Northampton Energy Supplier System

# Introduction

This system is for maintain the record of energy supplied in Northampton by the supplier. This system enabled the supplier to maintain record of customer accounts and their details, energy tariffs, consumers usage and generate invoices.

# Credentials

This system needs username and password to access the system. Else you can’t access the system.

The Username for this system is “admin” and password is “admin”.

# Class Diagram



# Features Implemented

|  |  |  |
| --- | --- | --- |
| **Feature** | **Implemented (Full/Partial)** | **Any Comments** |
| Login System | Full | No error validation |
| Record New Consumer details | Full | No error validation |
| Update Existing Customer details | Full | No error validation |
| Search Consumer by Account Number, Name and Phone No | Full | No error |
| Display Energy Usage  Dashboard | Full | Handled errors for consumer who has no energy consumed record |
| Display Chart for Annual Usage | Full | Handled the error for months which has no record |
| Generate and Create Invoice of consumed units for Individual Consumer | Full | Error if machine doesn’t have program to print or open pdf file. This error is also handled |
| Record Readings of Consumer | Full | No error validation (Except Month and Year) |
| Update the Payment Record of each Consumer | Full | No error |
| Added Output Screens for Errors | Partial |  |
| Consumed Units History of each Consumer | Full | No error |
| Record of Consumer and Their Usage is Saving in Files | Full |  |

# Explanation of the main sections/fragments of the code

## Get All Consumers from File

//Retrieve all Consumers from file  
public static ArrayList<Consumer> getAllConsumers() throws IOException {  
 Path path = Paths.*get*(*CONSUMERS\_FILE\_NAME*);  
 List<String> lines = Files.*readAllLines*(path, StandardCharsets.*UTF\_8*);  
  
 ArrayList<Consumer> consumers = new ArrayList<>();  
 for (String line : lines) {  
 Consumer consumer = *getConsumerFromLine*(line.split(Utils.*OUTPUT\_SPLITTER*));  
 consumers.add(consumer);  
 }  
  
 return consumers;  
}

## Get Consumer by Account Number

public static Consumer getConsumerByAccountNumber(String accountNumber) throws IOException {  
 Path path = Paths.*get*(*CONSUMERS\_FILE\_NAME*);  
 List<String> lines = Files.*readAllLines*(path, StandardCharsets.*UTF\_8*);  
 Consumer consumer = new Consumer();  
  
 for (String line : lines) {  
 if (line.split(Utils.*OUTPUT\_SPLITTER*)[0].equals(accountNumber)) {  
 consumer = *getConsumerFromLine*(line.split(Utils.*OUTPUT\_SPLITTER*));  
 break;  
 }  
 }  
  
 return consumer;  
}

## Get All Readings from File

//Retrieve all readings from file  
public static ArrayList<Reading> getAllReadings() throws IOException {  
 Path path = Paths.*get*(*READINGS\_FILE\_NAME*);  
 List<String> lines = Files.*readAllLines*(path, StandardCharsets.*UTF\_8*);  
 int count = 0;  
 ArrayList<Reading> readings = new ArrayList<>();  
 for (String line : lines) {  
 count++;  
 Reading reading = *getReadingsFromLine*(line.split(Utils.*OUTPUT\_SPLITTER*));  
 reading.setRecordNo(count);  
 readings.add(reading);  
 }  
  
 return readings;  
}

## Get Consumers All Readings

public static ArrayList<Reading> getAllReadingsByAccountNo(String accountNo) throws IOException {  
 Path path = Paths.*get*(*READINGS\_FILE\_NAME*);  
 List<String> lines = Files.*readAllLines*(path, StandardCharsets.*UTF\_8*);  
 int count = 0;  
 ArrayList<Reading> readings = new ArrayList<>();  
 for (String line : lines) {  
 count++;  
 Reading reading = *getReadingsFromLine*(line.split(Utils.*OUTPUT\_SPLITTER*));  
 if (reading.getConsumer().getAccountNumber().equals(accountNo)) {  
 reading.setRecordNo(count);  
 readings.add(reading);  
 }  
 }  
  
 return readings;  
}

## Create Consumer Current Month Invoice

public static void createInvoice(Reading reading, String fileName) throws FileNotFoundException, DocumentException {  
 Document document = new Document();  
  
 Consumer consumer = reading.getConsumer();  
  
 com.itextpdf.text.Font catFont = new com.itextpdf.text.Font(com.itextpdf.text.Font.FontFamily.*TIMES\_ROMAN*, 18,  
 com.itextpdf.text.Font.*BOLD*);  
 com.itextpdf.text.Font redFont = new com.itextpdf.text.Font(com.itextpdf.text.Font.FontFamily.*TIMES\_ROMAN*, 12,  
 com.itextpdf.text.Font.*NORMAL*, BaseColor.*RED*);  
 com.itextpdf.text.Font smallBold = new com.itextpdf.text.Font(com.itextpdf.text.Font.FontFamily.*TIMES\_ROMAN*, 12,  
 com.itextpdf.text.Font.*BOLD*);  
 com.itextpdf.text.Font smallFont = new com.itextpdf.text.Font(Font.FontFamily.*TIMES\_ROMAN*, 12);  
  
  
 PdfWriter.*getInstance*(document, new FileOutputStream(fileName));  
  
 document.open();  
  
 Paragraph preface = new Paragraph();  
  
 *addEmptyLine*(preface, 1);  
  
 preface.add(new Paragraph(consumer.getAccountNumber(), catFont));  
  
 *addEmptyLine*(preface, 1);  
  
 preface.add(new Paragraph(  
 "Consumer Name: " + consumer.getName(),  
 smallFont));  
  
 preface.add(new Paragraph(  
 "Consumer PhoneNo: " + consumer.getPhoneNo(),  
 smallFont));  
  
 preface.add(new Paragraph(  
 "Consumer Address: " + consumer.getCurrentAddress(),  
 smallFont));  
  
 preface.add(new Paragraph(  
 "Tariff Type: " + consumer.getEnergyTariff(),  
 smallFont));  
  
 *addEmptyLine*(preface, 1);  
  
 preface.add(new Paragraph(  
 "Open readings: " + reading.getOpeningReadings(),  
 smallFont));  
  
 preface.add(new Paragraph(  
 "Close readings: " + reading.getClosingReadings(),  
 smallFont));  
  
 preface.add(new Paragraph(  
 "Consumed Units: " + reading.getCurrentUsedUnits(),  
 smallFont));  
  
 *addEmptyLine*(preface, 1);  
  
 preface.add(new Paragraph(  
 "Cost Calculations ", smallBold));  
  
 preface.add(new Paragraph(  
 "Cost per Unit: " + reading.getCostPerUnit() + *CURRENCY\_SYMBOL*,  
 smallFont));  
  
 preface.add(new Paragraph(  
 "Energy Cost: " + reading.getCostWithoutTaxes() + *CURRENCY\_SYMBOL*,  
 smallFont));  
  
 preface.add(new Paragraph(  
 "Vat: " + Utils.*VAT* \* 100 + "%",  
 smallFont));  
  
 preface.add(new Paragraph(  
 "GST: " + Utils.*GST* \* 100 + "%",  
 smallFont));  
  
 preface.add(new Paragraph(  
 "Additional Charges: " + Utils.*ADDITIONAL\_CHARGES* \* 100 + "%",  
 smallFont));  
  
 preface.add(new Paragraph(  
 "Taxes & Additional Charges: " + reading.getTaxes() + *CURRENCY\_SYMBOL*,  
 smallFont));  
  
 preface.add(new Paragraph(  
 "Total Bill: " + reading.getCostWithoutTaxes() + " + " + reading.getTaxes() + " = " + reading.getTotalCost() + *CURRENCY\_SYMBOL*,  
 smallFont));  
  
 *addEmptyLine*(preface, 1);  
  
 SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");  
 Date date = new Date();  
  
 preface.add(new Paragraph(  
 "Bill is generated at " + formatter.format(date) + "\nNorthampton Energy Supplier ©2022",  
 redFont));  
  
 document.add(preface);  
 document.close();  
}

## Save Readings in File

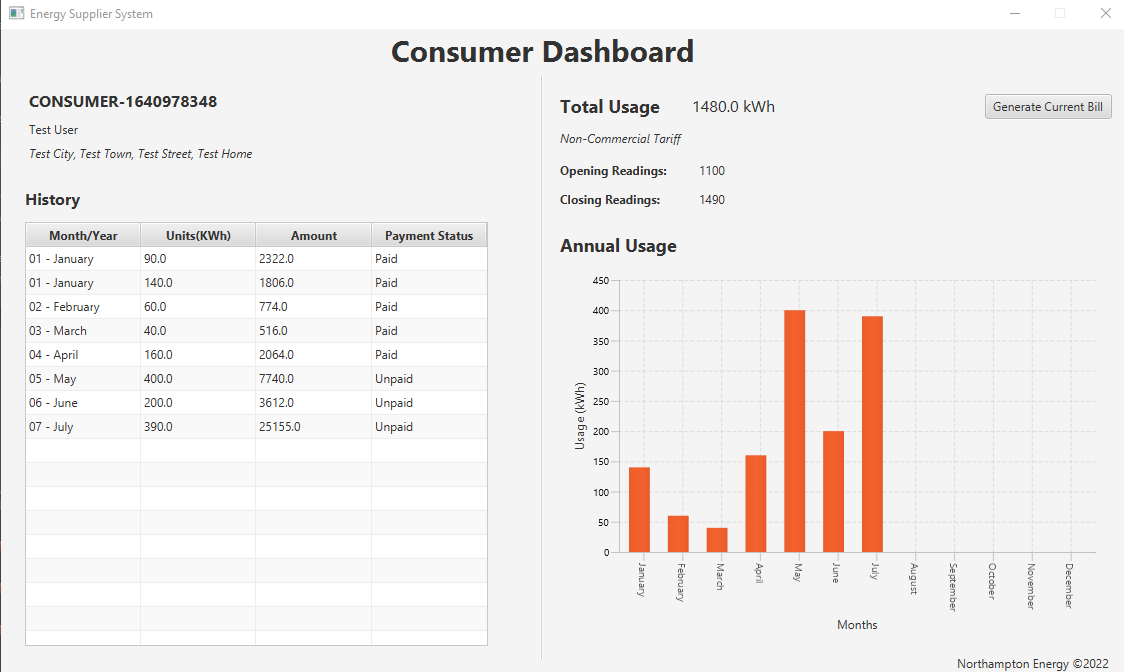
private void saveReadings(boolean isGenerateInvoice) {  
 try {  
 FileWriter fw = new FileWriter(*READINGS\_FILE\_NAME*, true);  
 PrintWriter pw = new PrintWriter(fw);  
  
 Reading reading = new Reading();  
 reading.setConsumer(Utils.*getConsumerByAccountNumber*(consumers.get(consumerIndex).getAccountNumber()));  
 reading.setMonth(monthsCombo.getValue().toString());  
 reading.setYear(yearsCombo.getValue().toString());  
 reading.setClosingReadings(closeReadingsTf.getText());  
 reading.setOpeningReadings(openReadingsTf.getText());  
 reading.setCostPerUnit(costTF.getText());  
 reading.setPaymentStatus(paymentBillCombo.getValue().toString());  
  
 String line = reading.getConsumer().getAccountNumber() + Utils.*INPUT\_SPLITTER* + reading.getMonth() + Utils.*INPUT\_SPLITTER* + reading.getYear() + Utils.*INPUT\_SPLITTER* + reading.getOpeningReadings() + Utils.*INPUT\_SPLITTER* + reading.getClosingReadings() + Utils.*INPUT\_SPLITTER* + reading.getCostPerUnit() + Utils.*INPUT\_SPLITTER* + reading.getPaymentStatus();  
  
 pw.println(line);  
  
 pw.flush();  
 pw.close();  
 fw.close();  
  
 //After Save  
 setOutputText("Success: Reading is added against account #" + consumers.get(consumerIndex).getAccountNumber());  
  
 openReadingsTf.clear();  
 closeReadingsTf.clear();  
  
 updateReadingsList();  
  
 if (isGenerateInvoice) {  
 String fileName = Utils.*generateFileName*(reading.getConsumer().getAccountNumber());  
 try {  
 Utils.*createInvoice*(reading, fileName);  
 Utils.*openFile*(fileName);  
 setOutputText("Success: Invoice is generated. and saved " + fileName);  
 } catch (Exception e) {  
 setOutputText("Error: " + e.getMessage());  
 System.*out*.println(e.getMessage());  
 e.printStackTrace();  
 }  
 }  
  
 } catch (Exception e) {  
 System.*out*.println("Error " + e.getMessage());  
 setOutputText("Error " + e.getMessage());  
 //new Alert(Alert.AlertType.ERROR, "Error " + e.getMessage()).showAndWait();  
 }  
}

## Save Consumer in File

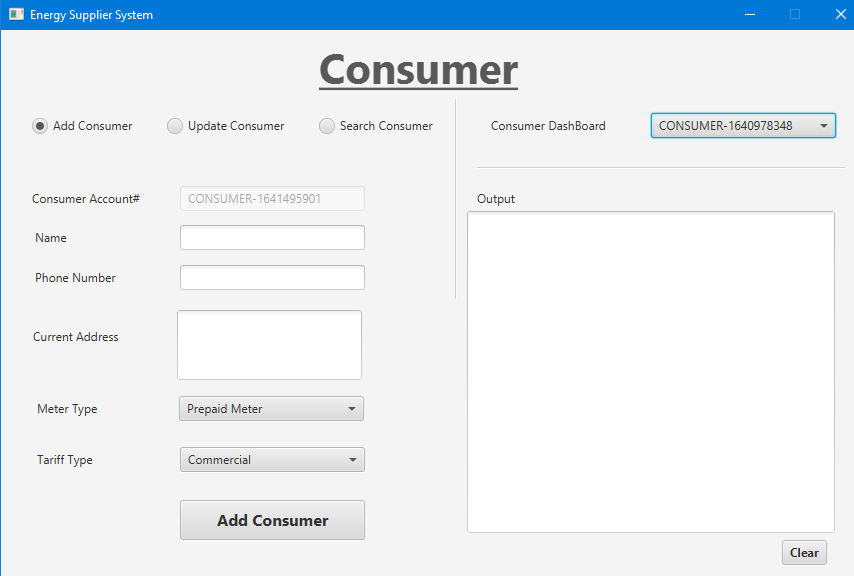
private void SaveConsumer() {  
 try {  
 FileWriter fw = new FileWriter(*CONSUMERS\_FILE\_NAME*, true);  
 PrintWriter pw = new PrintWriter(fw);  
  
 String line = accountTF.getText() + "|"  
 + nameTF.getText() + "|"  
 + phoneTf.getText() + "|"  
 + addressTF.getText() + "|"  
 + meterTypeCombo.getValue() + "|"  
 + tariffTypeCombo.getValue() + "|"  
 + java.time.LocalDateTime.*now*();  
 pw.println(line);  
  
 pw.flush();  
 pw.close();  
 fw.close();  
  
 //After Save  
 setOutputText("Success: Consumer record is Added against account #" + accountTF.getText());  
 accountTF.setText(generateRandomId());  
 disableAccountComponent(true);  
 disableComponents(false);  
  
 //tariffTypeCombo.getItems().removeAll(true)  
  
 setDefaultComoBoxValues();  
  
 nameTF.clear();  
 phoneTf.clear();  
 addressTF.clear();  
  
 loadConsumerComboBox();  
  
 } catch (Exception e) {  
 System.*out*.println("Error " + e.getMessage());  
 setOutputText("Error " + e.getMessage());  
 //new Alert(Alert.AlertType.ERROR, "Error " + e.getMessage()).showAndWait();  
 }  
}

# Screenshots of the system key features:

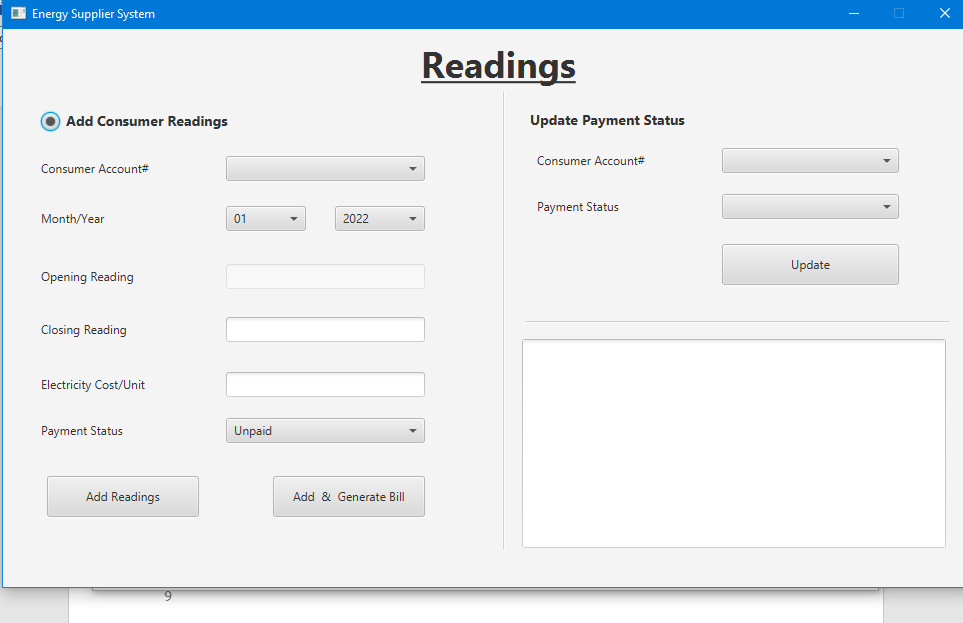
## Consumer Dashboard



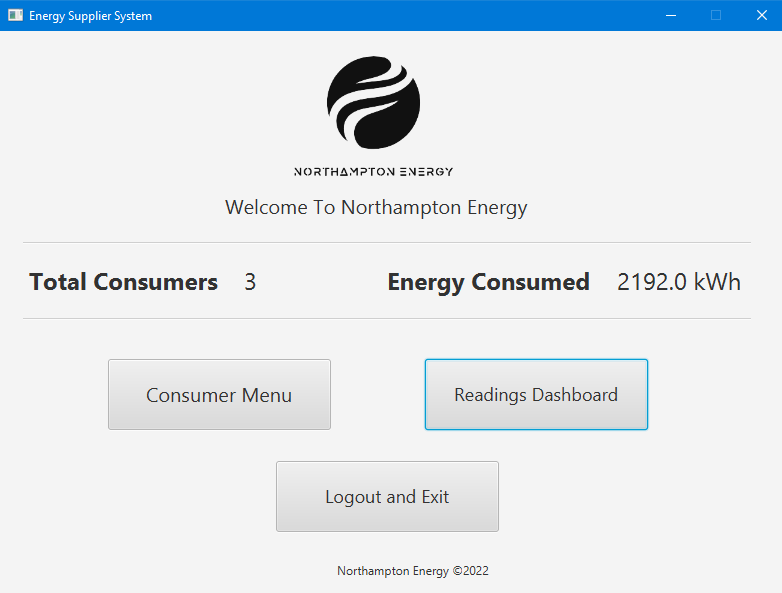
## Consumer Menu



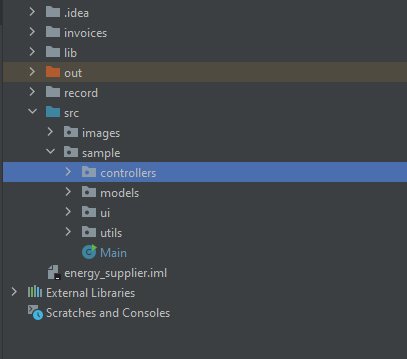
## Readings Menu



## Main Dashboard

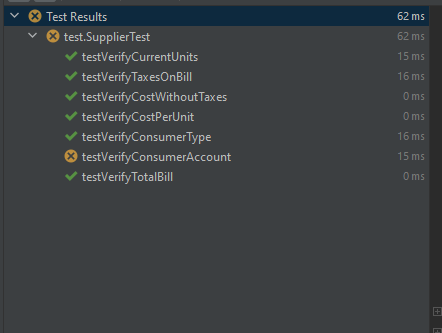


## Easily Understandable and Clean Code



# Evidence of Testing:

## 1.Black Box Testing:



## White Box Testing



## Weakness In System

Supplier can add a reading twice.

Supplier can add reading of a month multiple times against a consumer.

# References

GmbH, L., 2022. Creating PDF with Java and iText - Tutorial. [online] Vogella.com. Available at: <https://www.vogella.com/tutorials/JavaPDF/article.html> [Accessed 6 January 2022].

Tutorials.jenkov.com. 2022. [online] Available at: <http://tutorials.jenkov.com/javafx/tableview.html> [Accessed 6 January 2022].