Criterion C: Development

Word Count = 1151 words

(1) Introduction and Summary

The IDE used is IntelliJ and uses JavaFX to create the GUI. MySQL workbench to use the database.

N.B. - some pictures cannot have inline comments due to file size // please see the source code

(2) Summary of Key Techniques used:

- Use of additional libraries
- Loading Image on to GUI
- GUI table
- GUI pie chart
- GUI line chart
- Overloading
- SQL (Structured Query Language) command to communicate with MySQL database
- Use of sentinels or flags
- Error Handling (try and catch)
- Alert warning
- Use of Conditional
- File i/o
- Inheritance
- Parsing
- Array
- Multi-dimensional Array
- ArrayList
- User-defined objects
- Objects as data records
- Simple-Selection (if-else)

- Get and Set Method
- Loops
- Nested Loops
- User-defined methods
- User-defined methods with parameters
- User-defined methods with appropriate returns values (objects or primitive)
- Linear Search
- Substring

(3) Structure of program

I have constructed my program structure in object-oriented programming (OOPs) way, which helps me to break the program into smaller problems that I can maintain and solve. The main part of OOPs includes **encapsulation**, **inheritance** and **polymorphism**. In total, I have constructed 14 classes. Out of these 14 classes, 7 classes correspond to each page of the application which is linked to FXML files with the same name (building blocks of my GUI). With this many class OOPs allow many methods to be reused in similar scenarios. Encapsulation is also used throughout to avoid access from outside class for security and reduce error with many variables.

(4) Graphical User Interface

(4.1) Pie Chart

Figure 1 - Piechart code (nutrientController)

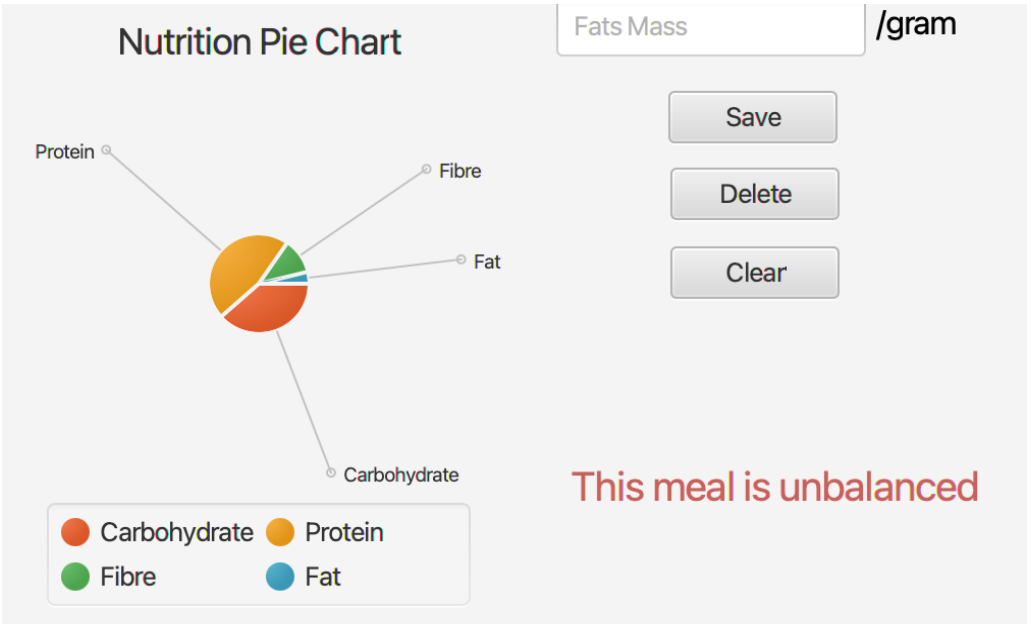
```
@FXML
private PieChart pieChart;
```

```
pieChart.getData().clear();
```

```
        //Parsing to Integer
        int tempCarb = Integer.parseInt(carb);
        int tempProtein = Integer.parseInt(protein);
        int tempFibre = Integer.parseInt(fibre);
        int tempFat = Integer.parseInt(fat);
        int temporaryMass = tempCarb + tempProtein + tempFibre + tempFat;//Calculating total Mass
        //Calculate mass percentage respected to the total mass
        int carbPercent = tempCarb/temporaryMass;
        int proteinPercent = tempProtein/temporaryMass;
        int fatPercent = tempFat/temporaryMass;
```

```
//Checking if meal is healthy and displaying message using condition from health website
if (((0.45<carbPercent)||(carbPercent<0.65))&&((0.2<proteinPercent)||(proteinPercent<0.35))&&
        (tempFibre>30)&&((0.2<fatPercent)||(fatPercent<0.35))){
    indicator.setText("This meal is balanced \n and healthy");
    indicator.setTextFill(Color.web( s: "#008450", v: 0.8));
}else{
    indicator.setText("This meal is unbalanced");
    indicator.setTextFill(Color.web( s: "#B81D13", v: 0.8));
}
//Setting the piechart with given data
PieChart.Data slice1 = new PieChart.Data( s: "Carbohydrate", tempCarb);
PieChart.Data slice2 = new PieChart.Data( s: "Protein", tempProtein);
PieChart.Data slice3 = new PieChart.Data( s: "Fibre", tempFibre);
PieChart.Data slice4 = new PieChart.Data( s: "Fat", tempFat);
pieChart.getData().add(slice1);
pieChart.getData().add(slice2);
pieChart.getData().add(slice3);
pieChart.getData().add(slice4);
```

Figure 2 - Pie Chart showing the input data in section(nutrientController)



Since my input and storage for numerical value is in string I parse it from string to integer to calculate the total mass or percentage of each. Through the use of the Pie Chart system from JavaFX I can define PieChart.Data objects for each nutrient mass as percentage in PieChart. Not only is data easier to display in terms of coding but also more visually attractive to table counterparts. Also, I use the mathematics condition to find out if the percentage of the meal is balanced, giving out appropriate advice responding to it.

(4.2) Line Graph

Figure 3 - Linehart (tempGraphController)

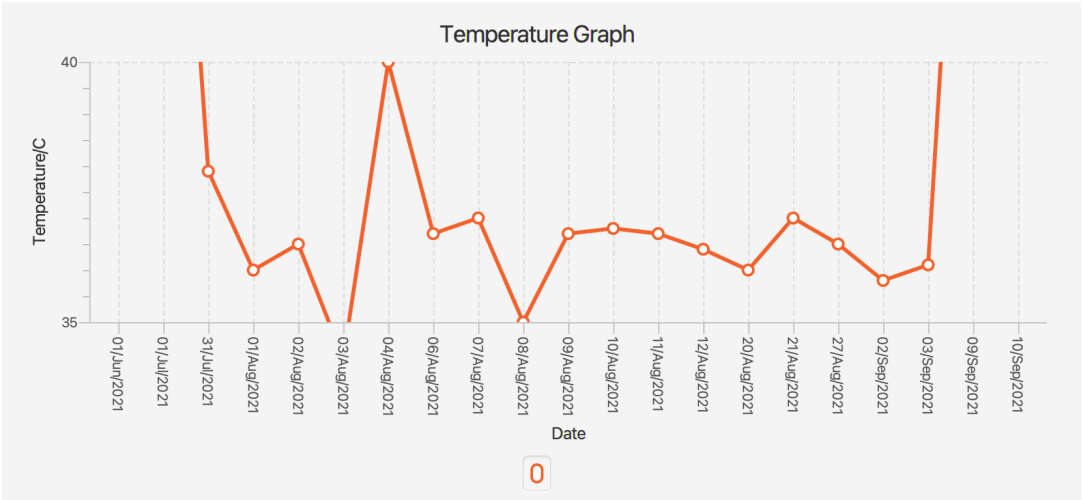


Figure 4 - Line Graph (tempGraphController)

```java
@FXML
private LineChart<?, ?> tempGraph; //Declaring LineChart
int account_id = getAccount();

@Override
public void initialize(URL url, ResourceBundle rb) { generate(); }

//Generate Temperature Graph
public void generate(){
    //Connect to Database
    DatabaseConnection connectNow = new DatabaseConnection();
    Connection connectDB = connectNow.getConnection();
    try {
        //Selecting data from database order by date
        preparedStatement = connectDB.prepareStatement( sql: "SELECT * FROM tracker WHERE user_id ="+account_id+" ORDER by dateCal");
        resultSet = preparedStatement.executeQuery();
        XYChart.Series temperature = new XYChart.Series();
        //Loop through result and set X axis to date and Y axis to temperature
        while(resultSet.next()) {
            temperature.getData().add(new XYChart.Data(convertDateToString(resultSet.getDate( columnLabel: "dateCal")), resultSet.getDouble( columnLabel: "temp")));
        }
        tempGraph.getData().addAll(temperature);
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}
//Method to convert Date to String
public static String convertDateToString(Date indate) {
    String dateString = null;
    SimpleDateFormat sdfr = new SimpleDateFormat( pattern: "dd/MMM/yyyy");
    try{
        dateString = sdfr.format(indate);
    }catch (Exception ex ){
        System.out.println(ex);
    }
    return dateString;
}
```

(4.3) CheckBox

Figure 5 - Checkbox (trackerController)

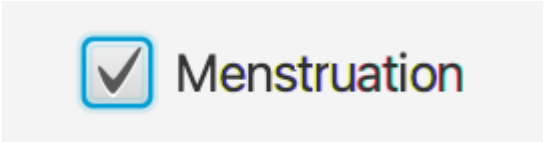
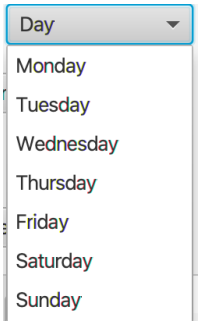


Figure 6 - Checkbox Algorithm if statement (trackerController)

```
String mens = "No"; //Declaring Variable for menstruation
if(menstruation.isSelected()){ mens = "Yes"; } //Check if box is tick
```

(4.4) ComboBox

Figure 7 - Dropdown ComboBox (plannerController)



(4.5) Loading Image

Figure 8 - Loading image from the folder (plannerController)

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    File lockFile = new File( pathname: "OMO/logo.png");
    Image lockImage = new Image(lockFile.toURI().toString());
    lockImageView.setImage(lockImage);
}
```

(5) Main Algorithm Explained

(5.1) Login & Register Functionality

Figure 9 - Entering data from the field to Database (registerController)

```
//Method to put registry data that entered from field into database
public void registerUser(){
    //Connect Datbase
    DatabaseConnection connectNow = new DatabaseConnection();
    Connection connectDB = connectNow.getConnection();
    //Getting Data from TextField
    String firstname = firstnameTextField.getText();
    String lastname = lastnameTextField.getText();
    String username = usernameTextField.getText();
    String password = setPasswordField.getText();
    //SQL commands to enter this data into new row on user_account table in database
    String insertFields = "INSERT INTO user_account(lastname, firstname, username, password) VALUES ('";
    String insertValues = firstname +"','"+ lastname +"','"+ username +"','"+ password +"')";
    String insertToRegister = insertFields + insertValues;
    try{
        //Execute Command
        Statement statement = connectDB.createStatement();
        statement.executeUpdate(insertToRegister);
        //Give Alert to show completion
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setHeaderText(null);
        alert.setContentText("User enter data successfully");
        alert.showAndWait();
    }catch (Exception e){
        e.printStackTrace();
        e.getCause();
    }
}
```

Get the text from each field, use MySQL command “INSERT” to enter value from each field into a database table of ‘user_account’ at the assigned column name, this database will be accessible with loginController too as they need to verify login content. An alert message will be displayed if registered successfully.

Figure 10 - Condition before submitting data (registerController)

```
@FXML
public void registerButtonOnAction(ActionEvent event) {
    //Check if any field is empty
    if (firstnameTextField.getText().isBlank() && lastnameTextField.getText().isBlank() &&
            confirmPasswordField.getText().isBlank() && setPasswordField.getText().isBlank() &&
            usernameTextField.getText().isBlank()) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setHeaderText(null);
        alert.setContentText("Please fill in the data!");
        alert.showAndWait();
    //Check if password is equal to confirm password
    }else if(!(setPasswordField.getText().equals(confirmPasswordField.getText()))) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setHeaderText(null);
        alert.setContentText("Password not match!");
        alert.showAndWait();
    //Check if username is already used, recalling method from DatabaseConnection
    }else if(!(CheckUsernameExists(usernameTextField.getText()))) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setHeaderText(null);
        alert.setContentText("Username already exist!");
        alert.showAndWait();
    }else{
        registerUser();
    }
}
```

Check condition from field to be sure that data is not null or already exists for username, if-else would help the organisation to see which condition is true or false, and can be easily modified or add more condition.

Figure 11 - Check Username Method For Registration (DatabaseConnection)

```
public static boolean CheckUsernameExists(String username)
{
    DatabaseConnection connectNow = new DatabaseConnection();
    Connection connectDB = connectNow.getConnection();
    boolean usernameExists = false;
    try
    {
        PreparedStatement st = connectDB.prepareStatement( sql: "select * from user_accounts order by username desc");
        ResultSet r1=st.executeQuery();
        String usernameCounter;
        if(r1.next())
        {
            usernameCounter =  r1.getString( columnLabel: "username");
            if(usernameCounter.equalsIgnoreCase(username))
            {
                System.out.println("It already exists");
                usernameExists = true;
            }
        }
    }
    catch (SQLException e)
    {
        System.out.println("SQL Exception: "+ e.toString());
    }
    return usernameExists;
}
```

Figure 12 - Login Function (loginController)

```
public void validateLogin() {
    DatabaseConnection connectNow = new DatabaseConnection();
    Connection connectDB = connectNow.getConnection();

    String username = usernameTextField.getText();
    String password = enterPasswordField.getText();
    String verifyLogin = "SELECT count(1) FROM user_account WHERE username = '" +
            username + "' AND password ='" + password + "'";
    try {
        Statement statement = connectDB.createStatement();
        ResultSet queryResult = statement.executeQuery(verifyLogin);
        while (queryResult.next()) {
            if (queryResult.getInt( columnIndex: 1) == 1) {
                try {
                    preparedStatement = connectDB.prepareStatement( sql: "SELECT * FROM user_account WHERE username = '" +
                            username + "'");
                    resultSet = preparedStatement.executeQuery();
                    while (resultSet.next()) {
                        giveUserID = resultSet.getInt( columnLabel: "account_id");
                    }
                } catch (SQLException throwables) {
                    throwables.printStackTrace();
                }
                String storeData = "INSERT INTO log(username, account_id) VALUES('"+username +"','"+ giveUserID +"')";
                try {
                    Statement statementOne = connectDB.createStatement();
                    statementOne.executeUpdate(storeData);
                } catch (Exception e) {
                    e.printStackTrace();
                    e.getCause();
                }

                new loadTab( fxmlFile: "lobby.fxml",  width: 1100,  height: 650);
                Stage stage = (Stage) loginButton.getScene().getWindow();
                stage.close();

            } else {
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setHeaderText(null);
                alert.setContentText("Invalid Login please try again!");
                alert.showAndWait();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
        e.getCause();
    }
}
```

First, it will gather the data from the two fields. Then at verifyLogin, I use MySQL commands “count(1)” to count the number of rows with the same username and password that enter from the field. Using a while loop to get to the result then get value from the database which will be 1 if the account is already registered, otherwise, it will be 0 which puts up an alert message.

If an account exists the program will select only the username and return the account_id then take that id and put it in a database table called ‘log’. This account_id will act as a key to go into the database when displaying data, to ensure that all data displayed will be from this person's database with this id.

Figure 13 - Login Field Validation (loginController)

```
@FXML
public void loginButtonOnAction() {
    //Check if the field is blank
    if (!usernameTextField.getText().isBlank() && !enterPasswordField.getText().isBlank()) {
        validateLogin();

    } else {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setHeaderText(null);
        alert.setContentText("Please fill in the data!");
        alert.showAndWait();
        //if field is blank it alert the user with appropriate message
    }
}
```

(5.2) Inheritance

Figure 14 - insideApp class hosting list of methods that are common in other class

```
public class insideApp{
    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;
    @FXML
    private Button homeButton;
    @FXML
    private Button close;

    //Close the tab with button
    public void closeOnAction() {
        Stage stage = (Stage) close.getScene().getWindow();
        stage.close();
    }

    //Go to the home page from other page
    public void homeOnAction() {
        new loadTab( fxmlFile: "lobby.fxml", width: 1100, height: 650);
        Stage stage = (Stage) homeButton.getScene().getWindow();
        stage.close();
    }

    //Method to check if it numeric
    public static boolean isNotNumber(String str){
        try{
            Double.parseDouble(str);
            return false;
        }catch(NumberFormatException e){
            return true;
        }
    }
```

Often applications will use a function such as closing or going back, to reduce redundancy it is good to create a class that maintains all the methods that can be recalled. It also makes maintenance easier as it requires only change here.

(5.3) Planners(Array) - Entering/Deleting/Loading

Figure 15 - Setting up Variable for Array and ComboBox (plannerController)

```
int account_id = getAccount();
private int DE = 0;
private int TE = 0;
private int TD = 0;
private int DD = 0;
@FXML
private GridPane matrix;
private final Label[][] label = new Label[7][14]; //Declaring size of the Label Array
private final Pane[][] pane = new Pane[7][14]; // Creating Pane with size of Array

//Setting up value for the ComboBox time and day
ObservableList<String> digTime = FXCollections.observableArrayList(
        ...es: "7:00", "8:00", "9:00","10:00","11:00","12:00","13:00","14:00","15:00","16:00",
        "17:00","18:00","19:00","20:00");
ObservableList<String> daily = FXCollections.observableArrayList(
        ...es: "Monday", "Tuesday", "Wednesday","Thursday","Friday","Saturday","Sunday");

@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    timeEnter.setItems(digTime);
    dayEnter.setItems(daily);
    timeDelete.setItems(digTime);
    dayDelete.setItems(daily);
    setMatrix();
    generateTaskList();
    generateLocationList();
    chooseTask();
    chooseLocation();
}
```

Figure 16 - Switch for the value that entered (plannerController)

```
private void timeToArray() {
    String x = timeEnter.getValue();
    switch (x) {
        case "7:00" -> TE = 0;
        case "8:00" -> TE = 1;
        case "9:00" -> TE = 2;
        case "10:00" -> TE = 3;
        case "11:00" -> TE = 4;
        case "12:00" -> TE = 5;
        case "13:00" -> TE = 6;
        case "14:00" -> TE = 7;
        case "15:00" -> TE = 8;
        case "16:00" -> TE = 9;
        case "17:00" -> TE = 10;
        case "18:00" -> TE = 11;
        case "19:00" -> TE = 12;
        case "20:00" -> TE = 13;
    }
}
```

```
private void timeToDelete() {
    String x = timeDelete.getValue();
    switch (x) {
        case "7:00" -> TD = 0;
        case "8:00" -> TD = 1;
        case "9:00" -> TD = 2;
        case "10:00" -> TD = 3;
        case "11:00" -> TD = 4;
        case "12:00" -> TD = 5;
        case "13:00" -> TD = 6;
        case "14:00" -> TD = 7;
        case "15:00" -> TD = 8;
        case "16:00" -> TD = 9;
        case "17:00" -> TD = 10;
        case "18:00" -> TD = 11;
        case "19:00" -> TD = 12;
        case "20:00" -> TD = 13;
    }
}
```

```
    private void dayToArray() {
        String y = dayEnter.getValue();
        switch (y) {
            case "Monday" -> DE = 0;
            case "Tuesday" -> DE = 1;
            case "Wednesday" -> DE = 2;
            case "Thursday" -> DE = 3;
            case "Friday" -> DE = 4;
            case "Saturday" -> DE = 5;
            case "Sunday" -> DE = 6;
        }
    }
```

```
    private void dayToDelete() {
        String y = dayDelete.getValue();
        switch (y) {
            case "Monday" -> DD = 0;
            case "Tuesday" -> DD = 1;
            case "Wednesday" -> DD = 2;
            case "Thursday" -> DD = 3;
            case "Friday" -> DD = 4;
            case "Saturday" -> DD = 5;
            case "Sunday" -> DD = 6;
        }
    }
```

Figure 17 - Creating Multi-Dimensional Array (plannerController)

```
private void setMatrix(){
    //Adding pane and text to the multidimensional array, to later store item
    for (int i = 0; i < label.length; i++) {
        for (int j = 0; j < label[i].length; j++) {
            label[i][j] = new Label();
            pane[i][j] = new Pane();
            matrix.add(pane[i][j], i, j);
            //Set default background color
            pane[i][j].setStyle("-fx-background-color: #F2C2C2;");
        }
    }
    try {
        //Connect Database
        DatabaseConnection connectNow = new DatabaseConnection();
        Connection connectDB = connectNow.getConnection();
        //Select data from planner table
        preparedStatement = connectDB.prepareStatement( sql: "SELECT * FROM planner WHERE user_id ="+account_id);
        resultSet = preparedStatement.executeQuery();
        //Loop through result
        while (resultSet.next()) {
            int theDay = resultSet.getInt( columnLabel: "day"); // Position of Horizontal Array
            int theTime = resultSet.getInt( columnLabel: "time"); // Position of Vertical Array
            //Using the position key set the text there
            label[theDay][theTime].setText(resultSet.getString( columnLabel: "task") +"\n" +
                    resultSet.getString( columnLabel: "location") +"\n"+
                    resultSet.getString( columnLabel: "addition"));
            //Display on the matrix grid
            matrix.add(label[theDay][theTime], theDay, theTime);
            //Look up the color column in database and get the hex id of it then make the position that color
            String colorHex = resultSet.getString( columnLabel: "color").substring(2,8);
            pane[theDay][theTime].setStyle("-fx-background-color: #" +colorHex+ ";");
            GridPane.setHalignment(label[theDay][theTime], HPos.CENTER);
        }
    } catch (SQLException ex) {
        Logger.getLogger(plannerController.class.getName()).log(Level.SEVERE,  msg: null, ex);
    }
}
```

The planner table is a multidimensional array with time and day represented by integer values corresponding to array position. Initially, I made a ComboBox with an observable array list for each time. Then using a switch to correspond each data value with an array row and column. Switch is used here as it is able to jump to the corresponding condition, as I have only a fixed set of values, improving performance and it is more comprehensible than if-else statements.

Firstly, I set up my multi-dimensional array to fit in with the grid with label[][] for text and pane[][] for setting backdrop colour. Using nested for loop, I am able to go through each row and column of the grid with i indicating row and j indicating column, this is useful to indicate specific grid box. Then (for) loop through results from the “planner” database table, to load in the day and time which is indicated by a number that represents its grid position. If both time and day array numbers are the same with the database it will create a label at the spot it found the day and time (label[i][j]). Afterwards, I set that grid box label with the text stored in the same row from the database. Using this system of storing and retrieving is efficient as each grid has designated coordinates and is easily manipulated. This is seen when I add a function to change the colour of the grid cell.

Figure 18 - Deleting value in the Array (plannerController)

```
@FXML //On action button to delete the select array
private void deleteArray() {
    //Check if user entered both value for position array
    if(timeDelete.getValue() != null && dayDelete.getValue() != null) {
        //Recall Method that convert ComboBox value to that of position array(INTEGER)
        dayToDelete();
        timeToDelete();
        try {
            //Connect Database
            DatabaseConnection connectNow = new DatabaseConnection();
            Connection connectDB = connectNow.getConnection();
            //Selected deleting only place with the position array input
            preparedStatement = connectDB.prepareStatement( sql: "DELETE FROM planner WHERE day = " + DD + " AND time = " + TD+ " AND user_id = " + account_id);
            preparedStatement.execute();
            new loadTab( fxmlFile: "planner.fxml",  width: 1100,  height: 650); //Refresh Page
            Stage stage = (Stage) deleteButton.getScene().getWindow();
            stage.close();
        } catch (SQLException ex) {
            Logger.getLogger(plannerController.class.getName()).log(Level.SEVERE,  msg: null, ex);
        }
    }
}
```

Similarly, to delete data, it will get data from comboBox translated to array number and check for it in the database.

Figure 19 - Setting up Variable for Array and ComboBox (plannerController)

```java
@FXML
private void save() {
    DatabaseConnection connectNow = new DatabaseConnection();
    Connection connectDB = connectNow.getConnection();
    String task = taskField.getText();
    String location = locationField.getText();
    String addition = addField.getText();

    if (timeEnter.getValue() == null && dayEnter.getValue() == null
            || task.isEmpty() || location.isEmpty() || addition.isEmpty()) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setHeaderText(null);
        alert.setContentText("Please Fill in the Data!");
        alert.showAndWait();
    } else {
        timeToArray();
        dayToArray();
        int day = DE;
        int time = TE;
        boolean FLAG = false;
        try {
            preparedStatement = connectDB.prepareStatement( sql: "SELECT * FROM planner WHERE user_id ="+account_id);
            resultSet = preparedStatement.executeQuery();
            while (resultSet.next() && !FLAG) {
                int theDay = resultSet.getInt( columnLabel: "day"); // 0
                int theTime = resultSet.getInt( columnLabel: "time"); // 1
                if ((DE == theDay) && (TE == theTime)) {
                    FLAG = true;
                }
            }
            if (FLAG) {
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setHeaderText(null);
                alert.setContentText("Date and Time already used!");
                alert.showAndWait();
            } else {
                String insertFields = "INSERT INTO planner(task, location, addition, day, time, user_id) VALUES ('";
                String insertValues = task + "','" + location + "','" + addition + "','" + day + "','" + time + "','"+ account_id + "')";
                String insertToPlanner = insertFields + insertValues;
                try {
                    Statement statement = connectDB.createStatement();
                    statement.executeUpdate(insertToPlanner);
                    new loadTab( fxmlFile: "planner.fxml", width: 1100, height: 650);
                    Stage stage = (Stage) enterButton.getScene().getWindow();
                    stage.close();
                } catch (Exception e) {
                    e.printStackTrace();
                    e.getCause();
                }
            }
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    }
}
```

Saving data to the array would use the getText() method to get the string from textField. Then it will receive the array position value from ComboBox. Next, it will load the database with that person's login ID to avoid collateral data. It will loop through all results in the database getting the day and time array position checking if they both are the same with ComboBox. FLAG will stop the code if it were to find data at the

same slot and give out an appropriate error message. This is efficient as otherwise, it will go through all the lists, which is slow. Lastly, using SQL command to enter into a database at this array slot.

Figure 20 - Initialising ListView

```
    @FXML
    private ListView<String> commonTaskList;
    @FXML
    private ListView<String> commonLocationList;
```

Figure 21 - Creating new ArrayList to store unlimited item

```
private void generateTaskList(){
    ArrayList<String> commonTask = new ArrayList<>();
    commonTask.add("Sleep");
    commonTask.add("Work");
    commonTask.add("Breakfast");
    commonTask.add("Lunch");
    commonTask.add("Dinner");
    commonTask.add("Exercise");
    commonTask.add("Leisure");
    commonTask.add("Shopping");

    for (String useTask : commonTask){
        commonTaskList.getItems().add(useTask);
    }
}
```

Figure 22 - Allow mouse action event, to display the chosen word from ArrayList to field

```
private void chooseTask(){
    commonTaskList.setOnMouseClicked(event ->{
        String item = commonTaskList.getSelectionModel().getSelectedItem();
        taskField.setText(item);
    });
}
```

ArrayList is implemented here as it has unlimited size perfect for adding more tasks for clients, which increases its feasibility to maintenance or further development.

(5.4) Lobby(ProgressBar and Calculation) - Parsing

Figure 23 - Method of getting the date from the date picker and using it to calculate progress in labour and showcasing it(lobbyController)

```
public void getDateNow() {
    SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd");
    String datenow = sdf.format(new Date());
    LocalDate myDate = myDatePicker.getValue();
    String assign = myDate.toString();
    dayLabel.setText(pregCount(assign, datenow));
    progress = Float.parseFloat(pregCount(assign, datenow))/280;
    if (progress<0.9){
        myProgressBar.setProgress(progress);
        progressLabel.setText(((int)Math.round(progress * 100)) + "%");
    }
}
```

Figure 24 - User-Defined method of calculation with return value(lobbyController)

```
private String pregCount(String past, String present){
    float pastYear = Float.parseFloat(past.substring(0,4));
    float pastMonth = Float.parseFloat(past.substring(5,7));
    float pastDay = Float.parseFloat(past.substring(8,10));
    float presentYear = Float.parseFloat(present.substring(0,4));
    float presentMonth = Float.parseFloat(present.substring(5,7));
    float presentDay = Float.parseFloat(present.substring(8,10));
    float diffYear = presentYear-pastYear;
    float diffMonth = (presentMonth-pastMonth)+(diffYear*12);
    float diffDay = (presentDay-pastDay)+(diffMonth*30);
    return String.valueOf((int)diffDay);
}
```

The date is entered and a today date(from importing SimpleDateFormat) is converted into string to share the same type thus compatible. Then I recall the pregCount() method with parameters of these two string dates, this method finds the date difference between the two. Since a string date is in format of “yyyy-MM-dd” we need to separate them for calculation. Using substring() to dissect, then converted to float as it is able to hold decimal points and take less space than Double data type, before converting to integer to remove decimal places. Lastly, convert to string to display as text.

(5.5) Tables - Entering/Deleting/Displaying Data

Figure 25 - Defining class necessary for table

```
    @FXML
    private TableView<modelNutTable> table;
    @FXML
    private TableColumn<modelNutTable, String> colDate;
    @FXML
    private TableColumn<modelNutTable, String> colFood;
    @FXML
    private TableColumn<modelNutTable, String> colCalories;
    @FXML
    private TableColumn<modelNutTable, String> colMass;
    @FXML
    private TableColumn<modelNutTable, String> colCarb;
    @FXML
    private TableColumn<modelNutTable, String> colProtein;
    @FXML
    private TableColumn<modelNutTable, String> colFibre;
    @FXML
    private TableColumn<modelNutTable, String> colFat;
    @FXML
    private PieChart pieChart;
    modelNutTable someNut = null;
    modelNutStorage someNutStorage = null;
    int account_id = getAccount();
    ObservableList<modelNutTable> nutList = FXCollections.observableArrayList();
```

Figure 26 - Getting Data From Database

```
private void refreshTable() {
    try {
        nutList.clear();
        DatabaseConnection connectNow = new DatabaseConnection();
        Connection connectDB = connectNow.getConnection();
        SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd");
        String today = sdf.format(new Date());
        preparedStatement = connectDB.prepareStatement( sql: "SELECT * FROM nutrient WHERE date = '"+today+"' AND user_id = "+account_id);
        resultSet = preparedStatement.executeQuery();
        while (resultSet.next()){
            nutList.add(new modelNutTable(
                    resultSet.getDate( columnLabel: "date"),
                    resultSet.getString( columnLabel: "food"),
                    resultSet.getDouble( columnLabel: "calories"),
                    resultSet.getDouble( columnLabel: "mass"),
                    resultSet.getDouble( columnLabel: "carb"),
                    resultSet.getDouble( columnLabel: "protein"),
                    resultSet.getDouble( columnLabel: "fibre"),
                    resultSet.getDouble( columnLabel: "fat")));
            table.setItems(nutList);
        }
    } catch (SQLException ex) {
        Logger.getLogger(trackerController.class.getName()).log(Level.SEVERE,  msg: null, ex);
    }
}
```

Figure 27 - modelNutTable class

```
public class modelNutTable {

    private Date date;
    private final String food;
    private final Double calories;
    private final Double mass;
    private final Double carb;
    private final Double protein;
    private final Double fibre;
    private final Double fat;
    public modelNutTable(Date date, String food, Double calories, Double mass, Double carb, Double protein, Double fibre, Double fat) {
        this.date = date;
        this.food = food;
        this.calories = calories;
        this.mass = mass;
        this.carb = carb;
        this.protein = protein;
        this.fibre = fibre;
        this.fat = fat;
    }
    public Date getDate() { return date; }
    public void setDate(Date date) { this.date = date; }
    public String getFood() { return food; }
    public Double getCalories() { return calories; }
    public Double getCarb() { return carb; }
    public Double getProtein() { return protein; }
    public Double getFibre() { return fibre; }
    public Double getFat() { return fat; }
    public Double getMass() { return mass; }
}
```

Figure 28 - Displaying the data from database into table

```
private void load(){
    DatabaseConnection connectNow = new DatabaseConnection();
    connectNow.getConnection();
    refreshTable();

    colDate.setCellValueFactory(new PropertyValueFactory<>( s: "date"));
    colFood.setCellValueFactory(new PropertyValueFactory<>( s: "food"));
    colCalories.setCellValueFactory(new PropertyValueFactory<>( s: "calories"));
    colMass.setCellValueFactory(new PropertyValueFactory<>( s: "mass"));
    colCarb.setCellValueFactory(new PropertyValueFactory<>( s: "carb"));
    colProtein.setCellValueFactory(new PropertyValueFactory<>( s: "protein"));
    colFibre.setCellValueFactory(new PropertyValueFactory<>( s: "fibre"));
    colFat.setCellValueFactory(new PropertyValueFactory<>( s: "fat"));
    table.setItems(nutList);
}
```

For the table, I first defined all components on FXML. Then connect to the database, also getting the today date which needs to be translated into String. Next using SQL command I open a database showing the data row with today's date and ID that user logged in with. Then I use while loop through all the rows and add data to the list. Here I recall a class modelNutTable, a class that creates constructor containing all my attributes and specific getter/setter (getter to get value, setter for date is to update the value) for all of them, this so the constructor can get data from all cell of that database row and inserts it onto the graph. Then it will be set on to table table.setItems(nutList). This method is great as if I were to add a new component for input it adds a new column on the database and adds one more on the constructor, easy for maintenance. nutList.clear() will clear all data on the table before inserting more data, to avoid data collision.

Figure 29 - Adding Data to the table

```
    private void save() {
        pieChart.getData().clear();
        DatabaseConnection connectNow = new DatabaseConnection();
        Connection connectDB = connectNow.getConnection();
        SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd");
        String datenow = sdf.format(new Date());
        String food = foodField.getText();
        String calories = calField.getText();
        String carb = carbField.getText();
        String protein = proteinField.getText();
        String fibre = fibreField.getText();
        String fat = fatField.getText();

        if (food.isEmpty() || calories.isEmpty() || carb.isEmpty()|| protein.isEmpty()|| fibre.isEmpty()|| fat.isEmpty()) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setHeaderText(null);
            alert.setContentText("Please Fill All DATA");
            alert.showAndWait();
        } else {
//            clean();
            int tempCarb = Integer.parseInt(carb);
            int tempProtein = Integer.parseInt(protein);
            int tempFibre = Integer.parseInt(fibre);
            int tempFat = Integer.parseInt(fat);
            int temporaryMass = tempCarb + tempProtein + tempFibre + tempFat;
            int carbPercent = tempCarb/temporaryMass;
            int proteinPercent = tempProtein/temporaryMass;
            int fatPercent = tempFat/temporaryMass;
            String mass = String.valueOf(temporaryMass);

            String insertFields = "INSERT INTO nutrient(date, food, calories, mass, carb, protein, fibre, fat, user_id) VALUES ('";
            String insertValues = datenow +"','"+ food +"','"+ calories +"','"+ mass +"','"+carb+"','"+protein+"','"+
                    fibre+"','"+fat+"','"+account_id+"')";
            String insertToNutrient = insertFields + insertValues;
            //Folding Piechart Algorithm Below
            ...
            try{
                Statement statement = connectDB.createStatement();
                statement.executeUpdate(insertToNutrient);
                refreshTable();
            }catch (Exception e){
                e.printStackTrace();
                e.getCause();
            }
            clean();
        }
    }
```

Figure 30 - Deleting Data of the table

```
private void deleteNutCell() {
    try {
        someNut = table.getSelectionModel().getSelectedItem();
        DatabaseConnection connectNow = new DatabaseConnection();
        Connection connectDB = connectNow.getConnection();
        preparedStatement = connectDB.prepareStatement( sql: "DELETE FROM nutrient WHERE fat = "+someNut.getFat()+
                " AND food = "+someNut.getFood()+ " AND user_id = " + account_id);
        preparedStatement.execute();

        refreshTable();
    } catch (SQLException ex) {
        Logger.getLogger(trackerController.class.getName()).log(Level.SEVERE,  msg: null, ex);
    }
}
```

(5.6) Other Unexplained Complexity (not explained below due to word count)

Figure 31 - Connect to database

```
public class DatabaseConnection {
    public Connection databaseLink;

    public Connection getConnection() {
        String databaseName = "demo_db";
        String databaseUser = "root";
        String databasePassword = [redacted]
        String url = "jdbc:mysql://localhost/" + databaseName;

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            databaseLink = DriverManager.getConnection(url, databaseUser, databasePassword);
        } catch (Exception e) {
            e.printStackTrace();
            e.getCause();
        }
        return databaseLink;
    }
```

Figure 32 - Rounding Decimal Place using Math.round

```
Double tempStore= Double.parseDouble(tempField.getText());
tempStore = Math.round(tempStore*10.0)/10.0;
String temp = String.valueOf(tempStore);
```

Figure 33 - Checking if usernames exists

```
public static boolean CheckUsernameExists(String username)
{
    DatabaseConnection connectNow = new DatabaseConnection();
    Connection connectDB = connectNow.getConnection();
    boolean usernameExists = false;
    try
    {
        PreparedStatement st = connectDB.prepareStatement( sql: "select * from user_accounts order by username desc");
        ResultSet r1=st.executeQuery();
        String usernameCounter;
        if(r1.next())
        {
            usernameCounter =  r1.getString( columnLabel: "username");
            if(usernameCounter.equalsIgnoreCase(username))
            {
                System.out.println("It already exists");
                usernameExists = true;
            }
        }
    }
    catch (SQLException e)
    {
        System.out.println("SQL Exception: "+ e.toString());
    }
    return usernameExists;
}
```

Figure 34 - getAccount() method relate to Figure Y

```
public int getAccount(){
    int account_id = 0;
    DatabaseConnection connectNow = new DatabaseConnection();
    Connection connectDB = connectNow.getConnection();
    try {
        preparedStatement = connectDB.prepareStatement( sql: "SELECT * FROM log ORDER BY log_id DESC LIMIT 1");
        resultSet = preparedStatement.executeQuery();
        while (resultSet.next()) {
            account_id = resultSet.getInt( columnLabel: "account_id");
        }
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
    System.out.println(account_id);
    return account_id;
}
```

Figure 35 - isNotNumber() boolean Method

```
public static boolean isNotNumber(String str){
    try{
        Double.parseDouble(str);
        return false;
    }catch(NumberFormatException e){
        return true;
    }
}
```

```
} else if(isNotNumber(weightField.getText()) || isNotNumber(tempField.getText())){
    Alert alertEmpty = new Alert(Alert.AlertType.ERROR);
    alertEmpty.setHeaderText(null);
    alertEmpty.setContentText("Data is not numeric!");
    alertEmpty.showAndWait();
}
```

Figure 36 - Loading new tab method increase visibility

```
public class loadTab {
    public loadTab(String fxmlFile, int width, int height){
        try{
            Parent root = FXMLLoader.load(getClass().getResource(fxmlFile));
            Stage Stage = new Stage();
            Stage.initStyle(StageStyle.UNDECORATED);
            Stage.setScene(new Scene(root, width, height));
            Stage.show();
        }catch(Exception e){
            e.printStackTrace();
            e.getCause();
        }
    }
}
```

```
public void calendarOnAction(){
    new loadTab( fxmlFile: "tracker.fxml",  width: 1100,  height: 650);
    Stage stage = (Stage) trackerButton.getScene().getWindow();
    stage.close();
}
public void nutrientOnAction(){
    new loadTab( fxmlFile: "nutrient.fxml", width: 1100, height: 650);
    Stage stage = (Stage) nutrientButton.getScene().getWindow();
    stage.close();
}
public void plannerOnAction(){
    new loadTab( fxmlFile: "planner.fxml", width: 1100, height: 650);
    Stage stage = (Stage) plannerButton.getScene().getWindow();
    stage.close();
}
```

Works Cited

1BestCsharp blog. “Java Project Tutorial - Make Login and Register Form Step by Step Using NetBeans and MySQL Database.” *Www.youtube.com*, www.youtube.com/watch?v=3vauM7axnRs&list=PLmFSPS0yJO4C_oRcXWBazIB9hxpJ1I2HL&index=5. Accessed 12 Feb. 2022.

All in One. “How to Create Time Table System in Java 2018.” *Www.youtube.com*, www.youtube.com/watch?v=xMMrfEkoAGA&list=PLmFSPS0yJO4C_oRcXWBazIB9hxpJ1I2HL&index=13. Accessed 12 Feb. 2022.

BoostMyTool. “Setup IntelliJ IDEA (2021) for JavaFX & SceneBuilder and Create Your First JavaFX Application.” *Www.youtube.com*, www.youtube.com/watch?v=ZfaPMLdgJxQ. Accessed 12 Feb. 2022.

Bro Code. “Java Open a New GUI Window .” *Www.youtube.com*, www.youtube.com/watch?v=HgkBvwgciB4&list=PLmFSPS0yJO4C_oRcXWBazIB9hxpJ1I2HL&index=19&t=432s. Accessed 12 Feb. 2022.

Code Amir. “JavaFX Scene Builder Tutorial 40 - Load and Adding Data on TableView from MySQL Database.” *Www.youtube.com*, www.youtube.com/watch?v=tw_NXq08NUE. Accessed 12 Feb. 2022.

Genuine Coder. “JavaFX TreeTableView Tutorial #3 : Editing Cells.” *Www.youtube.com*, www.youtube.com/watch?v=BNvVSU9nHDY&list=PLmFSPS0yJO4C_oRcXWBazIB9hxpJ1I2HL&index=26. Accessed 12 Feb. 2022.

Java Code Geeks. “Java Date and Calendar Tutorial.” *Www.youtube.com*, www.youtube.com/watch?v=il7eVsDPFoA&list=PLmFSPS0yJO4C_oRcXWBazIB9hxpJ1I2HL&index=5. Accessed 12 Feb. 2022.

Java Coding Community - Programming Tutorials. “JavaFX Login Form Tutorial Using Scene Builder | JavaFX and Scene Builder Tutorial | 2020 Version.” *Www.youtube.com*, www.youtube.com/watch?v=HBBtlwGpBek. Accessed 13 Feb. 2022.

KeepToo. “Java Swing : School Management System Inspiration.” *Www.youtube.com*, www.youtube.com/watch?v=z1_tvNY1dvg&list=PLmFSPS0yJO4C_oRcXWBazIB9hxpJ1I2HL&index=28. Accessed 12 Feb. 2022.

tookootek. “JavaFX Tutorial | Login and Register Screen GUI in IntelliJ & Scene Builder with MySQL - Part 1|2.” *Www.youtube.com*, www.youtube.com/watch?v=DH3dWzmkT5Y&t=148s. Accessed 12 Feb. 2022.

---. “JavaFX Tutorial | Search Bar and TableView Filter Result as You Search.” *Www.youtube.com*, www.youtube.com/watch?v=2M0L6w3tMOY&list=PLmFSPS0yJO4C_oRcXWBazIB9hxpJ1I2HL&index=15. Accessed 12 Feb. 2022.

Vasudev, Rakshith. “Build Your First OOP Application in Java with Example - Building a School Management System.” *Www.youtube.com*, www.youtube.com/watch?v=e0X00EoFQbE&list=PLmFSPS0yJO4C_oRcXWBazIB9hxpJ1I2HL&index=21&t=640s. Accessed 12 Feb. 2022.