

**DE‌ ‌LA‌ ‌SALLE‌ ‌UNIVERSITY‌ ‌**

**GOKONGWEI‌ ‌COLLEGE‌ ‌OF‌ ‌ENGINEERING‌ ‌**

**SY‌ ‌2019-2020,‌ ‌TERM‌ ‌1‌ ‌**

**‌**

**DATA‌ ‌STRUCTURE‌ ‌AND‌ ‌ALGORITM‌ ‌ANALYSIS‌ ‌LABORATORY‌ ‌**

**FINAL PROJECT ‌**

**‌**

**STUDENT‌ NAMES:‌ LLARENAS, LENARD RYAN S.**

**PORTUGAL,‌ ‌JOSEPH‌ ‌V.‌ ‌**

**STUDENT‌ ‌ID‌ ‌NUMBERS:‌ 11812508**

**‌‌11815027‌**

**STUDENT‌ ‌E-MAIL‌ ‌ADDRESSES:‌ lenard\_ryan\_llarenas@dlsu.edu.ph**

**‌‌ joseph\_portugal@dlsu.edu.ph‌ ‌**

**STUDENT‌ ‌CONTACT‌ ‌NOs.:‌ 09178715376**

**09953600070‌ ‌**

**‌**

**COURSE‌ ‌CODE:‌ ‌‌LBYCPA2‌ ‌**

**SECTION:‌ ‌‌EQ2‌ ‌ ‌**

**SCHEDULE:‌ ‌‌M‌ ‌W‌ ‌11:00‌ ‌–‌ ‌14:00‌ ‌**

**‌**

**INSTRUCTOR:‌‌ ‌ANA‌ ‌ANTONIETTE‌ ‌C.‌ ‌ILLAHI‌ ‌**

**‌**

**‌**

**‌ ‌**

**‌**

**DATE‌ ‌SUBMITTED:‌ ‌‌12 - 11 – 2019**

**CHAPTER 1**

1. Introduction

Organizing data is a broad, but a generally overlooked part when it comes to a usual business setting. The fact of the matter here is that most businesses being operated heavily rely on whatever database they need to store data into which their business deems important, be it for the purpose of record keeping, visualizing trends, budgeting purposes, organization of time (scheduling), inventory, and many other applications for the context of whatever business. Dental clinics are not an exception in this case, they are still businesses after all, given the fact that there are different settings for this-i.e., dental clinics for the hospital or private dental clinics, for the most part, handling data in the setting of dental clinics are an important aspect to a well-organized and efficient operation of that business. Record keeping and scheduling are probably the two most common data which dental clinics handle; that applies to the record of patients and their corresponding ailments, and the schedules of medical/dental appointments that the clinic and its dentist/s would expect for the foreseeable future. Not only that, but for dental clinics, or even just clinics, to look back at all of this data and see if there are any medical trends going on with the local people, as well and for budgeting purposes of said local clinic can be observed through any of this in the grand scheme of things.

1. Background of the Study

For the most part, dental clinics would operate on the use of the pen-and-paper type of record keeping, meaning that the dental clinics utilize dental exam records upon the initial checkup of the patient to hold the important medical data which the dental clinic deems necessary, dentists would also tend to use a notebook planner to write down the up and coming appointment schedules that the dentist and patient have agreed upon for their dental treatments. These are the usual records of data that dental clinics tend to operate upon. Traditional record keeping is not bad by any means on its own, but medical records, especially for clinics that have been operating for a long time, have a tendency to grow and pile up records upon records, making it difficult for the dental clinic to manage all of this data, let alone manually searching for a specific patient’s record.

1. Objective of the Study
2. General Objective

The main importance, and goal of this project is for the ease of use and quick handling of important data which proves important to dental clinics. This is to effectively replace the old pen-and-paper patient and schedule data recording and handling that most dental clinics still use to organize data up until this day. Also included here is the organized way to carry out the billing process of the clinic. The program has the capability to create, handle and manage a database of patient profiles and schedules for the use of the clinic this program is being operated into. Digital billing by the generation of a digital receipt file which can be printed is also included.

1. Specific Objective

* To be able to create patient profiles for the patients of the clinic the program is being used on. A patient profile includes the patient’s information which will prove to be important information for the clinic’s record-keeping; this information includes the patient’s name, gender, age group, and all the recorded dental ailments the patients possess.
* To be able to edit and delete any patient profile, if the program user wishes to modify or remove any profile in the patients’ database, respectively. Searching for a patient in the database to be able to view and evaluate their profile is an objective as well.
* To be able to compose an appointment schedule framework. This data organization as well as its key operations has to be user-friendly, reliable, and visually intelligible and coherent.
* To be able to search the scheduled appointment/s of a patient using the said framework.
* To bill out any finished appointment on the list of unfinished existing appointments in an organized and well visualized manner, meaning everything of that service - from the name of the patient treated, to the treatment done to said patient, to the date and estimated times of the operation, and finally the total bill of the whole operation is presented to the program user. This function of the program also includes the digital billing of that operation by saving a .pdf file of a digital receipt which can be printed if the program user wishes to do so.
* Finally, to reflect upon in an excel file, the consolidated records of all the patient profiles and appointments of that clinic in which the program is being operated into. A laid out data of all things important for the dental clinic, which can be utilized by the program to save all data, and then be loaded upon the next run of the program if, in fact, the program is exited and resumed again at a later time.

This consolidated “data of all things” can be utilized by the dental clinic for various purposes, such as, but not limited to, knowing certain dental/medical trends in patients in the local consensus (even by narrowing it down to trends in the local male or female population, age group, time frame, or a combination of any), budgeting purposes, how well the business is performing, any many other applications.

1. Scope and Delimitation
2. Scope

The scope of this project is the design and generation of a program generally for the use of dental clinics. This will effectively help the dental clinic and its dentist/s possess digital record keeping, specifically patient records and their schedule plan. Another functionality of the program created is to generate digital billing for the patients. The program is set to provide functions to create a patient profile, edit and delete a patient, and search for a specific patient. Creating schedules, specifically, dental appointment schedules are also included; as well as the searching of said schedule. Digital billing via outputting a receipt file, as mentioned, is also a feature of this program. All of the data handling will be made possible by the linked list data structure.

1. Delimitation

This system is only limited for the use of dental clinics only, and not for other medical applications, unless any further program modification will be involved. Another thing to consider is the fact that any data produced by the program is data only limited to the clinic in which the program is being operated into, understanding any dental trends based on this consolidated data set in the community can be accurate, but at the same time, be proven inconsistent since a variation of different data may vary from dental clinic to dental clinic.

1. Significance of the Study

Nowadays, dental clinics use non-electronic methods when it comes to storing patient records as well as other service related data. This process can be unreliable at times due to the possibility of human error. When it comes to obtaining data, it can also be a tedious process since a practitioner has to go through each file individually before acquiring the desired record. Besides this, storing information in files can be unsafe since it can be directly accessed through tangible means. Because of these factors, this project can provide a more superior approach in ways that non-computerized systems have disadvantages. An electronic system of data management is more reliable since its processes are affixed and consistent.

Hence, dental practitioners will benefit from this project. They can use the application in order to store, obtain, and organize patient information. In this manner, it would be much more convenient to them since they no longer need to undergo a lengthy and demanding process. Furthermore, they can also use this program to organize schedules for their patients’ appointments. In this fashion, weekly plans can be more easily managed and visually systematic.

1. Conceptual Framework



**CHAPTER 2**

1. Review of Related Literature/Studies

**Dental Records: An Overview**

According to BK Charangowda (2010), a dental record is a comprehensive document of a patient’s history of illnesses, physical examination, diagnosis, treatment, and management. Professionals in the field, such as dentists, are urged to form and preserve patient records. Because of the increased public consciousness surrounding healthcare especially at its legal and competency issues, a detailed proficiency of dental record approach is essential for any dental professional. Aside from being a legal obligation, maintaining reliable dental records is significant to achieve exemplary patient care. Dental records help provide consistency of care for the patient. For practitioners, it also serves as a principle for effective clinical practice and as an important skill as professionals.

Based on the study, the data in the dental record must principally be clinical in essence. The record comprises of a registration form that contains all the rudimentary personal information. The dental practitioners should be scrupulous when it comes to record keeping. The information should be precisely documented and must not be vague. The following are sample data of what may be filed in the dental record:

* Identification
* Dental history
* Clinical examination
* Diagnosis
* Treatment plan
* Medical History

When it comes to the maintenance of dental records, many dentists nowadays utilize computerized documentation systems. Electronic records are effective and offer patient-safety advantages. Traditionally, patient records are identified with the patient’s surname, first name, and middle name. These files are then organized in a manner that they can be retrieved easily (Charangowda, 2010). Such a process can also be applied for computerized medium.

**The importance of dental records in general and forensic dentistry**

A properly managed dental record has different benefits. In general, these records are significant for the provision of effective dental care as well as for attaining guaranteed continuity and success of treatment. Good management of these records allow for the supervision of a patient’s current oral health condition and can also be used to help in precautionary oral healthcare processes. It can also help in tracking whether a treatment is successful or not. For the practitioner, a comprehensive dental record may also be used as a defense in allegations of malpractice.

Dental records also serve as a means of communication between the two different practitioners in order to maintain continuity despite the absence of the primary clinician. Detailed records presents the adequate information needed to grant another practitioner with enough knowledge about the patient’s dental experience.

Besides these, dental records are also important for dental audit, which is very integral in maintaining control of quality. A dental audit thoroughly scrutinizes each element of dental care. It considers the patient information and the competency of the dental practitioner when it comes to diagnosing, treating, using resources, and practicing evidence-based dentistry. Such factors affect the quality of service evaluated by the patient and the professional.

Furthermore, dental records may be utilized for managing and organizing healthcare facilities and services, as well as for medical research and the establishment of health care statistics.

Lastly, an individual’s dental record has a crucial function in forensic dentistry. It is needed for the identification, discernment, and solution of a crime in civil accounts or in disaster situations (Devadiga, 2014).

All of these determinants demonstrate the need for a properly managed dental record system.

**CHAPTER 3**

1. Methodology

The generation of the program as a whole was made possible by the Java programming language. This will include the use of the JavaFX software platform, and the use of Scene Builder, in order to generate the user interface of the program. The IntelliJ IDEA IDE is the Integrated development environment, as well as the program’s compiler. The Java libraries utilized within the program are Apache POI, iText, and JFoenix. Apache POI is a Java API (application programming interface) for Microsoft Office documents (such as, but not limited to: Word, Excel, Visio, and Publisher), meaning that the programmer may read, write, and edit the files of the various Microsoft Office formats using the Java programming language. The specific Apache POI class that was heavily used throughout the creation process of the program is the Apache POI XSSF, this is a pure Java implementation of Microsoft Excel will make the Java-written program be able to read and write .xlsx/XML spreadsheet format Microsoft Excel files. The second library used is iText, which is a library that manipulates PDF files using the Java programming language, meaning this library will allow the programmer to read, write, and edit files of the .pdf file format. The last library utilized is the JFoenix Java library, which is the implementation of the Google Material Design using JavaFX-based components; this library would enhance the program’s visuals, such as better looking and interacting buttons, checkboxes, radio buttons, and text fields, these are some basic JavaFX elements, only that these elements give a more modern, minimalist, and clean feel into them. The data structure that the program heavily relies on in the context of data organization, management, and storage is the linked list data structure.

1. Software Design

As mentioned the graphical user interface is generated via the JavaFX software platform, as well as Scene Builder to create the main visual user interface itself (the .fxml file). The program relies on scene switching to access the next set of functions for the program user to have access to, for example, the main menu will ask the user to enter an operation in which the program can switch between ‘.fxml’ scenes depending on the user’s selection.

The first function, registering patient data, is done by inputting the patient’s name, gender, the age group of the patient, followed by specifying which dental diseases where observed after the initial checkup, with this information, the program creates a patient node object which holds all of this data and pushes it to the back of the linked list. Finally, this displays all of the corresponding information of the patient to the excel file in the appropriate row (a new patient means data stored will be placed at the latest row). In line with this, editing and deleting any patient information is possible. Searching for a patient is done by searching for the specific patients name (case insensitive), and matches this name with the different names that the program encounters through a linked list traversal, if matched accordingly, the next scene will initialize all of the text values there to display all of the information to the user correspondingly. Editing the patient as the user searches for any specific patient name, and lays out the entire patient information with the values set up as it appears in the patient profile, and all the user has to do is to edit whatever information the user wishes to edit, this will then make the program traverse and overwrite this information to the corresponding linked list node, and edit the corresponding row of the patient profile in the excel file as well. Deleting is the same thing, as the user enters the corresponding name, it will ask the user again if they are sure to delete this patient from the database, this will also modify the linked list to delete the user, and modify the excel file accordingly as well. In all of these functions, searching for a name that doesn't exist will not allow the user to proceed to the next scene, and instead prompt an error instead.

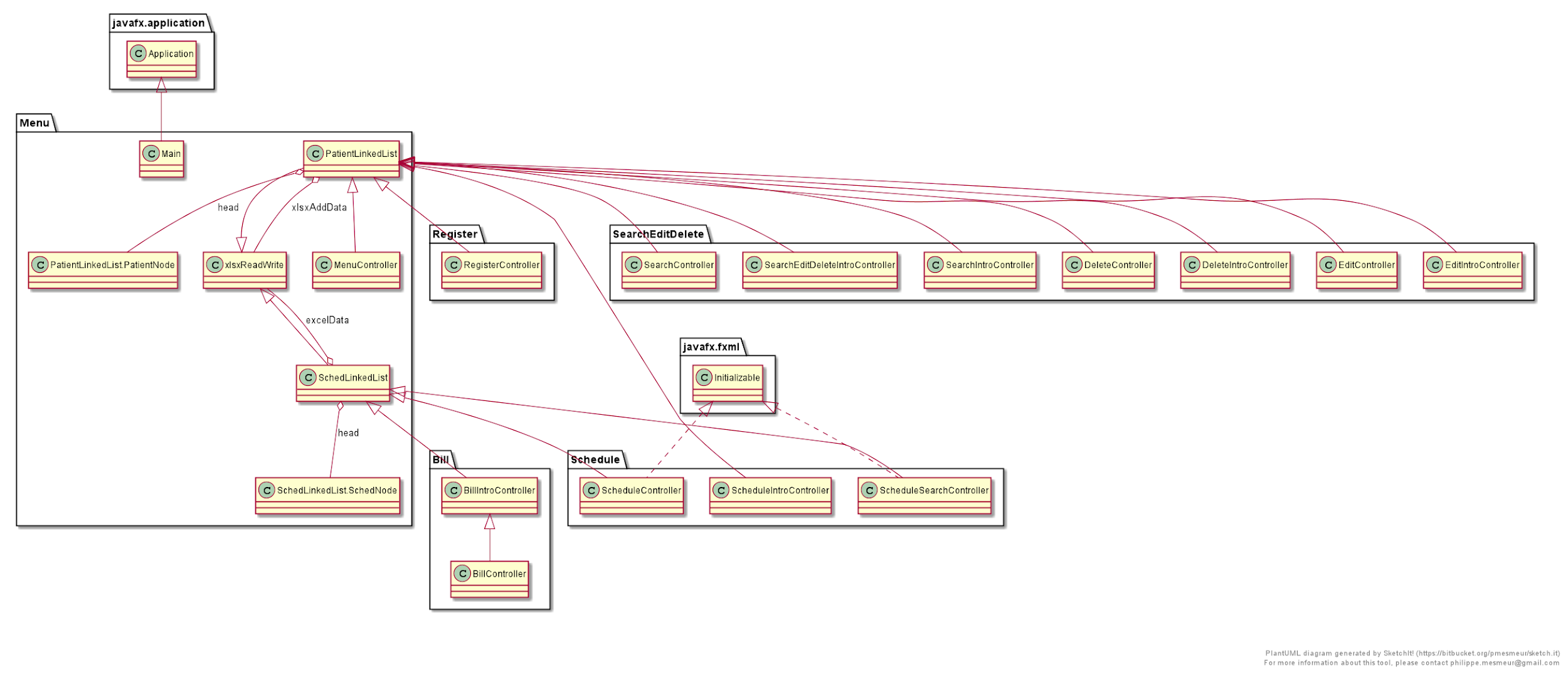
The next set of functions rely on the “scheduling of appointments” portion of the program. Before scheduling, the user is asked if the patient to make and appointment is a new or old patient, if the patient is new the program will return the user to the scene of patient creation, if the patient is old, it will proceed to the scene where the actual scheduling takes place. The user may input the information accordingly to how the user wishes to generate their appointment into the appointment calendar. This information will then be added to a separate linked list for the use of schedules this time, as well as be added to the excel file accordingly as well. If the user wishes to search for a patient’s scheduled appointment, there is a separate function for that. For this search schedule function, the user has to input the name of the patient he/she wishes to search for. Once inputted, the program will perform a depth-first search of the GridPane timetable. In this operation, each node of the GridPane will be visited and will be pushed to a predefined stack. Once there are no more available nodes to visit, the top of the stack will be popped and will return the node at the top. If this node contains a VBox which contains the inputted patient’s name, then the background color of the VBox will be set to yellow. Therefore, it highlights the scheduled appointment/s of the said patient.

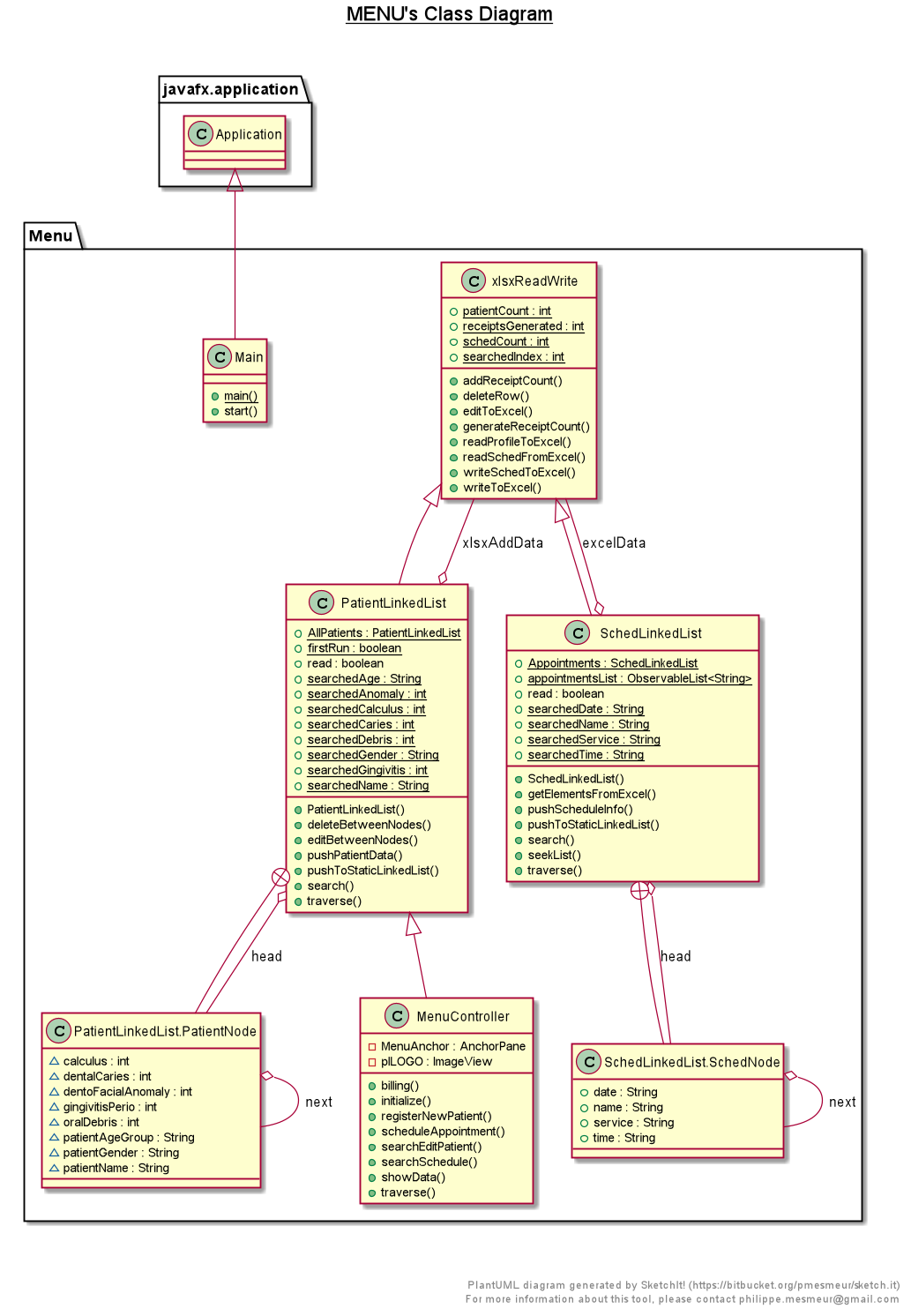
Billing is the final function that the user has access to. In billing, the program loads the linked list of appointments and displays them in a drop down menu for the user to select a corresponding appointment to bill. The user will also then have to input the price of their service and the quantity of that specific service; the price of the service, which is user inputted, since it is much better if a private dental clinic would be flexible in charging the patient for the service done to them. Failing to load a schedule in the menu of schedules will not let the user proceed, as there is nothing to bill in the first place. After determining which service to bill, the next scene would initialize text values which is the final billing is displayed to the user, which includes all necessary information for the billing process. The user may save this information and be generated as a PDF file, which acts as the digital receipt (pdf, which may also be printed) for the dental service performed.

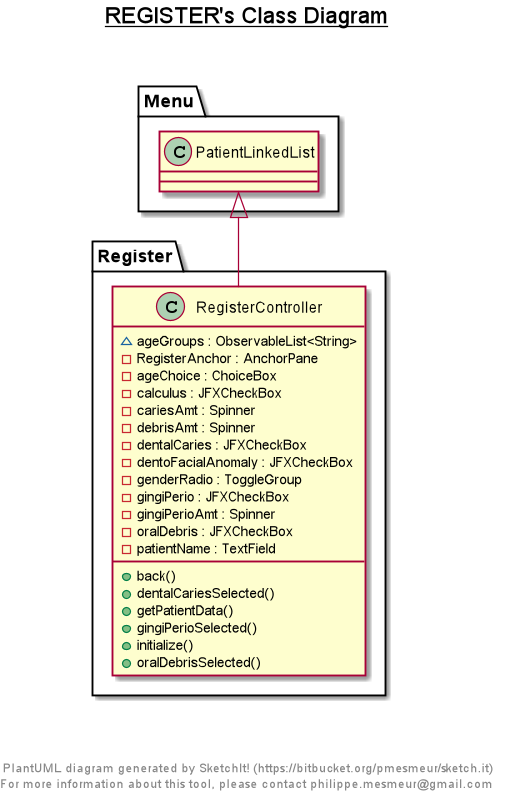
Do note that when the program is closed and then opened again, all the necessary data that exists will be initialized back to the corresponding patient and schedule linked lists, for the program to work with once again. This feature was made possible by loading back all of the data from the excel file to the program itself, every time the program is initialized.

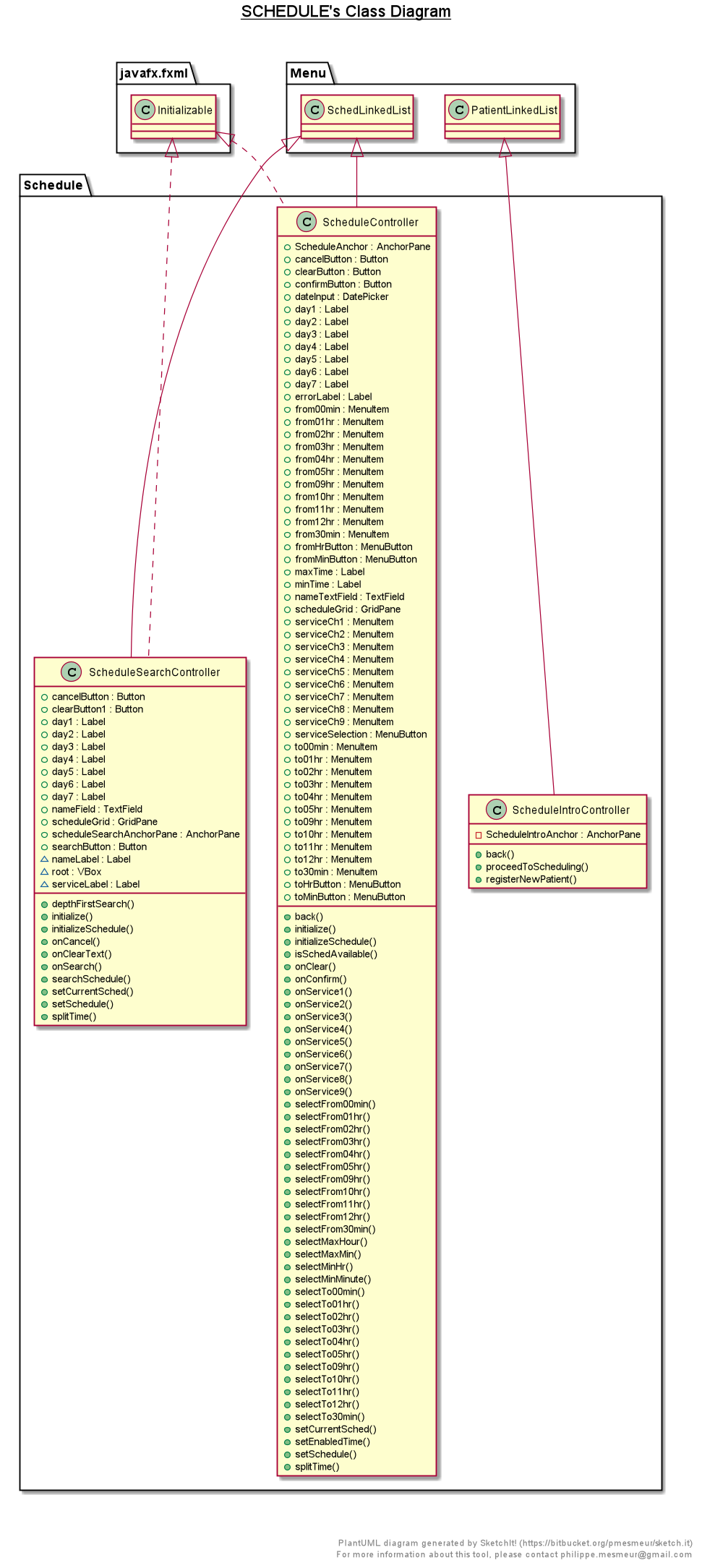
1. Flow Chart

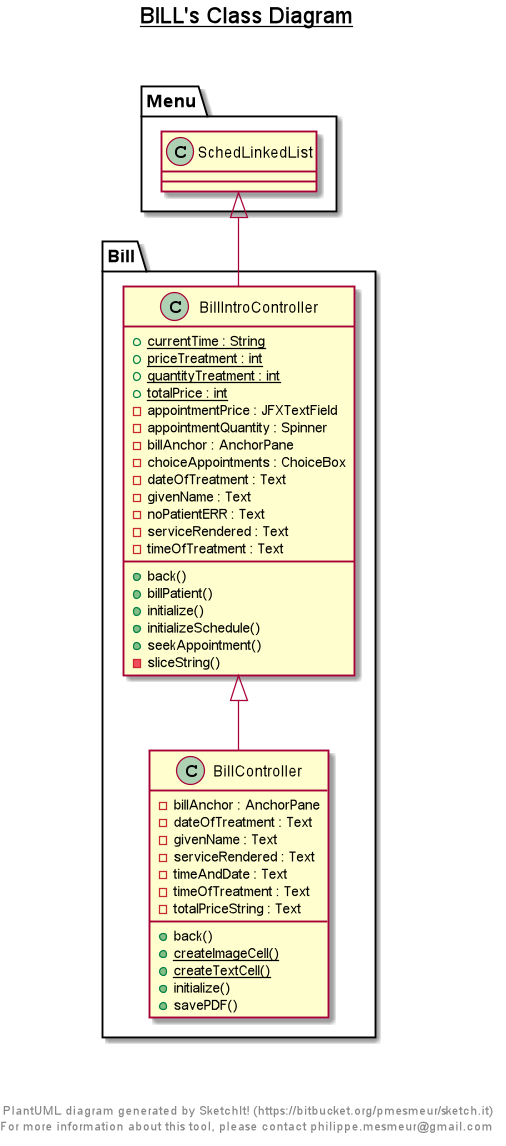
General Class Diagram



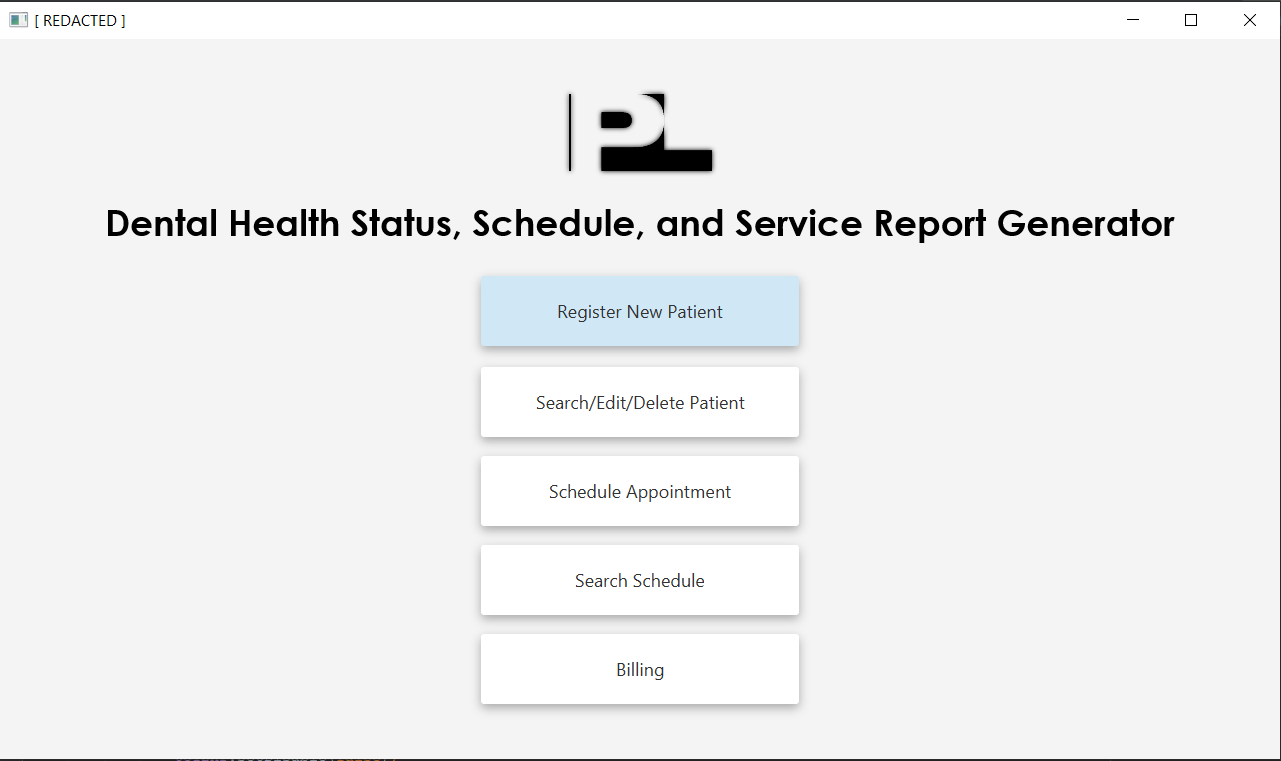




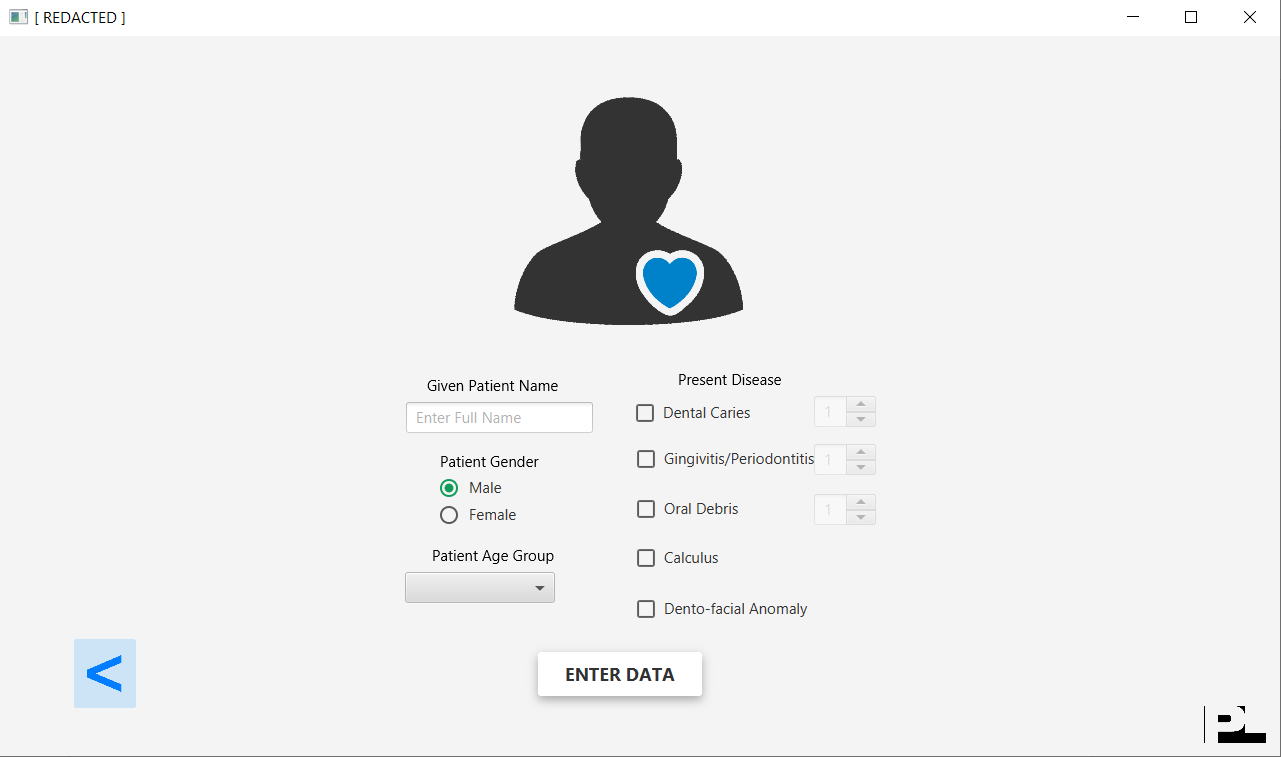




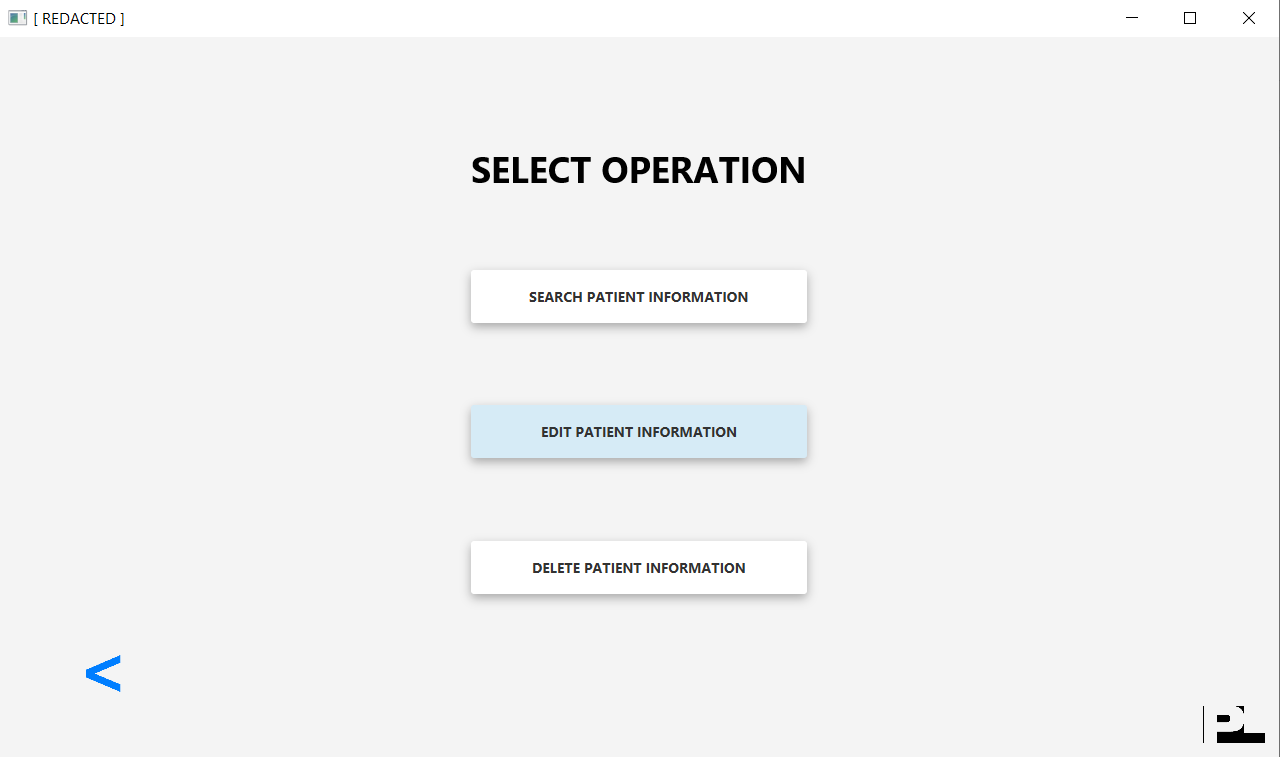
1. Screenshot of the system/program



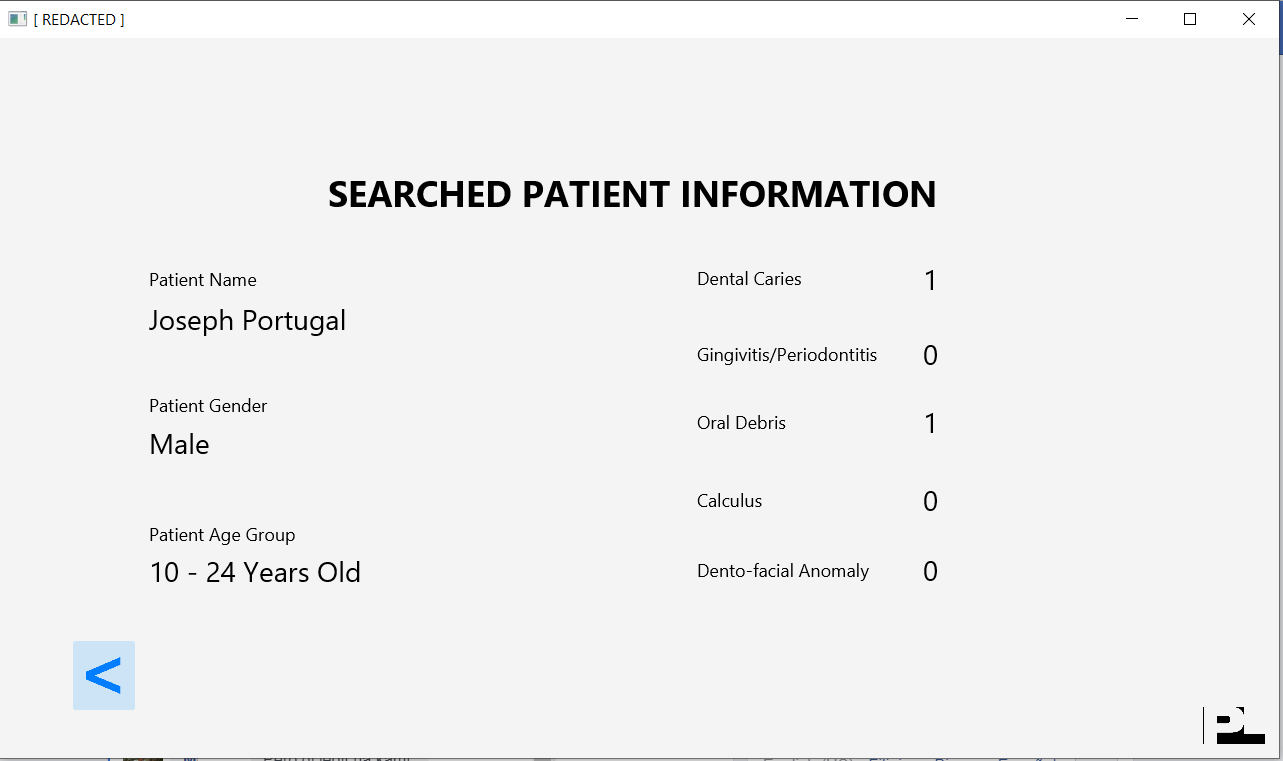
Main Menu



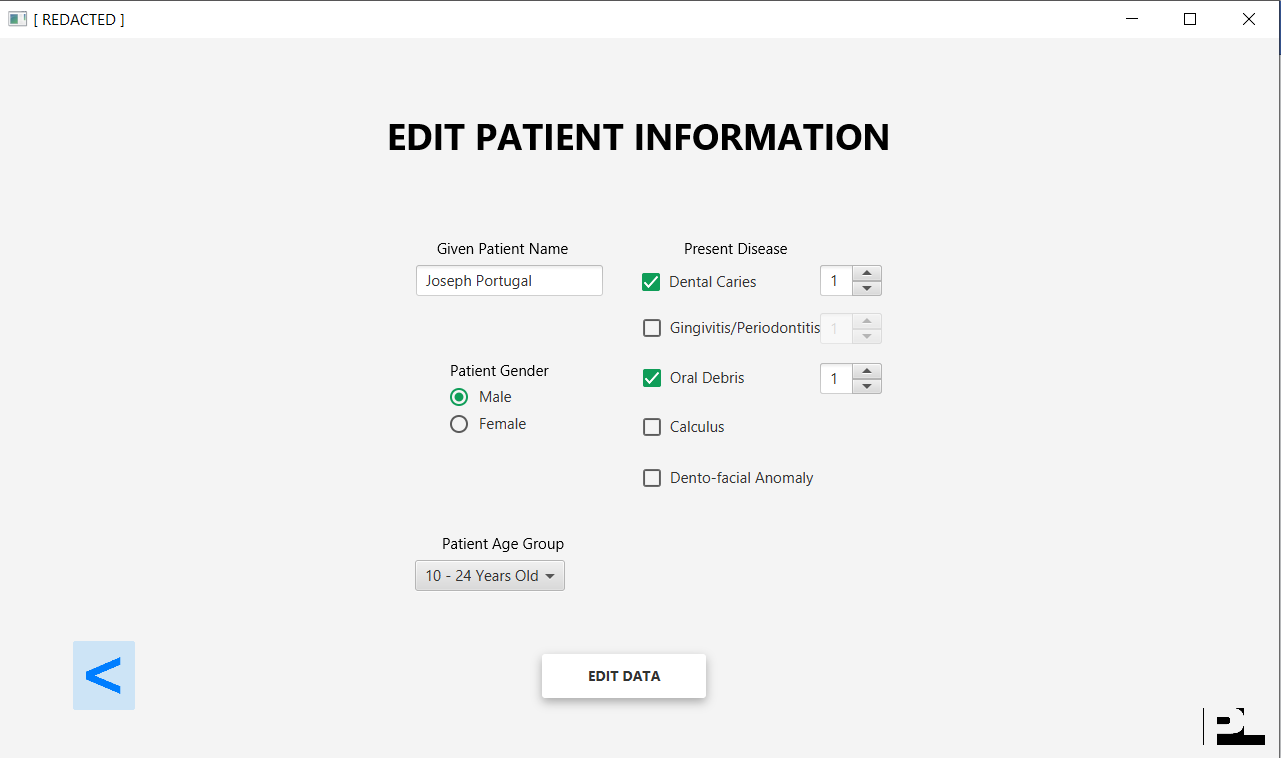
Register Patient



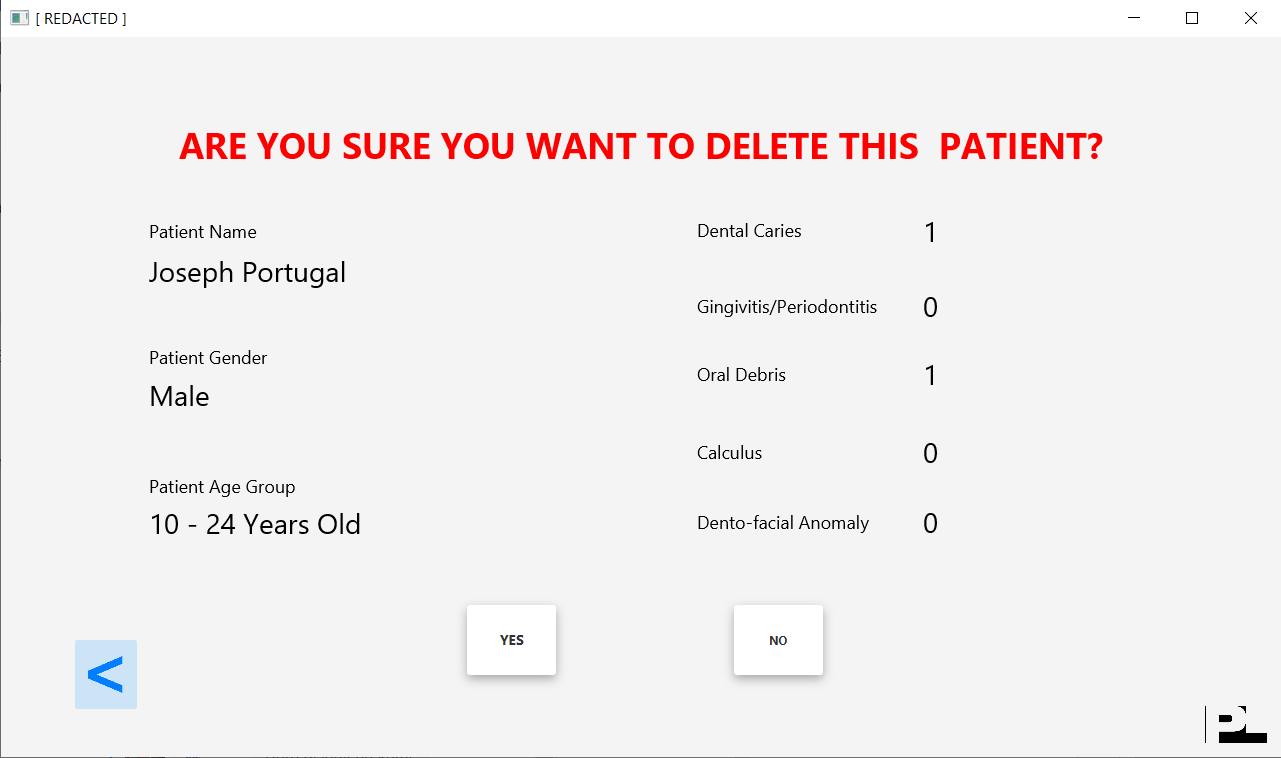
Search/Edit/Delete Patient Profile



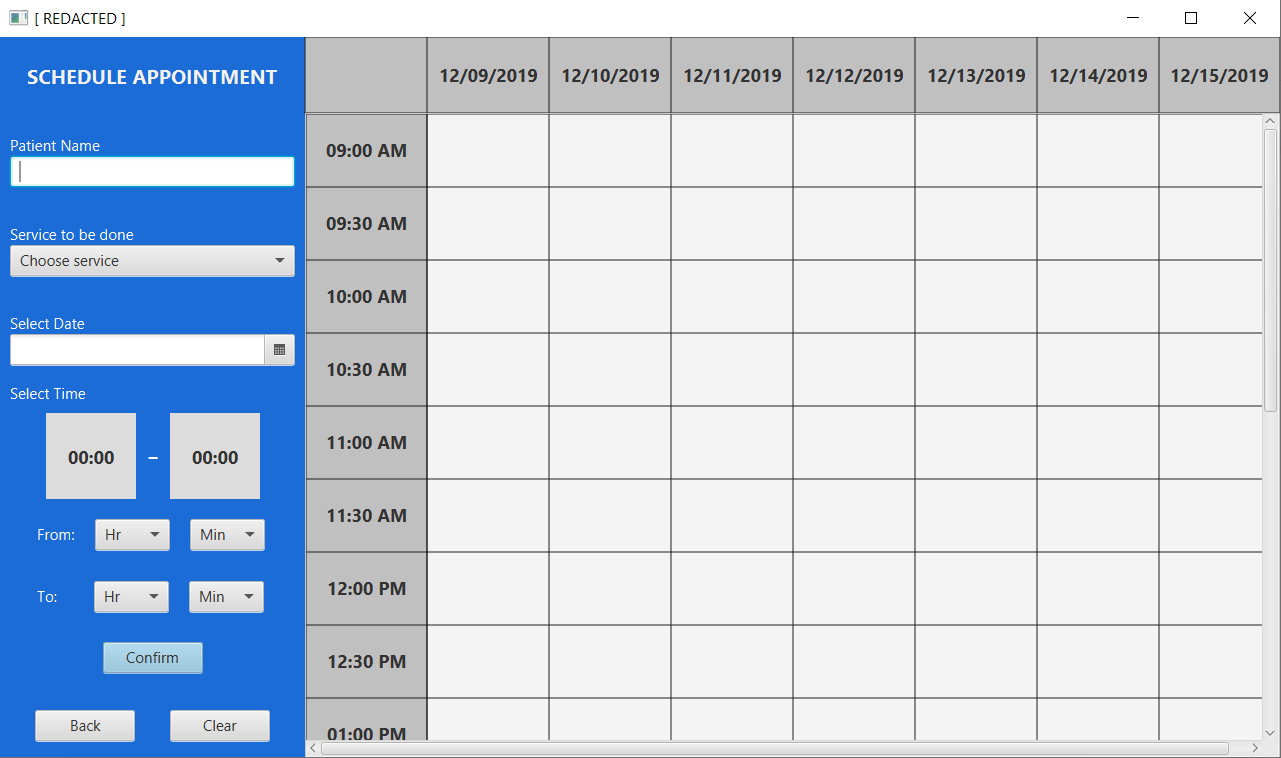
Search Patient Profile



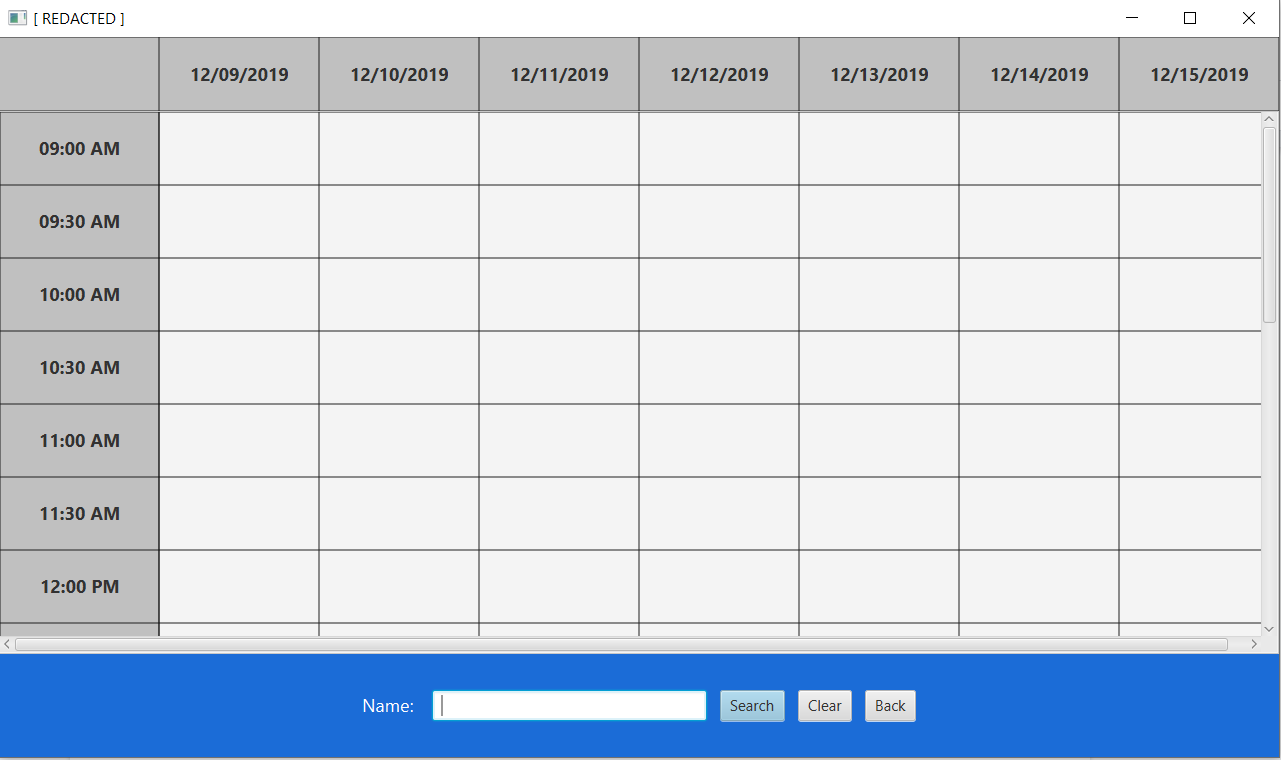
Edit Patient Profile



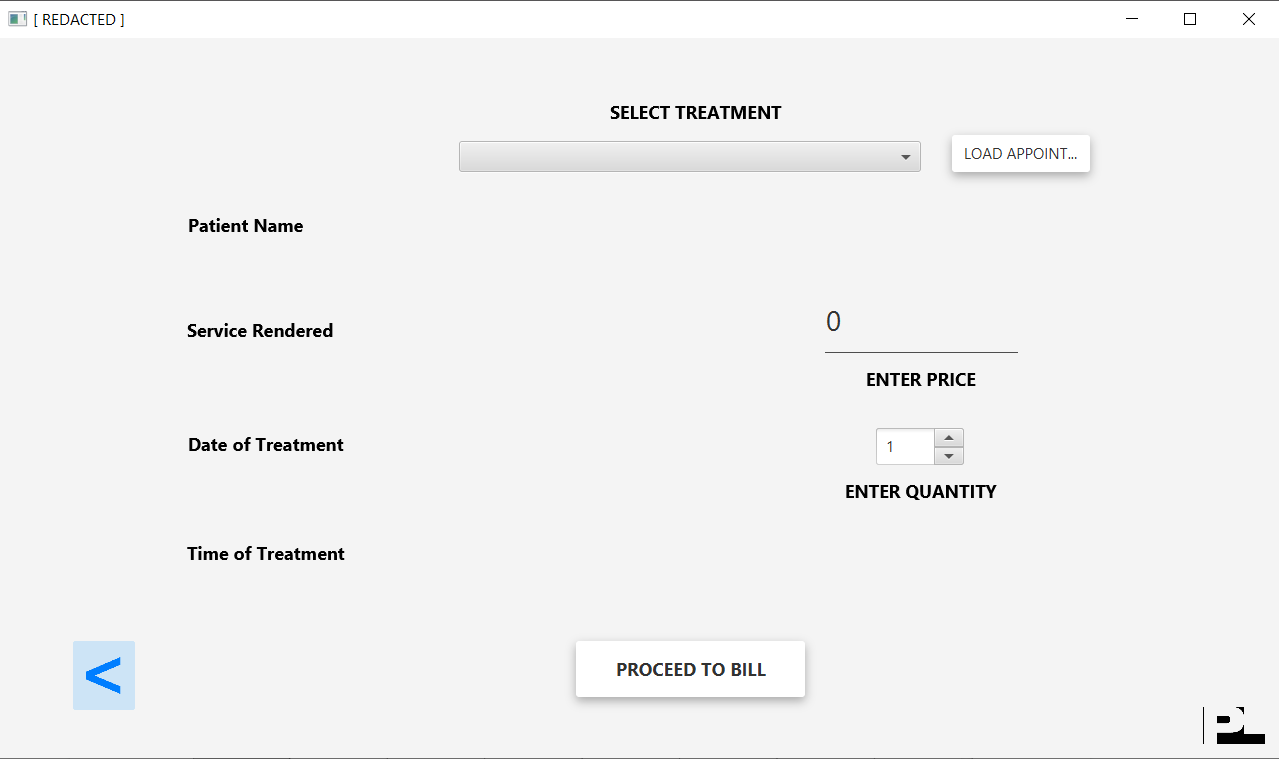
Delete Patient Profile



Schedule Appointment



Search Scheduled Appointment



Billing

**CHAPTER 4**

1. Result and Discussion

Overall, the program works as intended, and all of the features that were expected to be in the program were definitely present, and even more. The feature to add, edit, delete, and search for a patient functions appropriately. Scheduling was visually organized and appealing with the calendar schedule display feature for each appointment added. Linked list interaction ran properly too, as well as using the correct functions to modify and read the excel file according to the programs needs, no errors nor any unexpected spreadsheet inconsistencies were found in this aspect of the program runtime. Billing functioned well as organized and displayed accurately, the feature to print and save digital receipts was not initially planned, but was a welcome addition to the program. Error trapping was taken into mind here, as common errors such as no input and duplicate data where checked to post errors and stopped the program from proceeding any further as these happened during the runtime of the program. All of the program features where done with the dentist in mind here, as the programmers took the feedback of professional dentists, and applied it to cater the needs of a dentist to a program. The end result is a fully functioning program, which is known as a “Dental Health Status, Scheduler, and Service Report Generator”.

**CHAPTER 5**

1. Conclusion and Recommendation

In this project, a Dental Health Status, Scheduler, and Service Report Generator application was produced. The researchers were able to utilize the Java programming language as well as the JavaFX platform to generate a program that involves a graphical user interface and apply key data management operations. The program is user-friendly and visually intelligible in design. Moreover, it is also straightforward in its processes. The storing of its data can also be easily accessed whether in Microsoft Excel or in the program itself. Thus, all the operations of data management are made convenient for the user.

Although partial from being flawless, dental practitioners can still use this program as a means to store patients’ basic personal and dental information. It can store all the fundamental medical data needed for a practitioner to be knowledgeable of the patient’s current status. Furthermore, the program can also be used to manage a timetable of the scheduled dental appointments of every patient, which can make it convenient for the practitioner to handle weekly plans.

In line with these, there are still improvements that can be implemented in the program. As said before, the program is still not flawless. It may contain loopholes still unseen upon the conclusion of the project. Hence, these errors can be resolved when they will have been realized and solutions will then be produced. Aside from this, additional operations can also be appended in the program. As stated in Chapter 2, dental records are not only limited to one’s personal information and current dental status. Among many others, clinical examination, dental history, and treatment plan can also be included as well. Hence, this program is not only limited to its present functions, but can still be improved and augmented, as is for any other application in the technological frontier.

**REFERENCES**

Charangowda B. K. (2010). Dental records: An overview. *Journal of forensic dental sciences,*

*2*(1), 5–10. doi:10.4103/0974-2948.71050

Devadiga A. (2014). What's the deal with dental records for practicing dentists? Importance in

general and forensic dentistry. *Journal of forensic dental sciences, 6*(1), 9–15. doi:10.4103/0975-1475.127764

**APPENDICES**

1. Program Codes

**Menu.java**

package Menu;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Stage;

public class Main extends Application {

@Override

public void start(Stage primaryStage) throws Exception{

Parent root = FXMLLoader.load(getClass().getResource("MenuUI.fxml"));

primaryStage.setTitle("PL Dental Health Status, Scheduler, and Service Report Generator");

primaryStage.setScene(new Scene(root, 1280, 720));

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

**MenuController.java**

package Menu;

import javafx.animation.FadeTransition;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.image.ImageView;

import javafx.scene.layout.AnchorPane;

import javafx.util.Duration;

import java.io.IOException;

public class MenuController extends PatientLinkedList{

@FXML

private AnchorPane MenuAnchor;

@FXML

private ImageView plLOGO;

public void initialize() throws IOException {

if(firstRun == true){

readProfileToExcel();

generateReceiptCount();

}

firstRun = false;

FadeTransition fadeInMark = new FadeTransition(Duration.seconds(0.7), plLOGO);

fadeInMark.setFromValue(0.0);

fadeInMark.setToValue(1.0);

fadeInMark.play();

}

public void registerNewPatient(ActionEvent actionEvent) throws IOException {

AnchorPane nextAnchorPane = FXMLLoader.load(getClass().getResource("../Register/RegisterUI.fxml"));

MenuAnchor.getChildren().setAll(nextAnchorPane);

}

public void billing(ActionEvent actionEvent) throws IOException {

AnchorPane nextAnchorPane = FXMLLoader.load(getClass().getResource("../Bill/BillIntroUI.fxml"));

MenuAnchor.getChildren().setAll(nextAnchorPane);

}

public void searchEditPatient(ActionEvent actionEvent) throws IOException {

AnchorPane nextAnchorPane = FXMLLoader.load(getClass().getResource("../SearchEditDelete/SearchEditDeleteIntroUI.fxml"));

MenuAnchor.getChildren().setAll(nextAnchorPane);

}

public void scheduleAppointment(ActionEvent actionEvent) throws IOException {

AnchorPane nextAnchorPane = FXMLLoader.load(getClass().getResource("../Schedule/ScheduleIntroUI.fxml"));

MenuAnchor.getChildren().setAll(nextAnchorPane);

}

public void searchSchedule(ActionEvent actionEvent) throws IOException {

AnchorPane nextAnchorPane = FXMLLoader.load(getClass().getResource("../Schedule/ScheduleSearchUI.fxml"));

MenuAnchor.getChildren().setAll(nextAnchorPane);

}

public void traverse(ActionEvent actionEvent) throws IOException {

AllPatients.traverse();

}

public void showData(ActionEvent actionEvent){

AllPatients.traverse();

}

}

**PatientLinkedList.java**

package Menu;

import java.io.IOException;

public class PatientLinkedList<T> extends xlsxReadWrite{

private xlsxReadWrite xlsxAddData = new xlsxReadWrite();

public boolean read = false;

public static boolean firstRun = true;

public static String searchedName = "";

public static String searchedGender = "";

public static String searchedAge = "";

public static int searchedCaries = 0;

public static int searchedGingivitis = 0;

public static int searchedDebris = 0;

public static int searchedCalculus = 0;

public static int searchedAnomaly = 0;

private class PatientNode {

String patientName;

String patientGender;

String patientAgeGroup;

int dentalCaries;

int gingivitisPerio;

int oralDebris;

int calculus;

int dentoFacialAnomaly;

PatientNode next;

}

PatientNode head;

public PatientLinkedList() {

this.head = null;

}

public void pushPatientData(String name, String gender, String ageGroup, int dc, int gp, int od, int c, int dfa) throws IOException {

PatientNode tail = new PatientNode();

PatientNode temp = new PatientNode();

tail.patientName = name;

tail.patientGender = gender;

tail.patientAgeGroup = ageGroup;

tail.dentalCaries = dc;

tail.gingivitisPerio = gp;

tail.oralDebris = od;

tail.calculus = c;

tail.dentoFacialAnomaly = dfa;

tail.next = null;

if (head == null) {

head = tail;

}

else {

temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = tail;

}

patientCount = patientCount + 1;

if(!read){

xlsxAddData.writeToExcel(name, gender, ageGroup, dc, gp, od, c, dfa);

}

return;

}

public void traverse(){

PatientNode traverse = head;

for(int i = 0; i < patientCount-1; i++) {

System.out.println("\t\t| " + traverse.patientName + " - " + traverse.patientGender + " - " + traverse.patientAgeGroup + " |");

traverse = traverse.next;

}

}

public void editBetweenNodes(String name, String gender, String ageGroup, int dc, int gp, int od, int c, int dfa) throws IOException {

PatientNode traverse = head;

if (searchedIndex > 0) {

for (int i = 1; i < searchedIndex; i++) {

traverse = traverse.next;

}

traverse.patientName = name;

traverse.patientGender = gender;

traverse.patientAgeGroup = ageGroup;

traverse.dentalCaries = dc;

traverse.gingivitisPerio = gp;

traverse.oralDebris = od;

traverse.calculus = c;

traverse.dentoFacialAnomaly = dfa;

xlsxAddData.editToExcel(name, gender, ageGroup, dc, gp, od, c, dfa);

}

return;

}

public void deleteBetweenNodes() throws IOException {

PatientNode body = new PatientNode();

PatientNode traverse = head;

if (searchedIndex == 1) {

head = head.next;

patientCount = patientCount - 1;

xlsxAddData.deleteRow();

}

else if (searchedIndex > 1) {

for (int i = 1; i < searchedIndex; i++) {

body = traverse;

traverse = traverse.next;

}

body.next = traverse.next;

patientCount = patientCount - 1;

xlsxAddData.deleteRow();

}

return;

}

public void search(String name){

searchedIndex = 0;

searchedName = "";

searchedGender = "";

searchedAge = "";

PatientNode traverse = head;

for(int i = 0; i < patientCount-1; i++) {

if(traverse.patientName.toLowerCase().equals(name.toLowerCase())){

searchedIndex = i + 1;

searchedName = traverse.patientName;

searchedGender = traverse.patientGender;

searchedAge = traverse.patientAgeGroup;

searchedCaries = traverse.dentalCaries;

searchedGingivitis = traverse.gingivitisPerio;

searchedDebris = traverse.oralDebris;

searchedCalculus = traverse.calculus;

searchedAnomaly = traverse.dentoFacialAnomaly;

}

traverse = traverse.next;

}

}

public void pushToStaticLinkedList(String name, String gender, String ageGroup, int dc, int gp, int od, int c, int dfa) throws IOException{

AllPatients.read = true;

AllPatients.pushPatientData(name, gender, ageGroup, dc, gp, od, c, dfa);

}

public static PatientLinkedList AllPatients = new PatientLinkedList();

}

**SchedLinkedList.java**

package Menu;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import java.io.IOException;

public class SchedLinkedList<T> extends xlsxReadWrite{

private xlsxReadWrite excelData = new xlsxReadWrite();

public boolean read = false;

public static ObservableList<String> appointmentsList = FXCollections.observableArrayList();

public static String searchedName = "";

public static String searchedService = "";

public static String searchedDate = "";

public static String searchedTime = "";

public class SchedNode{

public String name;

public String service;

public String date;

public String time;

public SchedNode next;

}

public SchedNode head;

public SchedLinkedList(){

this.head = null;

}

public void search(String name){

searchedName = "";

searchedService = "";

searchedDate = "";

searchedTime = "";

SchedNode traverse = head;

while(traverse != null){

if(traverse.name.equals(name)){

searchedName = traverse.name;

searchedService = traverse.service;

searchedDate = traverse.date;

searchedTime = traverse.time;

}

traverse = traverse.next;

}

}

public void seekList(){

String scheduleNode;

SchedNode curr = head;

while(curr!=null) {

scheduleNode = (curr.name + " - " + curr.service + " - " + curr.date + " - " + curr.time);

appointmentsList.add(scheduleNode);

curr = curr.next;

}

}

public void pushScheduleInfo(String patientName, String srvc, String aptDate, String aptTime) throws IOException {

SchedNode tail = new SchedNode();

SchedNode current = new SchedNode();

tail.name = patientName;

tail.service = srvc;

tail.date = aptDate;

tail.time = aptTime;

if (head==null){

head = tail;

}

else{

current = head;

while (current.next!=null){

current = current.next;

}

current.next=tail;

}

traverse();

schedCount++;

if(!read){

excelData.writeSchedToExcel(patientName, srvc, aptDate, aptTime);

}

return;

}

public void pushToStaticLinkedList(String patientName, String srvc, String aptDate, String aptTime) throws IOException{

Appointments.read=true;

Appointments.pushScheduleInfo(patientName, srvc, aptDate, aptTime);

}

public void getElementsFromExcel() throws IOException {

excelData.readSchedFromExcel();

}

public void traverse(){

SchedNode curr = head;

while(curr!=null) {

System.out.println("\t\t| " + curr.name + " - " + curr.service + " - " + curr.date + " - " + curr.time + " |");

curr = curr.next;

}

}

public static SchedLinkedList Appointments = new SchedLinkedList();

}

**xlsxReadWrite.java**

package Menu;

import org.apache.poi.xssf.usermodel.XSSFRow;

import org.apache.poi.xssf.usermodel.XSSFSheet;

import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class xlsxReadWrite{

public static int patientCount = 1;

public static int schedCount = 0;

public static int receiptsGenerated = 0;

public static int searchedIndex = 0;

public void writeToExcel(String name, String gender, String age, int dc, int gp, int od, int c, int dfa) throws IOException {

FileInputStream myXLSXFile = new FileInputStream("ConsolidatedList.xlsx");

XSSFWorkbook wBook = new XSSFWorkbook(myXLSXFile);

XSSFSheet wSheet = wBook.getSheetAt(0);

XSSFRow row = wSheet.createRow(patientCount);

row.createCell(0).setCellValue(name);

row.createCell(1).setCellValue(age);

row.createCell(2).setCellValue(gender);

row.createCell(3).setCellValue(dc);

row.createCell(4).setCellValue(gp);

row.createCell(5).setCellValue(od);

row.createCell(6).setCellValue(c);

row.createCell(7).setCellValue(dfa);

myXLSXFile.close();

FileOutputStream output\_file = new FileOutputStream(new File("ConsolidatedList.xlsx"));

wBook.write(output\_file);

output\_file.close();

}

public void writeSchedToExcel(String name, String service, String date, String time) throws IOException{

FileInputStream myXLSXFile = new FileInputStream("ConsolidatedList.xlsx");

XSSFWorkbook wb = new XSSFWorkbook(myXLSXFile);

XSSFSheet schedSheet = wb.getSheetAt(1);

XSSFRow row = schedSheet.createRow(schedSheet.getLastRowNum()+1);

row.createCell(0).setCellValue(name);

row.createCell(1).setCellValue(service);

row.createCell(2).setCellValue(date);

row.createCell(3).setCellValue(time);

myXLSXFile.close();

FileOutputStream output = new FileOutputStream(new File("ConsolidatedList.xlsx"));

wb.write(output);

output.close();

}

public void deleteRow() throws IOException {

FileInputStream myXLSXFile = new FileInputStream("ConsolidatedList.xlsx");

XSSFWorkbook wBook = new XSSFWorkbook(myXLSXFile);

XSSFSheet wSheet = wBook.getSheetAt(0);

wSheet.removeRow(wSheet.getRow(searchedIndex+1));

if( searchedIndex+1 != wSheet.getLastRowNum()+1 ){

wSheet.shiftRows(searchedIndex+2, wSheet.getLastRowNum(), -1);

}

myXLSXFile.close();

FileOutputStream output\_file = new FileOutputStream(new File("ConsolidatedList.xlsx"));

wBook.write(output\_file);

output\_file.close();

}

public void readProfileToExcel() throws IOException {

PatientLinkedList currLinkedList = new PatientLinkedList();

XSSFWorkbook wb = new XSSFWorkbook("ConsolidatedList.xlsx");

XSSFSheet schedSheet = wb.getSheetAt(0);

for (int i = 2; i <= schedSheet.getLastRowNum(); i++) {

String name = schedSheet.getRow(i).getCell(0).getStringCellValue();

String age = schedSheet.getRow(i).getCell(1).getStringCellValue();

String gender = schedSheet.getRow(i).getCell(2).getStringCellValue();

double dc = schedSheet.getRow(i).getCell(3).getNumericCellValue();

double gp = schedSheet.getRow(i).getCell(4).getNumericCellValue();

double od = schedSheet.getRow(i).getCell(5).getNumericCellValue();

double c = schedSheet.getRow(i).getCell(6).getNumericCellValue();

double dfa = schedSheet.getRow(i).getCell(7).getNumericCellValue();

currLinkedList.pushToStaticLinkedList(name, gender, age, (int) dc, (int) gp, (int) od, (int) c, (int) dfa);

}

wb.close();

}

public void readSchedFromExcel() throws IOException {

SchedLinkedList currLinkedList = new SchedLinkedList();

XSSFWorkbook wb = new XSSFWorkbook("ConsolidatedList.xlsx");

XSSFSheet schedSheet = wb.getSheetAt(1);

for (int i = 1; i <=schedSheet.getLastRowNum(); i++) {

System.out.println("Iteration " +i);

String name = schedSheet.getRow(i).getCell(0).getStringCellValue();

String service = schedSheet.getRow(i).getCell(1).getStringCellValue();

String date = schedSheet.getRow(i).getCell(2).getStringCellValue();

String time = schedSheet.getRow(i).getCell(3).getStringCellValue();

currLinkedList.pushToStaticLinkedList(name, service, date, time);

}

wb.close();

}

public void editToExcel(String name, String gender, String age, int dc, int gp, int od, int c, int dfa) throws IOException {

FileInputStream myXLSXFile = new FileInputStream("ConsolidatedList.xlsx");

XSSFWorkbook wBook = new XSSFWorkbook(myXLSXFile);

XSSFSheet wSheet = wBook.getSheetAt(0);

XSSFRow row = wSheet.createRow(searchedIndex+1);

row.createCell(0).setCellValue(name);

row.createCell(1).setCellValue(age);

row.createCell(2).setCellValue(gender);

row.createCell(3).setCellValue(dc);

row.createCell(4).setCellValue(gp);

row.createCell(5).setCellValue(od);

row.createCell(6).setCellValue(c);

row.createCell(7).setCellValue(dfa);

myXLSXFile.close();

FileOutputStream output\_file = new FileOutputStream(new File("ConsolidatedList.xlsx"));

wBook.write(output\_file);

output\_file.close();

}

public void generateReceiptCount() throws IOException {

XSSFWorkbook wb = new XSSFWorkbook("ConsolidatedList.xlsx");

XSSFSheet schedSheet = wb.getSheetAt(2);

receiptsGenerated = (int) schedSheet.getRow(0).getCell(0).getNumericCellValue();

wb.close();

}

public void addReceiptCount() throws IOException {

FileInputStream myXLSXFile = new FileInputStream("ConsolidatedList.xlsx");

XSSFWorkbook wBook = new XSSFWorkbook(myXLSXFile);

XSSFSheet wSheet = wBook.getSheetAt(2);

XSSFRow row = wSheet.createRow(0);

row.createCell(0).setCellValue(receiptsGenerated);

myXLSXFile.close();

FileOutputStream output\_file = new FileOutputStream(new File("ConsolidatedList.xlsx"));

wBook.write(output\_file);

output\_file.close();

}

}

**RegisterController.java**

package Register;

import com.jfoenix.controls.JFXCheckBox;

import com.jfoenix.controls.JFXRadioButton;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.control.ChoiceBox;

import javafx.scene.control.Spinner;

import javafx.scene.control.SpinnerValueFactory;

import javafx.scene.control.TextField;

import javafx.scene.control.ToggleGroup;

import javafx.scene.layout.AnchorPane;

import java.io.IOException;

public class RegisterController extends Menu.PatientLinkedList{

ObservableList<String> ageGroups = FXCollections.observableArrayList("0 - 5 Years Old", "6 - 9 Years Old", "10 - 24 Years Old", "25 - 59 Years Old", "60+ Years Old");

@FXML

private AnchorPane RegisterAnchor;

@FXML

private ChoiceBox ageChoice;

@FXML

private ToggleGroup genderRadio;

@FXML

private TextField patientName;

@FXML

private Spinner cariesAmt;

@FXML

private Spinner gingiPerioAmt;

@FXML

private Spinner debrisAmt;

@FXML

private JFXCheckBox dentalCaries;

@FXML

private JFXCheckBox gingiPerio;

@FXML

private JFXCheckBox oralDebris;

@FXML

private JFXCheckBox calculus;

@FXML

private JFXCheckBox dentoFacialAnomaly;

public void initialize(){

ageChoice.setItems(ageGroups);

SpinnerValueFactory<Integer> cariesAmount = new SpinnerValueFactory.IntegerSpinnerValueFactory(1,99,1);

this.cariesAmt.setValueFactory(cariesAmount);

SpinnerValueFactory<Integer> gingiPerioAmount = new SpinnerValueFactory.IntegerSpinnerValueFactory(1,30,1);

this.gingiPerioAmt.setValueFactory(gingiPerioAmount);

SpinnerValueFactory<Integer> debrisAmount = new SpinnerValueFactory.IntegerSpinnerValueFactory(1,99,1);

this.debrisAmt.setValueFactory(debrisAmount);

}

public void dentalCariesSelected(ActionEvent event) throws IOException{

if(dentalCaries.isSelected()) cariesAmt.setDisable(false);

else cariesAmt.setDisable(true);

}

public void gingiPerioSelected(ActionEvent event) throws IOException{

if(gingiPerio.isSelected()) gingiPerioAmt.setDisable(false);

else gingiPerioAmt.setDisable(true);

}

public void oralDebrisSelected(ActionEvent event) throws IOException{

if(oralDebris.isSelected()) debrisAmt.setDisable(false);

else debrisAmt.setDisable(true);

}

public void getPatientData(ActionEvent event) throws IOException{

String name = patientName.getText();

JFXRadioButton selectedGender = (JFXRadioButton) genderRadio.getSelectedToggle();

String gender = selectedGender.getText();

String age = (String) ageChoice.getValue();

int dc = 0, gp = 0, od = 0, c = 0, dfa = 0;

if(dentalCaries.isSelected()){

dc = (int) cariesAmt.getValue();

}

if(gingiPerio.isSelected()){

gp = (int) gingiPerioAmt.getValue();

}

if(oralDebris.isSelected()){

od = (int) debrisAmt.getValue();

}

if(calculus.isSelected()) c = 1;

if(dentoFacialAnomaly.isSelected()) dfa = 1;

AllPatients.read = false;

AllPatients.pushPatientData(name, gender, age, dc, gp, od, c, dfa);

patientName.setText("");

dentalCaries.setSelected(false);

gingiPerio.setSelected(false);

oralDebris.setSelected(false);

calculus.setSelected(false);

dentoFacialAnomaly.setSelected(false);

cariesAmt.setDisable(true);

gingiPerioAmt.setDisable(true);

debrisAmt.setDisable(true);

}

public void back(ActionEvent event) throws IOException {

AnchorPane nextAnchorPane = FXMLLoader.load(getClass().getResource("../Menu/MenuUI.fxml"));

RegisterAnchor.getChildren().setAll(nextAnchorPane);

}

}

**ScheduleController.java**

package Schedule;

import Menu.SchedLinkedList;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.geometry.Pos;

import javafx.scene.Node;

import javafx.scene.control.\*;

import javafx.scene.layout.AnchorPane;

import javafx.scene.layout.GridPane;

import javafx.scene.layout.VBox;

import javafx.scene.paint.Color;

import java.io.IOException;

import java.net.URL;

import java.text.SimpleDateFormat;

import java.time.format.DateTimeFormatter;

import java.util.Arrays;

import java.util.Calendar;

import java.util.GregorianCalendar;

import java.util.ResourceBundle;

public class ScheduleController extends SchedLinkedList implements Initializable {

public AnchorPane ScheduleAnchor;

public TextField nameTextField;

public DatePicker dateInput;

public Label minTime;

public Label maxTime;

public MenuButton fromHrButton;

public MenuButton fromMinButton;

public MenuButton toHrButton;

public MenuButton toMinButton;

public Button cancelButton;

public Button confirmButton;

public MenuItem from09hr;

public MenuItem from10hr;

public MenuItem from11hr;

public MenuItem from12hr;

public MenuItem from01hr;

public MenuItem from02hr;

public MenuItem from03hr;

public MenuItem from04hr;

public MenuItem from05hr;

public MenuItem from00min;

public MenuItem from30min;

public MenuItem to09hr;

public MenuItem to10hr;

public MenuItem to11hr;

public MenuItem to12hr;

public MenuItem to01hr;

public MenuItem to02hr;

public MenuItem to03hr;

public MenuItem to04hr;

public MenuItem to05hr;

public MenuItem to00min;

public MenuItem to30min;

public GridPane scheduleGrid;

public Label day1;

public Label day2;

public Label day3;

public Label day4;

public Label day5;

public Label day6;

public Label day7;

public MenuButton serviceSelection;

public MenuItem serviceCh1;

public MenuItem serviceCh2;

public MenuItem serviceCh3;

public MenuItem serviceCh4;

public MenuItem serviceCh5;

public MenuItem serviceCh6;

public MenuItem serviceCh7;

public MenuItem serviceCh8;

public MenuItem serviceCh9;

public Button clearButton;

public Label errorLabel;

public void selectMinHr(ActionEvent actionEvent) {

}

public void selectMinMinute(ActionEvent actionEvent) {

}

public void selectMaxHour(ActionEvent actionEvent) {

}

public void selectMaxMin(ActionEvent actionEvent) {

}

public void onConfirm(ActionEvent actionEvent) throws IOException {

String fromHr = fromHrButton.getText();

String fromMin = fromMinButton.getText();

String toHr = toHrButton.getText();

String toMin = toMinButton.getText();

int x=0,y1=0, y2=0;

int minInit=0, minFin=0;

System.out.println(fromHr +":"+fromMin+" to "+toHr+":"+toMin);

if(fromMin.equals("00")){

minInit=0;

}

else if(fromMin.equals("30")){

minInit=1;

}

else if(toMin.equals("00")){

minFin=0;

}

else if(toMin.equals("30")){

minFin=1;

}

if(fromHr.equals("09")){

y1=0+minInit;

}

else if(fromHr.equals("10")){

y1=2+minInit;

}

else if(fromHr.equals("11")){

y1=4+minInit;

}

else if(fromHr.equals("12")){

y1=6+minInit;

}

else if(fromHr.equals("01")){

y1=8+minInit;

}

else if(fromHr.equals("02")){

y1=10+minInit;

}

else if(fromHr.equals("03")){

y1=12+minInit;

}

else if(fromHr.equals("04")){

y1=14+minInit;

}

else if(fromHr.equals("05")){

y1=16+minInit;

}

if(toHr.equals("09")){

y2=0+minFin;

}

else if(toHr.equals("10")){

y2=2+minFin;

}

else if(toHr.equals("11")){

y2=4+minFin;

}

else if(toHr.equals("12")){

y2=6+minFin;

}

else if(toHr.equals("01")){

y2=8+minFin;

}

else if(toHr.equals("02")){

y2=10+minFin;

}

else if(toHr.equals("03")){

y2=12+minFin;

}

else if(toHr.equals("04")){

y2=14+minFin;

}

else if(toHr.equals("05")){

y2=16+minFin;

}

Calendar date = Calendar.getInstance();

date.set(dateInput.getValue().getYear(), dateInput.getValue().getMonthValue()-1, dateInput.getValue().getDayOfMonth());

SimpleDateFormat format = new SimpleDateFormat("MM/dd/yyyy");

String[] dates = new String[7];

int delta = -date.get(GregorianCalendar.DAY\_OF\_WEEK) + 2; //add 2 if your week start on monday

date.add(Calendar.DAY\_OF\_MONTH, delta );

String dateInputString = dateInput.getValue().format(DateTimeFormatter.ofPattern("MM/dd/yyyy"));

for (int i = 0; i < 7; i++)

{

dates[i] = format.format(date.getTime());

date.add(Calendar.DAY\_OF\_MONTH, 1);

if(dates[i].equals(dateInputString)){

x=i;

}

}

System.out.println(Arrays.toString(dates));

if(!isSchedAvailable(x, y1, y2)){

errorLabel.setText("Unavailable schedule. Try again!");

errorLabel.setTextFill(Color.web("#FF9090"));

}

else if(isSchedAvailable(x, y1, y2)){

errorLabel.setText("Scheduling successful!");

errorLabel.setTextFill(Color.web("#4DED30"));

setSchedule(x, y1, y2, nameTextField.getText(), serviceSelection.getText());

Appointments.read=false;

Appointments.pushScheduleInfo(nameTextField.getText(), serviceSelection.getText(), dateInputString, minTime.getText() + "-" + maxTime.getText());

}

}

public boolean isSchedAvailable(int x, int y1, int y2){

boolean available = true;

ObservableList<Node> childrens = scheduleGrid.getChildren();

for(int y=y1; y<y2; y++) {

for (Node node : childrens) {

if (node instanceof VBox) {

if (scheduleGrid.getRowIndex(node) == y && scheduleGrid.getColumnIndex(node) == x) {

available = false;

}

}

}

}

return available;

}

public void setSchedule(int x, int y1, int y2, String name, String service){

for(int y=y1; y<y2; y++){

Label nameLabel = new Label(name);

nameLabel.setStyle("-fx-font-weight: bold");

Label serviceLabel = new Label(service);

VBox root = new VBox();

root.setAlignment(Pos.CENTER);

root.getChildren().addAll(nameLabel,serviceLabel);

scheduleGrid.add(root,x,y);

}

}

public void initializeSchedule(){

if(Appointments.head==null) {

try {

getElementsFromExcel();

} catch (IOException e) {

e.printStackTrace();

}

}

SchedNode curr = Appointments.head;

String[] time;

Calendar date = Calendar.getInstance();

SimpleDateFormat format = new SimpleDateFormat("MM/dd/yyyy");

String[] dates = new String[7];

int delta = -date.get(GregorianCalendar.DAY\_OF\_WEEK) + 2; //add 2 if your week start on monday

date.add(Calendar.DAY\_OF\_MONTH, delta );

for (int i = 0; i < 7; i++)

{

dates[i] = format.format(date.getTime());

date.add(Calendar.DAY\_OF\_MONTH, 1);

while(curr!=null){

if(curr.date.equals(dates[i])){

time = splitTime(curr.time);

setCurrentSched(curr.name, curr.service, i, time);

break;

}

curr=curr.next;

}

curr = Appointments.head;

}

day1.setText(dates[0]);

day2.setText(dates[1]);

day3.setText(dates[2]);

day4.setText(dates[3]);

day5.setText(dates[4]);

day6.setText(dates[5]);

day7.setText(dates[6]);

}

public void setCurrentSched(String name, String srvc, int x, String[] time){

String fromHr = time[0];

String fromMin = time[1];

String toHr = time[2];

String toMin = time[3];

int y1=0, y2=0;

int minInit=0, minFin=0;

System.out.println(fromHr +":"+fromMin+" to "+toHr+":"+toMin);

if(fromMin.equals("00")){

minInit=0;

}

else if(fromMin.equals("30")){

minInit=1;

}

else if(toMin.equals("00")){

minFin=0;

}

else if(toMin.equals("30")){

minFin=1;

}

if(fromHr.equals("09")){

y1=0+minInit;

}

else if(fromHr.equals("10")){

y1=2+minInit;

}

else if(fromHr.equals("11")){

y1=4+minInit;

}

else if(fromHr.equals("12")){

y1=6+minInit;

}

else if(fromHr.equals("01")){

y1=8+minInit;

}

else if(fromHr.equals("02")){

y1=10+minInit;

}

else if(fromHr.equals("03")){

y1=12+minInit;

}

else if(fromHr.equals("04")){

y1=14+minInit;

}

else if(fromHr.equals("05")){

y1=16+minInit;

}

if(toHr.equals("09")){

y2=0+minFin;

}

else if(toHr.equals("10")){

y2=2+minFin;

}

else if(toHr.equals("11")){

y2=4+minFin;

}

else if(toHr.equals("12")){

y2=6+minFin;

}

else if(toHr.equals("01")){

y2=8+minFin;

}

else if(toHr.equals("02")){

y2=10+minFin;

}

else if(toHr.equals("03")){

y2=12+minFin;

}

else if(toHr.equals("04")){

y2=14+minFin;

}

else if(toHr.equals("05")){

y2=16+minFin;

}

setSchedule(x, y1, y2, name, srvc);

}

public String[] splitTime(String time){

String[] splitTime = time.split("[-:]");

return splitTime;

}

public void onService1(ActionEvent actionEvent) {

serviceSelection.setText(serviceCh1.getText());

}

public void onService2(ActionEvent actionEvent) {

serviceSelection.setText(serviceCh2.getText());

}

public void onService3(ActionEvent actionEvent) {

serviceSelection.setText(serviceCh3.getText());

}

public void onService4(ActionEvent actionEvent) {

serviceSelection.setText(serviceCh4.getText());

}

public void onService5(ActionEvent actionEvent) {

serviceSelection.setText(serviceCh5.getText());

}

public void onService6(ActionEvent actionEvent) {

serviceSelection.setText(serviceCh6.getText());

}

public void onService7(ActionEvent actionEvent) {

serviceSelection.setText(serviceCh7.getText());

}

public void onService8(ActionEvent actionEvent) {

serviceSelection.setText(serviceCh8.getText());

}

public void onService9(ActionEvent actionEvent) {

serviceSelection.setText(serviceCh9.getText());

}

public void selectFrom09hr(ActionEvent actionEvent) {

fromHrButton.setText(from09hr.getText());

setEnabledTime();

}

public void selectFrom10hr(ActionEvent actionEvent) {

fromHrButton.setText(from10hr.getText());

setEnabledTime();

}

public void selectFrom11hr(ActionEvent actionEvent) {

fromHrButton.setText(from11hr.getText());

setEnabledTime();

}

public void selectFrom12hr(ActionEvent actionEvent) {

fromHrButton.setText(from12hr.getText());

setEnabledTime();

}

public void selectFrom01hr(ActionEvent actionEvent) {

fromHrButton.setText(from01hr.getText());

setEnabledTime();

}

public void selectFrom02hr(ActionEvent actionEvent) {

fromHrButton.setText(from02hr.getText());

setEnabledTime();

}

public void selectFrom03hr(ActionEvent actionEvent) {

fromHrButton.setText(from03hr.getText());

setEnabledTime();

}

public void selectFrom04hr(ActionEvent actionEvent) {

fromHrButton.setText(from04hr.getText());

setEnabledTime();

}

public void selectFrom05hr(ActionEvent actionEvent) {

fromHrButton.setText(from05hr.getText());

setEnabledTime();

}

public void selectFrom00min(ActionEvent actionEvent) {

fromMinButton.setText(from00min.getText());

minTime.setText(fromHrButton.getText() + ":" + fromMinButton.getText());

}

public void selectFrom30min(ActionEvent actionEvent) {

fromMinButton.setText(from30min.getText());

minTime.setText(fromHrButton.getText() + ":" + fromMinButton.getText());

setEnabledTime();

}

public void selectTo09hr(ActionEvent actionEvent) {

toHrButton.setText(to09hr.getText());

setEnabledTime();

}

public void selectTo10hr(ActionEvent actionEvent) {

toHrButton.setText(to10hr.getText());

setEnabledTime();

}

public void selectTo11hr(ActionEvent actionEvent) {

toHrButton.setText(to11hr.getText());

setEnabledTime();

}

public void selectTo12hr(ActionEvent actionEvent) {

toHrButton.setText(to12hr.getText());

setEnabledTime();

}

public void selectTo01hr(ActionEvent actionEvent) {

toHrButton.setText(to01hr.getText());

setEnabledTime();

}

public void selectTo02hr(ActionEvent actionEvent) {

toHrButton.setText(to02hr.getText());

setEnabledTime();

}

public void selectTo03hr(ActionEvent actionEvent) {

toHrButton.setText(to03hr.getText());

setEnabledTime();

}

public void selectTo04hr(ActionEvent actionEvent) {

toHrButton.setText(to04hr.getText());

setEnabledTime();

}

public void selectTo05hr(ActionEvent actionEvent) {

toHrButton.setText(to05hr.getText());

setEnabledTime();

}

public void selectTo00min(ActionEvent actionEvent) {

toMinButton.setText(to00min.getText());

maxTime.setText(toHrButton.getText() + ":" + toMinButton.getText());

}

public void selectTo30min(ActionEvent actionEvent) {

toMinButton.setText(to30min.getText());

maxTime.setText(toHrButton.getText() + ":" + toMinButton.getText());

}

public void setEnabledTime(){

if((fromMinButton.getText().equals("Min"))&&(!fromHrButton.getText().equals("Hr"))) minTime.setText(fromHrButton.getText() + ":00");

else if((!fromMinButton.getText().equals("Min"))&&(!fromHrButton.getText().equals("Hr"))) minTime.setText(fromHrButton.getText() + ":" + fromMinButton.getText());

if((toMinButton.getText().equals("Min"))&&(!toHrButton.getText().equals("Hr"))) maxTime.setText(toHrButton.getText() + ":00");

else if ((!toMinButton.getText().equals("Min"))&&(!toHrButton.getText().equals("Hr"))) maxTime.setText(toHrButton.getText() + ":" + toMinButton.getText());

if((fromHrButton.getText().equals(toHrButton.getText()))&&fromMinButton.getText().equals("00")) to00min.setDisable(true);

else to00min.setDisable(false);

switch (fromHrButton.getText()) {

case "09":

to09hr.setDisable(false);

to10hr.setDisable(false);

to11hr.setDisable(false);

to12hr.setDisable(false);

to01hr.setDisable(false);

to02hr.setDisable(false);

to03hr.setDisable(false);

to04hr.setDisable(false);

break;

case "10":

to09hr.setDisable(true);

to10hr.setDisable(false);

to11hr.setDisable(false);

to12hr.setDisable(false);

to01hr.setDisable(false);

to02hr.setDisable(false);

to03hr.setDisable(false);

to04hr.setDisable(false);

break;

case "11":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(false);

to12hr.setDisable(false);

to01hr.setDisable(false);

to02hr.setDisable(false);

to03hr.setDisable(false);

to04hr.setDisable(false);

break;

case "12":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(true);

to12hr.setDisable(false);

to01hr.setDisable(false);

to02hr.setDisable(false);

to03hr.setDisable(false);

to04hr.setDisable(false);

break;

case "01":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(true);

to12hr.setDisable(true);

to01hr.setDisable(false);

to02hr.setDisable(false);

to03hr.setDisable(false);

to04hr.setDisable(false);

break;

case "02":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(true);

to12hr.setDisable(true);

to01hr.setDisable(true);

to02hr.setDisable(false);

to03hr.setDisable(false);

to04hr.setDisable(false);

break;

case "03":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(true);

to12hr.setDisable(true);

to01hr.setDisable(true);

to02hr.setDisable(true);

to03hr.setDisable(false);

to04hr.setDisable(false);

break;

case "04":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(true);

to12hr.setDisable(true);

to01hr.setDisable(true);

to02hr.setDisable(true);

to03hr.setDisable(true);

to04hr.setDisable(false);

break;

case "05":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(true);

to12hr.setDisable(true);

to01hr.setDisable(true);

to02hr.setDisable(true);

to03hr.setDisable(true);

to04hr.setDisable(true);

break;

}

switch (toHrButton.getText()) {

case "09":

from09hr.setDisable(false);

from10hr.setDisable(true);

from11hr.setDisable(true);

from12hr.setDisable(true);

from01hr.setDisable(true);

from02hr.setDisable(true);

from03hr.setDisable(true);

from04hr.setDisable(true);

from05hr.setDisable(true);

break;

case "10":

from09hr.setDisable(false);

from10hr.setDisable(false);

from11hr.setDisable(true);

from12hr.setDisable(true);

from01hr.setDisable(true);

from02hr.setDisable(true);

from03hr.setDisable(true);

from04hr.setDisable(true);

from05hr.setDisable(true);

break;

case "11":

from09hr.setDisable(false);

from10hr.setDisable(false);

from11hr.setDisable(false);

from12hr.setDisable(true);

from01hr.setDisable(true);

from02hr.setDisable(true);

from03hr.setDisable(true);

from04hr.setDisable(true);

from05hr.setDisable(true);

break;

case "12":

from09hr.setDisable(false);

from10hr.setDisable(false);

from11hr.setDisable(false);

from12hr.setDisable(false);

from01hr.setDisable(true);

from02hr.setDisable(true);

from03hr.setDisable(true);

from04hr.setDisable(true);

from05hr.setDisable(true);

break;

case "01":

from09hr.setDisable(false);

from10hr.setDisable(false);

from11hr.setDisable(false);

from12hr.setDisable(false);

from01hr.setDisable(false);

from02hr.setDisable(true);

from03hr.setDisable(true);

from04hr.setDisable(true);

from05hr.setDisable(true);

break;

case "02":

from09hr.setDisable(false);

from10hr.setDisable(false);

from11hr.setDisable(false);

from12hr.setDisable(false);

from01hr.setDisable(false);

from02hr.setDisable(false);

from03hr.setDisable(true);

from04hr.setDisable(true);

from05hr.setDisable(true);

break;

case "03":

from09hr.setDisable(false);

from10hr.setDisable(false);

from11hr.setDisable(false);

from12hr.setDisable(false);

from01hr.setDisable(false);

from02hr.setDisable(false);

from03hr.setDisable(false);

from04hr.setDisable(true);

from05hr.setDisable(true);

break;

case "04":

from09hr.setDisable(false);

from10hr.setDisable(false);

from11hr.setDisable(false);

from12hr.setDisable(false);

from01hr.setDisable(false);

from02hr.setDisable(false);

from03hr.setDisable(false);

from04hr.setDisable(false);

from05hr.setDisable(true);

break;

case "05":

from09hr.setDisable(false);

from10hr.setDisable(false);

from11hr.setDisable(false);

from12hr.setDisable(false);

from01hr.setDisable(false);

from02hr.setDisable(false);

from03hr.setDisable(false);

from04hr.setDisable(false);

from05hr.setDisable(false);

break;

}

if(fromMinButton.getText().equals("30")){

switch (fromHrButton.getText()) {

case "09":

to09hr.setDisable(true);

to10hr.setDisable(false);

to11hr.setDisable(false);

to12hr.setDisable(false);

to01hr.setDisable(false);

to02hr.setDisable(false);

to03hr.setDisable(false);

to04hr.setDisable(false);

break;

case "10":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(false);

to12hr.setDisable(false);

to01hr.setDisable(false);

to02hr.setDisable(false);

to03hr.setDisable(false);

to04hr.setDisable(false);

break;

case "11":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(true);

to12hr.setDisable(false);

to01hr.setDisable(false);

to02hr.setDisable(false);

to03hr.setDisable(false);

to04hr.setDisable(false);

break;

case "12":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(true);

to12hr.setDisable(true);

to01hr.setDisable(false);

to02hr.setDisable(false);

to03hr.setDisable(false);

to04hr.setDisable(false);

break;

case "01":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(true);

to12hr.setDisable(true);

to01hr.setDisable(true);

to02hr.setDisable(false);

to03hr.setDisable(false);

to04hr.setDisable(false);

break;

case "02":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(true);

to12hr.setDisable(true);

to01hr.setDisable(true);

to02hr.setDisable(true);

to03hr.setDisable(false);

to04hr.setDisable(false);

break;

case "03":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(true);

to12hr.setDisable(true);

to01hr.setDisable(true);

to02hr.setDisable(true);

to03hr.setDisable(true);

to04hr.setDisable(false);

break;

case "04":

to09hr.setDisable(true);

to10hr.setDisable(true);

to11hr.setDisable(true);

to12hr.setDisable(true);

to01hr.setDisable(true);

to02hr.setDisable(true);

to03hr.setDisable(true);

to04hr.setDisable(true);

break;

}

}

}

public void back(ActionEvent actionEvent) throws IOException{

AnchorPane nextAnchorPane = FXMLLoader.load(getClass().getResource("../Menu/MenuUI.fxml"));

ScheduleAnchor.getChildren().setAll(nextAnchorPane);

}

public void onClear(ActionEvent actionEvent) throws IOException {

AnchorPane clear = FXMLLoader.load(getClass().getResource("../Schedule/ScheduleUI.fxml"));

ScheduleAnchor.getChildren().setAll(clear);

}

@Override

public void initialize(URL location, ResourceBundle resources) {

initializeSchedule();

}

}

**BillController.java**

package Bill;

import com.itextpdf.text.\*;

import com.itextpdf.text.pdf.PdfPCell;

import com.itextpdf.text.pdf.PdfPTable;

import com.itextpdf.text.pdf.PdfWriter;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.layout.AnchorPane;

import javafx.scene.text.Text;

import java.io.FileOutputStream;

import java.io.IOException;

public class BillController extends BillIntroController{

@FXML

private AnchorPane billAnchor;

@FXML

private Text givenName;

@FXML

private Text serviceRendered;

@FXML

private Text dateOfTreatment;

@FXML

private Text timeOfTreatment;

@FXML

private Text totalPriceString;

@FXML

private Text timeAndDate;

public void initialize(){

givenName.setText(searchedName);

serviceRendered.setText(searchedService);

dateOfTreatment.setText(searchedDate);

timeOfTreatment.setText(searchedTime);

timeAndDate.setText(currentTime);

if(totalPrice <= 0){

totalPriceString.setText("FREE");

}

else{

totalPriceString.setText(String.valueOf(totalPrice));

}

}

public void savePDF(ActionEvent event) throws IOException, DocumentException {

Document document = new Document(PageSize.LETTER);

PdfWriter writer = PdfWriter.getInstance(document, new FileOutputStream("Receipts/Receipt#" + receiptsGenerated + ".pdf"));

document.open();

PdfPTable logoAndName = new PdfPTable(2);

logoAndName.setWidthPercentage(100);

logoAndName.setWidths(new int[]{1, 6});

Image img = Image.getInstance("src/assets/Portu-Len\_LogoSmall.png");

Font title = FontFactory.getFont(FontFactory.HELVETICA\_BOLD, 16, BaseColor.BLACK);

Font fontBase = FontFactory.getFont(FontFactory.HELVETICA, 12, BaseColor.BLACK);

Font fontGrand = FontFactory.getFont(FontFactory.HELVETICA\_BOLD, 12, BaseColor.BLACK);

Chunk progName = new Chunk("Dental Health Status, Schedule, and Service Report Generator", title);

Chunk chunkName = new Chunk("Patient Name:\t\t\t" + searchedName, fontBase);

Chunk chunkService = new Chunk("Service Rendered:\t\t\t" + searchedService, fontBase);

Chunk chunkDate = new Chunk("Date of Treatment:\t\t\t" + searchedDate, fontBase);

Chunk chunkTime = new Chunk("Time of Treatment:\t\t\t" + searchedTime, fontBase);

Chunk chunkBillTime = new Chunk("Time of Billing:\t\t\t" + currentTime, fontBase);

Chunk chunkPrice = new Chunk("Price:\t\t\t" + priceTreatment, fontBase);

Chunk chunkQuantity = new Chunk("Quantity:\t\t\t" + quantityTreatment, fontBase);

Chunk chunkTotal = new Chunk("Grand Total:\t\t\t" + totalPrice, fontGrand);

Chunk spacing = new Chunk("\n");

Phrase phrase = new Phrase();

phrase.add("\n\n\n");

phrase.add(chunkName +"\n\n");

phrase.add(chunkService+"\n\n");

phrase.add(chunkDate+"\n\n");

phrase.add(chunkTime+"\n\n");

phrase.add(chunkBillTime+"\n\n");

phrase.add(chunkPrice+"\n\n");

phrase.add(chunkQuantity);

phrase.add("\n\n==========================\n\n");

phrase.add(chunkTotal);

logoAndName.addCell(createImageCell(img));

logoAndName.addCell(createTextCell(progName));

document.add(spacing);

document.add(logoAndName);

document.add(spacing);

document.add(phrase);

document.close();

receiptsGenerated = receiptsGenerated + 1;

addReceiptCount();

}

public static PdfPCell createImageCell(Image img) throws DocumentException, IOException {

PdfPCell cell = new PdfPCell(img, false);

cell.setBorder(Rectangle.NO\_BORDER);

return cell;

}

public static PdfPCell createTextCell(Chunk text) throws DocumentException, IOException {

PdfPCell cell = new PdfPCell();

Paragraph p = new Paragraph(text);

p.setAlignment(Element.ALIGN\_CENTER);

cell.addElement(p);

cell.setVerticalAlignment(Element.ALIGN\_CENTER);

cell.setBorder(Rectangle.NO\_BORDER);

return cell;

}

public void back(ActionEvent event) throws IOException {

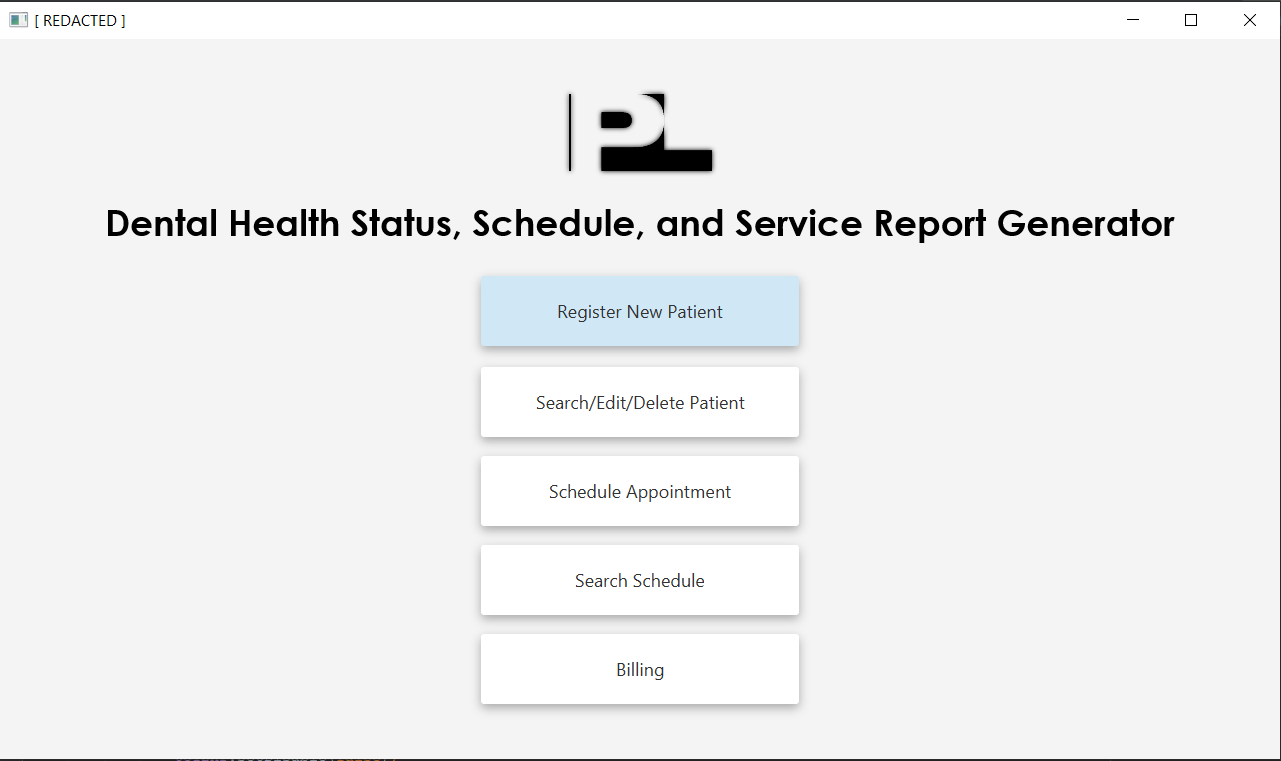
AnchorPane nextAnchorPane = FXMLLoader.load(getClass().getResource("../Menu/MenuUI.fxml"));

billAnchor.getChildren().setAll(nextAnchorPane);

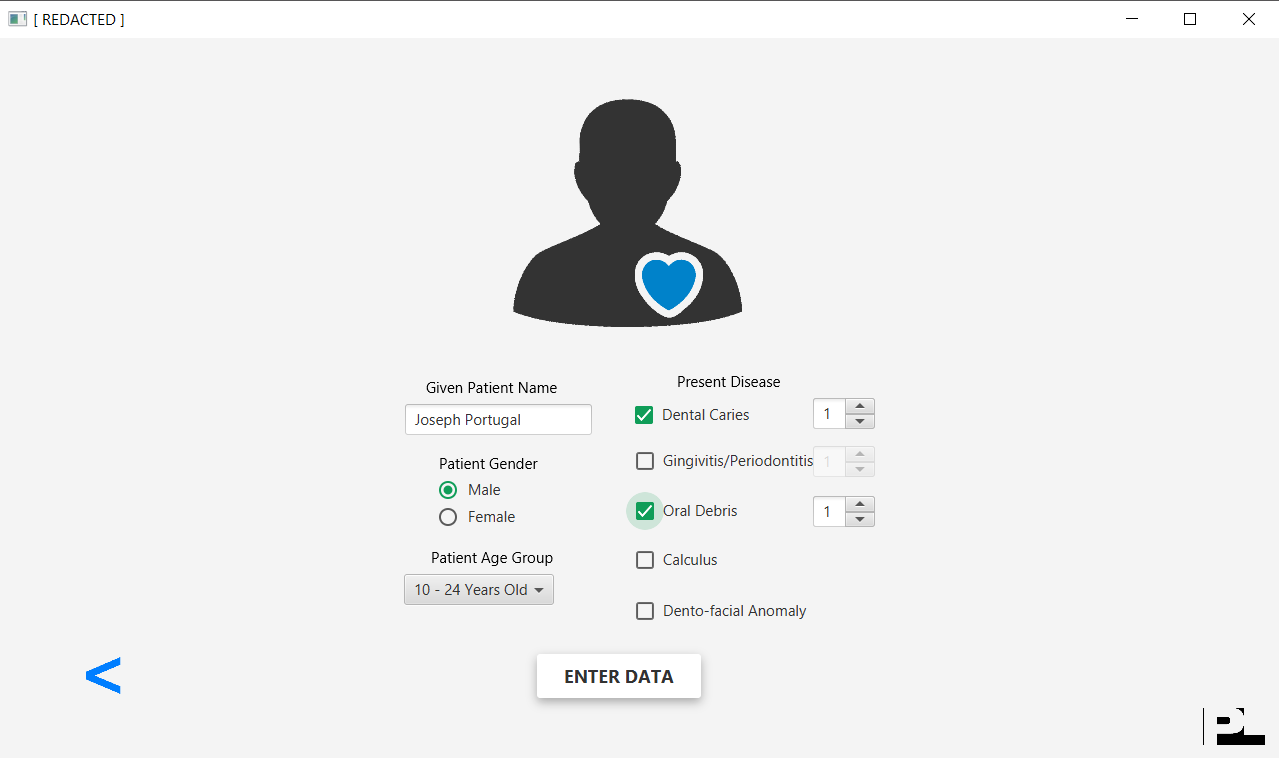
}

}

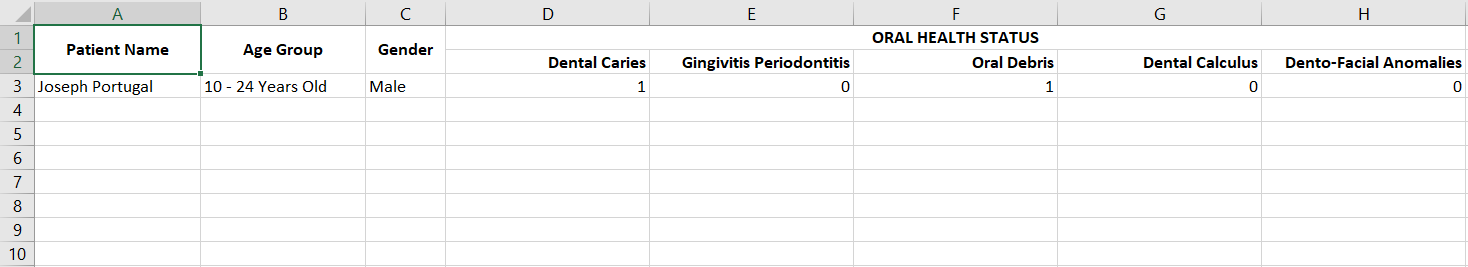
1. Screen Shots of all the output (with input and walkthrough of the system)
2. Here is the official Main Menu of the program. It contains five different operations. The user is free to choose whichever he or she decides.



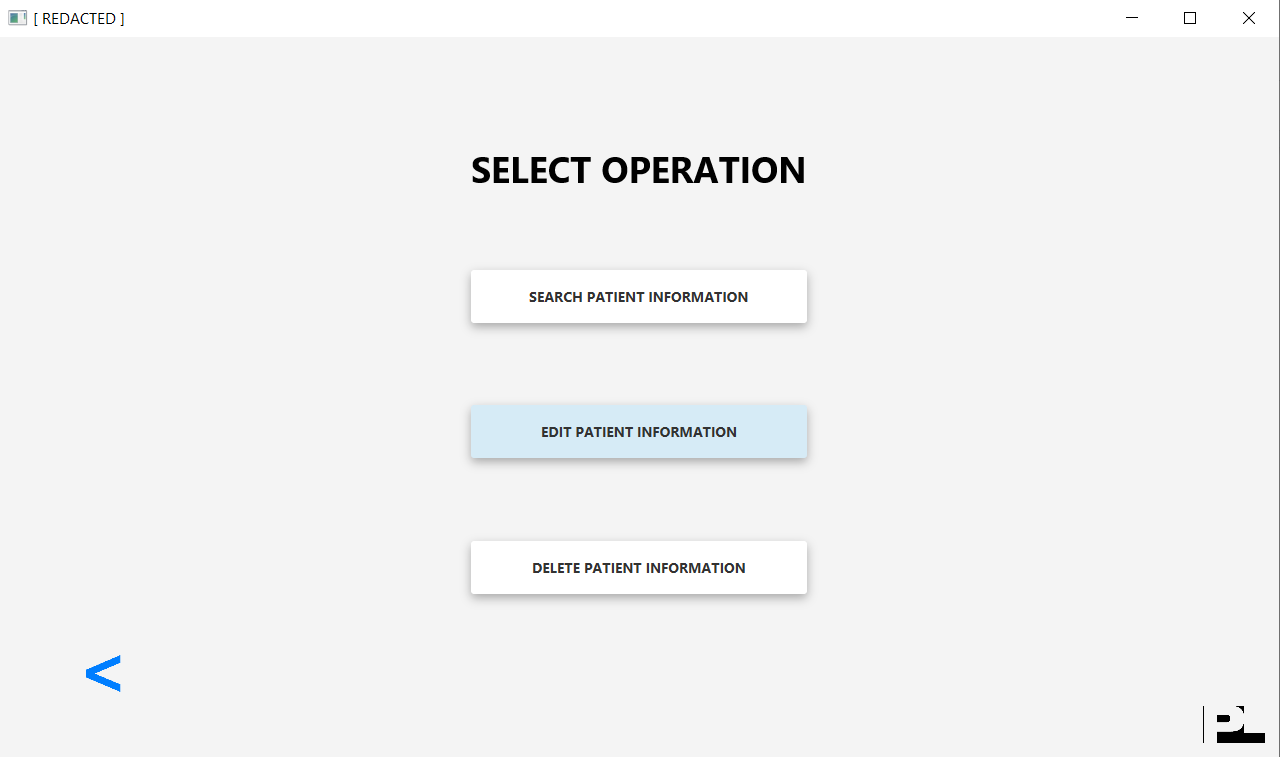
1. However, if there were no patient information in the database, the user has to register a new patient first. Here, the user has to input/select various information such as patient name, gender, age group, and current dental diseases.



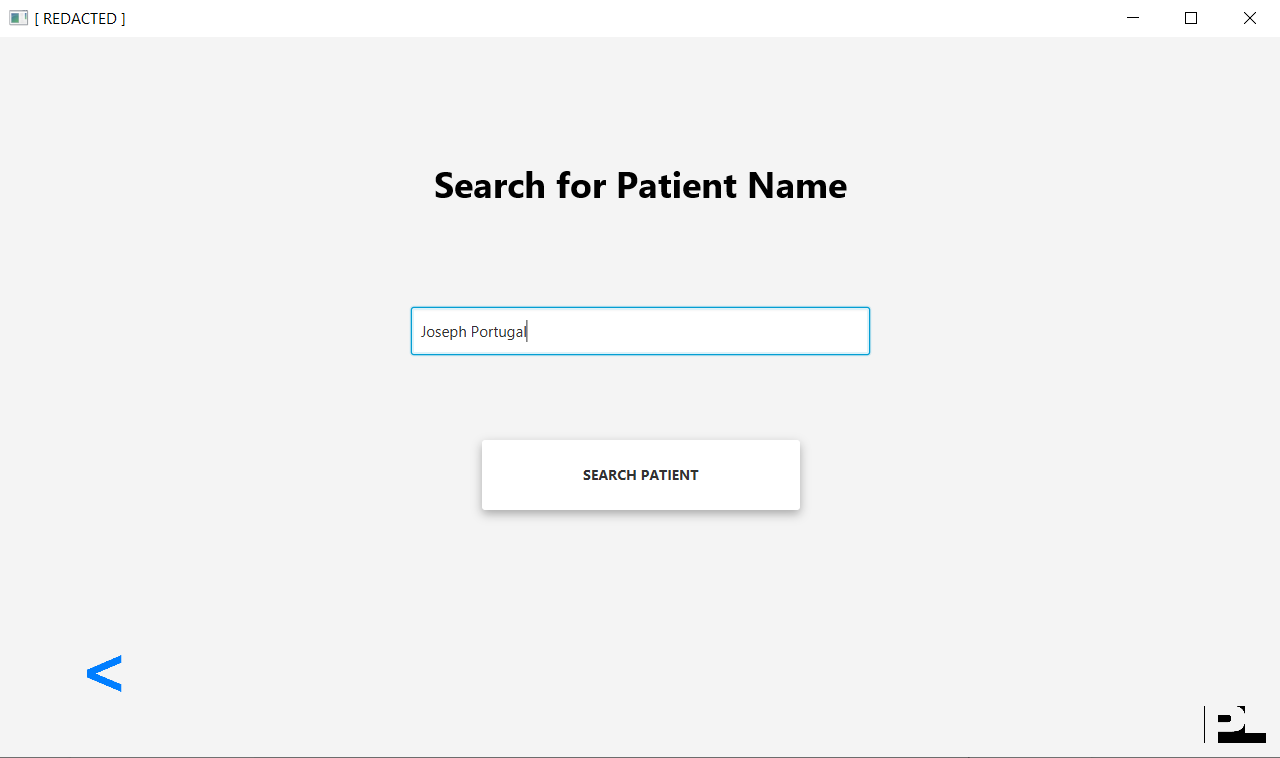
1. Once the user clicks the “ENTER DATA” button, the program collects the inputted data and stores them in the Excel file.



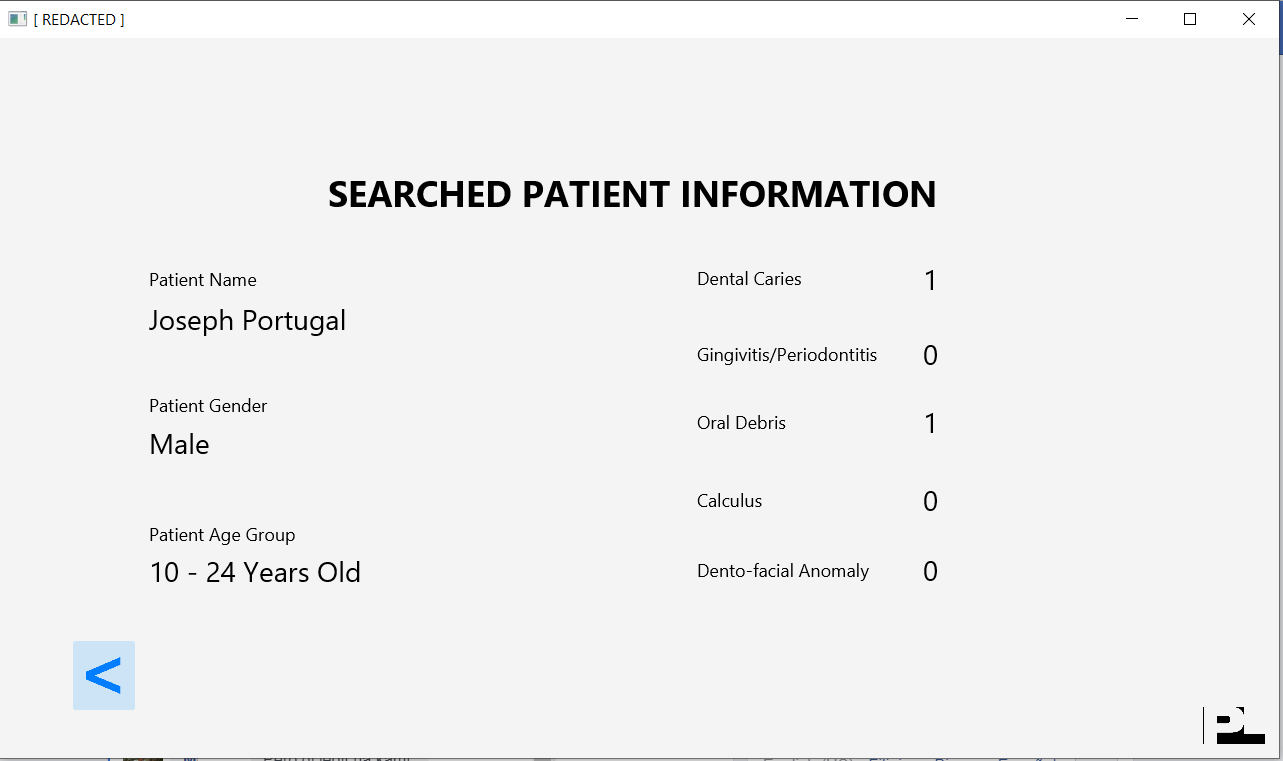
1. When the data has already been stored, the user can now do the following operations:



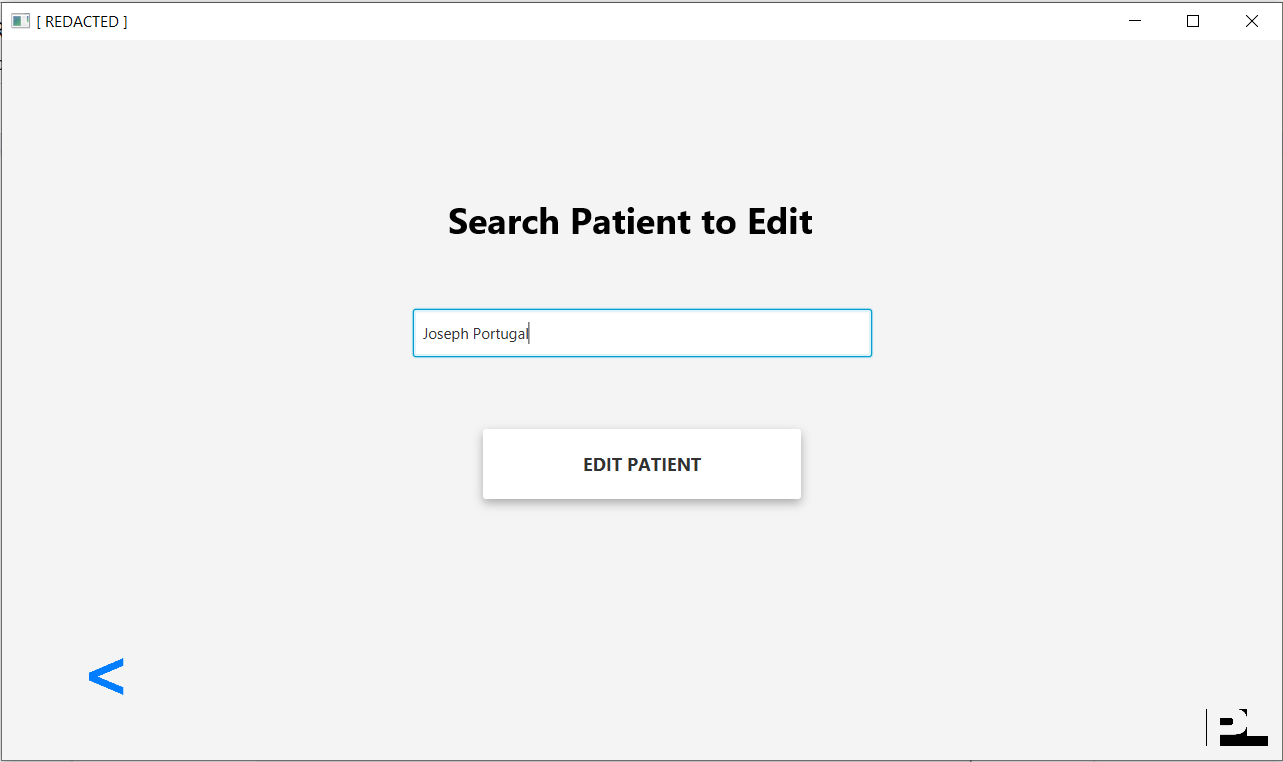
1. If the user were to choose the “Search Patient Information,” the user will be directed to an input.



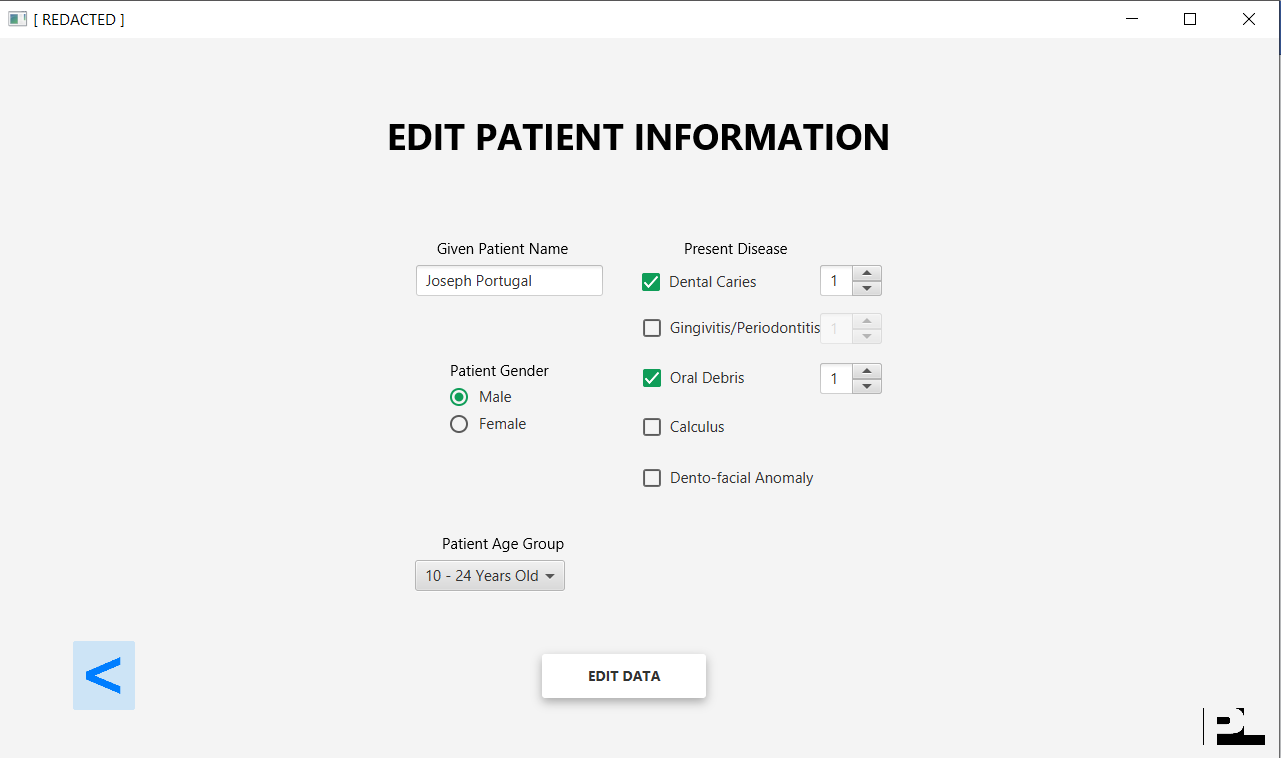
1. After the user inputs the name and clicks the button, the patient’s profile will be displayed to the user as illustrated below:



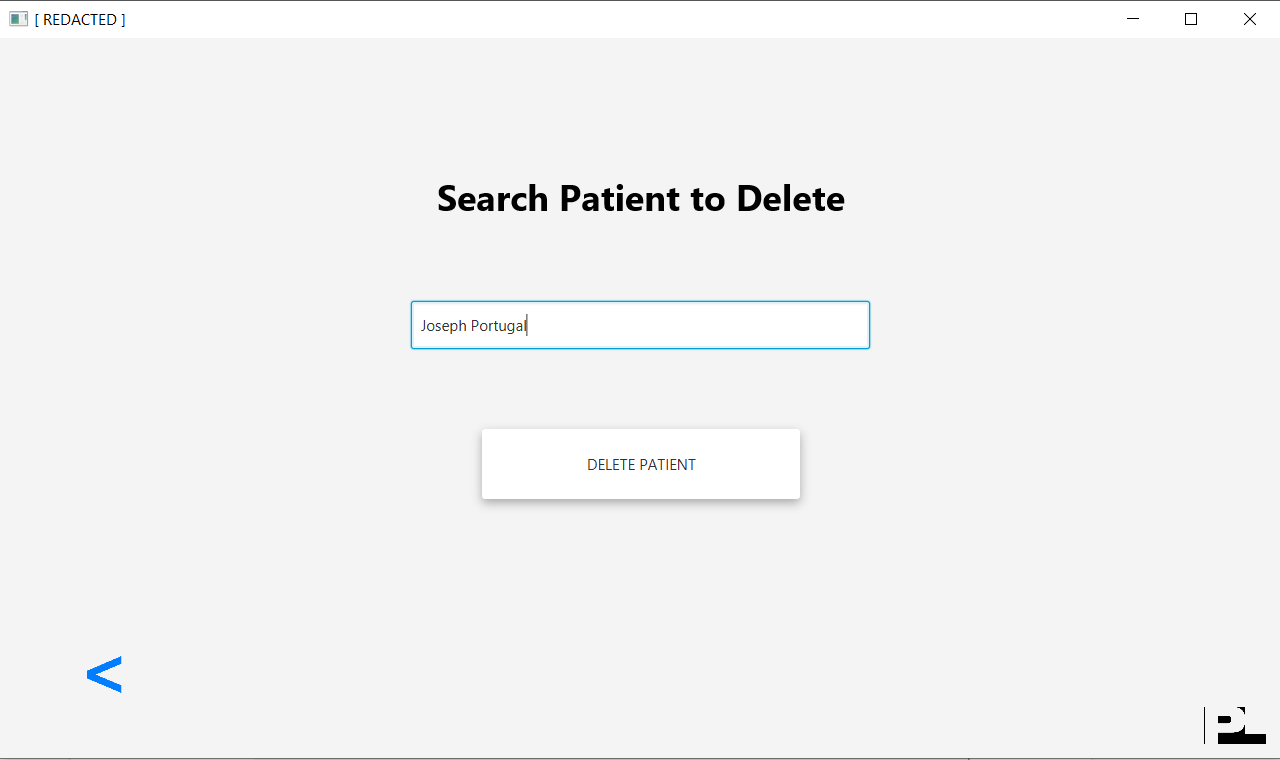
1. If the user were to choose the “Edit Patient Information,” the user will be directed to an input:



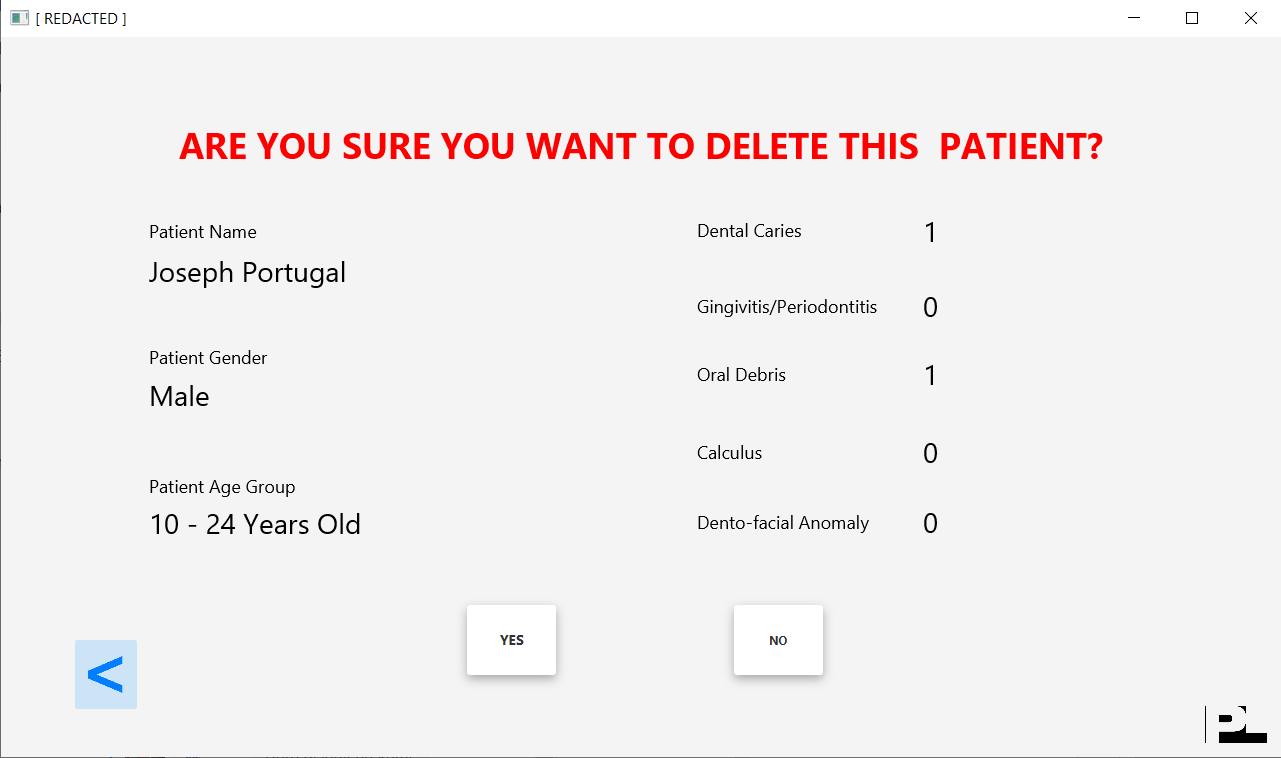
1. When the user has already inputted the patient’s name and clicks the button, he/she will be directed to a new interface that shows the patient’s current information; however, it is now editable. If the user were to change any information, it would reflect on the Excel file.



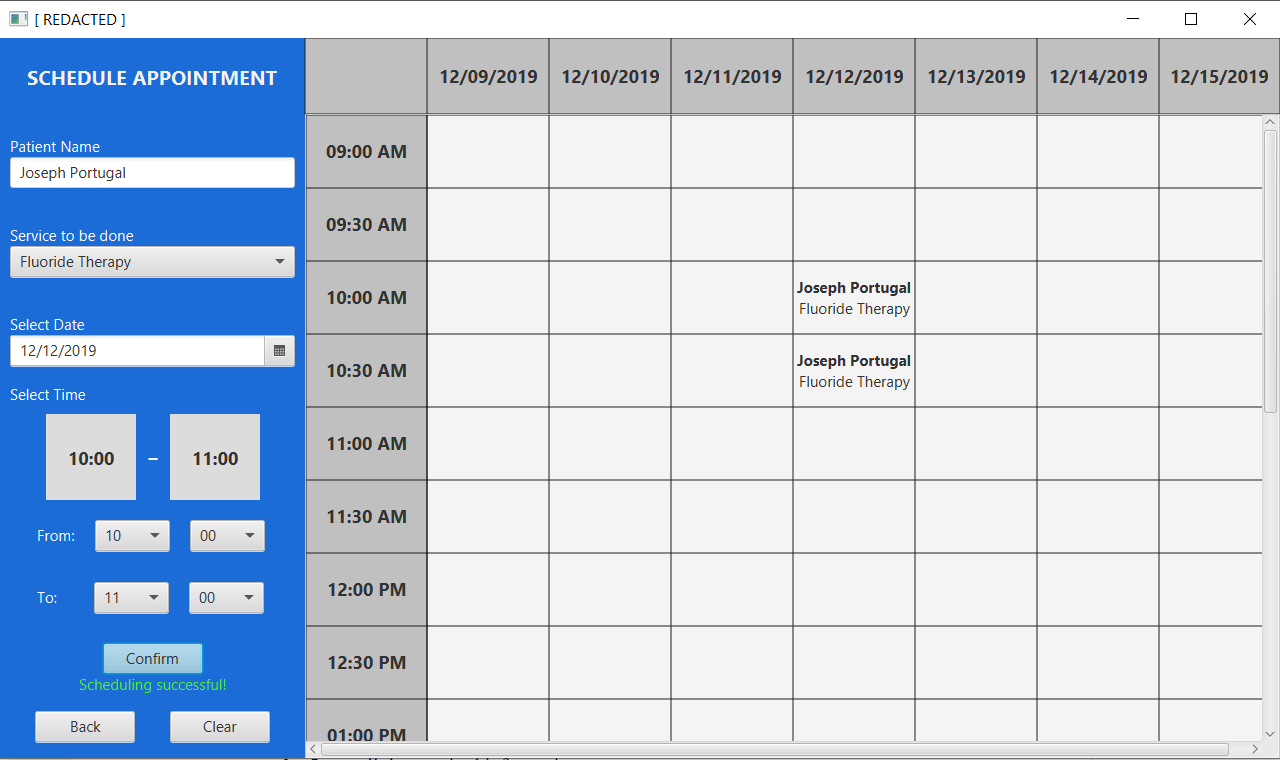
1. If the user were to choose the “Delete Patient Information,” the user will be directed to an input.



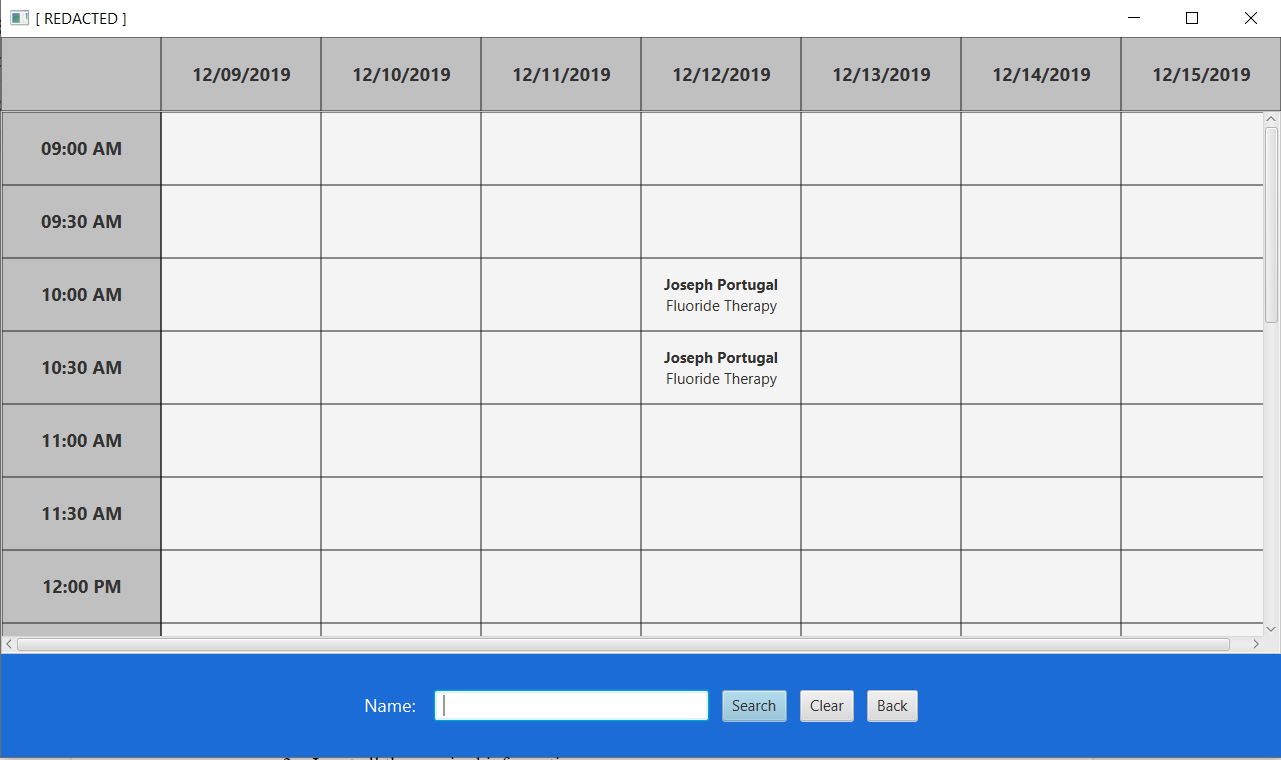
1. After the user inputs the name, the program asks the user whether he/she is certain of deleting the patient’s profile. If he/she agrees, the patient’s row in the Excel file shall be deleted. Otherwise, the user will be directed back to the main menu.



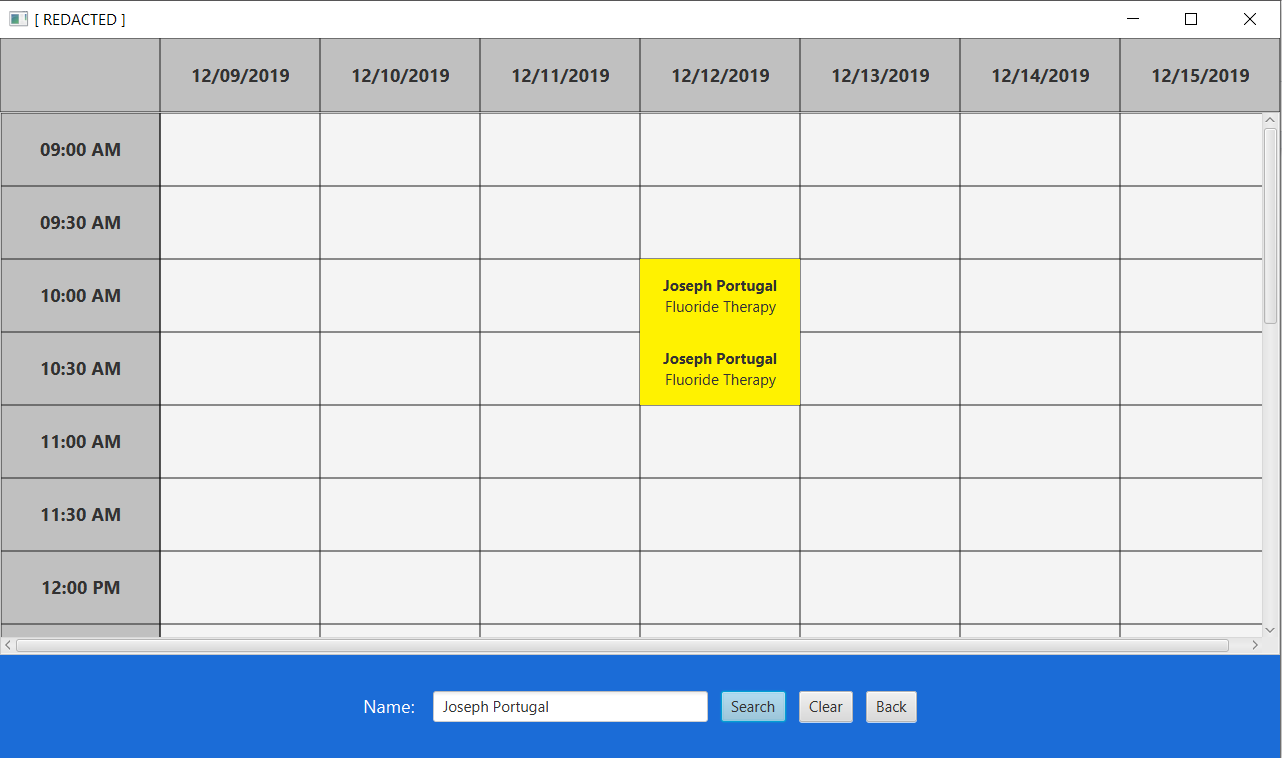
1. Aside from the search, edit, and delete patient information functions, the user can now schedule dental appointments. If one were to look to the right of the interface, it contains the required inputs for the appointment. The input for the patient name requires a text input. On the other hand, the service input is a drop-down menu. Hence, the user has to select a service from the choices. Next, the date selection is a DatePicker. If the user presses the calendar icon on the right, it displays a calendar and the user is free to choose a date. Once so, the selected date reflects on the input. Lastly, the inputs for the time are also drop-down menus. The user has to select a duration of the appointment. Once complete, the user can now confirm the appointment and it will now be illustrated on the timetable located on the right side of the interface. This schedule will also be stored in the Excel file.



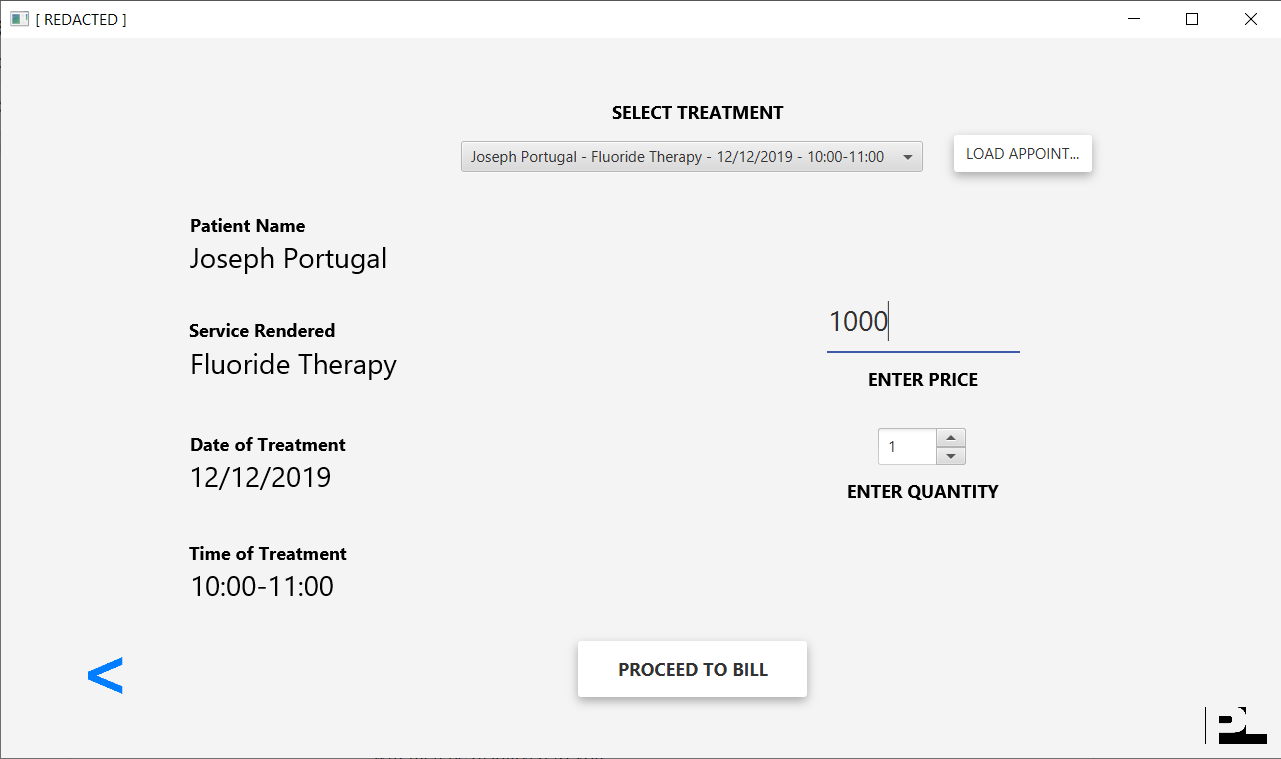
1. Once the user has set scheduled appointment/s, he or she can now use the “Search Schedule” function. Here, the interface shows the timetable containing the previously set appointments. Located at the bottom is the search input.



1. Once the user inputs the name and presses the “Search” button, the program highlights the patient’s scheduled appointments in yellow, which is an indication of the planned consultation.



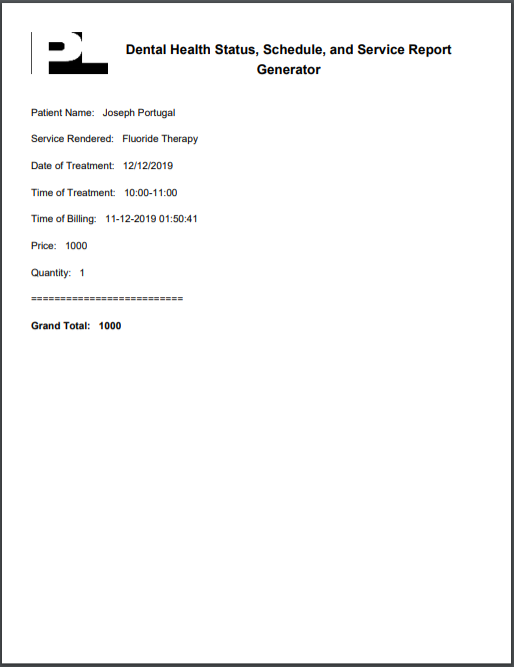
1. After the appointment, the user can now bill the patient for the service. The program asks the user to select from the previously done appointments. Once chosen and the “LOAD APPOINTMENT” button is pressed, the information of the appointment is displayed to the user. Here, the user can now input the desired price and select the quantity of the service done.



1. After the user decides to proceed to billing by clicking on the “PROCEED TO BILL” button, the information is then confirmed to the user. The user can now save the receipt as a PDF file.



1. The receipt is saved in the “Receipts” folder of the project and is outlined as below:



1. Step by step process on how to use the program (User Guide)
2. Click on the “Register New Patient” to register a new patient profile
3. Input all the required information
4. You may press the back button to return to the Main Menu
5. If you wish to search, edit, or delete a patient profile, click on the “Search/Edit/Delete Patient” button
   1. If you wish to search, click on the “SEARCH PATIENT INFORMATION” button and input the patient name you desire to search. The patient profile will then be displayed to you.
   2. If you wish to edit, click on the “EDIT PATIENT INFORMATION” button and input the patient name to edit that name’s profile. You will then be redirected to a new interface to edit any desired information of the profile.
   3. If you wish to delete, click on the “DELETE PATIENT INFORMATION” button and input the patient name to delete that name’s profile. After so, you will be asked whether you are certain that you want to delete the patient profile or not. If yes, the profile will be deleted. Otherwise, you will be taken back to the Main Menu.
6. If you wish to schedule an appointment for a patient, press the “Schedule Appointment” button
   1. If the patient is a new patient, select the “NEW PATIENT” button. Here, you will be asked to register the patient first before scheduling.
   2. If the patient is an old patient, select the “OLD PATIENT” button. Once so, you can now directly schedule the patient appointment. You have to input the patient’s name, the service to be done, and select the date and time of the appointment. When you click on the “Confirm” button, the schedule will appear on the week’s timetable if it were not in conflict with another scheduled appointment.
7. If you wish to search for a scheduled appointment, press the “Search Schedule” button.
   1. At this moment, the week’s timetable will be displayed to you along with the patients’ scheduled appointments. At the bottom of the interface is an input field that is used to search for a patient’s appointment. Input the patient’s name and click on the “Search” button. Eventually, that patient’s appointments will be highlighted in yellow to indicate the appointment’s date and time.
8. If you wish to provide the bill for the service that was done to a patient, click on “Billing” button.
   1. From this, you will be asked to select the patient’s appointment. After clicking the “LOAD APPOINTMENT” button on the right, the appointment details will be displayed to the user.
   2. Then, the user has to input the price and quantity of the service.
   3. Once you press the “PROCEED TO BILL” button, the receipt will then be issued as a PDF file located in the “Receipts” folder of the project.