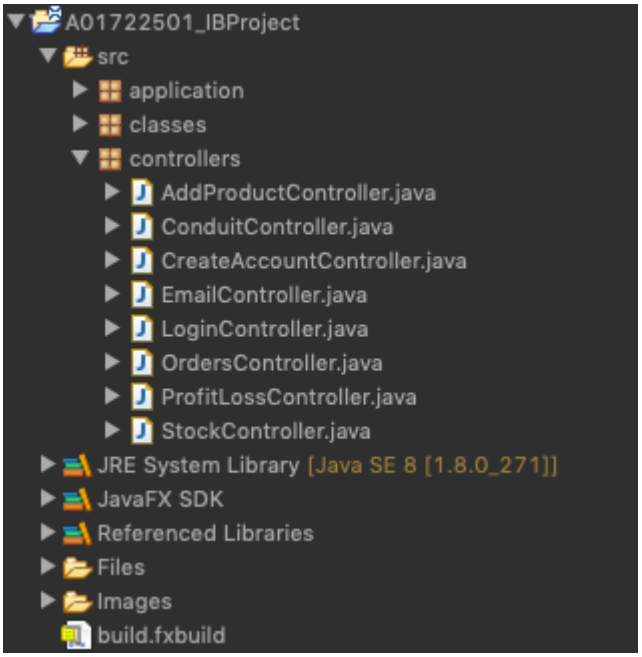
Criterion C: Development

The developed program is a java based program developed for Mr. X. It serves as a tool to keep track of his orders, Stock, Profits and losses(PL) and also to send text mails. The client can input the orders the business makes and with that information the program keeps track of the stock and PL. The program also features a login system to ensure only the people Mr. X wants, can access it.



The whole program uses Java FX elements with scene builder to create the button's, textfield's, image view's etc...

Program Structure

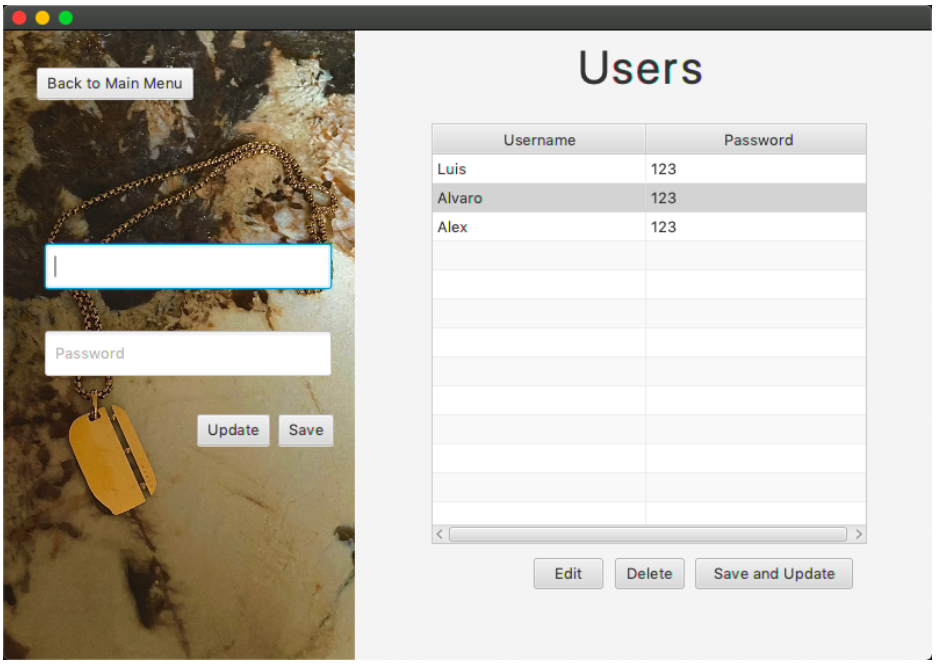
Login Window

“The login screen” is a way to validate the user entering the system, by cross referencing the inputted information with the csv database with the saved username and passwords.

```
@FXML
void LogInPress(ActionEvent event) throws IOException {

    if (Username.getText().equals("") || Password.getText().equals("")) {

        Alert alert = new Alert(AlertType.WARNING);
        alert.setTitle("Validate Fields");
        alert.setHeaderText(null);
        alert.setContentText("Please enter all fields");
        alert.showAndWait();
    }

    else {
    Path path = Paths.get("Files/Users.csv");
    try {
        BufferedReader br = Files.newBufferedReader(path,StandardCharsets.US_ASCII);
        String line = br.readLine();
        int validate = 0;
        while (line != null) {
            String [] logins = line.split(",");

            if (Username.getText().equals(logins[0])&&(Password.getText().equals(logins[1]))) {

                validate = 1;
                Scene();
            }

            line = br.readLine();

        }

        if (validate == 0) {
        Alert alert = new Alert(AlertType.WARNING);
        alert.setTitle("Incorrect Credentials");
        alert.setHeaderText(null);
        alert.setContentText("Your username and/or password is incorrect");
        alert.showAndWait();
        }
        Username.setText("");
        Password.setText("");

    }
    catch (IOException e) {
    }
}
}
```

Using LoginController.java I first check if there are any empty text fields and give a warning if there are. After that i read the csv file and check to see if the information inputted is correct, and send the user to the next screen, if not an alert message is displayed saying the username or password is incorrect.

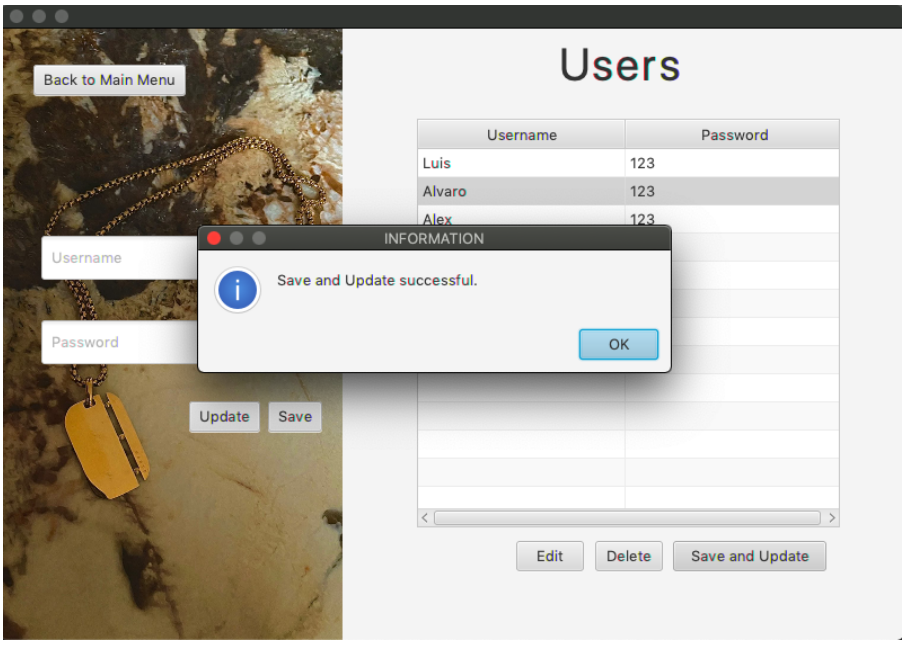
Create account screen



“The create account screen” saves new Users with a unique username and password, in a csv called Users.csv

```
@FXML
public void SavingAndUpdating(ActionEvent event) {

    FileWriter fileWriter = null;
    try {

        fileWriter = new FileWriter("Files/Users.csv");


        for (int i = 0; i < UsersTable.getItems().size(); i++) {

            fileWriter.append(UsersTable.getItems().get(i).getUsername());
            fileWriter.append(",");
            fileWriter.append(UsersTable.getItems().get(i).getPassword());
            fileWriter.append("\n");

        }
    }

    catch (Exception ex) {
        ex.printStackTrace();
    }


    try {

        fileWriter.flush();
        fileWriter.close();

        int ValidateFields = 1;

        if(ValidateFields == 1) {
            Alert alert = new Alert(AlertType.INFORMATION);
            alert.setTitle("INFORMATION");
            alert.setHeaderText(null);
            alert.setContentText("Save and Update successful.");
            alert.showAndWait();
        }

    }

    catch (IOException e){
        e.printStackTrace();
    }

}
```

After the user is finished saving information to the table he may click the “Save and Update” button using CreateAccountController.java to save the information in the pre existent csv. Using the SavingAndUpdating method the program uses the append method to save the information saved in the table into the csv database.

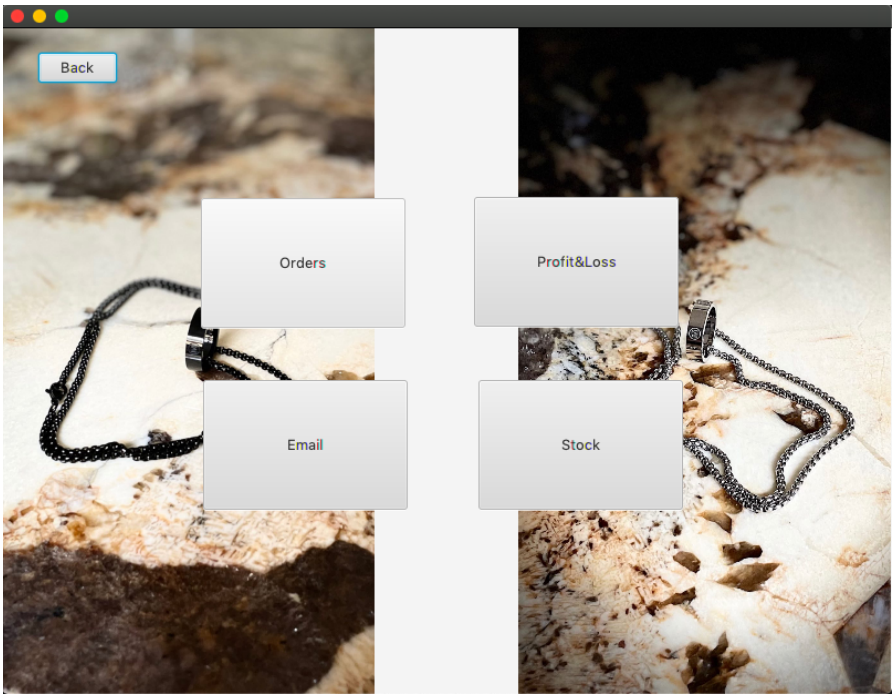


After saving the information, the program alerts the user if the save was successful

```java
@Override
public void initialize(URL location, ResourceBundle resources) {
    // TODO Auto-generated method stub

    UsernameColumn.setCellValueFactory(new PropertyValueFactory<Users, String>("Username"));
    PasswordColumn.setCellValueFactory(new PropertyValueFactory<Users, String>("Password"));

    Path path = Paths.get("Files/Users.csv");
    try {
        BufferedReader br = Files.newBufferedReader(path,StandardCharsets.US_ASCII);
        String line = br.readLine();
        while (line != null) {
            String [] characteristics = line.split(",");
            Users user = new Users(characteristics[0], characteristics[1]);
            UsersTable.getItems().add(user);
            line = br.readLine();
        }
    } catch (IOException e) {
    }

}
```

Using the initializable method, when the screen starts, it reads the csv and inputs the information to the tableview using a buffered reader.

Conduit Screen

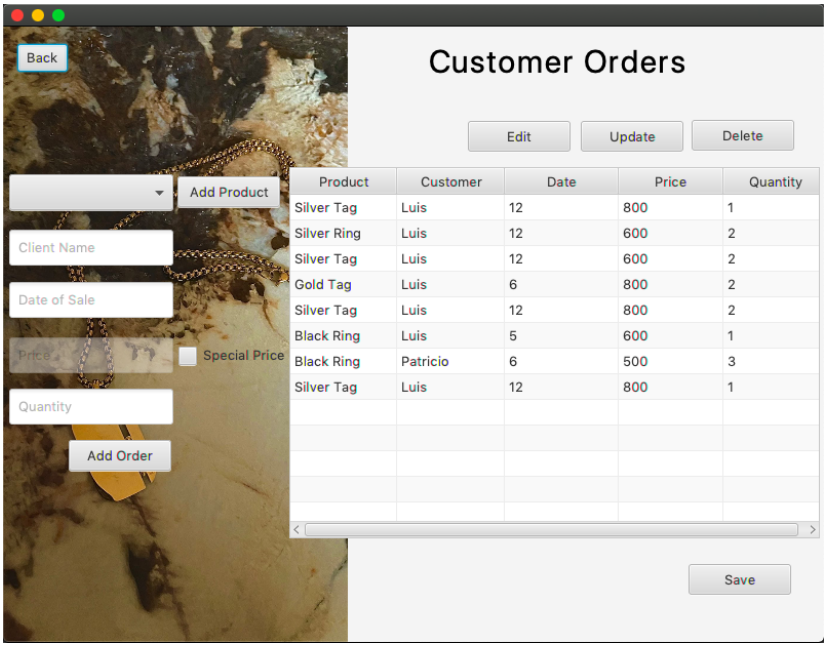


The screen serves as a “Conduit” or guide to all the other screens in the program(Orders, PL, Email and Stock)

```
@FXML
public void backToLogIn(ActionEvent event) {
    application.Main.mainStage = (Stage) backButton.getScene().getWindow();
    application.Main.mainStage.setScene(application.Main.scene1);
}

@FXML
public void goToOrders(ActionEvent event) throws IOException {
     orders = FXMLLoader.load(getClass().getResource("/application/OrderGUI.fxml"));
     scene4 = new Scene(orders);
    application.Main.mainStage = (Stage) orderButton.getScene().getWindow();
    application.Main.mainStage.setScene(scene4);


}

@FXML
public void goToProfitLoss(ActionEvent event) throws IOException {
    profitloss = FXMLLoader.load(getClass().getResource("/application/ProfitLossGUI.fxml"));
    scene5 = new Scene(profitloss);
    application.Main.mainStage = (Stage) profitLossButton.getScene().getWindow();
    application.Main.mainStage.setScene(scene5);

}

@FXML
public void goToEmail (ActionEvent event) {
    application.Main.mainStage = (Stage) emailButton.getScene().getWindow();
    application.Main.mainStage.setScene(application.Main.scene6);

}

@FXML
public void goToStock(ActionEvent event) throws IOException {

    stock = FXMLLoader.load(getClass().getResource("/application/StockGUI.fxml"));
    scene7 = new Scene(stock);
    application.Main.mainStage = (Stage) stockButton.getScene().getWindow();
    application.Main.mainStage.setScene(scene7);

}
```

Using ConduitController.java the program uses different methods to send the user to the screen that the user wants. Each button has a method, for example the “Orders” button uses the goToOrders method.

Order's Screen



The "Orders" screen uses the OrdersController.java class to add orders to a csv and the table in the view.

```java
@Override
public void initialize(URL location, ResourceBundle resources)   {
    // TODO Auto-generated method stub


    try {
        String csvSplit = (",");
        File file = new File("Files/Products.CSV");
        BufferedReader br = new BufferedReader(new FileReader(file)); // declare el buffered aqui
        String line;
        List<String> ProductsPrice = new ArrayList<String>();

        while((line = br.readLine()) != null) {
            String [] attributes = line.split(csvSplit);
            productChooser.getItems().add(attributes[0]);


            ProductsPrice.add(attributes[2]);

            productChooser.getSelectionModel().selectedIndexProperty().addListener(
                    (ObservableValue<? extends Number> ov, Number old_val,
                    Number new_val) -> {
                        priceField.setText(ProductsPrice.get(new_val.intValue()));
                        oldValue = ProductsPrice.get(new_val.intValue());

            });


        }


        br.close();

    }

    catch(FileNotFoundException e){
        new File("Files/Products.CSV");
        e.printStackTrace();
    }
    catch(IOException e) {
        e.printStackTrace();
    }


    productColumn.setCellValueFactory(new PropertyValueFactory<Orders, String>("productName"));
    customerColumn.setCellValueFactory(new PropertyValueFactory<Orders, String>("customerName"));
    dateColumn.setCellValueFactory(new PropertyValueFactory<Orders, String>("dateOfOrder"));
    priceColumn.setCellValueFactory(new PropertyValueFactory<Orders, Integer>("price"));
    quantityColumn.setCellValueFactory(new PropertyValueFactory<Orders, Integer>("quantity"));


    Path path = Paths.get("Files/Orders.csv");
    try {
        BufferedReader br = Files.newBufferedReader(path,StandardCharsets.US_ASCII);
        String line = br.readLine();
        while (line != null) {
            String [] characteristics = line.split(",");
            Orders order = new Orders(characteristics[0],characteristics[1],characteristics[2],characteristics[3],characteristics[4]);
            OrdersTable.getItems().add(order);
            line = br.readLine();
        }
    } catch (IOException e) {
    }
}
```

Using the intializable method when the screen is opened, It first reads “Products.csv” and gets the name of each unique product and sets those values into the choicebox. A listener is also used to set the base prices of each unique product. If a “Products.csv” does not exist a new file will be created.

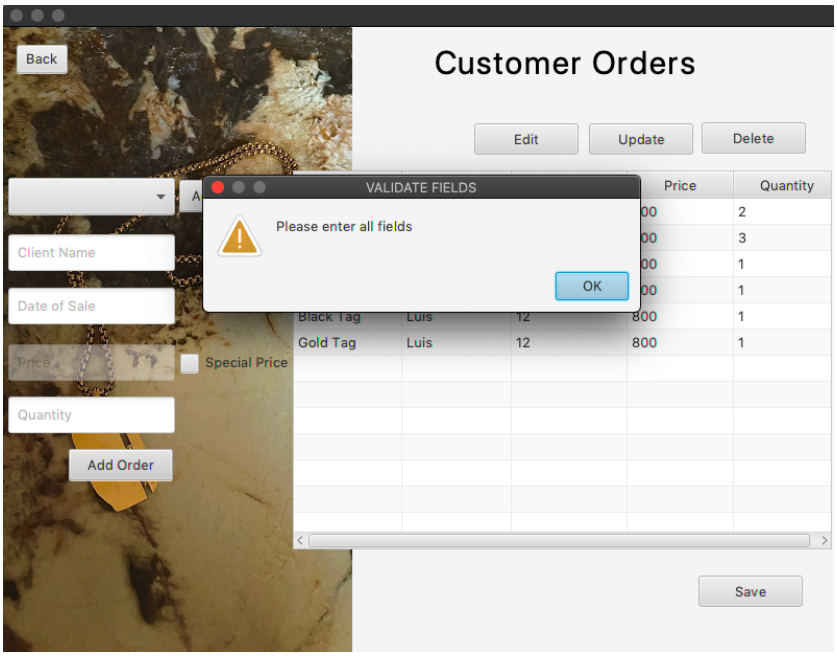
It also reads the orders csv using a buffered reader and sets the table values with the according information.

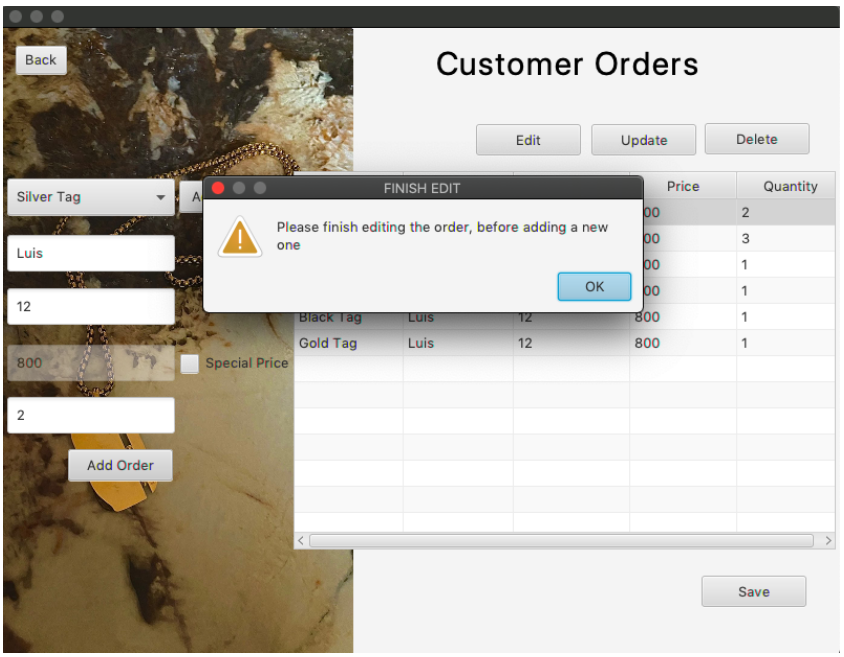
```
@FXML
public void AddingOrder(ActionEvent event) {
    int y = 0;
    if(x == 1) {
        Alert alert = new Alert(AlertType.WARNING);
        alert.setTitle("FINISH EDIT");
        alert.setHeaderText(null);
        alert.setContentText("Please finish editing the order, before adding a new one");
        alert.showAndWait();
    }

    else{
        if(clientNameField.getText().equals("") | dateOfSaleField.getText().equals("") | priceField.getText().equals("") | quantityField.getText().equals("")  ) {


            Alert alert = new Alert(AlertType.WARNING);
            alert.setTitle("VALIDATE FIELDS");
            alert.setHeaderText(null);
            alert.setContentText("Please enter all fields");
            alert.showAndWait();

        }

        else{

            try {
                int x = Integer.parseInt(quantityField.getText());

                if (x == 0) {

                    quantityField.setText("1");
                    Alert alert = new Alert(AlertType.INFORMATION);
                    alert.setTitle("INFORMATION");
                    alert.setHeaderText(null);
                    alert.setContentText("Your quantity has to be more than zero");
                    alert.showAndWait();

            }

                else {

                    y = y + 1;
                }
                System.out.println(y);

            }

            catch(NumberFormatException e) {
                e.printStackTrace();


                    quantityField.setText("1");

                    Alert alert = new Alert(AlertType.INFORMATION);
                    alert.setTitle("INFORMATION");
                    alert.setHeaderText(null);
                    alert.setContentText("Your quantity has to be a number");
                    alert.showAndWait();



            }

            try {
                int x = Integer.parseInt(priceField.getText());

                if (x == 0) {

                    quantityField.setText("1");
                    Alert alert = new Alert(AlertType.INFORMATION);
                    alert.setTitle("INFORMATION");
                    alert.setHeaderText(null);
                    alert.setContentText("Your price has to be more than zero");
                    alert.showAndWait();
```

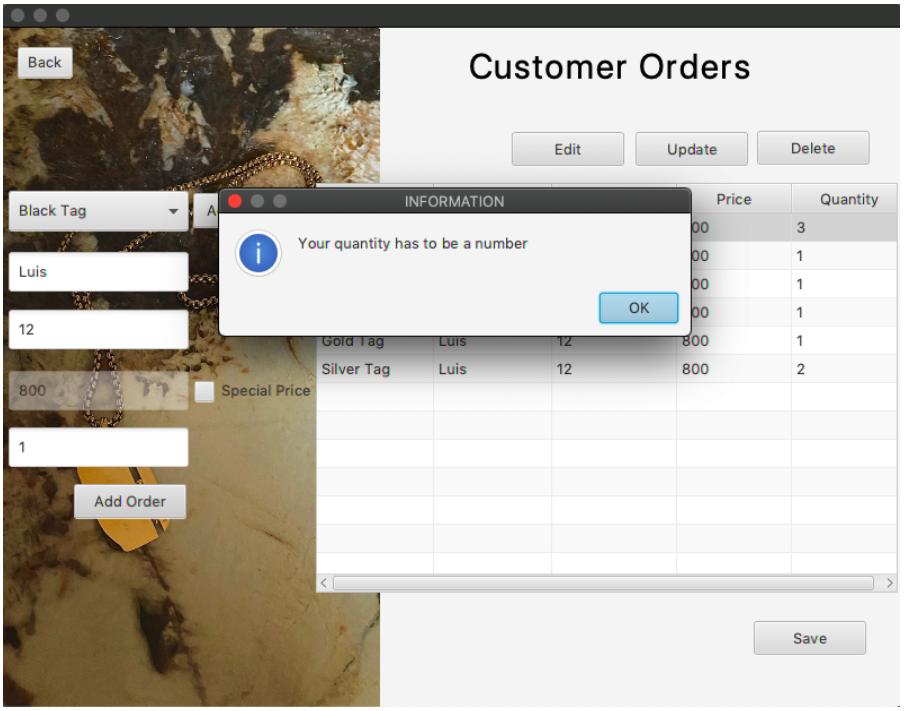
```
            else {
                if (y == 1) {

                    y = 2;
                }

            }

            System.out.println(y);
        }

        catch(NumberFormatException e) {
            e.printStackTrace();


                priceField.setText("");
                Alert alert = new Alert(AlertType.INFORMATION);
                alert.setTitle("INFORMATION");
                alert.setHeaderText(null);
                alert.setContentText("Your price has to be a number");
                alert.showAndWait();


        }
        // agregar check para que si sean numeros, el price y el quantity.

        if (y == 2) {

            Orders order = new Orders(productChooser.getSelectionModel().getSelectedItem(),clientNameField.getText(), dateOfSaleField.getText(), priceField.getText(), quar
            OrdersTable.getItems().addAll(order);

            productChooser.setValue(null);
            clientNameField.setText(null);
            dateOfSaleField.setText(null);
            priceField.setText(null);
            quantityField.setText(null);
            specialPrice.setSelected(false);

            Alert alert = new Alert(AlertType.INFORMATION);
            alert.setTitle("INFORMATION");
            alert.setHeaderText(null);
            alert.setContentText("Order has been added");
            alert.showAndWait();

            y = 0;
        }



    }
}
```

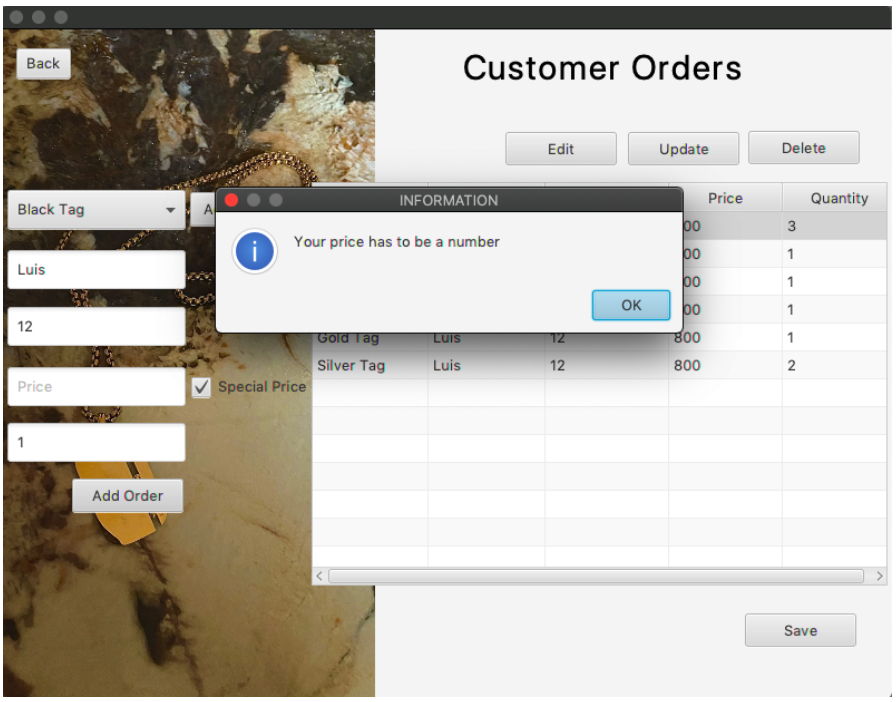
When the “add order” button is clicked the “AddingOrder” method is called, and it first checks if anything is being edited using the “EditingProduct” method and sends an alert if the user is editing something, if not it checks if any fields are empty and sends an alert if they are.

It also checks that everything is the right data-type. it finally adds the order to the table.




Back
Customer Orders
Edit
Update
Delete
Silver Tag
Luis
12
800
Special Price
2
Add Order
FINISH EDIT
Please finish editing the order, before adding a new one
OK
Price
Quantity
Black Tag Luis 12 800 1
Gold Tag Luis 12 800 1
Save


Back
Customer Orders
Edit
Update
Delete
Black Tag
Luis
12
800
Special Price
1
Add Order
INFORMATION
Your quantity has to be a number
OK
Price
Quantity
Gold Tag Luis 12 800 1
Silver Tag Luis 12 800 2
Save




```
@FXML
public void DeletingProduct(ActionEvent event) {

    Alert alert = new Alert(AlertType.CONFIRMATION);
    alert.setTitle("CONFIRMATION");
    alert.setHeaderText(null);
    alert.setContentText("Are you sure you want to delete this Order?");
    Optional <ButtonType> action = alert.showAndWait();

        if(action.get() == ButtonType.OK) {
            OrdersTable.getItems().removeAll(OrdersTable.getSelectionModel().getSelectedItem());

        } else if (action.get()==ButtonType.CANCEL){

        }
}
```

This method sends a confirmation and deletes the selection.

```
@FXML
public void SavingOrders(ActionEvent event) {
    FileWriter fileWriter = null;
    try {

        fileWriter = new FileWriter("Files/Orders.csv");


        for (int i = 0; i < OrdersTable.getItems().size(); i++) {

            fileWriter.append(OrdersTable.getItems().get(i).getProductName());
            fileWriter.append(",");
            fileWriter.append(OrdersTable.getItems().get(i).getCustomerName());
            fileWriter.append(",");
            fileWriter.append(OrdersTable.getItems().get(i).getDateOfOrder());
            fileWriter.append(",");
            fileWriter.append(OrdersTable.getItems().get(i).getPrice());
            fileWriter.append(",");
            fileWriter.append(OrdersTable.getItems().get(i).getQuantity());
            fileWriter.append("\n");

        }



    }

    catch (Exception ex) {
        ex.printStackTrace();
    }


    try {

        fileWriter.flush();
        fileWriter.close();
```

This method saves to “Orders.csv” and sends an alert of success.

```
@FXML
public void EditingProduct(ActionEvent event) {

    productChooser.setValue(OrdersTable.getSelectionModel().getSelectedItem().getProductName());
    clientNameField.setText(OrdersTable.getSelectionModel().getSelectedItem().getCustomerName());
    dateOfSaleField.setText(OrdersTable.getSelectionModel().getSelectedItem().getDateOfOrder());
    priceField.setText(OrdersTable.getSelectionModel().getSelectedItem().getPrice());
    quantityField.setText(OrdersTable.getSelectionModel().getSelectedItem().getQuantity());

    x = 1;

}
```

This method sets the fields with selected values.

```
@FXML
public void Updating(ActionEvent event) {


   if (x == 1) {
       if( clientNameField.getText().equals("") | dateOfSaleField.getText().equals("") | priceField.getText().equals("")) {


           Alert alert = new Alert(AlertType.WARNING);
           alert.setTitle("VALIDATE FIELDS");
           alert.setHeaderText(null);
           alert.setContentText("Please enter all fields");
           alert.showAndWait();

       }

       else {

       OrdersTable.getItems().removeAll(OrdersTable.getSelectionModel().getSelectedItem());

       Orders order = new Orders(productChooser.getSelectionModel().getSelectedItem(),clientNameField.getText(), dateOfSaleField.getText(),
               priceField.getText(), quantityField.getText() );
       OrdersTable.getItems().addAll(order);

       productChooser.setValue(null);
       clientNameField.setText(null);
       dateOfSaleField.setText(null);
       priceField.setText(null);
       quantityField.setText(null);
       specialPrice.setSelected(false);

       x = 0;

           }

   }

       else {

           Alert alert = new Alert(AlertType.INFORMATION);
           alert.setTitle("INFORMATION");
           alert.setHeaderText(null);
           alert.setContentText("There is nothing to edit, please choose something to edit.");
           alert.showAndWait();

       }
}
```

This method updates the values with the new information.

```
@FXML
public void ActivateSpecialPrice(ActionEvent event) throws IOException {



        if (specialPrice.isSelected()) {
            priceField.setEditable(true);
            priceField.setDisable(false);


        }

        else {
            priceField.setEditable(false);
            priceField.setDisable(true);
            priceField.setText(oldValue);



        }
```

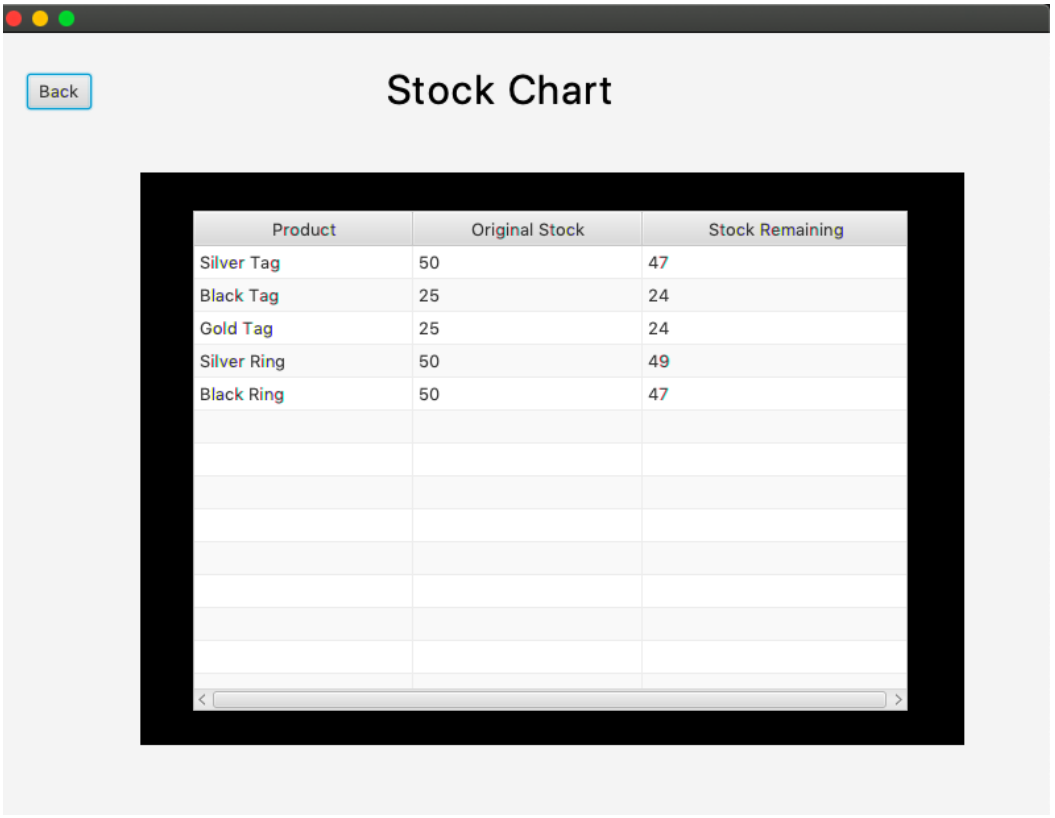
When the checkbox is clicked it makes the TextField editable, if it is clicked again it becomes uneditable and sets the original value.

Profit and Loss Screen


```java
@Override
public void initialize(URL location, ResourceBundle resources) {
    // TODO Auto-generated method stub
    productColumn.setCellValueFactory(new PropertyValueFactory<profitLoss, String>("productName"));
    earningsColumn.setCellValueFactory(new PropertyValueFactory<profitLoss, String>("Earnings"));
    costsColumn.setCellValueFactory(new PropertyValueFactory<profitLoss, String>("Capital"));
    profitColumn.setCellValueFactory(new PropertyValueFactory<profitLoss, String>("profits"));


    Path productsPath = Paths.get("Files/Products.csv");
    try {
        BufferedReader br = Files.newBufferedReader(productsPath,StandardCharsets.US_ASCII);
        String line = br.readLine();


        while (line != null) {
            String productsName = null; // listo
            int Earnings = 0; // listo
            String Capital = "0"; // Listo
            int Profits = 0;

            String [] characteristicsProducts = line.split(",");
            productsName = characteristicsProducts[0];
            Capital = characteristicsProducts[2];


            Path pathOrders = Paths.get("Files/Orders.csv");

                BufferedReader brOrder = Files.newBufferedReader(pathOrders,StandardCharsets.US_ASCII);
                String lineOrder = brOrder.readLine();

                while(lineOrder != null) {
                    String [] characteristicsO = lineOrder.split(",");

                    if (productsName.equals(characteristicsO[0])) {

                        Earnings = Earnings + (Integer.parseInt(characteristicsO[3]) * Integer.parseInt(characteristicsO[4]));

                    }
                    lineOrder = brOrder.readLine();
                }

            Profits = Earnings - Integer.parseInt(Capital);

            profitLoss PL = new profitLoss(productsName, String.valueOf(Earnings), Capital , String.valueOf(Profits));
            ProfitLossTable.getItems().add(PL);
            line = br.readLine();

        }
    }

    catch(IOException e) {
        e.printStackTrace();
    }
    catch(NumberFormatException e) {
        e.printStackTrace();

    }
```

ProfitLossController.java uses the intilizable method when the screen is opened and it reads the “Products.csv” and “Orders.csv” to calculate the values necessary and display them on the TableView.

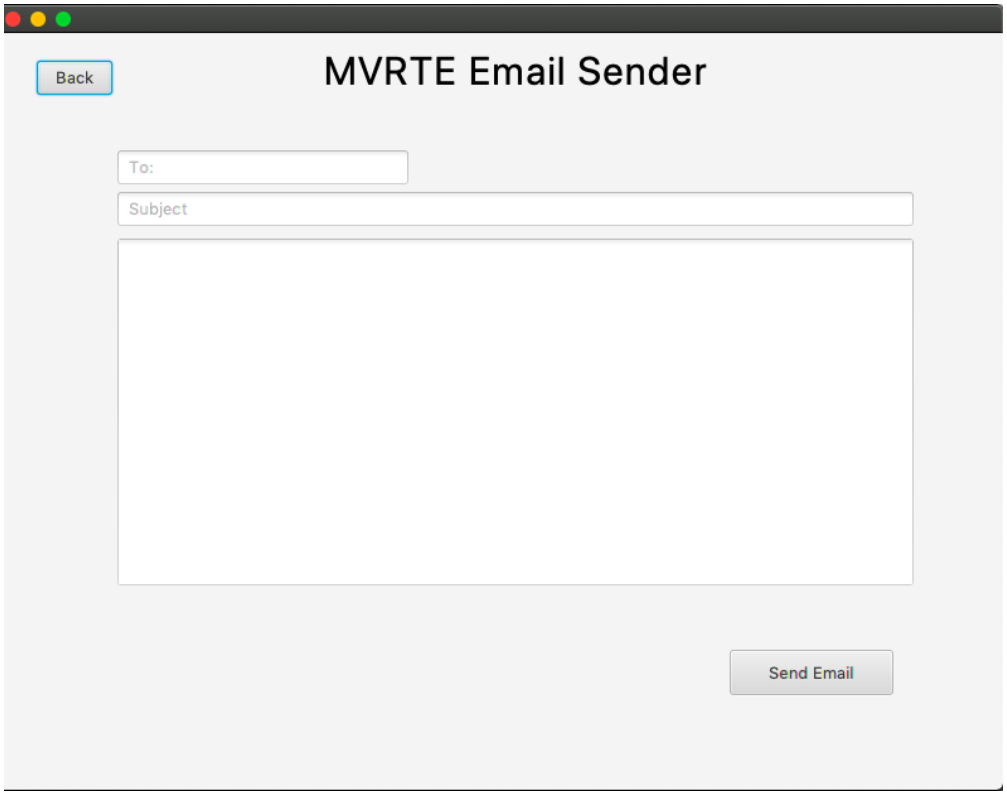
Stock Screen




```java
@Override
public void initialize(URL location, ResourceBundle resources) {
    // TODO Auto-generated method stub
    productColumn.setCellValueFactory(new PropertyValueFactory<Stock, String>("productName"));
    ogStockColumn.setCellValueFactory(new PropertyValueFactory<Stock, String>("ogStock"));
    stockRemainingColumn.setCellValueFactory(new PropertyValueFactory<Stock, String>("stockRemaining"));


    Path productsPath = Paths.get("Files/Products.csv");
    try {
        BufferedReader br = Files.newBufferedReader(productsPath,StandardCharsets.US_ASCII);
        String line = br.readLine();



        while (line != null) {
            String productsName = null; // listo
            int ogStock = 0;
            int RemainingStock = 0;

            String [] characteristicsProducts = line.split(",");
            productsName = characteristicsProducts[0];
            ogStock = Integer.parseInt(characteristicsProducts[3]);
            RemainingStock = ogStock;

            Path pathOrders = Paths.get("Files/Orders.csv");

                BufferedReader brOrder = Files.newBufferedReader(pathOrders,StandardCharsets.US_ASCII);
                String lineOrder = brOrder.readLine();

                while(lineOrder != null) {
                    String [] characteristicsO = lineOrder.split(",");

                    if (productsName.equals(characteristicsO[0])) {

                        RemainingStock = RemainingStock - Integer.parseInt(characteristicsO[4]);

                    }
                    lineOrder = brOrder.readLine();
                }


            Stock stock = new Stock(productsName, String.valueOf(ogStock), String.valueOf(RemainingStock));
            StockTable.getItems().add(stock);
            line = br.readLine();

        }
    }

    catch(IOException e) {
        e.printStackTrace();
    }
    catch(NumberFormatException e) {
        e.printStackTrace();

    }

}
```

StockController.java uses the initializable method, when the screen is opened it reads “Products.csv” and “Orders.csv” to calculate the values necessary and display them in the table view.

Email Screen

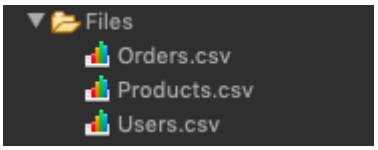


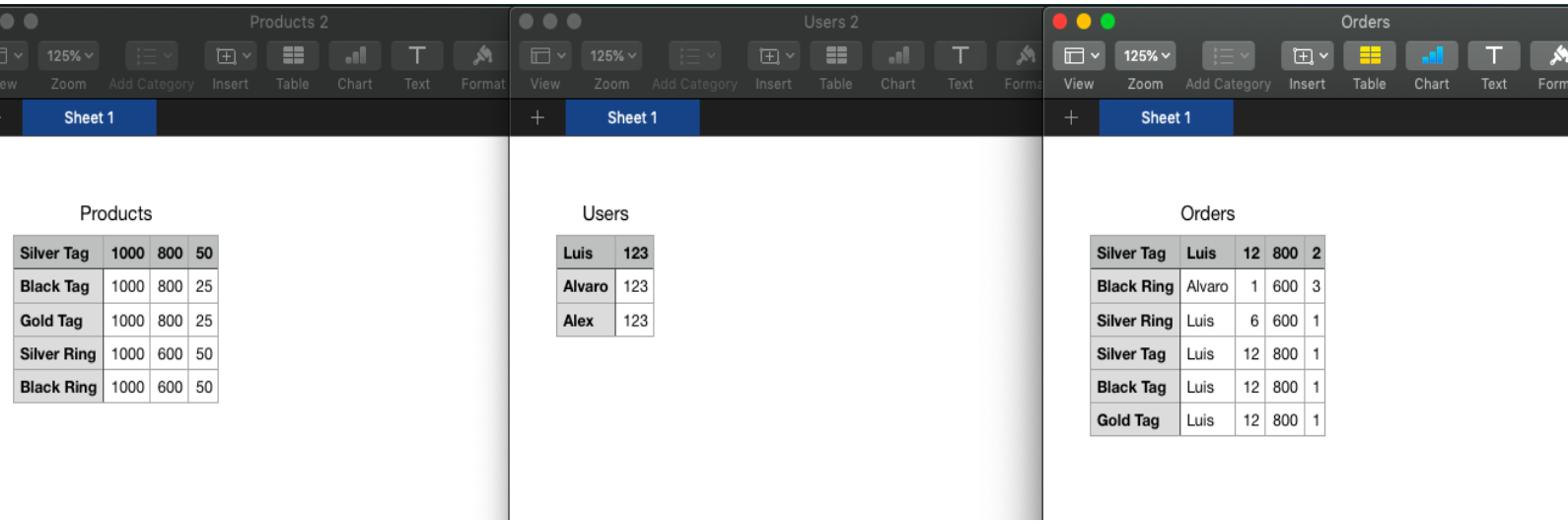

```
@FXML
public void Sending(ActionEvent event) {


    String toEmail = ToEmail.getText();
    String subject = Subject.getText();
    String message = MessageTextArea.getText();

     Properties props = new Properties();
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.host", "smtp.gmail.com");
        props.put("mail.smtp.port", "587");

        Session session = Session.getInstance(props,
                new javax.mail.Authenticator() {
                  protected PasswordAuthentication getPasswordAuthentication() {
                      return new PasswordAuthentication("MVRTESENDER@gmail.com", "MVRTEMAILSENDER1029!");
                  }
                });

               try {

                   Message mail = new MimeMessage(session);
                   mail.setFrom(new InternetAddress("MVRTESENDER@gmail.com"));
                   mail.setRecipients(Message.RecipientType.TO,
                       InternetAddress.parse(toEmail));
                   mail.setSubject(subject);
                   StringBuilder stringBuilder = new StringBuilder();
                   stringBuilder.append(message).append(System.lineSeparator());
                   mail.setText(stringBuilder.toString());
                   Transport.send(mail);
               } catch (MessagingException e) {
                   throw new RuntimeException(e);

               }

               ToEmail.setText("");
               Subject.setText("");
               MessageTextArea.setText("");

               Alert alert = new Alert(AlertType.CONFIRMATION);
               alert.setTitle("Succesful");
               alert.setHeaderText(null);
               alert.setContentText("Your mail has been sent!");
               alert.showAndWait();

}
```

The “Email Sender” screen uses EmailController.java to send emails.

EmailController.java has a method called “Sending” that is called when the send button is clicked. The method uses the javax.mail.jar library to send the emails using a predefined gmail provided by the client Mr.X.

Example of what is saved inside the “Orders.csv”, “Products.csv” and “Users.csv”





Products

Silver Tag	**1000**	**800**	**50**
Black Tag	1000	800	25
Gold Tag	1000	800	25
Silver Ring	1000	600	50
Black Ring	1000	600	50

Users

Luis	**123**
Alvaro	123
Alex	123

Orders

Silver Tag	**Luis**	**12**	**800**	**2**
Black Ring	Alvaro	1	600	3
Silver Ring	Luis	6	600	1
Silver Tag	Luis	12	800	1
Black Tag	Luis	12	800	1
Gold Tag	Luis	12	800	1

Word count: 708