|  |
| --- |
| Rental Management System |
| TCA 2 CSY2094 |

|  |
| --- |
| Richard Benny  1-9-2024 |

Table of Contents

[Project Overview 2](#_Toc152747785)

[Description of the problem 2](#_Toc152747786)

[Objectives and Scope 2](#_Toc152747787)

[Target users and Stakeholders 3](#_Toc152747788)

[Overview of the proposed solution. 3](#_Toc152747789)

[Functional Requirements 4](#_Toc152747790)

[Detailed functional requirements of the system. 4](#_Toc152747791)

[Use cases and user stories covering key features. 6](#_Toc152747792)

[Non-functional requirements. 8](#_Toc152747793)

[Prioritized list of requirements. 8](#_Toc152747794)

[System Architecture 10](#_Toc152747795)

[High-level architecture diagram. 10](#_Toc152747796)

[Description of key components and technologies. 10](#_Toc152747797)

[Database schema and descriptions. 11](#_Toc152747798)

[File structure and descriptions. 11](#_Toc152747799)

[UML Diagrams 12](#_Toc152747800)

[Use case diagrams. 12](#_Toc152747801)

[Class diagrams. 12](#_Toc152747802)

[Sequence diagrams for critical user workflows. 12](#_Toc152747803)

[State chart diagrams. 12](#_Toc152747804)

[Implementation 12](#_Toc152747805)

[Source code. 12](#_Toc152747806)

[User interface. 22](#_Toc152747807)

[Testing. 23](#_Toc152747808)

[Installation and deployment instructions. 23](#_Toc152747809)

[Conclusion. 23](#_Toc152747810)

[Summary. 23](#_Toc152747811)

[Potential enhancements and future work. 23](#_Toc152747812)

[Acknowledgements 23](#_Toc152747813)

[References 23](#_Toc152747814)

Project Overview

Description of the problem

The management of rental properties presents a complex set of difficulties that needs to be addressed efficiently and effectively. These challenges can span across different aspects of the system, including but not limited to:

1. **Property Management:** The task of managing all the different properties with a vast variety of attributes like location, size, type, or status can be overwhelming. This also includes keeping track of property maintenance certificates and tenant information within each property.
2. **Rent Collection:**  The process of collecting rent from each property in a timely manner can tedious and time-consuming. This is further complicated by late payments and tracking payment history.
3. **Communication:** Effective communication between tenants and landlords is very crucial for smooth operation. However, this can be complicated due to the lack of a centralised system to send notifications, updates, and other important information.
4. **Handling Complaints:** Addressing maintenance request effectively and in a timely manner is essential for tenant satisfaction. Having said that, tracking and managing these complaints from different properties can be a daunting task.
5. **Tedious Paperwork:** Renting a property is subject to many laws and regulations. So, Landlord or management team is often required to keep different documents regarding their properties and tenants. Storing these documents, safe and secure is a task on its own.

Objectives and Scope

The primary objective of this project is to develop and deploy a reliable system, which can handle all the different operational difficulties of landlords or management agencies.

This project aims to provide a seamless system for managing different properties in a user-friendly style to improve the efficiency and efficacy of the process. Tenants will be provided with a robust system to make rent payments, provide documentation, and to request maintenance. Additionally, the ability to have centralised control on properties, tenants and employees can be implemented depending on the users’ needs. The capability to raise, track and manage maintenance requests will be added. Likewise, a reliable system will be put in place to handle the collection and storage of different documents form tenants and certifications of the property, in a secure location. Overall, the aim of the project is to completely digitalise the process and to improve the communication between landlords and tenants.

As the project is developed as part of TCA 2 of my module, so the project is to be produced in one month. The main objective is to produce a software with the basic functionality implemented, as per the assignment brief.

Target users and Stakeholders

The primary users of this application are property owners or property management firms who are looking for a more reliable, and cheaper alternative to the current option. There are no stakeholders involved due to the project being developed for assignment.

Overview of the proposed solution.

The aim is to develop a desktop application using Java. Frontend will be made using Java Swing. The application will be able to add, remove, and update properties, and tenants. Additionally, the application will provide tenants with a portal to make payments, submit maintenance requests and track them, see their payment history.

Functional Requirements

Detailed functional requirements of the system.

Tenants: -

: Log in using tenant id and password.

: Interface for making payments.

: Add a late fee if payment is late. (depending on local laws)

: Get email confirmation of payment.

: Reset Password

: See payment history.

: Request for maintenance whenever they need.

: Track maintenance request submitted by them.

: Contact landlord from within the application.

: Ability to receive payment reminders.

: Able to view their contract and download it.

Landlords: -

: Log in using landlord id and password.

: Add, delete, or update information regarding their property.

: Add, delete, or update information regarding their tenants.

: Search tenants, house, maintenance request, payments

: Send user ID and temporary password when tenants are added via email.

: Reset tenant password manually.

: receive, process, and update maintenance requests.

: View details of rent payments, including who, amount etc.

: issue, view, and terminate contracts.

: View and edit staff timesheets.

: Process staff leave application.

: Schedule inspections and give feedback to tenants.

: Generate invoices and contracts on demand.

Employee: -

: Log in using employee id and password.

: Clock in and out for shifts and breaks.

: Submit requests for day off.

Admin: -

: Log in using Admin id and password.

: Add, remove, or update details regarding landlords.

: Add, delete, or update information regarding their property.

: Add, delete, or update information regarding their tenants.

: Add, delete, or update information regarding their staff.

: Access to error logs.

Use cases and user stories covering key features.

Use Case: Adding new property.

Use case name: Add New House.

Actors: landlord/property manager, system

Goal: Add a new house to the system.

Main flow:

1. User logs into the System.
2. Selects the house option from the menu bar.
3. Correctly inputs the needed fields.
4. Systems validates inputs and adds the new house.
5. System reflects the action in the list table.

User Story: As the landlord wants to add a new house to the portfolio.

Use Case: Adding new tenant.

Use case name: Add New Tenant.

Actors: landlord/property manager, system

Goal: Add a new tenant to the system.

Main flow:

1. User logs into the System.
2. Selects the tenant option from the menu bar.
3. Correctly inputs the needed values.
4. System validates the inputs.
5. System reflects the action in the list table.
6. System send an email to the tenant, providing their credentials.

User Story: As the landlord wants to add a new tenant.

Use Case: Processing requests.

Use case name: Process New Maintenance Requests.

Actors: landlord/property manager, system

Goal: process the newly arrived and existing maintenance request from tenants.

Main flow:

1. User logs into the System.
2. Selects the maintenance option from the menu bar.
3. Process the request and update the status.
4. System updates the necessary changes and updates the table.

User Story: As the landlord wants to process the maintenance requests.

Use Case: Adding/Updating payments

Use case name: Add new/update house expense.

Actors: landlord/property manager, system

Goal: Update/Add expense to an existing house.

Main flow:

1. User logs into the System.
2. Navigates to the payments tab.
3. Inputs the needed values.
4. System validates the input.
5. Add the expense on the house.
6. Reflect the action on the tables.

User Story: As the landlord wants to add/update expenses on a house.

Use Case: Resetting password.

Use case name: Update Tenant Password

Actors: landlord/property manager, system, tenant

Goal: Reset password for tenant.

Main flow:

1. Tenant requests a password change over email.
2. Landlord logs in to the system.
3. Navigates to the password reset tab.
4. Enters the username of the tenant.
5. Submits the request.
6. System validates the input.
7. System gives the landlord feedback on the operation.
8. System generates a random password.
9. Emails the tenant the new password.

User Story: As the tenant request a password change.

Use Case: User login.

Use case name: Login.

Actors: user, system

Goal: Login to the system

Main flow:

1. User launches the program.
2. Select the user type.
3. Provides the login details.
4. System validates the user.
5. Launches the dashboard based of the user type.

User Story: As a user wants to login.

Use Case: Makes new maintenance request

Use case name: Request maintenance.

Actors: tenant, system

Goal: Make a maintenance request to the landlord

Main flow:

1. Tenant logs into the System.
2. Navigates to the maintenance panel.
3. Enter the request description and submits.
4. System validates the input.
5. System adds the request to both landlord and tenant tables.

User Story: As the tenant wants to make a new maintenance request.

Use Case: Tenant resets the password

Use case name: Tenant Self Resets Password.

Actors: tenant, system

Goal: change password by tenant.

Main flow:

1. User logs into the System.
2. Navigates to the accounts section.
3. Enter new password and confirms it.
4. System validates the inputs.
5. System updates the password in database.
6. System sends a password change email.

User Story: As the tenant wants to update the his/her password.

Non-functional requirements.

1. Platform Compatibility: The target hardware specification is 4GB ram and entry-level processors. Screen resolution support for low-resolution displays.
2. Performance: Operational responsiveness should be less than two seconds for typical operations. Should support at least five concurrent users on the minimum specifications without significant degradation in performance.
3. Usability: Should be easy to use and navigate. The UI should be responsive.
4. Scalability: Smooth operation should be maintained even while handling large datasets like 200 tenants and 50 properties.
5. Security: Should implement security measures for access control. The password stored are saved after encryption.
6. Reliability: Errors should be handled appropriately to prevent the application form crashing.
7. Maintainability: The code should be easy to read and understand. Adequate comments and documentation should be provided.

Prioritized list of requirements.

**Tenants: -**

: Log in using tenant id and password.

: Interface for making payments.

: Add a late fee if payment is late. (depending on local laws)

: Get email confirmation of payment.

: Reset Password

: See payment history.

: Request for maintenance whenever they need.

: Track maintenance request submitted by them.

: Contact landlord from within the application.

**Landlords: -**

: Log in using landlord id and password.

: Add, delete, or update information regarding their property.

: Add, delete, or update information regarding their tenants.

: Search tenants, house, maintenance request, payments

: Send user ID and temporary password when tenants are added via email.

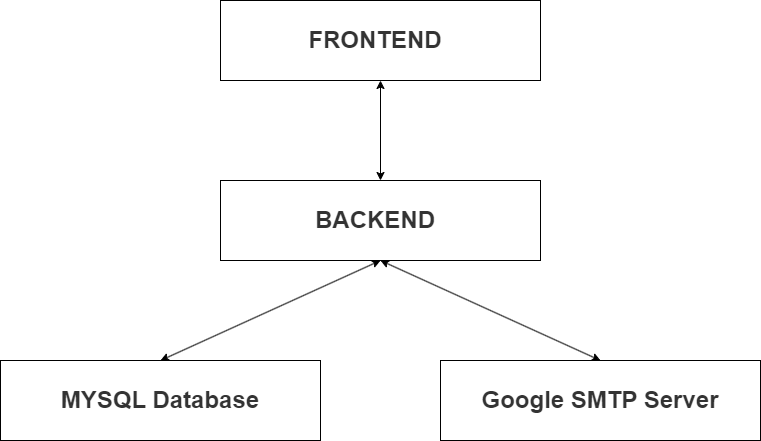
: Reset tenant password manually.

: receive, process, and update maintenance requests.

: View details of rent payments, including who, amount etc.

System Architecture

High-level architecture diagram.



Description of key components and technologies

* **Database:**  Connected to a MySQL server running on localhost. The Database is accessed only through the Database class. All the needed functions for manipulating the database are made within the class. Implements java.sql.Connection, java.sql.DriverManager, java.sql.PreparedStatement, java.sql.ResultSet, java.sql.SQLException.
* **SentEmail:** Connects with the Google SMTP Server. Provides function to send emails. Validates the user by reading the OAuth2.0 key file – Credentials.json. Generates a - token/StoredCredential.bin – file to validate the sender rather than performing validation every time an email is send.
* **FileConvertion:** A class to handle conversions form Blob to object as some class objects are stored in the database as Blob. These classes include Tenant and House. Each class implements the Serializable interface.
* **Hashing:** A class that performs all the one-way hash prevent the login details from being useful in the event of a data breach.
* **Build and Dependency Management:** Apache Maven serves as the primary build automation tool and dependency management system for the project. Maven simplifies the build process but defining project structures, managing dependencies, and facilitating project-wide configurations through its declarative XML-based POM(Project Object Model) files. (Maven Project, n.d.)

Database schema and descriptions

The database schema serves as a fundamental component supporting the data management and storage of Java project.

Schema name: objects

Table names:

1. houses :-
   1. houseId : VARCHAR(10)
   2. houseObject : BLOB
2. tenants :-
   1. tenantId : VARCHAR(10)
   2. tenantObject : BLOB
3. landlordpasswords:-
   1. id : INT
   2. username : VARCHAR(45)
   3. password : CHAR(64)
4. maintenance:-
   1. logId : VARCHAR(10)
   2. tenantName : VARCHAR(45)
   3. dateOfIssue : VARCHAR(10)
   4. houseId VARCHAR(10)
   5. tenantId : VARCHAR(10)
   6. description : LONGTEXT
   7. status : VARCHAR(45)
5. payments:-
   1. id : VARCHAR(45)
   2. tenantId : VARCHAR(45)
   3. date : VARCHAR(45)
   4. type : VARCHAR(45)
6. tenantjoindate:-
   1. id : INT
   2. tenantId : VARCHAR(45)
   3. date : VARCHAR(45)
   4. duedate : VARCHAR(45)
7. tenantpasswords:-
   1. id : INT
   2. username : VARCHAR(10)
   3. password : CHAR(64)

File structure and descriptions

Files also have significant role in the smooth operation of the Java project.

There are three file that stores important data inside. credentials.json and SESSION.txt are saved inside the file package of the project Whereas StoredCredential is stored inside token folder in the root directory.

credentials.json :- Stores the OAuth2.0 authorisation key for the email server to verify the client.

SESSION.txt :- Stores the username of the user for reference in the TenantController and LandlordController.

StoredCredential :- Stores the client secret key so client validation does not need to be done on every new instance of the SentEmail class.

UML Diagrams

Use case diagrams

Class diagrams

asdf

Sequence diagrams for critical user workflows

asdf

State chart diagrams

asdf

Implementation

Source code.

/\*Database.java\*/

package com.mycompany.rentalsystem.funcitons;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

/\*\*

 \* A class that establishes connection with the database.

 \*

 \* @author Richard

 \*/

public class Database {

    private String username = "root";

    private String password = "richard";

    private String database = "objects";

    private String url = "jdbc:mysql://localhost:3390/" + database;

    Connection connnection;

    PreparedStatement statement;

    /\*\*

     \* Constructor for the database class

     \*/

    public Database() {

        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

        } catch (ClassNotFoundException e) {

            System.out.println("connection failed");

            // add to logger

        }

        try {

            this.connnection = DriverManager.getConnection(url, username, password);

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

    /\*\*

     \* inserts value into the database

     \*

     \* @param table  - String table name

     \* @param values - String table heading with comma seperation

     \* @param record - Arraylist<?> id as String and object as Book

     \*/

    public void insert(String table, String values, ArrayList<Object> record) {

        /\* String values = columnArray.stream().collect(Collectors.joining(",")); \*/

        int arrLength = record.size();

        String placeHolder = "";

        for (int i = 0; i < arrLength; i++) {

            placeHolder += "?";

            if (i != arrLength - 1) {

                placeHolder += ",";

            }

        }

        String query = "INSERT INTO " + table + " (" + values + ") VALUES(" + placeHolder + ")";

        try {

            this.statement = connnection.prepareStatement(query);

            for (int i = 0; i < arrLength; i++) {

                this.statement.setObject(i + 1, record.get(i));

            }

            this.statement.execute();

            this.statement.close();

        } catch (SQLException e) {

            e.printStackTrace();

            // add to logger

        } catch (Exception exception) {

            exception.printStackTrace();

        }

    }

    /\*\*

     \* Fetches all the rows in a database.

     \* @param table String - table name

     \* @return ResultSet

     \*/

    public ResultSet findAll(String table) {

        try {

            this.statement = connnection.prepareStatement("SELECT \* FROM " + table);

            ResultSet result = this.statement.executeQuery();

            return result;

        } catch (Exception e) {

            e.printStackTrace();

        }

        return null;

    }

    /\*\*

     \* Finds rows that meets the where condition

     \*

     \* @param tableName String - name of table

     \* @param idColumnName String - name of the column in the where clause

     \* @param idValue String - value in the Where condition

     \* @return ResultSet

     \*/

    public ResultSet find(String tableName, String idColumnName, String idValue) {

        try {

            this.statement = connnection

                    .prepareStatement("SELECT \* FROM " + tableName + " WHERE " + idColumnName + " = ?");

            this.statement.setString(1, idValue);

            ResultSet result = this.statement.executeQuery();

            return result;

        } catch (Exception e) {

            e.printStackTrace();

        }

        return null;

    }

    /\*\*

     \* Updates the maintenance table; status of the request

     \*

     \* @param statusValue String - Status of the request

     \* @param idValue String - id of the request

     \*/

    public void updateMaintenance(String statusValue, String idValue) {

        try {

            this.statement = connnection.prepareStatement("UPDATE maintenance SET status = ? WHERE logId = ?");

            this.statement.setString(1, statusValue);

            this.statement.setString(2, idValue);

            this.statement.execute();

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

    /\*\*

     \* Find funtion specific to the tenant table

     \*

     \* @param id String - tenantID

     \* @return ResultSet

     \*/

    public ResultSet findTenant(String id) {

        try {

            this.statement = connnection.prepareStatement("SELECT \* FROM Tenants WHERE tenantId = ?");

            this.statement.setString(1, id);

            ResultSet result = this.statement.executeQuery();

            return result;

        } catch (Exception e) {

            e.printStackTrace();

        }

        return null;

    }

    /\*\*

     \* Find funtion specific to the house table

     \* @param id String - houseId

     \* @return ResultSet

     \*/

    public ResultSet findHouse(String id) {

        try {

            this.statement = connnection.prepareStatement("SELECT \* FROM houses WHERE houseId = ?");

            this.statement.setString(1, id);

            ResultSet result = this.statement.executeQuery();

            return result;

        } catch (Exception e) {

            e.printStackTrace();

        }

        return null;

    }

    /\*\*

     \* Deletes the specified row from the specified table

     \* @param table String - Name of the table

     \* @param columnLabel String - Name of the Column

     \* @param id String - value of the column

     \*/

    public void delete(String table, String columnLabel, String id) {

        String query = "DELETE FROM " + table + " WHERE " + columnLabel + " = " + id;

        try {

            this.statement = connnection.prepareStatement(query);

            this.statement.execute();

            this.statement.close();

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

    /\*\*

     \* Update funtion specific to the house table

     \* @param id String - houseId

     \* @param house Object - House instance

     \*/

    public void updateHouse(String id, Object house) {

        String query = "UPDATE houses SET houseObject = ? WHERE houseId = ?";

        try {

            byte[] object = FileConvertion.toByteArray(house);

            this.statement = connnection.prepareStatement(query);

            this.statement.setObject(1, object);

            this.statement.setObject(2, id);

            this.statement.execute();

            this.statement.close();

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    /\*\*

     \* Update funtion specific to the tenant table

     \* @param id String - tenatId

     \* @param tenant - Tenant instance

     \*/

    public void updateTenant(String id, Object tenant) {

        String query = "UPDATE tenants SET tenantObject = ? WHERE tenantId = ?";

        try {

            byte[] object = FileConvertion.toByteArray(tenant);

            this.statement = connnection.prepareStatement(query);

            this.statement.setObject(1, object);

            this.statement.setObject(2, id);

            this.statement.execute();

            this.statement.close();

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    /\*\*

     \* Update funtion specfic to the password table

     \*

     \* @param table String - table name

     \* @param newPassword String - new password to be added

     \* @param username String - id

     \*/

    public void updatePassword(String table, String newPassword, String username) {

        String query = "UPDATE " + table + " SET password = ? WHERE username = ?";

        try {

            this.statement = connnection.prepareStatement(query);

            this.statement.setString(1, newPassword);

            this.statement.setString(2, username);

            this.statement.execute();

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

    /\*\*

     \* find function specific for the getting the password of the user

     \* @param tableName String - name of table

     \* @param username - String - userId

     \* @return ResultSet

     \*/

    public ResultSet passwordCheck(String tableName, String username) {

        String query = "SELECT \* FROM " + tableName + " WHERE username = ?";

        try {

            this.statement = this.connnection.prepareStatement(query);

            this.statement.setString(1, username);

            return this.statement.executeQuery();

        } catch (Exception e) {

            e.printStackTrace();

        }

        return null;

    }

    @Override

    public String toString() {

        return "Database [username=" + username + ", password=" + password + ", database=" + database + ", url=" + url

                + "]";

    }

}

/\*FileConvertion.java\*/

package com.mycompany.rentalsystem.funcitons;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.sql.SQLException;

import java.sql.Blob;

/\*\*

 \* A class to handle convertions form one type to another

 \*/

public class FileConvertion {

    /\*\*

     \* converts the provided object to byte[]

     \* @param objectToConvert Object

     \* @return byte[] of the object

     \*/

    public static byte[] toByteArray(Object objectToConvert) {

        ByteArrayOutputStream bos = new ByteArrayOutputStream();

        ObjectOutputStream oos;

        try {

            oos = new ObjectOutputStream(bos);

            oos.writeObject(objectToConvert);

        } catch (IOException e) {

            e.printStackTrace();

        }

        byte[] object = bos.toByteArray();

        return object;

    }

    /\*\*

     \* Converts Blob into usable objects.

     \* @param objectBlob Blob

     \* @return Object

     \*/

    public static Object toObject(Blob objectBlob) {

        try {

            byte[] byteArray = objectBlob.getBytes(1, (int) objectBlob.length());

            ByteArrayInputStream byteArrInpStm = new ByteArrayInputStream(byteArray);

            ObjectInputStream objInpStm;

            Object Obj;

            objInpStm = new ObjectInputStream(byteArrInpStm);

            Obj = (Object) objInpStm.readObject();

            return Obj;

        } catch (ClassNotFoundException | IOException | SQLException e) {

            e.printStackTrace();

        }

        return null;

    }

}

/\* Hashing.java

 \* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

 \* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

 \*/

package com.mycompany.rentalsystem.funcitons;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

/\*\*

 \* the method takes in username and password, and then runs them through a hashing alogrithm.

 \* outputs the encoded password as a string.

 \* reference: https://www.baeldung.com/java-password-hashing, https://stackoverflow.com/questions/5531455/how-to-hash-some-string-with-sha-256-in-java

 \* @author Richard

 \*/

public class Hashing {

    /\*\*

     \* performs the one way hashing

     \* @param password String - user password

     \* @param username String - user username

     \* @return String the hashed password

     \* @throws NoSuchAlgorithmException

     \*/

    public static String doHashing(String password, String username) throws NoSuchAlgorithmException{

        String passwordText = username + password;

        MessageDigest digest = MessageDigest.getInstance("SHA-256");

        digest.update(passwordText.getBytes());

        byte[] encodedPasswordArray = digest.digest();

        StringBuilder encodedPassword = new StringBuilder();

        for (byte b : encodedPasswordArray){

            encodedPassword.append(Integer.toHexString(0xFF & b));

        }

        return encodedPassword.toString();

    }

}

/\*SendEmail.java\*/

package com.mycompany.rentalsystem.funcitons;

import java.io.ByteArrayOutputStream;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.security.GeneralSecurityException;

import java.util.Collections;

import java.util.Properties;

import java.util.Set;

import javax.mail.MessagingException;

import javax.mail.Session;

import javax.mail.internet.AddressException;

import javax.mail.internet.InternetAddress;

import com.google.api.client.auth.oauth2.Credential;

import com.google.api.client.extensions.java6.auth.oauth2.AuthorizationCodeInstalledApp;

import com.google.api.client.extensions.jetty.auth.oauth2.LocalServerReceiver;

import com.google.api.client.googleapis.auth.oauth2.GoogleAuthorizationCodeFlow;

import com.google.api.client.googleapis.auth.oauth2.GoogleClientSecrets;

import com.google.api.client.googleapis.javanet.GoogleNetHttpTransport;

import com.google.api.client.googleapis.json.GoogleJsonError;

import com.google.api.client.googleapis.json.GoogleJsonResponseException;

import com.google.api.client.http.javanet.NetHttpTransport;

import com.google.api.client.json.JsonFactory;

import com.google.api.client.json.gson.GsonFactory;

import com.google.api.client.util.store.FileDataStoreFactory;

import com.google.api.services.gmail.Gmail;

import com.google.api.services.gmail.GmailScopes;

import com.google.api.services.gmail.model.Message;

import javax.mail.internet.MimeMessage;

import org.apache.commons.codec.binary.Base64;

public class SentEmail {

    private static final String filePath = "src/main/java/com/mycompany/rentalsystem/files/credentials.json";

    private static final JsonFactory JsonFactory = GsonFactory.getDefaultInstance();

    private static final String fromEmailAddress = "uk.developer.java@gmail.com";

    private final Gmail service;

    /\*\*

     \* Constructor of the class.

     \*

     \* @throws GeneralSecurityException

     \* @throws IOException

     \*/

    public SentEmail() throws GeneralSecurityException, IOException {

        final NetHttpTransport HTTP\_TRANSPORT = GoogleNetHttpTransport.newTrustedTransport();

        service = new Gmail.Builder(HTTP\_TRANSPORT, JsonFactory, getCredentials(HTTP\_TRANSPORT))

                .setApplicationName("RENTAL MANAGEMENT")

                .build();

    }

    /\*\*

     \* static function to load the client secrets.

     \* @param HTTP\_TRANSPORT NetHttpTransport

     \* @return Credential

     \* @throws IOException

     \*/

    private static Credential getCredentials(final NetHttpTransport HTTP\_TRANSPORT) throws IOException {

        // Load client secrets.

        //InputStream in = SentEmail.class.getResourceAsStream(filePath);

        InputStream in = new FileInputStream(filePath);

        if (in == null) {

            throw new FileNotFoundException("Resource not found: " + filePath);

        }

        GoogleClientSecrets clientSecrets = GoogleClientSecrets.load(JsonFactory, new InputStreamReader(in));

        // Build flow and trigger user authorization request.

        GoogleAuthorizationCodeFlow flow = new GoogleAuthorizationCodeFlow.Builder(

                HTTP\_TRANSPORT, JsonFactory, clientSecrets, Set.of(GmailScopes.GMAIL\_SEND))

                .setDataStoreFactory(new FileDataStoreFactory(new java.io.File("token")))

                .setAccessType("offline")

                .build();

        LocalServerReceiver receiver = new LocalServerReceiver.Builder().setPort(8888).build();

        Credential credential = new AuthorizationCodeInstalledApp(flow, receiver).authorize("user");

        // returns an authorized Credential object.

        return credential;

    }

    /\*\*

     \* sents the email

     \* @param toEmailAddress

     \* @param subject

     \* @param messageBody

     \* @return

     \* @throws GeneralSecurityException

     \* @throws IOException

     \* @throws AddressException

     \* @throws MessagingException

     \*/

    public Message sentMail(String toEmailAddress, String subject, String messageBody)

            throws GeneralSecurityException, IOException, AddressException, MessagingException {

        // Encode as MIME message

        Properties props = new Properties();

        Session session = Session.getDefaultInstance(props, null);

        MimeMessage email = new MimeMessage(session);

        email.setFrom(new InternetAddress(fromEmailAddress));

        email.addRecipient(javax.mail.Message.RecipientType.TO,

                new InternetAddress(toEmailAddress));

        email.setSubject(subject);

        email.setText(messageBody);

        // Encode and wrap the MIME message into a gmail message

        ByteArrayOutputStream buffer = new ByteArrayOutputStream();

        email.writeTo(buffer);

        byte[] rawMessageBytes = buffer.toByteArray();

        String encodedEmail = Base64.encodeBase64URLSafeString(rawMessageBytes);

        Message message = new Message();

        message.setRaw(encodedEmail);

        try {

            // Create send message

            message = service.users().messages().send("me", message).execute();

            System.out.println("Message id: " + message.getId());

            System.out.println(message.toPrettyString());

            return message;

        } catch (GoogleJsonResponseException e) {

            GoogleJsonError error = e.getDetails();

            if (error.getCode() == 403) {

                System.err.println("Unable to send message: " + e.getDetails());

            } else {

                throw e;

            }

        }

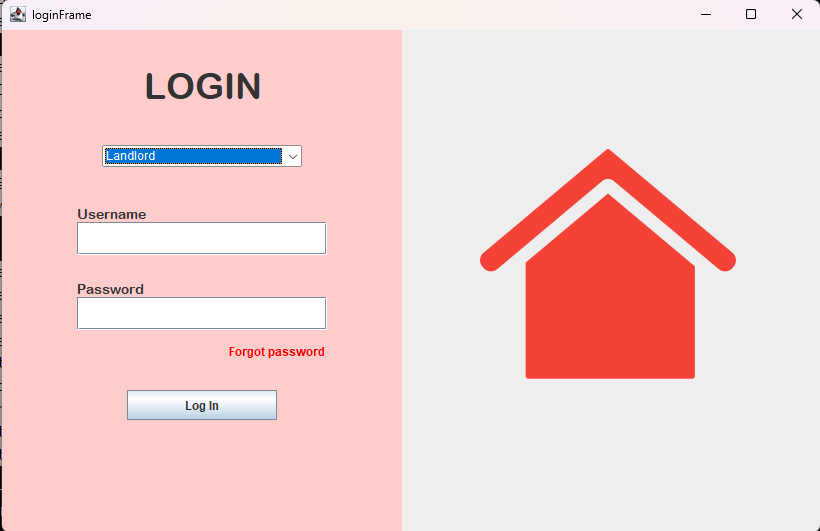
        return message;

    }

}

User interface

Login Page



Testing.

Installation and deployment instructions.

asdf

Conclusion.

asef

Summary.

asdf

Potential enhancements and future work.

asdf

Acknowledgements

I declare that this is all my own work and does not contain unreferenced material copied from any other source. I have read the University’s policy on plagiarism and understand the definition of plagiarism. If it is shown that material has been plagiarized, or I have otherwise attempted to obtain an unfair advantage for myself or others, I understand that I may face sanctions in accordance with the policies and procedures of the University. A mark of zero may be awarded, and the reason for that mark will be recorded on my file.

Furthermore, I acknowledge that if I have used code generated from AI tools such as ChatGPT or utilized third-party libraries.

References

UI design

Links of all competitors

Theme picker.

Java libraries used link to Javadoc.

Welcome to Apache Maven - https://maven.apache.org/

Bing AI - [Copilot with GPT-4 (bing.com)](https://www.bing.com/search?form=NTPCHB&q=Bing+AI&showconv=1)

`