**ENERGY SUPPLY SYSTEM REPORT**

**Implementation Table**

|  |  |  |  |
| --- | --- | --- | --- |
| SL. | Feature Implemented | Partial/Full | Any comments |
| 01. | Record details of a new customer (name, phone number, current address, energy tariff, meter type, etc.) | Full |  |
| 02. | Display energy usage dashboard of a particular customer (e.g., meter readings, account summary (payments, bills, tariff details and so on) |  |  |
| 03. | Create a monthly invoice/bill for each customer. The bill should also show detailed calculations of energy charges (i.e., conversion of meter readings to charges) |  |  |
| 04. | Record a payment for each customer (i.e., paid/unpaid) |  |  |
| 05. | Permanent storage of data and CRUD [Create, Read, Update and Delete] (object serialisation) |  |  |
| 06. | Input/Read energy meter readings from a file (object serialisation or text/csv files). |  |  |
| 07. | Search for a customer by account number or name | Full |  |
| 08. | Input/update tariff information (i.e., add/modify new/existing tariff offered) |  |  |
| 09. | Display annual energy usage chart and predicted monthly usage |  |  |
| 10. | Send an email containing the invoice as an attachment (pdf file) to a customer | No |  |
| 11. | Login feature for the system | Full |  |

**Implementation PHASE-1**

Implementing 01 (Partial), 11 (Full), Application Structure (MVC), FXML Multiple Scene Capability, GUI for Signup-Login, DAO Design Pattern for models, Junit Testing

**Model (DAO)**

public class CustomerDaoFile implements Dao<Customer> {  
  
 private static Hashtable<String, Customer> *customers*;  
 private final String fileName = "fileDB/customerDB.txt";  
  
 public CustomerDaoFile() {  
 *// read and init from file  
 customers* = new Hashtable<>();  
 readDataFromFile();  
 }  
  
 */\*\*  
 \* Read all Customers from file and init to Map  
 \*/* private synchronized void readDataFromFile() {  
 try {  
 ObjectInputStream in = new ObjectInputStream(new FileInputStream(fileName));  
 *customers* = (Hashtable) in.readObject();  
 in.close();  
 } catch (IOException | ClassNotFoundException e) {  
*// e.printStackTrace();* System.*out*.println("File was not created");  
 }  
 }  
  
 */\*\*  
 \* Read all Customers from file and init to Map  
 \*/* private synchronized void writeDataIntoFile() {  
 try {  
 FileOutputStream fout = new FileOutputStream(fileName);  
 ObjectOutputStream out = new ObjectOutputStream(fout);  
 out.writeObject(*customers*);  
 out.flush();  
 fout.close();  
 out.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Override  
 public Customer get(String id) {  
 return *customers*.get(id);  
 }  
  
 @Override  
 public List<Customer> getAll() {  
 return new ArrayList<>(*customers*.values());  
 }  
  
 @Override  
 public boolean save(Customer customer) {  
 Customer put = *customers*.put(customer.getUsername(), customer);  
 writeDataIntoFile();  
 return put == null;  
 }  
  
 @Override  
 public boolean update(String key, Customer customer) {  
 if (*customers*.get(key) == null) return false;  
 *customers*.computeIfPresent(customer.getUsername(), (k, v) -> customer);  
 writeDataIntoFile();  
 return true;  
 }  
  
 @Override  
 public boolean delete(Customer customer) {  
 boolean value = *customers*.remove(customer.getUsername()) != null;  
 writeDataIntoFile();  
 return value;  
 }  
}

In the model we have create 4 classes(‘Person, Customer,Dao, CustomerDaoFile) to build our model structure. In CustomerDaoFile we implement Dao interface to follow-up DAO design pattern. We used Hashtable as under layering data-structure for our customers so we can access O(1) to ant customer. Each time we create a object of CustomerDaoFile read previous data from file and when we update data in update in file. All reference objects are Serializable so we can store the hashtable as binary stream in file.

**Controller**

/\*\* A sample Controller Class

public class LoginController implements Stageable {  
  
 private Stage stage; *// main stage* @FXML  
 private ImageView logoImg; *// title logo* @FXML  
 private Label errorm; *// showing error messages* @FXML  
 private TextField username; *// entered username* @FXML  
 private PasswordField password; *// entered password  
  
 /\*\*  
 \* initialize at start  
 \*/* @FXML  
 public void initialize() {  
 Utils.*setImage*(logoImg, "images/system-logo.png"); *// set logo* }  
  
 *// Customer DAO* private final Dao<Customer> cusDao = new CustomerDaoFile();  
  
 */\*\*  
 \* Customer login  
 \*/* public void customerLogin(ActionEvent actionEvent) {  
 Customer customer = cusDao.get(username.getText());  
 *// if customer already registered, username veryfied* if (customer != null && !password.getText().equals("")) {  
 *// verify password* boolean verified = PasswordManager.*verify*(password.getText(), customer.getPassword());  
 if (verified) {  
 *// successful login* System.*out*.println("Login Success, load dashboard");  
 stage.setScene(App.*getScenes*().get(SceneName.*DASHBOARD*).getScene());  
 } else {  
 *// invalid password* errorm.setText(StringData.*invalidPass*);  
 }  
  
 } else {  
 errorm.setText(StringData.*invalidUserPass*);  
 }  
 }  
  
 */\*\*  
 \* Customer sign-up  
 \*/* public void customerSignUp(ActionEvent actionEvent) {  
 stage.setScene(App.*getScenes*().get(SceneName.*SIGNUP*).getScene()); *// load signup page* }  
  
 */\*\*  
 \* exit button handler  
 \*/* public void exitButtonAction(ActionEvent actionEvent) {  
 System.*exit*(0);  
 }  
  
 @Override  
 public void setStage(Stage stage) {  
 this.stage = stage;  
 }  
}

This is an example for JavaFX FXML Controller, The main prepose of this controller is handle login of a customer, in the ‘customerLogin()’ method we did that, using the File DAO object we checked is the username is already registered or not if registered we forward to verify password with help of ‘PasswordManager’ class. And showing error messages using a label named ‘errorm. All other controller has structure but different prepose and all Controllers should be implemented ‘Stageable’ interface to perform multi scene load capability.

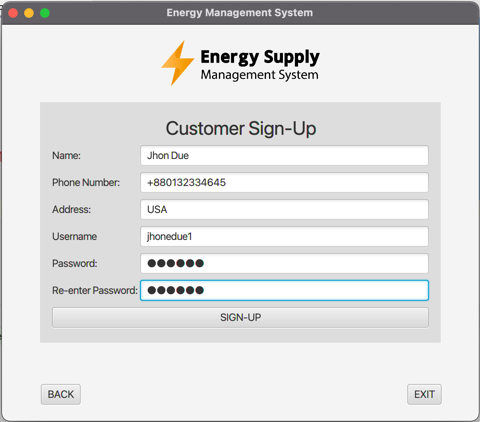
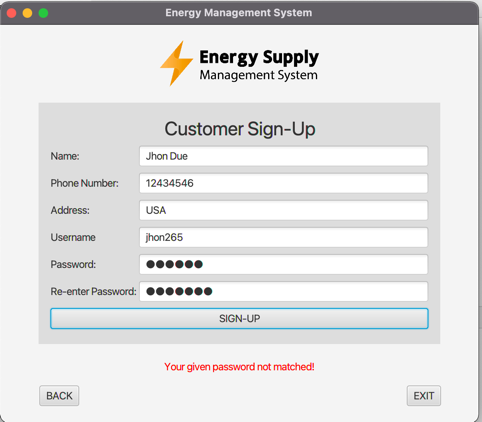
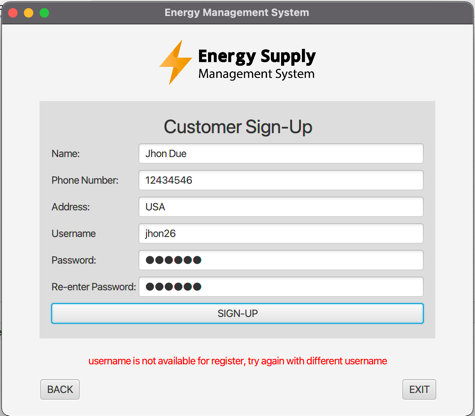
**View**

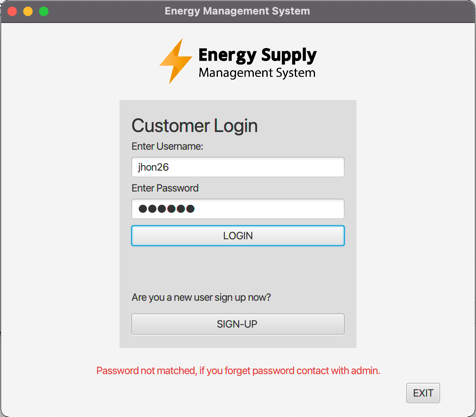
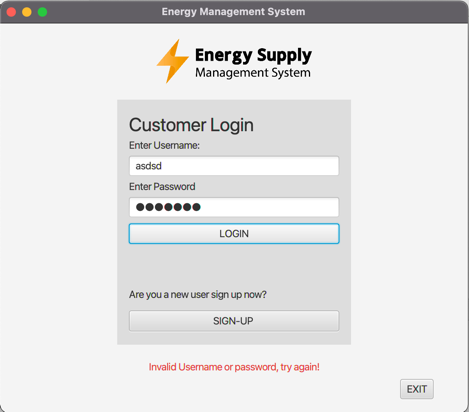
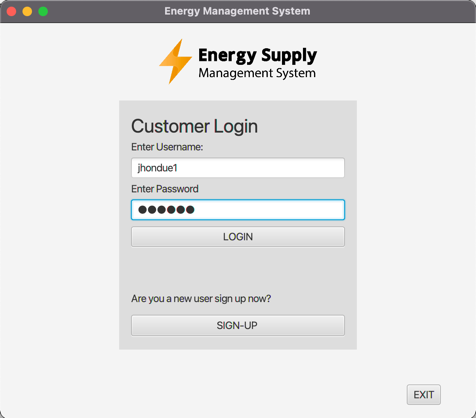
*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<!--Import all package-->  
  
<?*import javafx.geometry.Insets*?>  
<?*import javafx.scene.control.Button*?>  
<?*import javafx.scene.control.Label*?>  
<?*import javafx.scene.control.PasswordField*?>  
<?*import javafx.scene.control.TextField*?>  
<?*import javafx.scene.image.Image*?>  
<?*import javafx.scene.image.ImageView*?>  
<?*import javafx.scene.layout.AnchorPane*?>  
<?*import javafx.scene.layout.ColumnConstraints*?>  
<?*import javafx.scene.layout.GridPane*?>  
<?*import javafx.scene.layout.RowConstraints*?>  
<?*import javafx.scene.layout.VBox*?>  
<?*import javafx.scene.text.Font*?>  
<?*import javafx.scene.text.Text*?>  
  
<!--VBox Frame-->*<VBox alignment="TOP\_CENTER" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="500.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/17" xmlns:fx="http://javafx.com/fxml/1" fx:controller="system.controller.LoginController">  
 <children>  
 <ImageView fx:id="logoImg" fitHeight="150.0" fitWidth="200.0" pickOnBounds="true" preserveRatio="true">  
 <image>  
 <Image url="@../../../../images/system-logo.png" />  
 </image>  
 <VBox.margin>  
 <Insets top="10.0" />  
 </VBox.margin>  
 </ImageView>  
 <GridPane maxWidth="-Infinity" prefHeight="315.0" prefWidth="300.0" style="-fx-background-color: #dddddd;">  
 <columnConstraints>  
 <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />  
 </columnConstraints>  
 <rowConstraints>  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="15.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="15.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="40.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  
 </rowConstraints>  
 <VBox.margin>  
 <Insets top="20.0" />  
 </VBox.margin>  
 <children>  
 <Label text="Enter Username:" GridPane.rowIndex="1" />  
 <TextField fx:id="username" promptText="username" GridPane.rowIndex="2" />  
 <Button maxWidth="1.7976931348623157E308" mnemonicParsing="false" onAction="#customerLogin" prefHeight="24.0" prefWidth="270.0" text="LOGIN" GridPane.rowIndex="5" />  
 <Label text="Enter Password" GridPane.rowIndex="3" />  
 <PasswordField fx:id="password" promptText="password" GridPane.rowIndex="4" />  
 <Label text="Are you a new user sign up now?" GridPane.rowIndex="7" />  
 <Button layoutX="25.0" layoutY="189.0" maxWidth="1.7976931348623157E308" mnemonicParsing="false" onAction="#customerSignUp" prefHeight="27.0" prefWidth="270.0" text="SIGN-UP " GridPane.rowIndex="8" />  
 <Label text="Customer Login">  
 <font>  
 <Font name="System Bold" size="24.0" />  
 </font>  
 </Label>  
 <Text strokeType="OUTSIDE" strokeWidth="0.0" GridPane.rowIndex="6" />  
 </children>  
 <padding>  
 <Insets bottom="15.0" left="15.0" right="15.0" top="15.0" />  
 </padding>  
 </GridPane>  
 <Label fx:id="errorm" textFill="#e14242">  
 <VBox.margin>  
 <Insets top="20.0" />  
 </VBox.margin>  
 </Label>  
 <AnchorPane prefHeight="41.0" prefWidth="580.0">  
 <children>  
 <Button layoutX="503.0" layoutY="7.0" mnemonicParsing="false" onAction="#exitButtonAction" text="EXIT" />  
 </children>  
 </AnchorPane>  
 </children>  
 <padding>  
 <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />  
 </padding>  
</VBox>

All view in application are created using SceneBuilder tool, here is a example of a view. The code of login view.

**BLACK BOX TESTING OF USER LOGIN FEATURE**

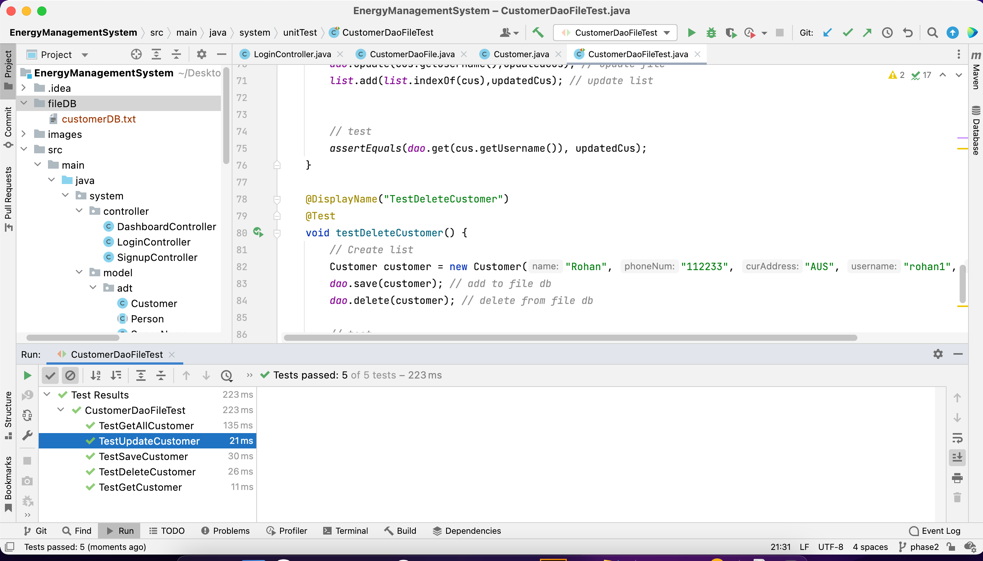
In the phase1 the program perfectly able to handle login & signup





**JUNIT TESTING**

package system.unitTest;  
  
import org.junit.jupiter.api.\*;  
import system.model.adt.Customer;  
import system.model.dao.CustomerDaoFile;  
import system.model.dao.Dao;  
  
import java.util.ArrayList;  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
  
public class CustomerDaoFileTest {  
  
 private static Dao<Customer> *dao*;  
 private static ArrayList<Customer> *list*;  
  
  
 @BeforeEach  
 void setup() {  
 *//log.info("@BeforeEach - executes before each test method in this class");  
 // "@BeforeAll - executes once before all test methods in this class"  
 dao* = new CustomerDaoFile();  
 *list* = new ArrayList<>();  
 Customer customer = new Customer("Jhon Due", "+0011223344", "USA",  
 "jhon12", "jhon1122");  
 ;  
 Customer customer2 = new Customer("Jony Hakon", "+003425334", "CA",  
 "jony23", "11jony");  
  
 *// save data  
 dao*.save(customer);  
 *dao*.save(customer2);  
 *list*.add(customer);  
 *list*.add(customer2);  
 }  
  
 @DisplayName("TestGetCustomer")  
 @Test  
 void testGetCustomer() {  
 Customer customer = new Customer("Jhon Due", "+0011223344", "USA",  
 "jhon12", "jhon1122");  
 *assertEquals*(*dao*.get(customer.getUsername()), customer);  
 }  
  
 @DisplayName("TestGetAllCustomer")  
 @Test  
 void testGetAllCustomer() {  
 *assertEquals*(*dao*.getAll(), *list*);  
 }  
  
 @DisplayName("TestSaveCustomer")  
 @Test  
 void testSaveCustomer() {  
 *// Create list* Customer customer = new Customer("Jocab", "112233", "AUS", "jocab1", "123456");  
 dao.save(customer); *// save on file  
 // test* assertEquals(dao.get(customer.getUsername()), customer);  
 dao.delete(customer);  
 }  
  
 @DisplayName("TestUpdateCustomer")  
 @Test  
 void testUpdateCustomer() {  
 *// Create list* Customer cus = new Customer("Jhon Due", "+0011223344", "USA",  
 "jhon12", "jhon1122");  
 Customer updatedCus = new Customer("Honor Milan", "+1111223344", "CA",  
 "jhon12", "jhon1122");  
  
 dao.update(cus.getUsername(),updatedCus); *// update file* list.add(list.indexOf(cus),updatedCus); *// update list  
  
  
 // test* assertEquals(dao.get(cus.getUsername()), updatedCus);  
 }  
  
 @DisplayName("TestDeleteCustomer")  
 @Test  
 void testDeleteCustomer() {  
 *// Create list* Customer customer = new Customer("Rohan", "112233", "AUS", "rohan1", "123456");  
 dao.save(customer); *// add to file db* dao.delete(customer); *// delete from file db  
  
 // test* assertEquals(dao.get(customer.getUsername()), null);  
 }  
  
}



The CustomerDaoFileTest is Unit tester for CustomerDaoFile

**Password Manager -> encode and verify method, encode method encode/hash the password using MD5 algorithm.**

public static String encode(String password) throws NoSuchAlgorithmException {  
 *// code copied from Javatpoint  
 /\* MessageDigest instance for MD5. \*/* MessageDigest m = MessageDigest.*getInstance*("MD5");  
 */\* Add plain-text password bytes to digest using MD5 update() method. \*/* m.update(password.getBytes());  
 */\* Convert the hash value into bytes \*/* byte[] bytes = m.digest();  
 */\* The bytes array has bytes in decimal form. Converting it into hexadecimal format. \*/* StringBuilder s = new StringBuilder();  
 for (byte aByte : bytes) {  
 s.append(Integer.*toString*((aByte & 0xff) + 0x100, 16).substring(1));  
 }  
 */\* Complete hashed password in hexadecimal format \*/* return s.toString();  
}

**Main Application -> Entry point of the application**

public class App extends Application {  
  
 *// Holds the information for various scenes to switch between* private static final Map<SceneName, Fxml> *scenes* = new HashMap<>();  
  
 *// All file path of fxml view* private static final String *LOGIN\_FXML* = "/fxml/login-view.fxml";  
 private static final String *SIGNUP\_FXML* = "/fxml/signup-view.fxml";  
 private static final String *DASHBOARD\_FXML* = "/fxml/dashboard-view.fxml";  
  
 @Override  
 public void start(Stage stage) {  
 *// register all scene  
 scenes*.put(SceneName.*LOGIN*, new Fxml(*LOGIN\_FXML*, SceneName.*LOGIN*, stage)); *// login  
 scenes*.put(SceneName.*SIGNUP*, new Fxml(*SIGNUP\_FXML*, SceneName.*SIGNUP*, stage)); *// signup  
 scenes*.put(SceneName.*DASHBOARD*, new Fxml(*DASHBOARD\_FXML*, SceneName.*DASHBOARD*, stage)); *// dashboard  
  
 // getScene() will load the FXML file the first time* stage.setScene(*scenes*.get(SceneName.*LOGIN*).getScene());  
 stage.setTitle(StringData.*title*);  
 stage.show();  
 }  
  
 *// @return a Map of the {@link Fxml} by {@link SceneName}* public static Map<SceneName, Fxml> getScenes() {  
 return *scenes*;  
 }  
  
 *// Update the scene Map with new FxmlInfo* public static synchronized void updateScenes(SceneName name, Fxml info) {  
 *scenes*.put(name, info);  
 }  
  
 *// main* public static void main(String[] args) {  
 *launch*(args);  
 }  
}

In here we use hashmap data structure to store all fxml Scene so we can load any scene anytime we want.

In the begin of class we sored all fxml files path and in the start method we add all fxml files path in the scene hashmap.

**Implementation PHASE-2**

Implementing 01 (Partial), 11 (Full),