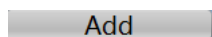


The code is structured in classes which represent each page of the program and a special class with the purpose of connecting the database to the application. Each class consists of functions and variables which make the functionality of the program possible. We chose to use IntelliJ IDEA to continue writing the code after the GUI has been implemented. We imported the project from NetBeans into IntelliJ. We started coding in Java, using the Swing library. Even if we were not familiar with coding a desktop application, we chose to use Java as a programming language because we had some experience with it since last year's project. We did some research on how to get an application development started and then we started implementing our own classes.

## Advisor

Field	Type	Description
Name	String	The full name of the advisor (login token)
ID	Integer	The unique ID
Password	String	Login token
Email	String	Email of the advisor
Contact Number	Integer	Contact number
Address	String	The address of the advisor

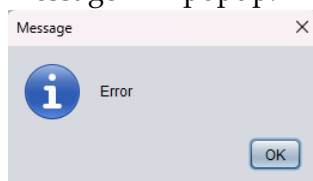
### Add



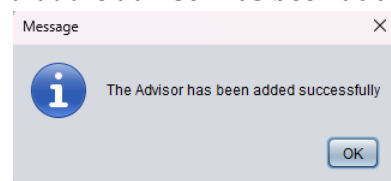
Access: Administrator

To add a new advisor, the system administrator must fill a form in. The form includes the fields above. After introducing the details, the user must press 'Add' in order to complete the action. One of the messages below will appear to inform the user.

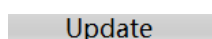
If the details do not correspond to the type required, an error message will popup.



If the data entered is correct, a message will tell the user that the advisor has been added.



### Update



Access: Administrator

## Source Code

To change the details of an advisor, the user must double-click on the field which will be changed, introduce the new details (of the right type), press 'Enter' and then the 'Update' Button

E.g. Table before update

ID	Name	Email	Contact Number	Address
34	advisor	wgr@gmail.com	07535255453	Str VDgs, No 577, ...
43	efev	emial	435454	bvgrbgfr
67	ad	eee	0766	gvfgf
211	Dennis Menace	d.menace@yahoo....	07855442200	14 Church View, W...
250	Travel Advisor	p.pitstop@3hotmai...	02045896358	57 Reservoir Road...
1999	test	email	0984954	fdvd

E.g. Table after update

ID	Name	Email	Contact Number	Address
34	advisor	wgr@gmail.com	07535255453	Str VDgs, No 577, ...
43	efev	emial	435454	bvgrbgfr
67	ad	eee	0766	gvfgf
211	Dennis Menace	d.menace@yahoo....	07855442200	14 Church View, W...
250	Travel Advisor	p.pitstop@3hotmai...	02045896358	57 Reservoir Road...
1999	test-update	email	0984954	fdvd

If the details entered do not have the right type, no change will be made.

## Staff

Field	Type	Description
Name	String	The full name of the advisor (login token)
ID	Integer	The unique ID
Password	String	Login token
User Type	String	The type of the user
Email	String	Email of the advisor
Contact Number	Integer	Contact number
Address	String	The address of the advisor

### Search by type

Select type ▼

Select type

All

Administrator

Travel Advisor

Manager

Access: Administrator

# Source Code

To help the user to search for a specific member of staff, there is a search engine which can be used to only display the staff of the selected type.

E.g. Table before selecting a type

Staff						
ID	Username	Password	UserType	Email	ContactNum...	Address
12	admin	admin	Administrator	lola45@hot...	07454875485	667 StreetNa...
21	test2	1a1dc91c90...	Administrartor	email@admin	0743763746	Address
34	advisor	advisor	Travel Advisor	wgr@gmail.c...	07535255453	Str VDgs, No...
43	efev	afdd21d64b...	Travel Advisor	emial	435454	bvgrbgfr
67	ad	d6e1c05c8a...	Travel Advisor	eee	0766	gvfgf
98	manager	manager	Manager	jhb@email	jhhkj	jhhjh
204	OfficeManag...	Passw1222	Manager	r.tommy@g...	07844968745	3 Wellington ...
211	Dennis Mena...	Gnasher	Travel Advisor	d.menace@y...	07855442200	14 Church Vi...
214	OfficeManag...	Passw1278	Manager	s.mic@hotm...	07588770014	199 Old Bedf...
220	Minnie Minx	NotiGirl	Manager	m.minx@gm...	02045789645	14 Church Vi...
240	admin005	Pass456	Administrator	d.antau@out...	07588459963	Owls Nest, L...

E.g. Table after selecting 'Administrator'

Staff						
ID	Username	Password	UserType	Email	ContactNum...	Address
12	admin	admin	Administrator	lola45@hotm...	07454875485	667 StreetNa...
240	admin005	Pass456	Administrator	d.antau@outl...	07588459963	Owls Nest, L...

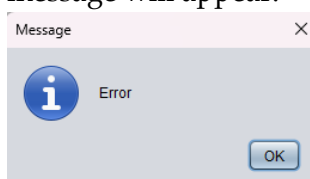
## Add

Add

Access: Administrator

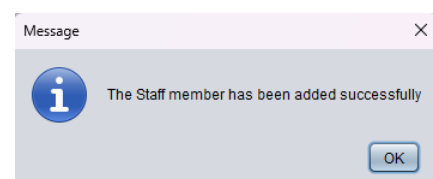
To add a new member of staff, the administrator can press the add button, which will redirect him to a form which must be completed. The data entered for adding a new member of staff must correspond with the types specified in the table above.

If the details entered are not correct, an error message will appear.



## Delete

A message to inform the user if the member has been added successfully.



Delete

Access: Administrator

The user can delete a member of staff by clicking on the row that contains the staff to be deleted. After selecting the row, press 'Delete' and the row will be deleted.

## Update

Update

Access: Administrator

The details of the staff can be updated anytime. The user must double-click on the field to be changed, press 'Enter' and then 'Update' for a successful change.

## Blank Stock

Field	Type	Description
Blank ID	String	The full name of the advisor (login token)
Blank Sold	Integer	If the blank is sold (0-> no, 1->yes)
Staff ID	String	The Id of the advisor to which the blank was assigned
Date Received	Date	The type of the user

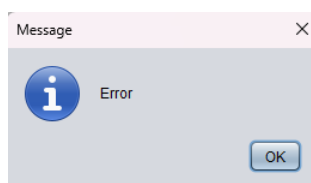
## Add

Add

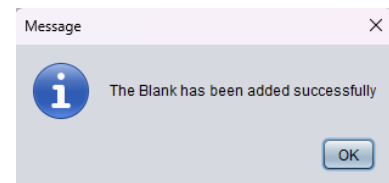
Access: Administrator

To add a new blank, the user must press the 'Add' button. A form to be completed will show up. The user has to input the right type of data in order to have a successful process.

If the process failed.



If the process was successful.



## Delete

Delete

Access: Administrator

The blanks can be deleted by the administrators. To delete, a row from the table must be selected and then press button 'Delete'. The row will be removed, which means that the blank has been deleted. If the blank is sold, it can not be removed.

E.g. Table before Delete

Blank Stock			
Blank ID	Blank Status	Advisor ID	Date
444	0	0	2020-02-20
444000000010	0	250	2019-04-01
444000000015	0	250	2019-04-01
444000000016	2	250	2019-04-01
444000000023	1	211	2019-04-01
444000000038	0	211	2019-04-01
44400949	0	122	2020-01-12
444433535	0	122	2020-10-09

E.g. Table after deleting the first row

Blank Stock			
Blank ID	Blank Status	Advisor ID	Date
444000000010	0	250	2019-04-01
444000000015	0	250	2019-04-01
444000000016	2	250	2019-04-01
444000000023	1	211	2019-04-01
444000000038	0	211	2019-04-01
44400949	0	122	2020-01-12
444433535	0	122	2020-10-09

## Update

Update

Access: Administrator

The details of the Blanks can be changed. Double-click on the field that will be changed, change the data, press 'Enter' and then 'Update'.

## Stock Turnover Report

Access: Administrator

For the Stock Turnover Report, the user must enter the data in  Since  Until .

After filling in the two fields, the user must press 'View' to generate the report.

View

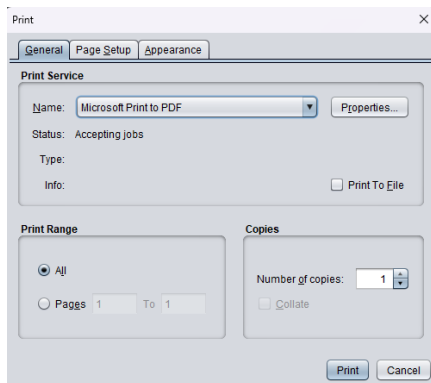
## Source Code

After pressing 'View', the table will be filled in with the data matching the input.

Print

By pressing print, a tab will be displayed. It shows different options for printing.

E.g. Tab for printing



E.g. The output of printing

### Stock Turnover Print

Blank Number	Assigned
10100000000001	0
10100000000020	0
10100000000026	0
10100000000032	0
1010000000004	0
10100000000046	0
10100000000049	0
1010000000005	1
20100000000001	0
20100000000002	1
20100000000005	0
20100000000010	0
42000000000015	1
4200000000002	1
42000000000031	1
42000000000038	1
42000000000046	1
4200000000005	0
44000000000001	1
44000000000006	0
44000000000007	1
44400000000016	0
44400000000015	0
44400000000016	2
44400000000023	0
44400000000038	0
44402949	0
4444333333	0

Page1

Advertiser ID
211
211
211
211
211
211
211
250
250
250
250
250
250
211
211
211
250
45
34
345
380
250
250
211
211
122

Page 2

## Assign to Advisor ( in progress)

Access: Manager

In order to assign a blank to an advisor, the manager must select an advisor by clicking on 'Advisor List', choosing an advisor and pressing select.

Advisor's Name	Dennis Menace
----------------	---------------

When the advisor has been selected, its name will be displayed automatically in the 'Advisor's Name' field. (see above)

Select type
Select type
444
440
420
101
201
451
452

There is a combo box where the type of blank must be selected.

Number of Blanks	
------------------	--

Once the type of blank and the advisor are chosen, in the 'Number of Blanks' will appear the actual amount of blanks, of the specific type, has the advisor.

To add or remove blanks from the advisor, the manager must change the data in the 'Number of blanks field' as desired (a lower or a higher number if available) and press confirm to make the changes.

If there are no more blanks of the select type available to be assigned to the advisor, the manager can press reassign.

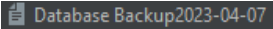
Advisor: X Owns: Y Blanks
---------------------------

In the text field above will appear all the information updated simultaneously.

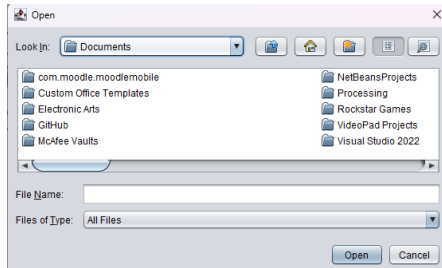
## Backup & Restore (Database)

Access: Administrator

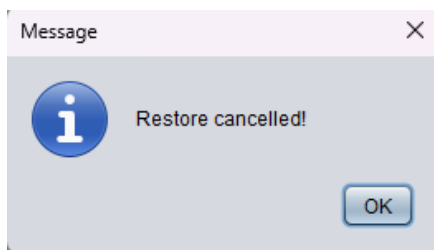
## Source Code

The administrator can create a Backup of the database by pressing 'BackUp'. A file will be created with the name ' DatabaseBackup-current date-'. 

In case the actual data is damaged, the user can restore older versions of the database. After pressing 'Restore', a file chooser will show up where the user can select a backup file from.



If no file was selected by the user, there will be a message saying that the restore has been cancelled.



## Customers

Field	Type	Description
Customer ID	Integer	The Id of the customer (used for relating
Customer Name	String	The full name of the customer
Customer Email	String	Customer's Email
Customer Contact Number	String	The contact number of the customer
Customer Address	String	Customer's address
Payment Information	String	The payment information (card or cash)
Customer Type	String	The type of the customer ( valued or regular)
Discount Type	String	The type of discount ( fixed, flexible)
Discount Rate	String	The discount rate
Payment ID	Integer	The payment id

### Find by

Find by

Find by

Find by ID

Find by Name

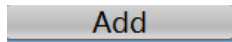
Access: Advisor & Manager



## Source Code

The advisor and the manager can see the information of their customers. To find a specific customer, they can select a 'Find by' option from the combo box. The customers can be identified by id or name, which are introduced in the text box (below 'Find by'). After the name or id has been introduced, the user must press on the 'Search' button (next to the text box). If there is no input, all the customers are displayed.

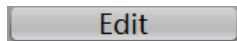
### Add



Access: Advisor

The advisors are able to add new customers. An empty row will be inserted into the table, where the user can introduce the required details. If not all the columns are filled in, the customer will not be added.

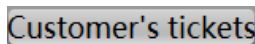
### Edit



Access: Advisor

The advisor can also change the details of the customers if needed. The process is similar as for the other categories (update).

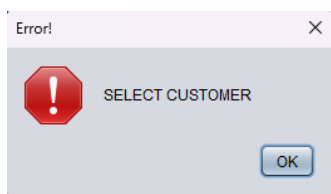
### Customer's Tickets



Access: Advisor & Manager

Both, the advisor and the manager can see what tickets the customers bought by selecting a customer and clicking on the 'Customer's tickets' button.

If no customer was selected before pressing on the 'Customer's tickets' button, an error message will appear.



If a customer has been selected and the 'Customer's tickets' is clicked, a new page with the tables below will appear. In the first tables are the tickets and their details.

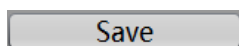
## Source Code

ID	Departure	Destination	Departure Ti...	Arrival Time	Flight Number	Blank Number	Customer ID
185	Rome IT	Zurich SWZ	2023-03-01 1...	2023-03-01 0...	7480	44400000004	606
Blank Number	Exchange Rate	Payment Type	Date	Taxes	Refunded	Name	Delayed

The second table is to see the payment details of a specific ticket. To view them, select a ticket and press on a row in the second table. You can see in the picture below that a row from the first table is selected and the payment details are now available in the other table.

ID	Departure	Destination	Departure Ti...	Arrival Time	Flight Number	Blank Number	Customer ID
185	Rome IT	Zurich SWZ	2023-03-01 1...	2023-03-01 0...	7480	44400000004	606
Blank Number	Exchange Rate	Payment Type	Date	Taxes	Refunded	Name	Delayed
4.440000000...	true	pay later (with...	2023-02-02	23.0	0	Sarah Brokle...	0

### Save



Access: Advisor & Manager

The advisors and managers can change the ticket details by selecting a field, enter the new data, press 'Enter' and then click on 'Save' button.

## Currency Exchange

Access: Advisor

The advisor has access to a page where the current exchanging rates are shown. The exchanging rates are then used when selling a ticket.

Country	Currency Name	Rate to USD
United Arab Emirates	AED	0.2720
Australia	AUD	0.5400
Brazil	BRL	0.1975
Canada	CAD	0.7400
Germany	EUR	1.0910
United Kingdom	GBP	1.2420
China	HKD	0.1270
India	INR	0.0120
Mexico	MXN	0.0550
Russia	RUB	0.0120
USA	USD	1.0000

When selling a ticket, the advisor must select the currency. For example, if the currency selected is BRL, the exchanging rate for BRL to USD will appear below, in the text box, automatically, corresponding to the table.

Exchange Rate:

## Sales (in progress)

Access: Advisor

The advisors can select a type of blank to see specific tickets, can select if the payment will be made at the moment or later, according to the customer's preference (it is set to 'now' as default) and can select the payment method, cash or card. If the selected payment is card, a new tab will be displayed, where the details of the card must be entered.

Select Blank Type

Pay now/later

Payment method

# Source Code

## Available tickets

There is a table, where the ticket to be sold must be selected.

Departure	Destination	Departure Time	Arrival Time	Flight No	Price
London Heathrow-...	Cancun Internation...	2023-03-28 12:00:00	2023-03-29 09:30:00	780.0	1250.00
London Stansted -...	Oslo Torp- TRF	2023-04-01 02:27:00	2023-04-01 06:54:00	475.0	55.99
London Luton-LTN	Wroclaw-WRo	2023-04-02 10:00:00	2023-04-02 12:15:00	732.0	120.00
Bombay-BOM	Wroclaw-WRo	2023-03-08 14:15:...	2023-03-08 15:15:29	701.0	140.98
Bombay-BOM	Delhi-DEL	2023-03-08 14:15:...	2023-03-08 15:15:29	211.0	70.09

## Select Customer

Select Customer

By clicking on 'Select Customer', the page with the customers will show up. The advisor must press on a customer, and it will automatically be selected. If no customer is selected, the sale cannot be successfully made.

## Commission

## Discount

Commission:  ; Discount

There are also fields for 'Commission' and 'Discount' where the advisor must enter the values required. They will be updated in the database for the customer accordingly.

There is also a field for exchanging rate which is explained above in 'Currency Exchange'.

## Save

Save

When pressing 'Save', the ticket will be sold to the selected customer and the data will be updated according to the inputs.

Note: Selling tickets is not fully working yet, it is in progress.

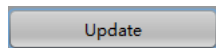
## Flights/ Tickets

Field	Type	Description
Departure	String	The place of departure
Destination	String	The place of arrival
Departure Date Time	Date Time	The departure time
Arrival Date Time	Date Time	The arrival time
Flight Num	Float	The number of the flight
Price	Decimal	The price of the ticket before deductions
Ticket ID	String	Ticket's Id

Access: Advisor

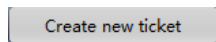
# Source Code

Update:



There is a table which shows the tickets available at that time. The details of the flights can be changed if need. The process is similar as for the other categories.

Create new ticket



The advisor has the possibility of creating a new ticket. For this to happen, the user must click on 'Create new ticket'. Once clicked, a new row will be added to the table, where the advisor can introduce the data for the new flight. To successfully add the ticket, the data must have the correct type and the 'Create new ticket' button must be clicked again.

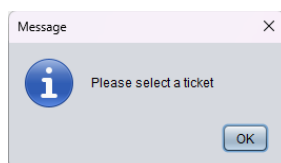
Departure	Destination	Departure Time	Arrival Time	Flight Number	Price	TicketID
London Heath...	Cancun Intern...	2023-03-28 1...	2023-03-29 0...	780.0	1250.00	TIK5876585540
London Stans...	Oslo Torp- TRF	2023-04-01 0...	2023-04-01 0...	475.0	55.99	TIK7485964569
London Luton...	Wroclaw-WRo	2023-04-02 1...	2023-04-02 1...	732.0	120.00	TIK7896585540
Bombay-BOM	Wroclaw-WRo	2023-03-08 1...	2023-03-08 1...	701.0	140.98	TIK7896587898
Bombay-BOM	Delhi-DEL	2023-03-08 1...	2023-03-08 1...	211.0	70.09	TIK7896587899
New ticket						

Departure	Destination	Departure Time	Arrival Time	Flight Number	Price	TicketID
New ticket	Destination	2023-02-10 1...	2020-03-10 1...	1.0	10.00	TIK00000
London Heath...	Cancun Intern...	2023-03-28 1...	2023-03-29 0...	780.0	1250.00	TIK5876585540
London Stans...	Oslo Torp- TRF	2023-04-01 0...	2023-04-01 0...	475.0	55.99	TIK7485964569
London Luton...	Wroclaw-WRo	2023-04-02 1...	2023-04-02 1...	732.0	120.00	TIK7896585540
Bombay-BOM	Wroclaw-WRo	2023-03-08 1...	2023-03-08 1...	701.0	140.98	TIK7896587898
Bombay-BOM	Delhi-DEL	2023-03-08 1...	2023-03-08 1...	211.0	70.09	TIK7896587899

Delete



To delete a ticket, the user must select one and press 'Delete'.



If no ticket has been selected, a message will be sent to inform the user.

E.g. The table after deleting the new created row

Departure	Destination	Departure Time	Arrival Time	Flight Number	Price	TicketID
London Heath...	Cancun Intern...	2023-03-28 1...	2023-03-29 0...	780.0	1250.00	TIK5876585540
London Stans...	Oslo Torp- TRF	2023-04-01 0...	2023-04-01 0...	475.0	55.99	TIK7485964569
London Luton...	Wroclaw-WRo	2023-04-02 1...	2023-04-02 1...	732.0	120.00	TIK7896585540
Bombay-BOM	Wroclaw-WRo	2023-03-08 1...	2023-03-08 1...	701.0	140.98	TIK7896587898
Bombay-BOM	Delhi-DEL	2023-03-08 1...	2023-03-08 1...	211.0	70.09	TIK7896587899

# Source Code

## Refund

Access: Advisor

### Ticket Id and Amount Field

Ticket ID

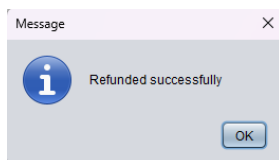
Amount

The advisor can refund a ticket. It is required to fill in the fields 'Ticket ID' and 'Amount'.

### Refund

Refund

After introducing the inputs, the user must press 'Refund'.

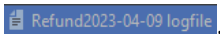


If the details entered are correct, there will be a message to confirm the transaction and the data will be updated in the database (it will set isRefunded, for the specific ticket, to 1 in the Payment table, in the database).

### Record:

Record

If the advisor wants to record this refund, when pressing 'Refund', a text file will open. The advisor can introduce all the information needed in the text file and save it. It will be saved as 'Refund-

*current date-logfile*'. 

## Commission

Access: Manager

### Select type

Select type

Select type

444

440

420

201

101

451

452

To add a commission rate, the manager must select the blank type.

### Commission Rate field

Commission Rate

Once the blank type has been selected from the combo box, the manager must introduce the rate.

Save

To successfully add this rate, the user must press 'Save'. This function will add a new commission rate in the database.

## Alerts

Customer ID	Customer Name	Purchase Date
623	Tom Mac	2023-02-02
456	Chris Smart	2023-01-01

Access: Manager

The manager can see the overdue payments. They appear in the 'Alerts' section. In the table above are shown the customers which are late to pay.

## Reports (in progress)

Access: Manager

;

There are four types of reports, except the Stock Turnover Report, we have Domestic Individual and Global and Interline Individual and Global.

Since  Until

No	AgentID	Sold	Fare	USD	Taxes	Cash Pay	Card Pay	Total Am	Commis

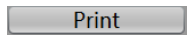
Net Debit  Total Net

View



In order to view data in the table, the manager must fill in the 'Since' and 'Until' fields and press 'View'. Once these steps are done, both tables will be filled in with the corresponding data. In the second table will appear the results of the calculations made using the first table.

Print




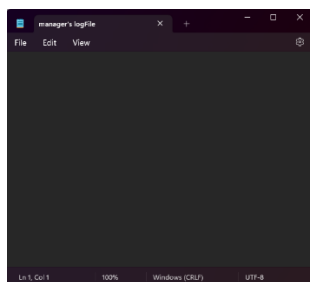
There is the possibility to print the report. Press 'Print' and the printing set-up tab will show up.

## Logs

Access: Manager

There is an option for the managers to open a text file where they can insert the data that they must keep in a logfile. It is saved as 'manager's logFile'.

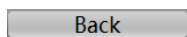
 manager's logFile



## Logout & Back Buttons

Back:

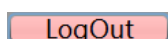
Access: Advisor, Administrator and Manager



There are 'Back' buttons on each page, excluding the menus and the login page. The role of it is to redirect the user to the previous page. We considered that we did not need to include a menu on each page for navigation as the application is not a complex one and by pressing 'back', the users can immediately get to the menu specific to their roles where all the functions are available. A menu along with the other buttons might be confusing for the users.

LogOut:

Access: Advisor, Administrator and Manager





A logout button is available on each of the three menus. By clicking it, the users get back to the login page.

The main actions used are action performed and mouse clicked.

## Database Connection

In DbConnection class we have a function getConnection().

The DBMS used is MySQL. To successfully make the connection, we downloaded a connector library, set up the connection parameters (URL, user and password). The URL consists of the server host address, port number and the table name. We used the user "in2018g10\_d" because this only has DML permission.

```
public static Connection getConnection() throws ClassNotFoundException {  
    Connection connection = null;  
  
    try {  
        Class.forName("com.mysql.cj.jdbc.Driver");  
        connection = DriverManager.getConnection("jdbc:mysql://smcse-stuproj00.city.ac.uk:3306/in2018g10", user: "in2018g10_d", password: "X4XtM3KT");  
        connection.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);  
        return connection;  
    } catch (SQLException e) {  
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, msg: null, e);  
    }  
  
    return null;  
}
```

To test the connection, we created the 'Login' class and connected it to the 'login' table.

We created two variables, username and password, and attributed values to them.

```
String username = userField.getText();  
char[] password = passField.getPassword();
```

Then we added the connection to the database

```
try(Connection connection = DbConnection.getConnection();){...} catch (ClassNotFoundException | SQLException e){  
    Logger.getLogger(Login.class.getName()).log(Level.SEVERE, msg: null, e);  
}
```

We created a prepared statement to execute an SQL query. We added an SQL statement to check if the username and the password introduced are registered in the database. The query is executed using the executeQuery() method, which returns a ResultSet containing the results of the query. If the ResultSet contains at least one row, the user's type is retrieved from the 'UserType' column in the 'login' table. Depending on the user's type, the user is redirected to the appropriate page such as Administrator Menu, Travel Advisor Menu and Manager Menu.

```

PreparedStatement preparedStatement = connection.prepareStatement( sql: "select UserType from login where Password =? and Username =? ");
preparedStatement.setString( parameterIndex: 1, String.valueOf(password));
preparedStatement.setString( parameterIndex: 2, username);

ResultSet resultSet = preparedStatement.executeQuery();

if(resultSet.next()){
    //depending on the user type, the user will be redirected to the right page
    type = resultSet.getString( columnLabel: "UserType");
    switch (resultSet.getString( columnLabel: "UserType")){
        case "Administrator":
            SystemAdminMenu systemAdminMenu1 = new SystemAdminMenu();
            systemAdminMenu1.setVisible(true);
            systemAdminMenu1.setDefaultCloseOperation(SystemAdminMenu.DISPOSE_ON_CLOSE);
            this.dispose();
            System.out.println( "logged");
            break;
        case "Travel Advisor":
            AdvisorMenu advisorMenu = new AdvisorMenu();
            advisorMenu.setVisible(true);
            this.dispose();
            advisorMenu.setDefaultCloseOperation(AdvisorMenu.DISPOSE_ON_CLOSE);
            break;
        case "Manager":
            OffManagerMenu offManagerMenu = new OffManagerMenu();
            offManagerMenu.setVisible(true);
            this.dispose();
            offManagerMenu.setDefaultCloseOperation(OffManagerMenu.DISPOSE_ON_CLOSE);
            break;
        default:
            break;
    }
}
}

```

The prepared statements in this code, improve the performance and security of the database operations by treating the query parameters as values, rather than being part of the SQL statement.

The connection with the database was successful so we moved on to coding the next pages.

As the application involves a lot of stored data, we had to connect most of the functions to the database.

### Connecting tables to database

We decided that the use of tables in the pages might help the user's navigability as the information is organized clearly. Below is an example of populating a table with the data from the database.

In this example, the query does not have parameters. We used `ResultSetMetaData` to retrieve the number of columns in the `ResultSet`, which is used to create a `DefaultTableModel` for the `JTable1`. The while loop is iterating over the `ResultSet`, getting all the values of each column for each row and adds them to an array. The array is then introduced in the `tableModel` by the `addRow()` method which adds a new row in the table with the values in the array.

```
private void initTicketTable() {

    try (Connection connection = DbConnection.getConnection()) {
        PreparedStatement preparedStatement = connection.prepareStatement("select * from Tickets");
        ResultSet resultSet = preparedStatement.executeQuery();
        ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

        int columnCount = resultSetMetaData.getColumnCount();
        tableModel = (DefaultTableModel) jTable1.getModel();
        tableModel.setRowCount(0);

        while (resultSet.next()) {
            Object[] a = new Object[7];
            for (int i = 0; i < columnCount; i++) {
                a[0] = resultSet.getString(columnLabel: "Departure");
                a[1] = resultSet.getString(columnLabel: "Destination");
                a[2] = resultSet.getString(columnLabel: "DepartureDateTime");
                a[3] = resultSet.getString(columnLabel: "ArrivalDateTime");
                a[4] = resultSet.getString(columnLabel: "FlightNumber");
                a[5] = resultSet.getString(columnLabel: "Price");
                a[6] = resultSet.getString(columnLabel: "TicketID");
            }
            tableModel.addRow(a);
        }
    } catch (ClassNotFoundException | SQLException e) {
        Logger.getLogger(TicketBooking.class.getName()).log(Level.SEVERE, msg: null, e);
    }
}
```

## Delete

This is a function example for deleting. We use a 'WHERE' clause with parameters to delete a row with the specific value of 'BlankID' and where the 'BlankSold' equals 0. This means that in this case, we cannot delete the blanks that have been sold.

We set the value of 'BlankID' using the setString() method of the prepared statement. The query will be executed with the specific value for 'BlankID', that was selected by the user.

The executeUpdate() is to execute the query and delete the row from the database. The initBlankStock() method is called to refresh the table with the updated rows.

If no row is selected, a message will be displayed to inform the user.

```
private void deleteButton1ActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_deleteButton1ActionPerformed

    try (Connection connection = DbConnection.getConnection()) {
        selectedRow = jTable1.getSelectedRow();
        if (selectedRow >= 0) {
            PreparedStatement preparedStatement = connection.prepareStatement("DELETE FROM BlankStock WHERE BlankID = ? AND BlankSold = 0");
            preparedStatement.setString(parameterIndex: 1, tableModel.getValueAt(selectedRow, column: 0).toString());
            preparedStatement.executeUpdate();
            initBlankStock(sqlStatement: "SELECT * FROM BlankStock");
        } else {
            JOptionPane.showMessageDialog(parentComponent: null, message: "Please select a row to be deleted!");
        }
    } catch (ClassNotFoundException | SQLException e) {
        Logger.getLogger(BlankStock.class.getName()).log(Level.SEVERE, msg: null, e);
    }
}

} //GEN-LAST:event_deleteButton1ActionPerformed
```

## Add

We used two different methods for the Add function.

1. Redirecting the user to a form that allows the user to add a new blank. This design provides a user-friendly way to allow the user perform different actions on the data, without overwhelming them with too many options on one screen. Moreover, this

approach improves the maintainability of the code by keeping the functionality of the forms separate.

```
private void addButtonActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_addButtonActionPerformed
    BlankADD blankADD = new BlankADD();
    blankADD.setVisible(true);
    dispose();
}GEN-LAST:event_addButtonActionPerformed
```

When 'Update' button from the adding blank form is clicked, the data is taken from the inputs(idInput, statusInput, advInput and dateInput) and is inserted into the 'BlankStock' table using the prepared statement. This SQL Statement is easier to implement, but it makes SQL injection attacks possible because the data is taken directly from the text fields, compared to the previous SQL Statement where the data is parameterized.

```
private void updateButtonActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_updateButtonActionPerformed
    //connection with the db
    try (Connection connection = DbConnection.getConnection()){
        //inserting the values in the table, correspondingly
        PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO BlankStock(BlankID, BlankSold, StaffID, DateReceived) Values (?,?,?,?)");

        preparedStatement.setString(1, idInput.getText());
        preparedStatement.setString(2, statusInput.getText());
        preparedStatement.setString(3, advIDInput.getText());
        preparedStatement.setString(4, dateInput.getText());

        //once the statements are executed successfully, it shows a message
        preparedStatement.execute();

        button = true;

        JOptionPane.showMessageDialog(parentComponent: null, message: "The Blank has been added successfully");

        dispose();
        BlankStock blankStock = new BlankStock();
        blankStock.setVisible(true);
    } catch (Exception e){
        JOptionPane.showMessageDialog(parentComponent: null, message: "Error");
    }
}GEN-LAST:event_updateButtonActionPerformed
```

## 2. Adding an empty row in the table

After connecting to the database, we count the number of rows in the 'Tickets' table by executing a SQL Statement. The Result Set is used to store the count of the 'FlightNumber' column in the result variable, while the 'rowCount' is used to store the number of rows in the table. If the 'rowCount' is equal to or less than the 'result', a new row is added. If it is greater, it means that there is an existing row in the 'jTable1', which has not been added in the 'Tickets' table in the database so it will insert it in the database table by executing the SQL Statement and 'jTable1' is updated by calling the 'initTicketTable()'.

```
try (Connection connection = DbConnection.getConnection()){

    PreparedStatement preparedStatement = connection.prepareStatement( sql: "select count(FlightNumber) from Tickets");
    ResultSet resultSet = preparedStatement.executeQuery();
    resultSet.next();
    int result = resultSet.getInt( columnLabel: "count(FlightNumber)");
    int rowCount = jTable1.getRowCount();

    //inserting the values for the new row
    if(rowCount > result){
        rowCount--;

        preparedStatement = connection.prepareStatement( sql: "INSERT INTO Tickets (\n"
            + "    Departure,\n"
            + "    Destination,\n"
            + "    DepartureDateTime,\n"
            + "    ArrivalDateTime,\n"
            + "    FlightNumber,\n"
            + "    Price,\n"
            + "    TicketID\n"
            + ") \n"
            + "VALUES (\n"
            + "    '" + tableModel.getValueAt(rowCount, column: 0) + "',\n"
            + "    '" + tableModel.getValueAt(rowCount, column: 1) + "',\n"
            + "    '" + tableModel.getValueAt(rowCount, column: 2) + "',\n"
            + "    '" + tableModel.getValueAt(rowCount, column: 3) + "',\n"
            + "    '" + tableModel.getValueAt(rowCount, column: 4) + "',\n"
            + "    '" + tableModel.getValueAt(rowCount, column: 5) + "',\n"
            + "    '" + tableModel.getValueAt(rowCount, column: 6) + "'\n"
            + ");");
        preparedStatement.execute();
        intiTicketTable();
    }
    else {
        tableModel.addRow(new Object[7]);
    }
}
catch (ClassNotFoundException | SQLException e){
    Logger.getLogger(TicketBooking.class.getName()).log(Level.SEVERE, msg: null, e);
}
}
```

## Update

After connecting to the database using the 'DbConnection.getConnection()', the selected row is stored in the 'selectedRow'. If no row is selected, a message will show up to inform the user to select a row. Otherwise, an SQL UPDATE Statement is used to update the data in the 'BlankStock' table (in the database), corresponding to the selected row. The data is retrieved from the 'JTable1' using the 'getValueAt()' method. The 'BlankID' is used to identify the row to be updated.

```
private void updateButton1ActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_updateButton1ActionPerformed

    try(Connection connection = DbConnection.getConnection()) {
        PreparedStatement preparedStatement = null;
        selectedRow = jTable1.getSelectedRow();

        if(selectedRow >=0){
            preparedStatement = connection.prepareStatement( sql: "update BlankStock set BlankSold = '"
                + tableModel.getValueAt(selectedRow, column: 1)
                + "', StaffID= '"
                + tableModel.getValueAt(selectedRow, column: 2)
                + "', DateReceived= '"
                + tableModel.getValueAt(selectedRow, column: 3)
                + "' where BlankID = '"
                + tableModel.getValueAt(selectedRow, column: 0) + "'");
            preparedStatement.execute();
            initBlankStock( sqlStatement: "select * from BlankStock");
        }
        else {
            JOptionPane.showMessageDialog( parentComponent: null, message: "Please select a row!");
        }
    }
    catch (SQLException | ClassNotFoundException e){
        JOptionPane.showMessageDialog( parentComponent: null, message: "Error");
    }
}
//GEN-LAST:event_updateButton1ActionPerformed
```

### Special feature – encryption

We decided to add encryption for the password. We used MD5 encryption as it is quick to implement. It provides a basic level of security for the user's passwords. If a hacker gains access to the database, it will not be able to see the original passwords. The MD5 encryption produced a fixed-length, unique output, which is stored in the database

```
//getting the password input
//we store it in a char for encryption
char[] password = passInput.getPassword();

//encrypt the password with md5 encryption
String encryption = "";
for( char e : password){
    encryption += e;
}

String md5EncryptedInput = Login.encrypt(encryption);

//connection with the db
try (Connection connection = DbConnection.getConnection()){

    //inserting the values in the table, correspondingly
    PreparedStatement preparedStatement = connection.prepareStatement("insert into login(Username, ID, Password, UserType, Email, ContactNumber, Address) Values (?, ?, ?, ?, ?, ?, ?)");

    preparedStatement.setString( parameterIndex: 1, nameInput.getText());
    preparedStatement.setString( parameterIndex: 2, idInput.getText());
    preparedStatement.setString( parameterIndex: 3, md5EncryptedInput);
    preparedStatement.setString( parameterIndex: 4, typeInput.getText());
    preparedStatement.setString( parameterIndex: 5, emailInput.getText());
    preparedStatement.setString( parameterIndex: 6, numberInput.getText());
    preparedStatement.setString( parameterIndex: 7, addressInput.getText());
    preparedStatement.execute();
}
```

The 'encrypt' method takes the password entered by the user, as a string, and uses the MessageDigest class from the Java security package to generate a one-way hash of the password using the MD5 algorithm. The hash is calculated when calling the method 'digest'. The resulting hash is converted to a string of hexadecimal digits using StringBuffer. The method returns the resulting string which is the encrypted password. It is not possible to reverse the process and recover the original password.

```
public static final String encrypt(String md5) {
    try {
        java.security.MessageDigest md = java.security.MessageDigest.getInstance("MD5");
        byte[] array = md.digest(md5.getBytes());
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < array.length; i++) {
            sb.append(Integer.toHexString( (array[i] & 0xFF | 0x100).substring(1, 3)));
        }
        return sb.toString();
    } catch (NoSuchAlgorithmException ex) {
        Logger.getLogger(Login.class.getName()).log(Level.SEVERE, msg: null, ex);
    }
}

return null;
}
```

During the development of the application, we encountered a few errors. We managed to get the solutions for most of them. The steps we made to find the solutions were to read the error, find the context, identify the row with issues, check for typos, syntax errors or other mistakes. Afterwards, we would correct the small issues and think of a solution that would solve the errors. Finally, we would test it. If it would not work, we would go through the steps again, until the error would disappear.

## Runtime components

Below is an analysis of the runtime components:

