Java Project Bike Rental TFI

By Matthew O’Connell Cantillon

CADVS2

Contents

[Requirements for code 2](#_Toc184987689)

[Design 3](#_Toc184987690)

[Scenes 5](#_Toc184987691)

[Code 6](#_Toc184987692)

[Main scene 6](#_Toc184987693)

[Login.java 6](#_Toc184987694)

[Controller.java 6](#_Toc184987695)

[Rent Bike Controller 6](#_Toc184987696)

[Return Bike Controller 6](#_Toc184987697)

[Maintenance 6](#_Toc184987698)

[Manager 6](#_Toc184987699)

[Where its all stored 7](#_Toc184987700)

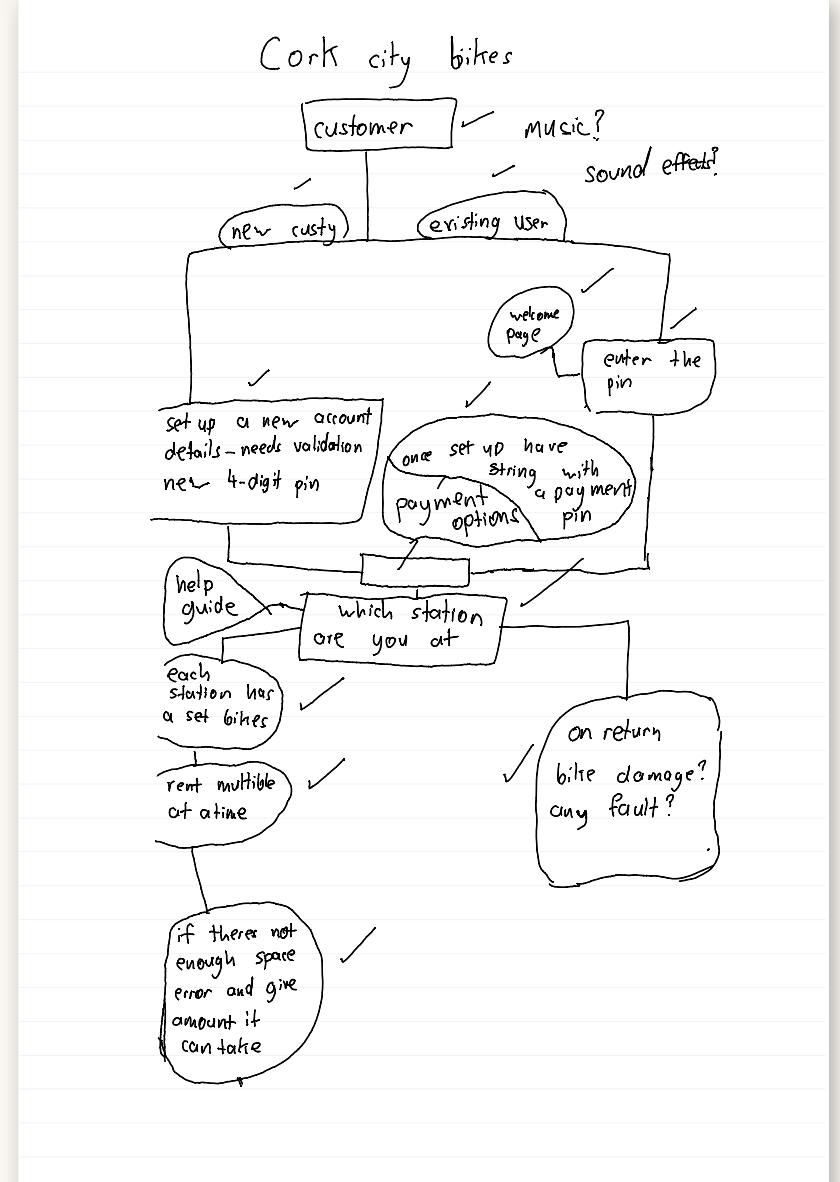
# Requirements for code

The project must allow Customers to rent, return bikes and to register for making an account

The project has different stations and number of bikes the user can rent/ rent from

The project must also have a maintenance scene for any issues the customer might like to submit, and a manager’s scene to allow for viewing revenue generated by customers.

The project must run and be able to withstand the Highly feared Bernie Trials of Doom.



# Design

Part 2

The Renting and returning sides.

Once you enter your pin and log in, you will be taken to a page that gives you the option to travel to either the renting side or the returning side

The Renting Page gives you three forms to fill out

1. Which station are you at
2. How many bikes
3. How long will you want

It has a label that updates with the overall cost, so you stay aware of how much you are spending. It also has a lovely feature where it will deduct the cost from the amount you submitted for a top-up and let you know if it needs to charge more from the card tied to your account.

The Returning Page gives the user three options, two text fields to report damage or if the bike needs maintenance and the final one is to simply return all bikes and logout to the main page

Part 1

The design idea behind my project was to have it as simple and easy to use as possible

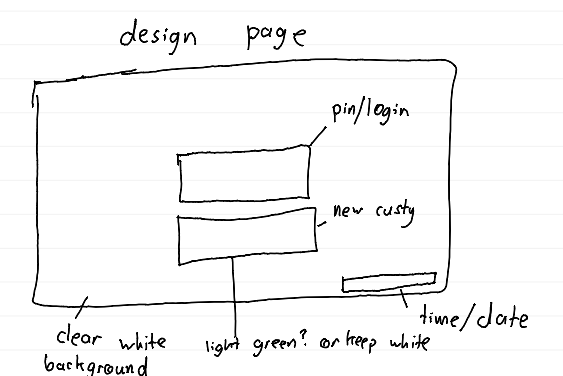
Every part would have either a label next to it or prompt text (For text Fields)

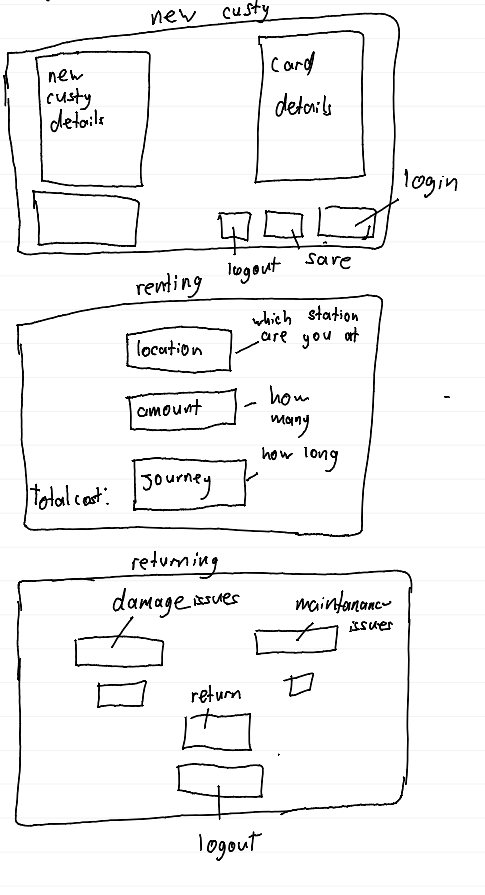
The Main Page has a few extras in it compared to the original drawing, with the addition of three extra buttons a login and a logout and a sign up.

The main page also has the live time and date as a bit of a feature rather than a usage.

The New Customer Page has two list views with labels and text fields for the user to input information such as name phone number county and card details (as well as other basic information)

It has three buttons for navigation a back button which takes you to the main page again, a save button which sends it all in for validation and then shows you an alert box, and then a login button which also takes you back to the login page for you to enter the newly made pin



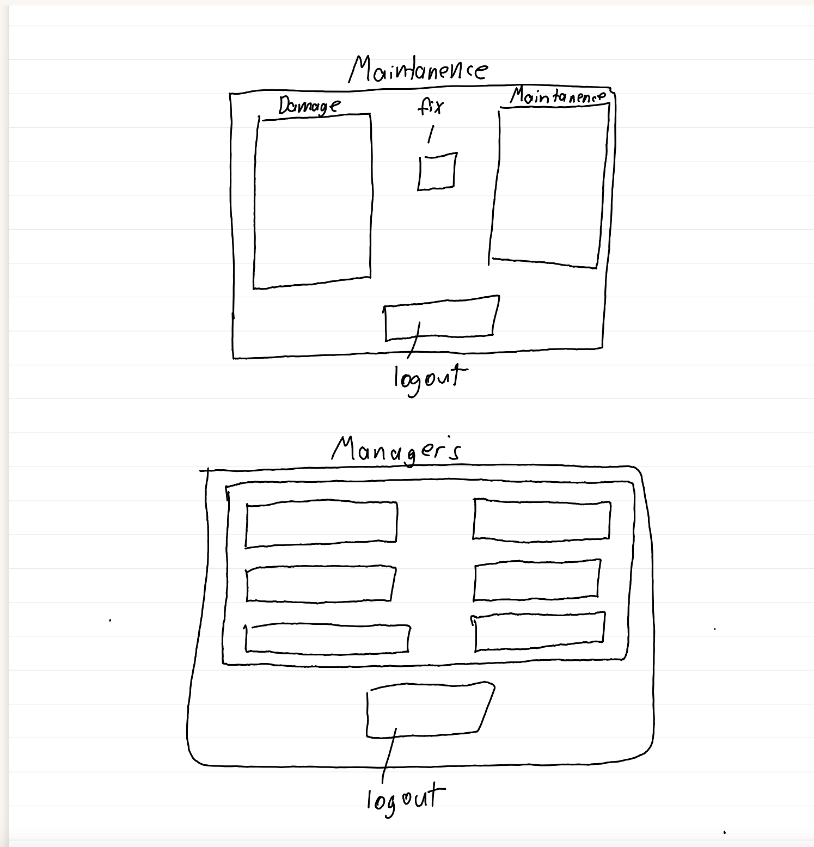


Part 3

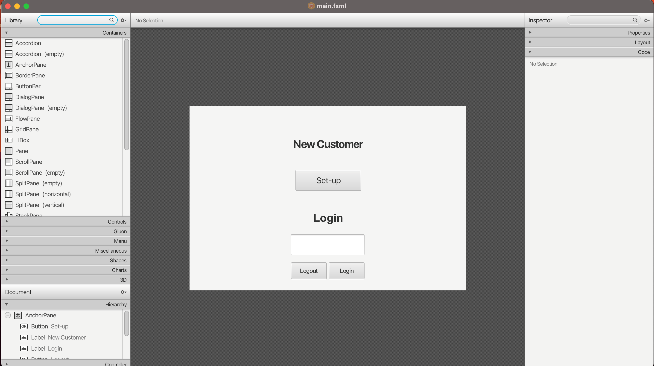
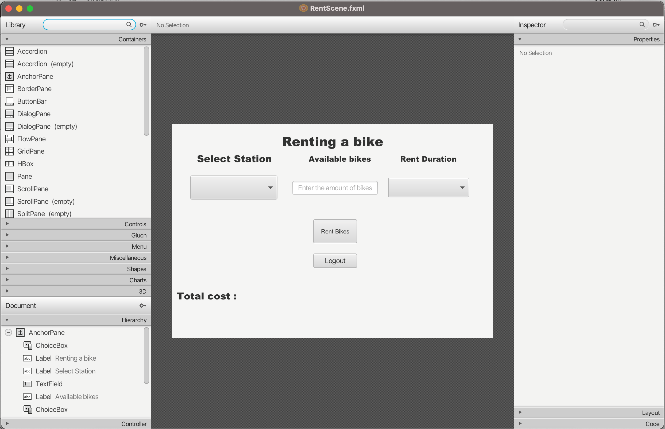
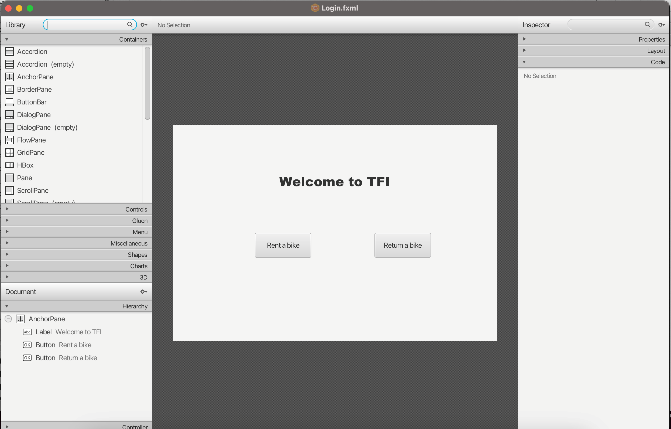
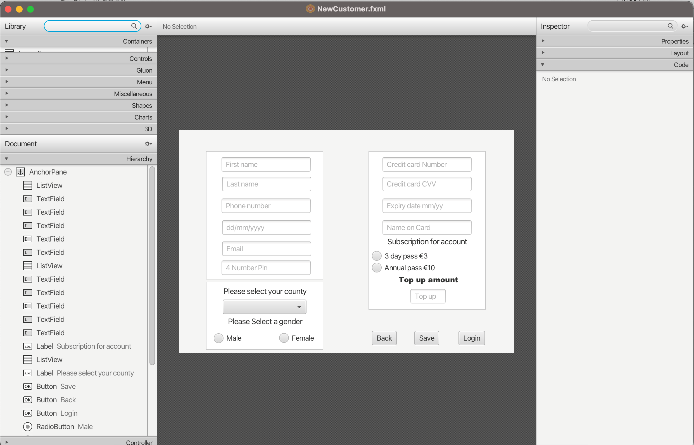
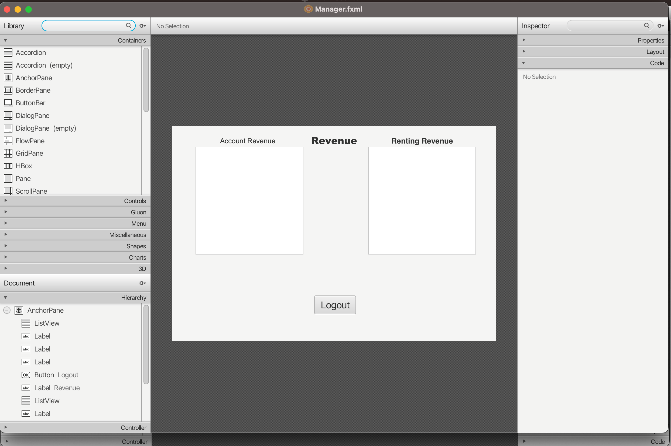
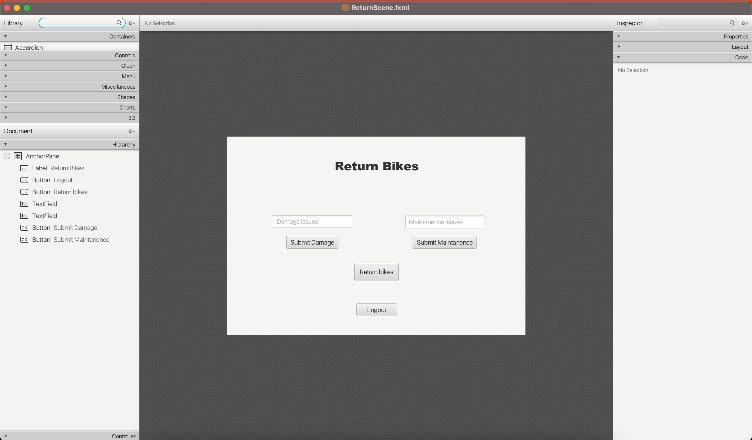
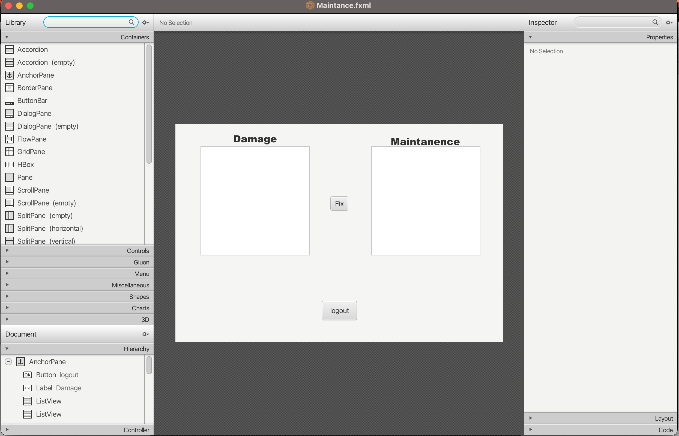
As an admin Staff the two important scenes are Maintenance and Manager.

The Maintenance has a list view with a bunch of labels that update depending on what the customer enters in the returning scene as well as a fix button which clears all the issues off, which “fixes” the bikes

The Managers scene again has a logout button and a list view with updating scenes, this is for showing how much revenue is being generated by the station and it also shows customer revenue as well



# Scenes



# Code

## Main scene

package application;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.stage.Stage;

import javafx.scene.Parent;

import javafx.scene.Scene;

public class Main extends Application {

@Override

public void start(Stage Stage) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource("main.fxml"));

Parent root = loader.load();

Scene scene = new Scene(root, 600, 400);

String css = this.getClass().getResource("application.css").toExternalForm();

scene.getStylesheets().add(css);

Stage.setScene(scene);

Stage.show();

} catch(Exception e) {

e.printStackTrace();

}

}

public static void main(String[] args) {

launch(args);

}}

## Login.java

package application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Stage;

public class Login {

public void start(Stage Stage) {

try {

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

Scene scene = new Scene(root,600,400);

//scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

String css = this.getClass().getResource("application.css").toExternalForm();

scene.getStylesheets().add(css);

Stage.setScene(scene);

Stage.show();

} catch(Exception e) {

e.printStackTrace();

}

}}

## Controller.java

package application;

import java.io.IOException;

import java.net.URL;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.Period;

import java.time.format.DateTimeFormatter;

import java.util.ResourceBundle;

import javafx.animation.Animation;

import javafx.animation.KeyFrame;

import javafx.animation.Timeline;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Node;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Alert.AlertType;

import javafx.scene.control.Button;

import javafx.scene.control.ButtonType;

import javafx.scene.control.ChoiceBox;

import javafx.scene.control.Label;

import javafx.scene.control.RadioButton;

import javafx.scene.control.TextField;

import javafx.scene.control.Toggle;

import javafx.scene.control.ToggleGroup;

import javafx.scene.layout.AnchorPane;

import javafx.stage.Stage;

import javafx.util.Duration;

public class Controller implements Initializable{

@FXML

private ChoiceBox<String> myChoiceBox;

private String[] county = {"Cork", "Dublin", "Kerry", "Antrim", "Armagh", "Carlow", "Cavan", "Clare", "Donegal", "Galway", "Kildare", "KilKenny", "Laois", "Leitrim","Limerick", "Mayo", "Other"};

@FXML

private TextField loginTextField;

@FXML

private Button loginButton;

@FXML

private Button logoutButton;

@FXML

private AnchorPane scenePane;

private static String CUSTOMER\_CODE = "";

private static String MAINTENANCE\_CODE = "2567";

private static String MANAGER\_CODE = "3579";

private double deposit = 150;

private static double topup = 0;

double bikecost = 0.0;

@FXML

public Label dateTimeLabel;

@FXML

public Label dateTimencLabel;

@FXML

private TextField nameFField;

@FXML

private TextField nameSField;

@FXML

private TextField phoneNumField;

@FXML

private TextField dobField;

@FXML

private TextField emailField;

@FXML

private TextField cardNumberField;

@FXML

private TextField expiryField;

@FXML

private TextField cardNameField;

@FXML

private TextField cvvField;

@FXML

private TextField pinField;

@FXML

private Label myLabel;

@FXML

private Label subLabel;

@FXML

private RadioButton maleRButton, femaleRbutton, BadAssATron;

@FXML

private Button switchtoRent;

@FXML

private RadioButton threedayRadio;

@FXML

private RadioButton annualdayButton;

@FXML

private ToggleGroup optionGroup;

@FXML

private Button switchtoReturn;

@FXML

private TextField topUpField;

@FXML

private void checkPayment() {

cardNumberField.textProperty().addListener((observable, oldValue, newValue) -> {

String digitsOnly = newValue.replaceAll("[^\\d]", "");

StringBuilder formatted = new StringBuilder();

for (int i = 0; i < digitsOnly.length(); i++) {

if (i > 0 && i % 4 == 0) {

formatted.append("/");

}

formatted.append(digitsOnly.charAt(i));

}

cardNumberField.setText(formatted.toString());

cardNumberField.positionCaret(formatted.length());

});

}

@FXML

private void checkdob() {

dobField.textProperty().addListener((observable, oldValue, newValue) -> {

String digitsOnly = newValue.replaceAll("[^\\d]", "");

if (digitsOnly.length() > 8) digitsOnly = digitsOnly.substring(0, 8);

StringBuilder formatted = new StringBuilder(digitsOnly);

if (digitsOnly.length() > 2) formatted.insert(2, "/");

if (digitsOnly.length() > 4) formatted.insert(5, "/");

dobField.setText(formatted.toString());

dobField.positionCaret(formatted.length());

});

}

@FXML

private void expirycheck() {

expiryField.textProperty().addListener((observable, oldValue, newValue) -> {

String digitsOnly = newValue.replaceAll("[^\\d]", "");

if (digitsOnly.length() > 4) digitsOnly = digitsOnly.substring(0, 4);

StringBuilder formatted = new StringBuilder(digitsOnly);

if (digitsOnly.length() > 2) formatted.insert(2, "/");

expiryField.setText(formatted.toString());

expiryField.positionCaret(formatted.length());

});

}

@FXML

private void verifyPin() {

String enteredPin = loginTextField.getText();

// Validate PIN length and format

if (enteredPin.length() != 4) {

showAlert("Invalid PIN", "Please enter a 4-digit PIN");

return;

}

// Check if PIN matches

if (enteredPin.equals(CUSTOMER\_CODE)) {

try {

// Load the new scene

FXMLLoader loader = new FXMLLoader(getClass().getResource("Login.fxml"));

Parent root = loader.load();

Scene scene = new Scene(root);

// Get the current stage

Stage stage = (Stage) loginTextField.getScene().getWindow();

stage.setScene(scene);

} catch (IOException e) {

e.printStackTrace();

showAlert("Error", "Could not load the next scene");

}

} else if (enteredPin.equals(MAINTENANCE\_CODE)) {

try {

// Load the new scene

FXMLLoader loader = new FXMLLoader(getClass().getResource("Maintance.fxml"));

Parent root = loader.load();

Scene scene = new Scene(root);

// Get the current stage

Stage stage = (Stage) loginTextField.getScene().getWindow();

stage.setScene(scene);

} catch (IOException e) {

e.printStackTrace();

showAlert("Error", "Could not load the next scene");

}

showAlert("Success", "Maintenance Login Successful");

} else if (enteredPin.equals(MANAGER\_CODE)) {

try {

// Load the new scene

FXMLLoader loader = new FXMLLoader(getClass().getResource("Manager.fxml"));

Parent root = loader.load();

Scene scene = new Scene(root);

// Get the current stage

Stage stage = (Stage) loginTextField.getScene().getWindow();

stage.setScene(scene);

} catch (IOException e) {

e.printStackTrace();

showAlert("Error", "Could not load the next scene");

}

showAlert("Success", "Manager Login Successful");

} else {

showAlert("Error", "Incorrect PIN");

loginTextField.clear();

}

}

private void showAlert(String title, String content) {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(content);

alert.showAndWait();

}

@FXML

public void pass(ActionEvent event) {

// Optional: Additional logging or processing

if (threedayRadio.isSelected()) {

}

if (annualdayButton.isSelected()) {

}

}

Stage stage;

public void logout(ActionEvent event) {

Alert alert = new Alert(AlertType.CONFIRMATION);

alert.setTitle("Logout");

alert.setHeaderText("Your about to logout!");

alert.setContentText("Are you sure you want to exit?");

if(alert.showAndWait().get() == ButtonType.OK) {

stage = (Stage) scenePane.getScene().getWindow();

stage.close();

}}

public void gender(ActionEvent event) {

ToggleGroup group = new ToggleGroup();

if(maleRButton.isSelected()) {

myLabel.setText(maleRButton.getText());

maleRButton.setUserData("male");

maleRButton.setSelected(true);

}

if(femaleRbutton.isSelected()) {

myLabel.setText(femaleRbutton.getText());

femaleRbutton.setUserData("Female");

femaleRbutton.setSelected(true);

}

/\*else if(BadAssATron.isSelected()) {

myLabel.setText(BadAssATron.getText());

BadAssATron.setUserData("BadAssATron");

BadAssATron.setSelected(true);

}\*/

maleRButton.setToggleGroup(group);

femaleRbutton.setToggleGroup(group);

// BadAssATron.setToggleGroup(group);

group.selectedToggleProperty().addListener((obserableValue, old\_toggle, new\_toggle) -> {

if (group.getSelectedToggle() != null) {

}

});

}

public void submit(ActionEvent event) {

if (bikeTypeGroup == null || bikeTypeGroup.getSelectedToggle() == null) {

showAlert(Alert.AlertType.WARNING, "Selection Error", "Please select a bike pass type");

return;

}

if (nameFField.getText().isEmpty() || nameSField.getText().isEmpty()) {

showAlert(Alert.AlertType.ERROR, "Input Error", "Please fill in your name correctly. Only letters are allowed.");

return;

}

// Enhanced DOB validation

String dobText = dobField.getText();

if (!isValidDateOfBirth(dobText)) {

showAlert(Alert.AlertType.ERROR, "Input Error",

"Please enter a valid date of birth. You must be at least 14 years old.");

return;

}

if (phoneNumField.getText().isEmpty() || emailField.getText().isEmpty() || !emailField.getText().contains("@")) {

showAlert(Alert.AlertType.ERROR, "Input Error", "Please provide a valid phone number and email address. Email must include '@'.");

return;

}

String phoneNumber = phoneNumField.getText();

// Validate if the phone number is numeric

if (!phoneNumber.matches("\\d+")) { // Checks if the input contains only digits

showAlert(Alert.AlertType.ERROR, "Input Error", "Phone number must contain only numbers.");

return;

}

// Optionally, check phone number length (e.g., must be 10 digits)

if (phoneNumber.length() != 10) {

showAlert(Alert.AlertType.ERROR, "Input Error", "Phone number must be exactly 10 digits long.");

return;

}

if (cardNameField.getText().isEmpty()) {

showAlert(Alert.AlertType.ERROR, "Input Error", "Please fill in your name correctly. Only letters are allowed.");

return;

}

if (cardNumberField.getText().isEmpty() || cvvField.getText().isEmpty()) {

showAlert(Alert.AlertType.ERROR, "Input Error", "Please provide a valid credit card number and or cvv.");

return;

}

String creditCard = cardNumberField.getText();

String creditCVV = cvvField.getText();

// Validate if the phone number contains only digits or '/'

if (!creditCard.matches("[\\d/]+")) { // Allows digits and '/' characters

showAlert(Alert.AlertType.ERROR, "Input Error", "Credit Card number must contain only numbers or '/'.");

return;

}

// Optionally, check phone number length (e.g., must be 10 digits)

if (creditCard.length() != 19) {

showAlert(Alert.AlertType.ERROR, "Input Error", "Credit Card must be 16 digits long.");

return;

}

if (!creditCVV.matches("\\d+")) { // Checks if the input contains only digits

showAlert(Alert.AlertType.ERROR, "Input Error", "Cvv number must contain only numbers.");

return;

}

// Optionally, check phone number length (e.g., must be 10 digits)

if (creditCVV.length() != 3) {

showAlert(Alert.AlertType.ERROR, "Input Error", "Cvv number must be 3 digits long.");

return;

}

CUSTOMER\_CODE = pinField.getText();

// Validate if the phone number is numeric

if (!CUSTOMER\_CODE.matches("\\d+")) { // Checks if the input contains only digits

showAlert(Alert.AlertType.ERROR, "Input Error", "Pin number must contain only numbers.");

return;

}

// Optionally, check phone number length (e.g., must be 10 digits)

if (CUSTOMER\_CODE.length() != 4) {

showAlert(Alert.AlertType.ERROR, "Input Error", "Pin number must be 4 digits long.");

return;}

double topupAmount = 0.0;

if (topUpField != null && !topUpField.getText().trim().isEmpty()) {

try {

topupAmount = Double.parseDouble(topUpField.getText().trim());

// Additional validation if needed

if (topupAmount < 0) {

showAlert(Alert.AlertType.ERROR, "Input Error", "Top-up amount cannot be negative");

return;

}

// Set the static topup variable

RentBikeController.topup = String.valueOf(topupAmount);

} catch (NumberFormatException e) {

showAlert(Alert.AlertType.ERROR, "Input Error", "Please enter a valid top-up amount");

return;

}

}

Alert alert = new Alert(AlertType.CONFIRMATION);

alert.setTitle("Payment Confirmation");

alert.setHeaderText(null);

alert.setContentText("When renting a bike a deposit of €150 will be placed on hold\n\n"

+ String.format("Top-up Amount: €%.2f\n", topupAmount)

+ String.format("Deposit Amount: €%.2f\n", deposit)

+ "Cork City Bikes Terms and Conditions:\n"

+ "1. Users must be 14 years or older.\n"

+ "2. Bicycles must be returned within the rental period.\n"

+ "3. Any damage to bicycles will incur repair costs.\n"

+ "4. Users are responsible for their own safety while using the bikes.\n"

+ "5. Cork City Bikes is not liable for personal injuries.\n\n"

+ "By renting a bike, you agree to abide by these terms.\n");

alert.showAndWait(); // Add this line to actually display the alert

}

@FXML

private ToggleGroup bikeTypeGroup; // This should match the fx:id in FXML if you use one

@FXML

private int calculateBikeCost() {

if (bikeTypeGroup == null) {

showAlert(Alert.AlertType.WARNING, "Error", "Bike type group initialization failed");

return 0;

}

Toggle selectedToggle = bikeTypeGroup.getSelectedToggle();

if (selectedToggle == null) {

showAlert(Alert.AlertType.WARNING, "Selection Error", "Please select a pass type");

return 0;

}

Object userData = selectedToggle.getUserData();

if (userData instanceof Number) {

return ((Number) userData).intValue();

}

showAlert(Alert.AlertType.WARNING, "Error", "Invalid pass type selection");

return 0;

}

@FXML

public double calculateTotalCost() {

int bikeCost = calculateBikeCost();

return deposit + topup + bikeCost;

}

public void showAlert(Alert.AlertType alertType, String title, String message) {

Alert alert = new Alert(alertType);

alert.setTitle(title);

alert.setHeaderText(null); // No header text

alert.setContentText(message);

alert.showAndWait();

}

private Stage stage1;

private Scene scene;

public void switchtoMain(ActionEvent event) throws IOException {

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

stage1 = (Stage)((Node)event.getSource()).getScene().getWindow();

scene = new Scene(root);

stage1.setScene(scene);

stage1.show();

}

public void switchtoNewCustomer(ActionEvent event) throws IOException {

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

stage1 = (Stage)((Node)event.getSource()).getScene().getWindow();

scene = new Scene(root);

stage1.setScene(scene);

stage1.show();

}

public void switchtoLogin(ActionEvent event) throws IOException {

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

stage1 = (Stage)((Node)event.getSource()).getScene().getWindow();

scene = new Scene(root);

stage1.setScene(scene);

stage1.show();

}

public void switchtoRent(ActionEvent event) throws IOException {

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

stage1 = (Stage)((Node)event.getSource()).getScene().getWindow();

scene = new Scene(root);

stage1.setScene(scene);

stage1.show();

}

public void switchtoReturn(ActionEvent event) throws IOException {

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

stage1 = (Stage)((Node)event.getSource()).getScene().getWindow();

scene = new Scene(root);

stage1.setScene(scene);

stage1.show();

}

private boolean isValidDateOfBirth(String dob) {

// Check if the date is in the correct format (DD/MM/YYYY)

if (!dob.matches("\\d{2}/\\d{2}/\\d{4}")) {

return false;

}

try {

// Parse the date

String[] parts = dob.split("/");

int day = Integer.parseInt(parts[0]);

int month = Integer.parseInt(parts[1]);

int year = Integer.parseInt(parts[2]);

// Validate month range

if (month < 1 || month > 12) {

return false;

}

// Validate day range based on month

int[] daysInMonth = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

if (day < 1 || day > daysInMonth[month - 1]) {

return false;

}

// Age validation (e.g., must be at least 14 years old)

LocalDate birthDate = LocalDate.of(year, month, day);

LocalDate currentDate = LocalDate.now();

Period age = Period.between(birthDate, currentDate);

return age.getYears() >= 14;

} catch (NumberFormatException | ArrayIndexOutOfBoundsException e) {

return false;

}

}

@Override

public void initialize(URL arg0, ResourceBundle arg1) {

bikeTypeGroup = new ToggleGroup();

if (threedayRadio != null) {

threedayRadio.setToggleGroup(bikeTypeGroup);

threedayRadio.setUserData(3);

}

if (annualdayButton != null) {

annualdayButton.setToggleGroup(bikeTypeGroup);

annualdayButton.setUserData(10);

}

if (myChoiceBox != null) {

myChoiceBox.getItems().addAll(county);

}

if (dateTimeLabel == null) {

return;

}

Timeline timeline = new Timeline(new KeyFrame(Duration.seconds(1), event -> {

LocalDateTime currentTime = LocalDateTime.now();

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");

String formattedTime = currentTime.format(formatter);

dateTimeLabel.setText(formattedTime);

}));

timeline.setCycleCount(Animation.INDEFINITE);

timeline.play();

}}

## Rent Bike Controller

package application;

import java.io.IOException;

import java.lang.reflect.Field;

import java.util.ArrayList;

import java.util.LinkedHashMap;

import java.util.List;

import java.util.Map;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Node;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.ChoiceBox;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

public class RentBikeController {

// Static list to maintain bike stations across scenes

private static List<BikeStation> stations = new ArrayList<>();

public static String topup = "";

public static String rentStation = "";

// Bike Station class

class BikeStation {

private String name;

private int totalCapacity;

private int availableBikes;

public BikeStation(String name, int totalCapacity) {

this.name = name;

this.totalCapacity = totalCapacity;

this.availableBikes = totalCapacity;

}

// Getters

public String getName() { return name; }

public int getAvailableBikes() { return availableBikes; }

public int getTotalCapacity() { return totalCapacity; }

}

// Rental duration and pricing

private static final Map<String, Double> RENTAL\_RATES = new LinkedHashMap<>();

@FXML

private ChoiceBox<String> stationChoiceBox;

@FXML

private ChoiceBox<String> rentalDurationChoiceBox;

@FXML

private Label availableBikesLabel;

@FXML

private TextField bikeRentQuantityField;

@FXML

private Label totalCostLabel;

@FXML

public void initialize() {

// Initialize stations if not already done

if (stations.isEmpty()) {

stations.add(new BikeStation("Ucc", 10));

stations.add(new BikeStation("Morrisons island", 5));

stations.add(new BikeStation("Kent Station", 8));

stations.add(new BikeStation("Grand Parade", 7));

}

// Populate station choice box

stationChoiceBox.getItems().clear();

for (BikeStation station : stations) {

stationChoiceBox.getItems().add(station.getName());

}

// Initialize rental rates

RENTAL\_RATES.put("1 Hour - €0.50", 0.5);

RENTAL\_RATES.put("2 Hours - €1.50", 1.5);

RENTAL\_RATES.put("3 Hours - €3.50", 3.50);

RENTAL\_RATES.put("4 Hours - €6.50", 6.50);

// Populate rental duration choice box

rentalDurationChoiceBox.getItems().addAll(RENTAL\_RATES.keySet());

// Station selection listener

stationChoiceBox.setOnAction(event -> updateAvailableBikesDisplay());

// Bike quantity listener to update total cost

bikeRentQuantityField.textProperty().addListener((observable, oldValue, newValue) -> {

updateTotalCost();

});

// Rental duration listener to update total cost

rentalDurationChoiceBox.setOnAction(event -> {

updateTotalCost();

});

}

// Update available bikes display

private void updateAvailableBikesDisplay() {

String selectedStation = stationChoiceBox.getValue();

BikeStation station = findStationByName(selectedStation);

if (station != null) {

availableBikesLabel.setText("Available Bikes: " + station.getAvailableBikes());

}

}

private void updateTotalCost() {

try {

// Validate inputs

int bikeQuantity = 0;

if (!bikeRentQuantityField.getText().isEmpty()) {

bikeQuantity = Integer.parseInt(bikeRentQuantityField.getText());

if (bikeQuantity < 0) {

totalCostLabel.setText("Quantity must be non-negative");

return;

}

}

String selectedDuration = rentalDurationChoiceBox.getValue();

String selectedStation = stationChoiceBox.getValue();

if (selectedStation == null || selectedDuration == null) {

totalCostLabel.setText("Select station and duration");

return;

}

BikeStation station = findStationByName(selectedStation);

if (station == null || bikeQuantity > station.getAvailableBikes()) {

totalCostLabel.setText("Invalid Bike Quantity");

return;

}

// Safely handle topup amount

double topupAmount = 0.0;

try {

topupAmount = topup != null && !topup.isEmpty()

? Double.parseDouble(topup)

: 0.0;

} catch (NumberFormatException e) {

totalCostLabel.setText("Invalid Topup Amount");

return;

}

double hourlyRate = RENTAL\_RATES.get(selectedDuration);

double totalCost = bikeQuantity \* hourlyRate;

// Calculate remaining amount

double remainingCost = totalCost - topupAmount;

// Prepare the display message

String costMessage;

if (remainingCost > 0) {

costMessage = String.format(

"Total Cost: €%.2f\n" +

"Top-up Applied: €%.2f\n" +

"Remaining to be Charged: €%.2f",

totalCost, topupAmount, remainingCost

);

} else {

costMessage = String.format(

"Total Cost: €%.2f\n" +

"Top-up Applied: €%.2f\n" +

"Excess Top-up: €%.2f (will be refunded)",

totalCost, topupAmount, Math.abs(remainingCost)

);

}

// Store total cost

// Manager.rentRev = String.valueOf(totalCost);

totalCostLabel.setText(costMessage);

} catch (NumberFormatException e) {

totalCostLabel.setText("Invalid Numeric Input");

}

}

// Rent bikes method

@FXML

private void rentBikes() {

// Validate all inputs

String selectedStation = stationChoiceBox.getValue();

String selectedDuration = rentalDurationChoiceBox.getValue();

if (selectedStation == null || selectedDuration == null) {

showAlert("Error", "Please select a station and rental duration");

return;

}

int bikesToRent;

try {

bikesToRent = Integer.parseInt(bikeRentQuantityField.getText());

} catch (NumberFormatException e) {

showAlert("Error", "Please enter a valid number of bikes");

return;

}

BikeStation station = findStationByName(selectedStation);

// Validate rental

if (station == null || bikesToRent <= 0 || bikesToRent > station.getAvailableBikes()) {

showAlert("Error", "Invalid number of bikes to rent");

return;

}

try {

Field availableBikesField = BikeStation.class.getDeclaredField("availableBikes");

availableBikesField.setAccessible(true);

// Reduce available bikes

int currentAvailable = station.getAvailableBikes();

availableBikesField.setInt(station, currentAvailable - bikesToRent);

// Calculate costs

double hourlyRate = RENTAL\_RATES.get(selectedDuration);

double totalCost = bikesToRent \* hourlyRate;

double topupAmount = topup != null && !topup.isEmpty()

? Double.parseDouble(topup)

: 0.0;

double remainingCost = totalCost - topupAmount;

Manager.addEntry(

"N/A", // You can modify this to capture actual account revenue if needed

String.valueOf(totalCost),

selectedStation

);

String confirmationMessage;

if (remainingCost > 0) {

confirmationMessage = String.format(

"Rented %d bike(s) from %s\n" +

"Total Cost: €%.2f\n" +

"Top-up Applied: €%.2f\n" +

"Remaining to be Charged: €%.2f",

bikesToRent, selectedStation, totalCost, topupAmount, remainingCost

);

} else {

confirmationMessage = String.format(

"Rented %d bike(s) from %s\n" +

"Total Cost: €%.2f\n" +

"Top-up Applied: €%.2f\n" +

"Excess Top-up: €%.2f (will be refunded)",

bikesToRent, selectedStation, totalCost, topupAmount, Math.abs(remainingCost)

);

}

// Show confirmation

showAlert("Rental Successful", confirmationMessage);

// Update display

updateAvailableBikesDisplay();

} catch (Exception e) {

showAlert("Error", "Could not process rental");

e.printStackTrace();

}

}

// Find station by name helper method

private BikeStation findStationByName(String name) {

return stations.stream()

.filter(station -> station.getName().equals(name))

.findFirst()

.orElse(null);

}

private Stage stage1;

private Scene scene;

public void switchtoMain(ActionEvent event) throws IOException {

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

stage1 = (Stage)((Node)event.getSource()).getScene().getWindow();

scene = new Scene(root);

stage1.setScene(scene);

stage1.show();

}

// Alert helper method

private void showAlert(String title, String content) {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(content);

alert.showAndWait();

}

}

## Return Bike Controller

package application;

import java.io.IOException;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Node;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.ButtonType;

import javafx.scene.control.TextField;

import javafx.scene.control.Alert.AlertType;

import javafx.scene.layout.AnchorPane;

import javafx.stage.Stage;

public class ReturnBikeController {

//private Parent root;

@FXML

TextField damTextField;

@FXML

TextField mainTextField;

public void submitmain(ActionEvent event) throws IOException {

String maintenance = mainTextField.getText();

Maintance.maintenanceMessage = maintenance;

// Switch back to main scene

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

Stage stage = (Stage) ((Node)event.getSource()).getScene().getWindow();

Scene scene = new Scene(root);

stage.setScene(scene);

stage.show();

}

public void submitDam(ActionEvent event) throws IOException {

String damage = damTextField.getText();

Maintance.damageMessage = damage;

// Switch back to main scene

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

Stage stage = (Stage) ((Node)event.getSource()).getScene().getWindow();

Scene scene = new Scene(root);

stage.setScene(scene);

stage.show();

}

@FXML

private Button returnBike;

@FXML

private AnchorPane scenePane;

@FXML

Stage stage;

public void returnBike() {

Alert alert = new Alert(AlertType.CONFIRMATION);

alert.setTitle("Return bikes");

alert.setHeaderText("Returning all bikes");

alert.setContentText("Are you finished cycling around?");

if(alert.showAndWait().get() == ButtonType.OK) {

}}

private Stage stage1;

private Scene scene;

public void switchtoMain(ActionEvent event) throws IOException {

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

stage1 = (Stage)((Node)event.getSource()).getScene().getWindow();

scene = new Scene(root);

stage1.setScene(scene);

stage1.show();

}

}

## Maintenance

package application;

import java.io.IOException;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Node;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Label;

import javafx.stage.Stage;

public class Maintance {

// Static fields to store messages across scenes

public static String damageMessage = "";

public static String maintenanceMessage = "";

@FXML

Label label1;

@FXML

Label label6;

// Method to update labels when Maintance scene loads

public void initialize() {

if (!damageMessage.isEmpty()) {

label1.setText("Damage: " + damageMessage);

}

if (!maintenanceMessage.isEmpty()) {

label6.setText("Maintenance: " + maintenanceMessage);

}

}

// Method to clear issues and show alert

@FXML

public void fixIssues() {

damageMessage = "";

maintenanceMessage = "";

label1.setText("");

label6.setText("");

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Issues Fixed");

alert.setHeaderText(null);

alert.setContentText("All reported issues have been resolved.");

alert.showAndWait();

}

private Stage stage1;

private Scene scene;

public void switchtoMain(ActionEvent event) throws IOException {

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

stage1 = (Stage)((Node)event.getSource()).getScene().getWindow();

scene = new Scene(root);

stage1.setScene(scene);

stage1.show();

}

}

## Manager

package application;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Node;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Label;

import javafx.stage.Stage;

public class Manager {

public static List<String> accountRevenues = new ArrayList<>();

public static List<String> rentRevenues = new ArrayList<>();

public static List<String> rentStations = new ArrayList<>();

@FXML

Label labelRev1;

@FXML

Label labelRev2;

@FXML

Label labelAcc1;

@FXML

Label labelAcc2;

@FXML

Label labelStation1;

@FXML

Label labelStation2;

// Method to update labels when Maintance scene loads

public void initialize() {

// Clear previous displays

clearLabels();

// Display first entry if exists

if (!accountRevenues.isEmpty()) {

labelRev1.setText("Account Revenue 1: " + accountRevenues.get(0));

}

if (!rentRevenues.isEmpty()) {

labelAcc1.setText("Rental Revenue 1: €" + rentRevenues.get(0));

}

if (!rentStations.isEmpty()) {

labelStation1.setText("Rental Station 1: " + rentStations.get(0));

}

// Display second entry if exists

if (accountRevenues.size() > 1) {

labelRev2.setText("Account Revenue 2: " + accountRevenues.get(1));

}

if (rentRevenues.size() > 1) {

labelAcc2.setText("Rental Revenue 2: €" + rentRevenues.get(1));

}

if (rentStations.size() > 1) {

labelStation2.setText("Rental Station 2: " + rentStations.get(1));

}

}

private void clearLabels() {

labelRev1.setText("Account Revenue 1: ");

labelAcc1.setText("Rental Revenue 1: ");

labelStation1.setText("Rental Station 1: ");

labelRev2.setText("Account Revenue 2: ");

labelAcc2.setText("Rental Revenue 2: ");

labelStation2.setText("Rental Station 2: ");

}

// Optional method to add new entry

public static void addEntry(String accountRev, String rentRev, String rentStation) {

// If we already have 2 entries, remove the first one

if (accountRevenues.size() >= 2) {

accountRevenues.remove(0);

rentRevenues.remove(0);

rentStations.remove(0);

}

// Add new entry

accountRevenues.add(accountRev);

rentRevenues.add(rentRev);

rentStations.add(rentStation);

}

private Stage stage1;

private Scene scene;

public void switchtoMain(ActionEvent event) throws IOException {

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

stage1 = (Stage)((Node)event.getSource()).getScene().getWindow();

scene = new Scene(root);

stage1.setScene(scene);

stage1.show();}}

# 

# Where its all stored

