
import swe4.rmi.client.EuroBetDbFactory;
import swe4.rmi.client.TeamUpdatedListener;

public class TeamDetailPanel extends PanelComponent {


        private TextField nameField;
        private ComboBox<String> groupBox;
        private BooleanBinding disableSaveCondition;
        private Team editedTeam = new Team();


        public TeamDetailPanel() {
                super("Details");



                createDetailFields();


                disableSaveCondition = Bindings.and(
                                nameField.textProperty().isEmpty(),
                                nameField.textProperty().isEmpty()
                );


                addHeaderButtonRight("team-save-button", "/save.png",
                    ↪ disableSaveCondition, event -> {

                        // save or update team

                        if(editedTeam.getId() != -1) {
                                // update team

                                EuroBetDbFactory.getInstance().updateTeam(editedTeam);

                                System.out.println("Existing team updated: " +
                                    ↪ editedTeam.toString());
                        } else {
                                // save new user

                                Integer newId = EuroBetDbFactory.getInstance().addTeam(
                                    ↪ editedTeam);
                                editedTeam.setId(newId);
                                System.out.println("New team saved: " + editedTeam.
                                    ↪ toString());
                        }

                });


                EuroBetDbFactory.getInstance().addTeamUpdatedListener(new
                    TeamUpdatedListener() {

                        @Override
                        public void onTeamUpdated(Team team) {
                                if(editedTeam.getId() == team.getId()) {
                                        // other client has changed current user
                                        editedTeam.clone(team);
                                }
                        }
                });
        }

        private void createDetailFields() {

                VBox details = new VBox();
                details.setSpacing(20);
                details.setId("team-detail-container");



                // team name
                VBox username = new VBox();
                username.setSpacing(3);
                Label nameTitle = new Label("Country:");
                nameField = new TextField ();
                nameField.textProperty().bindBidirectional(editedTeam.getNameProperty()
                    );
                username.getChildren().addAll(nameTitle, nameField);
                VBox.setMargin(username, new Insets(30, 30, 0, 30));


                // team group
                VBox group = new VBox();
                group.setSpacing(3);
                Label groupTitle = new Label("Group:");
                List<String> groups = EuroBetDbFactory.getInstance().getGroups();
        ObservableList<String> options = FXCollections.observableList(groups);
                groupBox = new ComboBox<String>(options);
                groupBox.valueProperty().bindBidirectional(editedTeam.getGroupProperty
                    ());
                group.getChildren().addAll(groupTitle, groupBox);
                VBox.setMargin(group, new Insets(0, 30, 0, 30));


                details.getChildren().addAll(
                                 username, group
                );


                addContent(details);

        }

        public void setDetails(Team team) {

                editedTeam.clone(team);
        }

        public void createNewDetail() {
                editedTeam.clone(new Team());
        }


}
```

TeamsPanel.java

```
package swe4.gui.panels;

import java.util.ArrayList;
import java.util.List;

import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.control.ListCell;
import javafx.scene.control.ListView;
import javafx.scene.layout.Border;
import javafx.scene.layout.BorderStroke;
import javafx.scene.layout.BorderStrokeStyle;
import javafx.scene.paint.Color;
import javafx.util.Callback;
import swe4.gui.components.PanelComponent;
import swe4.models.Team;
import swe4.rmi.client.EuroBetDbFactory;
import swe4.rmi.client.TeamAddedListener;
import swe4.rmi.client.TeamUpdatedListener;


public class TeamsPanel extends PanelComponent{

        private static final Border LIST_BORDER = new Border(
                        new BorderStroke(Color.TRANSPARENT, BorderStrokeStyle.NONE,
                            ↪ null, null)
                );
        private ListView<Team> teamListView;

        private List<TeamsPanelListener> listeners = new ArrayList<TeamsPanelListener
            ↪ >();

        public TeamsPanel() {
                super("All Teams");


                addHeaderButtonLeft("teams-add-button", "/plus.png", event -> {
                        // call all added listeners
            for (TeamsPanelListener l : listeners) {
                l.onAddNewTeam();
            }
                });


                createTeamsList();

                EuroBetDbFactory.getInstance().addTeamAddedListener(new
                    ↪ TeamAddedListener() {

                        @Override
                        public void onTeamAdded(Team team) {
                                teamListView = null;
                                createTeamsList();
                        }
                });

                EuroBetDbFactory.getInstance().addTeamUpdatedListener(new
                    ↪ TeamUpdatedListener() {

                        @Override
                        public void onTeamUpdated(Team team) {
                                teamListView = null;
                                createTeamsList();
                        }
                });

        }


        private void createTeamsList() {
                teamListView = new ListView<Team>();
                teamListView.setBorder(LIST_BORDER);

        List<Team> teams = EuroBetDbFactory.getInstance().getTeams();



        ObservableList<Team> items = FXCollections.observableList(teams);


        teamListView.setItems(items);


        teamListView.setCellFactory(new Callback<ListView<Team>, ListCell<Team>>(){

            @Override
            public ListCell<Team> call(ListView<Team> p) {

                ListCell<Team> cell = new ListCell<Team>(){

                    @Override
                    protected void updateItem(Team t, boolean bln) {
                        super.updateItem(t, bln);
                        if (t != null) {
                                setText(t.getName());
                        }
                    }
                };
                return cell;
            }
        });

        teamListView.getSelectionModel().selectedItemProperty()
                                .addListener(new ChangeListener<Team>() {

                @Override
                public void changed(ObservableValue<? extends Team> observable,
                                Team oldValue, Team newValue) {

                // call all added listeners
                for (TeamsPanelListener l : listeners) {
                    l.onTeamSelected(newValue);
                }
                }
        });

        addContent(teamListView);
        }


        public void addListener(TeamsPanelListener listener) {
                listeners.add(listener);
    }


}
```

TeamsPanelListener.java

```
package swe4.gui.panels;

import swe4.models.Team;

public interface TeamsPanelListener {
    void onTeamSelected(Team team);
    void onAddNewTeam();
}
```

UserDetailPanel.java

```
package swe4.gui.panels;

import java.io.Serializable;
import java.rmi.Remote;
import javafx.beans.binding.Bindings;
import javafx.beans.binding.BooleanBinding;
import javafx.geometry.Insets;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.layout.VBox;
import swe4.gui.ToggleSwitch;
import swe4.gui.components.PanelComponent;
import swe4.models.User;
import swe4.rmi.client.EuroBetDbFactory;
import swe4.rmi.client.UserUpdatedListener;

public class UserDetailPanel extends PanelComponent implements Serializable, Remote  {



        private static final long serialVersionUID = -7267372017525019410L;
        private ToggleSwitch switchActiv;
        private TextField nameField;
        private PasswordField passwordField;
        private PasswordField confirmField;
        private BooleanBinding disableSaveCondition;

        private User editedUser = new User();

        public UserDetailPanel() {
                super("Details");


                createDetailFields();

                disableSaveCondition = Bindings.or(
                                passwordField.textProperty().isNotEqualTo(confirmField.
                                    ↪ textProperty()),
                                nameField.textProperty().isEmpty()
                );

                addHeaderButtonRight("users-save-button", "/save.png",
                    ↪ disableSaveCondition, event -> {


```

```java
                        // save or update user

                        if(editedUser.getId() != -1) {
                                // update user

                                EuroBetDbFactory.getInstance().updateUser(editedUser);

                                System.out.println("Existing user updated: " + editedUser.toString());
                        } else {
                                // save new user

                                Integer newId = EuroBetDbFactory.getInstance().addUser(editedUser);
                                editedUser.setId(newId);
                                System.out.println("New user saved: " + editedUser.toString());
                        }
                });



                EuroBetDbFactory.getInstance().addUserUpdatedListener(new UserUpdatedListener() {

                        @Override
                        public void onUserUpdated(User user) {
                                if(editedUser.getId() == user.getId()) {
                                        // other client has changed current user
                                        editedUser.clone(user);
                                }
                        }
                });



        }

        private void createDetailFields() {

                VBox details = new VBox();
                details.setSpacing(20);
                details.setId("user-detail-container");


                // active
                VBox active = new VBox();
                Label activTitle = new Label("Active:");
                switchActiv = new ToggleSwitch();
                switchActiv.switchOnProperty().bindBidirectional(editedUser.getActiveProperty());

                VBox switchPane = new VBox();
                switchPane.getChildren().add(switchActiv);
                VBox.setMargin(switchPane, new Insets(-20, 0, 0, 250));
                active.getChildren().addAll(activTitle, switchPane);
                VBox.setMargin(active, new Insets(20, 30, 0, 30));


                // username
                VBox username = new VBox();
                username.setSpacing(3);
                Label nameTitle = new Label("Username:");
                nameField = new TextField ();
                nameField.textProperty().bindBidirectional(editedUser.getNameProperty());
                username.getChildren().addAll(nameTitle, nameField);
                VBox.setMargin(username, new Insets(0, 30, 0, 30));


                // password
                VBox password = new VBox();
                password.setSpacing(3);
                Label passwordTitle = new Label("Password:");
                passwordField = new PasswordField ();
                passwordField.textProperty().bindBidirectional(editedUser.getPasswordProperty());
                Label confirmTitle = new Label("Confirm:");
                confirmField = new PasswordField ();
                password.getChildren().addAll(passwordTitle, passwordField,
                                confirmTitle, confirmField);
                VBox.setMargin(password, new Insets(0, 30, 0, 30));

                details.getChildren().addAll(
                                active, username, password
                );


                addContent(details);

        }

        public void setDetails(User user) {

                editedUser.clone(user);
                confirmField.setText(editedUser.getPassword());
        }

        public void createNewDetail() {

                editedUser.clone(new User());
                confirmField.setText(editedUser.getPassword());
        }



```

```
}
```

UsersPanel.java

```
package swe4.gui.panels;

import java.util.ArrayList;
import java.util.List;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.control.ListCell;
import javafx.scene.control.ListView;
import javafx.scene.layout.Border;
import javafx.scene.layout.BorderStroke;
import javafx.scene.layout.BorderStrokeStyle;
import javafx.scene.paint.Color;
import javafx.util.Callback;
import swe4.gui.components.PanelComponent;
import swe4.models.User;
import swe4.rmi.client.EuroBetDbFactory;
import swe4.rmi.client.UserAddedListener;
import swe4.rmi.client.UserUpdatedListener;


public class UsersPanel extends PanelComponent {


        private static final Border LIST_BORDER = new Border(
                        new BorderStroke(Color.TRANSPARENT, BorderStrokeStyle.NONE,
                            ↪ null, null)
                );


        private ListView<User> personListView;
        private List<UsersPanelListener> listeners = new ArrayList<>();



        public UsersPanel() {
                super("All Users");

                addHeaderButtonLeft("users-add-button", "/plus.png", event -> {
                        // call all added listeners
            for (UsersPanelListener l : listeners) {
                l.onAddNewUser();
            }
                });
                createUsersList();



                EuroBetDbFactory.getInstance().addUserAddedListener(new
                    ↪ UserAddedListener() {

                        @Override
                        public void onUserAdded(User user) {
                                personListView = null;
                                createUsersList();
                        }
                });

                EuroBetDbFactory.getInstance().addUserUpdatedListener(new
                    ↪ UserUpdatedListener() {

                        @Override
                        public void onUserUpdated(User user) {
                                personListView = null;
                                createUsersList();
                        }
                });
        }

        private void createUsersList() {
                personListView = new ListView<User>();
        personListView.setBorder(LIST_BORDER);


        List<User> users = EuroBetDbFactory.getInstance().getUsers();

        ObservableList<User> items = FXCollections.observableList(users);


        personListView.setItems(items);


        personListView.setCellFactory(new Callback<ListView<User>, ListCell<User>>(){

            @Override
            public ListCell<User> call(ListView<User> p) {

                ListCell<User> cell = new ListCell<User>(){

                    @Override
                    protected void updateItem(User t, boolean bln) {
                        super.updateItem(t, bln);
                        if (t != null) {
                                setText(t.getName());
                        }
                    }
                };
                return cell;
            }
        });

        personListView.getSelectionModel().selectedItemProperty()
                                .addListener(new ChangeListener<User>() {

                @Override
                public void changed(ObservableValue<? extends User> observable,
                                User oldValue, User newValue) {

                // call all added listeners
                for (UsersPanelListener l : listeners) {
                    l.onUserSelected(newValue);
                }
                }
        });

        addContent(personListView);
        }


        public void addListener(UsersPanelListener listener) {
                listeners.add(listener);
    }


}
```

UsersPanelListener.java

```
package swe4.gui.panels;

import swe4.models.User;

public interface UsersPanelListener {
    void onUserSelected(User user);
    void onAddNewUser();
}
```

1.2.4 swe4.rmi.client

EuroBetDbFactory.java

```
package swe4.rmi.client;

```

```java

import java.io.Serializable;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.List;

import swe4.models.Bet;
import swe4.models.Game;
import swe4.models.Team;
import swe4.models.User;
import swe4.rmi.server.EuroBetDb;
import swe4.util.Util;


public class EuroBetDbFactory implements Serializable, Remote  {


        private static final long serialVersionUID = 1944025441928560415L;
        private static EuroBetDbFactory instance = null;
        private static EuroBetDb db;

        private List<UserAddedListener> userAddedListeners = new ArrayList<>();
        private List<UserUpdatedListener> userUpdatedListeners = new ArrayList<>();
        private List<TeamAddedListener> teamAddedListeners = new ArrayList<>();
        private List<TeamUpdatedListener> teamUpdatedListeners = new ArrayList<>();
        private List<GameAddedListener> gamesAddedListeners = new ArrayList<>();
        private List<GameUpdatedListener> gamesUpdatedListeners = new ArrayList<>();
        private List<BetAddedListener> betsAddedListeners = new ArrayList<>();
        private List<BetUpdatedListener> betsUpdatedListeners = new ArrayList<>();


        private EuroBetDbFactory() throws MalformedURLException, RemoteException,
            ↪ NotBoundException {

                String hostPort = Util.getHostPortArg(null);

                db = (EuroBetDb) Naming.lookup("rmi://" + hostPort + "/EuroBetDb");

        }

        public static EuroBetDbFactory getInstance() {
                if(instance == null) {

                        try {
                                instance = new EuroBetDbFactory();
                                System.out.println("Register factory: " + instance);
                                db.registerForNotification(instance);
                        } catch (MalformedURLException | RemoteException |
                            ↪ NotBoundException e) {
                                System.out.println("Error: Unable to connect to server! ");
                        }
            }
            return instance;
        }

        public void removeAllUsers() {
                try {
                        db.removeAllUsers();
                } catch (RemoteException e) {
                        System.out.println("Error: Unable to remove users from server!");
                }
        }
        public List<User> getUsers() {
                try {
                        return db.getUsers();
                } catch (RemoteException e) {
                        System.out.println("Error: Unable to fetch users from server!");
                        return null;
                }
        }


        public void updateUser(User user) {
                try {
                        db.updateUser(user);
                        onUserUpdated(user);
                } catch (RemoteException e) {
                        System.out.println("Error: Unable to update user at server!");
                }
        }


        public Integer addUser(User user) {
                try {
                        Integer newId =  db.addUser(user);
                        user.setId(newId);
                        onUserAdded(user);
                        return user.getId();
                } catch (RemoteException e) {
                        System.out.println("Error: Unable to add user at server!");
                        return -1;
                }
        }


        public void addUserAddedListener(UserAddedListener listener) {
                userAddedListeners.add(listener);
    }

```

```
        public void addUserUpdatedListener(UserUpdatedListener listener) {
                userUpdatedListeners.add(listener);
    }


        public void onUserAdded(User user) throws RemoteException {
                for(UserAddedListener l : userAddedListeners) {
                        l.onUserAdded(user);
                }
        }


        public void onUserUpdated(User user) throws RemoteException {
                for(UserUpdatedListener l : userUpdatedListeners) {
                        l.onUserUpdated(user);
                }
        }





        public void removeAllGroups() {
                try {
                        db.removeAllGroups();
                } catch (RemoteException e) {
                        System.out.println("Error: Unable to remove groups from server!
                            ↪ ");
                }
        }


        public Integer addGroup(Character group) {

                try {
                        return db.addGroup(group);
                } catch (RemoteException e) {
                        System.out.println("Error: Unable to add group to server!");
                        return null;
                }
        }

        public List<String> getGroups() {
                try {
                        return db.getGroups();
                } catch (RemoteException e) {
                        System.out.println("Error: Unable to fetch groups from server!"
                            ↪ );
                        return null;
                }
        }


```

```java

        public void removeAllTeams() {
                try {
                        db.removeAllTeams();
                } catch (RemoteException e) {
                        System.out.println("Error: Unable to remove teams from server!"
                            ↪ );
                }
        }

        public List<Team> getTeams() {
                try {
                        return db.getTeams();
                } catch (RemoteException e) {
                        System.out.println("Error: Unable to fetch teams from server!")
                            ↪ ;
                        return null;
                }
        }

        public void updateTeam(Team team) {
                try {
                        db.updateTeam(team);
                        onTeamUpdated(team);
                } catch (RemoteException e) {
                        System.out.println("Error: Unable to update team at server!");
                }
        }

        public Integer addTeam(Team team) {
                try {
                        Integer newId =  db.addTeam(team);
                        team.setId(newId);
                        onTeamAdded(team);
                        return team.getId();
                } catch (RemoteException e) {
                        System.out.println("Error: Unable to add team at server!");
                        return -1;
                }
        }


        public void onTeamAdded(Team team) throws RemoteException {
                for(TeamAddedListener l : teamAddedListeners) {
                        l.onTeamAdded(team);
                }
        }


        public void onTeamUpdated(Team team) throws RemoteException {
                for(TeamUpdatedListener l : teamUpdatedListeners) {
                        l.onTeamUpdated(team);
                }
```

```
	}


	public void addTeamAddedListener(TeamAddedListener listener) {
		teamAddedListeners.add(listener);
    }

	public void addTeamUpdatedListener(TeamUpdatedListener listener) {

		teamUpdatedListeners.add(listener);
    }






	public void removeAllGames() {
		try {
			db.removeAllGames();
		} catch (RemoteException e) {
			System.out.println("Error: Unable to remove games from server!"
			    ↪ );
		}
	}

	public List<Game> getGames() {
		try {
			return db.getGames();
		} catch (RemoteException e) {
			System.out.println("Error: Unable to fetch games from server!")
			    ↪ ;
			return null;
		}
	}

	public void updateGame(Game game) {
		try {
			db.updateGame(game);
			onGameUpdated(game);
		} catch (RemoteException e) {
			System.out.println("Error: Unable to update team at server!");
		}
	}

	public Integer addGame(Game game) {
		try {
			Integer newId =  db.addGame(game);
			game.setId(newId);
			onGameAdded(game);
			return game.getId();
		} catch (RemoteException e) {
			System.out.println("Error: Unable to add team at server!");
```

```java
				return -1;
			}
		}


		public void onGameAdded(Game game) throws RemoteException {
			for(GameAddedListener l : gamesAddedListeners) {
				l.onGameAdded(game);
			}
		}


		public void onGameUpdated(Game game) throws RemoteException {
			for(GameUpdatedListener l : gamesUpdatedListeners) {
				l.onGameUpdated(game);
			}
		}

		public void addGamesAddedListener(GameAddedListener listener) {
			gamesAddedListeners.add(listener);
	}

		public void addGamesUpdatedListener(GameUpdatedListener listener) {

			gamesUpdatedListeners.add(listener);
	}




		public void removeAllBets() {
			try {
				db.removeAllBets();
			} catch (RemoteException e) {
				System.out.println("Error: Unable to remove bets from server!")
				    ↪ ;
			}
		}

		public List<Bet> getBets() {
			try {
				return db.getBets();
			} catch (RemoteException e) {
				System.out.println("Error: Unable to fetch bets from server!");
				return null;
			}
		}

		public void updateBet(Bet bet) {
			try {
				db.updateBet(bet);
			} catch (RemoteException e) {
				System.out.println("Error: Unable to update bet at server!");
```

```
                }
        }

        public Integer addBet(Bet bet) {
                try {
                        Integer newId =  db.addBet(bet);
                        bet.setId(newId);
                        return bet.getId();
                } catch (RemoteException e) {
                        System.out.println("Error: Unable to add bet at server!");
                        return -1;
                }
        }


        public void onBetAdded(Bet bet) throws RemoteException {
                for(BetAddedListener l : betsAddedListeners) {
                        l.onBetAdded(bet);
                }
        }


        public void onBetUpdated(Bet bet) throws RemoteException {
                for(BetUpdatedListener l : betsUpdatedListeners) {
                        l.onBetUpdated(bet);
                }
        }

        public void addBetsAddedListener(BetAddedListener listener) {
                betsAddedListeners.add(listener);
    }

        public void addBetsUpdatedListener(BetUpdatedListener listener) {

                betsUpdatedListeners.add(listener);
    }


}
```

BetAddedListener.java

```
package swe4.rmi.client;

import swe4.models.Bet;

public interface BetAddedListener {
        public void onBetAdded(Bet bet);

}
```

BetUpdatedListener.java

```java
package swe4.rmi.client;

import swe4.models.Bet;

public interface BetUpdatedListener {
        public void onBetUpdated(Bet bet);

}
```

GameAddedListener.java

```java
package swe4.rmi.client;

import swe4.models.Game;

public interface GameAddedListener {
        public void onGameAdded(Game game);

}
```

GameUpdatedListener.java

```java
package swe4.rmi.client;

import swe4.models.Game;

public interface GameUpdatedListener {
        public void onGameUpdated(Game game);

}
```

TeamAddedListener.java

```java
package swe4.rmi.client;

import swe4.models.Team;

public interface TeamAddedListener {
        public void onTeamAdded(Team team);
}
```

TeamUpdatedListener.java

```java
package swe4.rmi.client;

import swe4.models.Team;

public interface TeamUpdatedListener {
        public void onTeamUpdated(Team team);

}
```

UserAddedListener.java

```java
package swe4.rmi.client;

import swe4.models.User;

public interface UserAddedListener {
        public void onUserAdded(User user);

}
```

UserUpdatedListener.java

```java
package swe4.rmi.client;

import swe4.models.User;

public interface UserUpdatedListener {
        public void onUserUpdated(User user);

}
```

1.2.5 swe4.rmi.server

EuroBetDb.java

```java
package swe4.rmi.server;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

import swe4.models.Bet;
import swe4.models.Game;
import swe4.models.Team;
import swe4.models.User;
import swe4.rmi.client.EuroBetDbFactory;

public interface EuroBetDb extends Remote {

        /**
         * removes all users from database
         * @throws RemoteException
         */
        void removeAllUsers() throws RemoteException;

        /**
         * get all users from database
         * @return
         * @throws RemoteException
         */
        List<User> getUsers() throws RemoteException;


        /**
         * updates an existing user in database
         * @param user
         * @throws RemoteException
         */
        void updateUser(User user) throws RemoteException;


        /**
         * creates a new user in database
         * @param user
         * @return id from created user
         * @throws RemoteException
         */
        Integer addUser(User user) throws RemoteException;


        /**
         * removes all groups from database
         * @throws RemoteException
         */
        void removeAllGroups() throws RemoteException;


        /**
         * creates a new group in database
         * @param group
         * @return group name
         * @throws RemoteException
         */
        Integer addGroup(Character group) throws RemoteException;


        /**
         * get all groups from database
         * @return
         * @throws RemoteException
         */
        List<String> getGroups() throws RemoteException;



        /**
         * removes all teams from database
         * @throws RemoteException
         */
        void removeAllTeams() throws RemoteException;

        /**
         * get all teams from database
         * @return
         * @throws RemoteException
         */
        List<Team> getTeams() throws RemoteException;



        /**
         * updates an existing team in database
         * @param user
         * @throws RemoteException
         */
        void updateTeam(Team team) throws RemoteException;


        /**
         * creates a new team in database
         * @param team
         * @return id from created team
         * @throws RemoteException
         */
        Integer addTeam(Team team) throws RemoteException;


        /**
         * removes all games from database
         * @throws RemoteException
         */
        void removeAllGames() throws RemoteException;

        /**
```

```
         * get all games from database
         * @return
         * @throws RemoteException
         */
        List<Game> getGames() throws RemoteException;


        /**
         * updates an existing team in database
         * @param user
         * @throws RemoteException
         */
        void updateGame(Game game) throws RemoteException;


        /**
         * creates a new game in database
         * @param game
         * @return id from created game
         * @throws RemoteException
         */
        Integer addGame(Game game) throws RemoteException;



        /**
         * removes all bets from database
         * @throws RemoteException
         */
        void removeAllBets() throws RemoteException;

        /**
         * get all games from database
         * @return
         * @throws RemoteException
         */
        List<Bet> getBets() throws RemoteException;


        /**
         * creates a new bet in database
         * @param bet
         * @return id from created bet
         * @throws RemoteException
         */
        Integer addBet(Bet bet) throws RemoteException;


        /**
         * updates an existing bet in database
         * @param bet
         * @throws RemoteException
         */
```

```java
        void updateBet(Bet bet) throws RemoteException;



        // called by clients to register for new server callbacks
        public void registerForNotification(EuroBetDbFactory n) throws RemoteException;
}
```

EuroBetDbServer.java

```java
package swe4.rmi.server;

import java.sql.*;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.List;

import swe4.models.Bet;
import swe4.models.Game;
import swe4.models.Team;
import swe4.models.User;
import swe4.rmi.client.EuroBetDbFactory;
import swe4.util.Util;



public class EuroBetDbServer implements EuroBetDb {


        private static final String DB_URL = "127.0.0.1";
        private static final String DB_PORT = "3306";
        private static final String DB_NAME = "EuroBetDb";
        private static final String DB_USER = "root";
        private static final String DB_PASSWORD = "root";

        private List<EuroBetDbFactory> clientList = new ArrayList<>();


        public EuroBetDbServer() throws RemoteException {
                super();
        }



        @Override
        public synchronized List<User> getUsers() throws RemoteException {
```

```java

        List<User> users = new ArrayList<>();
        ResultSet result = executeSqlSelect("SELECT * FROM users");

        System.out.println("--> getUsers");
        try {
            while(result.next()) {

                User user = new User(
                        result.getInt("idusers"),
                        result.getBoolean("active"),
                        result.getString("name"),
                        result.getString("password")
                );
                users.add(user);
                System.out.println("     -> User: " + user.toString());
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    return users;
    }



    @Override
    public void updateUser(User user) throws RemoteException {

        System.out.println("--> updateUser");
        System.out.println("     -> User: " + user.toString());

        executeSqlUpdate(
                "UPDATE users SET"
                        + " name = '" + user.getName() + "',"
                        + " password = '" + user.getPassword() + "', "
                        + " active = " + user.getActive()
                    + " WHERE idusers = " + user.getId()
                    + ";");

        notifiyOnUserUpdated(user);
    }


    @Override
    public Integer addUser(User user) throws RemoteException {

        System.out.println("--> addUser");
        System.out.println("     -> User: " + user.toString());

        Integer newId = executeSqlUpdate(
                "INSERT INTO users VALUES("
                        + " null, "
```

```
                                                + " '" + user.getName() + "',"
                                                + " '" + user.getPassword() + "', "
                                                + " " + user.getActive()
                                        + ");");
                user.setId(newId);
                notifiyOnUserAdded(user);
                return user.getId();
        }

        @Override
        public Integer addGroup(Character group) throws RemoteException {
                String groupStr = group.toString();
                System.out.println("--> addGroup");
                System.out.println("     -> group: " + groupStr);

                Integer newId = executeSqlUpdate(
                                "INSERT INTO groups VALUES("
                                                + " null, "
                                                + " '" + groupStr + "'"
                                        + ");");

                return newId;
        }

        @Override
        public List<String> getGroups() throws RemoteException {
                List<String> groups = new ArrayList<>();
                ResultSet result = executeSqlSelect("SELECT * FROM groups");

                System.out.println("--> getGroups");
                try {
                        while(result.next()) {

                                groups.add(result.getString("name"));
                                System.out.println("     -> Group: " + result.getString
                                    ↪ ("name"));
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        return groups;
        }


        @Override
        public List<Team> getTeams() throws RemoteException {
                List<Team> teams = new ArrayList<>();


                ResultSet result = executeSqlSelect("SELECT * FROM teams");

                System.out.println("--> getTeams");

```

```

		try {
			while(result.next()) {

				// get group

				String groupId = result.getString("groupId");
				ResultSet groupResult = executeSqlSelect("SELECT * FROM
				    ↪  groups WHERE Idgroups = " + groupId);
				String groupName = "";
				while(groupResult.next()) {
					groupName = groupResult.getString("name");
				}

				Team team = new Team(
						result.getInt("idteams"),
						result.getString("name"),
						groupName
				);
				teams.add(team);
				System.out.println("      -> Team: " + team.toString());
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}
	return teams;
	}



	@Override
	public void updateTeam(Team team) throws RemoteException {
		System.out.println("--> updateTeam");
		System.out.println("      -> Team: " + team.toString());

		Integer groupId = getGroupIdByName(team.getGroup());
		executeSqlUpdate(
				"UPDATE teams SET"
						+ " name = '" + team.getName() + "',"
						+ " groupId = " + groupId
					+ " WHERE idteams = " + team.getId()
					+ ";");

		notifiyOnTeamUpdated(team);
	}

	@Override
	public Integer addTeam(Team team) throws RemoteException {
		System.out.println("--> addTeam");
		System.out.println("      -> Team: " + team.toString());

		Integer groupId;

		if(team.getGroup().equals("")) {
			groupId = 1; // default group
		} else {
			groupId = getGroupIdByName(team.getGroup());
		}

		Integer newId = executeSqlUpdate(
				"INSERT INTO teams VALUES("
						+ " null, "
						+ " '" + team.getName() + "',"
						+ " " +  groupId
					+ ");");

		team.setId(newId);
		notifiyOnTeamAdded(team);
		return team.getId();
	}


	private Integer getGroupIdByName(String name) {
		ResultSet groupResult = executeSqlSelect("SELECT * FROM groups WHERE name = '" + name + "'");
		Integer id = -1;
		try {
			while(groupResult.next()) {
				id = groupResult.getInt("idgroups");
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}
		return id;
	}



	@Override
	public List<Game> getGames() throws RemoteException {
		List<Game> games = new ArrayList<>();
		ResultSet result = executeSqlSelect("SELECT * FROM games");
		System.out.println("--> getGames");
		try {
			while(result.next()) {

				// get teams
				Team teamA = getTeamById(result.getInt("teamAId"));
				Team teamB = getTeamById(result.getInt("teamBId"));

				Game game = new Game(
						result.getInt("idgames"),
						teamA,
						teamB,
						result.getBoolean("finished"),
						result.getInt("goalsTeamA"),
                                                result.getInt("goalsTeamB")
                                );
                                games.add(game);
                                System.out.println("     -> Game: " + game.toString());
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        return games;
        }


        @Override
        public void updateGame(Game game) throws RemoteException {
                System.out.println("--> updateGame");
                System.out.println("     -> Game: " + game.toString());


                executeSqlUpdate(
                                "UPDATE games SET"
                                                + " teamAId = " + game.getTeamA().getId
                                                    ↪ () + ","
                                                + " teamBId = " + game.getTeamB().getId
                                                    ↪ () + ","
                                                + " goalsTeamA = " + game.getGoalsTeamA
                                                    ↪ () + ","
                                                + " goalsTeamB = " + game.getGoalsTeamB
                                                    ↪ () + ","
                                                + " finished = " + game.getGameFinished
                                                    ↪ ()
                                        + " WHERE idgames = " + game.getId()
                                        + ";");

                notifiyOnGameUpdated(game);
        }



        @Override
        public Integer addGame(Game game) throws RemoteException {
                System.out.println("--> addGame");
                System.out.println("     -> game: " + game.toString());
                Integer newId = executeSqlUpdate(
                                "INSERT INTO games VALUES("
                                                + " null, "
                                                + " " + game.getTeamA().getId() + ","
                                                + " " + game.getTeamB().getId() + ","
                                                + " " + game.getGoalsTeamA() + ","
                                                + " " + game.getGoalsTeamB() + ","
                                                + " " + game.getGameFinished()
                                        + ");");

                game.setId(newId);
```

```
			notifiyOnGameAdded(game);
			return game.getId();
	}

	@Override
	public void removeAllBets() throws RemoteException {
			executeSqlUpdate("DELETE FROM bets WHERE idbets >= 0");
			System.out.println("--> removeAllBets");

	}

	@Override
	public List<Bet> getBets() throws RemoteException {
			List<Bet> bets = new ArrayList<>();
			ResultSet result = executeSqlSelect("SELECT * FROM bets");
			System.out.println("--> getBets");
			try {
				while(result.next()) {

					// get relationships
					Team teamA = getTeamById(result.getInt("teamAId"));
					Team teamB = getTeamById(result.getInt("teamBId"));
					User user = getUserById(result.getInt("userId"));

					Bet bet = new Bet(
							result.getInt("idbets"),
							user,
							teamA,
							teamB,
							result.getInt("goalsTeamA"),
							result.getInt("goalsTeamB")
					);
					bets.add(bet);
					System.out.println("      -> Bet: " + bet.toString());
				}
			} catch (SQLException e) {
				e.printStackTrace();
			}
	return bets;
	}

	@Override
	public Integer addBet(Bet bet) throws RemoteException {
			System.out.println("--> addBet");
			System.out.println("      -> bet: " + bet.toString());
			Integer newId = executeSqlUpdate(
					"INSERT INTO bets VALUES("
							+ " null, "
							+ " " + bet.getUser().getId() + ","
							+ " " + bet.getTeamA().getId() + ","
							+ " " + bet.getTeamB().getId() + ","
							+ " " + bet.getGoalsTeamA() + ","
							+ " " + bet.getGoalsTeamB()
```

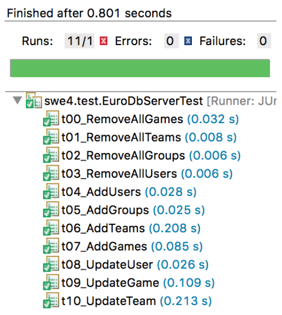
```java
                                         + ");");

                bet.setId(newId);
                notifiyOnBetAdded(bet);
                return bet.getId();
        }


        @Override
        public void updateBet(Bet bet) throws RemoteException {
                System.out.println("--> updateBet");
                System.out.println("      -> Bet: " + bet.toString());


                executeSqlUpdate(
                                "UPDATE bets SET"
                                                 + " userId = " + bet.getUser().getId() + ","
                                                 + " teamAId = " + bet.getTeamA().getId() + ","
                                                 + " teamBId = " + bet.getTeamB().getId() + ","
                                                 + " goalsTeamA = " + bet.getGoalsTeamA() + ","
                                                 + " goalsTeamB = " + bet.getGoalsTeamB()
                                         + " WHERE idbets = " + bet.getId()
                                         + ";");

                notifiyOnBetUpdated(bet);
        }


        @Override
        public void removeAllUsers() throws RemoteException {
                executeSqlUpdate("DELETE FROM users WHERE idusers >= 0");
                System.out.println("--> removeAllUsers");
        }



        @Override
        public void removeAllGroups() throws RemoteException {
                executeSqlUpdate("DELETE FROM groups WHERE idgroups >= 0");
                System.out.println("--> removeAllGroups");
        }



        @Override
        public void removeAllTeams() throws RemoteException {
                executeSqlUpdate("DELETE FROM teams WHERE idteams >= 0");
                System.out.println("--> removeAllTeams");
        }



        @Override
        public void removeAllGames() throws RemoteException {
                executeSqlUpdate("DELETE FROM games WHERE idgames >= 0");
                System.out.println("--> removeAllGames");
        }



        private String getGroupNameById(Integer id) {
                ResultSet groupResult = executeSqlSelect("SELECT * FROM groups WHERE idgroups = " + id );
                try {
                        while(groupResult.next()) {
                                return  groupResult.getString("name");
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return "";
        }

        private Team getTeamById(Integer id) {
                ResultSet result = executeSqlSelect("SELECT * FROM teams WHERE idteams = " + id);
                try {
                        while(result.next()) {
                                return new Team(
                                                result.getInt("idteams"),
                                                result.getString("name"),
                                                getGroupNameById(result.getInt("groupId"))
                                                );
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return null;
        }

        private User getUserById(Integer id) {
                ResultSet result = executeSqlSelect("SELECT * FROM users WHERE idusers = " + id);
                try {
                        while(result.next()) {
                                return new User(
                                                result.getInt("idusers"),
                                                result.getBoolean("active"),
                                                result.getString("name"),
                                                result.getString("password")
                                                );
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return null;
        }





        @Override
        public void registerForNotification(EuroBetDbFactory n) throws RemoteException
            ↪ {
                System.out.println("registerForNotification: " +n);
                clientList.add(n);
        }


        private void notifiyOnUserUpdated(User user) throws RemoteException {

                for(EuroBetDbFactory client : clientList) {
                        client.onUserUpdated(user);
                }
        }

        private void notifiyOnUserAdded(User user) throws RemoteException {
                for(EuroBetDbFactory client : clientList) {
                        client.onUserAdded(user);
                }
        }


        private void notifiyOnTeamUpdated(Team team) throws RemoteException {

                for(EuroBetDbFactory client : clientList) {
                        client.onTeamUpdated(team);
                }
        }

        private void notifiyOnTeamAdded(Team team) throws RemoteException {
                for(EuroBetDbFactory client : clientList) {
                        client.onTeamAdded(team);
                }
        }


        private void notifiyOnGameUpdated(Game game) throws RemoteException {

                for(EuroBetDbFactory client : clientList) {
                        client.onGameUpdated(game);
```

```
			}
		}

		private void notifiyOnGameAdded(Game game) throws RemoteException {
			for(EuroBetDbFactory client : clientList) {
				client.onGameAdded(game);
			}
		}

		private void notifiyOnBetUpdated(Bet bet) throws RemoteException {

			for(EuroBetDbFactory client : clientList) {
				client.onBetUpdated(bet);
			}
		}

		private void notifiyOnBetAdded(Bet bet) throws RemoteException {
			for(EuroBetDbFactory client : clientList) {
				client.onBetAdded(bet);
			}
		}



		private Connection getConnection() throws SQLException {
			return DriverManager.getConnection("jdbc:mysql://" + DB_URL + ":" +
			    ↪ DB_PORT + "/" + DB_NAME, DB_USER, DB_PASSWORD);
		}

		private ResultSet executeSqlSelect(String sql) {

			try {
				Connection connection = getConnection();
				Statement statement = connection.createStatement();
				return statement.executeQuery(sql);

			} catch (Exception e) {
				e.printStackTrace();
				return null;
			}
		}

		private Integer executeSqlUpdate(String sql) {

			try {
				Connection connection = getConnection();
				Statement statement = connection.createStatement();
				statement.executeUpdate(sql, Statement.RETURN_GENERATED_KEYS);
				ResultSet rs = statement.getGeneratedKeys();
				if (rs.next()) {
					int newId = rs.getInt(1);
					return newId;
				}
```

```
                        return 0;

                } catch (Exception e) {
                        e.printStackTrace();
                        return -1;
                }
        }







        public static void main(String[] args) {

                try {
                        String hostPort = Util.getHostPortArg(args);
                        EuroBetDb b = (EuroBetDb) new EuroBetDbServer();
                        EuroBetDb buffSkel = (EuroBetDb)UnicastRemoteObject.exportObject(b, 0);
                        LocateRegistry.createRegistry(Util.getPort(hostPort));

                        Naming.rebind("rmi://" + hostPort + "/EuroBetDb", buffSkel);

                        System.out.println("EuroBetDb Server running, waiting for connections ...");

                } catch (RemoteException e) {
                        e.printStackTrace();
                } catch (MalformedURLException e) {
                        e.printStackTrace();
                }
        }
}
```

1.3 Testfälle

EuroDbServerTest.java



```
Finished after 0.801 seconds
Runs: 11/11   Errors: 0   Failures: 0

swe4.test.EuroDbServerTest [Runner: JU
    t00_RemoveAllGames (0.032 s)
    t01_RemoveAllTeams (0.008 s)
    t02_RemoveAllGroups (0.006 s)
    t03_RemoveAllUsers (0.006 s)
    t04_AddUsers (0.028 s)
    t05_AddGroups (0.025 s)
    t06_AddTeams (0.208 s)
    t07_AddGames (0.085 s)
    t08_UpdateUser (0.026 s)
    t09_UpdateGame (0.109 s)
    t10_UpdateTeam (0.213 s)
```

```java
package swe4.test;

import static org.junit.Assert.*;

import java.util.List;
import org.junit.runners.MethodSorters;
import org.junit.FixMethodOrder;
import org.junit.Test;

import swe4.models.Game;
import swe4.models.Team;
import swe4.models.User;
import swe4.rmi.client.EuroBetDbFactory;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class EuroDbServerTest {


        EuroBetDbFactory server;
        private Integer zero = 0;


        @Test
        public void t00_RemoveAllGames() {
                Integer current;
                EuroBetDbFactory.getInstance().removeAllGames();
                current = EuroBetDbFactory.getInstance().getGames().size();
                assertEquals(zero, current);
        }

        @Test
        public void t01_RemoveAllTeams() {
                Integer current;
                EuroBetDbFactory.getInstance().removeAllTeams();
                current = EuroBetDbFactory.getInstance().getTeams().size();
                assertEquals(zero, current);

        }

        @Test
        public void t02_RemoveAllGroups() {
                Integer current;
                EuroBetDbFactory.getInstance().removeAllGroups();
                current = EuroBetDbFactory.getInstance().getGroups().size();
                assertEquals(zero, current);
        }

        @Test
        public void t03_RemoveAllUsers() {
                Integer current;
                EuroBetDbFactory.getInstance().removeAllUsers();
                current = EuroBetDbFactory.getInstance().getUsers().size();
                assertEquals(zero, current);
        }


        @Test
        public void t04_AddUsers() {
                User user = new User();

                user.setName("Robert Wurm");
                user.setActive(false);
                user.setPassword("1234");
                EuroBetDbFactory.getInstance().addUser(user);

                user.setName("Hansi Hinterseer");
                user.setActive(true);
                user.setPassword("123456");
                EuroBetDbFactory.getInstance().addUser(user);


                user.setName("Frau Holle");
                user.setActive(true);
                user.setPassword("xxx");
                EuroBetDbFactory.getInstance().addUser(user);



                user.setName("Asterix");
                user.setActive(true);
                user.setPassword("xx56x");
                EuroBetDbFactory.getInstance().addUser(user);



                List<User> allUsers = EuroBetDbFactory.getInstance().getUsers();
                assertNotNull(allUsers);
                assertEquals(4, allUsers.size());

                Boolean robertExists = false;
                Boolean asterixExists = false;
                for(User u : allUsers) {
                        if(u.getName().equals("Robert Wurm")) {
                                assertEquals(u.getPassword(), "1234");
                                assertEquals(u.getActive(), false);
                                robertExists = true;
                        }

                        if(u.getName().equals("Asterix")) {
                                assertEquals(u.getPassword(), "xx56x");
                                assertEquals(u.getActive(), true);
                                asterixExists = true;
                        }

                        if(u.getName().equals("Dummy !!!")) {
```

```
                        fail("username shouldnt exists!");
                    }
                }
                assertEquals(true, robertExists);
                assertEquals(true, asterixExists);
        }


        @Test
        public void t05_AddGroups() {

                EuroBetDbFactory.getInstance().addGroup('A');
                EuroBetDbFactory.getInstance().addGroup('B');
                EuroBetDbFactory.getInstance().addGroup('C');
                EuroBetDbFactory.getInstance().addGroup('D');
                EuroBetDbFactory.getInstance().addGroup('E');
                EuroBetDbFactory.getInstance().addGroup('F');

                List<String> allGroups = EuroBetDbFactory.getInstance().getGroups();

                assertNotNull(allGroups);
                assertEquals(6, allGroups.size());

                Boolean existsA = false;
                Boolean existsD = false;
                for(String g : allGroups) {
                        if(g.equals("A")) {
                                existsA = true;
                        }

                        if(g.equals("D")) {
                                existsD = true;
                        }
                        if(g.equals("Z")) {
                                fail("group shouldnt exists!");
                        }
                }
                assertEquals(true, existsA);
                assertEquals(true, existsD);
        }


        @Test
        public void t06_AddTeams() {
                Team team = new Team();
                Integer id;
                Integer unableToCreate = -1;

                team.setName("France");
                team.setGroup("Z");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertEquals(id, unableToCreate);
                team.setGroup("A");
```

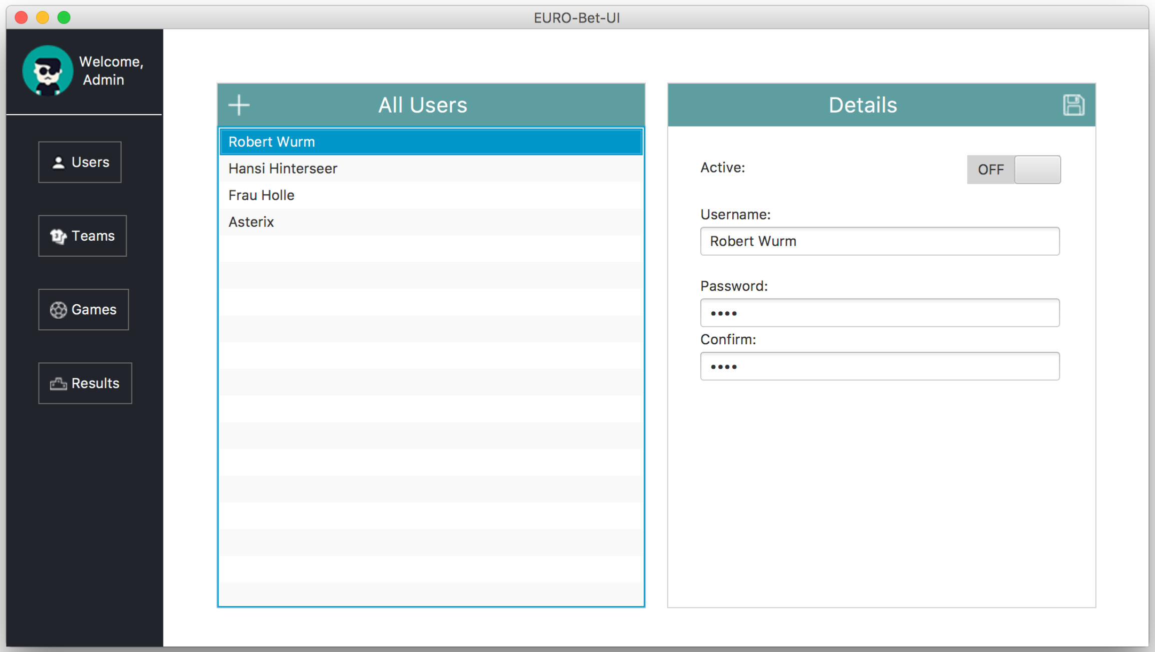
```
            id = EuroBetDbFactory.getInstance().addTeam(team);
            assertNotEquals(id, unableToCreate);


            team.setName("Switzerland");
            team.setGroup("A");
            id = EuroBetDbFactory.getInstance().addTeam(team);
            assertNotEquals(id, unableToCreate);


            team.setName("Romania");
            team.setGroup("A");
            id = EuroBetDbFactory.getInstance().addTeam(team);
            assertNotEquals(id, unableToCreate);


            team.setName("Albania");
            team.setGroup("A");
            id = EuroBetDbFactory.getInstance().addTeam(team);
            assertNotEquals(id, unableToCreate);


            team.setName("Wales");
            team.setGroup("B");
            id = EuroBetDbFactory.getInstance().addTeam(team);
            assertNotEquals(id, unableToCreate);

            team.setName("Slovakia");
            team.setGroup("B");
            id = EuroBetDbFactory.getInstance().addTeam(team);
            assertNotEquals(id, unableToCreate);

            team.setName("England");
            team.setGroup("B");
            id = EuroBetDbFactory.getInstance().addTeam(team);
            assertNotEquals(id, unableToCreate);

            team.setName("Russia");
            team.setGroup("B");
            id = EuroBetDbFactory.getInstance().addTeam(team);
            assertNotEquals(id, unableToCreate);

            team.setName("Germany");
            team.setGroup("C");
            id = EuroBetDbFactory.getInstance().addTeam(team);
            assertNotEquals(id, unableToCreate);

            team.setName("Poland");
            team.setGroup("C");
            id = EuroBetDbFactory.getInstance().addTeam(team);
            assertNotEquals(id, unableToCreate);

            team.setName("Northern Ireland");
```

```
                team.setGroup("C");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);

                team.setName("Ukraine");
                team.setGroup("C");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);

                team.setName("Croatia");
                team.setGroup("D");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);

                team.setName("Spain");
                team.setGroup("D");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);

                team.setName("Czech Republic");
                team.setGroup("D");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);

                team.setName("Turkey");
                team.setGroup("D");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);


                team.setName("Italy");
                team.setGroup("E");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);

                team.setName("Irland");
                team.setGroup("E");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);

                team.setName("Sweden");
                team.setGroup("E");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);

                team.setName("Belgium");
                team.setGroup("E");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);

                team.setName("Hungary");
                team.setGroup("F");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);

                team.setName("Iceland");
                team.setGroup("F");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);

                team.setName("Portugal");
                team.setGroup("F");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);

                team.setName("Austria");
                team.setGroup("F");
                id = EuroBetDbFactory.getInstance().addTeam(team);
                assertNotEquals(id, unableToCreate);

                List<Team> allTeams = EuroBetDbFactory.getInstance().getTeams();
                assertNotNull(allTeams);
                assertEquals(24, allTeams.size());

                Boolean austriaExists = false;
                Boolean walesExists = false;
                for(Team t : allTeams) {
                        if(t.getName().equals("Austria")) {
                                austriaExists = true;
                        }

                        if(t.getName().equals("Wales")) {
                                walesExists = true;
                        }

                        if(t.getName().equals("Lichtenstein")) {
                                fail("team shouldnt exists!");
                        }
                }
                assertEquals(true, austriaExists);
                assertEquals(true, walesExists);
        }


        @Test
        public void t07_AddGames() {
                Game game = new Game();
                Team austria = null;
                Team iceland = null;
                Team portugal = null;
                Team hungary = null;

                List<Team> allTeams = EuroBetDbFactory.getInstance().getTeams();

                for(Team t : allTeams) {
                        if(t.getName().equals("Austria")) { austria = t; }
                        if(t.getName().equals("Iceland")) { iceland = t; }
                        if(t.getName().equals("Hungary")) { hungary = t; }
                        if(t.getName().equals("Portugal")) { portugal = t; }
                }

                assertNotNull(austria);
                assertNotNull(iceland);
                assertNotNull(portugal);
                assertNotNull(hungary);



                game.setTeamA(austria);
                game.setTeamB(hungary);
                game.setGameFinished(true);
                game.setGoalsTeamA(0);
                game.setGoalsTeamB(2);
                EuroBetDbFactory.getInstance().addGame(game);


                game.setTeamA(portugal);
                game.setTeamB(iceland);
                game.setGameFinished(true);
                game.setGoalsTeamA(1);
                game.setGoalsTeamB(1);
                EuroBetDbFactory.getInstance().addGame(game);


                game.setTeamA(portugal);
                game.setTeamB(austria);
                game.setGameFinished(false);
                EuroBetDbFactory.getInstance().addGame(game);


                game.setTeamA(hungary);
                game.setTeamB(portugal);
                EuroBetDbFactory.getInstance().addGame(game);

                game.setTeamA(iceland);
                game.setTeamB(hungary);
                EuroBetDbFactory.getInstance().addGame(game);

                game.setTeamA(austria);
                game.setTeamB(iceland);
                EuroBetDbFactory.getInstance().addGame(game);
        }


        @Test
        public void t08_UpdateUser() {
                List<User> allUsers = EuroBetDbFactory.getInstance().getUsers();
                User robert = null;
                User hansi = null;
                User hansy = null;
                for(User u : allUsers) {
                        if(u.getName().equals("Robert Wurm")) { robert = u; }
                        if(u.getName().equals("Hansi Hinterseer")) { hansi = u; }
                        if(u.getName().equals("Hansy Hinterseer")) { hansy = u; }
                }
                assertNotNull(robert);
                assertNotNull(hansi);
                assertNull(hansy);


                robert.setName("Updated Robert");
                EuroBetDbFactory.getInstance().updateUser(robert);


                allUsers = EuroBetDbFactory.getInstance().getUsers();
                robert = null;
                hansi = null;
                for(User u : allUsers) {
                        if(u.getName().equals("Updated Robert")) { robert = u; }
                        if(u.getName().equals("Hansi Hinterseer")) { hansi = u; }
                        if(u.getName().equals("Hansy Hinterseer")) { hansy = u; }
                }
                assertNull(hansy);
                assertNotNull(robert);
                assertNotNull(hansi);

                assertEquals(robert.getName(), "Updated Robert");
                assertEquals(hansi.getName(), "Hansi Hinterseer");


                robert.setName("Robert Wurm");
                EuroBetDbFactory.getInstance().updateUser(robert);
        }


        @Test
        public void t09_UpdateGame() {
                Game game = null;
                List<Game> allGames;

                allGames = EuroBetDbFactory.getInstance().getGames();

                for(Game g : allGames) {
                        if(g.getTeamA().getName().equals("Austria") &&
                                g.getTeamB().getName().equals("Iceland")) {
                                game = g;
                        }
                }

                assertNotNull(game);

                Integer newGoalsTeamA = 10;
```

```
		Integer newGoalsTeamB = 0;
		game.setGameFinished(true);
		game.setGoalsTeamA(newGoalsTeamA);
		game.setGoalsTeamB(newGoalsTeamB);

		EuroBetDbFactory.getInstance().updateGame(game);

		game = null;
		allGames = EuroBetDbFactory.getInstance().getGames();

		for(Game g : allGames) {
			if(g.getTeamA().getName().equals("Austria") &&
				g.getTeamB().getName().equals("Iceland")) {
				game = g;
			}
		}

		assertNotNull(game);

		assertEquals(game.getGoalsTeamA(), newGoalsTeamA);
		assertNotEquals(game.getGoalsTeamB(), newGoalsTeamA);

		game.setGoalsTeamA(0);
		game.setGoalsTeamB(2);

		EuroBetDbFactory.getInstance().updateGame(game);
	}

	@Test
	public void t10_UpdateTeam() {
		Team austria = null;
		List<Team> allTeams;


		allTeams = EuroBetDbFactory.getInstance().getTeams();
		for(Team t:allTeams) {
			if(t.getName().equals("Austria")) { austria = t; };
		}
		assertNotNull(austria);

		austria.setName("Australia");
		EuroBetDbFactory.getInstance().updateTeam(austria);

		austria = null;
		allTeams = EuroBetDbFactory.getInstance().getTeams();
		for(Team t:allTeams) {
			if(t.getName().equals("Australia")) { austria = t; };
		}
		assertNotNull(austria);

		assertEquals(austria.getName(), "Australia");


```

```

            austria.setName("Austria");
            EuroBetDbFactory.getInstance().updateTeam(austria);

            austria = null;
            allTeams = EuroBetDbFactory.getInstance().getTeams();
            for(Team t:allTeams) {
                    if(t.getName().equals("Austria")) { austria = t; };
            }
            assertNotNull(austria);

            assertEquals(austria.getName(), "Austria");
        }
}
```

Users



Robert Wurm
Hansi Hinterseer
Frau Holle
Asterix

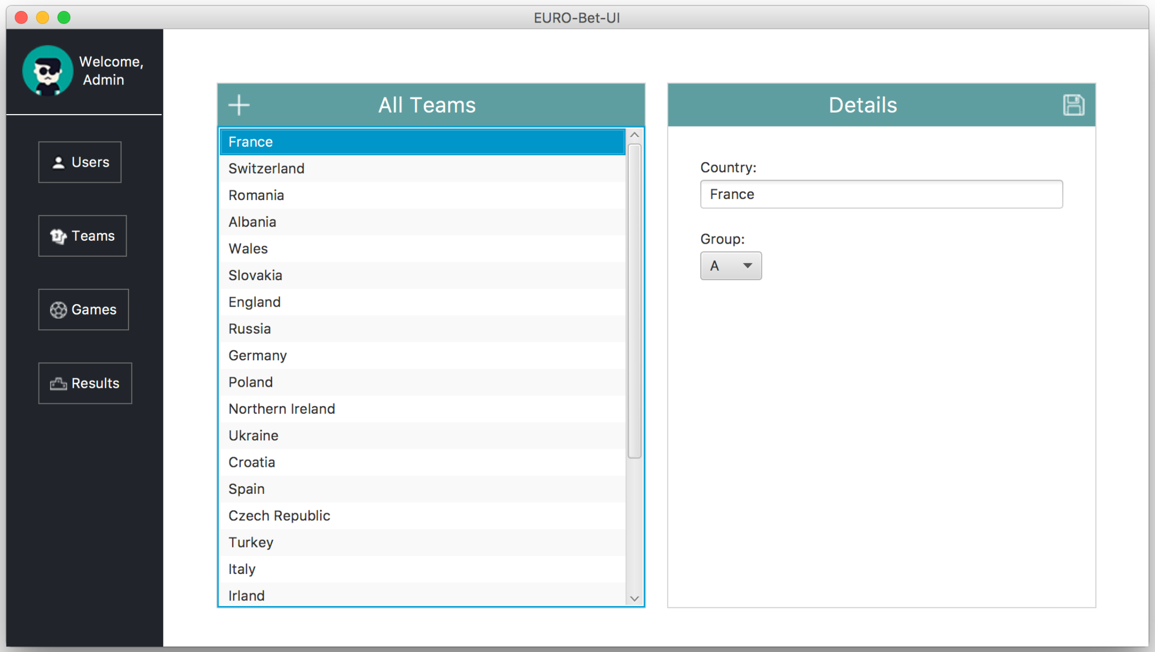
Active: OFF

Username: Robert Wurm

Password: ••••

Confirm: ••••

Teams

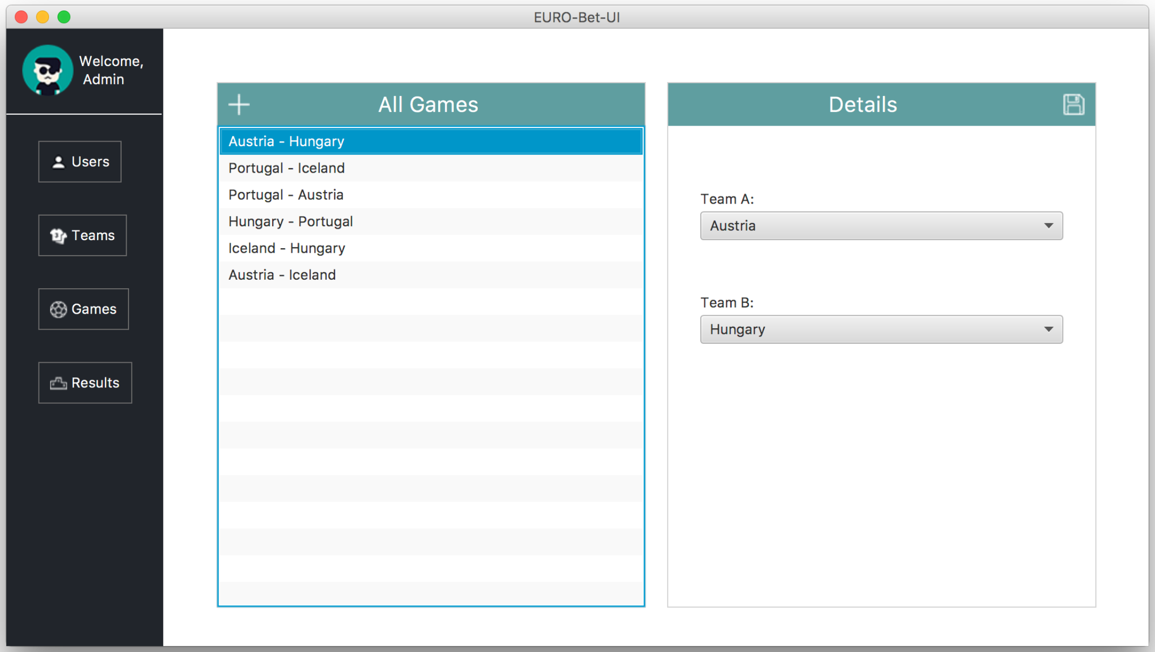


France
Switzerland
Romania
Albania
Wales
Slovakia
England
Russia
Germany
Poland
Northern Ireland
Ukraine
Croatia
Spain
Czech Republic
Turkey
Italy
Irland

Country: France

Group: A

Games

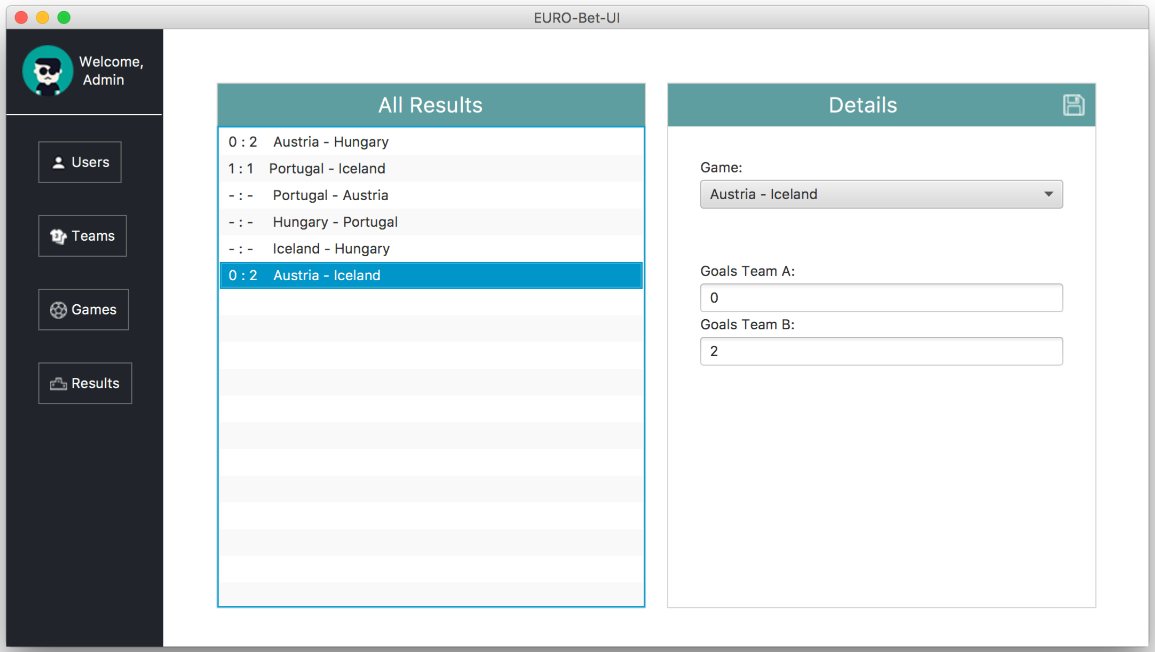


Austria - Hungary
Portugal - Iceland
Portugal - Austria
Hungary - Portugal
Iceland - Hungary
Austria - Iceland

Team A: Austria

Team B: Hungary

Results



All Results

0 : 2	Austria - Hungary
1 : 1	Portugal - Iceland
- : -	Portugal - Austria
- : -	Hungary - Portugal
- : -	Iceland - Hungary
0 : 2	Austria - Iceland

Details

Game: Austria - Iceland

Goals Team A: 0

Goals Team B: 2