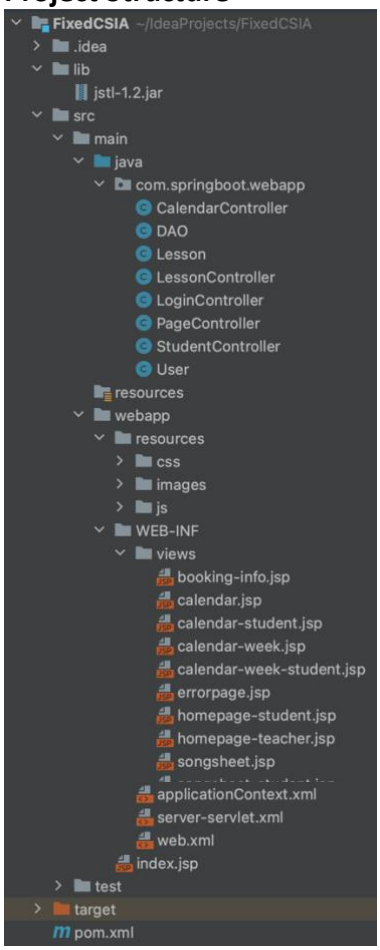
Project Structure



My program uses java because my client has an android phone. SpringMVC framework is used to create a website for students. Spring MVC is useful as it separates the model, view, and controller components, making the application easier to develop, and it can handle multithreading for multiple users.

My program uses object-oriented programming for modular development, making it easier to design. It is also flexible through inheritance and polymorphism. Due to the program being quite small, the drawback of OOP's slowness is negligible. There is a page controller, and smaller controllers to handle separate requests from forms. There is only one DAO, albeit the methods are grouped using regions, as data can be accessed from one place and methods can be accessed from one DAO.

JSP is used for the program frontend. CSS and JS links to these JSPs to improve non-functional aspects of the program. The GUI provides abstraction by only displaying the necessary details.

Classes follow naming conventions (Code Conventions).

Libraries:

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.LocalTime;
import java.util.Date;
import java.util.List;
import java.util.ArrayList;
import java.util.Locale;
import java.sql.*;
import java.time.format.DateTimeFormatter;
```

Controller and Request Mapping is used to easily access and request pages. This also allows form detail submission to easily access a method using action in JSP and Request Mapping in java. ModelAndView is used for navigating to different pages.

HTTP is used to create a session in which the current user can be stored. It can set attributes through HTTP request for JSP access.

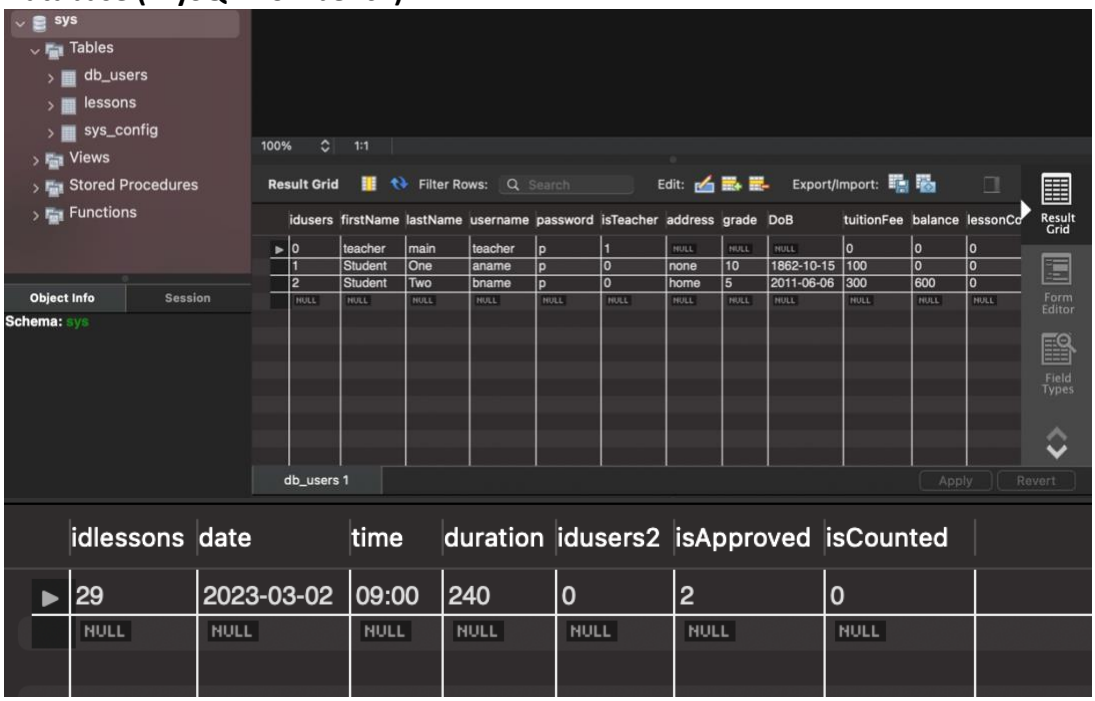
LocalDate and Time are used so lessons and their timings can be compared to the current time. DateFormat is used to convert the date string into a Date object.

```
<script src="resources/js/jquery-3.6.3.js"></script>
```

JQuery is used for the calendar as it can handle date and time.

The jstl library is also used for html so that looping through the array list of users and lessons is possible (Oracle).

Database (MySQLWorkbench):



A database is used so that the details of the users and lessons can be stored, read, updated, and deleted by the program easily. IsApproved refers to ‘blocked’ times when it is ‘2’. ‘0’ and ‘1’ is used for normal lessons which are approved or unapproved. IsCounted determines if a lesson has already passed and added to the lesson count so that it is not retrieved again. MySQL allows multithreading.

Code Explanations:

User class:

```
public class User {
    private int userID;
    private String firstName;
    private String lastName;
    private String username;
    private String password;
    private int isTeacher;
    private String address;
    private String studentGrade;
    private String studentDoB;
    private int tuitionFee;
    private float balance;
    private int lessonCount;
    private int lessonDuration;
    private String emailAddress;
    private String contactNumber;
```

```java
public User(String firstName, String lastName, String address, String contactNumber, int lessonDuration) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.address = address;
    this.contactNumber = contactNumber;
    this.lessonDuration = lessonDuration;
}

public User(int userID, float balance, int tuitionFee, int lessonCount){
    this.userID = userID;
    this.balance = balance;
    this.tuitionFee = tuitionFee;
    this.lessonCount = lessonCount;
}
```

The attributes and some methods. Both methods instantiate a user with parameters, a form of encapsulation. This is polymorphism and overloading as they are constructors with the same name but different parameters.

```java
public int getUserID() { return userID; }
public void setUserID(int userID) { this.userID = userID; }

public String getFirstName() { return firstName; }
public void setFirstName(String firstName) { this.firstName = firstName; }

public String getLastName() { return lastName; }
public void setLastName(String lastName) { this.lastName = lastName; }

public String getUsername() { return username; }
public void setUsername(String username) { this.username = username; }

public String getPassword() { return password; }
public void setPassword(String password) { this.password = password; }

public int getIsTeacher() { return isTeacher; }
public void setIsTeacher(int isTeacher) { this.isTeacher = isTeacher; }

public String getAddress() { return address; }
public void setAddress(String address) { this.address = address; }

public String getStudentGrade() { return studentGrade; }
public void setStudentGrade(String studentGrade) { this.studentGrade = studentGrade; }

public String getStudentDoB() { return studentDoB; }
public void setStudentDoB(String studentDoB) { this.studentDoB = studentDoB; }

public int getTuitionFee() { return tuitionFee; }
public void setTuitionFee(int tuitionFee) { this.tuitionFee = tuitionFee; }

public int getBalance() { return balance; }
public void setBalance(int balance) { this.balance = balance; }

public int getLessonCount() { return lessonCount; }
public void setLessonCount(int lessonCount) { this.lessonCount = lessonCount; }

public int getLessonDuration() { return lessonDuration; }
public void setLessonDuration(int lessonDuration) { this.lessonDuration = lessonDuration; }

public String getEmailAddress() { return emailAddress; }
public void setEmailAddress(String emailAddress) { this.emailAddress = emailAddress; }

public String getContactNumber() { return contactNumber; }
public void setContactNumber(String contactNumber) { this.contactNumber = contactNumber; }
```

Accessor and mutator methods.

Lesson class:

```java
public class lesson {
    private int lessonID;
    private String lessonDate;
    private String lessonTime;
    private int lDuration;
    private int uID;
    private int isApproved;
    private String userFirstName;
    private String userLastName;
    private String userAddress;
    private String userContact;
    private int userDuration;
    private int isCounted;
```

These are attributes for a lesson (including user attributes, accessed in the Controllers).

```java
public Lesson(int lessonID, String date, String time, int duration, int userID) {
    this.lessonID = lessonID;
    this.lessonDate = date;
    this.lessonTime = time;
    this.lDuration = duration;
    this.uID = userID;
}

public Lesson(int lessonID, String lessonDate, String lessonTime){
    this.lessonID = lessonID;
    this.lessonDate = lessonDate;
    this.lessonTime = lessonTime;
}
```

This provides encapsulation by only sending the data in the parameters. This prevents users from accessing unnecessary data and is more efficient. It is polymorphism— they have the same method name but different parameters. The modifier is public so these methods can be accessed.

```java
public int getLessonID() { return lessonID; }
public void setLessonID(int lessonID) { this.lessonID = lessonID; }

public String getLessonDate() { return lessonDate; }
public void setLessonDate(String lessonDate) { this.lessonDate = lessonDate; }

public String getLessonTime() { return lessonTime; }
public void setLessonTime(String lessonTime) { this.lessonTime = lessonTime; }

public int getlDuration() { return lDuration; }
public void setlDuration(int lDuration) { this.lDuration = lDuration; }

public int getuID() { return uID; }
public void setuID(int uID) { this.uID = uID; }

public int getIsApproved() { return isApproved; }
public void setIsApproved(int isApproved) { this.isApproved = isApproved; }

public String getUserFirstName() { return userFirstName; }
public void setUserFirstName(String userFirstName) { this.userFirstName = userFirstName; }

public String getUserLastName() { return userLastName; }
public void setUserLastName(String userLastName) { this.userLastName = userLastName; }

public String getUserAddress() { return userAddress; }
public void setUserAddress(String userAddress) { this.userAddress = userAddress; }

public String getUserContact() { return userContact; }
public void setUserContact(String userContact) { this.userContact = userContact; }

public int getUserDuration() { return userDuration; }
public void setUserDuration(int userDuration) { this.userDuration = userDuration; }
```

Accessor and mutator methods.

Login Controller:

```
@RequestMapping(value = "/login", method = RequestMethod.POST)
public ModelAndView display(HttpServletRequest request, HttpServletResponse response, Model m) throws ServletException, IOException, InterruptedException,
        SQLException, ClassNotFoundException, NoSuchMethodException, ParseException {
    try{ // due to the number of exceptions
        String username = request.getParameter( name: "username");
        String password = request.getParameter( name: "password");
        int iduser = DAO.loginFunction(username, password); // returns userID to show user exists and for use in order to set session later
        if (iduser != -1){
            // region Lesson Counter + Balancer
            ArrayList<User> userArrayList = DAO.doGetUserList(); // a list of all existing students
            for (int i = 0; i < (userArrayList != null ? userArrayList.size() : 0); i++){ // iterate through each user
                int userID = userArrayList.get(i).getUserID(); // get the user ID of the current user
                ArrayList<Lesson> userLessonsNotCounted = DAO.findUserIDLessonsNotCounted(userID); // find all the user's lessons which have not been counted
                for (int j = 0; j < (userLessonsNotCounted != null ? userLessonsNotCounted.size() : 0); j++){ // for each of these user's lessons
                    String date = userLessonsNotCounted.get(j).getLessonDate();
                    String time = userLessonsNotCounted.get(j).getLessonTime();
                    String datetime = date + " " + time; // combine date and time with " "
                    DateTimeFormatter formatter = DateTimeFormatter.ofPattern( pattern: "yyyy-MM-dd' 'HH:mm", Locale.getDefault());
                    LocalDateTime lessonDateTime = LocalDateTime.parse(datetime, formatter); // parse into LocalDateTime
                    LocalDateTime now = LocalDateTime.now(); // get the time as of now
                    if (now.isAfter(lessonDateTime)) { // if the lesson has already started
                        int lessonID = userLessonsNotCounted.get(j).getLessonID(); // get the lesson id
                        int isCounted = 1;
                        if (DAO.editIsCounted(isCounted, lessonID) != 1){ // edit is counted to be 1 so it is not retrieved again
                            return PageController.errorpage(request); // if not updated, send error page
                        }
                        int lessonCount = DAO.getLessonCount(userID); // get lesson count of the user
                        lessonCount = lessonCount + 1; // add 1 to user lesson count
                       if (DAO.editLessonCount(lessonCount, userID) != 1){ //update new lesson count
                           return PageController.errorpage(request); // if fails
                       }
                    }
                }
                int lessonCount = DAO.getLessonCount(userID); // get the updated lesson count of current user
                while (lessonCount >= 1){ // while lesson count is above 1, add 25% of tuition fee to the balance and minus 1 from lesson count
                    float tuitionFee = (float) DAO.getTuitionFee(userID);
                    float balance = DAO.getBalance(userID);
                    balance = (balance + (tuitionFee/4)); // add only 25% of monthly fee per lesson
                    if (DAO.editBalance(userID, balance) != 1){ // update balance
                        return PageController.errorpage(request); // if failure, errorpage
                    }
                    lessonCount = lessonCount - 1; // reduce until lesson count is 0
                }
                if (DAO.editLessonCount(lessonCount, userID) != 1){ // update final lesson count (should be 0)
                    return PageController.errorpage(request); // if fails, errorpage
                }
            }
            //endregion
            User u = DAO.getSessionUser(iduser); // try catch block in the method
            HttpSession session = request.getSession();
            session.setAttribute( name: "sessionUser", u); // set user details in session for retrieval and also validating user later on
            assert u != null;
            if (u.getIsTeacher() == 1) {
                return PageController.homepageT(request);
            } else if (u.getIsTeacher() == 0) {
                return PageController.homepageStudent(request);
            }
        } else {
            String msg = "username of password is wrong";
            m.addAttribute( attributeName: "error", msg);
            request.setAttribute( name: "errorMessage",  o: "this user does not exist");
            request.getRequestDispatcher( path: "/index.jsp").forward(request, response); // reload page instead of directing to error page
            Thread.sleep( millis: 100);
            return null;
        }
    } catch(Exception e){
        System.out.println("Error: " + e.getMessage());
        e.printStackTrace();
        return new ModelAndView( viewName: "errorpage"); // go to errorpage if an error is caught
    }
    return new ModelAndView( viewName: "errorpage");
}
```

The details are submitted through a html form. Then it checks if that user exists in the database. If the user cannot be found, the program reloads the page. Errors may occur, which will be caught through try-catch — returning the error page.

When a user successfully logs in, their lesson counts and outstanding balances are updated. An array list of users is made, then their IDs are used to get an array list of their uncounted lessons. An array list is used as it can have an indefinite capacity, and it can be easily iterated through a for loop. The iterator() method is not used as a for-loop is faster and there is no need for iterator's properties. The user's lesson count is only updated if the lesson has started, otherwise, my client may choose to cancel it. IsCounted will be updated to 1 so that it is not retrieved again. After all the pending lessons are iterated through the nested-for loop, a while loop is used. If lessonCount is above 1, ¼ of the monthly fee will be added.

Example of accessing database through DAO

```
public static ArrayList<Lesson> findUserIDLessonsNotCounted(int userID) {
    Connection conn = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    try {
        Class.forName( className: "com.mysql.cj.jdbc.Driver");
        conn = DriverManager.getConnection( url: mySqlConnect + myDB + "?useSSL=false", dbUser, dbPass);
        statement = conn.prepareStatement( sql: "SELECT * FROM lessons WHERE (idusers2 = ? AND isApproved!=? AND isCounted = ?)");
        statement.setInt( parameterIndex: 1, userID);
        statement.setInt( parameterIndex: 2,  x: 0);
        statement.setInt( parameterIndex: 3,  x: 0);
        resultSet = statement.executeQuery();
        ArrayList<Lesson> userLessonList = new ArrayList<>();
        while (resultSet.next()) {
            String date = resultSet.getString( columnLabel: "date");
            String time = resultSet.getString( columnLabel: "time");
            int lessonID = resultSet.getInt( columnLabel: "idlessons");
            Lesson userLesson = new Lesson(lessonID, date, time);
            userLessonList.add(userLesson);
        }
        return userLessonList;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    } finally {
        try {
            if (resultSet != null) {
                resultSet.close();
            }
            if (statement != null) {
                statement.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

This method uses the userID and gets the corresponding uncounted lessons from the database. A PreparedStatement is more secure as it separates data, preventing SQL injections. If an error occurs, nothing will be returned. Afterwards, the connection is closed. Error may occur, hence try-catch.

Example of Editing through DAO:

```
public static int editUser(int userID, String firstName, String lastName, String username, String password, String address, String studentGrade,
                           String studentDoB, int tuitionFee, int lessonDuration, int lessonCount, String emailAddress, String phoneNumber) {
    Connection con = null;
    PreparedStatement stm = null;
    try {
        Class.forName( className: "com.mysql.cj.jdbc.Driver");
        con = DriverManager.getConnection( url: mySqlConnect + myDB + "?useSSL=false", dbUser, dbPass);
        stm = con.prepareStatement( sql: "UPDATE db_users SET firstName = ?, lastName = ?, username = ?, password = ?, address = ?, grade = ?, DoB = ?," +
                " tuitionFee = ?, lessonDuration = ?, emailAddress = ?, contactNumber = ?, lessonCount = ? WHERE idusers = ? ");
        stm.setString( parameterIndex: 1, firstName);
        stm.setString( parameterIndex: 2, lastName);
        stm.setString( parameterIndex: 3, username);
        stm.setString( parameterIndex: 4, password);
        stm.setString( parameterIndex: 5, address);
        stm.setString( parameterIndex: 6, studentGrade);
        stm.setString( parameterIndex: 7, studentDoB);
        stm.setInt( parameterIndex: 8, tuitionFee);
        stm.setInt( parameterIndex: 9, lessonDuration);
        stm.setString( parameterIndex: 10, emailAddress);
        stm.setString( parameterIndex: 11, phoneNumber);
        stm.setInt( parameterIndex: 12, lessonCount);
        stm.setInt( parameterIndex: 13, userID);
        return stm.executeUpdate();
    } catch (Exception e) { // what is log
        System.out.println(e.getMessage());
        return -1;
    } finally {
        try {
            if (stm != null) {
                stm.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

This uses PreparedStatement as it prevents SQL injections and is more secure. The connection is closed so the limit is not reached.

Request Mapping when date is submitted

```
@RequestMapping("/submitDate")
public static ModelAndView submitDate(String lessonDate, HttpServletRequest request) throws ParseException {
    try{
        String date = request.getParameter( name: "date-string"); // gets the date the user clicked on in calendar
        if (date == null){
            date = lessonDate; // sometimes they make a lesson in calendar week, so use the date of this lesson, so they can see it in the week
        }
        //region Date Format
        DateFormat inputDateFormat = new SimpleDateFormat( pattern: "yyyy-MM-dd", Locale.getDefault());
        Date inputDate = inputDateFormat.parse(date); // parse into a date
        Calendar c = GregorianCalendar.getInstance();
        c.setTime(inputDate);
        c.setFirstDayOfWeek(Calendar.MONDAY); // find this week, starting monday
        c.set(Calendar.DAY_OF_WEEK, Calendar.MONDAY); // get the date of monday in the week
        DateFormat df = new SimpleDateFormat( pattern: "yyyy-MM-dd", Locale.getDefault());
        String monday = df.format(c.getTime()); // finding what monday is
        c.add(Calendar.DATE,  amount: 1); // add one day to the date in order to find tuesday
        String tuesday = df.format(c.getTime());
        c.add(Calendar.DATE,  amount: 1);
        String wednesday = df.format(c.getTime());
        c.add(Calendar.DATE,  amount: 1);
        String thursday = df.format(c.getTime());
        c.add(Calendar.DATE,  amount: 1);
        String friday = df.format(c.getTime());
        c.add(Calendar.DATE,  amount: 1);
        String saturday = df.format(c.getTime());
        c.add(Calendar.DATE,  amount: 1);
        String sunday = df.format(c.getTime());
        String[] weekdays; // initialising String array
        weekdays = new String[]{monday, tuesday, wednesday, thursday, friday, saturday, sunday}; // array with capacity 7
        ArrayList<Lesson>[] dayLessons; // array of arraylist of lessons
        dayLessons = new ArrayList[7]; // representing each day of the week
        for (int i = 0; i<weekdays.length; i++){
            dayLessons[i] = DAO.getDayLessons(weekdays[i]); // set each day to an array list of all the lessons on that day
        }
        if (sunday == date || sunday.equals(date)){ // sometimes clicking on sunday gives the following week, this amends this issue
            date = saturday;
        }
        // endregion
        HttpSession session = request.getSession();
        User sessionUser = (User) session.getAttribute( name: "sessionUser");
        if (sessionUser.getIsTeacher() == 0){ // validate if user is teacher or student
            PageController.calendarWeekStudent(date, dayLessons, request);
            return new ModelAndView( viewName: "calendar-week-student");
        } else if (sessionUser.getIsTeacher() == 1){
            PageController.calendarWeek(date, dayLessons, request);
            return new ModelAndView( viewName: "calendar-week");
        }
    } catch (Exception e){
        System.out.println("Error: " + e.getMessage());
        e.printStackTrace();
        return new ModelAndView( viewName: "errorpage"); // returns error page if error is caught
    }
    return new ModelAndView( viewName: "errorpage");
}
```

The method gets the date string submitted when the user clicks on a day, and finds the dates of the week it lies in. Date is in a String instead of Date as java and mySQL Dates are different. Using the dates of the week, an array with capacity 7 is created, representing Monday–Sunday. An array of an array list of lesson objects is used as arrays are faster to iterate through and there is a known number of days. Each index has the array list of lessons on that day, retrieved through the DAO. This is a 2-dimensional structure of arrays/array lists. Array lists instead of linked lists because they are faster in accessing data.

It submits the dayLessons to the calendar week controller.

Controller for CalendarWeek

```java
@RequestMapping("/calendar-week")
public static ModelAndView calendarWeek(String date, ArrayList<Lesson>[] dayLessons, HttpServletRequest request) {
    request.setAttribute( name: "dateInput", date); // date for finding the week around it
    ArrayList<String> studentList = DAO.getUsernameList(); // get all students
    request.setAttribute( name: "studentList", studentList); // set in jsp so teacher can select user instead of typing
    assert studentList != null; // prevent null point exceptions
    int studentListLength = studentList.size();
    request.setAttribute( name: "studentListLength", studentListLength); // javascript knows how many to iterate through
    HttpSession session = request.getSession();
    User sessionUser = (User) session.getAttribute( name: "sessionUser");
    request.setAttribute( name: "sessionUser", sessionUser); // for making block timings
    String errorMessage = (String) request.getAttribute( name: "errorMessage"); // error message from methods which might call this one
    request.setAttribute( name: "errorMessage", errorMessage);
    //region Set Weekday Lesson Arrays & Set Lengths
    ArrayList<Lesson> mondayLessons = dayLessons[0];
    request.setAttribute( name: "mondayLength", mondayLessons.size()); // so javascript knows how much to iterate: other weekdays below...
    //endregion

    //region Set Weekday Lesson Details
    request.setAttribute( name: "mondayLessons", mondayLessons);
    for (Lesson lesson : mondayLessons) { // enhanced for loop (iterate through the lessons of mondayLessons)
        User s = DAO.findLessonStudent(lesson.getuID()); // get the student who owns the lesson's details
        assert s != null; // prevent null point exceptions
        lesson.setUserFirstName(s.getFirstName()); // set in lesson class
        lesson.setUserLastName( s.getLastName());
        lesson.setUserAddress(  s.getAddress());
        lesson.setUserContact(  s.getContactNumber());
        lesson.setUserDuration( s.getLessonDuration());
    } // other weekdays below
    request.setAttribute( name: "tuesdayLessons", tuesdayLessons);
    for (Lesson lesson : tuesdayLessons) {...}
    request.setAttribute( name: "wednesdayLessons", wednesdayLessons);
    for (Lesson lesson : wednesdayLessons) {...}
    request.setAttribute( name: "thursdayLessons", thursdayLessons);
    for (Lesson lesson : thursdayLessons) {...}
    request.setAttribute( name: "fridayLessons", fridayLessons);
    for (Lesson lesson : fridayLessons) {...}
    request.setAttribute( name: "saturdayLessons", saturdayLessons);
    for (Lesson lesson : saturdayLessons) {...}
    request.setAttribute( name: "sundayLessons", sundayLessons);
    for (Lesson lesson : sundayLessons) {...}
    //endregion
    if (sessionUser.getIsTeacher() == 1){
        return new ModelAndView( viewName: "calendar-week");
    } else{
        return new ModelAndView( viewName: "errorpage");
    }
}
```

Tuesday-Sunday is cut out in the first region.
This sets the date the user click on as an Attribute so javascript can find the week. The student list is a set attribute for my client to easily select users from a list. The controller verifies the user status.

Calendar Week JS

```
var lengthIndex = 0;
var sessionUserID = $('#sessionUserID').val(); // get user id of session user for validation later
var sessionUserTeacher = $('#sessionUserTeacher').val(); // check if user is a teacher for validation later
const mondayLength = $('#mondayLength').val();
const tuesdayLength = $('#tuesdayLength').val();
const wednesdayLength = $('#wednesdayLength').val();
const thursdayLength = $('#thursdayLength').val();
const fridayLength = $('#fridayLength').val();
const saturdayLength = $('#saturdayLength').val();
const sundayLength = $('#sundayLength').val();
while ((lengthIndex < mondayLength) || (lengthIndex < tuesdayLength) || (lengthIndex < wednesdayLength) || (lengthIndex < thursdayLength) ||
(lengthIndex < fridayLength) || (lengthIndex < saturdayLength) || (lengthIndex < sundayLength)){ // iterate through the dayLessons until the last lesson (max length)
    var c = 0;
    for (var c = 0; c < e.length; c++){ // iterate through the days of the week
        var firstName = $('#firstName_'+c+lengthIndex).val();
        var lastName = $('#lastName_'+c+lengthIndex).val();
        var startTime = $('#time_'+c+lengthIndex).val();
        var lessonDate = $('#date_'+c+lengthIndex).val();
        var lessonUserID = $('#userID_'+c+lengthIndex).val();
        var lessonDuration = $('#duration_'+c+lengthIndex).val();
        var isApproved = $('#approved_'+c+lengthIndex).val();
        var endTime;
        var userAddress = $('#address_'+c+lengthIndex).val();
        var userContact = $('#contact_'+c+lengthIndex).val();
        if (sessionUserTeacher == 1){ // if user is a teacher
            if (firstName == null){ // non-existent lesson
                l.append(`<div id="blank-box"></div>`);  // add transparent box in between lessons
            } else if (isApproved == 1){ // approved student lesson
                var start = new Date(`${lessonDate} ${startTime}`); // get the start time as a date
                var end = new Date(start.getTime() + lessonDuration * 60000); // end time is the time + duration in minutes
                var hours = end.getHours().toString().padStart(2, '0');
                var minutes = end.getMinutes().toString().padStart(2, '0');
                var endTime = `${hours}:${minutes}`; // display end time in HH:MM format
                // add on purple lesson box showing lesson
                l.append(`<div id="lesson-box"><b>${firstName} ${lastName}</b><br>${startTime} - ${endTime}<br>${userAddress}<br> ${userContact}</div>`);
            } else if (isApproved == 0){ // unapproved student lesson
                var start = new Date(`${lessonDate} ${startTime}`);
                var end = new Date(start.getTime() + lessonDuration * 60000);
                var hours = end.getHours().toString().padStart(2, '0');
                var minutes = end.getMinutes().toString().padStart(2, '0');
                var endTime = `${hours}:${minutes}`; // display end time in HH:MM format
                // append pink lessons box showing lesson details
                l.append(`<div id="lesson-box1"><b>${firstName} ${lastName}</b><br>${startTime} - ${endTime}<br>${userAddress}<br> ${userContact}</div>`);
            } else if (isApproved == 2){ // if it is a blocked lessons
                var start = new Date(`${lessonDate} ${startTime}`);
                var end = new Date(start.getTime() + lessonDuration * 60000);
                var hours = end.getHours().toString().padStart(2, '0');
                var minutes = end.getMinutes().toString().padStart(2, '0');
                var endTime = `${hours}:${minutes}`;
                // append blue block with only time (name Block just for video Crit D)
                l.append(`<div id="lesson-box2"><b></b><br>${startTime} - ${endTime}</div>`);
            }
        } else if (sessionUserTeacher != 1){ // if the user is a student
            if (firstName == null ){ // if the lesson does not exist
            l.append(`<div id="blank-box"></div>`); // transparent lesson box
            } else if (isApproved == 1){ // for approved lessons
                if (sessionUserID == lessonUserID){ // if the user on the site owns the lessons, show details
                    var start = new Date(`${lessonDate} ${startTime}`);
                    var end = new Date(start.getTime() + lessonDuration * 60000);
                    var hours = end.getHours().toString().padStart(2, '0');
                    var minutes = end.getMinutes().toString().padStart(2, '0');
                    var endTime = `${hours}:${minutes}`;
                    // append purple box with details
                    l.append(`<div id="lesson-box"><b>${firstName} ${lastName}</b><br>${startTime} - ${endTime}<br>${userAddress}<br> ${userContact}</div>`);
                } else if (sessionUserID != lessonUserID){ // if the user does not match the lesson's user, hide details
                    var start = new Date(`${lessonDate} ${startTime}`);
                    var end = new Date(start.getTime() + lessonDuration * 60000);
                    var hours = end.getHours().toString().padStart(2, '0');
                    var minutes = end.getMinutes().toString().padStart(2, '0');
                    var endTime = `${hours}:${minutes}`;
                    // append purple box without details
                    l.append(`<div id="lesson-box">${startTime} - ${endTime}</div>`);
                }
            } else if (isApproved != 1){ // unapproved or blocked lesson
                if (sessionUserID == lessonUserID){ // if the user on the site owns the lessons, show details
                    var start = new Date(`${lessonDate} ${startTime}`);
                    var end = new Date(start.getTime() + lessonDuration * 60000);
                    var hours = end.getHours().toString().padStart(2, '0');
                    var minutes = end.getMinutes().toString().padStart(2, '0');
                    var endTime = `${hours}:${minutes}`;
                    // append pink box with details for unapproved
                    l.append(`<div id="lesson-box1"><b>${firstName} ${lastName}</b><br>${startTime} - ${endTime}<br>${userAddress}<br> ${userContact}</div>`);
                } else if (sessionUserID != lessonUserID){ // if the user does not match the lesson's user, hide details
                    var start = new Date(`${lessonDate} ${startTime}`);
                    var end = new Date(start.getTime() + lessonDuration * 60000);
                    var hours = end.getHours().toString().padStart(2, '0');
                    var minutes = end.getMinutes().toString().padStart(2, '0');
                    var endTime = `${hours}:${minutes}`;
                    // append purple box without details for blocked and unapproved
                    l.append(`<div id="lesson-box">${startTime} - ${endTime}</div>`);
                }
            }
        }
    }
    lengthIndex = lengthIndex + 1; // add to lengthIndex until while loop breaks
}
```

This JavaScript function is sent its values from hidden JSP input. LessonIndex must be less one of the dayLessons's lengths. The while loop iterates through lessonIndex. A nested for-loop in the while loop iterates through the variable, c, representing weekdays to add lessons as a table. The session user's status is verified here so students cannot access the details of other lessons. The calendar generator below, (not shown here) adapts someone else's code (B8bop) as it would be inefficient to code a new calendar. JavaScript is used for many pages due to its ability to perform calculations. A html library could be used, but JavaScript is very compatible with both HTML and CSS.

Homepage JSP:

```
<table>
    <tbody>
        <c:forEach items="${approvedLessonsToday}" var="lesson" varStatus="loop">
            <tr>
                <td>
                    <div class="lesson-box">
                    <h2>${lesson.userFirstName} ${lesson.userLastName}</h2>
                    <h3 style="display: none;" class="text" id="lessonDate${loop.index}">${lesson.lessonDate}</h3> <br>
                    <script>
                      const lessonDate${loop.index} = new Date(document.querySelector("#lessonDate${loop.index}").innerHTML);
                      const options${loop.index} = { year: 'numeric', month: '2-digit', day: '2-digit' };
                      document.querySelector("#lessonDate${loop.index}").innerHTML = lessonDate${loop.index}.toLocaleDateString('en-GB', options${loop.index});
                    </script>

                    <h3 class="text" id="lessonTime${loop.index}">${lesson.lessonTime}</h3>
                    <h2 style="display: none;" id="lDuration${loop.index}">${lesson.lDuration}</h2>
                    <h3 class="text"> - </h1><h3 class="text" id="lessonEndTime${loop.index}"></h3> <br>

                    <script>
                      const startTime${loop.index} = document.querySelector("#lessonTime${loop.index}").innerHTML.split(':');
                      console.log(startTime${loop.index});
                      const hours${loop.index} = parseInt(startTime${loop.index}[0], 10);
                      console.log(hours${loop.index});
                      const minutes${loop.index} = parseInt(startTime${loop.index}[1], 10);
                      console.log(minutes${loop.index});
                      const duration${loop.index} = parseInt(document.querySelector("#lDuration${loop.index}").innerHTML, 10);
                      console.log(duration${loop.index});

                      let endTime${loop.index} = new Date();
                      endTime${loop.index}.setHours(hours${loop.index}, minutes${loop.index} + duration${loop.index}, 0, 0);
                      console.log(endTime${loop.index});
                      console.log(endTime${loop.index});

                      if (endTime${loop.index}.getMinutes() >= 60) {
                        endTime${loop.index}.setMinutes(endTime${loop.index}.getMinutes() % 60);
                        hours${loop.index} = (hours${loop.index} + Math.floor(duration${loop.index} / 60)) % 24;
                        endTime${loop.index}.setHours(hours${loop.index});
                      }
                      console.log(endTime${loop.index});

                      const options1${loop.index} = { hour: 'numeric', minute: '2-digit',  hour12: false };
                      document.querySelector("#lessonEndTime${loop.index}").innerHTML = endTime${loop.index}.toLocaleTimeString('en-GB', options1${loop.index});
                    </script>

                    <h3 class="text">${lesson.userAddress}</h3> <br>
                    <h3 class="text">${lesson.userContact}</h3>

                    </div>
                </td>
            </tr>
        </c:forEach>
    </tbody>
</table>
```

The code above shows the usage of the jstl library through c:forEach (Tutorials Point) to loop lessons in html in order to display each lesson with its details, which I decided to use from searching on Stack Overflow (BalusC).

Homepage CSS:

```css
#homeinfo{
    width: 100%;
    position: relative;
}
#todayclmn{
    position relative;
    float: left;
    width: 59%;
    padding: 2px;
    height: 100%;
}

#today{
    position: relative;
    left: 15px;
}

#bookingreqbutton{
    position: relative;
    left: 15px;
    top: -10px;
    color: white;
    background-color: #FE82AA;
    background: linear-gradient(90deg, rgba(250,125,165,1) 1%, rgb(255, 136, 169, 0.9) 100%);
    height: 30px;
    border: 0px;
    border-radius: 7px;
    cursor: pointer;
    text-decoration: none;
    width: fit-content;
    padding: 5px 10px;

}

#bookingreqbutton:hover, #bookingreqbutton:focus {background-color:#EE729A }

@media only screen and (max-width: 980px) {
    h1{
    font-size: 40px;
    }
    h2{
    font-size: 35px;
    }
    p{
    font-size: 30px;
    }
    h3{
    font-size: 30px;
    }
    #homeinfo{
        width: 100%;
        position: relative;
    }
    #todayclmn{
        position relative;
        float: left;
        width: 59%;
        padding: 2px;
        height: auto;
    }
    #today{
        position: relative;
        left: 15px;
    }
    #bookingreqbutton{
        position: relative;
        left: 15px;
        top: -10px;
        color: white;
        background: linear-gradient(90deg, rgba(250,125,165,1) 1%, rgb(255, 136, 169, 0.9) 100%);
        border-radius: 7px;
        cursor: pointer;
        text-decoration: none;
        padding: 10px 20px;
        font-size: 20px;
    }
    #bookingreqbutton:hover, #bookingreqbutton:focus {background-color:#EE729A }
```

CSS is used so that the interface looks nice and the phone version is functional due to it needing a different GUI.

Bibliography

B8bop. *Code Pen*. n.d. March 2023. <https://codepen.io/B8bop>.

BalusC. *Stack Overflow*. n.d. March 2023. <https://stackoverflow.com/questions/2148658/iterate-over-elements-of-list-and-map-using-jstl-cforeach-tag>.

Tutorials Point. n.d. <https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm>.

Oracle. n.d. March 2023. <https://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/c/tld-summary.html>.

Code Conventions. 20 April 1999. March 2023. <https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html>.