Documentation for Assignment 1: To Do App

Class: WEBD3102 J2EE

Author: Lauren MacDonald

Date: February 22nd, 2024

1 CONTENTS

2 Project Overview

For completion of WEBD3102 Assignment 1, develop a task manager web application using JSP, Servlet, JDBC and MySQL. This document serves as an overview of the design and implementation of this application.

3 Project Requirements

3.1 Requirements – Use JSP, Servlet, JDBC, MySQL

3.1.1 JSP

Use Java Server Pages (JSP) to create dynamic web pages.

3.1.2 Servlet

Use Servlets to handle and respond to requests from the web page.

3.1.3 JDBC

Use Java database connectivity to access the MySQL database and perform CRUD operations.

3.1.4 MySQL

Set up a MySQL database to store the data.

3.2 Derived Requirements

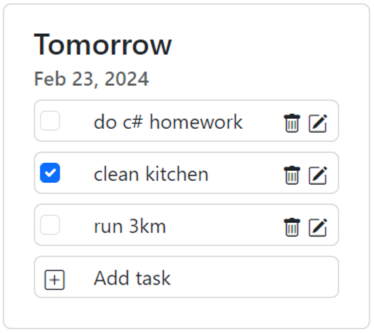
Create a responsive web application using model view controller (MVC) design.

4 Design Plans

4.1 UI Design

The to do list application has a simple user interface design with an option to see tasks due today or tasks due in the next 7 days. Each day will display the list of tasks in a card format with a checkbox for completion, an edit, and a delete button. The card will also have an add task button to add more tasks.

Color design meets accessibility requirements, as does proper aria labelling of icons. Bootstrap was used for styling to maintain simplicity and any adjustments to the styling was done in a CSS file titled 'styles.css'.



Picture 1 UI card design example

4.2 Database Schema Design

The database schema design is a simple table to hold task information, such as id, name, due date, priority category, completion status and date created. In future updates to the application the schema design can be updated to include a user table and integration for user authorization and authentication. Other information such as task category (household, schoolwork, fun, sports, etc.) could be added for further sorting and organization of tasks.

4.2.1 Task Table Data Dictionary

- taskId: int, primary key. This key is automatically assigned when a new record is created as uses auto-increment.
- taskName: varchar (100). The name of the task.
- dueDate: date. The due date of the task.
- category: varchar(150). The priority category of the task (high, medium, low).
- dateCreated: date. This date automatically assigns the current_timestamp for when the record is created.

4.3 Application Design

The application was designed to follow rough MVC architecture. General functionality is separated into different classes to keep things neat and organized.

4.3.1 Application Design

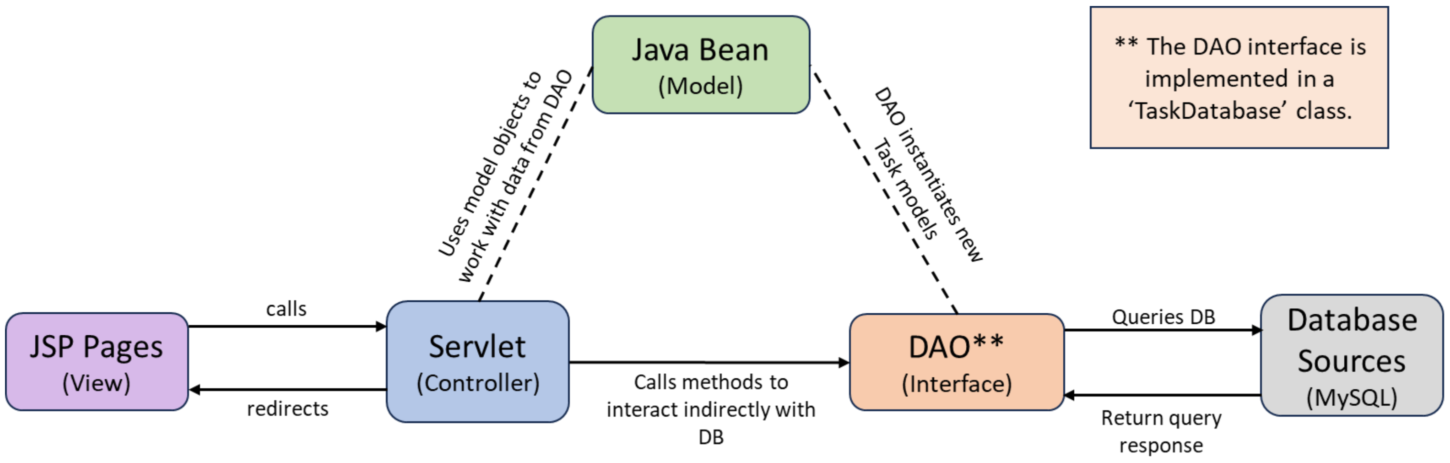
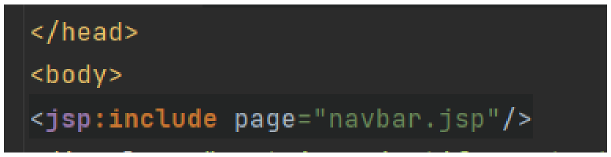

Java Bean
(Model)
** The DAO interface is implemented in a 'TaskDatabase' class.
Uses model objects to work with data from DAO
DAO instantiates new Task models
JSP Pages
(View)
calls
redirects
Servlet
(Controller)
Calls methods to interact indirectly with DB
DAO**
(Interface)
Queries DB
Return query response
Database Sources
(MySQL)

Figure 1 Application design diagram.

As seen in figure 1, the MySQL database is accessed via the data access object (DAO) interface (implemented in a class called TaskDatabase). This class includes methods to query the database (CRUD operations) and instantiates new Task models/beans to be used to hold the returned data. The servlet accesses these DAO methods to supply data to the JSP pages and update or delete any data sent via JSP requests. The servlet class may also instantiate new Task beans when necessary to send data back through the DAO to the database. The JSP pages holds HTML and displays data through JSP and Java code to dynamically load information to the user's browser.

4.4 JSP Design



```
</head>
<body>
<jsp:include page="navbar.jsp"/>
```

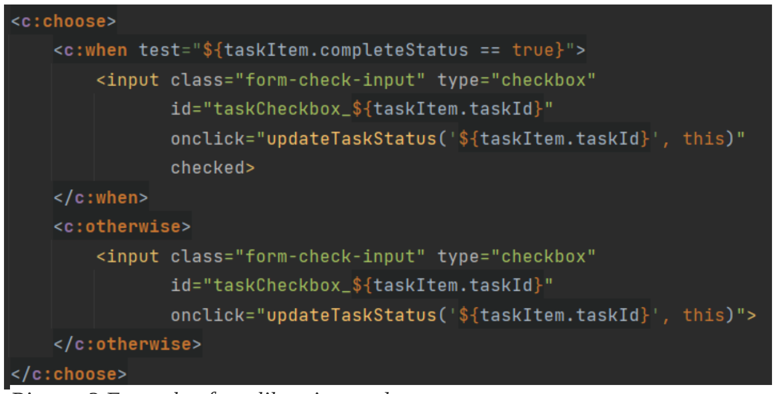
Picture 2 Example of navbar component added to page

4.4.1 'Components'

A few 'component' JSP pages were created to store JSP code that would otherwise be repeated, such as navigation and footers. In future iterations of this application, this component method could be used for the cards that display the task information in both the 7-day list and today options of tasks.

4.4.2 Tag libraries

Tag libraries such as the core library provided by java.sun.com were used to keep the code clean. Tag commands such as choose/when/otherwise were used. Tag libraries were also used to format date output.



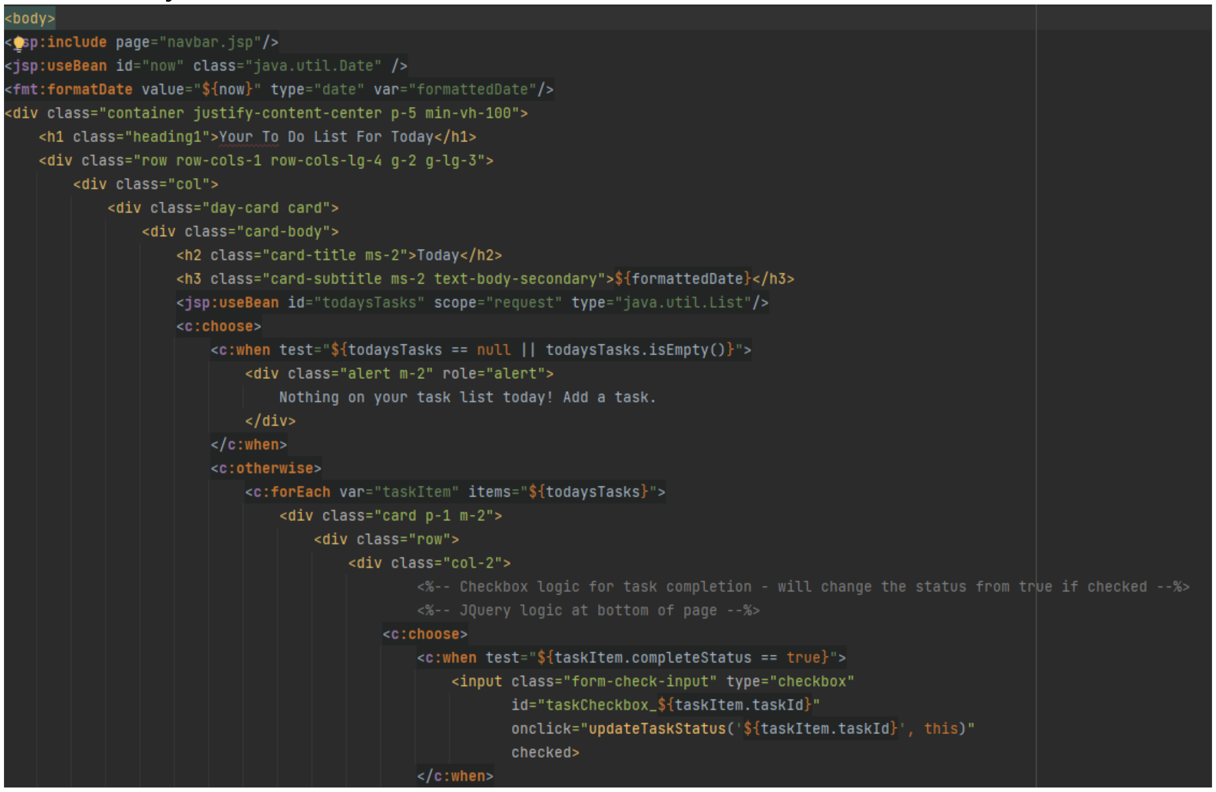
```
<c:choose>
    <c:when test="${taskItem.completeStatus == true}">
        <input class="form-check-input" type="checkbox"
               id="taskCheckbox_${taskItem.taskId}"
               onclick="updateTaskStatus('${taskItem.taskId}', this)"
               checked>
    </c:when>
    <c:otherwise>
        <input class="form-check-input" type="checkbox"
               id="taskCheckbox_${taskItem.taskId}"
               onclick="updateTaskStatus('${taskItem.taskId}', this)">
    </c:otherwise>
</c:choose>
```

Picture 3 Example of tag libraries used.

5 IMPLEMENTATION

5.1 JSP PAGES

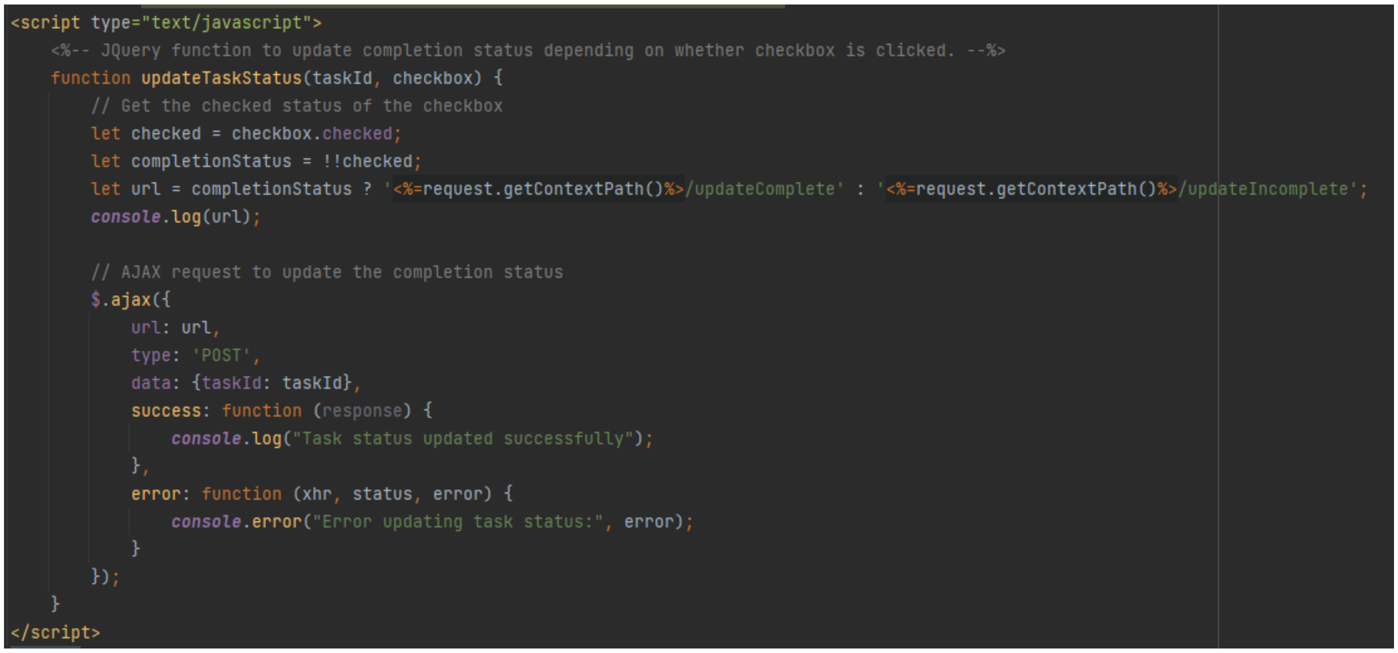
5.1.1 Today View



```
<body>
<jsp:include page="navbar.jsp"/>
<jsp:useBean id="now" class="java.util.Date" />
<fmt:formatDate value="${now}" type="date" var="formattedDate"/>
<div class="container justify-content-center p-5 min-vh-100">
    <h1 class="heading1">Your To Do List For Today</h1>
    <div class="row row-cols-1 row-cols-lg-4 g-2 g-lg-3">
        <div class="col">
            <div class="day-card card">
                <div class="card-body">
                    <h2 class="card-title ms-2">Today</h2>
                    <h3 class="card-subtitle ms-2 text-body-secondary">${formattedDate}</h3>
                    <jsp:useBean id="todaysTasks" scope="request" type="java.util.List"/>
                    <c:choose>
                        <c:when test="${todaysTasks == null || todaysTasks.isEmpty()}">
                            <div class="alert m-2" role="alert">
                                Nothing on your task list today! Add a task.
                            </div>
                        </c:when>
                        <c:otherwise>
                            <c:forEach var="taskItem" items="${todaysTasks}">
                                <div class="card p-1 m-2">
                                    <div class="row">
                                        <div class="col-2">
                                            <%-- Checkbox logic for task completion - will change the status from true if checked --%>
                                            <%-- JQuery logic at bottom of page --%>
                                        <c:choose>
                                            <c:when test="${taskItem.completeStatus == true}">
                                                <input class="form-check-input" type="checkbox"
                                                       id="taskCheckbox_${taskItem.taskId}"
                                                       onclick="updateTaskStatus('${taskItem.taskId}', this)"
                                                       checked>
                                            </c:when>
```

Picture 4 Screenshot of today.jsp.

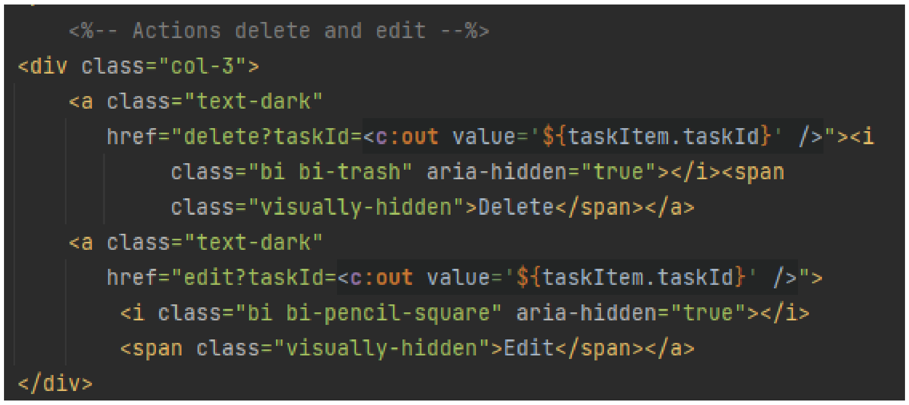
The 'today' view includes one card to display the tasks due for that day. A Java Bean for the java.util.Date class is used to access today's date to display on the card. A choose/when/otherwise statement is used to see if the list of tasks provided via the servlet is empty, and if there is nothing in the list of tasks then a message prompting the user to add a task is shown instead. If the list of tasks is not empty, a for-each loop will iterate through the list to print the tasks to the screen, including a completion checkbox, edit, and delete buttons.



```
<script type="text/javascript">
    <%-- JQuery function to update completion status depending on whether checkbox is clicked. --%>
    function updateTaskStatus(taskId, checkbox) {
        // Get the checked status of the checkbox
        let checked = checkbox.checked;
        let completionStatus = !!checked;
        let url = completionStatus ? '<%=request.getContextPath()%>/updateComplete' : '<%=request.getContextPath()%>/updateIncomplete';
        console.log(url);

        // AJAX request to update the completion status
        $.ajax({
            url: url,
            type: 'POST',
            data: {taskId: taskId},
            success: function (response) {
                console.log("Task status updated successfully");
            },
            error: function (xhr, status, error) {
                console.error("Error updating task status:", error);
            }
        });
    }
</script>
```

Picture 5 Screenshot of Javascript AJAX function.

A small Javascript script is used to update the completion status depending on whether the checkbox is checked. If the box is checked and the user clicks the checkbox, it will send a request to the servlet with the task id number to update the data in the database. The same is true for the opposite, as seen in the URL variable initialization.

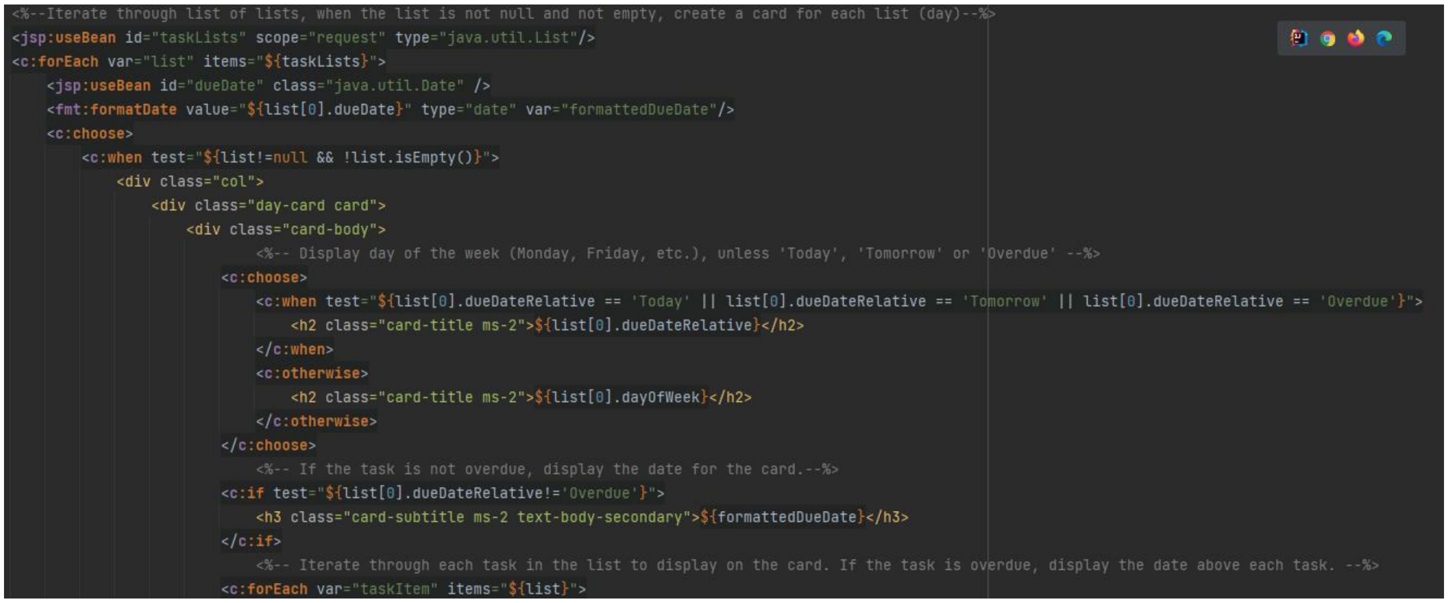


```
    <%-- Actions delete and edit --%>
<div class="col-3">
    <a class="text-dark"
       href="delete?taskId=<c:out value='${taskItem.taskId}' />"><i
            class="bi bi-trash" aria-hidden="true"></i><span
            class="visually-hidden">Delete</span></a>
    <a class="text-dark"
       href="edit?taskId=<c:out value='${taskItem.taskId}' />">
        <i class="bi bi-pencil-square" aria-hidden="true"></i>
        <span class="visually-hidden">Edit</span></a>
</div>
```

Picture 6 Screenshot of edit and delete code.

The links for edit and delete are visualized by icons (pencil for edit and trashcan for delete). If either of these icons are clicked, the task id will be forwarded to the servlet via the link URL and parameter. If edit is selected, the servlet will redirect to the form with the data associated with the Task object matching the task id so the form is pre-loaded with the details. If delete is chosen, the servlet directs the task id to the applicable delete method which will query the database in form of a delete statement and delete the record. To meet accessibility requirements, aria tags and span classes with contextual information are included for individuals who use alternative methods of accessing the website such as screen readers.

5.1.2 List view



```
<%--Iterate through list of lists, when the list is not null and not empty, create a card for each list (day)--%>
<jsp:useBean id="taskLists" scope="request" type="java.util.List"/>
<c:forEach var="list" items="${taskLists}">
    <jsp:useBean id="dueDate" class="java.util.Date" />
    <fmt:formatDate value="${list[0].dueDate}" type="date" var="formattedDueDate"/>
    <c:choose>
        <c:when test="${list!=null && !list.isEmpty()}">
            <div class="col">
                <div class="day-card card">
                    <div class="card-body">
                        <%-- Display day of the week (Monday, Friday, etc.), unless 'Today', 'Tomorrow' or 'Overdue' --%>
                        <c:choose>
                            <c:when test="${list[0].dueDateRelative == 'Today' || list[0].dueDateRelative == 'Tomorrow' || list[0].dueDateRelative == 'Overdue'}">
                                <h2 class="card-title ms-2">${list[0].dueDateRelative}</h2>
                            </c:when>
                            <c:otherwise>
                                <h2 class="card-title ms-2">${list[0].dayOfWeek}</h2>
                            </c:otherwise>
                        </c:choose>
                            <%-- If the task is not overdue, display the date for the card.--%>
                        <c:if test="${list[0].dueDateRelative!='Overdue'}">
                            <h3 class="card-subtitle ms-2 text-body-secondary">${formattedDueDate}</h3>
                        </c:if>
                            <%-- Iterate through each task in the list to display on the card. If the task is overdue, display the date above each task. --%>
                        <c:forEach var="taskItem" items="${list}">
```

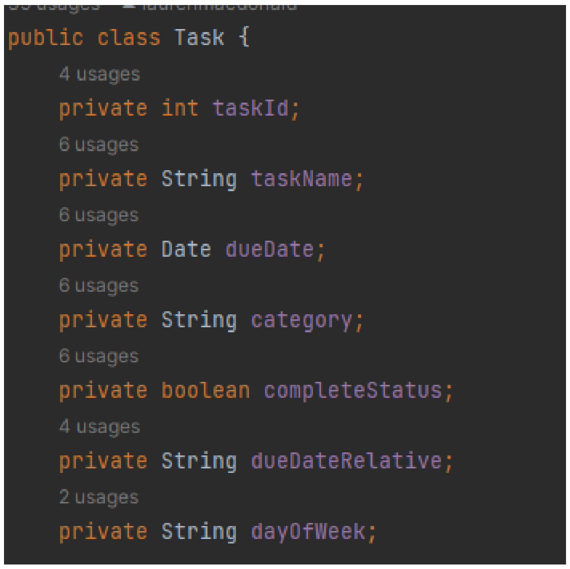
Picture 7 Screenshot of list view code

The list view uses similar logic for displaying tasks, however the data is accessed via a list of lists. The lists hold data relating to the relative due date of the task (today, tomorrow, 2 days, 3 days… 7 days, overdue) and the list holding these lists are iterated through to create the cards to hold them. The lists inside are then iterated through to display the applicable tasks.

Before these distinctions are made, a few choose/when/otherwise statements are used to display the date information (Today, Tomorrow, Overdue, or Monday, Tuesday, etc.). The dates will always display today, tomorrow and overdue, then if there are other days with tasks, they will display the day of the week. These relative dates are defined in the TaskDatabase class (implements the DAO interface). Once the dates are defined, the for-each loop is used to iterate through the list of tasks to display the information to the user.

The same Javascript logic is used for updating the completion status of the tasks by checkbox clicks and the same request logic is used for the edit and delete functions.

5.2 MODEL/BEAN CLASS

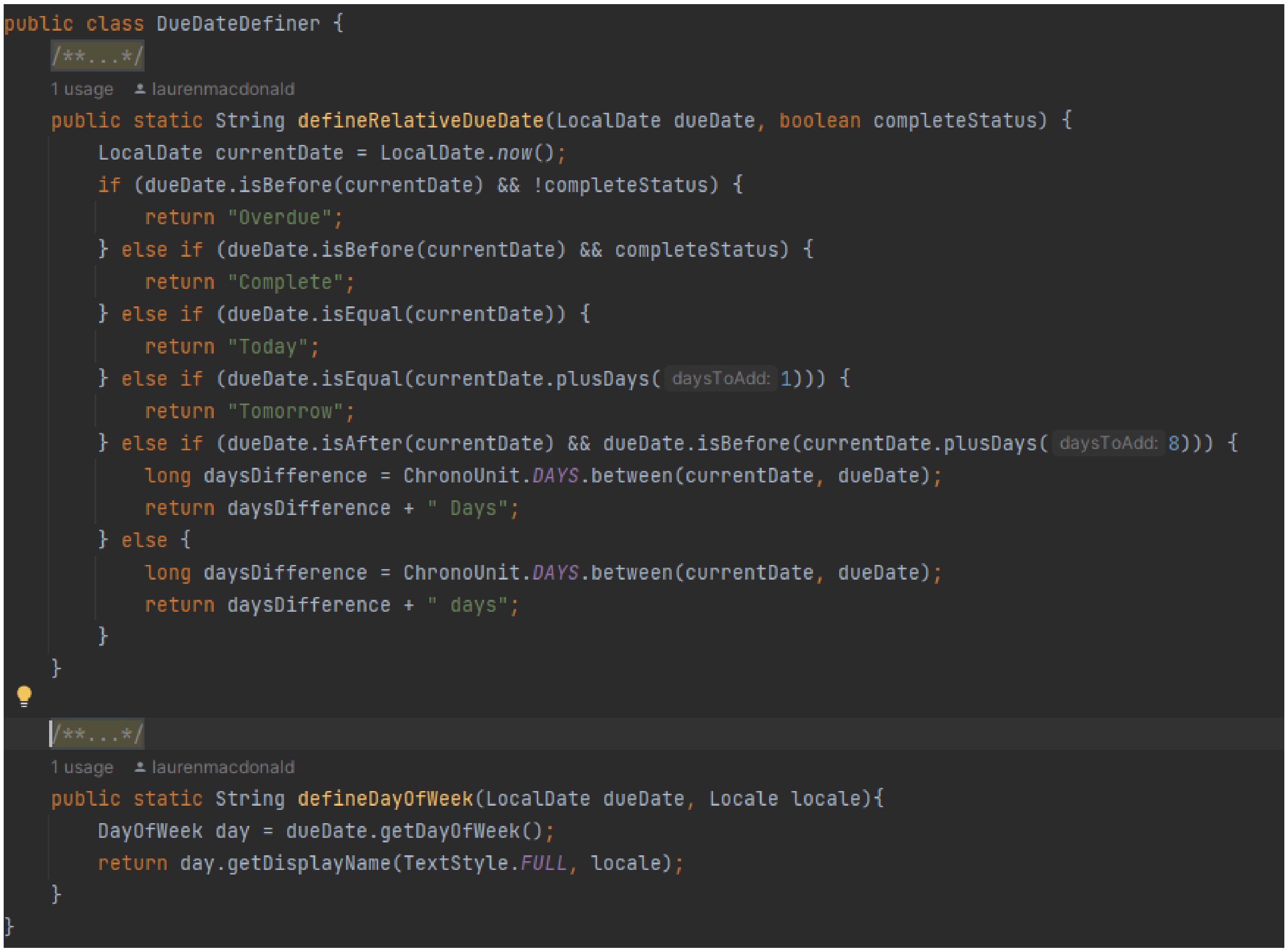


```
public class Task {
    4 usages
    private int taskId;
    6 usages
    private String taskName;
    6 usages
    private Date dueDate;
    6 usages
    private String category;
    6 usages
    private boolean completeStatus;
    4 usages
    private String dueDateRelative;
    2 usages
    private String dayOfWeek;
```

Picture 8 Screenshot of Task class attributes.

The Task class (model or bean) holds the same attributes as the task table in the database, but includes two relative due date attributes dueDateRelative and dayOfWeek for displaying to the user in an organized manner. These dates are defined via the DueDateDefiner class upon instantiation in the TaskDatabase class (where the objects are created from database data).

5.2.1 Due Date Definer Class



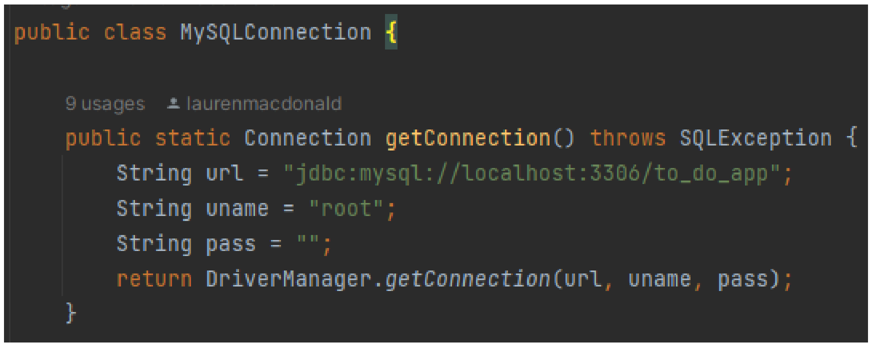
```
public class DueDateDefiner {
    /**...*/
    1 usage  laurenmacdonald
    public static String defineRelativeDueDate(LocalDate dueDate, boolean completeStatus) {
        LocalDate currentDate = LocalDate.now();
        if (dueDate.isBefore(currentDate) && !completeStatus) {
            return "Overdue";
        } else if (dueDate.isBefore(currentDate) && completeStatus) {
            return "Complete";
        } else if (dueDate.isEqual(currentDate)) {
            return "Today";
        } else if (dueDate.isEqual(currentDate.plusDays( daysToAdd: 1))) {
            return "Tomorrow";
        } else if (dueDate.isAfter(currentDate) && dueDate.isBefore(currentDate.plusDays( daysToAdd: 8))) {
            long daysDifference = ChronoUnit.DAYS.between(currentDate, dueDate);
            return daysDifference + " Days";
        } else {
            long daysDifference = ChronoUnit.DAYS.between(currentDate, dueDate);
            return daysDifference + " days";
        }
    }

    /**...*/
    1 usage  laurenmacdonald
    public static String defineDayOfWeek(LocalDate dueDate, Locale locale){
        DayOfWeek day = dueDate.getDayOfWeek();
        return day.getDisplayName(TextStyle.FULL, locale);
    }
}
```

Picture 9 Screenshot of the due date definer class

The due date definer class has two static methods for determining the two types of relative due dates and use java.util classes to handle the date data.

5.3 Database Connections

5.3.1 Database Connection Class

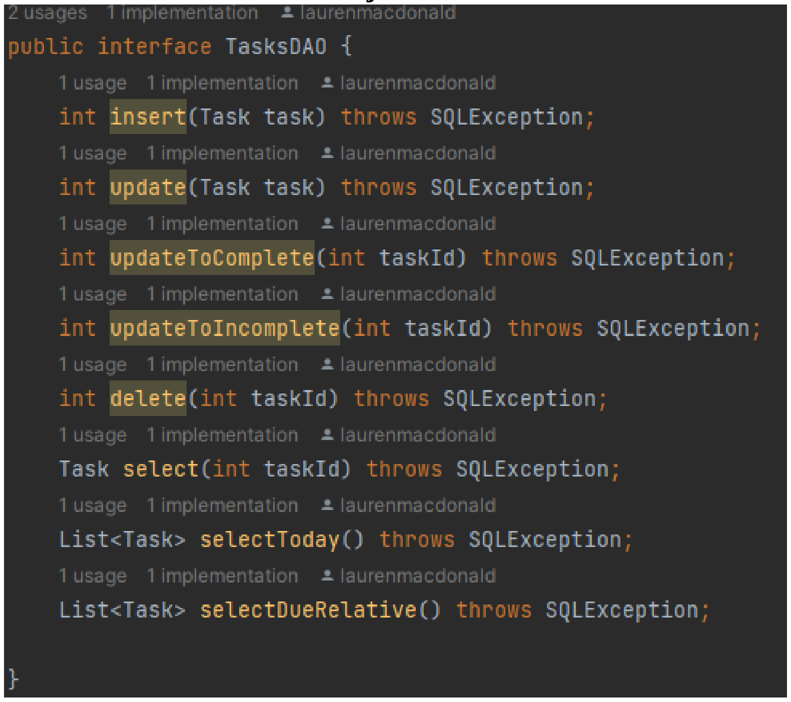


```
public class MySQLConnection {

    9 usages  laurenmacdonald
    public static Connection getConnection() throws SQLException {
        String url = "jdbc:mysql://localhost:3306/to_do_app";
        String uname = "root";
        String pass = "";
        return DriverManager.getConnection(url, uname, pass);
    }
```

Picture 10 Screenshot of MySQLConnection class.

The MySQLConnection class holds the static Connection method to connect to the MySQL database. This method is used to make a connection to the database to query the data.

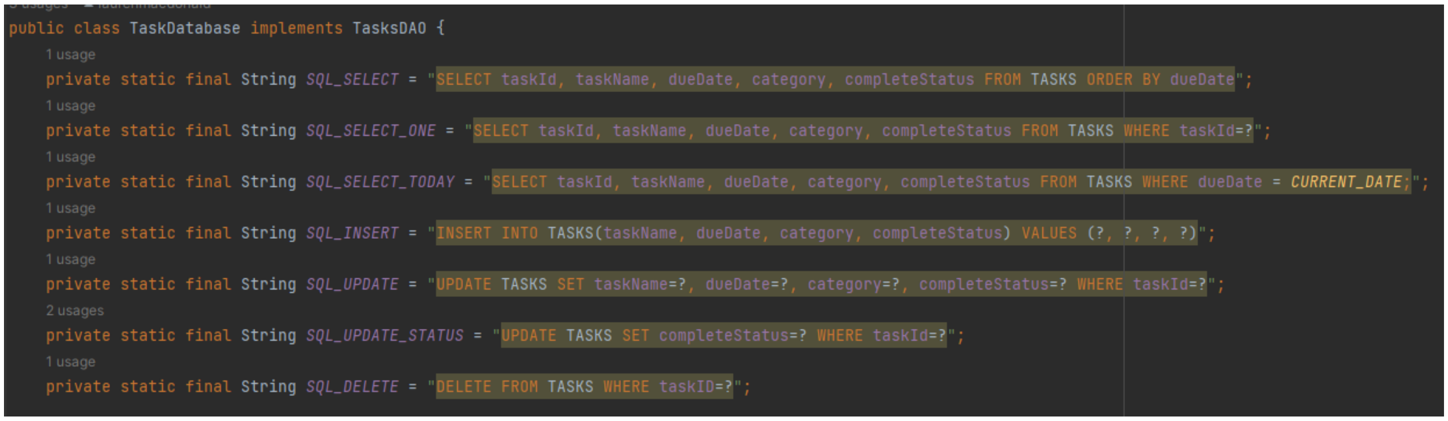
5.3.2 Task Data Access Object



```
2 usages  1 implementation  laurenmacdonald
public interface TasksDAO {
    1 usage  1 implementation  laurenmacdonald
    int insert(Task task) throws SQLException;
    1 usage  1 implementation  laurenmacdonald
    int update(Task task) throws SQLException;
    1 usage  1 implementation  laurenmacdonald
    int updateToComplete(int taskId) throws SQLException;
    1 usage  1 implementation  laurenmacdonald
    int updateToIncomplete(int taskId) throws SQLException;
    1 usage  1 implementation  laurenmacdonald
    int delete(int taskId) throws SQLException;
    1 usage  1 implementation  laurenmacdonald
    Task select(int taskId) throws SQLException;
    1 usage  1 implementation  laurenmacdonald
    List<Task> selectToday() throws SQLException;
    1 usage  1 implementation  laurenmacdonald
    List<Task> selectDueRelative() throws SQLException;

}
```

Picture 11 Screenshot of TasksDAO interface.

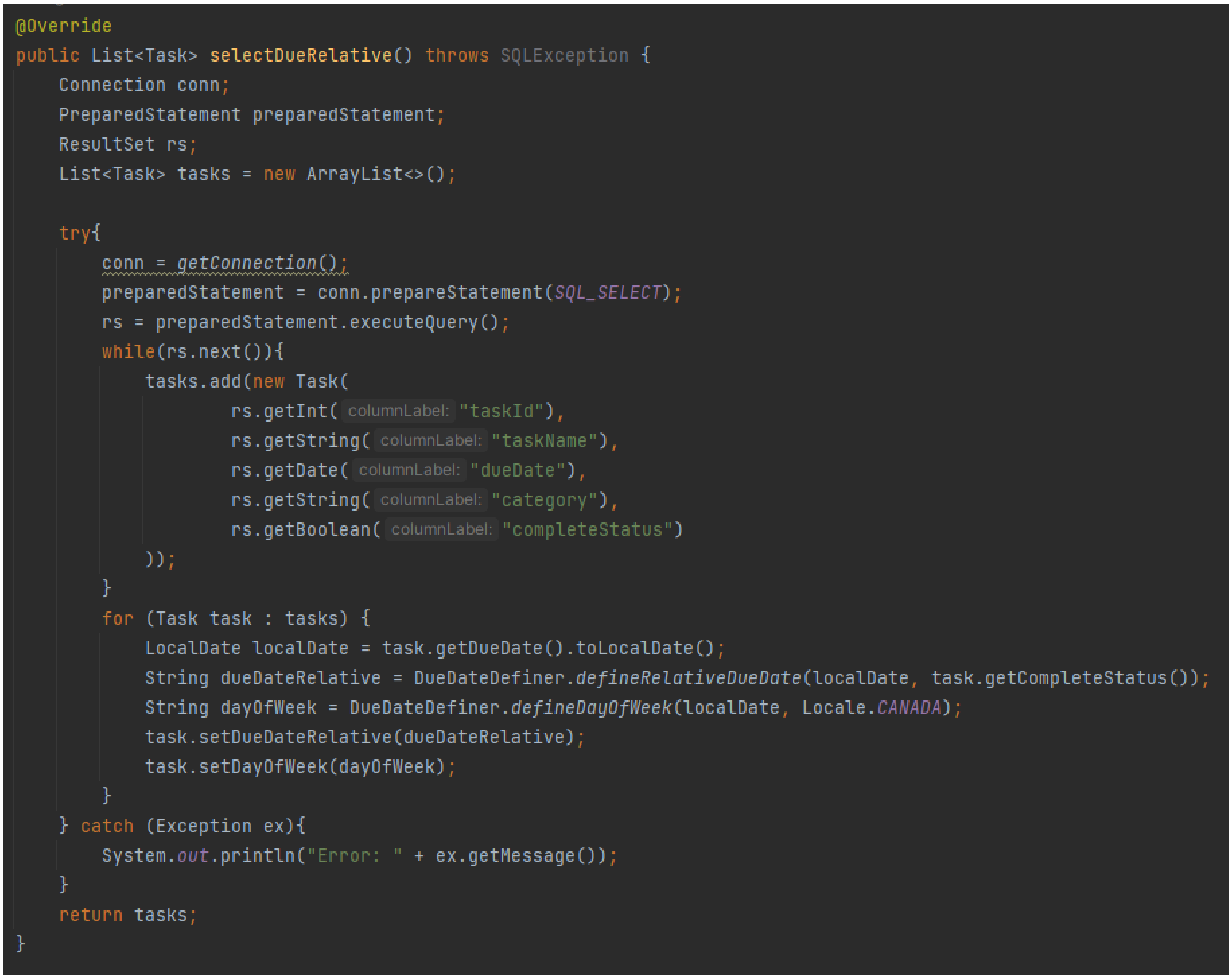
The data access object for Tasks is set up as an interface which is implemented in the TaskDatabase class.



```
public class TaskDatabase implements TasksDAO {
    1 usage
    private static final String SQL_SELECT = "SELECT taskId, taskName, dueDate, category, completeStatus FROM TASKS ORDER BY dueDate";
    1 usage
    private static final String SQL_SELECT_ONE = "SELECT taskId, taskName, dueDate, category, completeStatus FROM TASKS WHERE taskId=?";
    1 usage
    private static final String SQL_SELECT_TODAY = "SELECT taskId, taskName, dueDate, category, completeStatus FROM TASKS WHERE dueDate = CURRENT_DATE;";
    1 usage
    private static final String SQL_INSERT = "INSERT INTO TASKS(taskName, dueDate, category, completeStatus) VALUES (?, ?, ?, ?)";
    1 usage
    private static final String SQL_UPDATE = "UPDATE TASKS SET taskName=?, dueDate=?, category=?, completeStatus=? WHERE taskId=?";
    2 usages
    private static final String SQL_UPDATE_STATUS = "UPDATE TASKS SET completeStatus=? WHERE taskId=?";
    1 usage
    private static final String SQL_DELETE = "DELETE FROM TASKS WHERE taskID=?";
```

Picture 12 Screenshot 1 of TaskDatabase class.

The TaskDatabase class implements the TasksDAO interface and holds the SQL queries used in interacting with the database.



```
@Override
public List<Task> selectDueRelative() throws SQLException {
    Connection conn;
    PreparedStatement preparedStatement;
    ResultSet rs;
    List<Task> tasks = new ArrayList<>();

    try{
        conn = getConnection();
        preparedStatement = conn.prepareStatement(SQL_SELECT);
        rs = preparedStatement.executeQuery();
        while(rs.next()){
            tasks.add(new Task(
                    rs.getInt( columnLabel: "taskId"),
                    rs.getString( columnLabel: "taskName"),
                    rs.getDate( columnLabel: "dueDate"),
                    rs.getString( columnLabel: "category"),
                    rs.getBoolean( columnLabel: "completeStatus")
            ));
        }
        for (Task task : tasks) {
            LocalDate localDate = task.getDueDate().toLocalDate();
            String dueDateRelative = DueDateDefiner.defineRelativeDueDate(localDate, task.getCompleteStatus());
            String dayOfWeek = DueDateDefiner.defineDayOfWeek(localDate, Locale.CANADA);
            task.setDueDateRelative(dueDateRelative);
            task.setDayOfWeek(dayOfWeek);
        }
    } catch (Exception ex){
        System.out.println("Error: " + ex.getMessage());
    }
    return tasks;
}
```

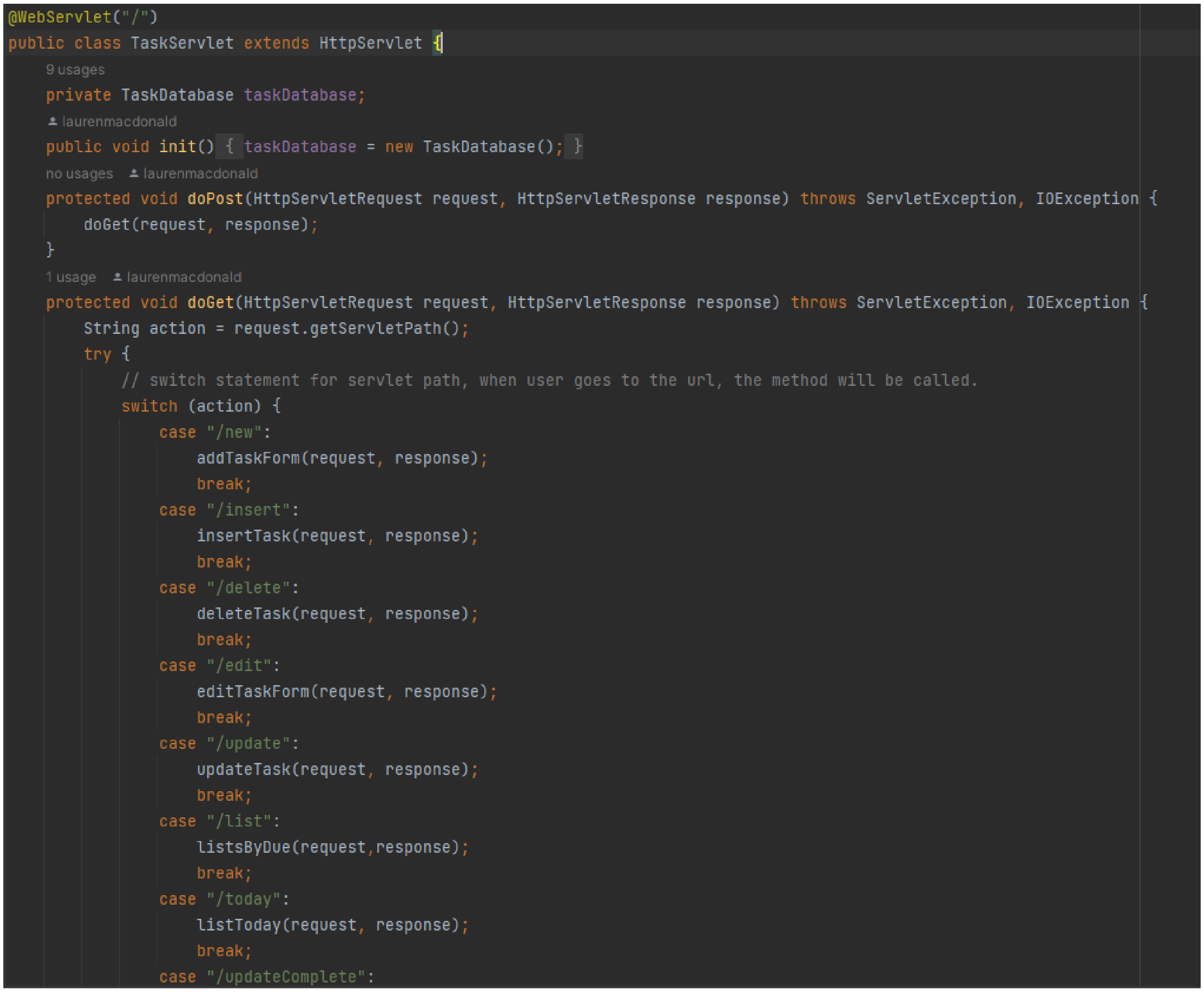
Picture 13 Screenshot 2 of the TaskDatabase class.

An example of one of the implemented methods is the selectDueRelative() method, which queries the database to get a list of all the tasks in the tasks table. New task object/beans are created, and relative due dates are created using the

DueDateDefiner utility class to sort the tasks by relative due dates. The method returns one list of all task objects, which have their relative due date and day of week attributes added.

5.4 Servlet Class

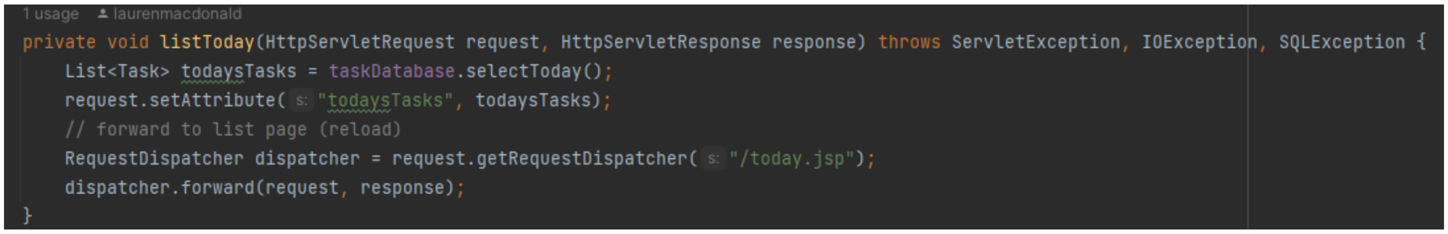
5.4.1 Servlet Class Initiation



```
@WebServlet("/")
public class TaskServlet extends HttpServlet {
    9 usages
    private TaskDatabase taskDatabase;
    laurenmacdonald
    public void init() { taskDatabase = new TaskDatabase(); }
    no usages  laurenmacdonald
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }
    1 usage  laurenmacdonald
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String action = request.getServletPath();
        try {
            // switch statement for servlet path, when user goes to the url, the method will be called.
            switch (action) {
                case "/new":
                    addTaskForm(request, response);
                    break;
                case "/insert":
                    insertTask(request, response);
                    break;
                case "/delete":
                    deleteTask(request, response);
                    break;
                case "/edit":
                    editTaskForm(request, response);
                    break;
                case "/update":
                    updateTask(request, response);
                    break;
                case "/list":
                    listsByDue(request,response);
                    break;
                case "/today":
                    listToday(request, response);
                    break;
                case "/updateComplete":
```

Picture 14 Screenshot 1 of servlet class

The servlet class connects to the database via the TaskDatabase class and handles any requests and responses from the JSP pages. A switch statement in the doGet method handles directing the request from the URL/servlet path to the appropriate method for CRUD operation interaction with the database.



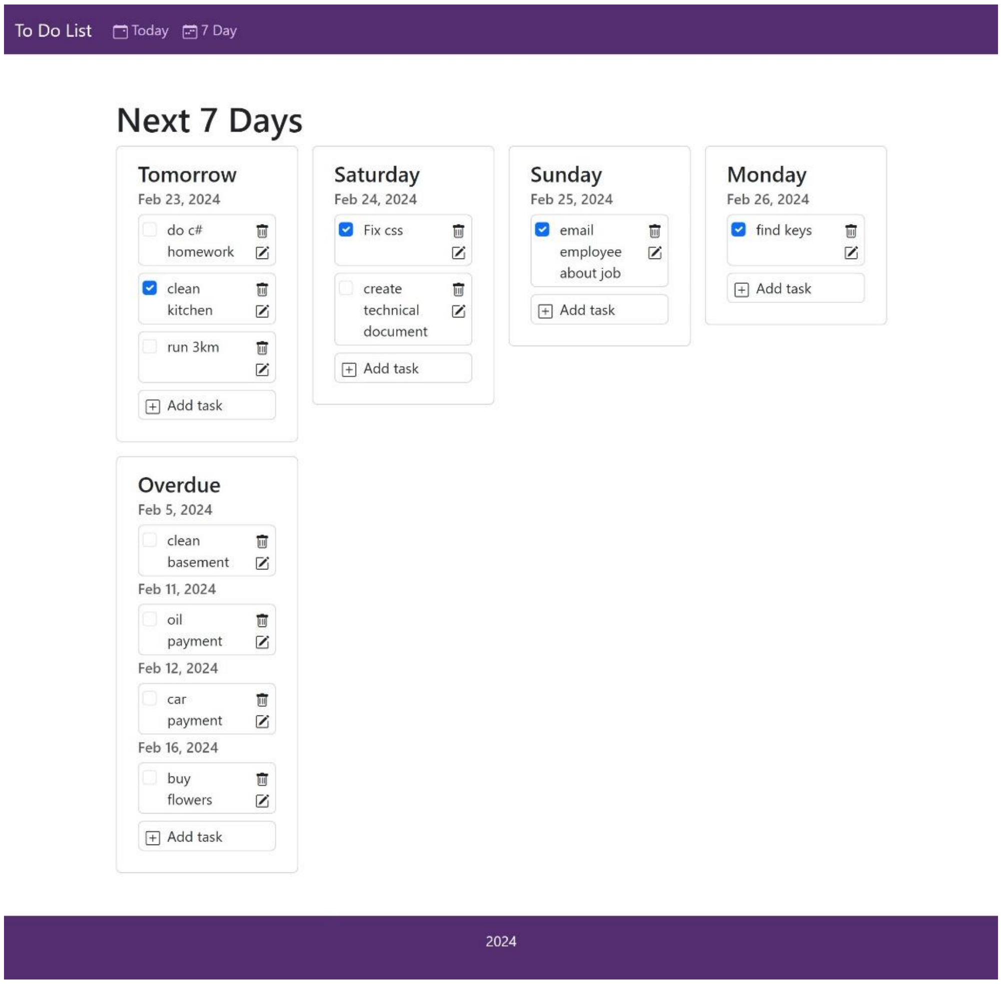
```
1 usage  laurenmacdonald
private void listToday(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException, SQLException {
    List<Task> todaysTasks = taskDatabase.selectToday();
    request.setAttribute( s: "todaysTasks", todaysTasks);
    // forward to list page (reload)
    RequestDispatcher dispatcher = request.getRequestDispatcher( s: "/today.jsp");
    dispatcher.forward(request, response);
}
```

Picture 15 Screenshot 2 of servlet class.

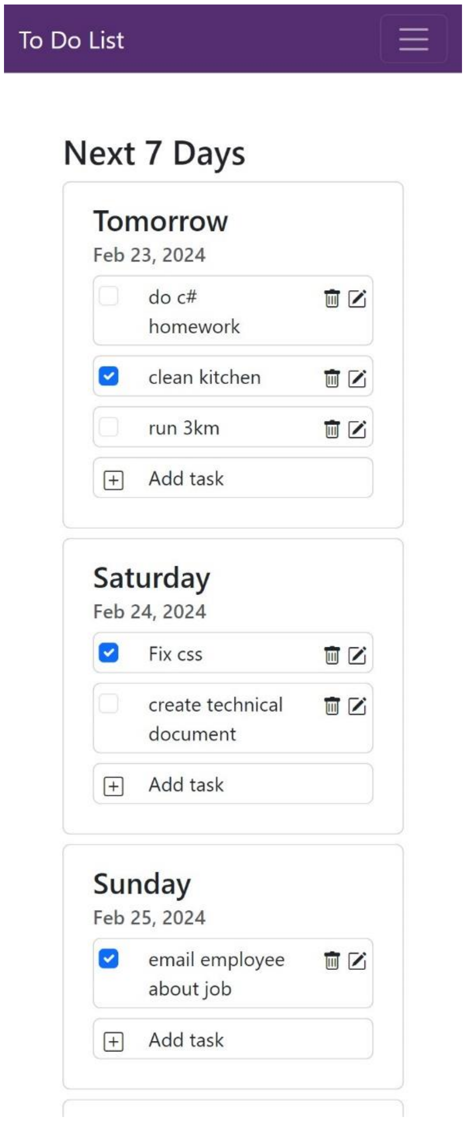
The above screenshot depicts the listToday method which is called when the user accesses the JSP page to see the tasks they have due for today. A new list of Task beans is instantiated via calling the method from the task database class to get a list of all tasks with the due date relating to ‘today’. This list is set as an attribute in the request, labelled ‘todaysTasks’ to be accessed via the JSP code in the today JSP page (iterate through and display information to user). The dispatcher sets the forward URL to the appropriate JSP page and redirects or reloads the page (aka loads the page with the task information). There are many methods that use this similar logic within this class.

6 APPLICATION SCREENSHOTS

The application starts at index.jsp, which redirects to the list of all tasks due over the next 7 days.



Picture 16 Screenshot of desktop 7 days page.


To Do List

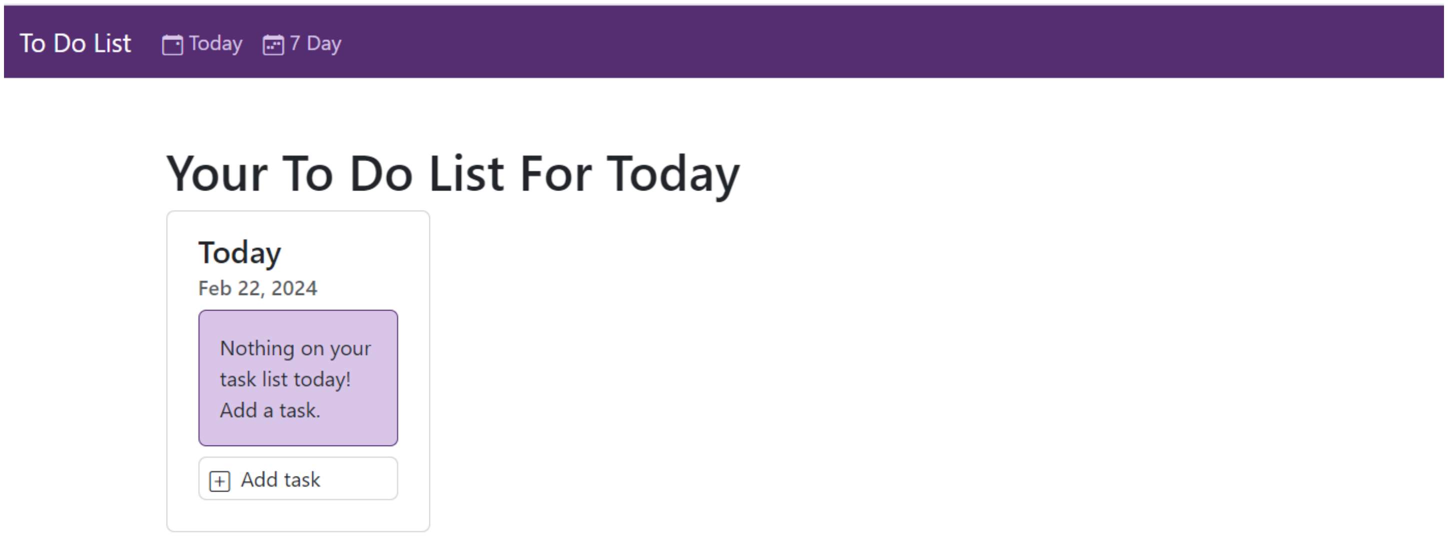
Next 7 Days

Tomorrow
Feb 23, 2024
- [ ] do c# homework
- [x] clean kitchen
- [ ] run 3km
Add task

Saturday
Feb 24, 2024
- [x] Fix css
- [ ] create technical document
Add task

Sunday
Feb 25, 2024
- [x] email employee about job
Add task

Picture 17 Screenshot of mobile 7 days page.

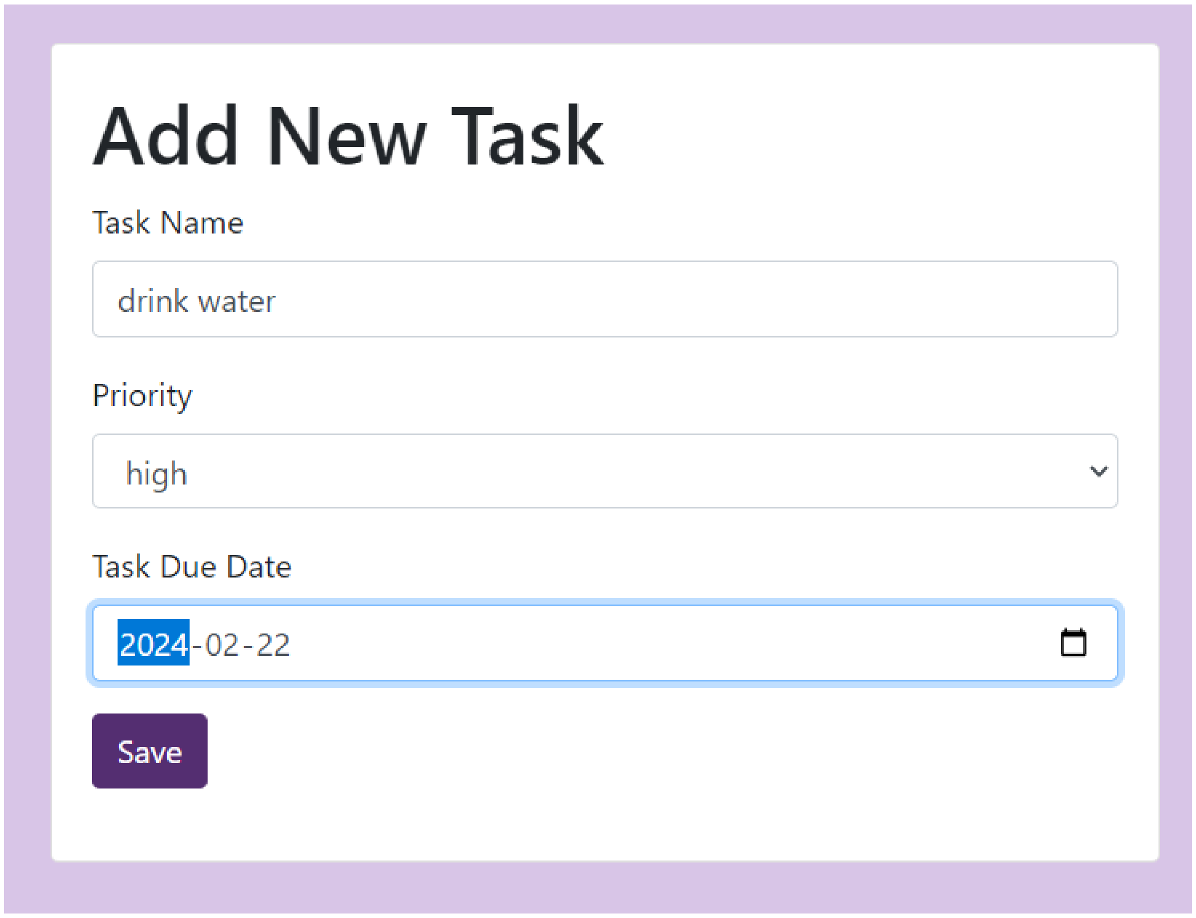

To Do List Today 7 Day

Your To Do List For Today

Today
Feb 22, 2024

Nothing on your task list today! Add a task.

Picture 18 Screenshot of desktop today page with error.

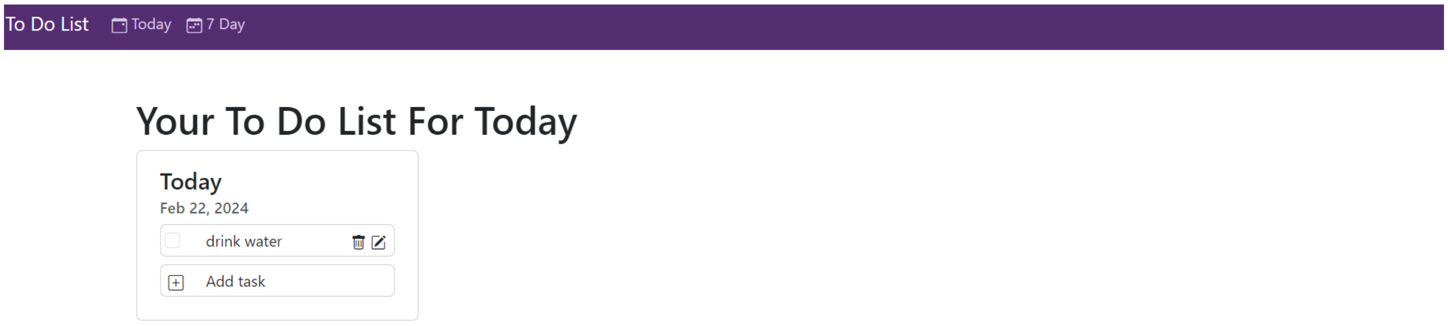

Add New Task

Task Name: drink water

Priority: high

Task Due Date: 2024-02-22

Picture 19 Screenshot of add new task form.


To Do List Today 7 Day

Your To Do List For Today

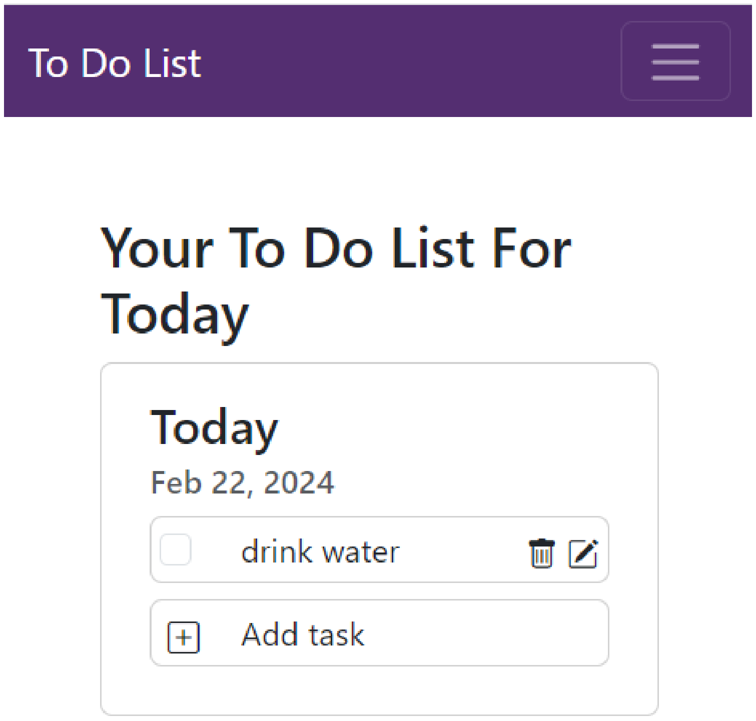
Today

Feb 22, 2024

drink water

Add task

Picture 20 Screenshot of updated today page.


To Do List

Your To Do List For Today

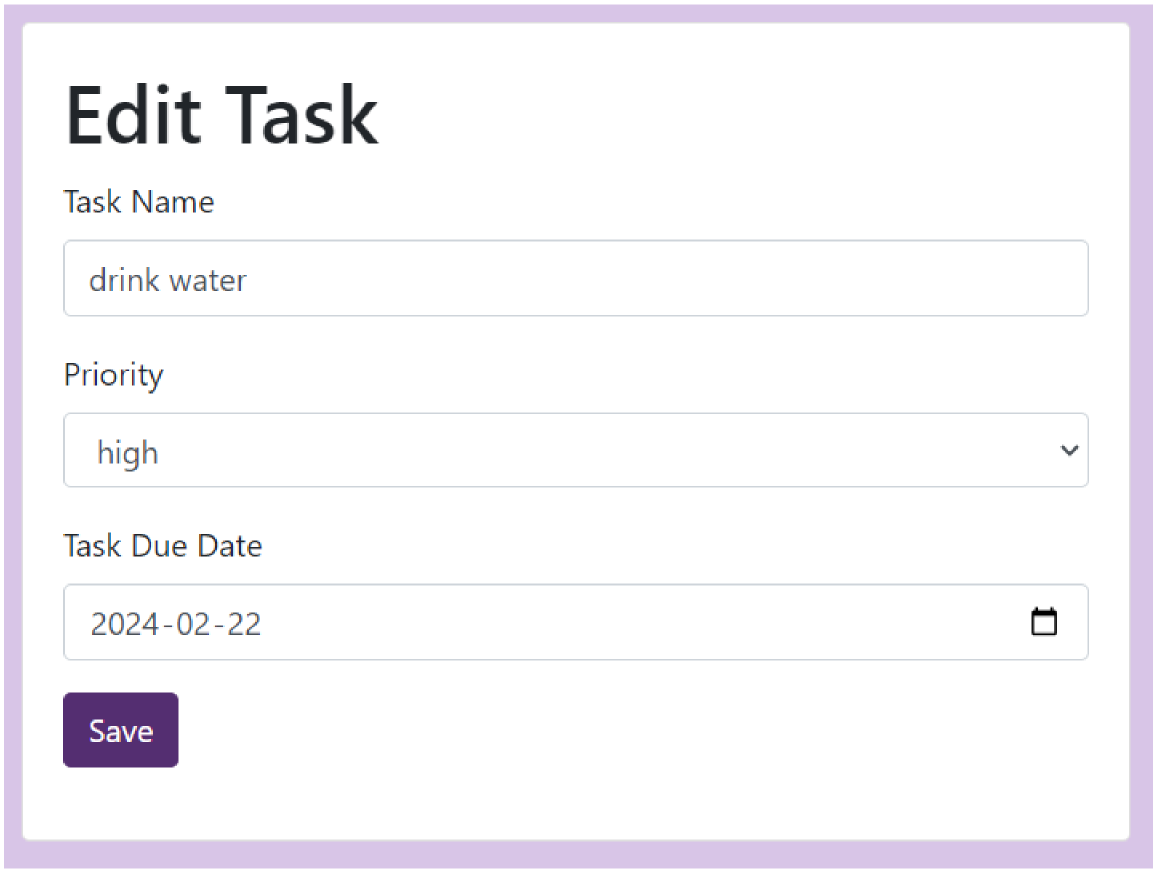
Today

Feb 22, 2024

drink water

Add task

Picture 21 Screenshot of mobile today page.


Edit Task

Task Name

drink water

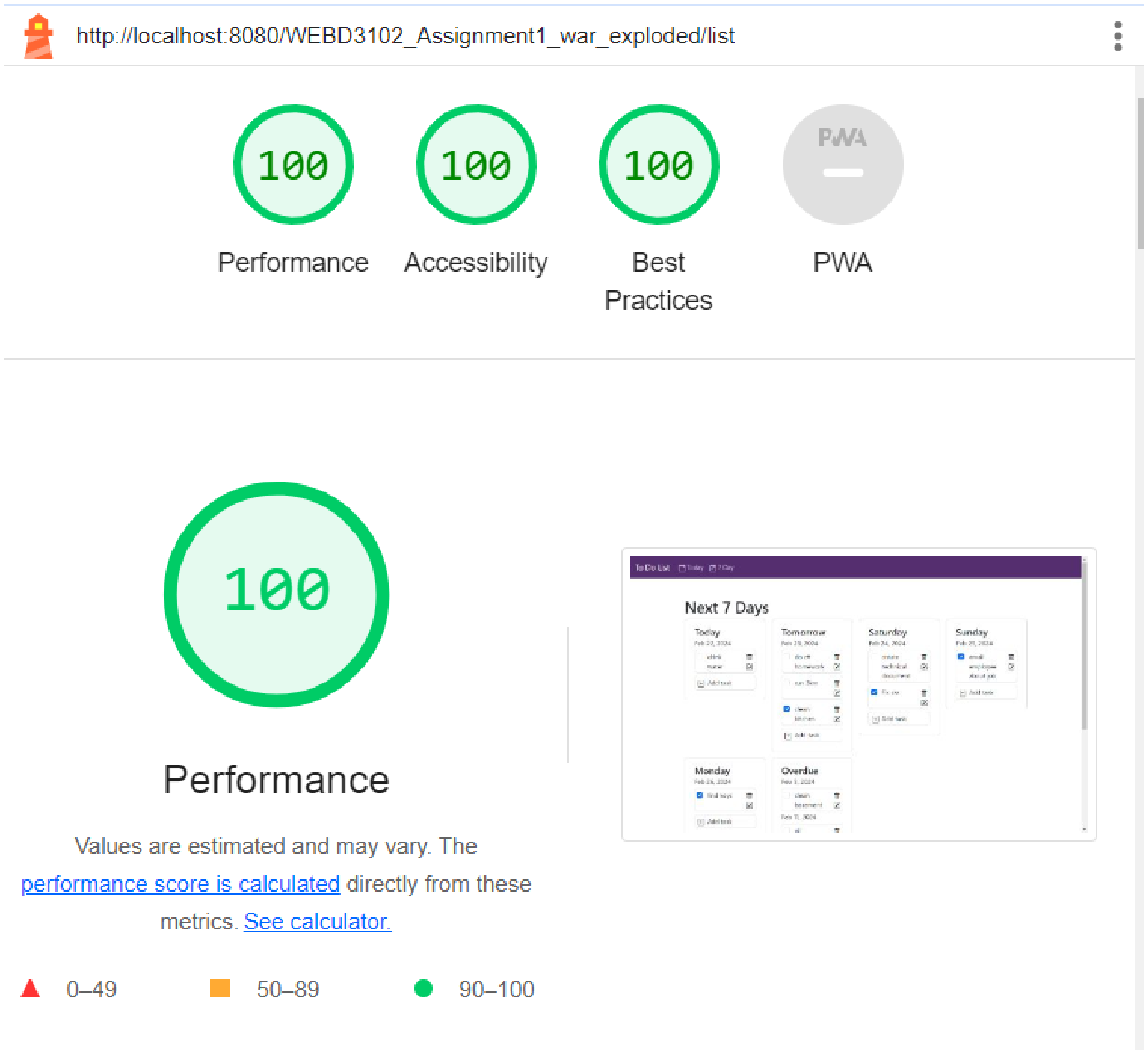
Priority

high

Task Due Date

2024-02-22

Picture 22 Screenshot of edit task form.


http://localhost:8080/WEBD3102_Assignment1_war_exploded/list

100 Performance

100 Accessibility

100 Best Practices

PWA

100

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

Picture 23 Chrome Dev Tools Lighthouse audit

The above screenshot demonstrates that performance, accessibility, and best practices standards have been met.