Criteria C: Development

The complex techniques used in the project are noted below.

Contents

Development Environment

The system I created consists of 2 parts: a desktop application built on the JavaFX platform, and a database server on the MySQL platform. To achieve this I utilised the following software: **NetBeans IDE**[1], **Gluon Scene Builder**[2] and **MySQL Workbench**[3].

NetBeans IDE was used to program and live test the Java code of the application. Gluon Scene Builder is a drag & drop, rapid application development software that is used to create FXML files (XML-based user interface markup language for JavaFX platform[4]). The elements in the user interface were assigned uniform attributes using Cascading Stylesheets (CSS). In figure 1, the editable FXML form is displayed, the left-side pane shows the node hierarchy, and the right-side pane shows a node's attributes. Nodes can be assigned CSS sheets, such as the blue buttons in figure 1 have the same CSS sheet which gives the application a uniform appearance.

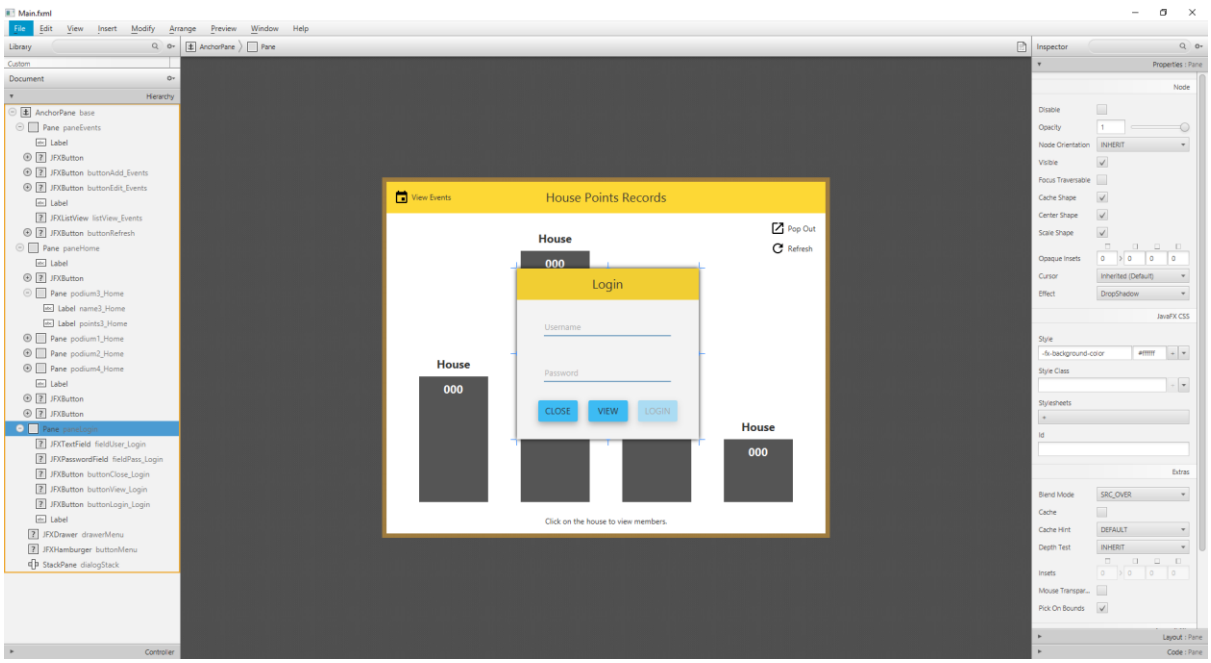

Figure 1: Gluon Scene Builder – FXML Visual Designer.

---

[1] Website: netbeans.org/
[2] Website: gluonhq.com/products/scene-builder/
[3] Website: www.mysql.com/products/workbench/
[4] Taken from: http://docs.oracle.com/javase/8/javafx/fxml-tutorial/why_use_fxml.htm

Database for backend storage

MySQL Workbench was used to create the MySQL database and relevant tables, and monitor whether the application can correctly access and modify the data.

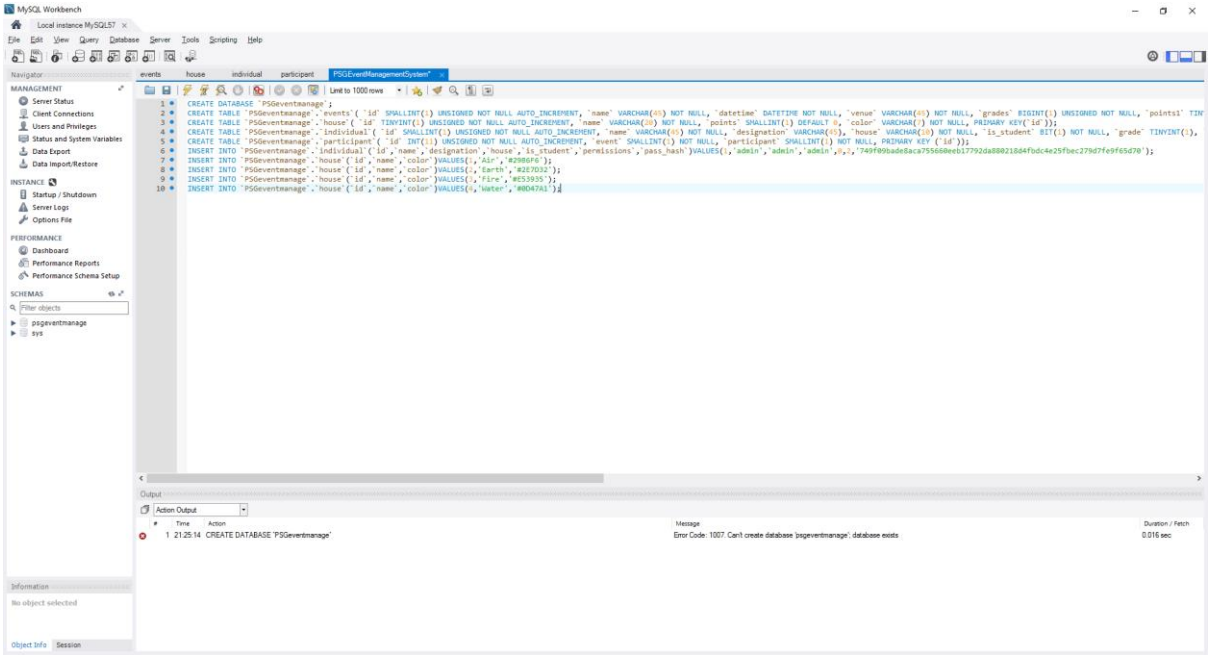

Figure 2: MySQL Workbench – Database setup SQL file.

Object Oriented Programming features

Encapsulation

Encapsulation refers to the practise of including variables and methods into a single component that is private from all other classes. The data from these classes is accessed using accessor methods.

Inheritance

Inheritance is a relationship between two object that signifies that one object (subclass) is a specialised form of another object (superclass). This allows the subclass to use all the variables and methods of the superclass, along with any that it declares.

Polymorphism

Polymorphism is the ability to perform a single action in different ways depending on the requirements. There are two types of polymorphism: static and dynamic. Static polymorphism, also known as Overloading, is when methods of the same name have different parameters (figure 4). This increases the “readability” of the code and helps a new developer understand the purpose of a method. Dynamic polymorphism, also known as Overriding, is when a subclass overrides the actions of the superclass (figure 3). This allows for execution of actions specific to the subclass.

```
9   //    Inheritance
10  public class CellPersonController implements Initializable {
11
12      @FXML
13      private Label name;
14      @FXML
15      private Label grade;
16      @FXML
17      private Label points;
18
19  //    Dynamic Polymorpism
20      @Override
21      public void initialize(URL url, ResourceBundle rb) {
22      }
23
24  //    Encapsulation
25      public void setGrade(String grade){
26          this.grade.setText(grade);
27      }
28
29      public void setName(String name){
30          this.name.setText(name);
31      }
32
33      public void setPoints(int points){
34          this.points.setText(String.valueOf(points));
35      }
36  }
37
```

Figure 3: NetBeans IDE – Encapsulation, Inheritance, Overriding Polymorphism

```
135     private void launchPersonEditor() { //No parameter (Polymorphism)
136         try {
137             Stage stage = new Stage();
138             stage.initModality(Modality.APPLICATION_MODAL);
139             FXMLLoader loader = new FXMLLoader(getClass().getResource("/FXML/PersonRecordEditor.fxml"));
140             Parent root = loader.load();
141             Scene scene = new Scene(root);
142             scene.getStylesheets().add("/CSS/Main.css");
143             stage.getIcons().add(new Image("images/PW_Symbol.jpg"));
144             stage.setTitle("PSG Event Management System - Editing Person");
145             stage.setScene(scene);
146             stage.setResizable(false);
147             stage.sizeToScene();
148             stage.show();
149         } catch (IOException ex) {
150             Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, null, ex);
151         }
152     }
153
154     private void launchPersonEditor(int id) { //With parameter (Polymorphism)
155         try {
156             Stage stage = new Stage();
157             stage.initModality(Modality.APPLICATION_MODAL);
158             FXMLLoader loader = new FXMLLoader(getClass().getResource("/FXML/PersonRecordEditor.fxml"));
159             Parent root = loader.load();
160             Scene scene = new Scene(root);
161             scene.getStylesheets().add("/CSS/Main.css");
162             PersonRecordEditorController personRec = loader.getController();
163
164             ResultSet person = SQL.loadPerson(id);
165             person.first();
```

Figure 4: NetBeans IDE – Static Polymorphism / Overloading

Parsing Files

The application's settings are stored on a text file. The program parses this text file to create a connection with the database. In the menu, the school's logo is displayed, this is done by using a node called ImageView, which parses an image file. Furthermore, most of the buttons in the User Interface have a graphic attached. These graphics were taken from Google's open source repository of icons[5].

Figure 5: Some graphics added to buttons.

Hierarchical Composite Data Structure

Data returned by SQL queries is stored in a data structure for SQL known as a Result Set. This is an Array List of records from the database that have been retrieved using the SELECT statement.

```
private static void updatePoints(int EvID, int win1, int point1, int win2, int point2, int win3, int point3, int win4, int point4){
    try {
        //House
        ResultSet house = conn.createStatement().executeQuery("SELECT id, points FROM house;");
        while (house.next()){
            if (house.getInt(1) == win1){
                int val = house.getInt(1)+point1;
                conn.createStatement().executeUpdate("UPDATE house SET points = "+ val +" WHERE id = "+ win1 +";");
            }
            if (house.getInt(1) == win2){
                int val = house.getInt(1)+point2;
                conn.createStatement().executeUpdate("UPDATE house SET points = "+ val +" WHERE id = "+ win2 +";");
            }
            if (house.getInt(1) == win3){
                int val = house.getInt(1)+point3;
                conn.createStatement().executeUpdate("UPDATE house SET points = "+ val +" WHERE id = "+ win3 +";");
            }
            if (house.getInt(1) == win4){
                int val = house.getInt(1)+point4;
                conn.createStatement().executeUpdate("UPDATE house SET points = "+ val +" WHERE id = "+ win4 +";");
            }
        }
        //Participants
        ResultSet participants = participantSelection(EvID);
        while (participants.next()){
            ResultSet person = conn.createStatement().executeQuery("SELECT points, house FROM individual WHERE id = "+ participants.getInt(1) +";");
            person.next();
            int houseNum = houseInt(person.getString(2));
            int val = 0;
            if (houseNum == win1){
                val = person.getInt(1)+point1;
            }
            if (houseNum == win2){
                val = person.getInt(1)+point2;
            }
            if (houseNum == win3){
                val = person.getInt(1)+point3;
            }
```

Figure 6: Result Set usage

[5] Icons downloaded from: material.io/icons/

Additional Libraries

I used a visual library of JavaFX elements, JFoenix[6], which is designed based on Google's Material Design Specifications. I chose this because it is created with the aim of having reactive User Interfaces that are simple to follow.

Since this library contains many advanced features, I followed a tutorial for the implementation of this library by Genuine Coder[7]. This helped me in implementing the kind of features that I had envisioned.

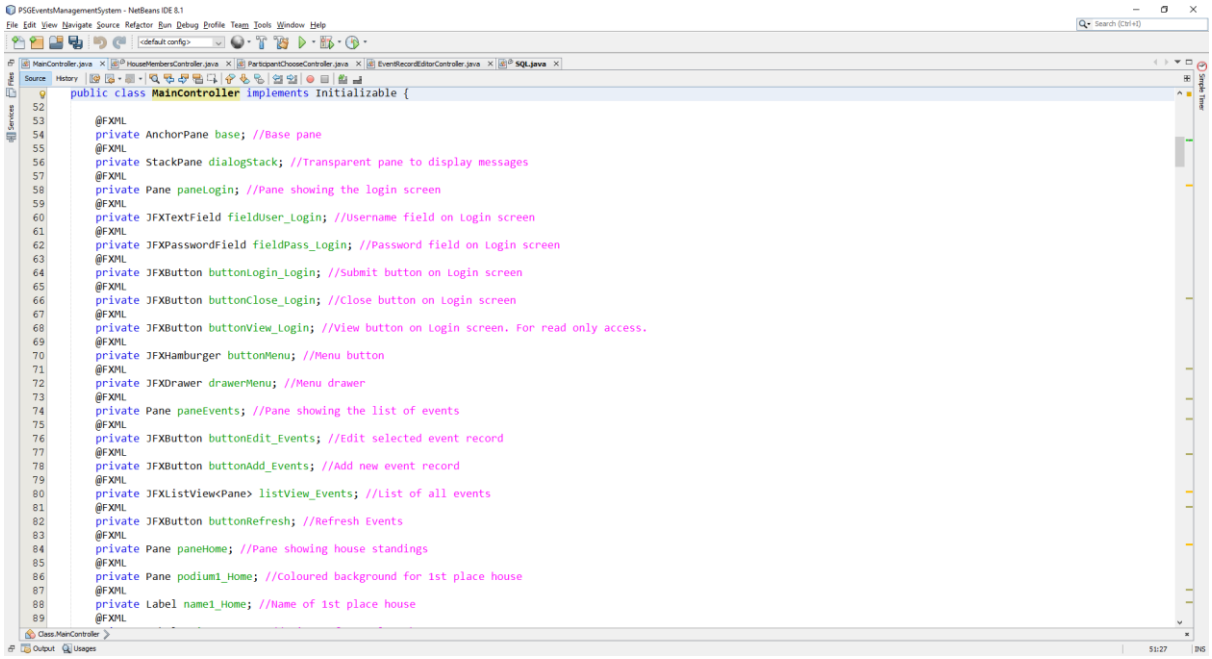

Figure 7: Nodes implemented from library. All nodes "JFX…" are from the JFoenix library.

Animations

To better implement the Material Design, I added animations to the transition between the various screens of the main FXML class. This is where the login, home and events list panes are present.

---

[6] Website: *jfoenix.com* and open source code: github.com/jfoenixadmin/JFoenix
[7] https://www.youtube.com/playlist?list=PLhs1urmduZ29LNYi_MaoU60JemQ6Aei6A

Password Hashing

For the purpose of maintaining security, all passwords that are stored in the database are hashed using SHA256. This is a one way function, meaning that once converted into cypher-text, the data cannot be retrieved.

```
//            On button click, login.
            buttonLogin_Login.setOnAction(Event -> {
                final String hashed = Hashing.sha256().hashString(fieldPass_Login.getText(), StandardCharsets.UTF_8).toString(); //Convert password input into cypertext
                String login = SQL.login(fieldUser_Login.getText(), hashed); //Verify credentials from SQL Database
                fieldPass_Login.setText(""); //Clear password field
//                Scene trasition animations
                ScaleTransition scale = new ScaleTransition(Duration.millis(300));
                scale.setInterpolator(Interpolator.EASE_BOTH);
                JFXSnackbar msg_Login = new JFXSnackbar(dialogStack); //Message container
                if (login.substring(0, 5).equals("match")) {
                    fieldUser_Login.setText(""); //Clear username field
                    scale.setNode(paneLogin);
                    scale.setByX(-1);
                    scale.setByY(-1);
                    scale.play(); //Run animation
                    scale.setOnFinished(e1 -> { //Run following code after the animation ends
                        paneLogin.setVisible(false);
                        scale.setNode(paneHome);
                        scale.setDuration(Duration.ONE);
                        scale.play();
                        scale.setOnFinished(e2 -> {
                            scale.setToX(1.0);
                            scale.setToY(1.0);
                            scale.setDuration(Duration.millis(300));
                            paneHome.setVisible(true);
                            visiblePane = paneHome;
                            scale.play();
                            scale.setOnFinished(e3 -> {
                                msg_Login.show("Login Successful. Access Level: "+login.substring(5, 6).replaceAll("1", "Edit Only").replaceAll("2", "Add/Edit Only").←
replaceAll("3", "Full Access"), 2000);
                                buttonMenu.setVisible(true);
                                drawerMenu.setVisible(true);
                                accessLevel(login.substring(5, 6));
                            });
                        });
                    });
                } else {
```

Figure 8: Actions when the login button is pressed. Password is hashed and compared to the database. Animations of transitions are created and executed,

Word Count: 820

Bibliography

Dimitriou, Kostas and Markos Hatzitaskos. *Core Computer Science*. Berkshire: Express Publishing, 2015. Book.

Gluon. *Scene Builder - Gluon*. n.d. Web. 25 Feb 2017. <http://gluonhq.com/products/scene-builder/>.

Meyer, David. *JavaFX 2.0 arrives and heads for open source | ZDNet*. 6 Oct 2011. Article. 25 Feb 2017. <http://www.zdnet.com/article/javafx-2-0-arrives-and-heads-for-open-source/>.

MySQL. *MySQL :: MySQL Workbench*. n.d. Web. 25 Feb 2017. <https://www.mysql.com/products/workbench/>.

Oracle. *Welcome to NetBeans*. n.d. Web. 25 Feb 2017.