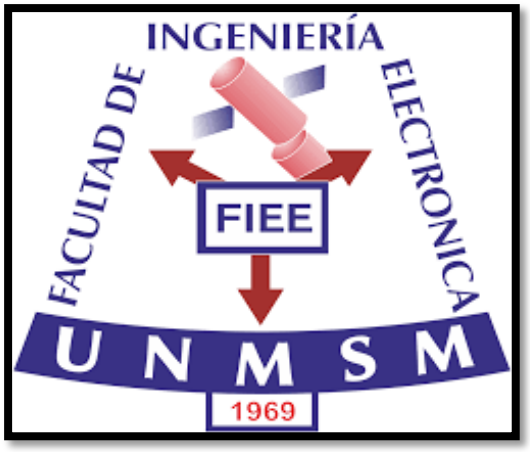
Informe Final



Sistema de Matricula de un Colegio

Curso : Programación Orientada a Objetos.

Profesor: Herminio Paucar.

Grupo: G2

INTEGRANTES:

- CABELLO NIETO, JOEL MAXIMILIANO 18190089
- CHAUCA PRINQUE, JHONNY JUNIOR 18190294
- ECHANDIA RAMOS CARLOS JOEL 18190302
- LEDESMA MENDOZA LINO JOAQUIN 18190307
- TORIBIO CHAVEZ JULIO CESAR 18190344
- Vilca ríos OSCAR PAOLO 18190323
- ZUÑIGA SALAS FRANK 18190325

2019

Introducción:

Actualmente el proceso de matricula en cualquiera colegio del estado es un proceso que llega a ser muy problemático debido a los diversos problemas que se encuentra en el proceso, como la escasa cantidad de vacantes o que el alumno no cumpla algunos requisitos.

Además, esa solo es uno de los diversos problemas que aquejan a los colegios estatales, una vez que los padres logran matricular a sus pequeños hijos no sabrán nada de como van en sus estudios a menos que algunos de los dos tenga tiempo libre y se disponga a ir a consultar, pero en la actualidad son pocos los que disponen de eso tiempo tan valioso. Que hace que estos no sepan del desempeño de su menor hijo en el colegio.

Por eso este proyecto tratara de dar una solución a esos problemas para así poder evitar lo problemático que puede ser la matricula de un alumno y también poder saber el desempeño de este sin la necesidad de estar visitando al colegio constantemente.

Motivación y Problema:

Actualmente casi todos los colegios estatales aún siguen usando un sistema de matrícula presencial que ocasiona que todos los años se forman inmensas colas con la finalidad de obtener una vacante, por eso vamos a vamos a desarrollar un software que haga que todo el papeleo que conlleva una matrícula sea más rápido y fácil.

Lo que nos motivó a desarrollar un sistema que ofrezca de manera eficiente a los padres de familia una manera de simplificar, agilizar y mejorar el proceso de matrícula para sus menores hijos. A su vez se busca brindar de forma continua las notas sobre el menor hijo.

Requerimientos del Sistema:

un requerimiento es la condición o capacidad que debe tener un sistema, producto, servicio o componente para satisfacer un contrato, estándar, especificación, u otros documentos formalmente establecido.
Se deben definir en la fase inicial junto con los stakeholders implicados para obtener una visión completa y compartida de todas las piezas y poder priorizar en base a los objetivos del proyecto.
Los requerimientos no te indican que diseño debe tener tu producto o como desarrollarlo. Te indican que features, funciones y contenidos se espera que tenga, y como deben los usuarios interactuar con él.

Funcionales:

Los requerimientos funcionales de un sistema, son aquellos que describen cualquier actividad que este deba realizar, en otras palabras, el comportamiento o función particular de un sistema o software cuando se cumplen ciertas condiciones.

Procesos. -

- Permitirá acceder al programa con una cuenta

- Permitirá el registrar de un alumno

- Se podrá modificar los datos del alumno

- Permitirá la elección del horario a estudiar

- Se podrá modificar el horario escogido

- Control y reporte de alumnos en cada periodo académico

- Información compuesta de alumnos, profesores y apoderados

- Historial académico del alumno

Interfaz Grafica. –

El programa contara con un Login donde se podrá ingresar como Alumno o como Profesor donde asignara diferentes ventanas dependiendo de como te registraste. El usuario dependiendo de si se loguea como Profesor o como alumno podrá ver o registrar, modificar o eliminar las notas o datos de un alumno.

Seguridad. –

El sistema solo aceptara usuarios que tengan un usuario y una contraseña brindadas por la institución educativa.

Técnicas:

Definen las condiciones que el programa debe cumplir, así como las limitaciones que el este posee.

Accesibilidad:

El programa podrá ser utilizado por todos los miembros la institución educativa dependiendo de un rango que puede ser o bien profesor o un alumno

Encriptación:

Solo los usuarios logeados como Profesor podrán agregar, modificar o eliminar las notas y/o datos de los alumnos.

Entorno:

Los datos serán guardados en una base de datos MySQL

Transicionales:

Son las condiciones del producto que deben implementarse para que pueda ejecutarse exitosamente.

Formación:

Si bien es cierto el orden y separación de cada operación es importante, por ello es necesario implementar un correcto uso de las jerarquías operacionales.

Conversión de datos:

Para este proceso debemos implementar un sistema de interfaz gráfica (Java Fx), de este modo podrá ser visto de manera sencilla y agradable por el usuario.

Código fuente:

-Validación de usuarios con el uso de condicionales, para permitir el acceso a una Stage (ventana) a través de la clase primaryStage dependiendo si se quiere ingresas como Alumno o Profesor • Para sacar de un Stage a otro se usa la clase FXMLLoader que carga el archivo XML correspondiente (ln. 36 y 45)

```
32          if (txtUser.getText().equals("Profesor") && txtPass.getText().equals("123")) {
33
34              lblStatus.setText("Acceso permitido");
35              Stage primaryStage = new Stage();
36              Parent root = FXMLLoader.load(getClass().getResource("/application/Main.fxml"));
37              Scene scene = new Scene(root,1006,697);
38              primaryStage.setScene(scene);
39              primaryStage.show();
40
41          }else if (txtUser.getText().equals("Alumno") && txtPass.getText().equals("123")){
42
43              lblStatus.setText("Acceso permitido");
44              Stage primaryStage = new Stage();
45              Parent root = FXMLLoader.load(getClass().getResource("/application/AlumnoView.fxml"));
46              Scene scene = new Scene(root,690,471);
47              primaryStage.setScene(scene);
48              primaryStage.show();
49
50          }else{
51              lblStatus.setText("Acceso denegado");
52          }
```

-Este método nos carga el Login a través del FXML (ln. 15) mostrando la primera ventana

```
13      public void start(Stage stage) throws Exception {
14
15          Parent parent = (Parent) FXMLLoader.load(getClass().getResource("/application/Login1.fxml"));
16
17          Scene scene = new Scene(parent);
18          stage.setScene(scene);
19          stage.setTitle("Matricula");
20          stage.show();
21
22      }
23
24
```

-Tenemos que inicializar los atributos de el clase Alumno ,ya que el programa nos pedirá que lo ingresemos para rellenar la base de datos. Usando constructores y métodos Getter.

```
public class Alumno {

    private int idAlumno;
    private String codigo;
    private String nombre;
    private String apellido;
    private String curso;
    private double pc;
    private double parcial;
    private double final1;
    private double promedio;

    public Alumno (int idAlumno, String codigo, String nombre, String apellido, String curso,
            double pc,double parcial,double final1, double promedio) {
        this.idAlumno = idAlumno;
        this.codigo = codigo;
        this.nombre = nombre;
        this.apellido = apellido;
        this.curso = curso;
        this.pc = pc;
        this.parcial = parcial;
        this.final1 = final1;
        this.promedio = promedio;
    }
```

- Primero importamos la clase Decimalformat para poder calcular el promedio(ln.89)

- Hacemos la consulta a la base de datos mediante el query(consulta), ejecutamos la consulta con el método executeQuery(ln.95)

- Listamos todos los datos hasta ahora a través del showBooks (ln. 96) Y por ultimo limpiamos las casillas (ln 97-104)

```
88      private void btnAgregar() {
89          DecimalFormat df = new DecimalFormat("#.00");
90          String query = "insert into registro values("+txtIdAlumno.getText()+",'"+txtCodigo.getText()+"',"
91                  + "'"+txtNombre.getText()+"','"+txtApellido.getText()+"','"+txtCurso.getText()+"',"
92                  + ""+txtPC.getText()+","+txtParcial.getText()+","+txtFinal.getText()+","
93                  + ""+ df.format((Double.parseDouble(txtParcial.getText())+Double.parseDouble(txtFinal.getText())
94                  +Double.parseDouble(txtPC.getText()))/3)+")";
95          executeQuery(query);
96          showBooks();
97          txtIdAlumno.setText("");
98          txtCodigo.setText("");
99          txtNombre.setText("");
100         txtApellido.setText("");
101         txtCurso.setText("");
102         txtPC.setText("");
103         txtParcial.setText("");
104         txtFinal.setText("");
105     }
```

- Primero importamos la clase Decimalformat para poder calcular el promedio(ln.109)

- Se hace lo mismo que en lo anterior solo que ya no se limpian las casillas
- Solo varia la sentencia SQL a "UPDATE"(ln. 110)

```
108     private void btnModificar() {
109     DecimalFormat df = new DecimalFormat("#.00");
110     String query = "UPDATE registro SET Codigo='"+txtCodigo.getText()+"',Nombre='"+txtNombre.getText()+"'"
111             + ",Apellido='"+txtApellido.getText()+"',Curso='"+txtCurso.getText()+"',PC="+txtPC.getText()+","
112             + "Parcial="+txtParcial.getText()+",Final"+txtFinal.getText()+",Promedio="
113             +df.format((Double.parseDouble(txtParcial.getText())+Double.parseDouble(txtFinal.getText())
114             +Double.parseDouble(txtPC.getText()))/3)+")\" WHERE ID="+txtIdAlumno.getText()+"";
115     executeQuery(query);
116     showBooks();
117
118     }
```

- Para eliminar solo se necesita el ID del alumno y la función del query ejecuta la consulta en la base de datos eliminando el dato seleccionada

```
121     private void btnEliminar() {
122         String query = "DELETE FROM registro WHERE ID="+txtIdAlumno.getText()+"";
123         executeQuery(query);
124         showBooks();
125     }
```

- Este método se conecta a la base dato con la clase Connection y luego con el Try Catch intentan ejecutar una sentencia SQL mediante el "createStatement()" si no logra conectar muestra un error

```
127     public void executeQuery(String query) {
128         Connection conn = getConnection();
129         Statement st;
130         try {
131             st = conn.createStatement();
132             st.executeUpdate(query);
133         } catch (Exception e) {
134             e.printStackTrace();
135         }
136     }
137
```

- Este método permite la conexión del programa con la base de datos mediante una URL en la que se brinda el host , el nombre del esquema , el usuario y la contraseña
- Si no logra conectar bota otro error

```
143     public Connection getConnection() {
144         Connection conn;
145         try {
146             conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/matricula","root","root");
147             return conn;
148         }
149         catch (Exception e){
150             e.printStackTrace();
151             return null;
152         }
153     }
```

- Este método carga datos de la DB a la tabla , a través de las clases setCellValuerFactory que asigna un valor a cada celda

```
179    public void showBooks() {
180
181        ObservableList<Alumno> list = getAlumnoList();
182
183        colID.setCellValueFactory(new PropertyValueFactory<Alumno,Integer>("idAlumno"));
184        colCodigo.setCellValueFactory(new PropertyValueFactory<Alumno,String>("codigo"));
185        colNombre.setCellValueFactory(new PropertyValueFactory<Alumno,String>("nombre"));
186        colApellido.setCellValueFactory(new PropertyValueFactory<Alumno,String>("apellido"));
187        colCurso.setCellValueFactory(new PropertyValueFactory<Alumno,String>("curso"));
188        colPC.setCellValueFactory(new PropertyValueFactory<Alumno,Double>("pc"));
189        colParcial.setCellValueFactory(new PropertyValueFactory<Alumno,Double>("parcial"));
190        colFinal.setCellValueFactory(new PropertyValueFactory<Alumno,Double>("final1"));
191        colPromedio.setCellValueFactory(new PropertyValueFactory<Alumno,Double>("promedio"));
192
193        TableView.setItems(list);
194    }
195
```

- Cuando se logea como Alumno el programa solo permite observar los datos guardados en el DB.
- El método showBooks se encargó de ellos mostrando todos los datos que están guardados.

```
101        public void showBooks() {
102
103            ObservableList<Alumno> list = getAlumnoList();
104
105            colID.setCellValueFactory(new PropertyValueFactory<Alumno,Integer>("idAlumno"));
106            colCodigo.setCellValueFactory(new PropertyValueFactory<Alumno,String>("codigo"));
107            colNombre.setCellValueFactory(new PropertyValueFactory<Alumno,String>("nombre"));
108            colApellido.setCellValueFactory(new PropertyValueFactory<Alumno,String>("apellido"));
109            colCurso.setCellValueFactory(new PropertyValueFactory<Alumno,String>("curso"));
110            colPC.setCellValueFactory(new PropertyValueFactory<Alumno,Double>("pc"));
111            colParcial.setCellValueFactory(new PropertyValueFactory<Alumno,Double>("parcial"));
112            colFinal.setCellValueFactory(new PropertyValueFactory<Alumno,Double>("final1"));
113            colPromedio.setCellValueFactory(new PropertyValueFactory<Alumno,Double>("promedio"));
114
115            TableView.setItems(list);
116
117
118        }
```

Diagrama de clases: