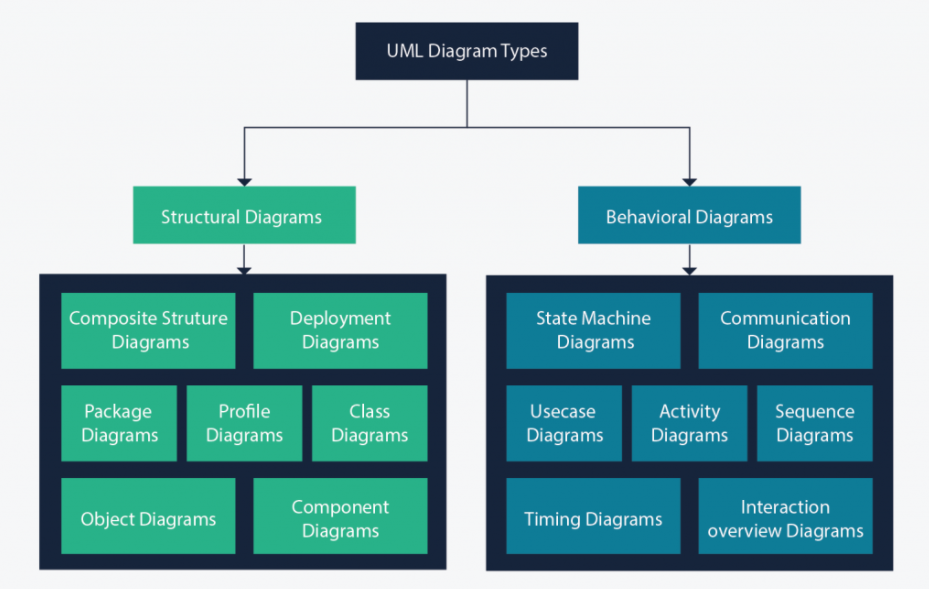
GROUP 10

INVENTION MANAGEMENT SYSTEM

* The Invention Management System Database is made to store,
  + - * Invention details
      * Inventor details
      * Awards Received etc.

**UML DIAGRAM:**



**Structural diagrams:**

* Structural diagrams show the things in the modeled system.
* In a more technical term, they show different objects in a system.

**Behavioral diagrams:**

* Behavioral Diagrams show what should happen in a system.
* They describe how the objects interact with each other to create a functioning system.

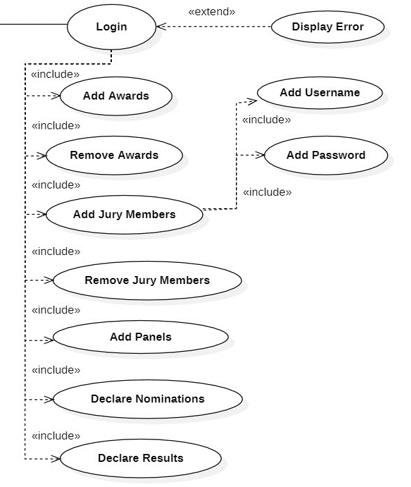
**Use Case Diagram:**

* Use case diagrams give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions interact.
* It’s a great starting point for any project discussion because you can easily identify the main actors involved and the main processes of the system.
* This Use Case Diagram depicts the High-level view of the Invention Management system.
* It also provides the scenarios in which the application interacts with,
  + - * Inventor
      * Jury
      * Admin

|  |  |
| --- | --- |
| **Actor Category** | **Actor** |
| Primary Actor | Jury, Admin |
| Secondary Actor | Inventor |

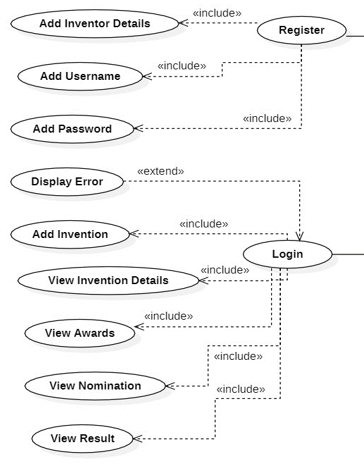
There are total of Twenty-Six use cases that represent the specific functionality of Invention Management System.

Each actor interacts with a particular use case.

****

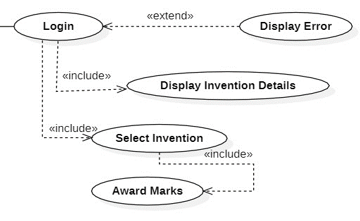
**Functionality of Admin:**

* Login to take the overall Control of the Data Base
* Add or Remove Awards
* Add or Remove Panel
* Add or Remove Jury
* Declare Nominations
* Declare Results
* Admin takes the overall control of the database or in other words say one of the primary Actors.
* Admin have to just login inside the database and gets the overall control.

****

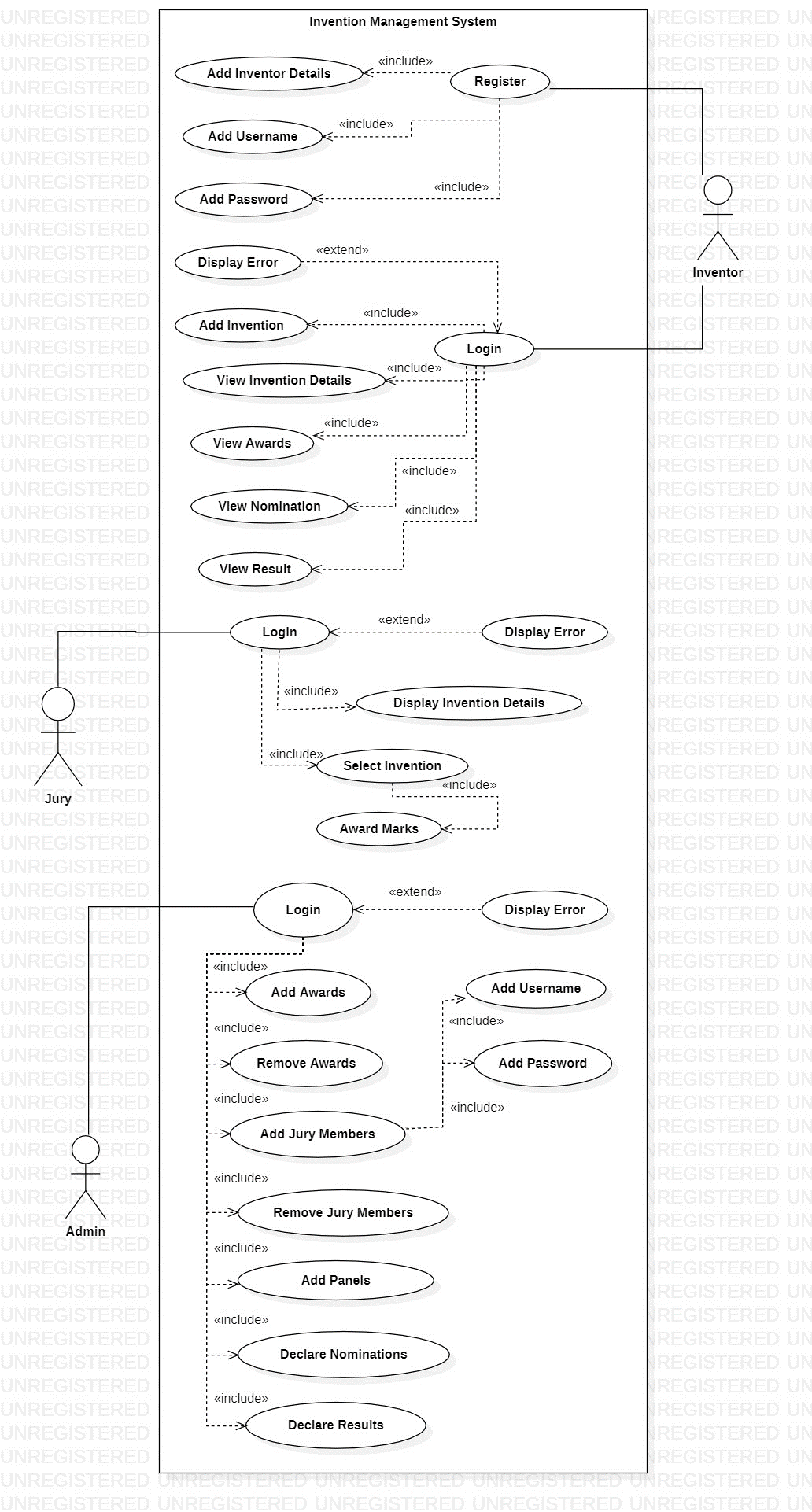
**Functionality of Inventor Actor:**

* Register in the database to create Account
* Register for New Username and Password
* Login using his/her credentials
* Add the details of his/her invention
* View the Invention details
* View Awards
* View Nominations
* View Results
* Inventor can create the account in the database to register his/her invention.
* Once registered Inventor needs to set the username and password to login into the database next time.
* Once Inventor Actor successfully logins, gets option to add the invention to the database.
* If Inventor is not able to login successfully then, they will get the Error message.
* Inventor can also view the invention details that he/she have enclosed while registering in the database.
* Inventor can view the results of the invention.

****

**Functionality of Jury Actor:**

* Login to Invention Management System
* View Invention Details that are Displayed when logged In.
* Select the Invention
* Award Marks for the Selected Invention.
* Jury can login to the Invention management system using the credentials provided by the Admin.
* Once they successfully login inside the Portal they get access to all invention.
* If they are not able to login successfully then they will get the Error message.
* From the list of inventions given they can select the allotted invention.
* They can look into the inventions and award marks to the selected Invention.



**Class Diagram:**

* Class diagrams are the main building block of any object-oriented solution.
* It shows the classes in a system, attributes, and operations of each class and the relationship between each class.
* In most modeling tools, a class has three parts.
* Name at the top, attributes in the middle and operations or methods at the bottom.
* In a large system with many related classes, classes are grouped together to create class diagrams.
* Different relationships between classes are shown by different types of arrows.

**Public (+):**

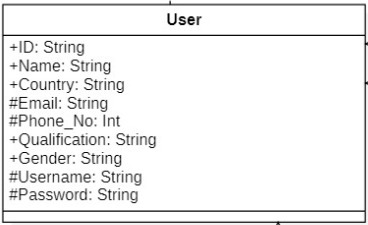
* Public members are visible to all other classes.
* This means that any other class can access a public field or method.
* Further, other classes can modify public fields unless the field is declared as final.

**Protected (#):**

* The **protected** keyword is an access modifier used for attributes, methods and constructors, making them accessible in the same package and subclasses.

**Private (-):**

* The methods or data members declared as private are accessible only within the class in which they are declared.
* The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
* Any other class of the same package will not be able to access these members.



* **User** is Implemented from the Interface **Login Details.**

**+ ID:**

* Unique ID to identify the User.
* It is of type String.

**+ Name:**

* Name of the User.
* It is of type String.

**+ Country:**

* Country in which the user resides.
* It is of type String.

**+ Qualification:**

* An [experience](https://dictionary.cambridge.org/dictionary/english/experience) that makes the user [suitable](https://dictionary.cambridge.org/dictionary/english/suitable) for a [particular](https://dictionary.cambridge.org/dictionary/english/particular) [job](https://dictionary.cambridge.org/dictionary/english/job) or [activity](https://dictionary.cambridge.org/dictionary/english/activity).
* It is of type String.

**+ Gender:**

* To specify the Gender of the User.
* It is of type String.

**# Email:**

* Email of the User.
* It is of type String.

**# Phone\_No:**

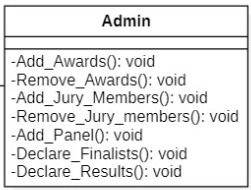
* Phone Number of the User.
* It is of type Integer.

**# Username:**

* Unique Username that each user has to login to his/her account.
* It is of type String.

**# Password:**

* Password that each user has to login to his/her account.
* It is of type String.



Admin class is Inherited from the Parent Class User.

It Inherits all properties of the User and it has its own Methods.

**- Add\_Awards():**

* This Method is used to add the available Awards to the DataBase.
* These Awards will be given to the Inventor based on their Invention.
* It doesn’t take any parameter and doesn’t return anything.
* It just asks for the name of the Award that has to added when it is called.

**- Remove\_Awards():**

* This method is used to remove the Awards from the DataBase.
* It doesn’t take any parameter and doesn’t return anything.

**- Add\_Jury\_Members():**

* This Method is used to add the details of the Jury Members to the DataBase.
* It doesn’t take any parameter and doesn’t return anything.
* It just asks for the details of the Jury that has to added when it is called.

**- Remove\_Jury\_Members():**

* This method is used to remove the details of the Jury Members from the DataBase.
* It doesn’t take any parameter and doesn’t return anything.

**- Add\_Panel():**

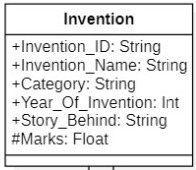
* This Method is used to add the Panel to the DataBase.
* It doesn’t take any parameter and doesn’t return anything.
* It just asks for the details of the Panel that has to added when it is called.

**- Declare\_Finalists():**

* This method is used to declare the names of the Inventors who have been selected to the Finals.

**- Declare\_Results():**

* This method is used to declare Finale results.



**+ Invention\_ID:**

* Unique ID to identify the Invention.
* It is of type String.

**+ Invention\_Name:**

* Name of the Invention.
* It is of type String.

**+ Category:**

* This attribute defines the category In which the Invention belongs to.
* It is of type String.

**+ Year\_Of\_Invention:**

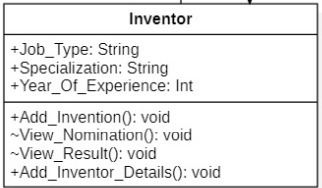
* Year in which the invention has been invented.
* It is of type Integer.

**+ Story\_Behind:**

* This defines the motive and the reason behind Inventing the particular Invention.
* It is of type String.

**# Marks:**

* Marks that has been awarded for the particular invention by the Jury.



**+ Job\_Type:**

* This attribute stores the current job of the Inventor
* This is of type String.

**+ Specialization:**

* This stores the academic specialization of the inventor.
* This is of type String.

**+ Year\_Of\_Experience**

* This attribute stores the years of experience of the inventor.
* It is of type integer(int).

**- Add\_Invention()**

* Using this method, we add the inventions of the inventor which has been nominated to the database.

**- View\_Nomination()**

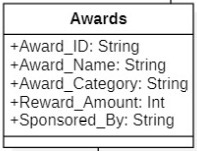
* This method displays the inventions of the inventor which have been nominated.

**- View\_Result()**

* This method displays the inventions which has been shortlisted as winners.

**- Add\_Inventor\_Details()**

* This method is used to add the details of the invention into the database.



**+ Award\_ID**

* This attribute is used to store the id of the award.
* This is of type String.

**+ Award\_Name**

* This attribute is used to store the name of the award.
* This is of type String.

**+ Award\_Category**

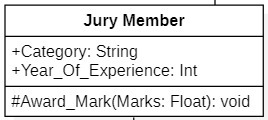
* The invention category is stored in this attribute.
* This is of type String.

**+ Reward\_Amount**

* The prize money given to the winner in this category is stored in this attribute.
* This is of type integer(int).

**+ Sponsored\_By**

* The sponsor for the prize money is stored in this attribute.
* This is of type String.



**+ Category**

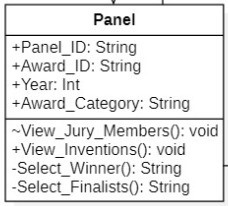
* The category in the award ceremony where the particular person is part of the jury is stored in this attribute.
* This is of type String.

**+ Year\_Of\_Experience**

* The years of experience of the Jury Member is stored in this attribute.
* It is of type Integer(int).

**# Award\_Marks**

* It stores the marks that has been awarded for the invention.
* This takes marks as the parameter which is of type Float.
* It doesn’t return anything.



**+ Panel\_ID**

* This attribute stores the ID of the particular panel.
* It is of type String.

**+ Award\_ID**

* This attribute stores the ID of the award for which the particular panel is judging.
* It is of type String.

**+ Year:**

* It stores the year of the award ceremony.
* It is of type Integer(int).

**+ Award\_Category:**

* This attribute stores the category of the award for which the panel is judging for.
* It is of type String.

**- View\_Jury\_Members()**

* This method displays the jury members present in the particular panel.

**- View\_Inventions()**

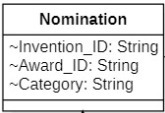
* This method displays the inventions which are part of the category the panel is judging over.

**- Select\_Winner()**

* This method is used to determine the winner in the particular category.
* The winner is returned as a String.

**- Select\_Finalists()**

* This method is used to find out the qualifying inventions in the preliminary round.
* The qualifying inventions are returned as a String.



**+ Invention\_ID**

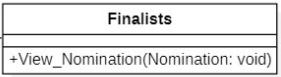
* This attribute stores the invention ID of the particular invention which has been nominated.
* It is of type String.

**+ Award\_ID**

* This attribute stores the award ID of the invention which has been nominated.
* It is of type String.

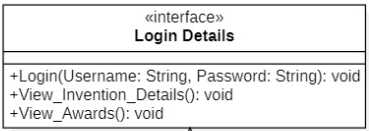
**+ Category**

* This attribute stores the category of the invention,
* It is of type String.



**+ View\_Nominations:**

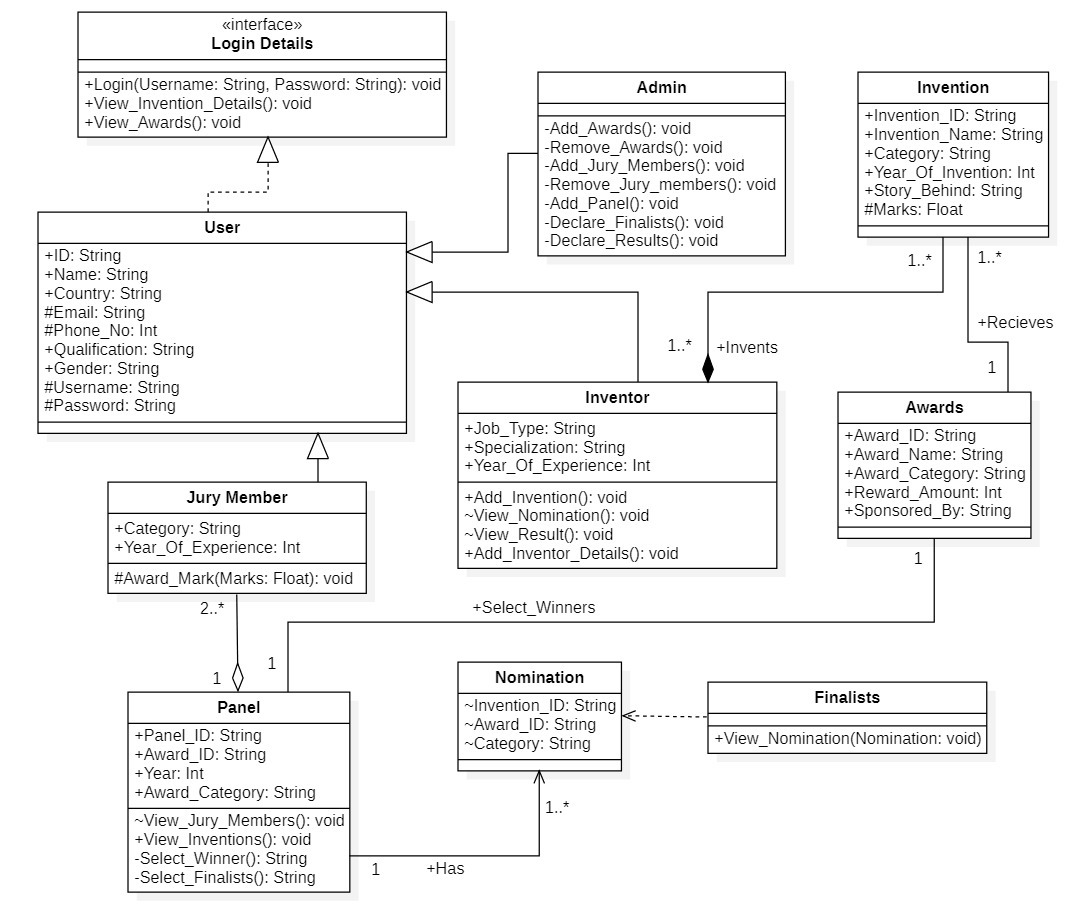
* It shows all the nominations of the finalists**.**

****

**Login:**

* This takes the Username and Password as arguments which is of type String.
* It doesn’t return anything.
* Using this credential, the user can login to the Portal/System.

|  |  |
| --- | --- |
| **ASSOCIATION** | Invention Award  Awards Panel |
| * One Award can be given to one or more Invention. * Panel selects only one winner for a particular Award. * One Award can be selected by only One Panel. |
| **REALIZATION** | Login Details User |
| * **User** implements an interface **Login Details**, thereby inheriting the abstract methods of the **Login details**. |
| **GENERALIZATION** | USER Admin    Inventor    Jury Member |
| * Admin Inherits the properties of User. * Inventor Inherits the properties of User. * Jury Member Inherits the properties of User. |
| **DEPENDENCY** | Nominations Finalists |
| * Finalists Dependent on the Nominations since an object of Nominations is being used by the Finalists. |
| **AGGREGATION** | Jury Member Panel |
| * The Jury Member can exist independently of the Panel. |
| **COMPOSITION** | Invention Inventor |
| * Invention cannot exist without the Inventor. |



**INHERITANCE**

* Here user is the parent class and it has three child class or sub class.
  + - * Admin
      * Inventor
      * Jury Members
* User have all data members and attributes that are common to all three sub classes.
* Here the basic Details are taken as the common attributes and made the sub classes to inherit from the parent class.
* ID
* Name
* Country
* Email
* Phone
* Qualification
* Gender
* Username
* Password

The common operation for admin, inventor, jury members is that all have to login to the portal before entering into the database.

* So, login operation and it's common attributes are included in the parent class.
* All those three actors can view both the Invention and Award details.

**STATIC**

* Here we have used static variable to differentiate the ADMIN and the JURY.
* Here we have declared as if the flag is 1 then the admin class will be triggered and all functionalities associated with the admin will be visible.
* If the flag is 0 then the jury class will be triggered and all functionalities associated with the jury will be visible.
* Static holds the same value in all places and thus it is helpful in interacting with different object.

**OVER RIDING**

* If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.
* Here we have used over riding concept in the operations **viewnomination()** and **viewawards()** rather than declaring them individually.
* This improves readability and helps in better implementation.

**JDBC**

* JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
* The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.
* Making a connection to a database.
* Creating SQL statements.
* Executing SQL queries in the database.
* Viewing & Modifying the resulting records.

## **Create a Connection**

* Before doing any work, we must create a connection to the database:
* **Class.forName("org.postgresql.Driver");**
* **Connection c = DriverManager.getConnection("jdbc: postgresql:DATABASELOCATION");**
* Replace **DATABASELOCATION** with the (preferably relative) path to your database file.

## **Close a Connection**

After finishing working with the database, we must close its connection:

c.close();

## **Using Statements**

* Statements can be used to make updates, like this INSERT:
* Statement stmt = c.createStatement();
* String sql = "INSERT INTO employees (name, salary, address) "+ "VALUES ('" + name + "', " + salary + ", '" + address + "')";
* stmt.executeUpdate(sql);

**Or queries, like this SELECT:**

* Statement stmt = c.createStatement();
* String sql = "SELECT \* FROM employees";
* ResultSet rs = stmt.executeQuery(sql);
* When making queries, the results are returned as a Result Set.
* We must always remember to close the Statement:
* stmt.close();

## **Prepared Statements**

* Statements **are not recommended** for queries, and they can't be used at all for queries that include anything other than INTEGERs, REALs or TEXTs.
* In this situation, and when we're concerned about security, we use **Prepared Statements**.
* Prepared Statements can be used to make updates, like this INSERT:
* String sql = "INSERT INTO employees (name, phone, salary, dob, photo) "+ "VALUES (?,?,?,?,?)";

PreparedStatement prep = c.prepareStatement(sql);

prep.setString(1, "Bob");

prep.setInt(2, 666666666);

prep.setDouble(3, 35000.00);

prep.setDate(4, anSqlDateObject);

prep.setBytes(5, aBytesArray);

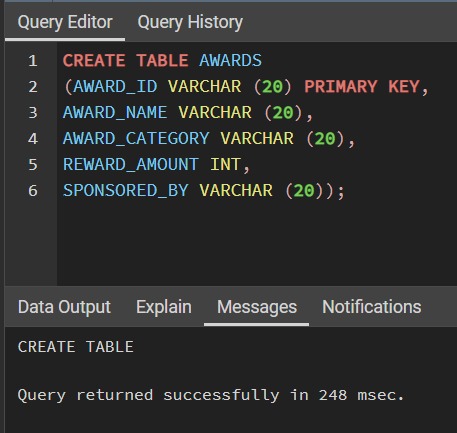
prep.executeUpdate();

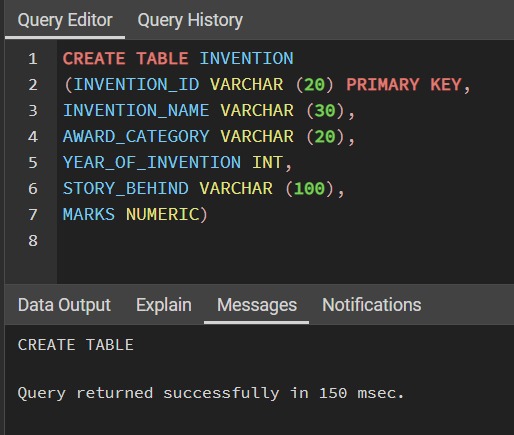
**Or queries, like this SELECT:**

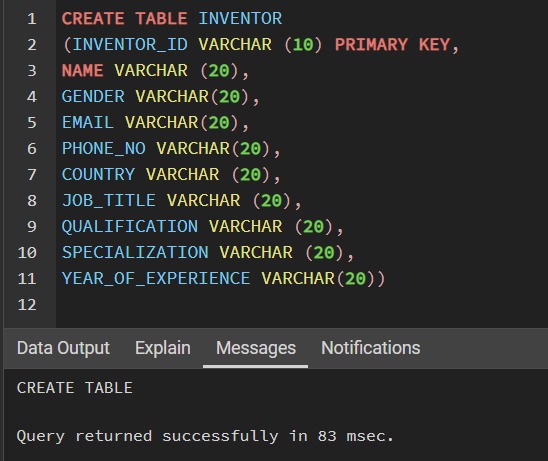
* String sql = "SELECT \* FROM employees WHERE name LIKE ?";
* PreparedStatement prep = c.prepareStatement(sql);
* prep.setString(1, "%XYZ%");
* ResultSet rs = prep.executeQuery();
* Again, when making queries, the results are returned as a Result Set.
* As always, rememeber to close the Prepared Statement:
* prep.close();

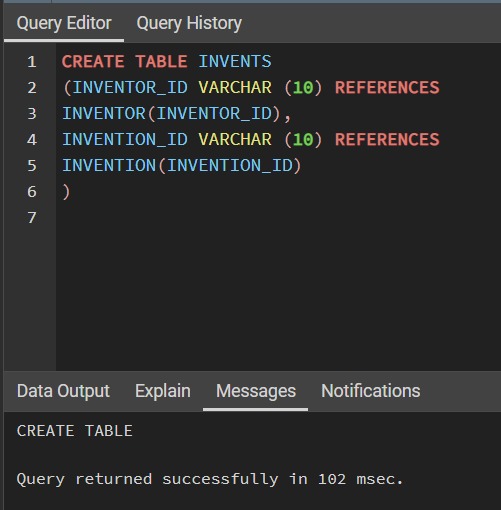
## **Processing Result Sets**

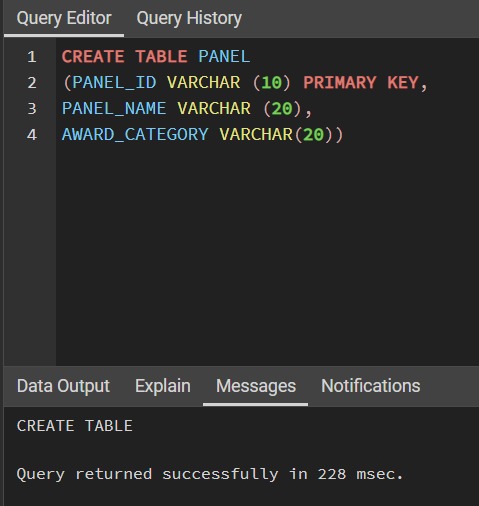
* When making queries, either with Statements or Prepared Statements, the results are returned as a Result Set.
* We can iterate over a Result Set and access its contents:
* Again, we mustn't forget to close the Result Set:
* rs.close();

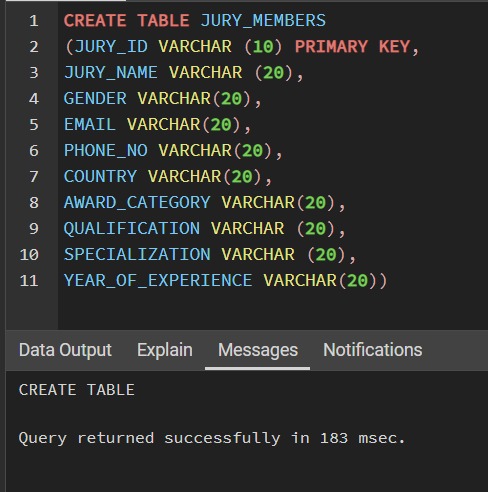


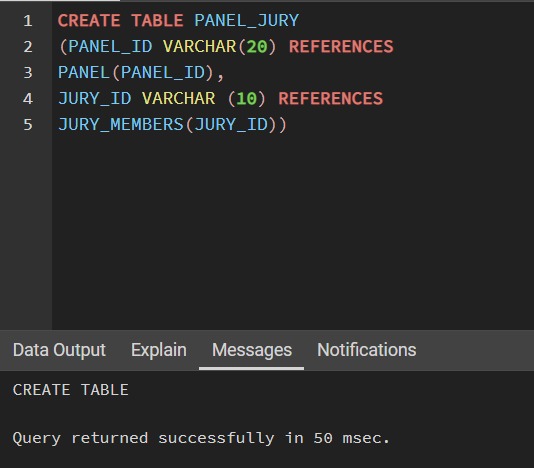


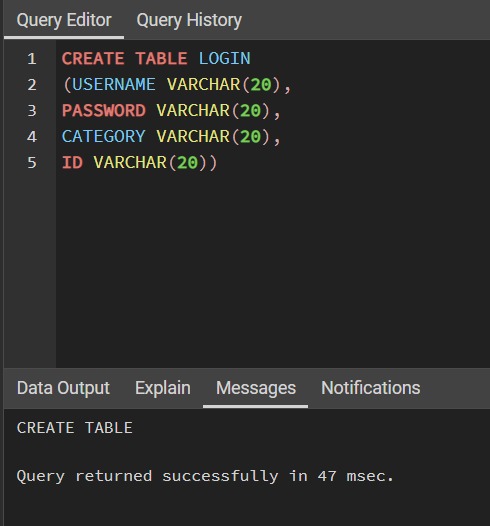


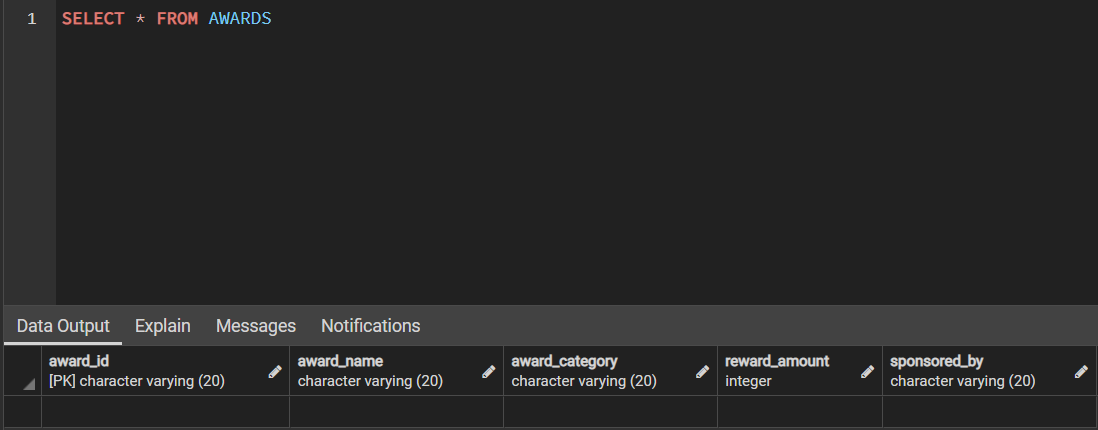


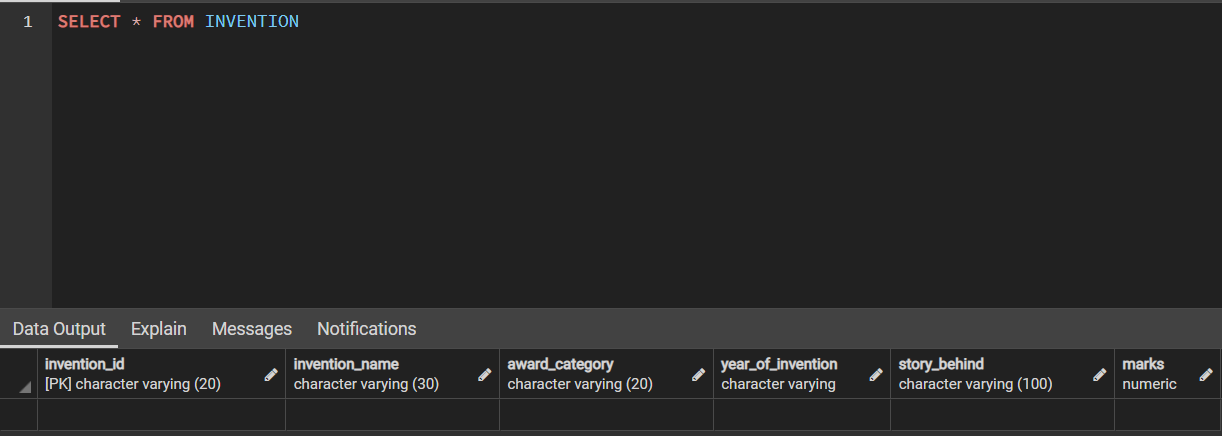




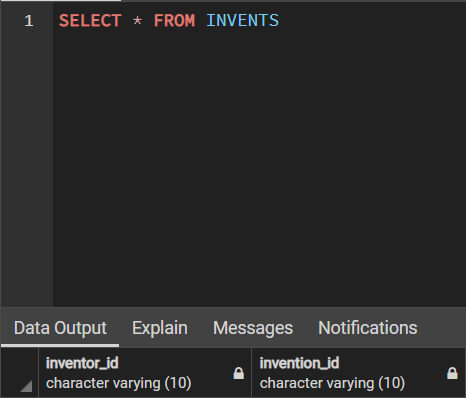




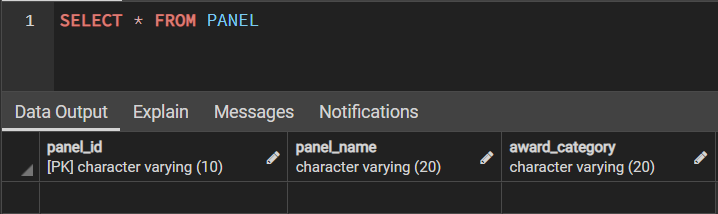


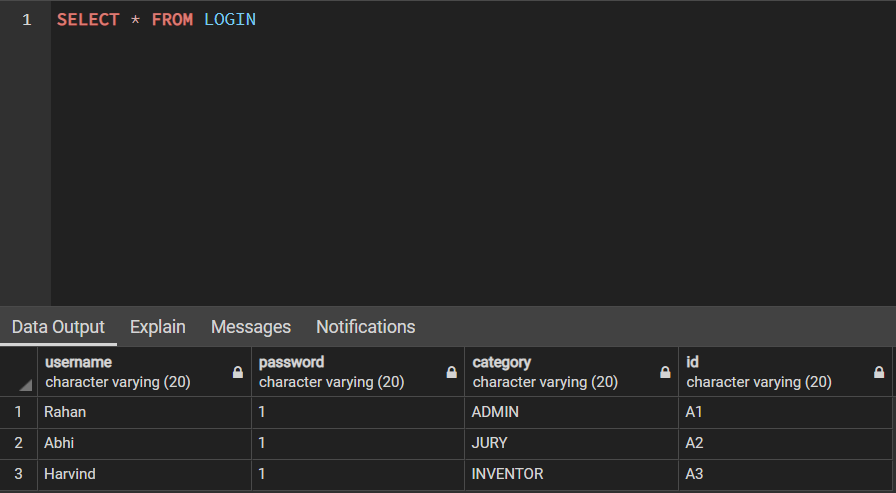


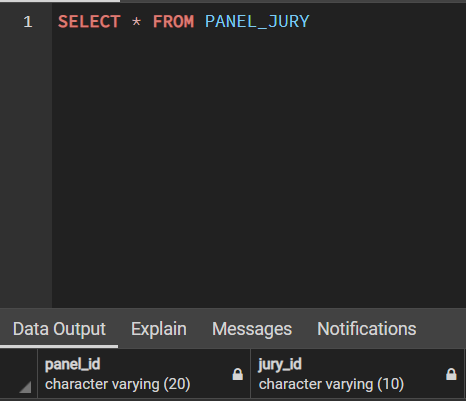












**SWING**

* A component is an independent visual control and Java Swing Framework contains a large set of these components which provide rich functionalities and allow high level of customization.
* They all are derived from JComponent class.
* All these components are lightweight components.
* This class provides some common functionality like pluggable look and feel, support for accessibility, drag and drop, layout, etc.
* A container holds a group of components.
* It provides a space where a component can be managed and displayed. Containers are of two types.

1. **Top level Containers**
   * It inherits Component and Container of AWT.
   * It cannot be contained within other containers.
   * Heavyweight.
   * **Example:** JFrame, JDialog.
2. **Lightweight Containers**
   * It inherits JComponent class.
   * It is a general purpose container.
   * It can be used to organize related components together.
   * **Example:** JPanel

**JFrame:**

* new JFrame(String title)make a new frame with optional title
* setVisible(true)make a frame appear on the screen
* add(Component comp) place the given component or container inside the frame
* setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE) make it so that the program exits when the frame is closed
* setSize(int width, int height) gives the frame a fixed size in pixels

**JDialog**

* new JDialog(Frame parent, String title, boolean modal)make a new JDialog with given parent and title. If modalis set, the parent will be locked until the dialog is closed
* JOptionPane.showMessageDialog(parent, message) static method to pop up a dialog with just a message and OK button
* JOptionPane.showConfirmDialog(parent, message) static method to pop up a dialog with a message and Yes and No buttons
* JOptionPane.showInputDialog(parent, message) static method to pop a dialog with a message and a textfield for entering information

**JLabel**

* new JLabel(String text)creates a new label with the given text
* getText()returns the text showing on the label
* setText()sets label’s text JButton
* new JButton(String text) creates a new button with text
* getText()returns the text showing on the button
* setText(String text)sets button’stext
* new JTextArea(int lines, int columns) create a new text area with preferred size for the given number of lines and columns JTextField
* new JTextField(int columns)create a new field, the given number of columns wide.

**Event:**

* Event Object that contains detailed information about the event
* getSource() returns a reference to the object to which the event occurred

**ActionEvent:**

* getActionCommand() returns the command string associated with this action.
* getWhen()returns a timestamp of when this event occurred.

**Listener:**

* Listener Object that can be attached to a component to listen for events.
* Contains a method that is automatically called when an event occurs.

**Swing JButton**

* JButton class provides functionality of a button.
* It is used to create button component.

### JTextField

* JTextField is used for taking input of single line of text. It is most widely used text component.

### JCheckBox

* The JcheckBox class is used to create checkbox in swing framework.

### JRadioButton

* Radio button is a group of related button in which only one can be selected.
* JRadioButton class is used to create a radio button in Frames.

### JComboBox

* Combo box is a combination of text fields and drop-down list.
* JComboBox component is used to create a combo box in Swing.

### JLabel

* In Java, Swingtoolkit contains a JLabel Class.
* It is under package javax.swing.JLabel class.
* It is used for placing text in a box.
* Only Single line text is allowed and the text cannot be changed directly.

### JTextArea

* In Java, Swing toolkit contains a JTextArea Class.
* It is under package javax.swing.JTextArea class.
* It is used for displaying multiple-line text.

### JPasswordField

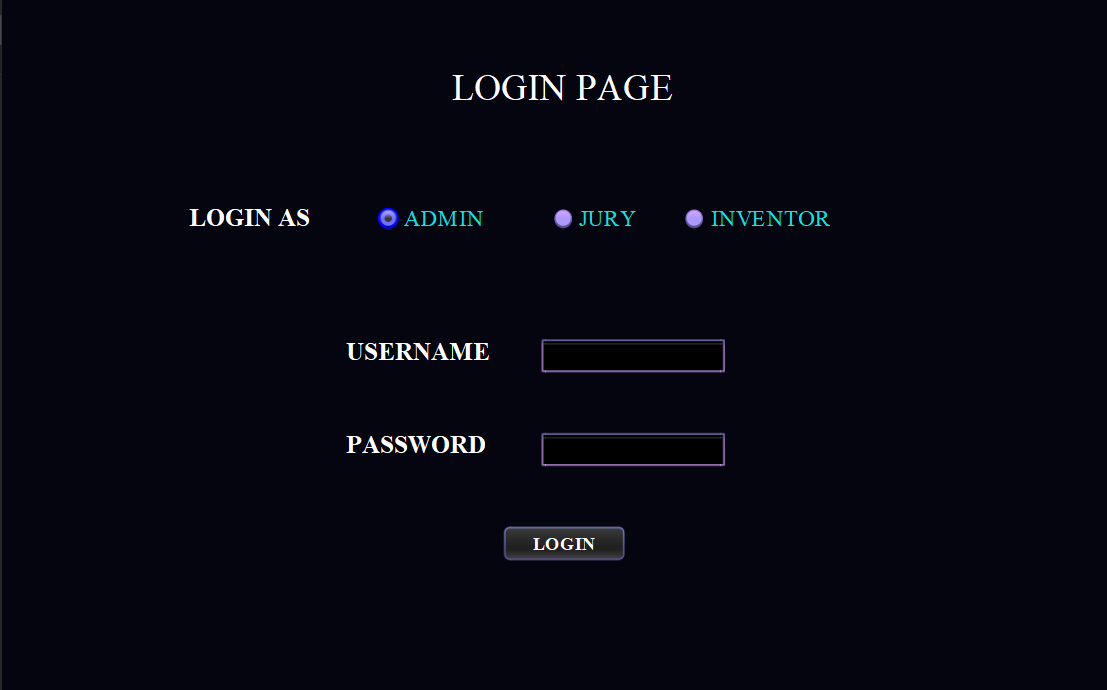
* In Java, Swing toolkit contains a JPasswordField Class.
* It is under package javax.swing.JPasswordField class.
* It is specifically used for password and it can be edited.

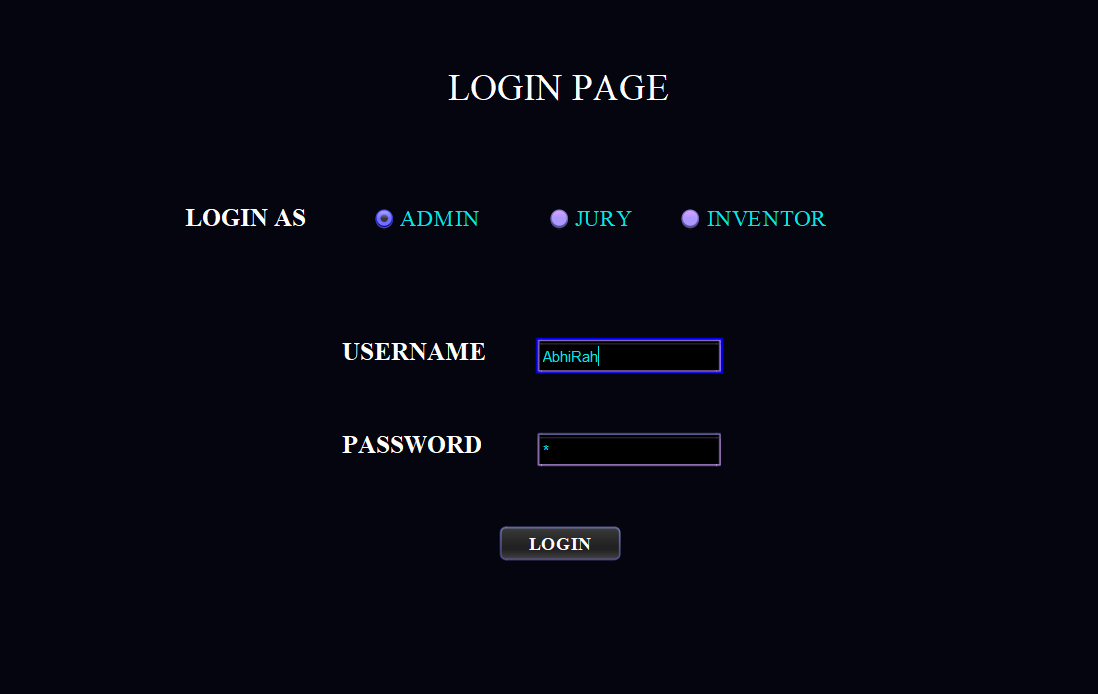
### JTable

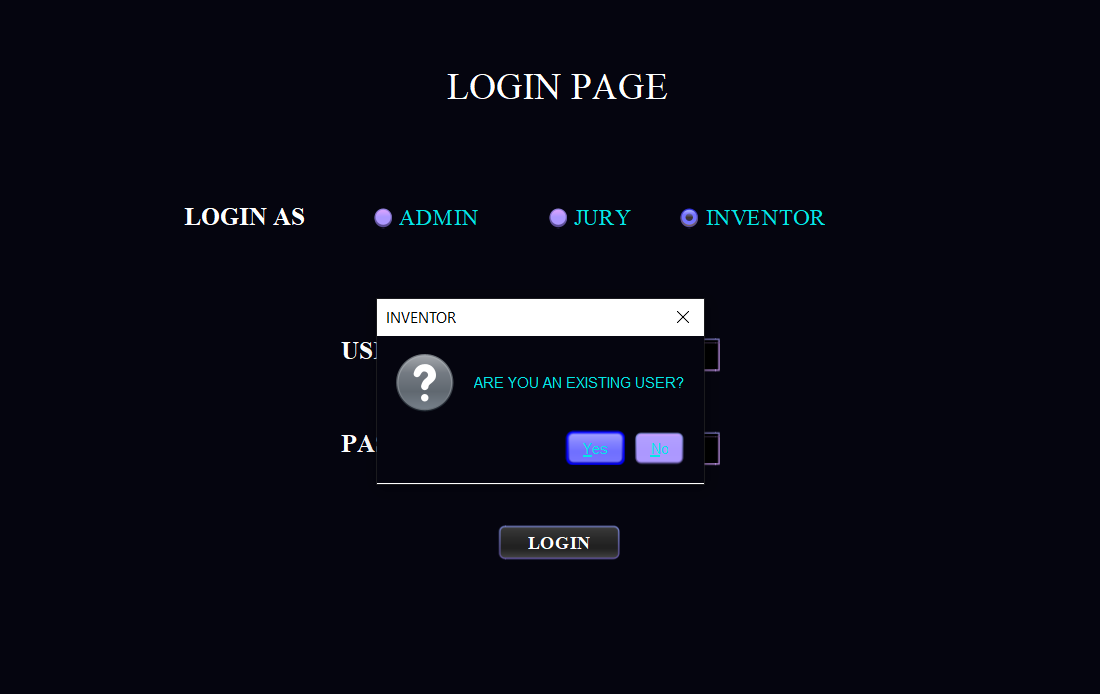
* In Java, Swing toolkit contains a JTable Class.
* It is under package javax.swing.JTable class.
* It used to draw a table to display data.

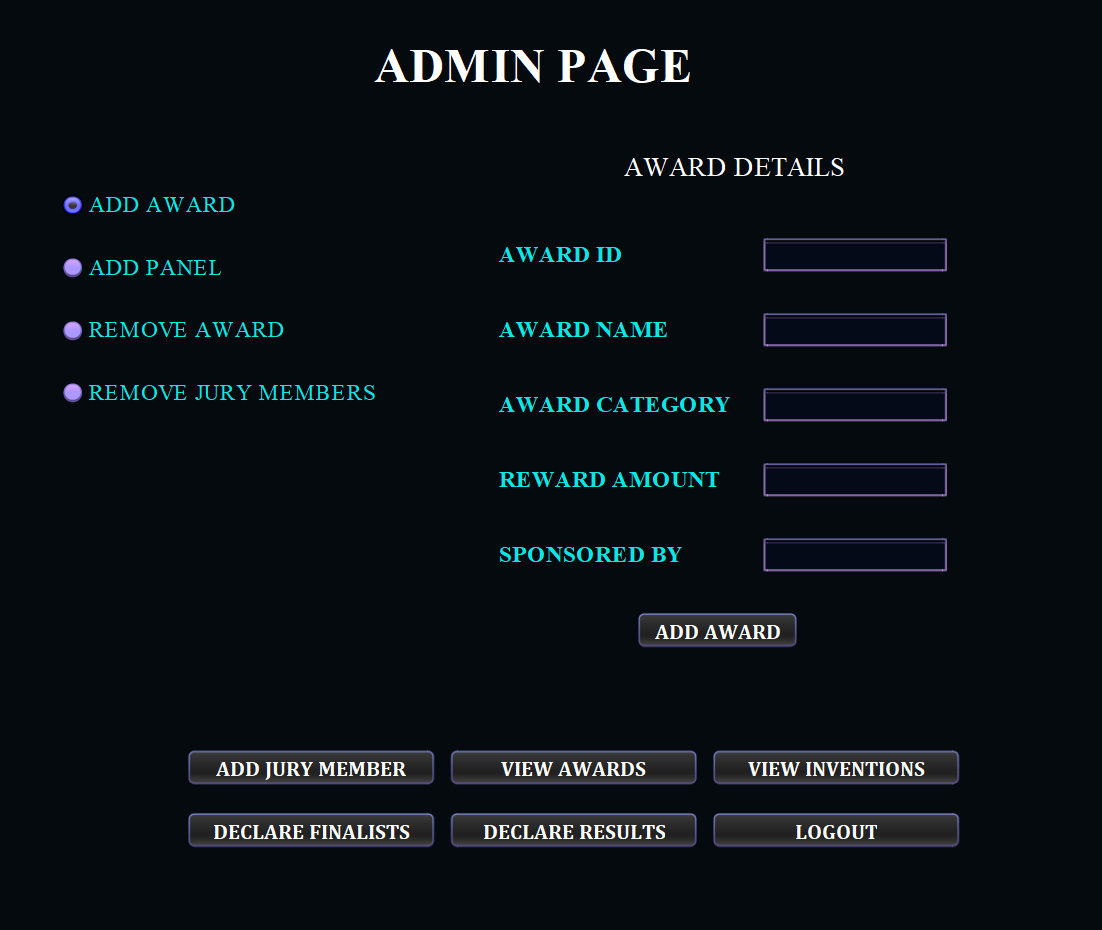
### JList

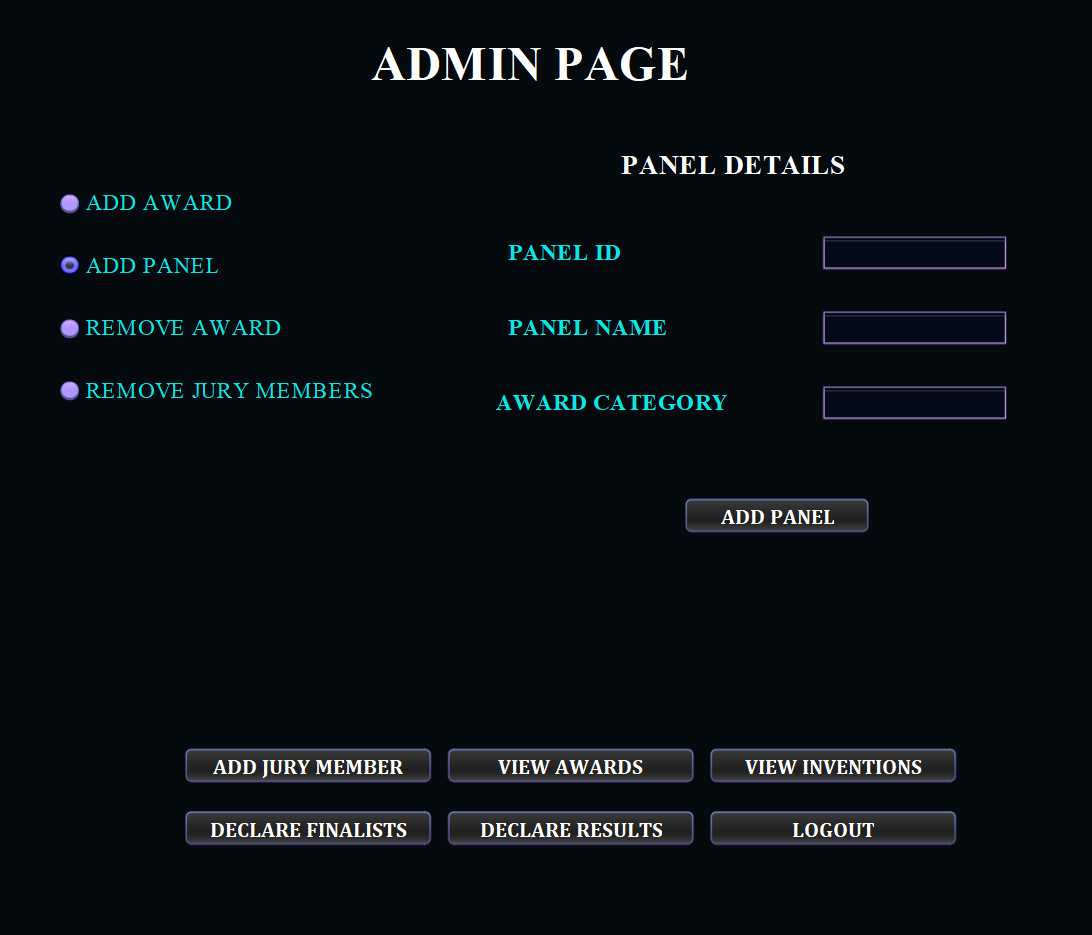
* In Java, Swing toolkit contains a JList Class.
* It is under package javax.swing.JList class.
* It is used to represent a list of items together.
* One or more than one item can be selected from the list.

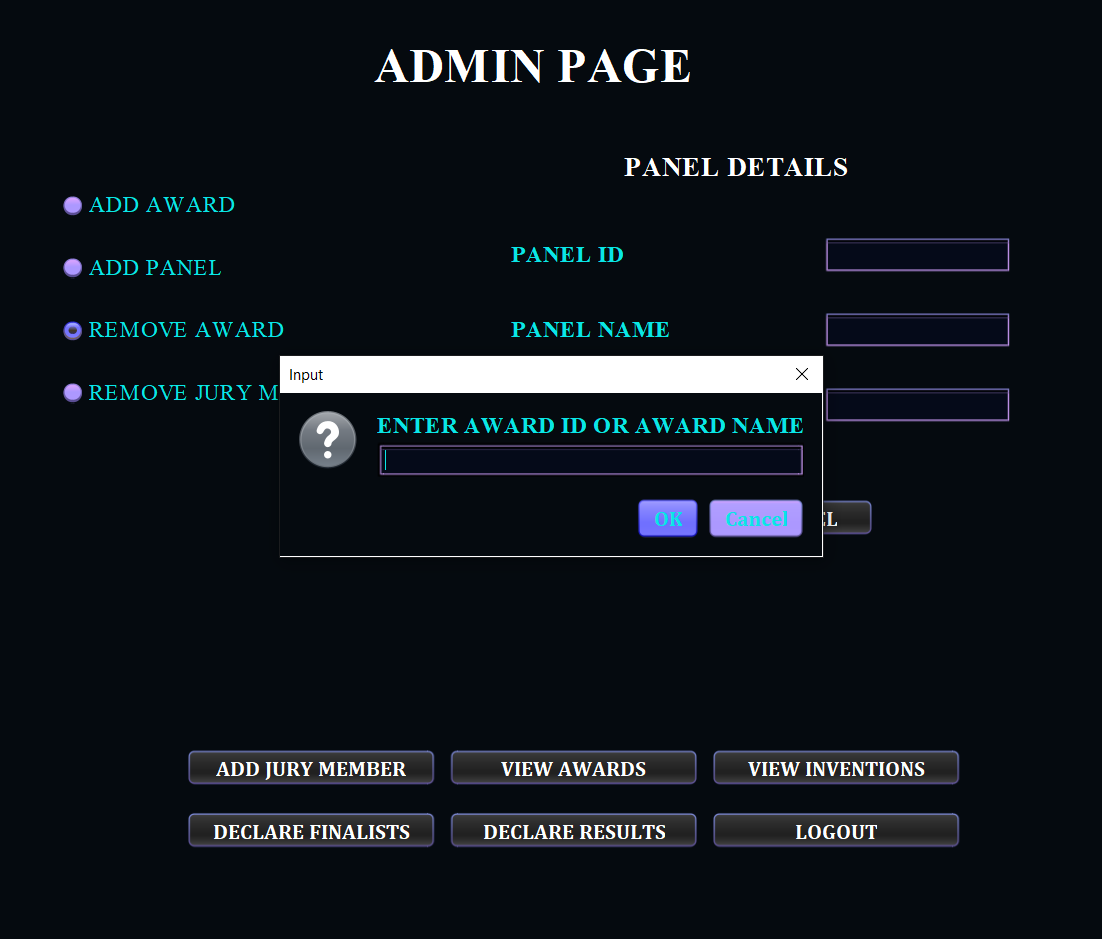
****

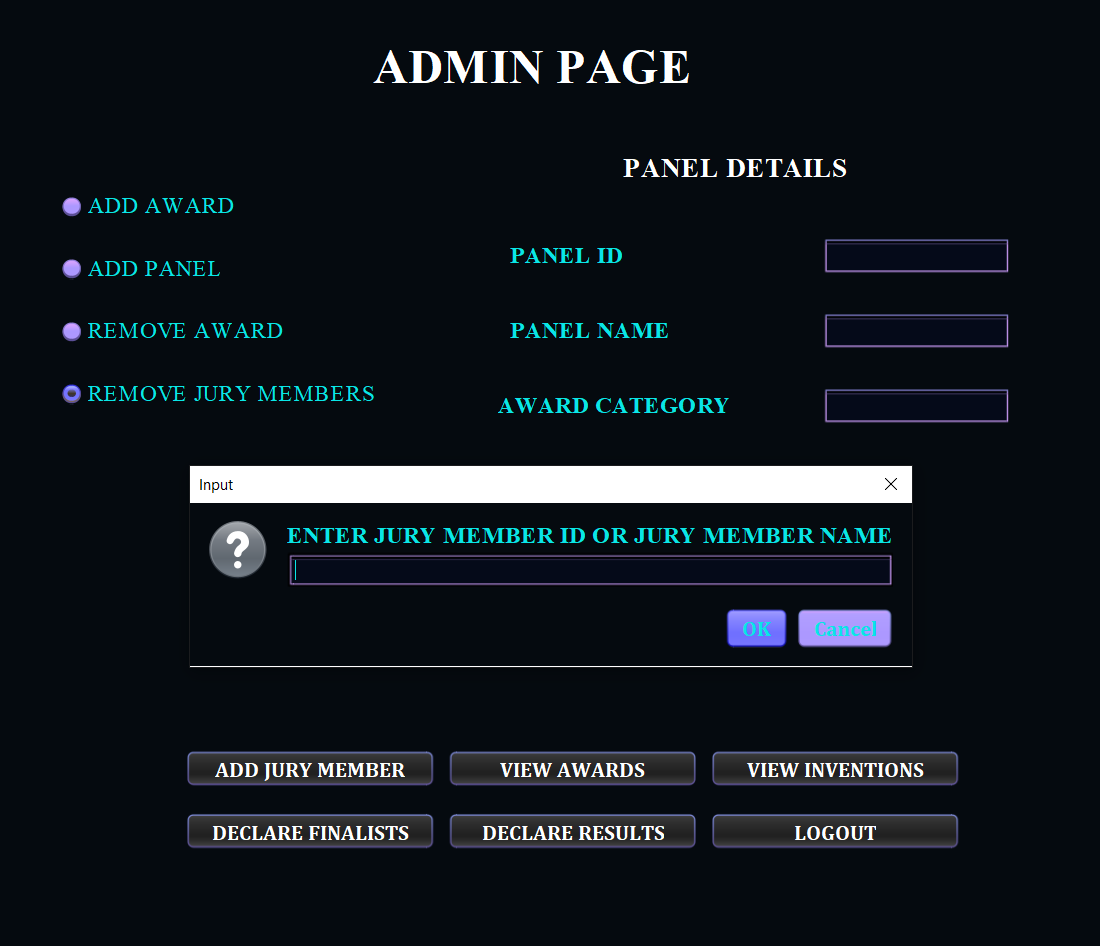
****

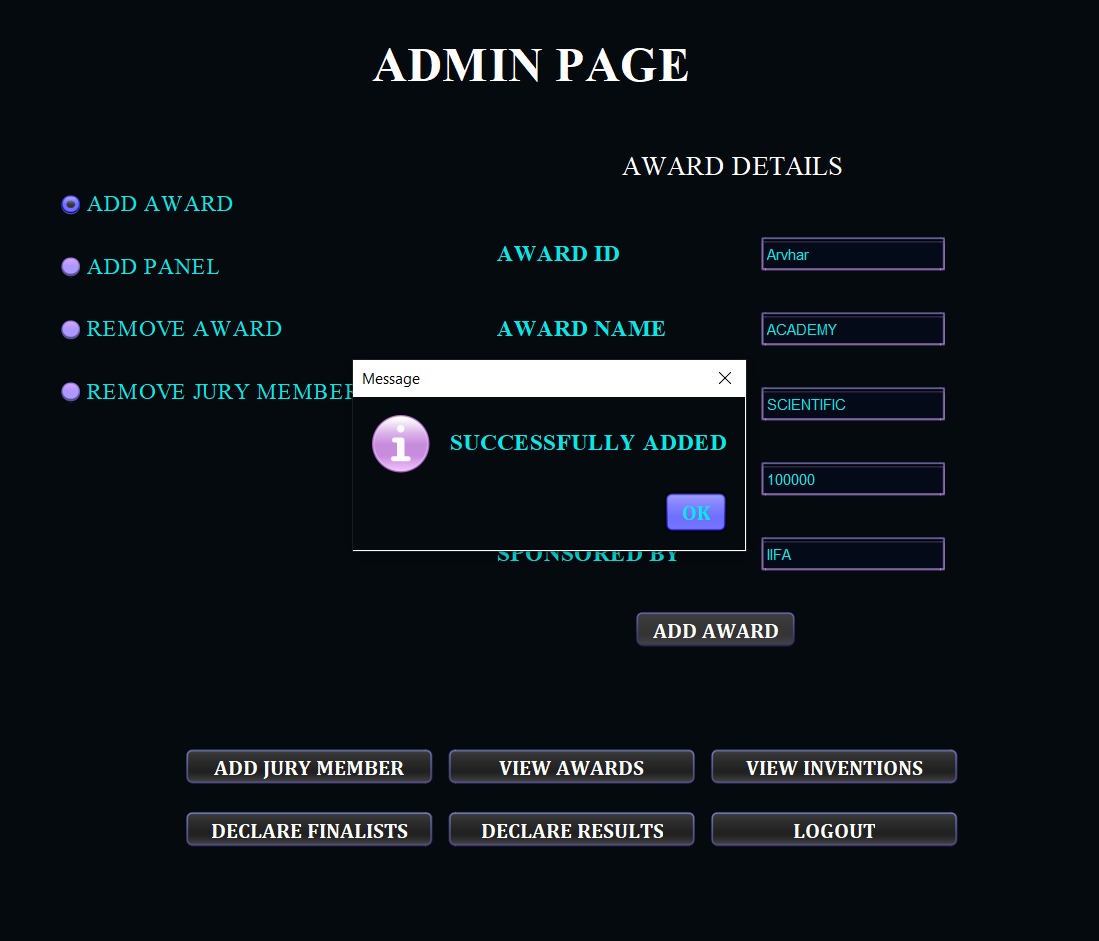
****

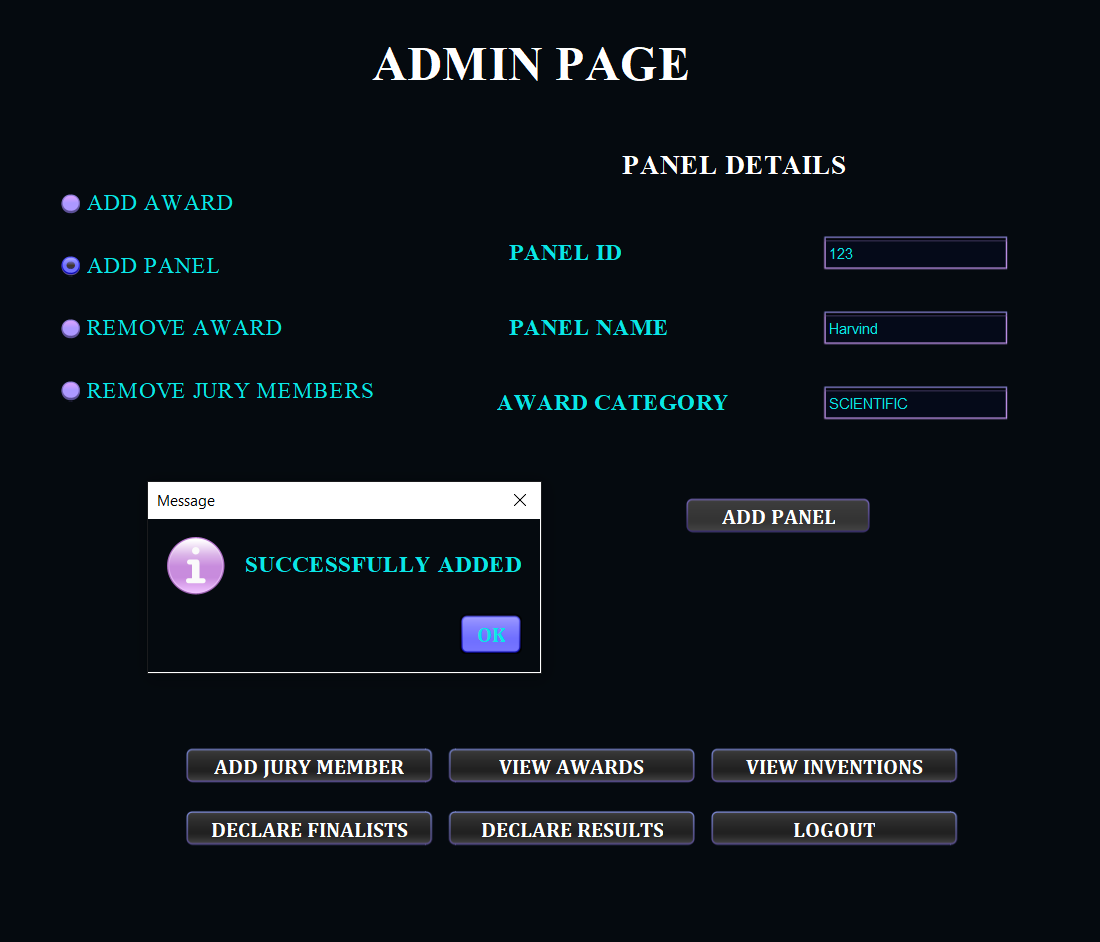
****

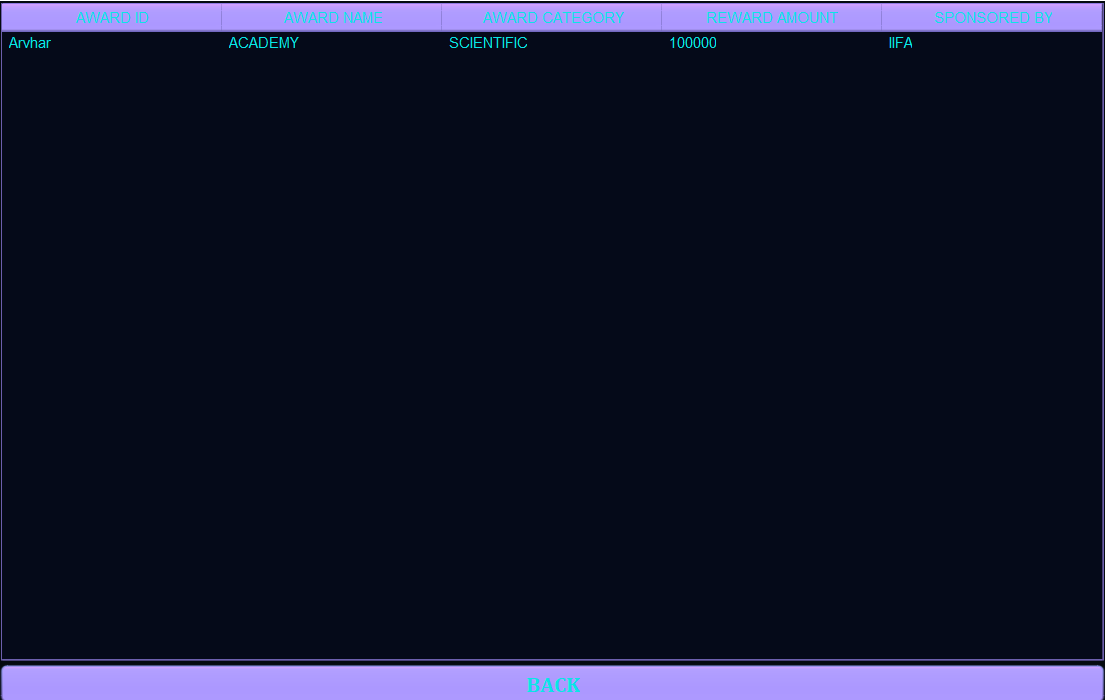
****

****

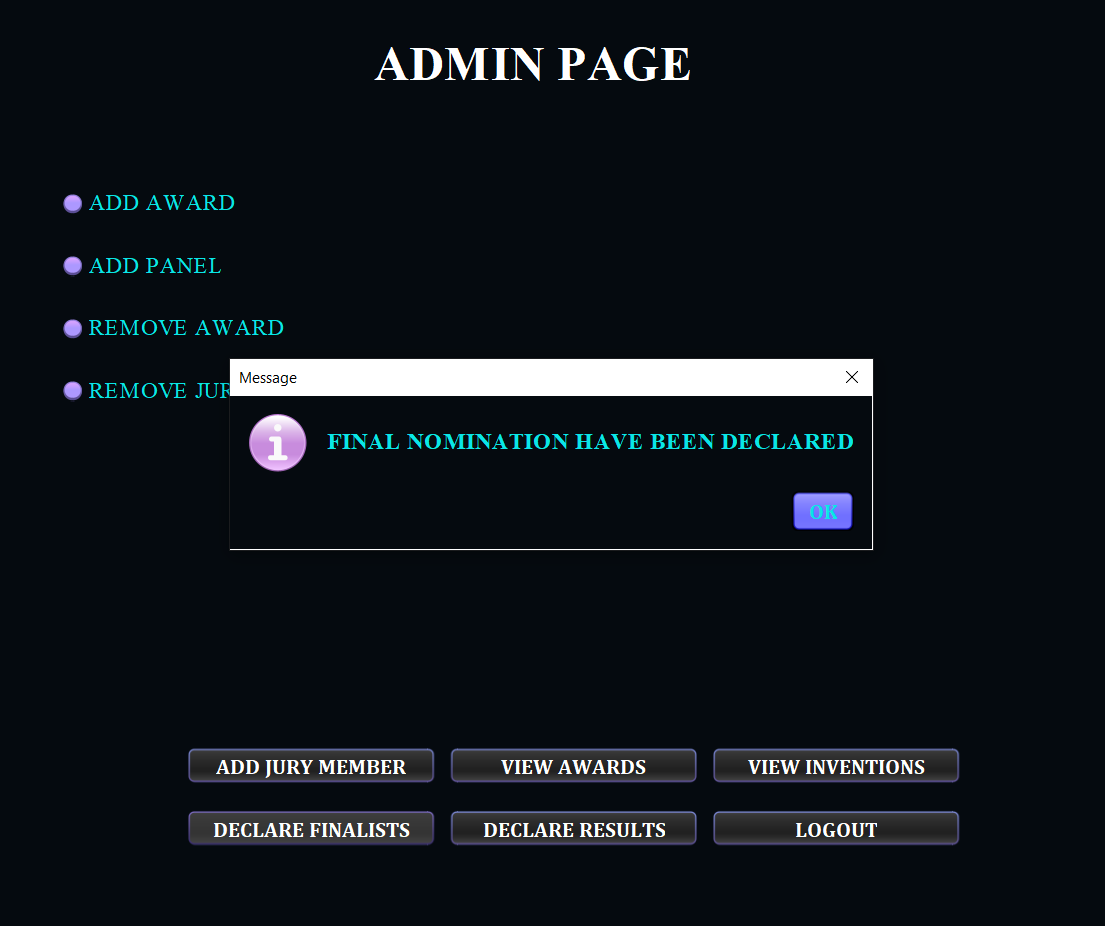
****

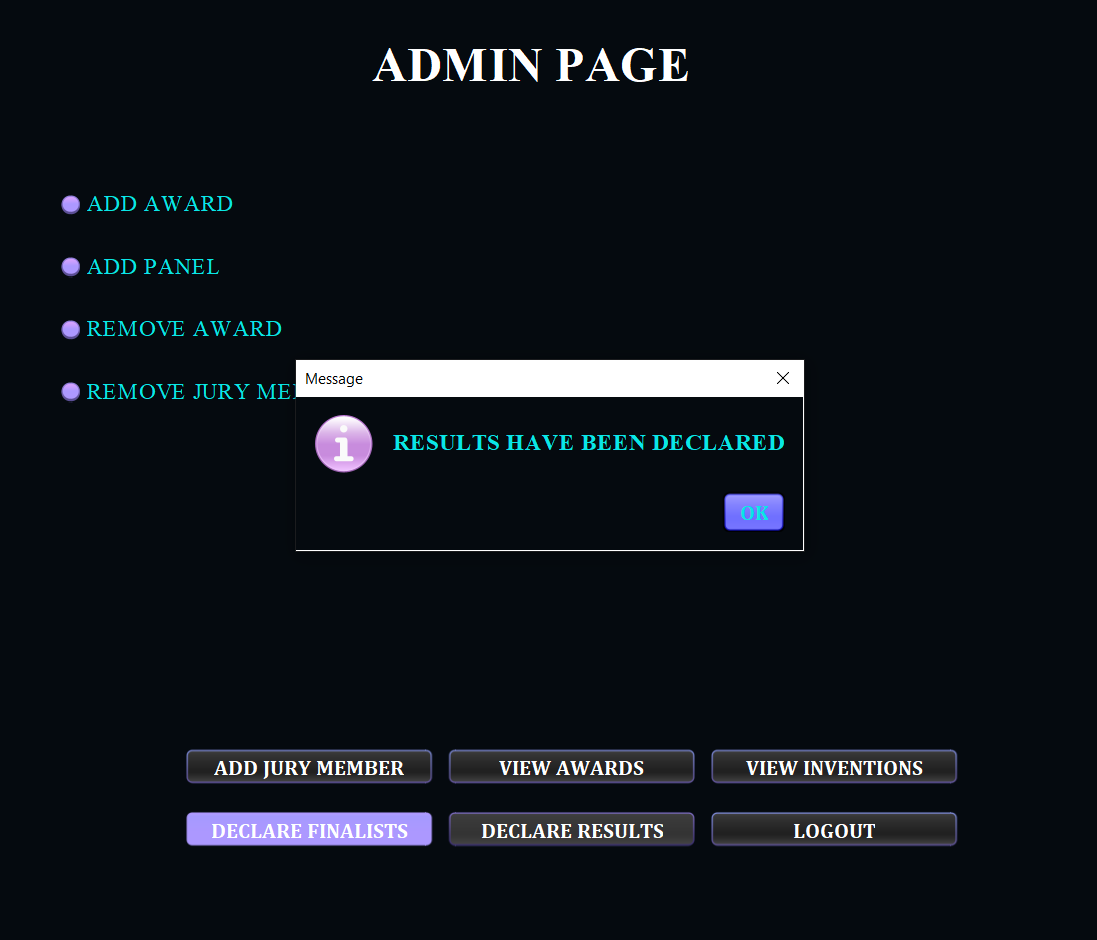
****

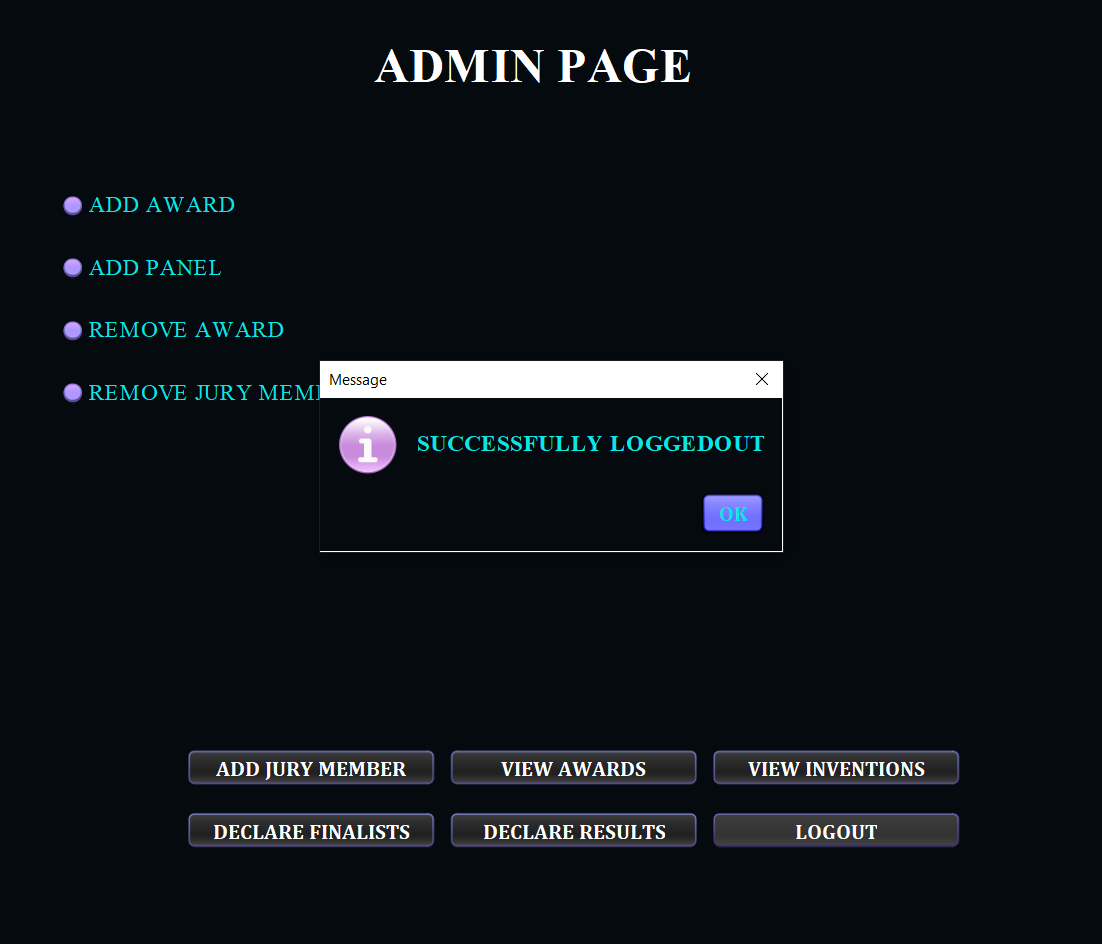
****

****

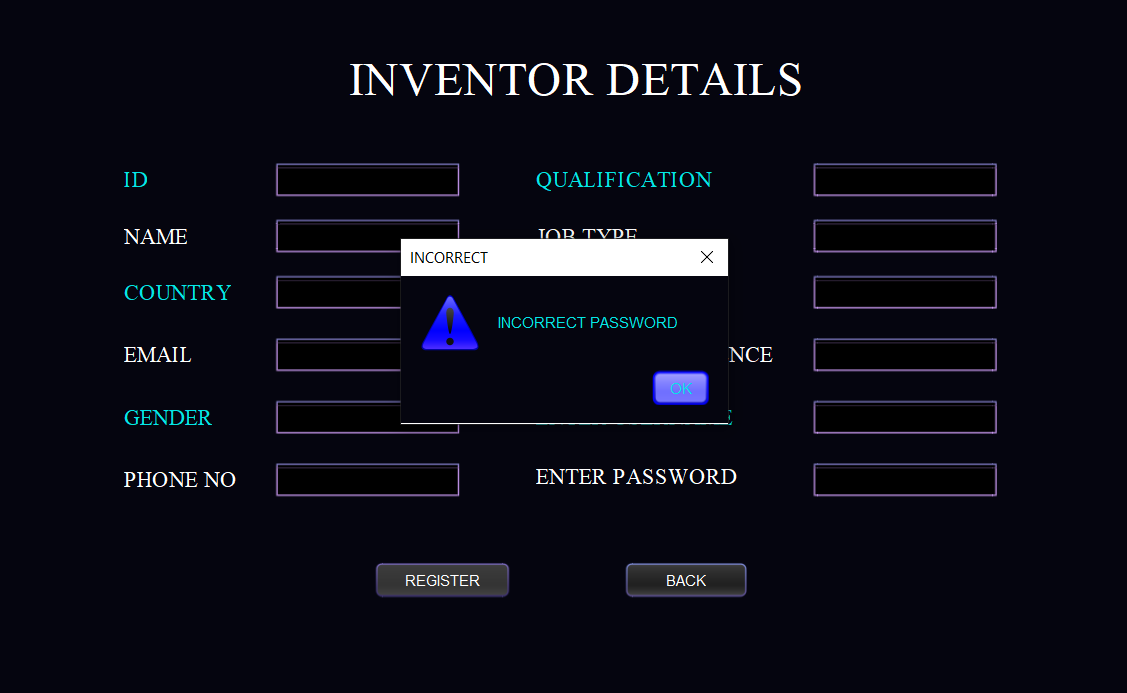
****

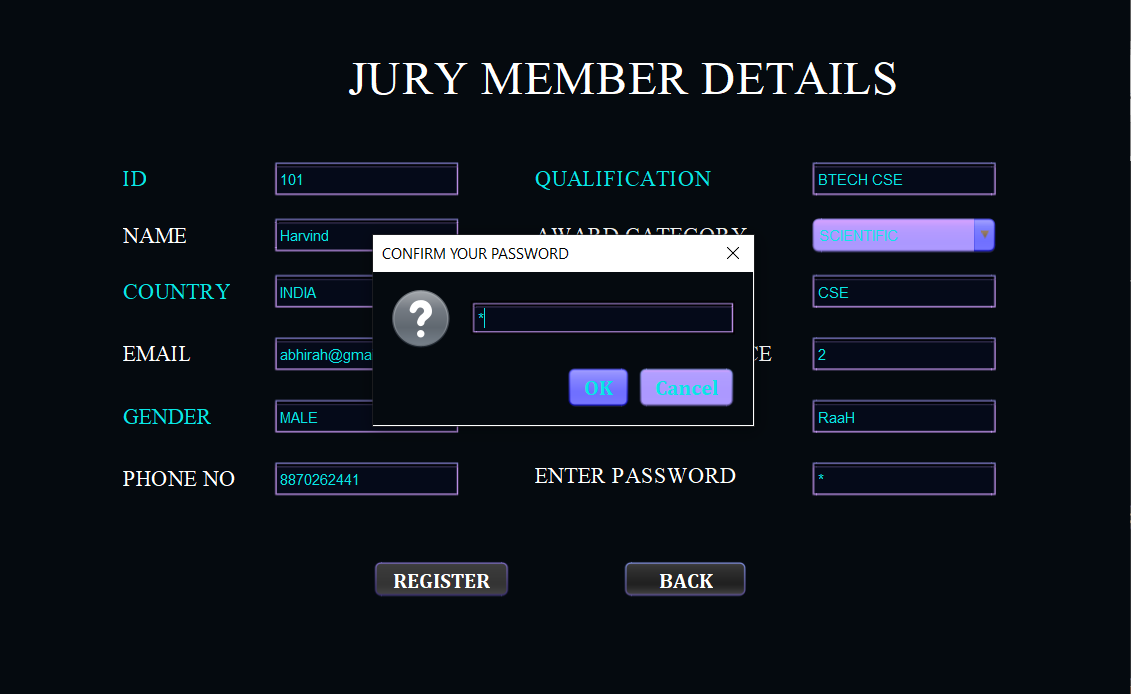
****

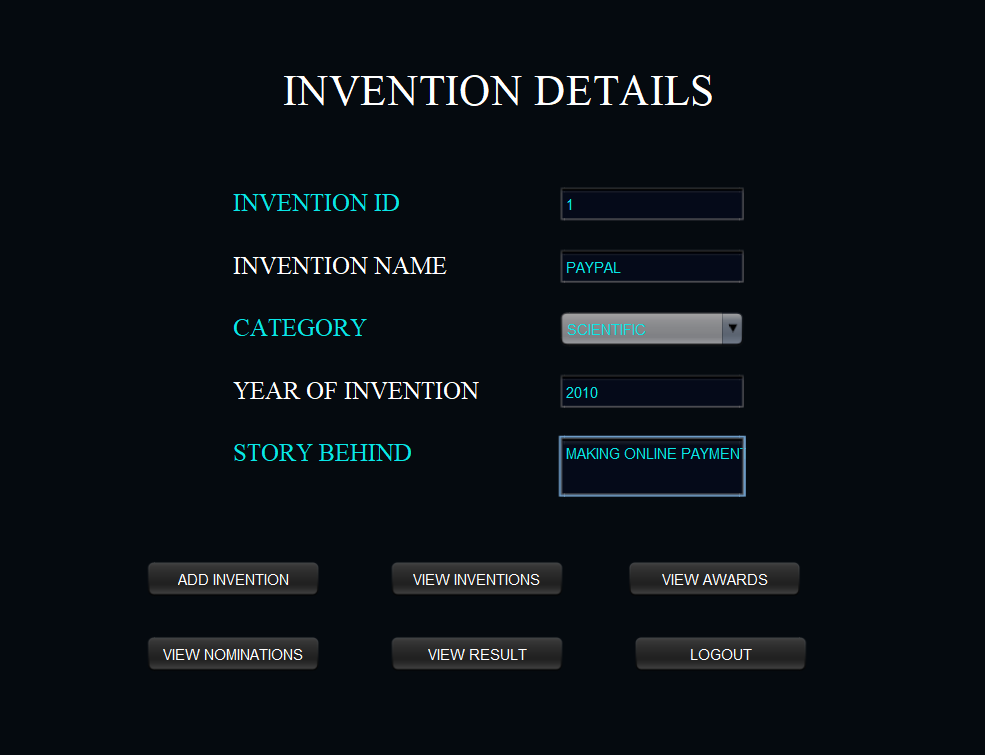
****

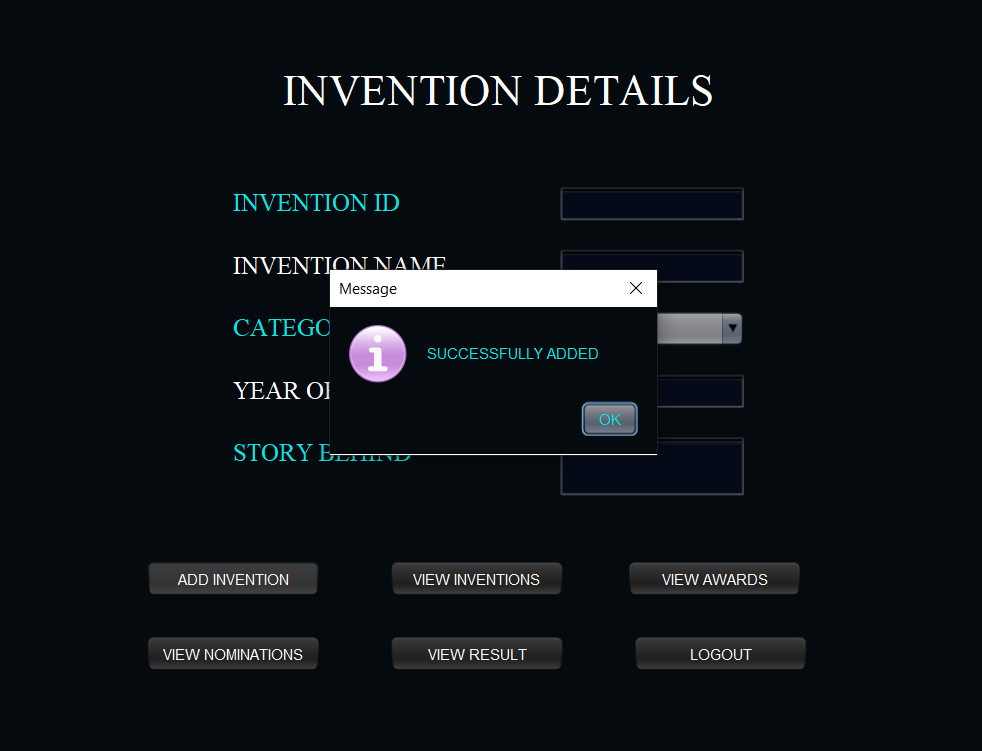
****

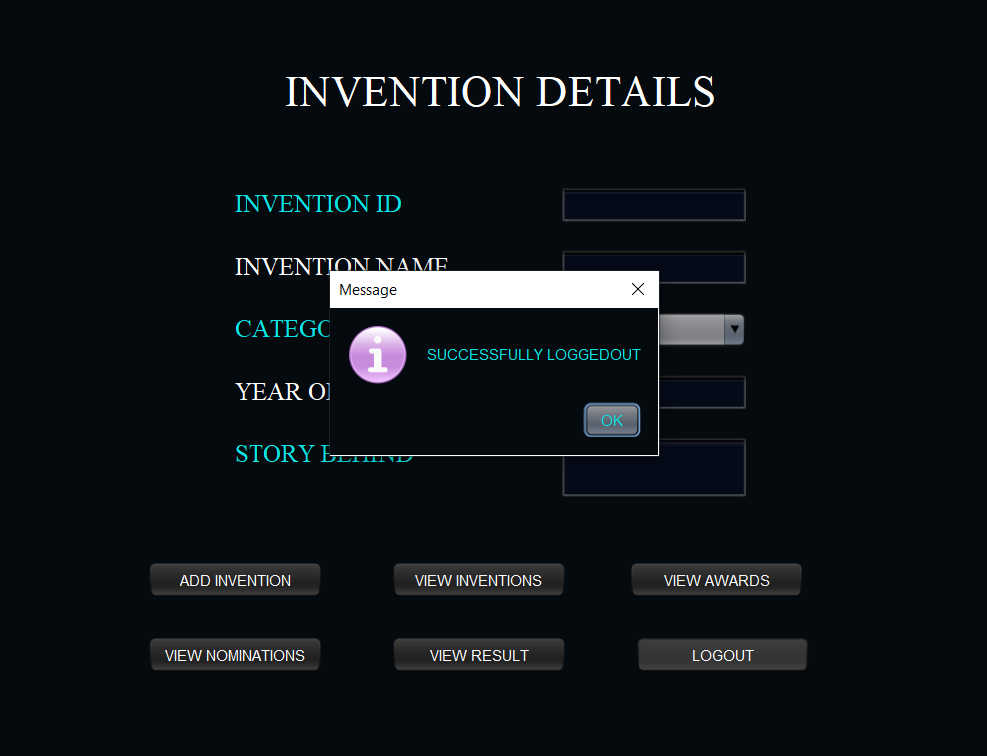


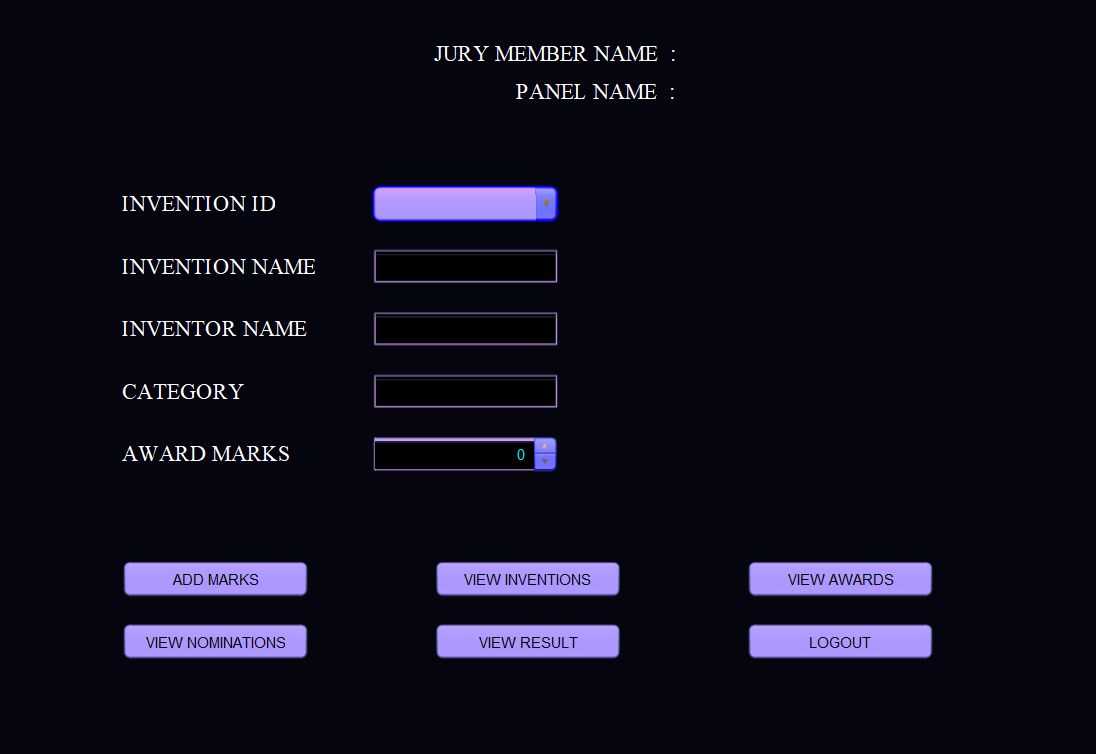




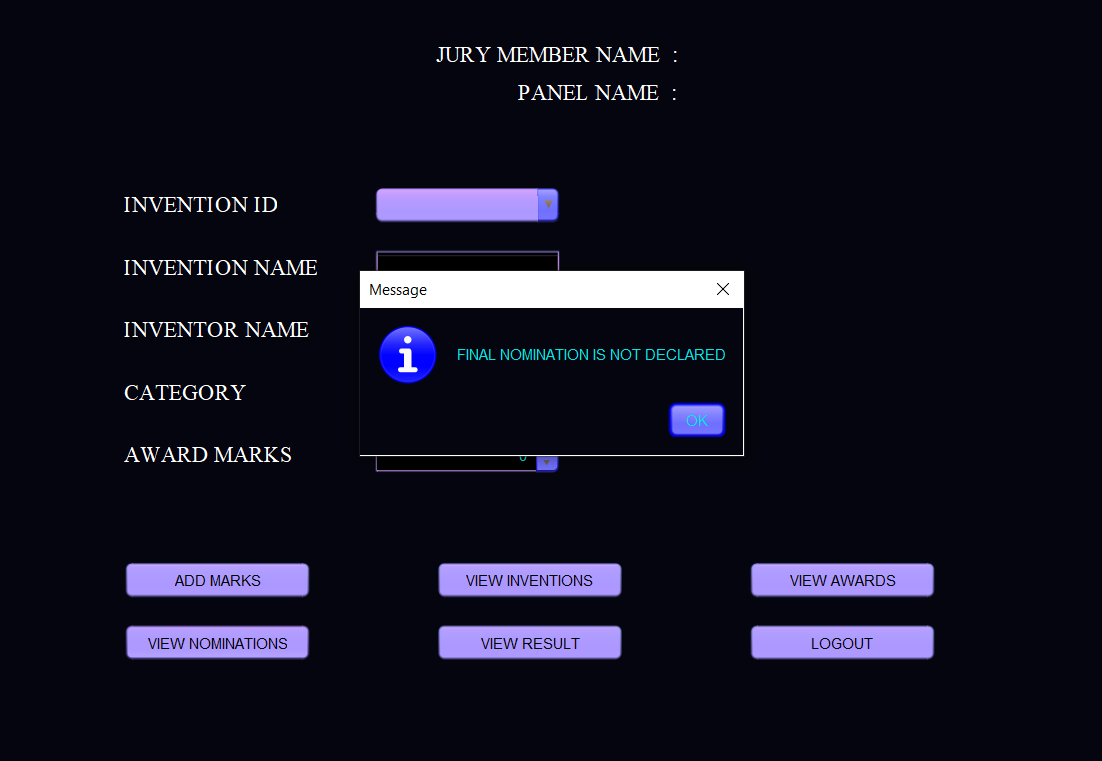


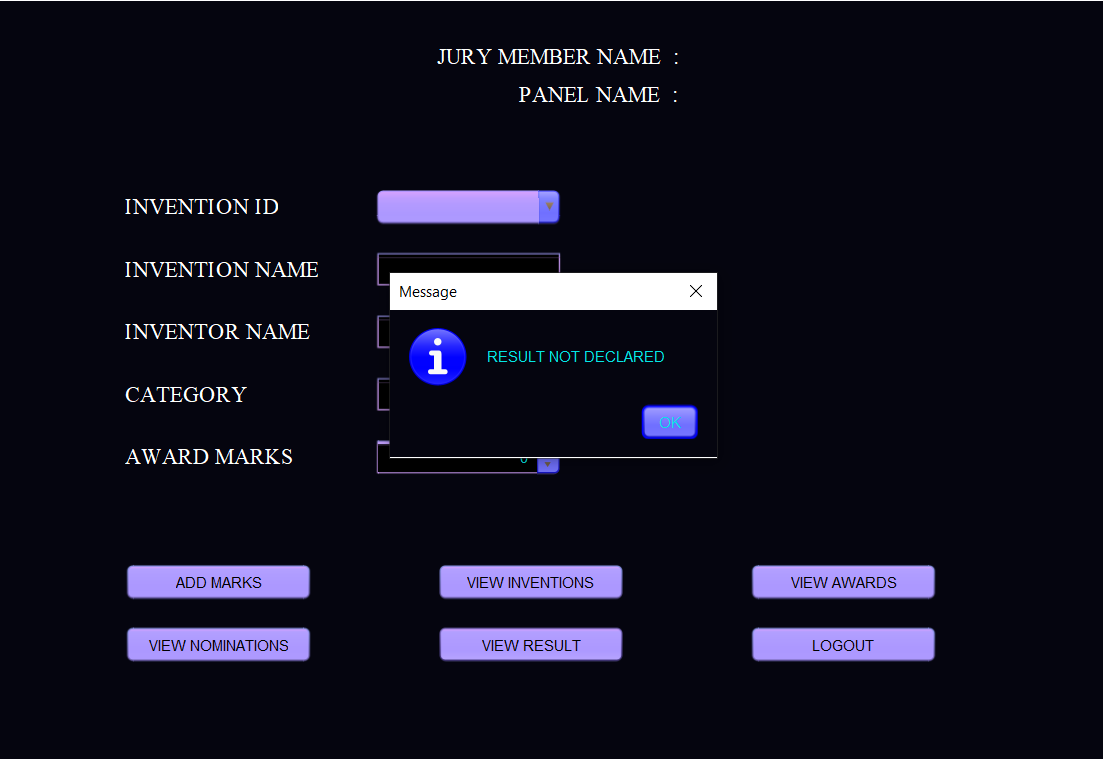


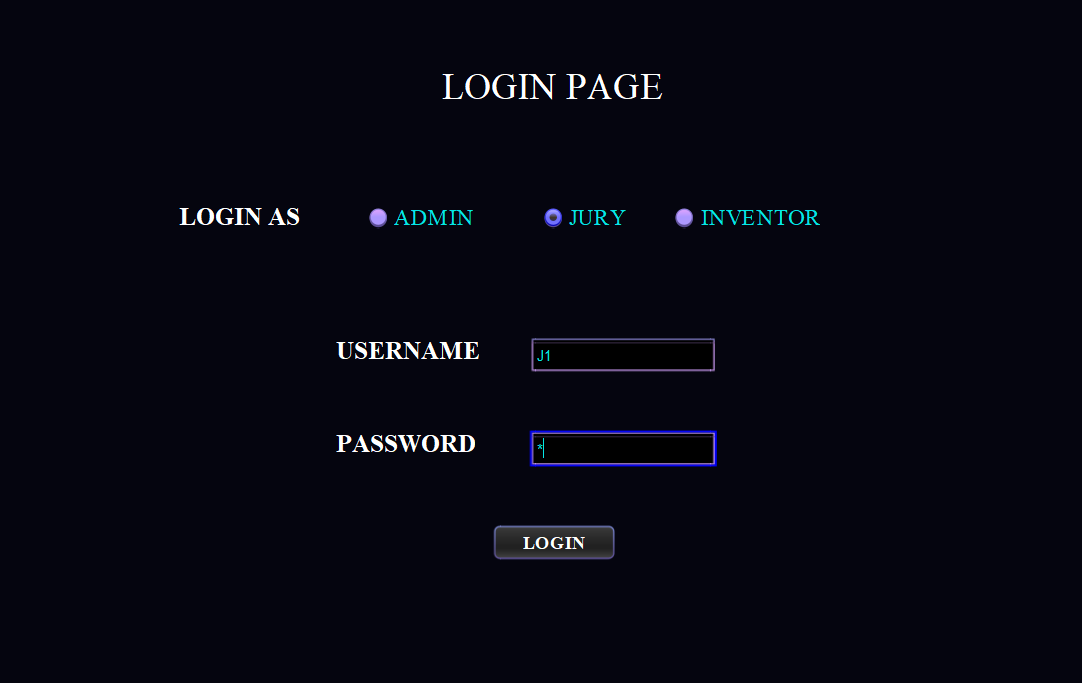


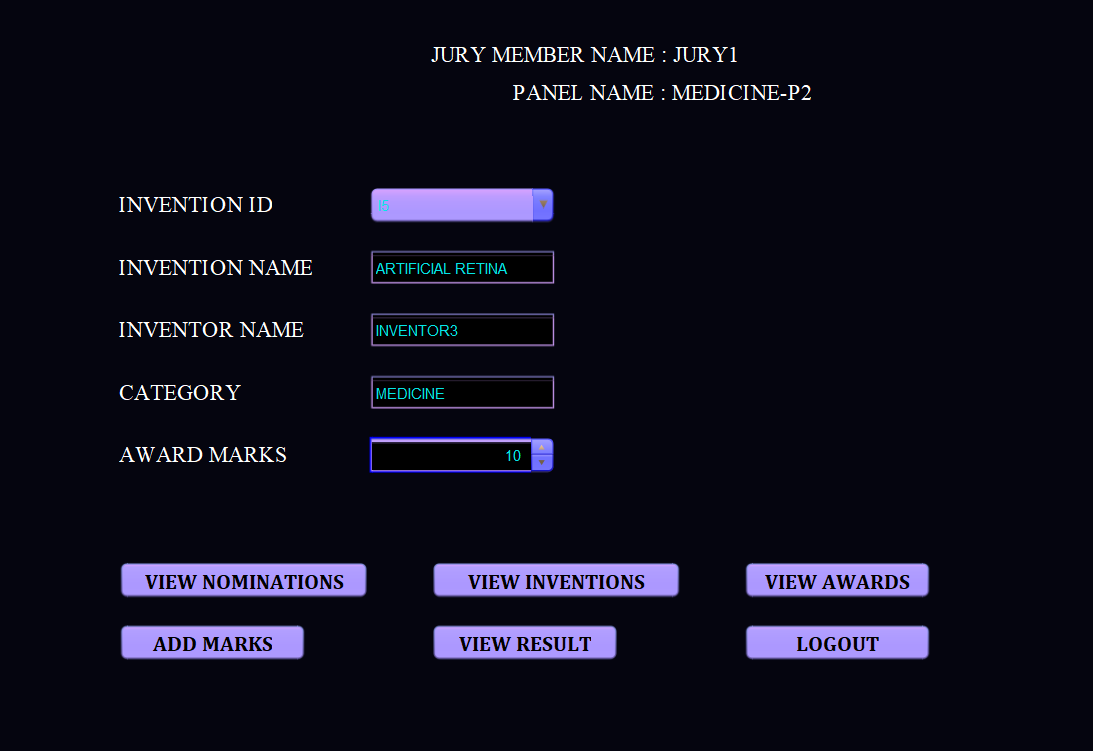


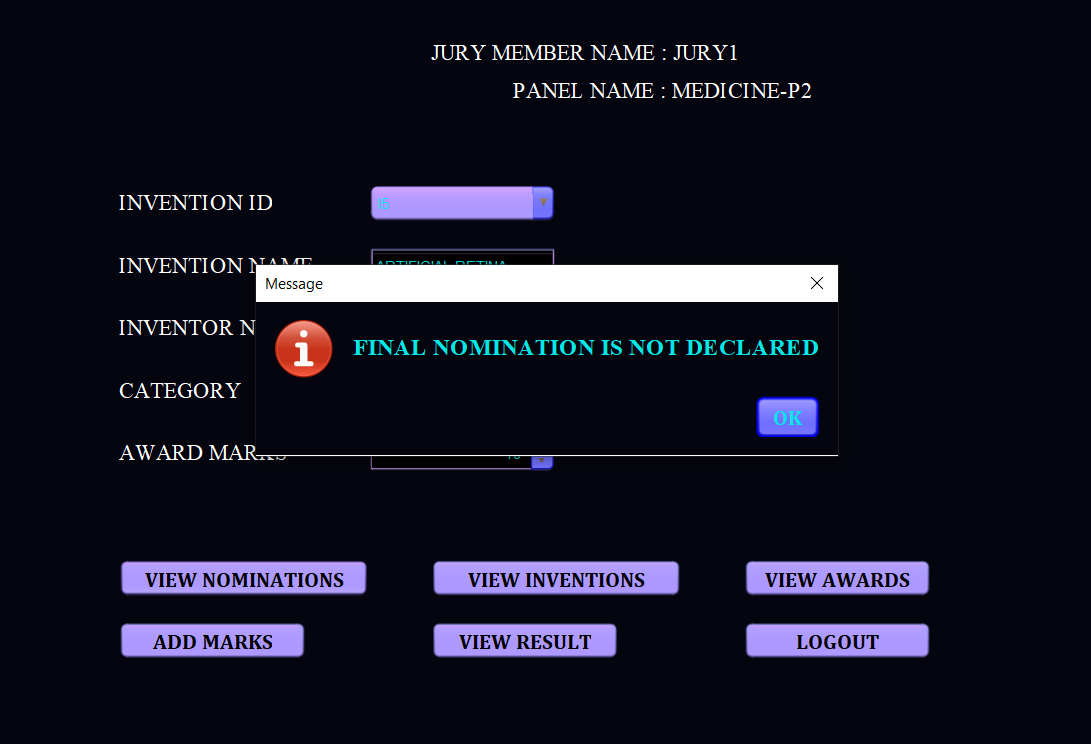


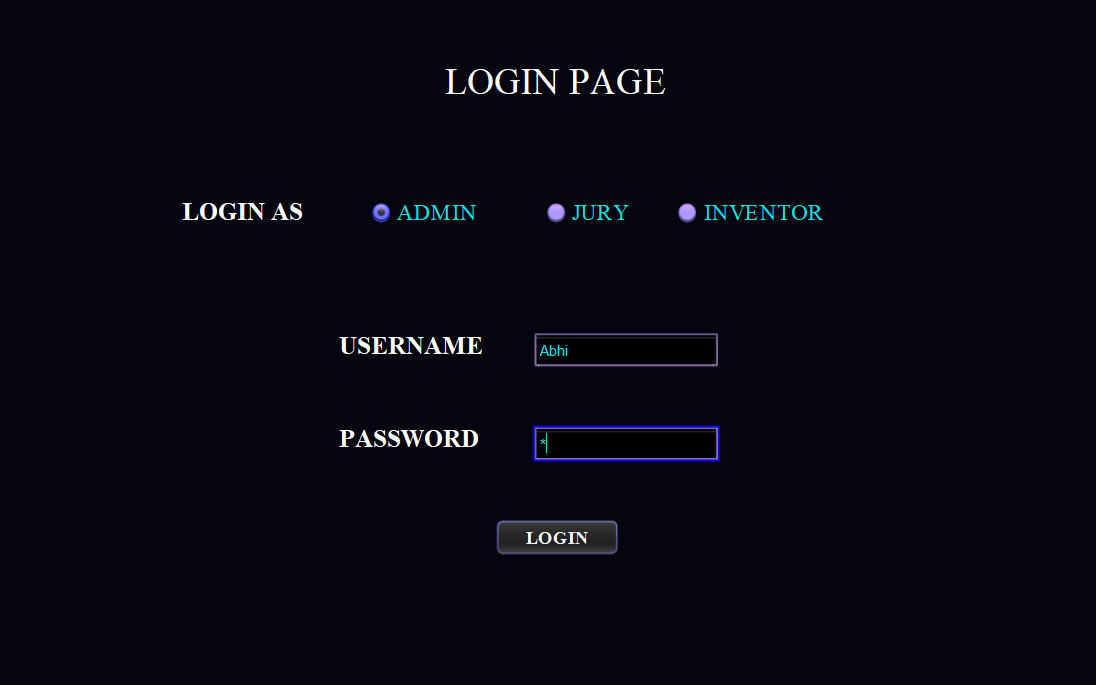


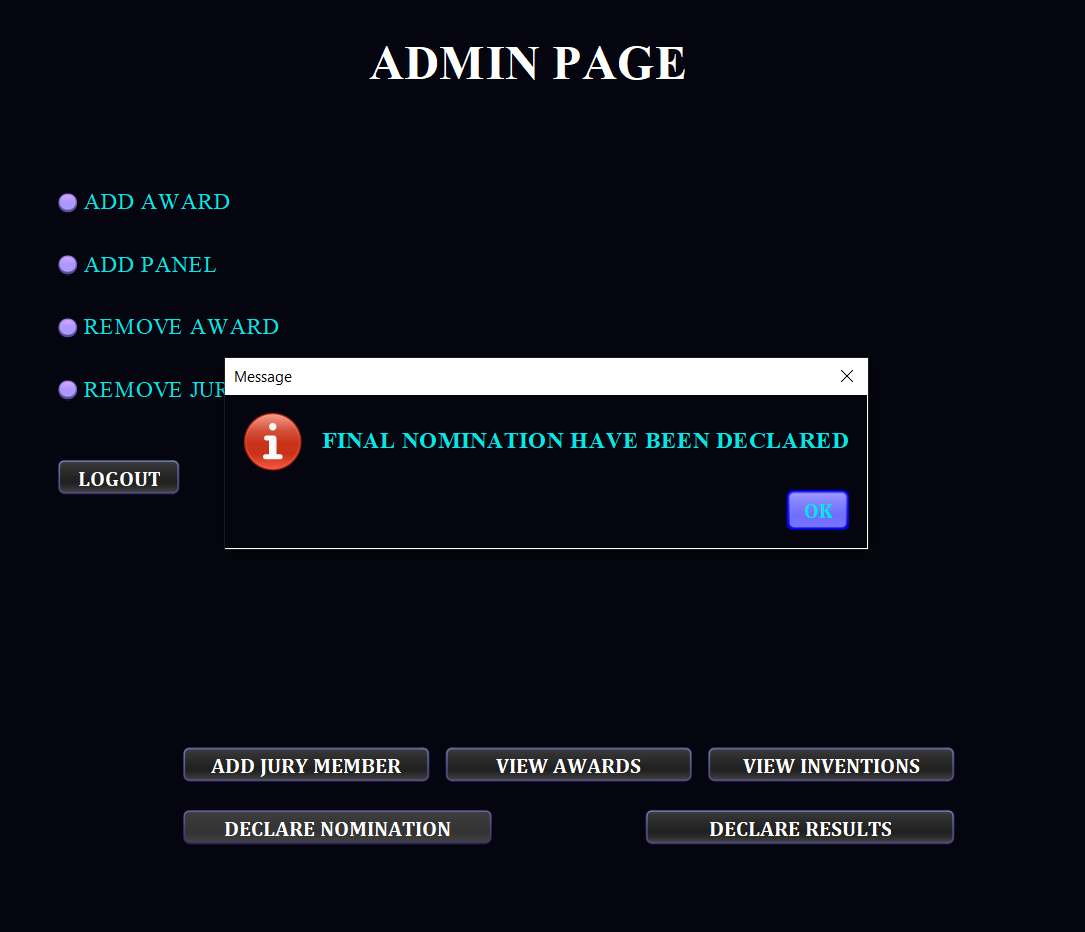


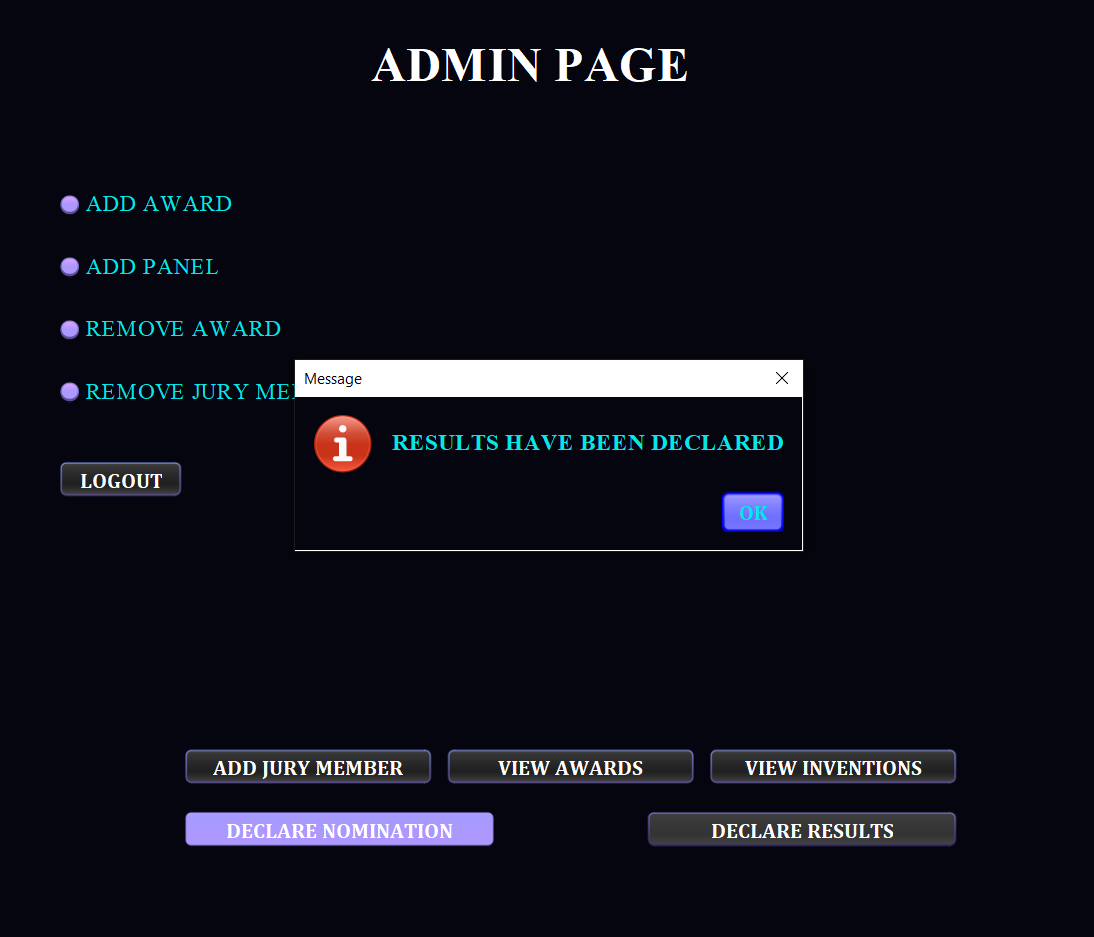


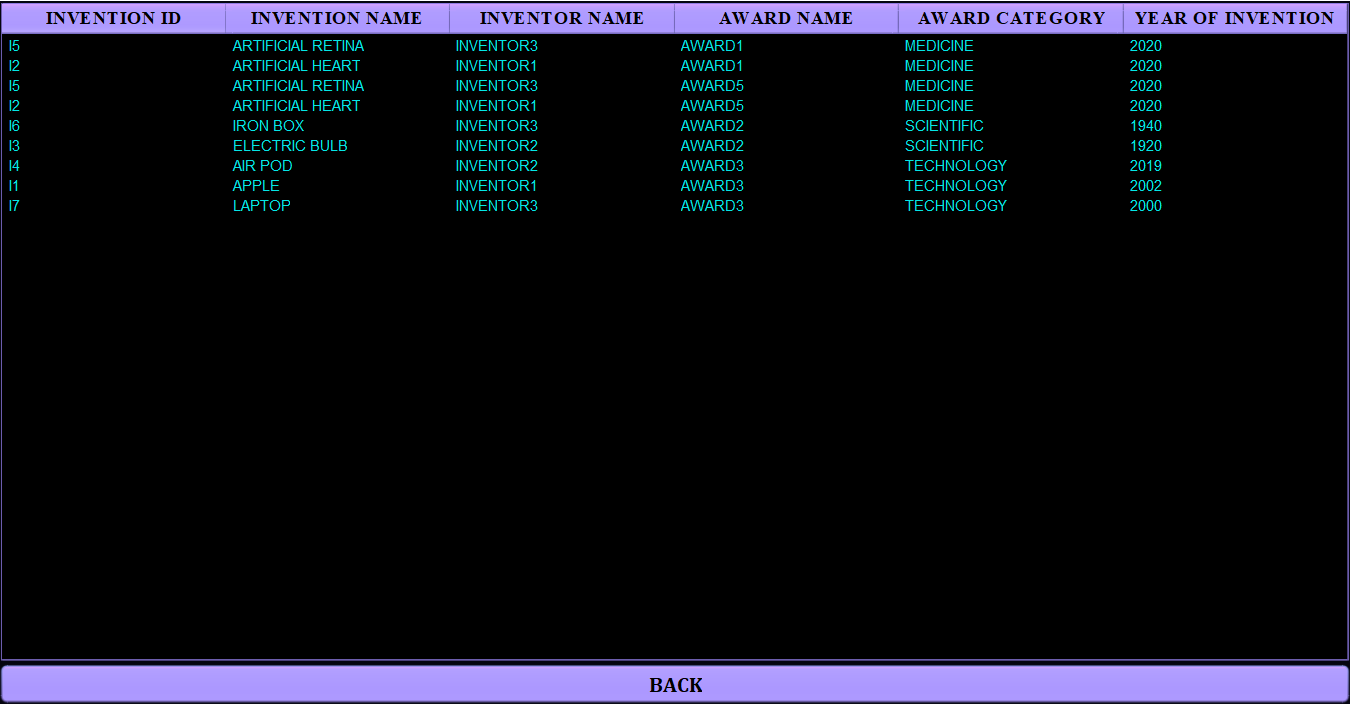


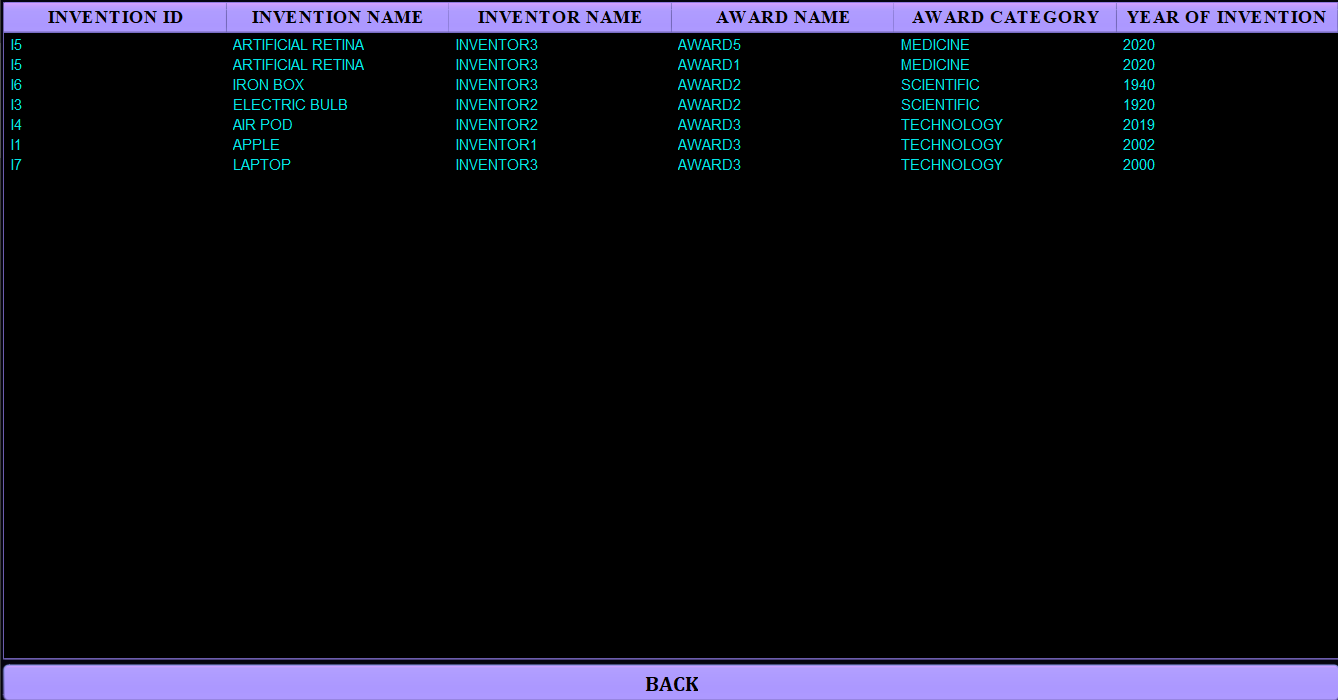
****

****

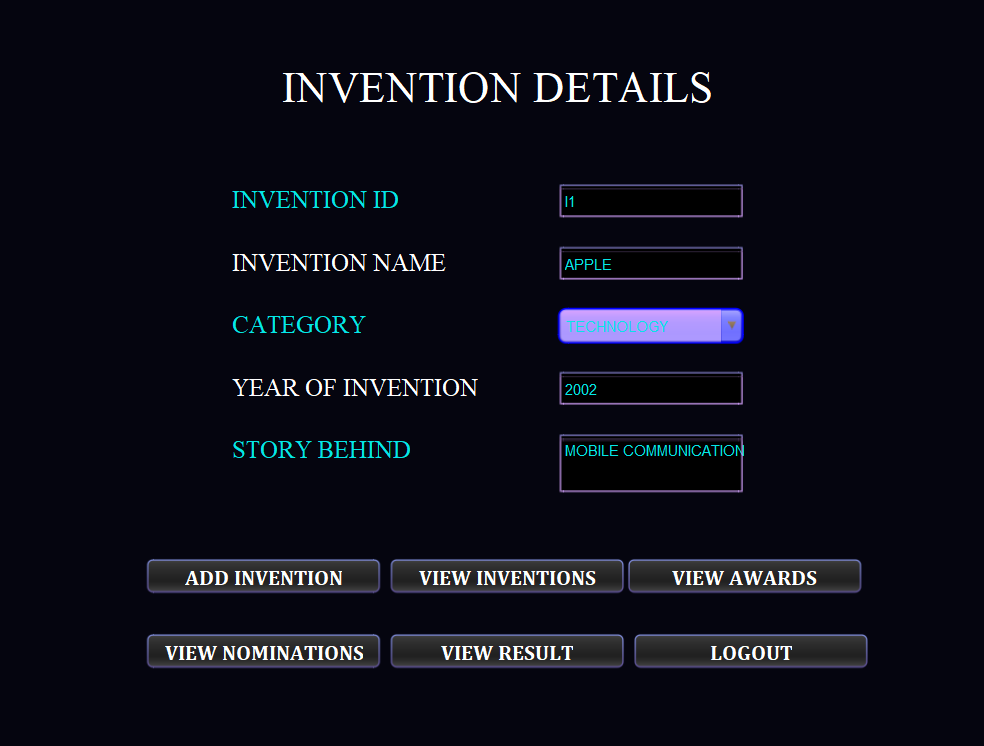
****

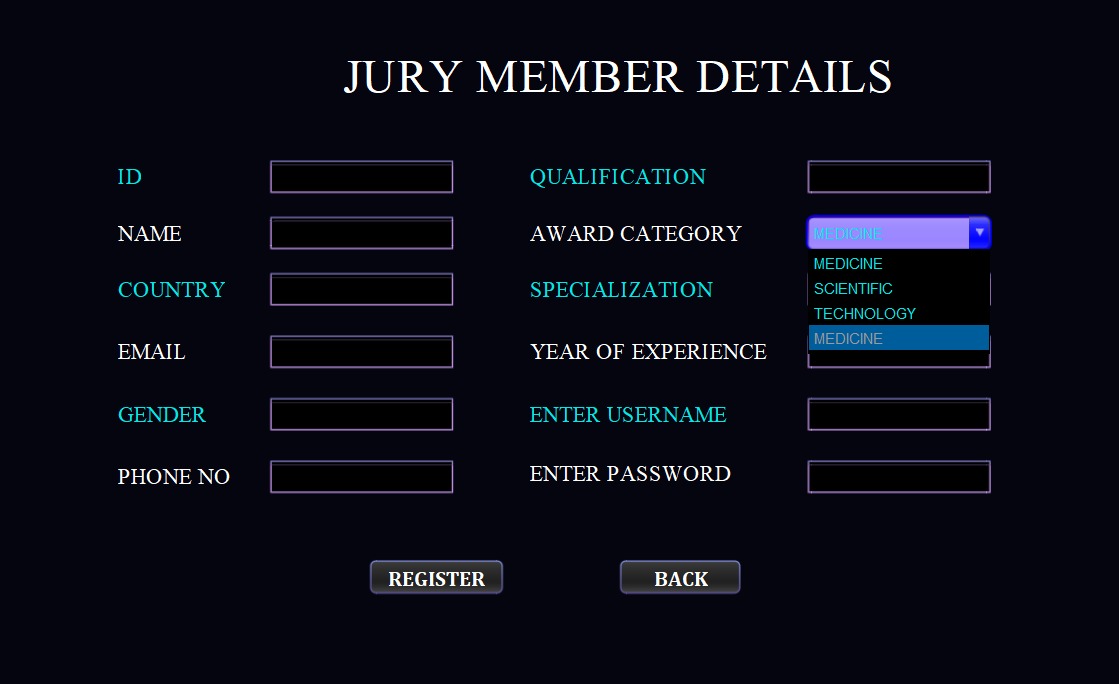
****

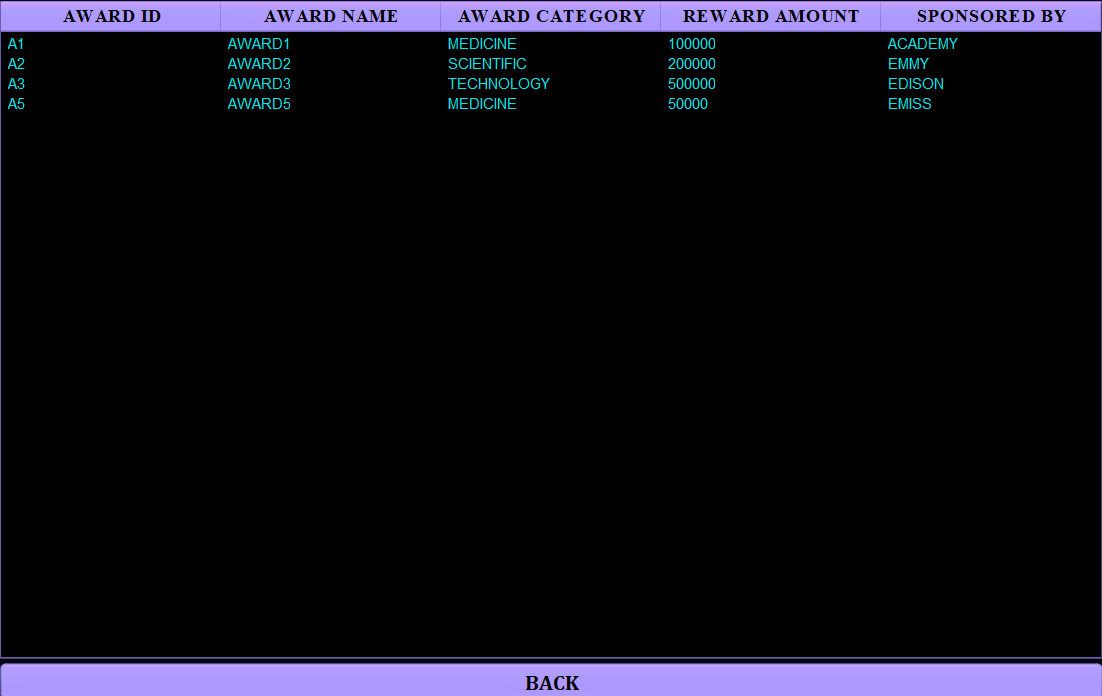
****

****

****

****

****

****

**Done By,**

* S. Abhishek - AM.EN.U4CSE19147
* Rahan Manoj – AM.EN.U4CSE19144
* Harsha Sathish – AM.EN.U4CSE19123
* Arvind Kumar K – AM.EN.U4CSE19109

**THANKYOU!!!**