

UNIWERSYTET RZESZOWSKI
WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH
INSTYTUT INFORMATYKI



Yevhenii Usik
134980

Informatyka

Dokumentacja projektu Java na temat: "System do organizowania turniejów w DotA 2"

Praca projektowa

Praca wykonana pod kierunkiem
mgr inż. Ewa Żeslawska

Rzeszów 2025

Spis treści

1. Streszczenie.....	6
2. Opis założień Projektu.....	7
2.1. Cel, problem, kroki realizacji projektu.....	7
2.2. Wymagania:	7
2.2.1. Funkcjonalne	7
2.2.2. Niefunkcjonalne	7
3. Opis struktury Projektu	8
3.1. Cała Struktura Projektu	8
3.1.1. Main	9
3.1.2. Util and Service.....	10
3.1.3. Controllers.....	11
3.1.4. Models.....	21
3.1.5. Resources	22
3.2. Wykorzystane Technologie.....	22
3.2.1. Wymagania sprzętowe	22
4. Harmonogram Realizacji Projektu	23
4.0.1. Repozytorium.....	23
5. Prezentacja warstwy użytkowej projektu.....	24
5.1. Welcome.fxml.....	24
5.2. Login.fxml	24
5.3. Register.fxml.....	25
5.4. TournamentSelection.fxml	26
5.5. TournamentInfo.fxml.....	29
6. Podsumowanie.....	30
Spis rysunków	31
Spis listingów	32

1. Streszczenie

Ten projekt to oparta na Javie aplikacja DOTA 2 Tournament Manager zbudowana z JavaFX dla interfejsu użytkownika i Hibernate do interakcji z bazą danych. Pozwala użytkownikom widzieć krótkę najważniejszą informacje o turniejach, a także jeżeli użytkownik posiada rolę administratora, ma dodatkowe funkcje. Aplikacja zapewnia funkcje dodawania, edytowania i usuwania turniejów, a także zarządzania powiązanymi podmiotami. Schemat bazy danych został zaprojektowany do obsługi złożonych relacji między obiektyami, dzięki czemu nadaje się do organizowania i śledzenia wydarzeń e-sportowych DOTA 2. Projekt wykorzystuje Maven do zarządzania zależnościami i ma modułową, łatwą w utrzymaniu strukturę.

2. Opis założen Projektu

2.1. Cel, problem, kroki realizacji projektu

Celem projektu jest stworzenie aplikacji do zarządzania turniejami DOTA 2, która umożliwi organizatorom kompleksowe zarządzanie danymi. Projekt rozwiązuje problem braku wygodnego, zintegrowanego narzędzia desktopowego umożliwiającego sprawne zarządzanie danymi dotyczącymi organizacji i przebiegu turniejów e-sportowych, zwłaszcza w przypadku gry DOTA 2. Źródłem problemu jest brak aplikacji desktopowej ze względu na trudności zarządzania wieloma powiązanymi ze sobą podmiotami, co często prowadzi do rozproszenia danych oraz trudności w ich aktualizacji i analizie.

Aby rozwiązać ten problem, konieczne było zaprojektowanie i wdrożenie aplikacji, która zapewnia centralizację danych, intuicyjny interfejs użytkownika oraz możliwość łatwego dodawania, edytowania i usuwania informacji o wszystkich kluczowych elementach turnieju. Musiałem upewnić się, że baza danych ma odpowiednią strukturę, że jest zintegrowana z graficznym interfejsem użytkownika oraz że system jest bezpieczny i wydajny.

Analiza wymagań, projektowanie bazy danych i interfejsu, implementacja funkcjonalności, testowanie i wdrożenie to kroki, które zostały podjęte w ramach projektu.

2.2. Wymagania:

2.2.1. Funkcjonalne

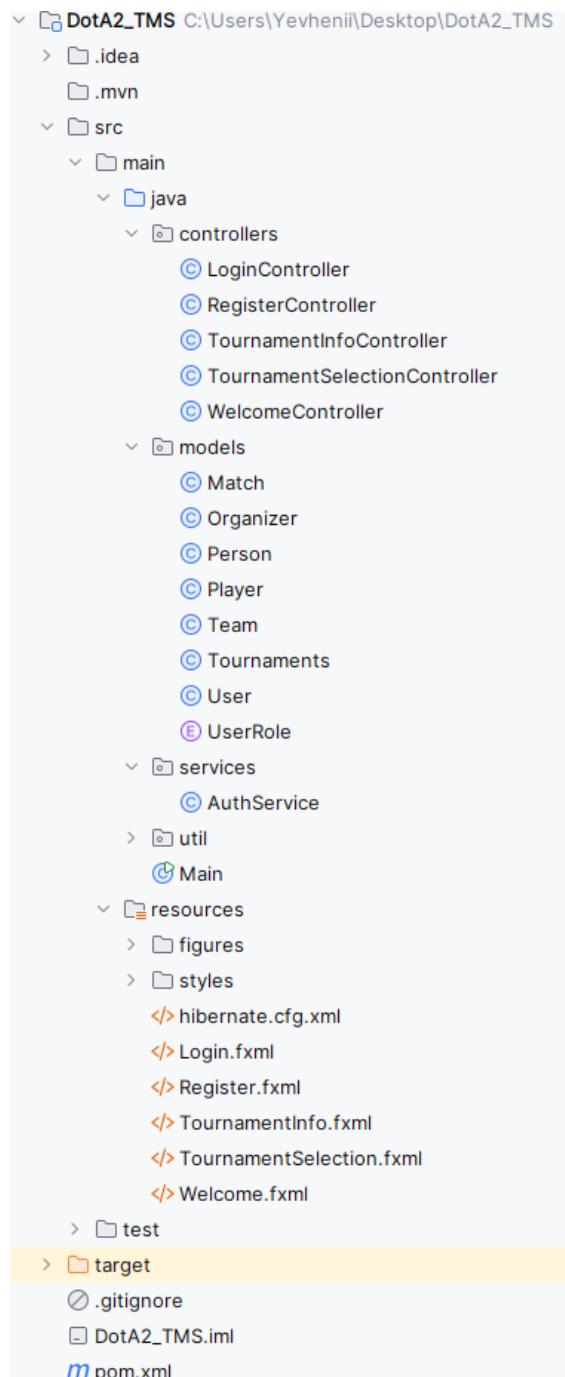
- Możliwość dodawania, edytowania i usuwania turniejów.
- Przeglądanie listy turniejów oraz szczegółowych informacji o każdym z nich.
- Autoryzacja użytkowników i zarządzanie poziomami dostępu.
- Wybór typu turnieju (jeżeli offline - trzeba wprowadzić lokalizację) i organizatora turnieju z listy dostępnych (tylko dla administratora).

2.2.2. Niefunkcjonalne

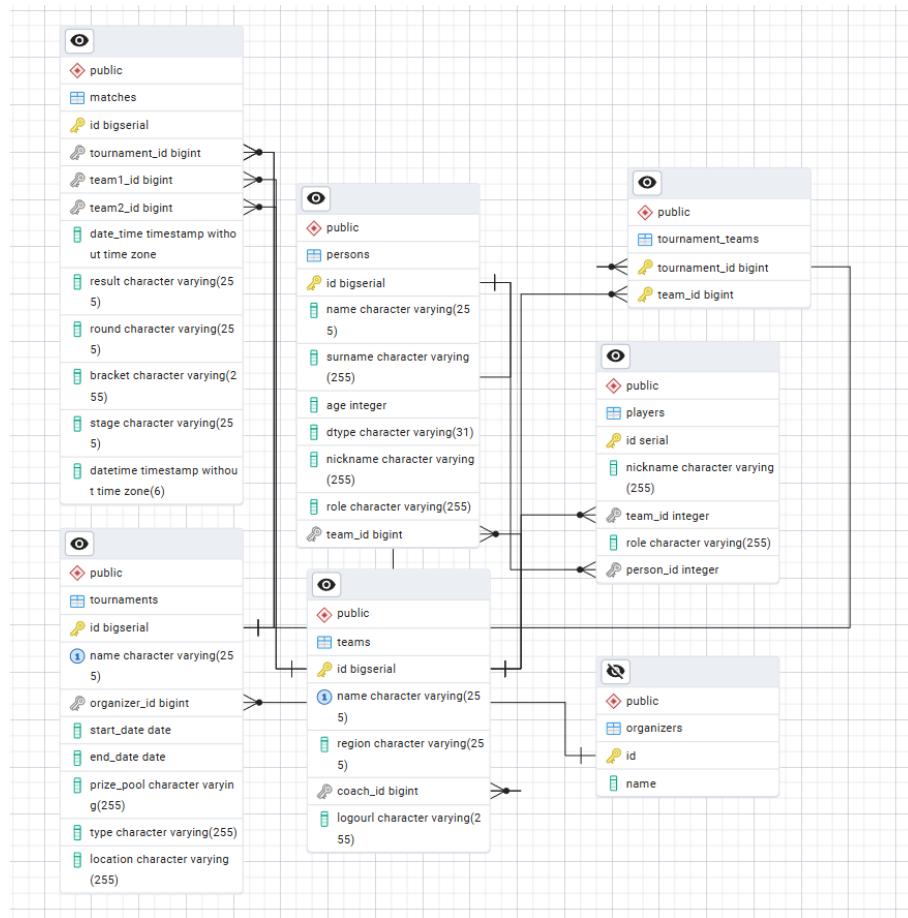
- Intuicyjny, z odpowiednią estetyką (temat DOTA2) interfejs.
- Bezpieczeństwo przechowywanych danych (uwierzytelnianie, autoryzacja, hashowanie haseł).
- Łatwość utrzymania i rozbudowy systemu.
- Łatwość wprowadzania zmian w bazie danych i aplikacji.

3. Opis struktury Projektu

3.1. Cała Struktura Projektu



Rys. 3.1. Struktura całego projektu



Rys. 3.2. Baza danych

3.1.1. Main

Main po prostu uruchamia aplikacje i ustawia ikonę i title dla programu, zaczyna się ona od Welcome.fxml (interfejs zademonstruje w następującym rozdziale).

Listing 3.1. Main: jak zaczyna się program

```

1 public class Main extends Application {
2     @Override
3     public void start(Stage stage) throws Exception{
4         Parent root = FXMLLoader.load(getClass().getResource("Welcome.fxml"));
5         Scene scene = new Scene(root);
6         stage.setScene(scene);
7         stage.setTitle("DOTA2_TMS");
8         Image icon = new Image("figures/dotaIcon.png");
9         stage.getIcons().add(icon);
10        stage.setResizable(false);
11        stage.show();
12
13    }
14    public static void main(String[] args) {
15        launch(args);
16    }
17 }
```

3.1.2. Util and Service

Listing 3.2. Klasa HibernateUtil: odpowiedzialna za konfigurację bazy danych

```

1 public class HibernateUtil {
2     private static final SessionFactory sessionFactory = buildSessionFactory();
3
4     private static SessionFactory buildSessionFactory() {
5         try {
6             return new org.hibernate.cfg.Configuration()
7                 .configure()
8                 .buildSessionFactory();
9         } catch (Throwable ex) {
10             System.err.println("Initial SessionFactory creation failed." + ex);
11             throw new ExceptionInInitializerError(ex);
12         }
13     }
14     public static SessionFactory getSessionFactory() {
15         return sessionFactory;
16     }
17     public static void shutdown() {
18         getSessionFactory().close();
19     }
20 }
```

Listing 3.3. Klasa AuthService: autoryzacja z wyszukiwaniem w bazie konta i dehashowaniem hasła BCryptem

```

1 public static boolean login(String username, String password) {
2     try (Session session = HibernateUtil.getSessionFactory().openSession()) {
3         Query<User> query = session.createQuery("FROM User WHERE username = :username", User.class);
4         query.setParameter("username", username);
5         User user = query.uniqueResult();
6
7         if (user != null && org.mindrot.jbcrypt.BCrypt.checkpw(password, user.getPassword())) {
8             session.beginTransaction();
9             session.merge(user);
10            session.getTransaction().commit();
11            currentUser = user;
12            return true;
13        }
14    } catch (Exception e) {
15        e.printStackTrace();
16    }
17    return false;
18 }
```

Listing 3.4. Klasa AuthService: rejestracja ze sprawdzeniem czy istnieje już konto i hashowanie hasła nowego użytkownika

```

1 public static boolean register(String username, String password, String role) {
2     try (Session session = HibernateUtil.getSessionFactory().openSession()) {
3         Query<User> query = session.createQuery("FROM User WHERE username = :username", User.class);
4         query.setParameter("username", username);
```

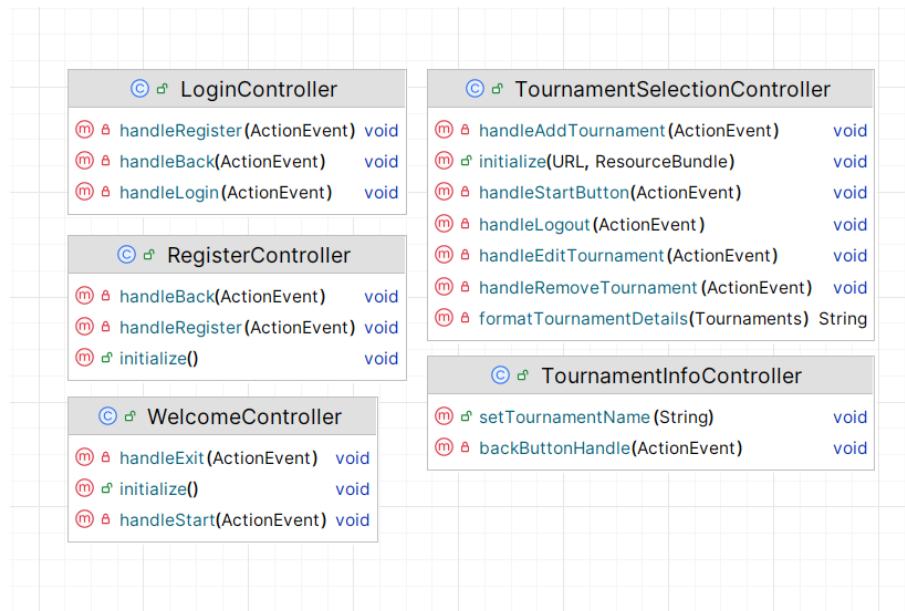
```

5         User existingUser = query.uniqueResult();
6         if (existingUser != null) {
7             return false;
8         }
9         String hashedPassword = org.mindrot.jbcrypt.BCrypt.hashpw(password, org.
mindrot.jbcrypt.BCrypt.gensalt());
10        User newUser = new User();
11        newUser.setUsername(username);
12        newUser.setPassword(hashedPassword);
13        newUser.setRole(UserRole.valueOf(role));
14        newUser.setCreatedAt(java.time.LocalDateTime.now());
15        session.beginTransaction();
16        session.persist(newUser);
17        session.getTransaction().commit();
18        return true;
19    } catch (Exception e) {
20        e.printStackTrace();
21    }
22    return false;
23 }
```

3.1.3. Controllers

Pakiet controllers zawiera klasy odpowiedzialne za backend fxml plików.

Rys. 3.3. Metody kontrolerów



Listing 3.5. Klasa LoginController: handleBack i handleRegister - przekazania na nowe strony; handleLogin - logowanie użytkownika z wykorzystaniem wyżej opisanym AuthService dla autentyfikacji usera i przy poprawnych danych - przekazanie na nową stronę

```

1 @FXML
2 private void handleLogin(ActionEvent event) {
3     try {
4         String username = usernameField.getText();
5         String password = passwordField.getText();
```

```

6         if (username.isEmpty() || password.isEmpty()) {
7             Alert alert = new Alert(Alert.AlertType.ERROR);
8             alert.setTitle("Invalid login data");
9             alert.setHeaderText("There are empty fields");
10            alert.setContentText("Please enter all login information.");
11            alert.showAndWait();
12            usernameField.clear();
13            passwordField.clear();
14        } else if (AuthService.login(username, password)) {
15            Alert alert = new Alert(Alert.AlertType.INFORMATION);
16            alert.setTitle("Login successful");
17            alert.setHeaderText("Welcome, " + username + "!");
18            alert.setContentText("Successfully logged in.");
19            alert.showAndWait();
20            FXMLLoader loader = new FXMLLoader(getClass().getResource("/
TournamentSelection.fxml"));
21            Parent root = loader.load();
22            Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
23            stage.setScene(new Scene(root));
24        } else {
25            Alert alert = new Alert(Alert.AlertType.ERROR);
26            alert.setTitle("Login error");
27            alert.setHeaderText("Invalid login data");
28            alert.setContentText("Please check your login information and try again.
"));
29            alert.showAndWait();
30            usernameField.clear();
31            passwordField.clear();
32        }
33    } catch (Exception e) {
34        Alert alert = new Alert(Alert.AlertType.ERROR, "Unexpected error: " + e.
getMessage());
35        alert.showAndWait();
36    }
37 }
38 @FXML
39 private void handleRegister(ActionEvent event) {
40     try {
41         FXMLLoader loader = new FXMLLoader(getClass().getResource("/Register.fxml"))
;
42         Parent root = loader.load();
43         Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
44         stage.setScene(new Scene(root));
45     } catch (Exception e) {
46         Alert alert = new Alert(Alert.AlertType.ERROR, "Unexpected error: " + e.
getMessage());
47         alert.showAndWait();
48     }
49 }
50 @FXML
51 private void handleBack(ActionEvent event) {
52     try {
53         FXMLLoader loader = new FXMLLoader(getClass().getResource("/Welcome.fxml"));
54         Parent root = loader.load();
55         Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
56         stage.setScene(new Scene(root));
57     } catch (Exception e) {

```

```

58         Alert alert = new Alert(Alert.AlertType.ERROR, "Unexpected error: " + e.
59             getMessage());
60         alert.showAndWait();
61     }
62 }
```

Listing 3.6. Klasa RegisterController: handleRegister - rejestracja nowego użytkownika z wykorzystaniem AuthService; umożliwia rejestrację jako Admin za pomocą sprawdzenia adminCodeField (admin code)

```

1  @FXML
2  public void initialize() {
3      roleComboBox.setItems(FXCollections.observableArrayList("USER", "ADMIN"));
4      roleComboBox.setValue("USER");
5      roleComboBox.setOnAction(e -> {
6          boolean isAdmin = "ADMIN".equals(roleComboBox.getValue());
7          adminCodeField.setVisible(isAdmin);
8          adminCodeField.setManaged(isAdmin);
9      });
10 }
11
12 @FXML
13 private void handleRegister(ActionEvent event) {
14     try {
15         String username = usernameField.getText();
16         String password = passwordField.getText();
17         String role = roleComboBox.getValue();
18         if (username.isEmpty() || password.isEmpty() || role == null) {
19             Alert alert = new Alert(Alert.AlertType.ERROR, "Please fill in all
fields.");
20             alert.showAndWait();
21             return;
22         }
23         if ("ADMIN".equals(role)) {
24             String adminCode = adminCodeField.getText();
25             if (!"1111".equals(adminCode)) {
26                 Alert alert = new Alert(Alert.AlertType.ERROR, "Invalid admin code."
);
27                 alert.showAndWait();
28                 return;
29             }
30         }
31         boolean success = AuthService.register(username, password, role);
32         if (success) {
33             Alert alert = new Alert(Alert.AlertType.INFORMATION, "Registration
successful! You can now log in.");
34             alert.showAndWait();
35             FXMLLoader loader = new FXMLLoader(getClass().getResource("/Login.fxml")
);
36             Parent root = loader.load();
37             Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
38             stage.setScene(new Scene(root));
39         } else {
40             Alert alert = new Alert(Alert.AlertType.ERROR, "Registration failed.
Username may already exist.");
41             alert.showAndWait();
42         }
}
```

```

43         } catch (Exception e) {
44             Alert alert = new Alert(Alert.AlertType.ERROR, "Unexpected error: " + e.
45             getMessage());
46             alert.showAndWait();
47         }
48     }
49
50     @FXML
51     private void handleBack(ActionEvent event) {
52         try {
53             FXMLLoader loader = new FXMLLoader(getClass().getResource("/Login.fxml"));
54             Parent root = loader.load();
55             Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
56             stage.setScene(new Scene(root));
57         } catch (Exception e) {
58             Alert alert = new Alert(Alert.AlertType.ERROR, "Unexpected error: " + e.
59             getMessage());
60             alert.showAndWait();
61         }
62     }

```

Listing 3.7. Klasa TournamentSelectionController: w tej klasie przy inicjalizacji jest kontrola roli użytkownika i zrealizowane CRUD na turniejach(używamy HibernateUtil opisany wyżej); W przyszłości ulepszę tę klasę

```

1  @Override
2  public void initialize(URL location, ResourceBundle resources) {
3      try (Session session = HibernateUtil.getSessionFactory().openSession()) {
4          Query<Tournaments> query = session.createQuery("FROM models.Tournaments",
5          Tournaments.class);
6          List<Tournaments> tournaments = query.list();
7          for (Tournaments t : tournaments) {
8              String details = formatTournamentDetails(t);
9              listTournaments.getItems().add(details);
10         }
11     } catch (Exception e) {
12         Alert alert = new Alert(Alert.AlertType.ERROR, "Could not load tournaments:
13         " + e.getMessage());
14         alert.showAndWait();
15     }
16
17     User user = AuthService.getCurrentUser();
18     if (user != null) {
19         String userInfo = user.getUsername();
20         if ("ADMIN".equals(user.getRole().toString())) {
21             userInfo += " (Admin)";
22             addTournamentButton.setVisible(true);
23             addTournamentButton.setManaged(true);
24             removeTournamentButton.setVisible(true);
25             removeTournamentButton.setManaged(true);
26             editTournamentButton.setVisible(true);
27             editTournamentButton.setManaged(true);
28         } else {
29             addTournamentButton.setVisible(false);
30             addTournamentButton.setManaged(false);

```

```
31         editTournamentButton.setVisible(false);
32         editTournamentButton.setManaged(false);
33     }
34     UserRoleLabel.setText(userInfo);
35 }
36
37 listTournaments.getSelectionModel().selectedItemProperty().addListener((
38 observable, oldValue, newValue) -> {
39     if (newValue != null) {
40         selectedTournament = newValue.split("\n")[0].substring("Name: ".length());
41     }
42     choiceLabel.setText(selectedTournament);
43 } else {
44     selectedTournament = null;
45     choiceLabel.setText("No tournament selected");
46 }
47 });
48
49 @FXML
50 private void handleStartButton(ActionEvent event) {
51     try {
52         if (selectedTournament != null && !selectedTournament.isEmpty()) {
53             FXMLLoader loader = new FXMLLoader(getClass().getResource("/
54 TournamentInfo.fxml"));
55             Parent root = loader.load();
56             TournamentInfoController controller = loader.getController();
57             controller.setTournamentName(selectedTournament);
58             Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
59             stage.setScene(new Scene(root));
60         } else {
61             choiceLabel.setText("Please select a tournament!");
62         }
63     } catch (Exception e) {
64         Alert alert = new Alert(Alert.AlertType.ERROR, "Unexpected error: " + e.
65 getMessage());
66         alert.showAndWait();
67     }
68 }
69
70 @FXML
71 private void handleLogout(ActionEvent event) {
72     try {
73         services.AuthService.logout();
74         FXMLLoader loader = new FXMLLoader(getClass().getResource("/Login.fxml"));
75         Parent root = loader.load();
76         Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
77         stage.setScene(new Scene(root));
78     } catch (Exception e) {
79         Alert alert = new Alert(Alert.AlertType.ERROR, "Unexpected error: " + e.
80 getMessage());
81         alert.showAndWait();
82     }
83 }
84
85 @FXML
86 private void handleAddTournament(ActionEvent event) {
```

```

83     Dialog<ButtonType> dialog = new Dialog<>();
84     dialog.setTitle("Add Tournament");
85     dialog.setHeaderText("Enter tournament details:");
86     GridPane grid = new GridPane();
87     grid.setHgap(10);
88     grid.setVgap(10);
89
90     TextField nameField = new TextField();
91     nameField.setPromptText("Name");
92     TextField prizePoolField = new TextField();
93     prizePoolField.setPromptText("Prize Pool");
94     DatePicker startDatePicker = new DatePicker();
95     startDatePicker.setPromptText("Start Date");
96     DatePicker endDatePicker = new DatePicker();
97     endDatePicker.setPromptText("End Date");
98
99     ComboBox<String> organizerComboBox = new ComboBox<>();
100    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
101        List<Organizer> organizers = session.createQuery("FROM models.Organizer",
102            Organizer.class).list();
103        organizerComboBox.setItems(FXCollections.observableArrayList(
104            organizers.stream().map(Organizer::getName).toList()
105        ));
106    } catch (Exception e) {
107        organizerComboBox.setPromptText("No organizers found");
108    }
109    organizerComboBox.setPromptText("Select Organizer");
110
111    ChoiceBox<String> typeChoiceBox = new ChoiceBox<>();
112    typeChoiceBox.getItems().addAll("Online", "Offline");
113    typeChoiceBox.setValue("Online");
114
115    TextField locationField = new TextField();
116    locationField.setPromptText("Location");
117    locationField.setVisible(false);
118    locationField.setManaged(false);
119
120    typeChoiceBox.getSelectionModel().selectedItemProperty().addListener((obs,
121        oldValue, newVal) -> {
122        boolean isOffline = "Offline".equals(newVal);
123        locationField.setVisible(isOffline);
124        locationField.setManaged(isOffline);
125    });
126
127    grid.add(new Label("Name:"), 0, 0);
128    grid.add(nameField, 1, 0);
129    grid.add(new Label("Organizer:"), 0, 1);
130    grid.add(organizerComboBox, 1, 1);
131    grid.add(new Label("Start Date:"), 0, 2);
132    grid.add(startDatePicker, 1, 2);
133    grid.add(new Label("End Date:"), 0, 3);
134    grid.add(endDatePicker, 1, 3);
135    grid.add(new Label("Prize Pool:"), 0, 4);
136    grid.add(prizePoolField, 1, 4);
137    grid.add(new Label("Type:"), 0, 5);
138    grid.add(typeChoiceBox, 1, 5);
139    grid.add(new Label("Location:"), 0, 6);

```

```

138     grid.add(locationField, 1, 6);
139
140     dialog.getDialogPane().setContent(grid);
141     dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK, ButtonType.CANCEL)
142 ;
143     dialog.setResultConverter(dialogButton -> {
144         if (dialogButton == ButtonType.OK) {
145             try (Session session = HibernateUtil.getSessionFactory().openSession())
146             {
147                 session.beginTransaction();
148                 Tournaments tournament = new Tournaments();
149                 tournament.setName(nameField.getText());
150                 String organizerName = organizerComboBox.getValue();
151                 Organizer selectedOrganizer = null;
152                 if (organizerName != null) {
153                     selectedOrganizer = session.createQuery("FROM models.Organizer
154 WHERE name = :name", Organizer.class)
155                         .setParameter("name", organizerName)
156                         .uniqueResult();
157                 }
158                 tournament.setOrganizer(selectedOrganizer);
159                 tournament.setStartDate(startDatePicker.getValue());
160                 tournament.setEndDate(endDatePicker.getValue());
161                 tournament.setPrizePool(prizePoolField.getText());
162                 tournament.setType(typeChoiceBox.getValue());
163                 if ("Offline".equals(typeChoiceBox.getValue())) {
164                     tournament.setLocation(locationField.getText());
165                 } else {
166                     tournament.setLocation(null);
167                 }
168                 session.persist(tournament);
169                 session.getTransaction().commit();
170             } catch (Exception e) {
171                 Alert alert = new Alert(Alert.AlertType.ERROR, "Could not add
172 tournament: " + e.getMessage());
173                 alert.showAndWait();
174             }
175             listTournaments.getItems().clear();
176             try (Session session = HibernateUtil.getSessionFactory().openSession())
177             {
178                 Query<Tournaments> query = session.createQuery("FROM models.
179 Tournaments", Tournaments.class);
180                 List<Tournaments> tournaments = query.list();
181                 for (Tournaments t : tournaments) {
182                     listTournaments.getItems().add(formatTournamentDetails(t));
183                 }
184             } catch (Exception e) {
185                 Alert alert = new Alert(Alert.AlertType.ERROR, "Could not load
186 tournaments: " + e.getMessage());
187                 alert.showAndWait();
188             }
189             return null;
190         });
191         dialog.showAndWait();
192     }
193 }
194

```

```

188     @FXML
189     private void handleRemoveTournament(ActionEvent event) {
190         if (selectedTournament != null) {
191             try (Session session = HibernateUtil.getSessionFactory().openSession()) {
192                 session.beginTransaction();
193                 Query<Tournaments> query = session.createQuery("FROM models.Tournaments
194 WHERE name = :name", Tournaments.class);
195                 query.setParameter("name", selectedTournament);
196                 Tournaments tournament = query.uniqueResult();
197                 if (tournament != null) {
198                     session.remove(tournament);
199                     session.getTransaction().commit();
200
201                     String selectedDetails = listTournaments.getSelectionModel().
202                         getSelectedItem();
203                     listTournaments.getItems().remove(selectedDetails);
204                     choiceLabel.setText("Tournament removed.");
205                 }
206             } catch (Exception e) {
207                 Alert alert = new Alert(Alert.AlertType.ERROR, "Could not remove
208 tournament: " + e.getMessage());
209                 alert.showAndWait();
210             }
211         }
212     }
213
214     @FXML
215     private void handleEditTournament(ActionEvent event) {
216         if (selectedTournament == null || selectedTournament.isEmpty()) {
217             Alert alert = new Alert(Alert.AlertType.WARNING, "Please select a tournament
218 to edit.");
219             alert.showAndWait();
220             return;
221         }
222         Dialog<ButtonType> dialog = new Dialog<>();
223         dialog.setTitle("Edit Tournament");
224         dialog.setHeaderText("Edit tournament details:");
225         GridPane grid = new GridPane();
226         grid.setHgap(10);
227         grid.setVgap(10);
228         TextField nameField = new TextField();
229         nameField.setPromptText("Name");
230         TextField prizePoolField = new TextField();
231         prizePoolField.setPromptText("Prize Pool");
232         DatePicker startDatePicker = new DatePicker();
233         startDatePicker.setPromptText("Start Date");
234         DatePicker endDatePicker = new DatePicker();
235         endDatePicker.setPromptText("End Date");
236         InputStream iconStream = getClass().getResourceAsStream("/figures/dotaIcon.png")
237 ;
238         if (iconStream != null) {
239             ImageView icon = new ImageView(new Image(iconStream));
240             icon.setFitHeight(48);
241             icon.setFitWidth(48);
242             dialog.setGraphic(icon);
243         }
244         ComboBox<String> organizerComboBox = new ComboBox<>();

```

```

240     try (Session session = HibernateUtil.getSessionFactory().openSession()) {
241         List<Organizer> organizers = session.createQuery("FROM models.Organizer",
242             Organizer.class).list();
243         organizerComboBox.setItems(FXCollections.observableArrayList(
244             organizers.stream().map(Organizer::getName).toList()
245         ));
246     } catch (Exception e) {
247         organizerComboBox.setPromptText("No organizers found");
248     }
249     organizerComboBox.setPromptText("Select Organizer");
250     grid.add(new Label("Organizer:"), 0, 1);
251     ChoiceBox<String> typeChoiceBox = new ChoiceBox<>();
252     typeChoiceBox.getItems().addAll("Online", "Offline");
253     typeChoiceBox.setValue("Online");
254     TextField locationField = new TextField();
255     locationField.setPromptText("Location");
256     locationField.setVisible(false);
257     locationField.setManaged(false);
258     typeChoiceBox.getSelectionModel().selectedItemProperty().addListener((obs,
259     oldValue, newVal) -> {
260         boolean isOffline = "Offline".equals(newVal);
261         locationField.setVisible(isOffline);
262         locationField.setManaged(isOffline);
263     });
264     grid.add(new Label("Name:"), 0, 0);
265     grid.add(nameField, 1, 0);
266     grid.add(new Label("Start Date:"), 0, 2);
267     grid.add(startDatePicker, 1, 2);
268     grid.add(new Label("End Date:"), 0, 3);
269     grid.add(endDatePicker, 1, 3);
270     grid.add(new Label("Prize Pool:"), 0, 4);
271     grid.add(prizePoolField, 1, 4);
272     grid.add(new Label("Type:"), 0, 5);
273     grid.add(typeChoiceBox, 1, 5);
274     grid.add(new Label("Location:"), 0, 6);
275     grid.add(locationField, 1, 6);
276     dialog.getDialogPane().setContent(grid);
277     dialog.getDialogPane().getButtonType().addAll(ButtonType.OK, ButtonType.CANCEL)
278 ;
279     dialog.setResultConverter(dialogButton -> {
280         if (dialogButton == ButtonType.OK) {
281             try (Session session = HibernateUtil.getSessionFactory().openSession())
282             {
283                 session.beginTransaction();
284                 Query<Tournaments> query = session.createQuery("FROM models.
285 Tournaments WHERE name = :name", Tournaments.class);
286                 query.setParameter("name", selectedTournament);
287                 Tournaments tournament = query.uniqueResult();
288                 if (tournament != null) {
289                     tournament.setName(nameField.getText());
290                     String organizerName = organizerComboBox.getValue();
291                     Organizer selectedOrganizer = null;
292                     if (organizerName != null) {
293                         selectedOrganizer = session.createQuery("FROM models.
294 Organizer WHERE name = :name", Organizer.class)
295                             .setParameter("name", organizerName)

```

```

291             .uniqueResult();
292         }
293         tournament.setOrganizer(selectedOrganizer);
294         tournament.setStartDate(startDatePicker.getValue());
295         tournament.setEndDate(endDatePicker.getValue());
296         tournament.setPrizePool(prizePoolField.getText());
297         tournament.setType(typeChoiceBox.getValue());
298         if ("Offline".equals(typeChoiceBox.getValue())) {
299             tournament.setLocation(locationField.getText());
300         } else {
301             tournament.setLocation(null);
302         }
303         session.merge(tournament);
304         session.getTransaction().commit();
305     }
306     } catch (Exception e) {
307         Alert alert = new Alert(Alert.AlertType.ERROR, "Could not edit
tournament: " + e.getMessage());
308         alert.showAndWait();
309     }
310     listTournaments.getItems().clear();
311     try (Session session = HibernateUtil.getSessionFactory().openSession())
{
312         Query<models.Tournaments> query = session.createQuery("FROM models.
Tournaments", models.Tournaments.class);
313         List<models.Tournaments> tournaments = query.list();
314         for (Tournaments t : tournaments) {
315             listTournaments.getItems().add(formatTournamentDetails(t));
316         }
317     } catch (Exception e) {
318         Alert alert = new Alert(Alert.AlertType.ERROR, "Could not load
tournaments: " + e.getMessage());
319         alert.showAndWait();
320     }
321 }
322     return null;
323 });
324 try {
325     try (Session session = HibernateUtil.getSessionFactory().openSession()) {
326         Query<Tournaments> query = session.createQuery("FROM models.Tournaments
WHERE name = :name", Tournaments.class);
327         query.setParameter("name", selectedTournament);
328         Tournaments tournament = query.uniqueResult();
329         if (tournament != null) {
330             nameField.setText(tournament.getName());
331             prizePoolField.setText(tournament.getPrizePool());
332             startDatePicker.setValue(tournament.getStartDate());
333             endDatePicker.setValue(tournament.getEndDate());
334             organizerComboBox.setValue(tournament.getOrganizer() != null ?
tournament.getOrganizer().getName() : null);
335             typeChoiceBox.setValue(tournament.getType());
336             if ("Offline".equals(tournament.getType())) {
337                 locationField.setText(tournament.getLocation());
338                 locationField.setVisible(true);
339                 locationField.setManaged(true);
340             } else {
341                 locationField.setVisible(false);

```

```

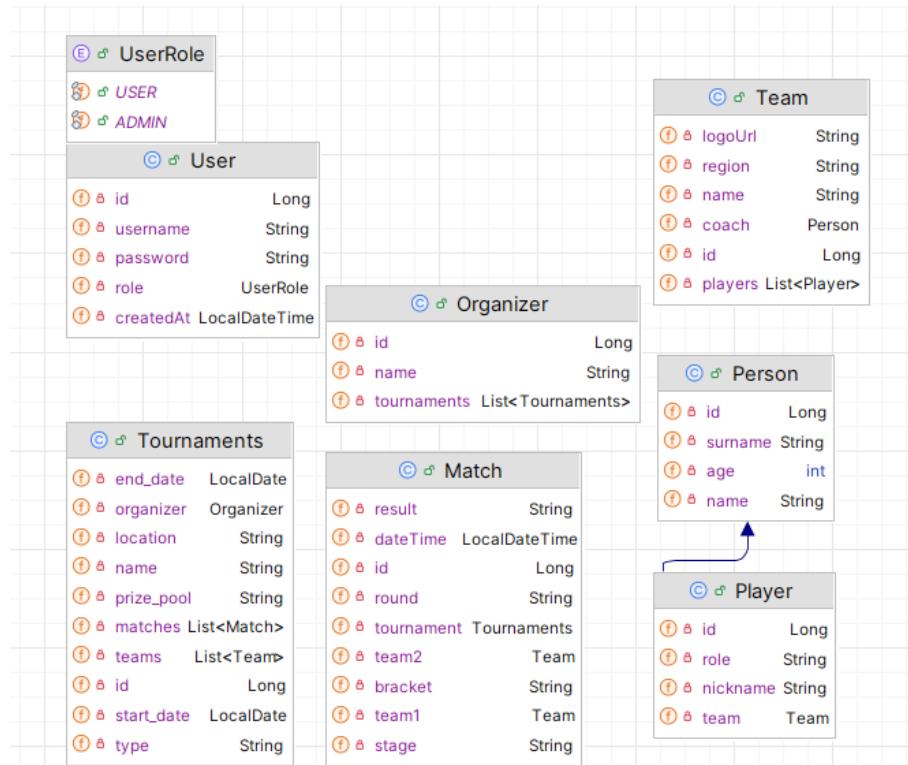
342         locationField.setManaged(false);
343     }
344   }
345 }
346 } catch (Exception e) {
347     Alert alert = new Alert(Alert.AlertType.ERROR, "Could not load tournament
348 details: " + e.getMessage());
349     alert.showAndWait();
350 }
351 dialog.showAndWait();
}

```

3.1.4. Models

Pakiet models zawiera klasy, które są mapowane na tabele w bazie danych.

Rys. 3.4. Pola i dziedziczenie klas, każda klasa ma gettery i settery, a także metodę ToString



Listing 3.8. Przykład mapowania klas na klasie Tournaments

```

1 @Entity
2 @Table(name = "tournaments")
3 public class Tournaments {
4     @Id
5     @GeneratedValue(strategy = GenerationType.IDENTITY)
6     private Long id;
7
8     @Column(nullable = false, unique = true)
9     private String name;
10
11    @ManyToOne

```

```

12     @JoinColumn(name = "organizer_id")
13     private Organizer organizer;
14
15     @ManyToMany
16     @JoinTable(
17         name = "tournament_teams",
18         joinColumns = @JoinColumn(name = "tournament_id"),
19         inverseJoinColumns = @JoinColumn(name = "team_id")
20     )
21     private List<Team> teams;
22
23     @OneToMany(mappedBy = "tournament", cascade = CascadeType.ALL, orphanRemoval = true)
24     private List<Match> matches;
25
26     @Column
27     private LocalDate start_date;
28     @Column
29     private LocalDate end_date;
30     @Column
31     private String prize_pool;
32     @Column
33     private String type;
34     @Column
35     private String location;

```

3.1.5. Resources

Pakiet resources zawiera pliki fxml, które są odpowiedzialne za wygląd aplikacji. Każdy plik jest rozpatrzony w następnym rozdziale.

Zawiera plik css, który daje ten ładny wygląd aplikacji, jest on wykorzystywany w każdym pliku fxml, aby zapewnić spójność stylu. Jest folder figures z obrazkami.

3.2. Wykorzystane Technologie

W projekcie wykorzystano następujące technologie:

- **IntelliJ IDEA 2025.1.2** - IDE do tworzenia aplikacji w języku Java.
- **Maven 4.0.0** - budowanie projektu i zarządzanie zależnościami.
- **JavaFX 21.0.7** - projektowanie GUI.
- **JavaFX SceneBuilder** - projektowanie GUI.
- **PostgreSQL 17.5** - baza danych.
- **Hibernate 6.2.7** - warstwa ORM.
- **BCrypt** - bezpieczne hashowanie haseł.

3.2.1. Wymagania sprzętowe

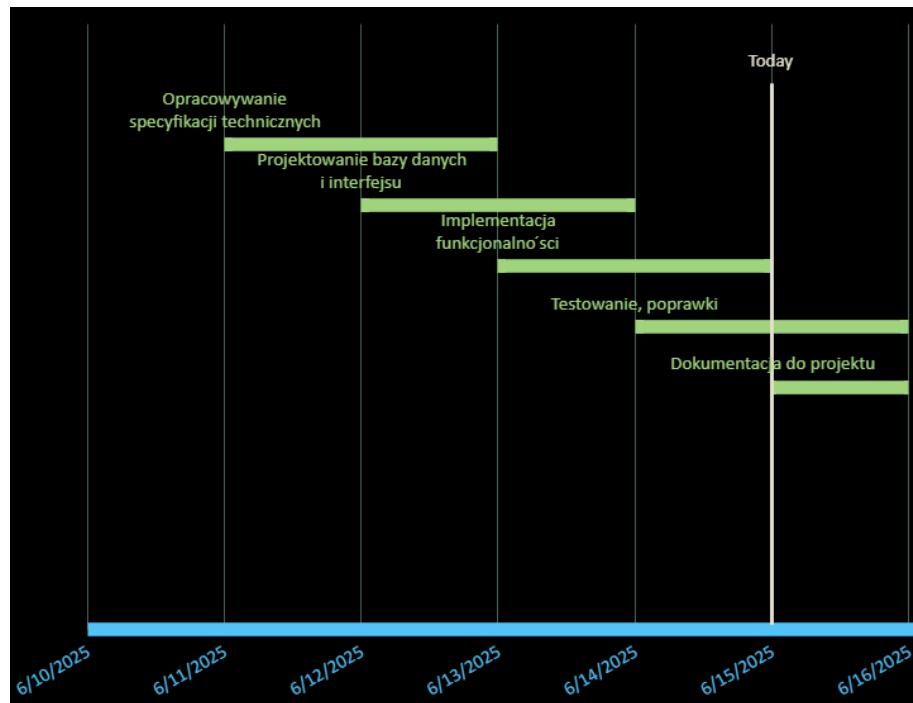
Aby uruchomić aplikację, wymagany jest komputer z systemem operacyjnym Windows 10/11, IntelliJ Idea, Java 21, PostgreSQL i JavaFX 21.0.7.

Wszystko trzeba poprawnie skonfigurować, aby aplikacja działała poprawnie. W razie potrzeby wrzucę filmik z instrukcją repozytorium.

4. Harmonogram Realizacji Projektu

Najtrudniejszym było zapoznać się z nowymi technologiami, takimi jak JavaFX i Hibernate, oraz LaTeX, ponieważ wcześniej pracowałem głównie z JDBC i Swing. Jednakże, dzięki książkom, tutorialom na YouTube, stackoverflow, udało mi się szybko opanować podstawy tych technologii.

Odpowiednio do wymagań, stworzyłem diagram Gantta, który widziano poniżej:



Rys. 4.1. Diagram Gantta

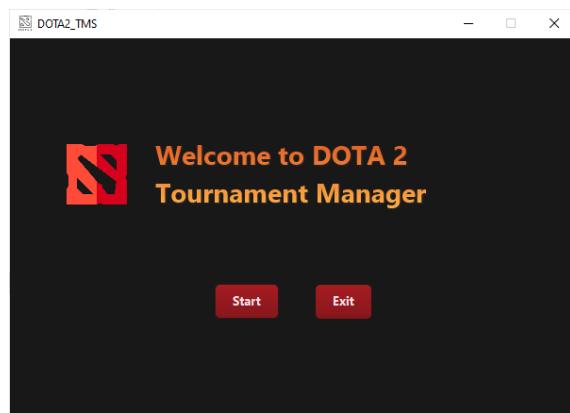
4.0.1. Repozytorium

Kod źródłowy projektu jest dostępny w repozytorium GitHub pod adresem: <https://github.com/kaylor1x/ProjectJava1rok>

5. Prezentacja warstwy użytkowej projektu

5.1. Welcome.fxml

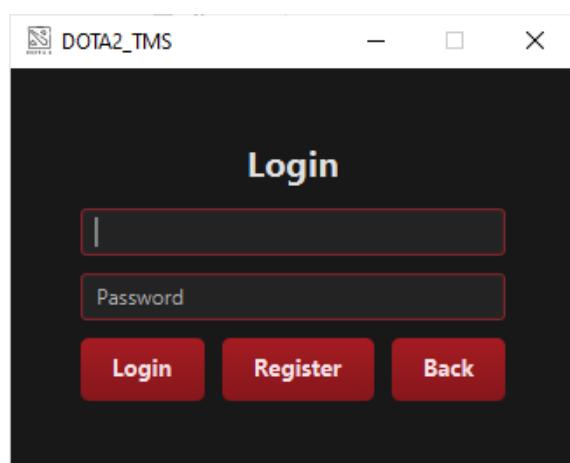
Pierwszy ekran, można nacisnąć exit i wyjść lub start i zacząć logowanie.



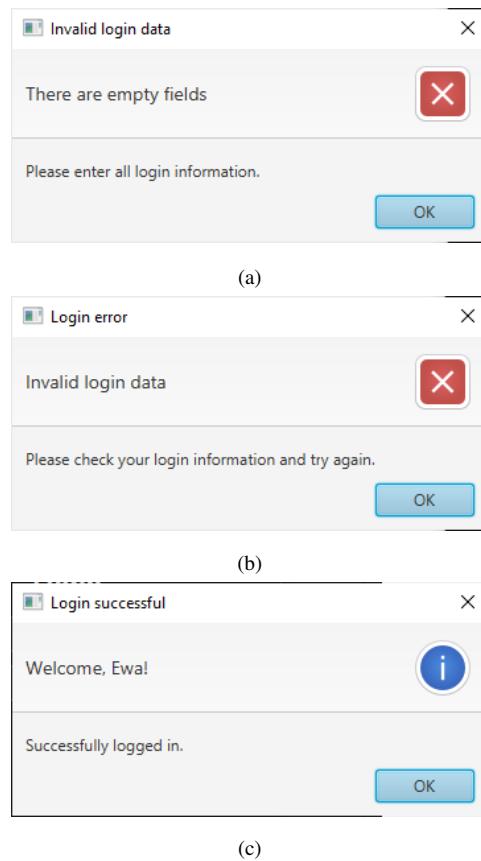
Rys. 5.1. Ekran powitalny

5.2. Login.fxml

Ekran logowania, gdzie można się zalogować lub zarejestrować lub wrócić do ekranu powitalnego.



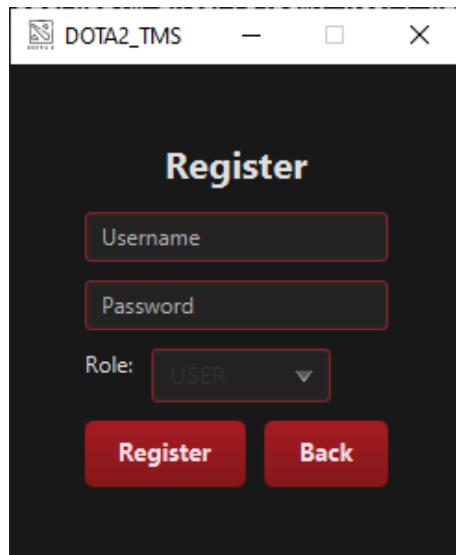
Rys. 5.2. Ekran logowania



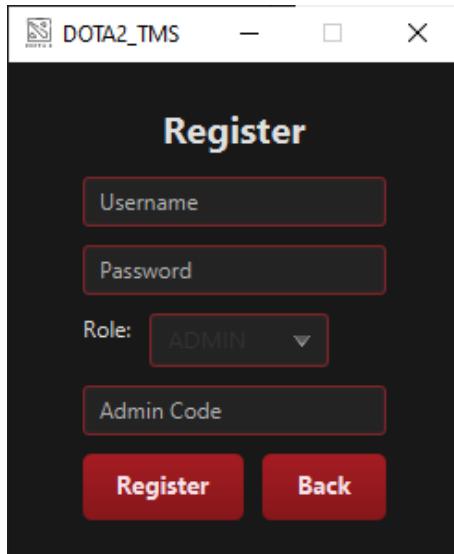
Rys. 5.3. Błędne i Udane logowania: (a) Empty fields error, (b) Invalid login data(nieznaleziono w db), (c) Successful login.

5.3. Register.fxml

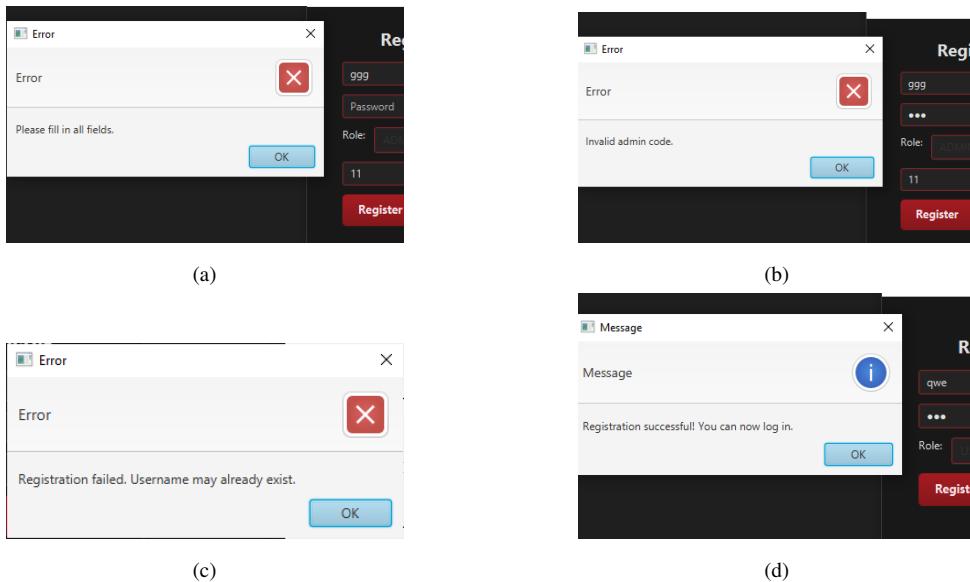
Ekran rejestracji, gdzie można się zarejestrować lub wrócić do ekranu logowania. Przy logowaniu możesz wybrać rolę.



Rys. 5.4. Ekran rejestracji z wybranym User



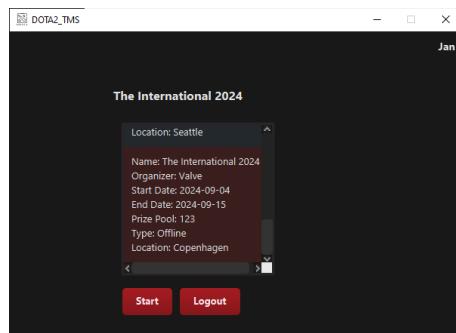
Rys. 5.5. Ekran rejestracji z wybranym Admin



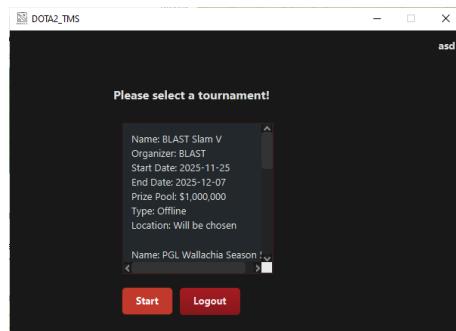
Rys. 5.6. Błędne i Udane rejestracji: (a) Empty fields error, (b) Invalid admin code, (c) User already exists, (d) Successful registration.

5.4. TournamentSelection.fxml

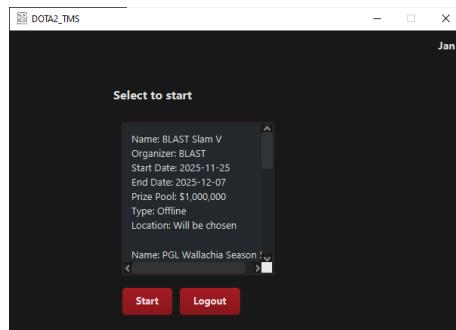
Ekran wyboru turnieju, gdzie można wybrać dostępny turniej lub wylogować się i wrócić do ekranu logowania. Jeżeli jesteś adminem, możesz dodać, edytować lub usunąć turniej.



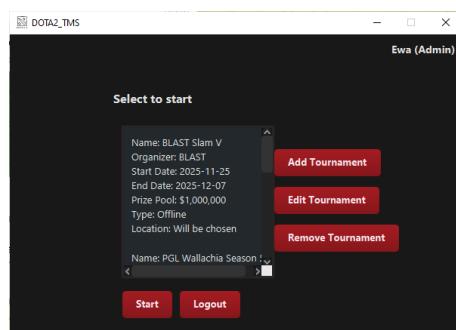
Rys. 5.9. Gdy klikasz na turniej górny label zmienia się



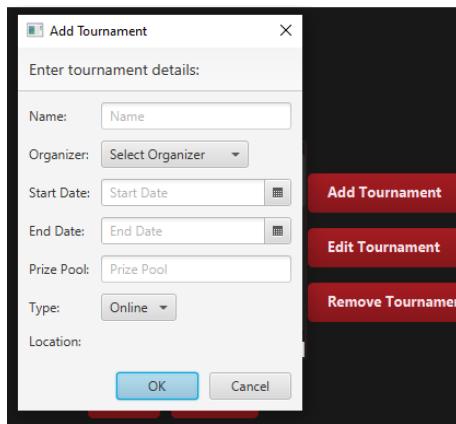
Rys. 5.10. Gdy klikasz start, prosi o wyborze turnieju



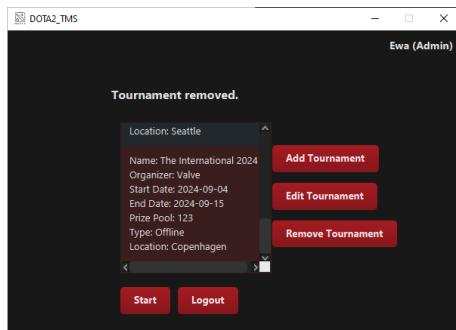
Rys. 5.7. Ekran wyboru turnieju jako User



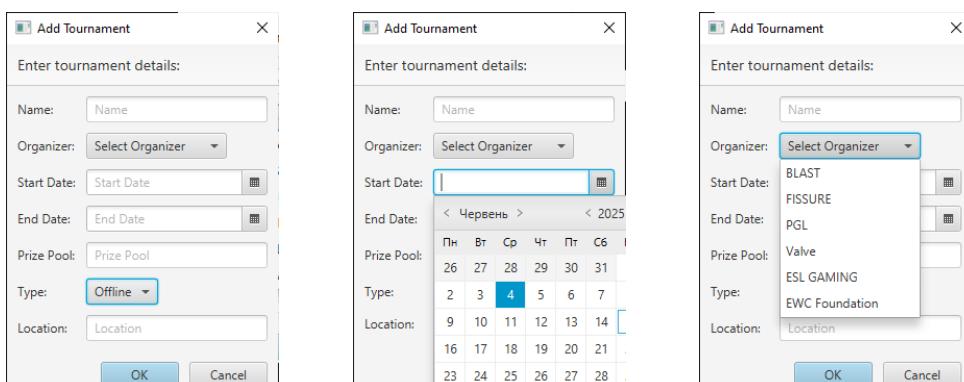
Rys. 5.8. Ekran wyboru turnieju jako Admin



Rys. 5.12. Gdy klikasz add, otwiera się okno z formularzem do dodania turnieju

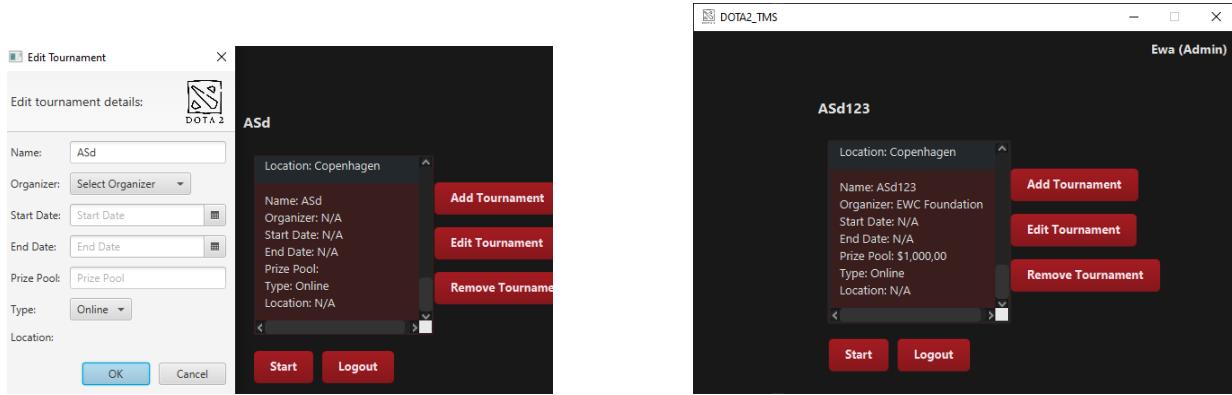


Rys. 5.11. Gdy klikasz remove, wyrzuca go z bazy danych i tabeli, zmienia się górny label



(a) Typ offline, pojawia sie miejsce pro- (b) Możesz wybrać dowolną datę rozpo- (c) Możesz wybrać organizatora tur-
wadzenia wczęcia turnieju nieju

Rys. 5.13. Formularz do dodania lub edycji turnieju: (a), (b), (c).



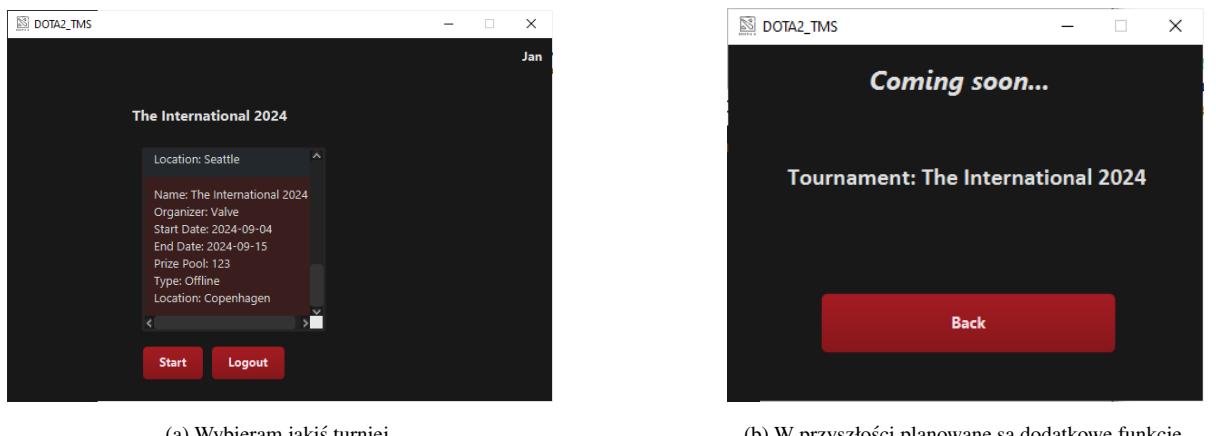
(a) Przy edycji już wprowadzone dane wypełniają odpowiadające im pola

(b) Wprowadziłem jakieś zmiany w formularzu

Rys. 5.14. Edycja turnieju: (a), (b).

5.5. TournamentInfo.fxml

Ekran informacji o turnieju, nie ma jeszcze żadnych odpowiednich funkcji, tylko Back żeby wrócić do wyboru.



(a) Wybieram jakiś turniej

(b) W przyszłości planowane są dodatkowe funkcje

Rys. 5.15. Ekran informacji o turnieju: (a), (b).

6. Podsumowanie

Ten projekt jest moim pierwszym projektem w ogóle, zdecydowałem się zaimplementować go w JavaFX i Hibernate, z którymi jestem mniej zaznajomiony, ponieważ w laboratoriach używaliśmy JDBC i Swing, które moim zdaniem są łatwiejsze. Ale tak czy inaczej, do tej pory zaimplementowałem następujące funkcje:

- Wygodny i odpowiadający tematowi DOTA 2 interfejs użytkownika.
- Rejestracja, logowanie użytkowników.
- Zarządzanie poziomami dostępu.
- Haszowanie haseł użytkowników w bazie danych.
- Dodawanie, edytowanie i usuwanie turniejów (wszystko jest połączone z db).
- Przeglądanie listy turniejów oraz szczegółowych informacji o każdym z nich.
- Wybór typu turnieju (jeżeli offline - trzeba wprowadzić lokalizację) i organizatora turnieju z listy dostępnych (tylko dla administratora).

W przyszłości planuję dodać więcej funkcji, takich jak:

- Po wyborze turnieju, zwiększyć funkcjonalności dla użytkownika widzieć informacje o drużynach, graczech, meczach tego turnieju.
- Dla administratora możliwość zarządzania wszystkimi aspektami turnieju.
- Dodać możliwość przeglądania nowości o turniejach DOTA 2(web integracja).
- Ulepszenie już zrobionych funkcji.

Spis rysunków

3.1	Struktura całego projektu	8
3.2	Baza danych	9
3.3	Metody kontrolerów	11
3.4	Pola i dziedziczenie klas, każda klasa ma gettery i settery, a także metodę ToString	21
4.1	Diagram Gantta	23
5.1	Ekran powitalny	24
5.2	Ekran logowania	24
5.3	Błędne i Udane logowania: (a) Empty fields error, (b) Invalid login data(nieznaleziono w db), (c) Successful login.	25
5.4	Ekran rejestracji z wybranym User	25
5.5	Ekran rejestracji z wybranym Admin	26
5.6	Błędne i Udane rejestracji: (a) Empty fields error, (b) Invalid admin code, (c) User already exists, (d) Successful registration.	26
5.9	Gdy klikasz na turniej górny label zmienia się	27
5.10	Gdy klikasz start, prosi o wybór turnieju	27
5.7	Ekran wyboru turnieju jako User	27
5.8	Ekran wyboru turnieju jako Admin	27
5.12	Gdy klikasz add, otwiera się okno z formularzem do dodania turnieju	28
5.11	Gdy klikasz remove, wyrzuca go z bazy danych i tabeli, zmienia się górny label	28
5.13	Formularz do dodania lub edycji turnieju: (a), (b), (c).	28
5.14	Edycja turnieju: (a), (b).	29
5.15	Ekran informacji o turnieju: (a), (b).	29

Spis listingów

3.1	Main: jak zaczyna się program	9
3.2	Klasa HibernateUtil: odpowiedzialna za konfigurację bazy danych	10
3.3	Klasa AuthService: autoryzacja z wyszukiwaniem w bazie konta i dehashowaniem hasła BCryptem	10
3.4	Klasa AuthService: rejestracja ze sprawdzeniem czy istnieje już konto i hashowanie hasła nowego użytkownika	10
3.5	Klasa LoginController: handleBack i handleRegister - przekazania na nowe strony; handleLogin - logowanie użytkownika z wykorzystaniem wyżej opisanym AuthService dla autentyfikacji usera i przy poprawnych danych - przekazanie na nową stronę	11
3.6	Klasa RegisterController: handleRegister - rejestracja nowego użytkownika z wykorzystaniem AuthService; umożliwia rejestrację jako Admin za pomocy sprawdzenia adminCode-Field (admin code	13
3.7	Klasa TournamentSelectionController: w tej klasie przy inicjalizacji jest kontrola roli użytkownika i zrealizowane CRUD na turniejach(używamy HibernateUtil opisany wyżej); W przyszłości ulepszę tę klasę	14
3.8	Przykład mapowania klas na klasie Tournaments	21

Załącznik nr 2 do Zarządzenia nr 228/2021 Rektora Uniwersytetu Rzeszowskiego z dnia 1 grudnia 2021 roku w sprawie ustalenia procedury antyplagiatowej w Uniwersytecie Rzeszowskim

OŚWIADCZENIE STUDENTA O SAMODZIELNOŚCI PRACY

.....Yevhenii Ušik

Imię (imiona) i nazwisko studenta

Wydział Nauk Ścisłych i Technicznych

.....Informatyka

Nazwa kierunku

.....134980.....

Numer albumu

1. Oświadczam, że moja praca projektowa pt.: Przygotowanie dokumentacji do projektu w systemie **LATEX**
 - 1) została przygotowana przeze mnie samodzielnie*,
 - 2) nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2021 r., poz. 1062) oraz dóbr osobistych chronionych prawem cywilnym,
 - 3) nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,
 - 4) nie była podstawą otrzymania oceny z innego przedmiotu na uczelni wyższej ani mnie, ani innej osobie.
2. Jednocześnie wyrażam zgodę/nie wyrażam zgody** na udostępnienie mojej pracy projektowej do celów naukowo–badawczych z poszanowaniem przepisów ustawy o prawie autorskim i prawach pokrewnych.

Rzeszów , 15.06.2025

(miejscowość, data)

Yevhenii Ušik

(czytelny podpis studenta)

* Uwzględniając merytoryczny wkład prowadzącego przedmiot

** – niepotrzebne skreślić