

Section C—Product Development

Inheritance	Inheritance is a process in which a sub-class inherits methods from a parent class. Almost every class in the system is a sub-class of the javax.Swing.JFrame parent class, whose methods allow the development of GUI.
Database Connection	The front-end of the system was connected to a MySQL database with 5 tables, and connection between the front-end and back-end allowed the transfer and storage of data.
MySQL Queries	MySQL Query Language was used to perform commands on the data in the travel database. These included the retrieval, updation and deletion of data.
Error Check and Validation	Errors were eliminated by the utilisation of a try-catch block for many parts of the program, which generated the stack trace for any errors in the program. Additionally, validation was used to confirm that inputs were being correctly made by the user.
Abstract Data Structure – Linked List	Linked List was used to store data of daily expenses. Since linked lists are dynamic data structures, they don't have a fixed size. This allowed the user to input daily expenses without a specific limitation.
Encapsulation	Encapsulation is the process of wrapping blocks of code. It was used to neatly display large amounts of generated code, creating greater functionality.
Google Mail Integration	An email verification system was implemented to secure the creation of the user's account. The Google Mail API was accessed to perform this task.
External application communication	The program interacted with external applications on the system, including the maps and camera system, creating additional functionality for the user.
Graphical User Interface	GUI components were designed using the Netbeans IDE, which allowed greater interactivity and simplicity for the user.
Java Libraries	Java Libraries were used to import important methods for GUI generation and the creation of tables in the system.

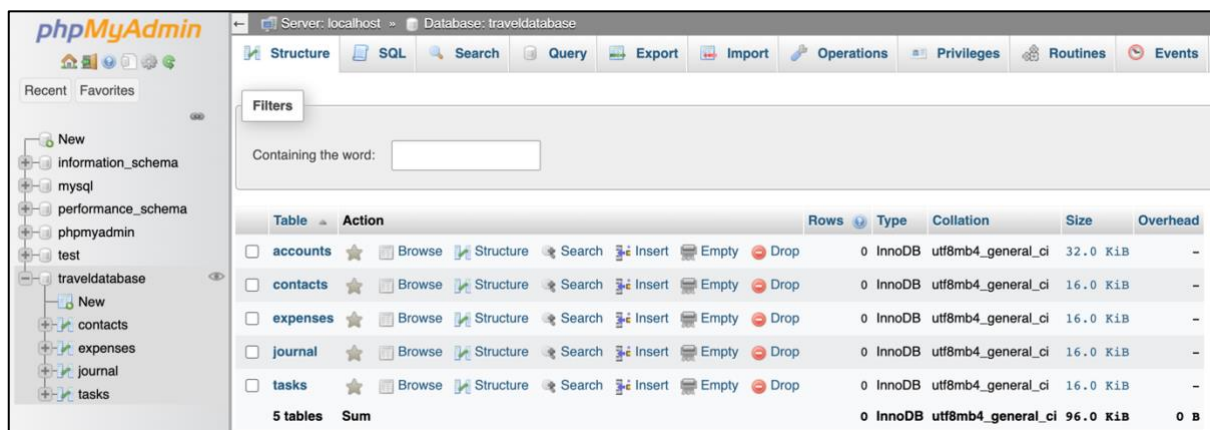
Inheritance

In object-oriented programming, inheritance is the process in which a sub-class inherits the methods from a parent class. In order to create a Graphical User interface for the user to interact with, the methods from the class `javax.Swing.JFrame` were inherited.

```
public class VerificationCodePage extends javax.swing.JFrame {
```

Database Connection

A MySQL database was connected to the front-end of the system to allow the transfer, updation, and deletion of data. phpMyAdmin was used to create the data tables in the database.



The screenshot shows the phpMyAdmin web interface. The left sidebar displays a tree view of databases, with 'traveldatabase' selected. The main panel shows the 'Structure' tab for the 'traveldatabase' database. It lists five tables: accounts, contacts, expenses, journal, and tasks. Each table has a star icon, a 'Browse' button, and links for 'Structure', 'Search', 'Insert', 'Empty', and 'Drop'. The table details show they are all InnoDB tables with utf8mb4_general_ci collation and a size of 16.0 KiB. A summary row at the bottom indicates 5 tables with a total size of 96.0 KiB.

Table	Action	Rows	Type	Collation	Size	Overhead
accounts	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 KiB	-
contacts	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	16.0 KiB	-
expenses	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	16.0 KiB	-
journal	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	16.0 KiB	-
tasks	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	16.0 KiB	-
5 tables	Sum	0	InnoDB	utf8mb4_general_ci	96.0 KiB	0 B

```
package travel_management_system;
import java.sql.*;

public class DBConnection {
    static final String DB_URL = "jdbc:mysql://localhost:3306/traveldatabase";
    static final String USER = "root";
    static final String PASS = "";

    public static Connection connectDB()
    {
        Connection conn = null;
        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");
            conn = DriverManager.getConnection(DB_URL, USER, PASS);
            return conn;
        }
        catch(Exception ex)
        {
            System.out.println("There were errors connecting to the database.");
            return null;
        }
    }
}
```

MySQL Queries

MySQL queries were used to execute commands on the data in the database. 4 types of statements were used: SELECT (to select data from the database), INSERT (to insert data into the database), UPDATE (to update existing data in the database), and TRUNCATE (to delete the contents of a data table at once).

```
String date = jTextField1.getText();
String journal = jTextField2.getText();
Connection dbconn = DBConnection.connectDB();
if (dbconn != null){
    try {
        PreparedStatement st = (PreparedStatement) dbconn.prepareStatement("Select * from accounts");
        ResultSet res = st.executeQuery();
        while(res.next())
        {
            id = res.getInt("id");
        }
        PreparedStatement st1 = (PreparedStatement) dbconn.prepareStatement("INSERT INTO journal (id, date, journal) VALUES (?, ?, ?)");
        st1.setInt(1, id);
        st1.setString(2, date);
        st1.setString(3, journal);
        st1.executeUpdate();
    }
    catch (SQLException ex) {
        Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);
    }
}
else{
    System.out.println("Database connection not available.");
}
```

```
if (dbconn != null){
    try {
        PreparedStatement st = (PreparedStatement) dbconn.prepareStatement("UPDATE accounts SET password =" + newpassword2);
        st.executeUpdate();
    }
    catch (SQLException ex) {
        Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

```
if (tblModel.getRowCount() == 0)
{
    PreparedStatement st2 = (PreparedStatement) dbconn.prepareStatement("TRUNCATE TABLE contacts");
    st2.execute();
}
```

Error Check and Validation

Error debugging was effectively done using a try-catch block. The catch block generated the stack trace was any error in the program, allowing it to be more easily debugged. Validation was used to ensure that login fields were not empty and that passwords entered twice matched.

```
try {
    PreparedStatement st = (PreparedStatement) dbconn.prepareStatement("UPDATE accounts SET password =" + newpassword2);
    st.executeUpdate();
}
catch (SQLException ex) {
    Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);
}
```

```
String newpassword1 = String.valueOf(jPasswordField1.getPassword());
String newpassword2 = String.valueOf(jPasswordField2.getPassword());
if (newpassword1.isEmpty() || newpassword2.isEmpty())
{
    JOptionPane.showMessageDialog(this, "Please enter all required fields.", "Error", JOptionPane.ERROR_MESSAGE);
}
else{
if (!newpassword1.equals(newpassword2))
{
    JOptionPane.showMessageDialog(this, "Passwords do not match.", "Error", JOptionPane.ERROR_MESSAGE);
}
}
```

Abstract Data Structure – Linked List

A linked list was used to store the daily expenses inputted by the user. Because a linked list is a dynamic data structure, it was used instead of an array to allow the user to input their expenses for an arbitrary amount of days without a size restriction.

```
package travel_management_system;

public class node {
    double num;
    node next;

    node()
    {
        next = null;
    }

    node(double num, node next)
    {
        this.num = num;
        this.next = next;
    }
}
```

```
public class LinkedList {
    node head = new node();

    void add(double num)
    {
        node temp = new node(num, null);
        if (head == null)
        {
            head = temp;
        }
        else
        {
            temp.next = head;
            head = temp;
        }
    }

    void delete()
    {
        if (head != null)
        {
            head = head.next;
        }
        else
        {
            System.out.println("The linked list is empty.");
        }
    }

    String print()
    {
        node temp = head;
        String list = "";
        while (temp != null)
        {
            System.out.println(temp.num);
            list = list + temp.num + ", ";
            temp = temp.next;
        }
        list = list.substring(0, list.length() - 2);
        list = "( " + list + " )";
        return list;
    }
}
```

```

LinkedList dailyExpenses = new LinkedList();
double budget;
String dailyExpense;
budget = Double.parseDouble(jTextField1.getText());
dailyExpense = jTextField2.getText();
Connection dbconn = DBConnection.connectDB();
int day_count = 0;
dailyExpenses.add(Double.parseDouble(dailyExpense));
day_count = day_count + 1;
total_expenses = total_expenses + Double.parseDouble(dailyExpense);
if (flag)
{
    total_expenses = Double.parseDouble(dailyExpense);
    for (int i = 0; i < day_count; i++)
    {
        dailyExpenses.delete();
    }
    flag = false;
}

```

Encapsulation

Encapsulation is the process of wrapping blocks of code for greater functionality, readability and accessibility. Many chunks of self-generated code for GUI components were encapsulated in the program.

```

@SuppressWarnings("unchecked")
Generated Code

```

```

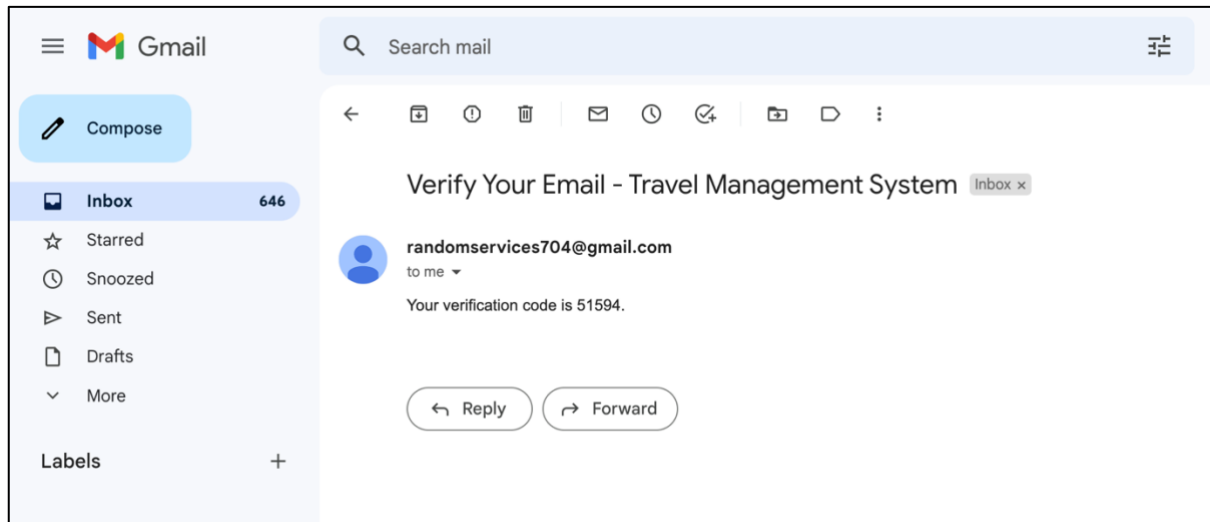
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new InputExpenses().setVisible(true);
        }
    });
}

```

Google Mail Integration

A mail verification email was sent with a verification code when an account was first created. This created secure access to the system and allowed the user to confirm the account was theirs.



```

public class SendEmail {
    final String senderEmail = "randomservices704@gmail.com";
    final String senderPassword = "dwfvvetwdzozjzmqdm"; //email password
    final String emailSMTPserver = "smtp.gmail.com";
    final String emailServerPort = "465";
    String receiverEmail = null;
    static String emailSubject;
    static String emailBody;
    public SendEmail(String receiverEmail, String subject, String body) {
        //receiver email
        this.receiverEmail = receiverEmail;
        //subject
        this.emailSubject = subject;
        //body
        this.emailBody = body;

        Properties props = new Properties();
        props.put("mail.smtp.user", senderEmail);
        props.put("mail.smtp.host", emailSMTPserver);
        props.put("mail.smtp.port", emailServerPort);
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.socketFactory.port", emailServerPort);
        props.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
        props.put("mail.smtp.socketFactory.fallback", "false");
        SecurityManager security = System.getSecurityManager();

        try {
            Authenticator auth = new SMTPAuthenticator();
            Session session = Session.getInstance(props, auth);
            MimeMessage msg = new MimeMessage(session);
            msg.setText(emailBody);
            //System.out.println(emailBody);
            msg.setSubject(emailSubject);
            msg.setFrom(new InternetAddress(senderEmail));
            msg.addRecipient(Message.RecipientType.TO, new InternetAddress(receiverEmail));
            Transport.send(msg);
            System.out.println("Message sent successfully");
        }

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

```

External application communication

The program allowed the execution of command lines to open other applications including “Maps” and “Photo Booth”, used to click an image. This added additional functionality to the program and allowed easier management of relevant local applications.

```

try {
    // TODO add your handling code here:
    Runtime.getRuntime().exec("open -a Maps");
} catch (IOException ex) {
    Logger.getLogger(HomePage.class.getName()).log(Level.SEVERE, null, ex);
}

```

Graphical User Interface

GUI components were used in the application to build interactivity. The success criteria below shows samples of the GUI.

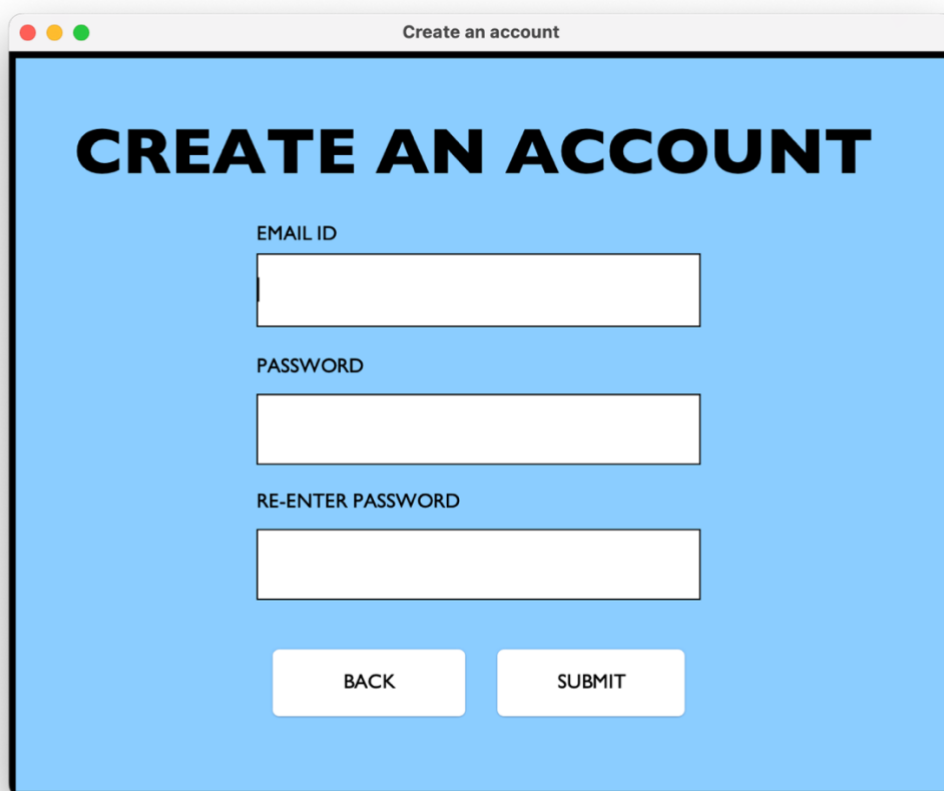
Java Libraries

Java libraries were imported into the program to enable the usage of various options such as database connection, displaying the error stack trace in logs, and generating MySQL queries.

```
import javax.swing.JOptionPane;  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.logging.Level;  
import java.util.logging.Logger;
```

Success Criteria – 1

- a. Allow the user to create an account with their email and a secure password



CREATE AN ACCOUNT

EMAIL ID

PASSWORD

RE-ENTER PASSWORD

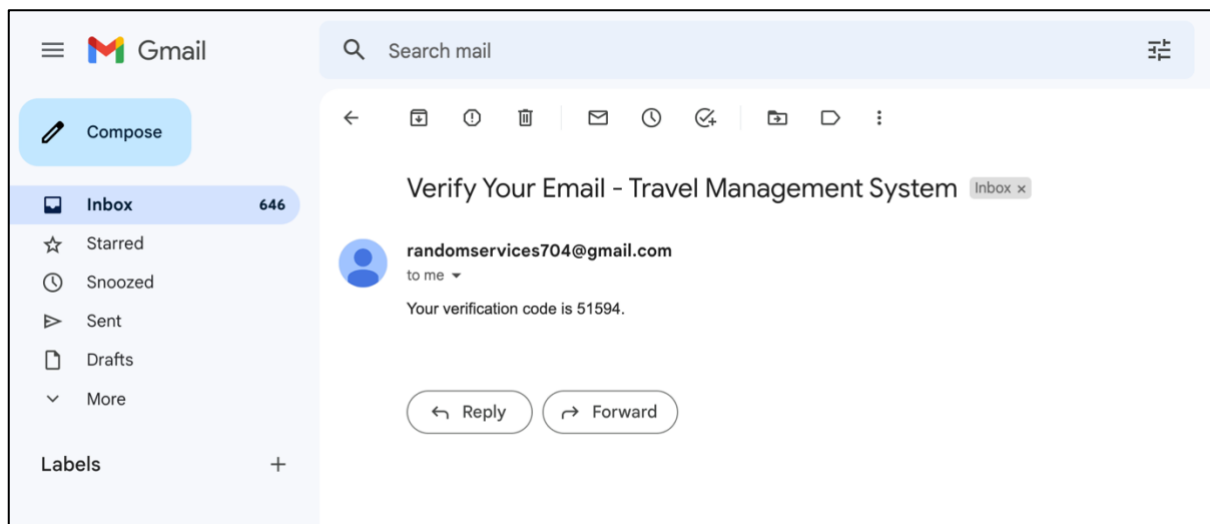
BACK SUBMIT

```

Connection dbconn = DBConnection.connectDB();
String email = jTextField2.getText();
String password1 = String.valueOf(jPasswordField3.getPassword());
String password2 = String.valueOf(jPasswordField2.getPassword());
if (dbconn != null){
    try {
        PreparedStatement st = (PreparedStatement) dbconn.prepareStatement("Select * from accounts");
        ResultSet res = st.executeQuery();
        while(res.next())
        {
            String mail = res.getString("username");
            if(email.equals(mail))
            {
                JOptionPane.showMessageDialog(this, "The account with the entered email already exists. Please log in instead.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    } catch (SQLException ex) {
        Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);
    }
}
else{
    System.out.println("Database connection not available.");
}
if (email.isEmpty() || password1.isEmpty() || password2.isEmpty())
{
    JOptionPane.showMessageDialog(this, "Please enter all required fields.", "Error", JOptionPane.ERROR_MESSAGE);
}
else{
    if (!password1.equals(password2))
    {
        JOptionPane.showMessageDialog(this, "Passwords do not match.", "Error", JOptionPane.ERROR_MESSAGE);
    }
}

```

b. Send the user a verification email upon account creation



Verify your email

VERIFY YOUR EMAIL

PLEASE ENTER THE VERIFICATION CODE SENT TO YOUR EMAIL:

BACK

SUBMIT

```
dispose();
VerificationCodePage verificationcodepage = new VerificationCodePage();
verificationcodepage.setTitle("Verify your email");
verificationcodepage.setVisible(true);
int randomNum = ThreadLocalRandom.current().nextInt(10000, 100001);
SendEmail send = new SendEmail(email, "Verify Your Email - Travel Management System", "Your verification code is " + randomNum + ".");
if (dbconn != null){
try {
    PreparedStatement st = (PreparedStatement) dbconn.prepareStatement("INSERT INTO accounts (id, username, password) VALUES (" + randomNum + ", \"\" + email
    st.executeUpdate();
    PreparedStatement st1 = (PreparedStatement) dbconn.prepareStatement("INSERT INTO expenses (id) VALUES (" + randomNum + ")");
    st1.executeUpdate();
}
}
catch (SQLException ex) {
    Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);
}
```

```

public class SendEmail {
    final String senderEmail = "randomservices704@gmail.com";
    final String senderPassword = "dwfvewtdzojzmqdm"; //email password
    final String emailSMTPserver = "smtp.gmail.com";
    final String emailServerPort = "465";
    String receiverEmail = null;
    static String emailSubject;
    static String emailBody;
    public SendEmail(String receiverEmail, String subject, String body) {
        //receiver email
        this.receiverEmail = receiverEmail;
        //subject
        this.emailSubject = subject;
        //body
        this.emailBody = body;

        Properties props = new Properties();
        props.put("mail.smtp.user", senderEmail);
        props.put("mail.smtp.host", emailSMTPserver);
        props.put("mail.smtp.port", emailServerPort);
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.socketFactory.port", emailServerPort);
        props.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
        props.put("mail.smtp.socketFactory.fallback", "false");
        SecurityManager security = System.getSecurityManager();

        try {
            Authenticator auth = new SMTPAuthenticator();
            Session session = Session.getInstance(props, auth);
            MimeMessage msg = new MimeMessage(session);
            msg.setText(emailBody);
            //System.out.println(emailBody);
            msg.setSubject(emailSubject);
            msg.setFrom(new InternetAddress(senderEmail));
            msg.addRecipient(Message.RecipientType.TO, new InternetAddress(receiverEmail));
            Transport.send(msg);
            System.out.println("Message sent successfully");
        }

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

```


id	username	password
33043	simply.mj21@gmail.com	1234

- c. Allow the user to change their password

```
Connection dbconn = DBConnection.connectDB();
String newpassword1 = String.valueOf(jPasswordField1.getPassword());
String newpassword2 = String.valueOf(jPasswordField2.getPassword());
if (newpassword1.isEmpty() || newpassword2.isEmpty())
{
    JOptionPane.showMessageDialog(this, "Please enter all required fields.", "Error", JOptionPane.ERROR_MESSAGE);
}
else{
    if (!newpassword1.equals(newpassword2))
    {
        JOptionPane.showMessageDialog(this, "Passwords do not match.", "Error", JOptionPane.ERROR_MESSAGE);
    }
    else
    {
        if (dbconn != null){
            try {
                PreparedStatement st = (PreparedStatement) dbconn.prepareStatement("UPDATE accounts SET password =" + newpassword2);
                st.executeUpdate();
            }
            catch (SQLException ex) {
                Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        else{
            System.out.println("Database connection not available.");
        }
        JOptionPane.showMessageDialog(this, "Your password has been successfully updated.", "Success", JOptionPane.INFORMATION_MESSAGE);
        dispose();
        HomePage homepage = new HomePage();
        homepage.setTitle("Home");
        homepage.setVisible(true);
    }
}
```

Success Criteria – 2

- a. Allow the user to login to their account if the username and password entered is correct



```

Connection dbconn = DBConnection.connectDB();
if (dbconn != null){
try {
    PreparedStatement st = (PreparedStatement) dbconn.prepareStatement("Select * from accounts WHERE username = ? AND password = ?");
    st.setString(1, username);
    st.setString(2, password);
    ResultSet res = st.executeQuery();
    if (res.next())
    {
        dispose();
        HomePage home = new HomePage();
        home.setTitle("Home");
        home.setVisible(true);
    }
    else{
        JOptionPane.showMessageDialog(this, "Username and password are incorrect.", "Error", JOptionPane.ERROR_MESSAGE);
    }
} catch (SQLException ex) {
    Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);
}}
else{System.out.println("Database connection not available.");}

```

Success Criteria – 3

- a. Allow the user to input their total travel budget and daily expenses

```

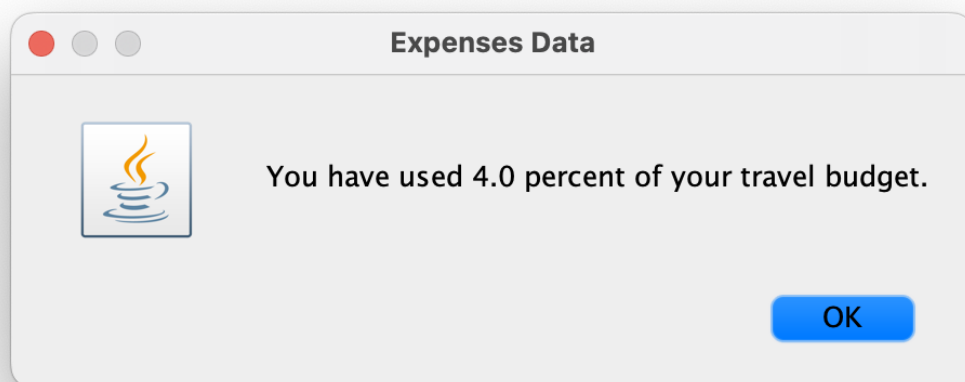
int id = 0;
LinkedList dailyExpenses = new LinkedList();
double budget;
String dailyExpense;
budget = Double.parseDouble(jTextField1.getText());
dailyExpense = jTextField2.getText();
Connection dbconn = DBConnection.connectDB();
int day_count = 0;
dailyExpenses.add(Double.parseDouble(dailyExpense));
day_count = day_count + 1;
total_expenses = total_expenses + Double.parseDouble(dailyExpense);
if (flag)
{
    total_expenses = Double.parseDouble(dailyExpense);
    for (int i = 0; i < day_count; i++)
    {
        dailyExpenses.delete();
    }
    flag = false;
}
if (dbconn != null){
    try {
        PreparedStatement st = (PreparedStatement) dbconn.prepareStatement("Select * from accounts");
        ResultSet res = st.executeQuery();
        while(res.next())
        {
            id = res.getInt("id");
        }
        PreparedStatement st1 = (PreparedStatement) dbconn.prepareStatement("INSERT INTO expenses (id, budget, dailyExpenses, totalExpenses) VALUES (" + id
        st1.executeUpdate();
    }
    catch (SQLException ex) {
        Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);
    }
}
else{
    System.out.println("Database connection not available.");
}

```

b. Calculate their total travel expenses

id	budget	dailyExpenses	totalExpenses
33043	0	NULL	NULL
33043	1000	10	10
33043	1000	30	40

c. Show the user the percentage of their budget they have exhausted



```

double budget = 0;
double travel_expenses = 0;
Connection dbconn = DBConnection.connectDB();
if (dbconn != null){
    try {
        PreparedStatement st = (PreparedStatement) dbconn.prepareStatement("Select * from expenses WHERE totalExpenses=(SELECT MAX(totalExpenses) FROM expenses)");
        ResultSet res = st.executeQuery();
        while(res.next())
        {
            travel_expenses = res.getDouble("totalExpenses");
            budget = res.getDouble("budget");
        }
    }
    catch (SQLException ex) {
        Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);
    }
}
else{
    System.out.println("Database connection not available.");
}
JOptionPane.showMessageDialog(this, "You have used " + (100 * travel_expenses)/budget + " percent of your travel budget.", "Expenses Data", JOptionPane.INFORMATION);

```

Success Criteria – 4

- a. Allow the user to open and access Maps through the system

```
Runtime.getRuntime().exec("open -a Maps");
```

Success Criteria – 5

- a. Allow the user to input their daily itinerary

Completed	Task
Yes	Go swimming
Yes	Go jogging
No	Visit the mall


```
String comboValue = jComboBox1.getSelectedItem().toString();
if (comboValue.equals("TASK COMPLETE"))
{
    comboValue = "Yes";
}
else{
    comboValue = "No";
}
if (jTextField1.getText().isEmpty())
{
    JOptionPane.showMessageDialog(this, "Please enter all required fields.", "Error", JOptionPane.ERROR_MESSAGE);
}
else{
    String data[] = {comboValue, jTextField1.getText()};
    DefaultTableModel tb1Model = (DefaultTableModel) jTable1.getModel();
    tb1Model.addRow(data);
    jTextField1.setText("");
}
}
```

b. Allow the user to save and update their itinerary

id	completed	task
33043	Yes	Go swimming
33043	Yes	Go jogging
33043	No	Visit the mall

```
Connection dbconn = DBConnection.connectDB();
DefaultTableModel tb1Model = (DefaultTableModel) jTable1.getModel();
if (dbconn != null){
    try {
        if (tb1Model.getRowCount() == 0)
        {
            PreparedStatement st2 = (PreparedStatement) dbconn.prepareStatement("TRUNCATE TABLE tasks");
            st2.execute();
        }
        else{
            for (int i = 0; i < tb1Model.getRowCount(); i++)
            {
                PreparedStatement st = (PreparedStatement) dbconn.prepareStatement("Select * from accounts");
                ResultSet res = st.executeQuery();
                while(res.next())
                {
                    id = res.getInt("id");
                }
                String completed = tb1Model.getValueAt(i, 0).toString();
                String task = tb1Model.getValueAt(i, 1).toString();
                PreparedStatement st1 = (PreparedStatement) dbconn.prepareStatement("INSERT INTO tasks (id, completed, task) VALUES (?, ?, ?)");
                st1.setInt(1, id);
                st1.setString(2, completed);
                st1.setString(3, task);
                st1.executeUpdate();
            }
        }
    } catch (SQLException ex) {
        Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);
    }
    else{
        System.out.println("Database connection not available.");
    }
}
```

Success Criteria – 6

- a. Allow the user to add, delete and edit the journal daily
- b. Allow the user to mark each write-up with a specific date

Manage Itinerary

UPDATE JOURNAL

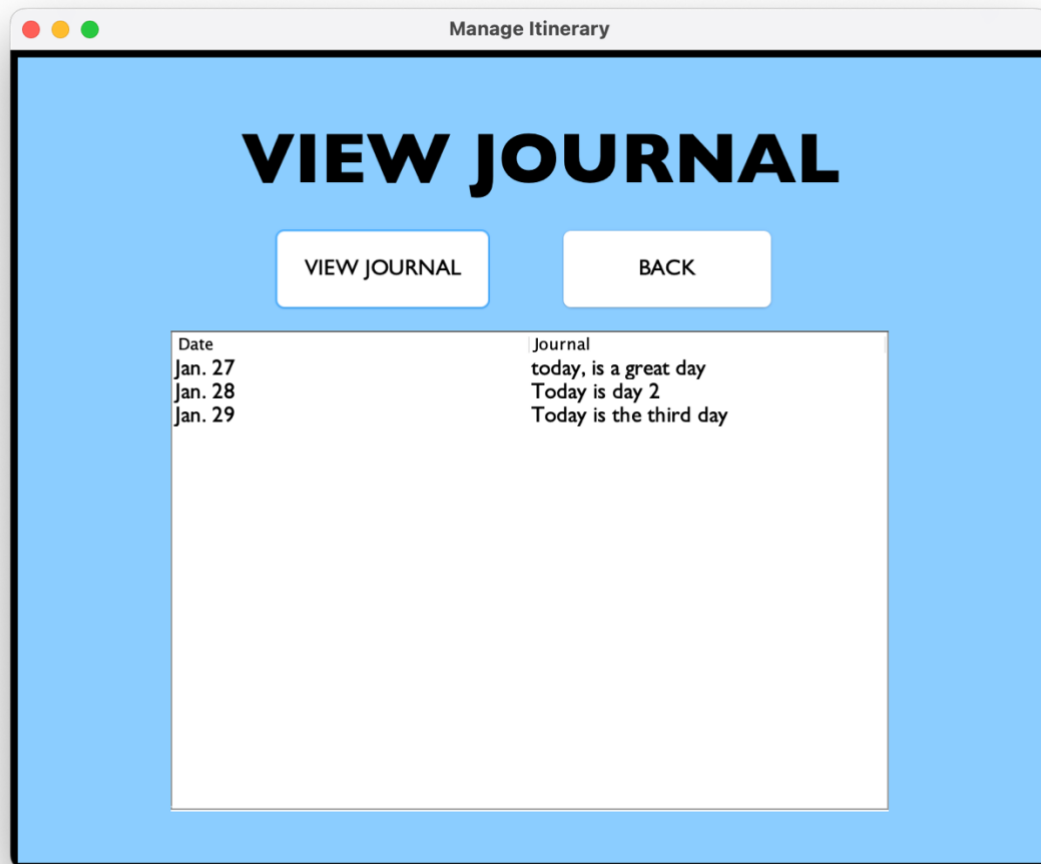
DATE

JOURNAL

BACK SUBMIT

```
String date = jTextField1.getText();
String journal = jTextField2.getText();
Connection dbconn = DBConnection.connectDB();
if (dbconn != null){
    try {
        PreparedStatement st = (PreparedStatement) dbconn.prepareStatement("Select * from accounts");
        ResultSet res = st.executeQuery();
        while(res.next())
        {
            id = res.getInt("id");
        }
        PreparedStatement st1 = (PreparedStatement) dbconn.prepareStatement("INSERT INTO journal (id, date, journal) VALUES (?, ?, ?)");
        st1.setInt(1, id);
        st1.setString(2, date);
        st1.setString(3, journal);
        st1.executeUpdate();
    }
    catch (SQLException ex) {
        Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);
    }
}
else{
    System.out.println("Database connection not available.");
}
```

- c. Allow the user to view the journal



```
Connection dbconn = DBConnection.connectDB();
if (dbconn != null){
    try {
        PreparedStatement st = (PreparedStatement) dbconn.prepareStatement("Select * from journal");
        ResultSet rs = st.executeQuery();
        DefaultTableModel tm = (DefaultTableModel)jTable2.getModel();
        tm.setRowCount(0);
        while(rs.next()){

            Object o[] = {rs.getString("date"),rs.getString("journal")};
            tm.addRow(o);

        }
        catch (SQLException ex) {
            Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    else{
        System.out.println("Database connection not available.");
    }
}
```

id	date	journal
33043	Jan. 27	today, is a great day
33043	Jan. 28	Today is day 2
33043	Jan. 29	Today is the third day

Success Criteria – 7

- Allow the user to click an image using the system

```
Runtime rt = Runtime.getRuntime();
try {
    String cmd = "open -a Camera";
    Process pr = rt.exec(cmd);
} catch (IOException ex) {
    Logger.getLogger(HomePage.class.getName()).log(Level.SEVERE, null, ex);
}
```

Success Criteria – 8

- Allow the user to add important contact details
- Allow the user to mark certain contacts with high importance, some with moderate importance and others with low importance

Manage Itinerary

CONTACTS

CONTACT NUMBER:

CONTACT NAME:

CONTACT IMPORTANCE:

BACK ADD DELETE SAVE VIEW

Importance	Name	Number
MEDIUM	9820412345	tom
MEDIUM	9979674532	mahesh
LOW	7846877773	dharmesh
HIGH	4434313589	paul

```

Connection dbconn = DBConnection.connectDB();
DefaultTableModel tb1Model = (DefaultTableModel) jTable1.getModel();
if (dbconn != null){
    try {
        if (tb1Model.getRowCount() == 0)
        {
            PreparedStatement st2 = (PreparedStatement) dbconn.prepareStatement("TRUNCATE TABLE contacts");
            st2.execute();
        }
        else{
            for (int i = 0; i < tb1Model.getRowCount(); i++)
            {
                PreparedStatement st = (PreparedStatement) dbconn.prepareStatement("Select * from accounts");
                ResultSet res = st.executeQuery();
                while(res.next())
                {
                    id = res.getInt("id");
                }
                String importance = tb1Model.getValueAt(i, 0).toString();
                String name = tb1Model.getValueAt(i, 1).toString();
                String number = tb1Model.getValueAt(i, 2).toString();
                PreparedStatement st1 = (PreparedStatement) dbconn.prepareStatement("INSERT INTO contacts (id, importance, name, number) VALUES (?, ?, ?, ?)");
                st1.setInt(1, id);
                st1.setString(2, importance);
                st1.setString(3, name);
                st1.setString(4, number);
                st1.executeUpdate();
            }
        }
        catch (SQLException ex) {
            Logger.getLogger(LoginPage.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    else{
        System.out.println("Database connection not available.");
    }
}

```

id	importance	name	number
33043	MEDIUM	9820412345	tom
33043	MEDIUM	9979674532	mahesh
33043	LOW	7846877773	dharmesh
33043	HIGH	4434313589	paul