

Lesson Planbook App

Built into database to start:

Grades - 1-3

Subjects - Math, ELA, Science, Social Studies

Units - math units for grades 1-3

To enable users to find subjectIds (necessary to view units), users can view all subjects (READ).

To enable users to find unitIds (necessary to create lessons or to view all unit's lessons), users can view all units (by grade) (READ).

User can create lessons with name, objective, and content (CREATE).

User can edit lesson content (UPDATE).

User can delete a lesson (DELETE).

User can view lessons - individually (by id) or by unit (READ).

User can view a list of all lessons (READ).

User can "fill their planbook" by scheduling lessons for the days of the week (CREATE).

User can view their lesson by day (READ).

User can view their lessons for the week (READ).

User can reschedule a lesson for another day (UPDATE).

User can "clear their planbook", clearing all the lessons from the days, not deleting the lessons themselves (DELETE).

Database:

The screenshot shows the MySQL Workbench interface. The top menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. Below the menu is a toolbar with icons for various database operations. The Navigator pane on the left shows a list of schemas: children, customers, employees, goaltivity, jeep, pbdb, sakila, social_media_db, students, sys, teams, and world. The main editor pane displays a SQL script for creating a database and tables. The script is as follows:

```
1 • drop database if exists planbook;
2
3 • create database planbook;
4
5 • use planbook;
6
7 • drop table if exists dayslessons;
8 • drop table if exists lesson;
9 • drop table if exists unit;
10 • drop table if exists day;
11 • drop table if exists subject;
12 • drop table if exists grade;
13
14 • create table if not exists grade (
15     gradeNumber int unique not null,
16     gradeName varchar(15) not null,
17     primary key (gradeNumber)
18 );
19
20 • create table if not exists subject (
21     subjectId int auto_increment not null,
22     subjectName varchar(15) not null,
23     primary key (subjectId)
24 );
25
26 • create table if not exists day (
27     dayId int unique not null,
28     dayOfTheWeek varchar(10) not null,
29     primary key (dayId)
30 );
```

The bottom pane shows the 'Administration' tab with the 'Schemas' sub-tab selected. It displays the message 'No object selected'.

```
31
32 • create table if not exists unit (
33     unitId int auto_increment not null,
34     unitName varchar(25) not null,
35     subjectId int not null,
36     gradeNumber int not null,
37     primary key (unitId),
38     foreign key (subjectId) references subject(subjectId),
39     foreign key (gradeNumber) references grade(gradeNumber)
40 );
41
42 • create table lesson (
43     lessonId int not null auto_increment,
44     lessonName varchar(75) not null,
45     objective varchar(255) not null,
46     content text not null,
47     unitId int not null,
48     primary key (lessonId),
49     foreign key (unitId) references unit(unitId)
50 );
51
52 • create table DaysLessons (
53     dayslessonsId int not null auto_increment,
54     dayId int not null,
55     lessonId int not null,
56     primary key (dayslessonsId),
57     foreign key (dayId) references day(dayId),
58     foreign key (lessonId) references lesson(lessonID)
59 );
60
```

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator

playbook-schema* playbookDB

Schemas

Filter objects

- children
- customers
- employees
- goaltivity
- jeep
- pdbb
- sakila
- social_media_db
- students
- sys
- teams
- world

Administration Schemas


Information

No object selected

Limit to 1000 rows

```
1 * INSERT INTO day (dayId, dayOfTheWeek) VALUES (1, "Monday");
2 * INSERT INTO day (dayId, dayOfTheWeek) VALUES (2, "Tuesday");
3 * INSERT INTO day (dayId, dayOfTheWeek) VALUES (3, "Wednesday");
4 * INSERT INTO day (dayId, dayOfTheWeek) VALUES (4, "Thursday");
5 * INSERT INTO day (dayId, dayOfTheWeek) VALUES (5, "Friday");
6
7 * INSERT INTO grade (gradeNumber, gradeName) VALUES (1, "FirstGrade");
8 * INSERT INTO grade (gradeNumber, gradeName) VALUES (2, "SecondGrade");
9 * INSERT INTO grade (gradeNumber, gradeName) VALUES (3, "ThirdGrade");
10 -- INSERT INTO grade (gradeNumber, gradeName) VALUES (4, "FourthGrade");
11 -- INSERT INTO grade (gradeNumber, gradeName) VALUES (5, "FifthGrade");
12
13 * INSERT INTO subject (subjectId, subjectName) VALUES (1, "Math");
14 * INSERT INTO subject (subjectId, subjectName) VALUES (2, "ELA");
15 * INSERT INTO subject (subjectId, subjectName) VALUES (3, "Science");
16 * INSERT INTO subject (subjectId, subjectName) VALUES (4, "Social Studies");
17
18 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (1, "Geometry", 1, 1);
19 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (2, "PlaceValue", 1, 1);
20 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (3, "AdditionAndSubtraction", 1, 1);
21 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (4, "Measurement", 1, 1);
22 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (5, "Time", 1, 1);
23 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (6, "Money", 1, 1);
24 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (7, "Data", 1, 1);
25
26 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (8, "Geometry", 1, 2);
27 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (9, "PlaceValue", 1, 2);
28 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (10, "AdditionAndSubtraction", 1, 2);
29 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (11, "Measurement", 1, 2);
30 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (12, "Time", 1, 2);
31 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (13, "Money", 1, 2);
32 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (14, "Data", 1, 2);
33 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (15, "Multiplication", 1, 2);
34
35 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (16, "Geometry", 1, "3");
36 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (17, "PlaceValue", 1, 3);
37 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (18, "AdditionAndSubtraction", 1, 3);
38 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (19, "Measurement", 1, 3);
39 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (20, "Time", 1, 3);
40 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (21, "Data", 1, 3);
41 * INSERT INTO unit (unitId, unitName, subjectId, gradeNumber) VALUES (22, "MultiplicationAndDivision", 1, 3);
42
43 * INSERT INTO lesson (lessonName, objective, content, unitId) VALUES ("User Created Lesson1", "User Created Lesson1 Objective", "User Created Lesson1 Content", 1);
44 * INSERT INTO lesson (lessonName, objective, content, unitId) VALUES ("User Created Lesson2", "User Created Lesson2 Objective", "User Created Lesson2 Content", 1);
45 * INSERT INTO lesson (lessonName, objective, content, unitId) VALUES ("User Created Lesson3", "User Created Lesson3 Objective", "User Created Lesson3 Content", 1);
```

Swagger:

 Swagger
Version: 1.6.2 (2018-04-04)


v3/api-docs

Explore

Planbook Service

[View Swagger docs](#)

Servers

http://localhost:8080 - Local server. 

basic-lesson-controller

PUT

 /lesson Updates Lesson Content

POST

 /lesson Creates a Lesson

DELETE

 /lesson Deletes a Lesson

GET

 /lesson/{unitId} Returns a List of Lessons by Unit

GET

 /lesson/{lessonId} Returns a Lesson

GET

 /lesson/all Returns a list of Lessons

basic-days-lessons-controller

PUT

 /dayslessons Updates the Day of a Lesson

POST

 /dayslessons Assigns Lesson to Day

DELETE

 /dayslessons Unassigns a Lesson

GET

 /dayslessons/{dayOfTheWeek} Returns a list of Lessons for the Day

GET

 /dayslessons/all Returns a list of Lessons for the Week

DELETE

 /dayslessons/all Clears the Week's schedule of Lessons

basic-unit-controller

GET

 /unit Returns a list of Units by Subject by Grade

basic-subject-controller

GET

 /subject/all Returns a list of Subjects

Create Lesson

POST /lesson Creates a Lesson

Creates a Lesson

Parameters Cancel

Name	Description
lessonName * required string (query)	The lesson name <input type="text" value="User Created Lesson4"/>
objective * required string (query)	The lesson objective <input type="text" value="User Created Lesson4_Objective"/>
content * required string (query)	The lesson content <input type="text" value="User Created Lesson4_Content"/>
unitId * required integer (32 bit int) (query)	The unit id <input type="text" value="1"/>

Execute Clear

Get Lesson By ID / shows lesson 4 just created

GET /lesson/{lessonId} Returns a Lesson

Returns a Lesson

Parameters Cancel

Name	Description
lessonId * required integer (32 bit int) (query)	The lesson id <input type="text" value="4"/>

Execute Clear

Responses

curl

```
curl -X GET "http://localhost:8080/lesson/{lessonId}/lessonId=4" -H "accept: application/json"
```

Request URL

```
http://localhost:8080/lesson/{lessonId}/lessonId=4
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "lessonId": 4, "lessonName": "User Created Lesson", "objective": "User Created Lesson Objective", "content": "User Created Lesson Content", "unitId": 1 }</pre> Download

Update Lesson

basic-lesson-controller

PUT /lesson Updates Lesson Content

Updates Lesson Content

Parameters Cancel

Name	Description
lessonid * <small>required</small> integer(32bit) (query)	The lesson id
newContent * <small>required</small> string (query)	The new lesson content

Execute

Clear

Response

Out

```
curl -X PUT "http://localhost:8080/lesson/lessonid=4&newContent=UserCreatedLesson4_NEWCONTENT" -H "accept: application/json"
```

Request URL

```
http://localhost:8080/lesson/lessonid=4&newContent=UserCreatedLesson4_NEWCONTENT
```

List Lessons By Unit / shows updated lesson 4

GET `/lesson/{unitId}` Returns a List of Lessons by Unit

Returns a List of Lessons by Unit

Parameters Cancel

Name	Description
unitId <small>required</small> <code>integer(32bit)</code> <i>(query)</i>	The unit id

Execute Clear

Responses

200

Code **Details**

Request

curl -X GET "http://localhost:8080/lesson/{unitId}/unitId=1" -H "accept: application/json"

Request URL

http://localhost:8080/lesson/{unitId}/unitId=1

Server response

Response body

```
{
  "lessonId": 1,
  "lessonName": "User Created Lesson",
  "objective": "User Created lesson objective",
  "content": "User Created Lesson Content",
  "unitId": 1
},
{
  "lessonId": 2,
  "lessonName": "User Created Lesson",
  "objective": "User Created lesson objective",
  "content": "User Created Lesson Content",
  "unitId": 1
},
{
  "lessonId": 3,
  "lessonName": "User Created Lesson",
  "objective": "User Created lesson objective",
  "content": "User Created Lesson Content",
  "unitId": 1
},
{
  "lessonId": 4,
  "lessonName": "User Created Lesson",
  "objective": "User Created lesson objective",
  "content": "User Created Lesson NEW OBJECTIVE",
  "unitId": 1
}
```

Response headers

```
connection: keep-alive
content-type: application/json
date: Wed, 06 Sep 2017 22:38:37 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Download

Delete a Lesson

DELETE /lesson Deletes a Lesson

Deletes a Lesson

Parameters

Cancel

Name	Description
lessonid <small>* required</small> integer(\$int32) (query)	The lesson id
<input type="text" value="4"/>	

Execute

Clear

Responses

Curl

```
curl -X DELETE "http://localhost:8080/lesson/lessonId=4" -H "accept: application/json"
```

Request URL

```
http://localhost:8080/lesson/lessonId=4
```

Server response

Code

Details

200

Response body

Download

Response headers

```
connection: keep-alive  
content-type: application/json  
date: Wed, 06 Sep 2023 22:35:42 GMT  
keep-alive: timeout=60  
transfer-encoding: chunked
```

Responses

List All Lessons / shows lesson 4 has been deleted

The screenshot displays a REST client interface with the following sections:

- Method and URL:** GET /lesson/all. Description: Returns a list of Lessons.
- Parameters:** No parameters. A "Cancel" button is present.
- Buttons:** "Execute" and "Clear" buttons.
- Response:** A "200" status code is shown.
- Request Log:** curl -X GET "http://localhost:8080/lesson/all" -H "accept: application/json"
- Request URL:** http://localhost:8080/lesson/all
- Response Body:** A JSON array of two lesson objects. The second object, lesson 4, has a null value for the "content" field, indicating it has been deleted.

```
{
  "lessonId": 1,
  "lessonName": "User Created Lesson1",
  "objective": "User Created Lesson1 Objective",
  "content": "User Created Lesson1 Content",
  "unitId": 1
},
{
  "lessonId": 2,
  "lessonName": "User Created Lesson2",
  "objective": "User Created Lesson2 Objective",
  "content": "User Created Lesson2 Content",
  "unitId": 1
},
{
  "lessonId": 3,
  "lessonName": "User Created Lesson3",
  "objective": "User Created Lesson3 Objective",
  "content": "User Created Lesson3 Content",
  "unitId": 1
},
{
  "lessonId": 4,
  "lessonName": "User Created Lesson4",
  "objective": "User Created Lesson4 Objective",
  "content": null,
  "unitId": 1
}
}
```
- Response Headers:** A section for viewing headers, with a "Download" button.

Assign a Lesson to a Day

POST /daysLessons Assign Lesson to Day

Assigns Lesson to Day

Parameters

Cancel

Name	Description
dayId <small>required</small> integer(int32) (query)	The day id (i.e., '1')
lessonId <small>required</small> integer(int32) (query)	The lesson id

Execute

Clear

Responses

Curl

```
curl -X POST "http://localhost:8080/daysLessons/dayId-1/lessonId-1" -H "accept: application/json" -d ""
```

Request URL

```
http://localhost:8080/daysLessons/dayId-1/lessonId-1
```

Server response

List Lessons Schedule for the Week / shows Lesson 1 (and 2 &3) assigned

GET

/daysLessons/all Returns a list of Lessons for the Week.

Returns a list of Lessons for the Week

Parameters

No parameters

Cancel

Execute

Clear

Responses

Curl

curl -X GET "http://localhost:8080/daysLessons/all" -H "accept: application/json"

Request URL

http://localhost:8080/daysLessons/all

Server response

Code

Details

200

Response body

```
{
  "dayId": 1,
  "dayOfWeek": "Monday",
  "lessons": [
    {
      "lessonId": 1,
      "lessonName": "User Created Lesson",
      "objective": "User Created Lesson Objective",
      "content": "User Created Lesson Content",
      "moduleId": 1,
      "content": "User Created Lesson Content",
      "moduleId": 1
    },
    {
      "lessonId": 1,
      "lessonName": "User Created Lesson",
      "objective": "User Created Lesson Objective",
      "content": "User Created Lesson Content",
      "moduleId": 1,
      "content": "User Created Lesson Content",
      "moduleId": 1
    },
    {
      "lessonId": 1,
      "lessonName": "User Created Lesson",
      "objective": "User Created Lesson Objective",
      "content": "User Created Lesson Content",
      "moduleId": 1,
      "content": "User Created Lesson Content",
      "moduleId": 1
    }
  ]
}
```

Download

Reschedule a day's Lessons

basic-days-lessons-controller

PUT /daysLessons Updated the Day of a Lesson

Updated the Day of a Lesson

Parameters Cancel

Name	Description
lessonId * required Integer(32bit) (query)	The id of lesson to be rescheduled
<input type="text" value="1"/>	
newDay * required Integer(32bit) (query)	The id of the day to schedule the lesson
<input type="text" value="2"/>	

Execute

Clear

Responses

Call
curl -X PUT "http://localhost:8080/daysLessons?lessonId=1&newDay=2" -H "accept: application/json"

Request URL
http://localhost:8080/daysLessons?lessonId=1&newDay=2

Server response

Code	Details
200	<div>Response body <div></div><div>Download</div></div> <div>Response headers connection: keep-alive content-type: application/json date: Wed, 04 Sep 2024 22:42:12 GMT keep-alive: timeout=60 transfer encoding: chunked</div>

Remove a Lesson from a Day

DELETE /daysLessons Unschedule a Lesson

Unschedule a Lesson

Parameters Cancel

Name	Description
lessonId <small>* required</small>	The lesson id
integer (limit 32) (query)	<input type="text" value="3"/>

Execute Clear

Responses

Call

```
curl -X DELETE "http://localhost:8080/daysLessons?lessonId=3" -H "accept: application/json"
```

Request URL

```
http://localhost:8080/daysLessons?lessonId=3
```

Server response

Code	Details
200	<div>Response body</div> <div></div>

Show a Day's Lessons / shows lesson 2 rescheduled from Monday (dayId 1) to Tuesday (dayId 2)

GET

/daysLessons/{dayOfTheWeek}

Returns a list of Lessons for the Day

Returns a list of Lessons for the Day

Parameters

Cancel

Name	Description
dayOfTheWeek	The day of the week (i.e., 'Monday')
string (query)	<input type="text" value="Tuesday"/>

ExecuteClear

Responses

curl -X GET "http://localhost:8080/daysLessons/{dayOfTheWeek}/{dayOfTheWeek=Tuesday}" -H "accept: application/json"

Request URL

http://localhost:8080/daysLessons/{dayOfTheWeek}/{dayOfTheWeek=Tuesday}

Server response

Code	Details
200	<div>Response body</div> <div><pre>{ "lessonId": 1, "lessonName": "User Created Lesson", "subject": "User Created Lesson Subject", "content": "User Created Lesson Content", "unitId": 1 }</pre></div> <div>Download</div>

Response headers

```
connection: keep-alive
content-type: application/json
date: Wed, 08 Sep 2021 23:12:40 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Successful

Cleared All Lessons from the Week

DELETE /daysLessons/all Clears the Week's schedule of Lessons

Clears the Week's schedule of Lessons

Parameters Cancel

No parameters

Execute

Clear

Responses

Curl

```
curl -X DELETE "http://localhost:8080/daysLessons/all" -H "accept: application/json"
```

Request URL

```
http://localhost:8080/daysLessons/all
```

Server response

Code	Details
200	<div>Response headers<pre>Connection: keep-alive Content-Length: 0 Date: Wed, 04 Sep 2025 23:18:25 GMT Keep-Alive: timeout=60</pre></div>

Responses

Code	Description	Links
200	The week's lessons were cleared	No links

Cleared of lessons

GET /daysLessons/all Returns a list of Lessons for the Week

Returns a list of Lessons for the Week

Parameters Cancel

No parameters

Execute

Clear

Responses

Curl

```
curl -X GET "http://localhost:8080/daysLessons/all" -H "accept: application/json"
```

Request URL

```
http://localhost:8080/daysLessons/all
```

Server response

Code	Details
200	<div>Response body<pre>[]</pre></div> <div>Response headers<pre>Connection: keep-alive</pre></div>

View Subjects

GET

/subject/all Returns a list of Subjects

Returns a list of Subjects

Parameters

No parameters

Cancel

Execute

Clear

Responses

Diff

curl -X GET "http://localhost:8080/subject/all" -H "accept: application/json"

Request URL

http://localhost:8080/subject/all

Server response

Code

Details

200

Response body

```
{
  "subjectId": 1,
  "subjectName": "Math"
},
{
  "subjectId": 2,
  "subjectName": "ELA"
},
{
  "subjectId": 3,
  "subjectName": "Science"
},
{
  "subjectId": 4,
  "subjectName": "Social Studies"
}
```

Download

View Units

GET /unit Returns a List of Units by Subject by Grade

Returns a List of Units by Subject by Grade

Parameters

Cancel

Name	Description
subjectId * required Integer(32bit) (query)	The subject id <input type="text" value="1"/>
gradeNumber * required Integer(32bit) (query)	The grade number <input type="text" value="1"/>

Execute

Clear

Responses

Curl

curl -X GET "http://localhost:8080/unit/subjectId=1&gradeNumber=1" -H "accept: application/json"

Request URL

http://localhost:8080/unit/subjectId=1&gradeNumber=1

Server response

Code

Details

200

Response body

```
[{"unitId": 1, "unitName": "Integer", "subjectId": 1, "gradeNumber": 1}, {"unitId": 2, "unitName": "Place Value", "subjectId": 1, "gradeNumber": 1}, {"unitId": 3, "unitName": "Addition/Subtraction", "subjectId": 1, "gradeNumber": 1}, {"unitId": 4, "unitName": "Measurement", "subjectId": 1, "gradeNumber": 1}, {"unitId": 5, "unitName": "Time", "subjectId": 1, "gradeNumber": 1}]
```

Download

Response headers

ComponentScanMarker.java

```
1 package staples.heather;  
2  
3 public interface ComponentScanMarker {  
4  
5 }  
6
```

application.yaml

```
1 spring:  
2   datasource:  
3     password: Jinghy6713  
4     username: root  
5     url: jdbc:mysql://localhost:3306/planbook  
6  
7   logging:  
8     level:  
9       root: warn  
10      '[staples.heather]': debug
```

```

1 GlobalErrorHandler.java
2 package staples.heather.planbook.errorHandler;
3
4 import java.time.ZonedDateTime;
5
6 @RestControllerAdvice
7 @Slf4j
8 public class GlobalErrorHandler {
9     private enum LogStatus {
10         STACK_TRACE, MESSAGE_ONLY
11     }
12
13     @ExceptionHandler({ConstraintViolationException.class})
14     @ResponseStatus(code = HttpStatus.BAD_REQUEST)
15     public Map<String, Object> handleConstraintViolationException(ConstraintViolationException e, WebRequest webRequest) {
16         return createExceptionMessage(e, HttpStatus.BAD_REQUEST, webRequest, LogStatus.MESSAGE_ONLY);
17     }
18
19     @ExceptionHandler({DataIntegrityViolationException.class})
20     @ResponseStatus(code = HttpStatus.BAD_REQUEST)
21     public Map<String, Object> handleConstraintViolationException(DataIntegrityViolationException e, WebRequest webRequest) {
22         return createExceptionMessage(e, HttpStatus.BAD_REQUEST, webRequest, LogStatus.MESSAGE_ONLY);
23     }
24
25     @ExceptionHandler({Exception.class})
26     @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
27     public Map<String, Object> handleException(Exception e, WebRequest webRequest) {
28         return createExceptionMessage(e, HttpStatus.INTERNAL_SERVER_ERROR, webRequest, LogStatus.MESSAGE_ONLY);
29     }
30
31     @ExceptionHandler({NoSuchElementException.class})
32     @ResponseStatus(code = HttpStatus.NOT_FOUND)
33     public Map<String, Object> handleNoSuchElementException(
34         NoSuchElementException e, WebRequest webRequest) {
35         return createExceptionMessage(e, HttpStatus.NOT_FOUND, webRequest, LogStatus.STACK_TRACE);
36     }
37
38     private Map<String, Object> createExceptionMessage(Exception e, HttpStatus status, WebRequest webRequest, LogStatus logStatus) {
39         Map<String, Object> error = new HashMap<>();
40         String timestamp = ZonedDateTime.now().format(DateTimeFormatter.RFC_1123_DATE_TIME);
41
42         if(webRequest instanceof ServletWebRequest) {
43             error.put("uri", ((ServletWebRequest) webRequest).getRequest().getRequestURI());
44         }
45
46         error.put("message", e.toString());
47         error.put("status code", status.value());
48         error.put("timestamp", timestamp);
49         error.put("reason", status.getReasonPhrase());
50
51         if(logStatus == LogStatus.MESSAGE_ONLY) {
52             log.error("Exception: {}", e.toString());
53         }
54         else {
55             log.error("Exception: {}", e);
56         }
57
58         return error;
59     }
60 }

```

Planbook.java

```
1 package staples.heather.planbook;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7 @SpringBootApplication(scanBasePackageClasses = { ComponentScanMarker.class })
8 public class Planbook {
9
10     public static void main(String[] args) {
11         SpringApplication.run(Planbook.class, args);
12     }
13 }
```

BasicDaysLessonsController.java

```
1  package staples.heather.planbook.controller;
2
3  import java.util.List;
4
5  @RestController
6  @Slf4j
7  public class BasicDaysLessonsController implements DaysLessonsController{
8
9      @Autowired
10     private DaysLessonsService daysLessonsService;
11
12
13     public List<Lesson> fetchLessonsByDay(String dayOfTheWeek) {
14         return daysLessonsService.fetchLessonsByDay(dayOfTheWeek);
15     }
16
17     public List<Day> listWeeksLessons() {
18         return daysLessonsService.listWeeksLessons();
19     }
20
21     public void assignLessonsToDay(int dayId, int lessonId) {
22         daysLessonsService.assignLessonsToDay(dayId, lessonId);
23     }
24
25     public int updateDayOfLesson(int lessonId, int newDay) {
26         return daysLessonsService.updateDayOfLesson(lessonId, newDay);
27     }
28
29     public void clearWeeksLessons() {
30         daysLessonsService.clearWeeksLessons();
31     }
32
33     public int deleteLessonFromDay(int lessonId) {
34         return daysLessonsService.deleteLessonFromDay(lessonId);
35     }
36
37 }
```

*BasicLessonController.java

```
1 package staples.heather.planbook.controller;
2
3 import java.util.List;
4
5
6
7
8
9
10
11 @RestController
12 @Slf4j
13 public class BasicLessonController implements LessonController {
14
15     @Autowired
16     private LessonService lessonService;
17
18     @Override
19     public List<Lesson> fetchLessonById(int lessonId) {
20         return lessonService.fetchLessonById(lessonId);
21     }
22
23     @Override
24     public List<Lesson> fetchLessonsByUnit(int unitId) {
25         return lessonService.fetchLessonsByUnit(unitId);
26     }
27
28     @Override
29     public void createLesson(String lessonName, String objective, String content, int unitId) {
30         lessonService.createLesson(lessonName, objective, content, unitId);
31     }
32
33     @Override
34     public int updateLesson(int lessonId, String newContent) {
35         return lessonService.updateLesson(lessonId, newContent);
36     }
37
38     @Override
39     public int deleteLesson(int lessonId) {
40         return lessonService.deleteLesson(lessonId);
41     }
42
43
44     @Override
45     public List<Lesson> listAllLessons() {
46         return lessonService.listAllLessons();
47     }
48 }
```

BasicSubjectController.java

```
1 package staples.heather.planbook.controller;
2
3 import java.util.List;
4
14
15 @RestController
16 @Slf4j
17 public class BasicSubjectController implements SubjectController{
18
19     @Autowired
20     private SubjectService subjectService;
21
22     // @Override
23     // public List<Subject> fetchSubject(String subjectName) {
24     //     return subjectService.fetchSubject(subjectName);
25     // }
26
27     @Override
28     public List<Subject> listAllSubjects() {
29         return subjectService.listAllSubjects();
30     }
31
32     // @Override
33     // public void createSubject(String subjectName) {
34     //     subjectService.createSubject(subjectName);
35     // }
36
37     // @Override
38     // public void updateSubject(String oldName, String newName) {
39     //     // TODO Auto-generated method stub
40     //     subjectService.updateSubject(oldName, newName);
41     // }
42
43     // @Override
44     // public void deleteSubject(String subjectName) {
45     //     subjectService.deleteSubject(subjectName);
46     // }
47
48 }
```



```
*BasicUnitController.java
1 package staples.heather.planbook.controller;
2
3 import java.util.List;
4
5
6
7
8
9
10
11 @RestController
12 @Slf4j
13 public class BasicUnitController implements UnitController{
14
15     @Autowired
16     private UnitService unitService;
17
18     @Override
19     public List<Unit> fetchUnitsBySubjectByGrade(int subjectId, int gradeNumber) {
20         return unitService.fetchUnitsBySubjectByGrade(subjectId, gradeNumber);
21     }
22 //
23 //     @Override
24 //     public List<Unit> listAllUnits() {
25 //         return unitService.listAllUnits();
26 //     }
27 //
28 //     @Override
29 //     public void createUnit(String unitName, int subjectId, int gradeNumber) {
30 //         unitService.createUnit(unitName, subjectId, gradeNumber);
31 //     }
32 //
33 //     @Override
34 //     public void updateUnit(String oldName, String newName) {
35 //         unitService.updateUnit(oldName, newName);
36 //     }
37 //
38 //     public void deleteUnit(int id) {
39 //         unitService.deleteUnit(id);
40 //     }
41
42 }
```

DaysLessonsController.java

Planbook/src/main/java/staples/heather/planbook/controller/DaysLessonsController.java

```
2
3 import java.util.List;
4 import javax.validation.constraints.Pattern;
5 import org.hibernate.validator.constraints.Length;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.validation.annotation.Validated;
8 import org.springframework.web.bind.annotation.DeleteMapping;
9 import org.springframework.web.bind.annotation.GetMapping;
10 import org.springframework.web.bind.annotation.PostMapping;
11 import org.springframework.web.bind.annotation.PutMapping;
12 import org.springframework.web.bind.annotation.RequestMapping;
13 import org.springframework.web.bind.annotation.RequestParam;
14 import org.springframework.web.bind.annotation.ResponseStatus;
15 import io.swagger.v3.oas.annotations.OpenAPIDefinition;
16 import io.swagger.v3.oas.annotations.Operation;
17 import io.swagger.v3.oas.annotations.Parameter;
18 import io.swagger.v3.oas.annotations.info.Info;
19 import io.swagger.v3.oas.annotations.media.Content;
20 import io.swagger.v3.oas.annotations.media.Schema;
21 import io.swagger.v3.oas.annotations.responses.ApiResponse;
22 import io.swagger.v3.oas.annotations.servers.Server;
23 import staples.heather.planbook.Planbook;
24 import staples.heather.planbook.entity.Day;
25 import staples.heather.planbook.entity.Grade;
26 import staples.heather.planbook.entity.Lesson;
27
28 @Validated
29 @RequestMapping("/daysLessons")
30 @OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
31     @Server(url = "http://localhost:8080", description = "Local server.")})
32
33 public interface DaysLessonsController {
34
35     //@formatter:off
36     @Operation(
37         summary = "Returns a list of Lessons for the Day",
38         description = "Returns a list of Lessons for the Day",
39         responses = {
40             @ApiResponse(
41                 responseCode = "200",
42                 description = "A list of lessons was returned",
43                 content = @Content(
44                     mediaType = "application/json"
```

```

44         mediaType = "application/json",
45         schema = @Schema(implementation = Planbook.class))),
46     @ApiResponse(
47         responseCode = "400",
48         description = "The requested parameters were invalid",
49         content = @Content(mediaType = "application/json")),
50     @ApiResponse(
51         responseCode = "404",
52         description = "No lessons were found with the input criteria",
53         content = @Content(mediaType = "application/json")),
54     @ApiResponse(
55         responseCode = "500",
56         description = "An unplanned error occurred",
57         content = @Content(mediaType = "application/json"))
58 },
59 parameters = {
60     @Parameter(
61         name = "dayOfTheWeek",
62         allowEmptyValue = false,
63         required = true,
64         description = "The day of the week (i.e., 'Monday')",
65     )
66 }
67
68 @GetMapping("/{dayOfTheWeek}")
69 @ResponseStatus(code = HttpStatus.OK)
70 List<Lesson> fetchLessonsByDay(
71     @RequestParam(required = true)
72     String dayOfTheWeek);
73
74
75 @Operation(
76     summary = "Returns a list of Lessons for the Week",
77     description = "Returns a list of Lessons for the Week",
78     responses = {
79         @ApiResponse(
80             responseCode = "200",
81             description = "A list of lessons was returned",
82             content = @Content(
83                 mediaType = "application/json",
84                 schema = @Schema(implementation = Planbook.class))),
85         @ApiResponse(

```

```

86         responseCode = "400",
87         description = "The requested parameters were invalid",
88         content = @Content(mediaType = "application/json")),
89     @ApiResponse(
90         responseCode = "404",
91         description = "No lessons were found with the input criteria",
92         content = @Content(mediaType = "application/json")),
93     @ApiResponse(
94         responseCode = "500",
95         description = "An unplanned error occurred",
96         content = @Content(mediaType = "application/json"))
97     }
98
99 )
100 @ResponseStatus(code = HttpStatus.OK)
101 @GetMapping("/all")
102 List<Day> listWeeksLessons();
103
104 @Operation(
105     summary = "Assigns Lesson to Day",
106     description = "Assigns Lesson to Day",
107     responses = {
108         @ApiResponse(
109             responseCode = "201",
110             description = "A lesson was assigned to a day",
111             content = @Content(
112                 mediaType = "application/json",
113                 schema = @Schema(implementation = Planbook.class))),
114         @ApiResponse(
115             responseCode = "400",
116             description = "The requested parameters were invalid",
117             content = @Content(mediaType = "application/json")),
118         @ApiResponse(
119             responseCode = "500",
120             description = "An unplanned error occurred",
121             content = @Content(mediaType = "application/json"))
122     },
123     parameters = {
124         @Parameter(
125             name = "dayId",
126             allowEmptyValue = false,
127             required = true,

```

```

128         description = "The day id (i.e., '1')",
129     @Parameter(
130         name = "lessonId",
131         allowEmptyValue = false,
132         required = true,
133         description = "The lesson id"),
134     )
135 )
136 @ResponseStatus(code = HttpStatus.CREATED)
137 @PostMapping
138 void assignLessonsToDay(@RequestParam(required = true) int dayId, @RequestParam(required = true) int lessonId);
139
140 @Operation(
141     summary = "Updates the Day of a Lesson",
142     description = "Updates the Day of a Lesson",
143     responses = {
144         @ApiResponse(
145             responseCode = "200",
146             description = "A lesson was rescheduled for another day",
147             content = @Content(
148                 mediaType = "application/json",
149                 schema = @Schema(implementation = Planbook.class))),
150         @ApiResponse(
151             responseCode = "400",
152             description = "The requested parameters were invalid",
153             content = @Content(mediaType = "application/json")),
154         @ApiResponse(
155             responseCode = "404",
156             description = "No lessons were found with the input criteria",
157             content = @Content(mediaType = "application/json")),
158         @ApiResponse(
159             responseCode = "500",
160             description = "An unplanned error occurred",
161             content = @Content(mediaType = "application/json"))
162     },
163     parameters = {
164         @Parameter(
165             name = "lessonId",
166             allowEmptyValue = false,
167             required = true,
168             description = "The id of lesson to be rescheduled"),
169         @Parameter(

```

```

170         name = "newDay",
171         allowEmptyValue = false,
172         required = true,
173         description = "The id of the day to schedule the lesson")
174     }
175 }
176 @ResponseStatus(code = HttpStatus.OK)
177 @PutMapping
178 int updateDayOfLesson(@RequestParam(required = true) int lessonId, @RequestParam(required = true) int newDay);
179
180 @Operation(
181     summary = "Clears the Week's schedule of Lessons",
182     description = "Clears the Week's schedule of Lessons",
183     responses = {
184         @ApiResponse(
185             responseCode = "200",
186             description = "The week's lessons were cleared",
187             content = @Content(
188                 mediaType = "application/json",
189                 schema = @Schema(implementation = Planbook.class))),
190         @ApiResponse(
191             responseCode = "400",
192             description = "The requested parameters were invalid",
193             content = @Content(mediaType = "application/json")),
194         @ApiResponse(
195             responseCode = "500",
196             description = "An unplanned error occurred",
197             content = @Content(mediaType = "application/json"))
198     }
199 )
200 @ResponseStatus(code = HttpStatus.OK)
201 @DeleteMapping("/all")
202 void clearWeeksLessons();
203
204 @Operation(
205     summary = "Unschedule a Lesson",
206     description = "Unschedule a Lesson",
207     responses = {
208         @ApiResponse(
209             responseCode = "200",
210             description = "A lesson is no longer scheduled for a particular day",
211             content = @Content(

```

```

212         mediaType = "application/json",
213         schema = @Schema(implementation = Planbook.class))),
214     @ApiResponse(
215         responseCode = "400",
216         description = "The requested parameters were invalid",
217         content = @Content(mediaType = "application/json")),
218     @ApiResponse(
219         responseCode = "404",
220         description = "No lesson was found with the input criteria",
221         content = @Content(mediaType = "application/json")),
222     @ApiResponse(
223         responseCode = "500",
224         description = "An unplanned error occurred",
225         content = @Content(mediaType = "application/json"))
226 },
227 parameters = {
228     @Parameter(
229         name = "lessonId",
230         allowEmptyValue = false,
231         required = true,
232         description = "The lesson id"),
233 }
234 )
235 @ResponseStatus(code = HttpStatus.OK)
236 @DeleteMapping
237 int deleteLessonFromDay(@RequestParam(required = true) int lessonId);
238
239 // @formatter:on
240 }
241

```

LessonController.java

```
1 package staples.heather.planbook.controller;
2
3 import java.util.List;
4
5 @Validated
6 @RequestMapping("/lesson")
7 @OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
8     @Server(url = "http://localhost:8080", description = "Local server.")})
9
10 public interface LessonController {
11
12     // @formatter:off
13     @Operation(
14         summary = "Returns a Lesson",
15         description = "Returns a Lesson",
16         responses = {
17             @ApiResponse(
18                 responseCode = "200",
19                 description = "A lesson was returned",
20                 content = @Content(
21                     mediaType = "application/json",
22                     schema = @Schema(implementation = Planbook.class))),
23             // @ApiResponse(
24             //     responseCode = "400",
25             //     description = "The requested parameters were invalid",
26             //     content = @Content(mediaType = "application/json")),
27             // @ApiResponse(
28             //     responseCode = "404",
29             //     description = "No lesson was found with the input criteria",
30             //     content = @Content(mediaType = "application/json")),
31             @ApiResponse(
32                 responseCode = "500",
33                 description = "An unplanned error occurred",
34                 content = @Content(mediaType = "application/json"))
35         },
36         parameters = {
37             @Parameter(
38                 name = "lessonId",
39                 allowEmptyValue = false,
40                 required = true,
41                 description = "The lesson id"),
42         }
43     )
44 }
```



```

68     )
69
70     @GetMapping("/{lessonId}")
71     @ResponseStatus(code = HttpStatus.OK)
72     List<Lesson> fetchLessonById(
73         @RequestParam(required = true)
74         int id);
75
76     @Operation(
77         summary = "Returns a List of Lessons by Unit",
78         description = "Returns a List of Lessons by Unit",
79         responses = {
80             @ApiResponse(
81                 responseCode = "200",
82                 description = "A list of unit lessons was returned",
83                 content = @Content(
84                     mediaType = "application/json",
85                     schema = @Schema(implementation = Planbook.class)),
86             //
87             @ApiResponse(
88                 responseCode = "404",
89                 description = "No lessons were found with the input criteria",
90                 content = @Content(mediaType = "application/json")),
91             @ApiResponse(
92                 responseCode = "500",
93                 description = "An unplanned error occurred",
94                 content = @Content(mediaType = "application/json"))
95         },
96         parameters = {
97             @Parameter(
98                 name = "unitId",
99                 allowEmptyValue = false,
100                 required = true,
101                 description = "The unit id"),
102         }
103     )
104
105
106     @GetMapping("/{unitId}")
107     @ResponseStatus(code = HttpStatus.OK)
108     List<Lesson> fetchLessonsByUnit(
109         @RequestParam(required = true) int unitId);

```

```

110
111
112
113 @Operation(
114     summary = "Returns a list of Lessons",
115     description = "Returns a list of Lessons",
116     responses = {
117         @ApiResponse(
118             responseCode = "200",
119             description = "A list of lessons is returned",
120             content = @Content(
121                 mediaType = "application/json",
122                 schema = @Schema(implementation = Planbook.class))),
123         // @ApiResponse(
124         //     responseCode = "400",
125         //     description = "The requested parameters were invalid",
126         //     content = @Content(mediaType = "application/json")),
127         // @ApiResponse(
128         //     responseCode = "404",
129         //     description = "No lessons were found with the input criteria",
130         //     content = @Content(mediaType = "application/json")),
131         @ApiResponse(
132             responseCode = "500",
133             description = "An unplanned error occurred",
134             content = @Content(mediaType = "application/json"))
135     }
136 )
137
138 @GetMapping("/all")
139 @ResponseStatus(code = HttpStatus.OK)
140 List<Lesson> listAllLessons();
141
142
143 @Operation(
144     summary = "Creates a Lesson",
145     description = "Creates a Lesson",
146     responses = {
147         @ApiResponse(
148             responseCode = "201",
149             description = "A lesson was created",
150             content = @Content(
151                 mediaType = "application/json",

```

```

152         schema = @Schema(implementation = Planbook.class)),
153     @ApiResponse(
154         responseCode = "400",
155         description = "The requested parameters were invalid",
156         content = @Content(mediaType = "application/json")),
157     @ApiResponse(
158         responseCode = "500",
159         description = "An unplanned error occurred",
160         content = @Content(mediaType = "application/json"))
161 },
162 parameters = {
163     @Parameter(
164         name = "lessonName",
165         allowEmptyValue = false,
166         required = true,
167         description = "The lesson name"),
168     @Parameter(
169         name = "objective",
170         allowEmptyValue = false,
171         required = true,
172         description = "The lesson objective"),
173     @Parameter(
174         name = "content",
175         allowEmptyValue = false,
176         required = true,
177         description = "The lesson content"),
178     @Parameter(
179         name = "unitId",
180         allowEmptyValue = false,
181         required = true,
182         description = "The unit id")
183 }
184 )
185 @ResponseStatus(code = HttpStatus.CREATED)
186 @PostMapping
187 void createLesson(@RequestParam(required = true) String lessonName, @RequestParam(required = true) String objective,
188     @RequestParam(required = true) String content, @RequestParam(required = true) int unitId);
189
190 @Operation(
191     summary = "Updates Lesson Content",
192     description = "Updates Lesson Content",
193     responses = {

```

```

194     @ApiResponse(
195         responseCode = "200",
196         description = "A lesson's content was updated",
197         content = @Content(
198             mediaType = "application/json",
199             schema = @Schema(implementation = Planbook.class)),
200     @ApiResponse(
201         responseCode = "400",
202         description = "The requested parameters were invalid",
203         content = @Content(mediaType = "application/json")),
204     @ApiResponse(
205         responseCode = "404",
206         description = "No lesson was found with the input criteria",
207         content = @Content(mediaType = "application/json")),
208     @ApiResponse(
209         responseCode = "500",
210         description = "An unplanned error occurred",
211         content = @Content(mediaType = "application/json"))
212 },
213 parameters = {
214     @Parameter(
215         name = "lessonId",
216         allowEmptyValue = false,
217         required = true,
218         description = "The lesson id"),
219     @Parameter(
220         name = "newContent",
221         allowEmptyValue = false,
222         required = true,
223         description = "The new lesson content")
224 }
225 )
226 @ResponseStatus(code = HttpStatus.OK)
227 @PutMapping
228 int updateLesson(@RequestParam(required = true) int lessonId, @RequestParam(required = true) String newContent);
229
230 @Operation(
231     summary = "Deletes a Lesson",
232     description = "Deletes a Lesson",
233     responses = {
234         @ApiResponse(
235             responseCode = "200",

```

```
236         description = "A lesson was deleted",
237         content = @Content(
238             mediaType = "application/json",
239             schema = @Schema(implementation = Planbook.class)),
240     @ApiResponse(
241         responseCode = "400",
242         description = "The requested parameters were invalid",
243         content = @Content(mediaType = "application/json")),
244     @ApiResponse(
245         responseCode = "404",
246         description = "No lesson was found with the input criteria",
247         content = @Content(mediaType = "application/json")),
248     @ApiResponse(
249         responseCode = "500",
250         description = "An unplanned error occurred",
251         content = @Content(mediaType = "application/json"))
252 },
253 parameters = {
254     @Parameter(
255         name = "lessonId",
256         allowEmptyValue = false,
257         required = true,
258         description = "The lesson id"),
259 }
260 )
261 @ResponseStatus(code = HttpStatus.OK)
262 @DeleteMapping
263 int deleteLesson(@RequestParam(required = true) int lessonId);
264
265 }
266
267
268 //@formatter:on
269
270
```

SubjectController.java

Maximize

```
package staples.heather.planbook.controller;
```

```
2
3+ import java.util.List;
26
27 @Validated
28 @RequestMapping("/subject")
29 @OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
30     @Server(url = "http://localhost:8080", description = "Local server.")})
31
32 public interface SubjectController {
33     public static final int NAME_MAX_LENGTH = 25;
34
35     //@formatter:off
36
37     @Operation(
38         summary = "Returns a list of Subjects",
39         description = "Returns a list of Subjects",
40         responses = {
41             @ApiResponse(
42                 responseCode = "200",
43                 description = "A list of subjects was returned",
44                 content = @Content(
45                     mediaType = "application/json",
46                     schema = @Schema(implementation = Planbook.class))),
47             @ApiResponse(
48                 responseCode = "400",
49                 description = "The requested parameters were invalid",
50                 content = @Content(mediaType = "application/json")),
51             @ApiResponse(
52                 responseCode = "404",
53                 description = "No subjects were found with the input criteria",
54                 content = @Content(mediaType = "application/json")),
55             @ApiResponse(
56                 responseCode = "500",
57                 description = "An unplanned error occurred",
58                 content = @Content(mediaType = "application/json"))
59         }
60     )
61
62     @ResponseStatus(code = HttpStatus.OK)
63     @GetMapping("/all")
64     List<Subject> listAllSubjects();
```

```
66     //@formatter:on
67 }
68
```

```

1 package staples.heather.planbook.controller;
2
3 import java.util.List;
4
5 @Validated
6 @RequestMapping("/unit")
7 @OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
8     @Server(url = "http://localhost:8080", description = "Local server.")})
9
10 public interface UnitController {
11
12     // @formatter:off
13     @Operation(
14         summary = "Returns a List of Units by Subject by Grade",
15         description = "Returns a List of Units by Subject by Grade",
16         responses = {
17             @ApiResponse(
18                 responseCode = "200",
19                 description = "A list of units was returned",
20                 content = @Content(
21                     mediaType = "application/json",
22                     schema = @Schema(implementation = Planbook.class))),
23             @ApiResponse(
24                 responseCode = "400",
25                 description = "The requested parameters were invalid",
26                 content = @Content(mediaType = "application/json")),
27             @ApiResponse(
28                 responseCode = "404",
29                 description = "No unit was found with the input criteria",
30                 content = @Content(mediaType = "application/json")),
31             @ApiResponse(
32                 responseCode = "500",
33                 description = "An unplanned error occurred",
34                 content = @Content(mediaType = "application/json"))
35         },
36         parameters = {
37             @Parameter(
38                 name = "subjectId",
39                 allowEmptyValue = false,
40                 required = true,
41                 description = "The subject id"),
42             @Parameter(
43                 name = "gradeNumber",
44                 allowEmptyValue = false,
45                 required = true,
46                 description = "The grade number"),
47         }
48     )
49     @GetMapping
50     @ResponseStatus(code = HttpStatus.OK)
51     List<Unit> fetchUnitsBySubjectByGrade(@RequestParam(required = true) int subjectId, @RequestParam(required = true) int gradeNumber);
52
53     // @formatter:on
54 }

```

DaysLessonsDao.java

```
1 package staples.heather.planbook.dao;
2
3 import java.util.List;
4
5
6
7
8
9 public interface DaysLessonsDao {
10
11     List<Day> listWeeksLessons();
12
13     List<Lesson> fetchLessonsByDay(String dayOfTheWeek);
14
15     void assignLessonsToDay(int dayId, int lessonId);
16
17     void clearWeeksLessons();
18
19     int deleteLessonFromDay(int lessonId);
20
21     int updateDayOfLesson(int lessonId, int newDay);
22
23     //void deleteLessonByDay(int dayId);
24 }
25
```



```

1 package staples.heather.planbook.dao;
2
3 import java.sql.ResultSet;
4
5
6
7
8
9
10
11
12
13
14
15
16
17 @Component
18 @Slf4j
19 public class DefaultDaysLessonsDao implements DaysLessonsDao {
20
21     @Autowired
22     private NamedParameterJdbcTemplate jdbcTemplate;
23
24     //displays lessons for the week - READ
25     @Override
26     public List<Day> listWeeksLessons() {
27
28         String sqlFetch = "SELECT * from day inner join daysLessons on day.dayId = daysLessons.dayId "
29             + "INNER JOIN lesson on lesson.lessonId = daysLessons.lessonId group by day.dayId order by day.dayId";
30
31         Map<String,Object> params = new HashMap<>();
32
33         return jdbcTemplate.query(sqlFetch, params, new RowMapper<>() {
34
35             @Override
36             public Day mapRow(ResultSet rs, int rowNum) throws SQLException {
37                 return Day.builder()
38                     .dayId(rs.getInt("dayId"))
39                     .dayOfTheWeek(rs.getString("dayOfTheWeek"))
40                     .lessons(fetchLessonsByDay(rs.getString("dayOfTheWeek")))
41                     .build();
42             }
43         });
44
45         //displays lessons for the day - READ
46         @Override
47         public List<Lesson> fetchLessonsByDay(String dayOfTheWeek) {
48             String sqlFetch = "SELECT * FROM lesson INNER JOIN daysLessons on lesson.lessonId = daysLessons.lessonId "
49                 + "INNER JOIN day on day.dayId = daysLessons.dayId WHERE dayOfTheWeek = :dayOfTheWeek";
50
51             Map<String,Object> params = new HashMap<>();
52             params.put("dayOfTheWeek", dayOfTheWeek);
53
54             return jdbcTemplate.query(sqlFetch, params, new RowMapper<>() {
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

55
56 @Override
57 public Lesson mapRow(ResultSet rs, int rowNum) throws SQLException {
58     return Lesson.builder()
59         .lessonId(rs.getInt("lessonId"))
60         .lessonName(rs.getString("lessonName"))
61         .objective(rs.getString("objective"))
62         .content(rs.getString("content"))
63         .unitId(rs.getInt("unitId"))
64         .build();
65     });
66 }
67
68 //disassociates lessons from days of the week - DELETE
69 @Override
70 public void clearWeeksLessons() {
71     String sql = "DELETE FROM daysLessons";
72
73     Map<String, Object> params = new HashMap<>();
74     jdbcTemplate.update(sql, params);
75 }
76
77 //unassigns lesson from day - DELETE
78 @Override
79 public int deleteLessonFromDay(int lessonId) {
80     String sql = "DELETE FROM daysLessons WHERE lessonId = :lessonId";
81
82     Map<String, Object> params = new HashMap<>();
83     params.put("lessonId", lessonId);
84     if (jdbcTemplate.update(sql, params) == 0) {
85         throw new NoSuchElementException();
86     };
87     return jdbcTemplate.update(sql, params);
88 }
89
90 //assigns lessons to day of the week for planbook - CREATE
91 @Override
92 public void assignLessonsToDay(int dayId, int lessonId) {
93
94     String sqlCreate = "INSERT INTO daysLessons (dayId, lessonId) VALUES (:dayId, :lessonId)";
95
96     Map<String, Object> params = new HashMap<>();
97     params.put("dayId", dayId);

```

```

99
100 jdbcTemplate.update(sqlCreate, params);
101 }
102
103 //change lesson day - UPDATE
104 @Override
105 public int updateDayOfLesson(int lessonId, int newDay) {
106
107     String sqlUpdate = "UPDATE daysLessons SET dayId = :newDay WHERE lessonId = :lessonId";
108
109     Map<String, Object> params = new HashMap<>();
110     params.put("newDay", newDay);
111     params.put("lessonId", lessonId);
112
113     if (jdbcTemplate.update(sqlUpdate, params) == 0) {
114         throw new NoSuchElementException();
115     };
116     return jdbcTemplate.update(sqlUpdate, params);
117 }
118
119
120 }
121

```

BasicSubjectController.java

```
1 package staples.heather.planbook.controller;
2
3
14
15
16
17
18
19 @Autowired
20 private SubjectService subjectService;
21
22 // @Override
23 // public List<Subject> fetchSubject(String subjectName) {
24 //     return subjectService.fetchSubject(subjectName);
25 // }
26 // @Override
27 public List<Subject> listAllSubjects() {
28     return subjectService.listAllSubjects();
29 }
30
31 // @Override
32 // public void createSubject(String subjectName) {
33 //     subjectService.createSubject(subjectName);
34 // }
35
36 // @Override
37 // public void updateSubject(String oldName, String newName) {
38 //     // TODO Auto-generated method stub
39 //     subjectService.updateSubject(oldName, newName);
40 // }
41
42 // @Override
43 // public void deleteSubject(String subjectName) {
44 //     subjectService.deleteSubject(subjectName);
45 // }
46 // }
47
48 }
```

Multiple markers at this line

- The import staples.heather.planbook.entity.Day is never used
- The import javax.validation.constraints.Pattern is never used
- The import staples.heather.planbook.service.LessonService is never used
- The import staples.heather.planbook.entity.Lesson is never used
- The import org.hibernate.validator.constraints.Length is never used

Controller{

```
BasicUnitController.java
1 package staples.heather.planbook.controller;
2
3 import java.util.List;
4
5
6
7
8
9
10
11 @RestController
12 @Slf4j
13 public class BasicUnitController implements UnitController{
14
15     @Autowired
16     private UnitService unitService;
17
18     @Override
19     public List<Unit> fetchUnitsBySubjectByGrade(int subjectId, int gradeNumber) {
20         return unitService.fetchUnitsBySubjectByGrade(subjectId, gradeNumber);
21     }
22 //
23 //     @Override
24 //     public List<Unit> listAllUnits() {
25 //         return unitService.listAllUnits();
26 //     }
27 //
28 //     @Override
29 //     public void createUnit(String unitName, int subjectId, int gradeNumber) {
30 //         unitService.createUnit(unitName, subjectId, gradeNumber);
31 //     }
32 //
33 //     @Override
34 //     public void updateUnit(String oldName, String newName) {
35 //         unitService.updateUnit(oldName, newName);
36 //     }
37 //
38 //     public void deleteUnit(int id) {
39 //         unitService.deleteUnit(id);
40 //     }
41
42 }
```

DaysLessonsController.java

```
1 package staples.heather.planbook.controller;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28 @Validated
29 @RequestMapping("/daysLessons")
30 @OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
31     @Server(url = "http://localhost:8080", description = "Local server.")})
32
33 public interface DaysLessonsController {
34
35     // @formatter:off
36     @Operation(
37         summary = "Returns a list of Lessons for the Day",
38         description = "Returns a list of Lessons for the Day",
39         responses = {
40             @ApiResponse(
41                 responseCode = "200",
42                 description = "A list of lessons was returned",
43                 content = @Content(
44                     mediaType = "application/json",
45                     schema = @Schema(implementation = Planbook.class))),
46             @ApiResponse(
47                 responseCode = "400",
48                 description = "The requested parameters were invalid",
49                 content = @Content(mediaType = "application/json")),
50             @ApiResponse(
51                 responseCode = "404",
52                 description = "No lessons were found with the input criteria",
53                 content = @Content(mediaType = "application/json")),
54             @ApiResponse(
55                 responseCode = "500",
56                 description = "An unplanned error occurred",
57                 content = @Content(mediaType = "application/json"))
58         },
59         parameters = {
60             @Parameter(
61                 name = "dayOfTheWeek",
62                 allowEmptyValue = false,
63                 required = true,
64                 description = "The day of the week (i.e., 'Monday')"),
65         }
66     )
67 }
```

```

67
68 @GetMapping("/{dayOfTheWeek}")
69 @ResponseStatus(code = HttpStatus.OK)
70 List<Lesson> fetchLessonsByDay(
71     @RequestParam(required = true)
72     String dayOfTheWeek);
73
74
75 @Operation(
76     summary = "Returns a list of Lessons for the Week",
77     description = "Returns a list of Lessons for the Week",
78     responses = {
79         @ApiResponse(
80             responseCode = "200",
81             description = "A list of lessons was returned",
82             content = @Content(
83                 mediaType = "application/json",
84                 schema = @Schema(implementation = Planbook.class))),
85         @ApiResponse(
86             responseCode = "400",
87             description = "The requested parameters were invalid",
88             content = @Content(mediaType = "application/json")),
89         @ApiResponse(
90             responseCode = "404",
91             description = "No lessons were found with the input criteria",
92             content = @Content(mediaType = "application/json")),
93         @ApiResponse(
94             responseCode = "500",
95             description = "An unplanned error occurred",
96             content = @Content(mediaType = "application/json"))
97     }
98 )
99
100 @ResponseStatus(code = HttpStatus.OK)
101 @GetMapping("/all")
102 List<Day> listWeeksLessons();
103
104 @Operation(
105     summary = "Assigns Lesson to Day",
106     description = "Assigns Lesson to Day",
107     responses = {
108         @ApiResponse(

```

```

109         responseCode = "201",
110         description = "A lesson was assigned to a day",
111         content = @Content(
112             mediaType = "application/json",
113             schema = @Schema(implementation = Planbook.class)),
114     @ApiResponse(
115         responseCode = "400",
116         description = "The requested parameters were invalid",
117         content = @Content(mediaType = "application/json")),
118     @ApiResponse(
119         responseCode = "500",
120         description = "An unplanned error occurred",
121         content = @Content(mediaType = "application/json"))
122 },
123 parameters = {
124     @Parameter(
125         name = "dayId",
126         allowEmptyValue = false,
127         required = true,
128         description = "The day id (i.e., '1')"),
129     @Parameter(
130         name = "lessonId",
131         allowEmptyValue = false,
132         required = true,
133         description = "The lesson id"),
134 }
135 )
136 @ResponseStatus(code = HttpStatus.CREATED)
137 @PostMapping
138 void assignLessonsToDay(@RequestParam(required = true) int dayId, @RequestParam(required = true) int lessonId);
139
140 @Operation(
141     summary = "Updates the Day of a Lesson",
142     description = "Updates the Day of a Lesson",
143     responses = {
144         @ApiResponse(
145             responseCode = "200",
146             description = "A lesson was rescheduled for another day",
147             content = @Content(
148                 mediaType = "application/json",
149                 schema = @Schema(implementation = Planbook.class))),
150         @ApiResponse(

```



```

151         responseCode = "400",
152         description = "The requested parameters were invalid",
153         content = @Content(mediaType = "application/json")),
154     @ApiResponse(
155         responseCode = "404",
156         description = "No lessons were found with the input criteria",
157         content = @Content(mediaType = "application/json")),
158     @ApiResponse(
159         responseCode = "500",
160         description = "An unplanned error occurred",
161         content = @Content(mediaType = "application/json"))
162 },
163 parameters = {
164     @Parameter(
165         name = "lessonId",
166         allowEmptyValue = false,
167         required = true,
168         description = "The id of lesson to be rescheduled"),
169     @Parameter(
170         name = "newDay",
171         allowEmptyValue = false,
172         required = true,
173         description = "The id of the day to schedule the lesson")
174 }
175 )
176 @ResponseStatus(code = HttpStatus.OK)
177 @PostMapping
178 int updateDayOfLesson(@RequestParam(required = true) int lessonId, @RequestParam(required = true) int newDay);
179
180* @Operation(
181     summary = "Clears the Week's schedule of Lessons",
182     description = "Clears the Week's schedule of Lessons",
183     responses = {
184         @ApiResponse(
185             responseCode = "200",
186             description = "The week's lessons were cleared",
187             content = @Content(
188                 mediaType = "application/json",
189                 schema = @Schema(implementation = Planbook.class))),
190         @ApiResponse(
191             responseCode = "400",
192             description = "The requested parameters were invalid",

```

```

193         content = @Content(mediaType = "application/json")),
194     @ApiResponse(
195         responseCode = "500",
196         description = "An unplanned error occurred",
197         content = @Content(mediaType = "application/json"))
198     }
199 )
200 @ResponseStatus(code = HttpStatus.OK)
201 @DeleteMapping("/all")
202 void clearWeeksLessons();
203
204 @Operation(
205     summary = "Unschedulates a Lesson",
206     description = "Unschedulates a Lesson",
207     responses = {
208         @ApiResponse(
209             responseCode = "200",
210             description = "A lesson is no longer scheduled for a particular day",
211             content = @Content(
212                 mediaType = "application/json",
213                 schema = @Schema(implementation = Planbook.class))),
214         @ApiResponse(
215             responseCode = "400",
216             description = "The requested parameters were invalid",
217             content = @Content(mediaType = "application/json")),
218         @ApiResponse(
219             responseCode = "404",
220             description = "No lesson was found with the input criteria",
221             content = @Content(mediaType = "application/json")),
222         @ApiResponse(
223             responseCode = "500",
224             description = "An unplanned error occurred",
225             content = @Content(mediaType = "application/json"))
226     },
227     parameters = {
228         @Parameter(
229             name = "lessonId",
230             allowEmptyValue = false,
231             required = true,
232             description = "The lesson id"),
233     }
234 )
235
236 @ResponseStatus(code = HttpStatus.OK)
237 @DeleteMapping
238 int deleteLessonFromDay(@RequestParam(required = true) int lessonId);
239
240 // @formatter:on
241 }
242

```

```

1 package staples.heather.planbook.controller;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19 @Validated
20 @RequestMapping("/lesson")
21 @OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
22     @Server(url = "http://localhost:8080", description = "Local server.")})
23
24 public interface LessonController {
25
26     // @formatter:off
27     @Operation(
28         summary = "Returns a Lesson",
29         description = "Returns a Lesson",
30         responses = {
31             @ApiResponse(
32                 responseCode = "200",
33                 description = "A lesson was returned",
34                 content = @Content(
35                     mediaType = "application/json",
36                     schema = @Schema(implementation = Planbook.class)),
37             @ApiResponse(
38                 responseCode = "404",
39                 description = "No lesson was found with the input criteria",
40                 content = @Content(mediaType = "application/json")),
41             @ApiResponse(
42                 responseCode = "500",
43                 description = "An unplanned error occurred",
44                 content = @Content(mediaType = "application/json"))
45         },
46         parameters = {
47             @Parameter(
48                 name = "lessonId",
49                 allowEmptyValue = false,
50                 required = true,
51                 description = "The lesson id"),
52         }
53     )
54
55     @GetMapping("/{lessonId}")
56     @ResponseStatus(code = HttpStatus.OK)
57     List<Lesson> fetchLessonById(

```

```

68     @RequestParam(required = true)
69     int id);
70
71     @Operation(
72         summary = "Returns a List of Lessons by Unit",
73         description = "Returns a List of Lessons by Unit",
74         responses = {
75             @ApiResponse(
76                 responseCode = "200",
77                 description = "A list of unit lessons was returned",
78                 content = @Content(
79                     mediaType = "application/json",
80                     schema = @Schema(implementation = Planbook.class))),
81             //
82             @ApiResponse(
83                 responseCode = "404",
84                 description = "No lessons were found with the input criteria",
85                 content = @Content(mediaType = "application/json")),
86             @ApiResponse(
87                 responseCode = "500",
88                 description = "An unplanned error occurred",
89                 content = @Content(mediaType = "application/json"))
90         },
91         parameters = {
92             @Parameter(
93                 name = "unitId",
94                 allowEmptyValue = false,
95                 required = true,
96                 description = "The unit id"),
97         }
98     )
99
100
101     @GetMapping("/{unitId}")
102     @ResponseStatus(code = HttpStatus.OK)
103     List<Lesson> fetchLessonsByUnit(
104         @RequestParam(required = true) int unitId);
105
106
107
108     @Operation(
109         summary = "Returns a list of Lessons",

```

```

110     description = "Returns a list of Lessons",
111     responses = {
112         @ApiResponse(
113             responseCode = "200",
114             description = "A list of lessons is returned",
115             content = @Content(
116                 mediaType = "application/json",
117                 schema = @Schema(implementation = Planbook.class))),
118         @ApiResponse(
119             responseCode = "500",
120             description = "An unplanned error occurred",
121             content = @Content(mediaType = "application/json"))
122     }
123 )
124
125 @GetMapping("/all")
126 @ResponseStatus(code = HttpStatus.OK)
127 List<Lesson> listAllLessons();
128
129
130 @Operation(
131     summary = "Creates a Lesson",
132     description = "Creates a Lesson",
133     responses = {
134         @ApiResponse(
135             responseCode = "201",
136             description = "A lesson was created",
137             content = @Content(
138                 mediaType = "application/json",
139                 schema = @Schema(implementation = Planbook.class))),
140         @ApiResponse(
141             responseCode = "400",
142             description = "The requested parameters were invalid",
143             content = @Content(mediaType = "application/json")),
144         @ApiResponse(
145             responseCode = "500",
146             description = "An unplanned error occurred",
147             content = @Content(mediaType = "application/json"))
148     },
149     parameters = {
150         @Parameter(
151             name = "lessonName",

```

```

152         allowEmptyValue = false,
153         required = true,
154         description = "The lesson name"),
155     @Parameter(
156         name = "objective",
157         allowEmptyValue = false,
158         required = true,
159         description = "The lesson objective"),
160     @Parameter(
161         name = "content",
162         allowEmptyValue = false,
163         required = true,
164         description = "The lesson content"),
165     @Parameter(
166         name = "unitId",
167         allowEmptyValue = false,
168         required = true,
169         description = "The unit id")
170 }
171 }
172 @ResponseStatus(code = HttpStatus.CREATED)
173 @PostMapping
174 void createLesson(@RequestParam(required = true) String lessonName, @RequestParam(required = true) String objective,
175     @RequestParam(required = true) String content, @RequestParam(required = true) int unitId);
176
177 @Operation(
178     summary = "Updates Lesson Content",
179     description = "Updates Lesson Content",
180     responses = {
181         @ApiResponse(
182             responseCode = "200",
183             description = "A lesson's content was updated",
184             content = @Content(
185                 mediaType = "application/json",
186                 schema = @Schema(implementation = Planbook.class))),
187         @ApiResponse(
188             responseCode = "400",
189             description = "The requested parameters were invalid",
190             content = @Content(mediaType = "application/json")),
191         @ApiResponse(
192             responseCode = "404",
193             description = "No lesson was found with the input criteria",

```

```

194         content = @Content(mediaType = "application/json")),
195     @ApiResponse(
196         responseCode = "500",
197         description = "An unplanned error occurred",
198         content = @Content(mediaType = "application/json"))
199 },
200 parameters = {
201     @Parameter(
202         name = "lessonId",
203         allowEmptyValue = false,
204         required = true,
205         description = "The lesson id"),
206     @Parameter(
207         name = "newContent",
208         allowEmptyValue = false,
209         required = true,
210         description = "The new lesson content")
211 }
212 )
213 @ResponseStatus(code = HttpStatus.OK)
214 @PutMapping
215 int updateLesson(@RequestParam(required = true) int lessonId, @RequestParam(required = true) String newContent);
216
217 @Operation(
218     summary = "Deletes a Lesson",
219     description = "Deletes a lesson",
220     responses = {
221         @ApiResponse(
222             responseCode = "200",
223             description = "A lesson was deleted",
224             content = @Content(
225                 mediaType = "application/json",
226                 schema = @Schema(implementation = Planbook.class))),
227         @ApiResponse(
228             responseCode = "400",
229             description = "The requested parameters were invalid",
230             content = @Content(mediaType = "application/json")),
231         @ApiResponse(
232             responseCode = "404",
233             description = "No lesson was found with the input criteria",
234             content = @Content(mediaType = "application/json")),
235         @ApiResponse(

```

```
236         responseCode = "500",
237         description = "An unplanned error occurred",
238         content = @Content(mediaType = "application/json"))
239     },
240     parameters = {
241         @Parameter(
242             name = "lessonId",
243             allowEmptyValue = false,
244             required = true,
245             description = "The lesson id"),
246     }
247 )
248 @ResponseStatus(code = HttpStatus.OK)
249 @DeleteMapping
250 int deleteLesson(@RequestParam(required = true) int lessonId);
251
252 }
253
254
255 // @formatter:on
256
257
```



```

1 package staples.heather.planbook.controller;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27 @Validated
28 @RequestMapping("/subject")
29 @OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
30     @Server(url = "http://localhost:8080", description = "Local server.")})
31
32 public interface SubjectController {
33
34     // @formatter:off
35     @Operation(
36         summary = "Returns a list of Subjects",
37         description = "Returns a list of Subjects",
38         responses = {
39             @ApiResponse(
40                 responseCode = "200",
41                 description = "A list of subjects was returned",
42                 content = @Content(
43                     mediaType = "application/json",
44                     schema = @Schema(implementation = Planbook.class))),
45             @ApiResponse(
46                 responseCode = "400",
47                 description = "The requested parameters were invalid",
48                 content = @Content(mediaType = "application/json")),
49             @ApiResponse(
50                 responseCode = "404",
51                 description = "No subjects were found with the input criteria",
52                 content = @Content(mediaType = "application/json")),
53             @ApiResponse(
54                 responseCode = "500",
55                 description = "An unplanned error occurred",
56                 content = @Content(mediaType = "application/json"))
57         }
58     )
59
60     @ResponseStatus(code = HttpStatus.OK)
61     @GetMapping("/all")
62     List<Subject> listAllSubjects();
63
64     // @formatter:on
65
66

```

UnitController.java

```
1 package staples.heather.planbook.controller;
2
3 import java.util.List;
4
5 @Validated
6 @RequestMapping("/unit")
7 @OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
8     @Server(url = "http://localhost:8080", description = "Local server.")})
9
10 public interface UnitController {
11
12     // @formatter:off
13     @Operation(
14         summary = "Returns a List of Units by Subject by Grade",
15         description = "Returns a List of Units by Subject by Grade",
16         responses = {
17             @ApiResponse(
18                 responseCode = "200",
19                 description = "A list of units was returned",
20                 content = @Content(
21                     mediaType = "application/json",
22                     schema = @Schema(implementation = Planbook.class)),
23             @ApiResponse(
24                 responseCode = "400",
25                 description = "The requested parameters were invalid",
26                 content = @Content(mediaType = "application/json")),
27             @ApiResponse(
28                 responseCode = "404",
29                 description = "No unit was found with the input criteria",
30                 content = @Content(mediaType = "application/json")),
31             @ApiResponse(
32                 responseCode = "500",
33                 description = "An unplanned error occurred",
34                 content = @Content(mediaType = "application/json"))
35         },
36         parameters = {
37             @Parameter(
38                 name = "subjectId",
39                 allowEmptyValue = false,
40                 required = true,
41                 description = "The subject id"),
42             @Parameter(
43                 name = "gradeNumber",
44                 allowEmptyValue = false,
45                 required = true,
46                 description = "The grade number")
47         }
48     )
49 }
```

```

63         allowEmptyValue = false,
64         required = true,
65         description = "The grade number",
66     }
67 }
68
69 @GetMapping
70 @ResponseStatus(code = HttpStatus.OK)
71 List<Unit> fetchUnitsBySubjectByGrade(@RequestParam(required = true) int subjectId, @RequestParam(required = true) int gradeNumber);
72
73 // @formatter:on
74 }

```

DaysLessonsDao.java

```

1 package staples.heather.planbook.dao;
2
3 import java.util.List;
4
5
6
7
8
9 public interface DaysLessonsDao {
10
11     List<Day> listWeeksLessons();
12
13     List<Lesson> fetchLessonsByDay(String dayOfTheWeek);
14
15     void assignLessonsToDay(int dayId, int lessonId);
16
17     void clearWeeksLessons();
18
19     int deleteLessonFromDay(int lessonId);
20
21     int updateDayOfLesson(int lessonId, int newDay);
22
23     //void deleteLessonByDay(int dayId);
24 }
25

```

```

DefaultDaysLessonsDao.java
1 package staples.heather.planbook.dao;
2
3 import java.sql.ResultSet;
4
5
6
7 @Component
8 @Slf4j
9 public class DefaultDaysLessonsDao implements DaysLessonsDao {
10
11     @Autowired
12     private NamedParameterJdbcTemplate jdbcTemplate;
13
14     //displays lessons for the week - READ
15     @Override
16     public List<Day> listWeeksLessons() {
17
18         String sqlFetch = "SELECT * from day inner join daysLessons on day.dayId = daysLessons.dayId "
19             + "INNER JOIN lesson on lesson.lessonId = daysLessons.lessonId group by day.dayId order by day.dayId";
20
21         Map<String, Object> params = new HashMap<>();
22
23         return jdbcTemplate.query(sqlFetch, params, new RowMapper<>() {
24
25             @Override
26             public Day mapRow(ResultSet rs, int rowNum) throws SQLException {
27                 return Day.builder()
28                     .dayId(rs.getInt("dayId"))
29                     .dayOfTheWeek(rs.getString("dayOfTheWeek"))
30                     .lessons(fetchLessonsByDay(rs.getString("dayOfTheWeek")))
31                     .build();
32             }
33         });
34     }
35
36     //displays lessons for the day - READ
37     @Override
38     public List<Lesson> fetchLessonsByDay(String dayOfTheWeek) {
39         String sqlFetch = "SELECT * FROM lesson INNER JOIN daysLessons on lesson.lessonId = daysLessons.lessonId "
40             + "INNER JOIN day on day.dayId = daysLessons.dayId WHERE dayOfTheWeek = :dayOfTheWeek";
41
42         Map<String, Object> params = new HashMap<>();
43         params.put("dayOfTheWeek", dayOfTheWeek);
44
45         return jdbcTemplate.query(sqlFetch, params, new RowMapper<>() {
46
47
48
49
50
51
52
53
54
55

```

```

55
56  @Override
57  public Lesson mapRow(ResultSet rs, int rowNum) throws SQLException {
58      return Lesson.builder()
59          .lessonId(rs.getInt("lessonId"))
60          .lessonName(rs.getString("lessonName"))
61          .objective(rs.getString("objective"))
62          .content(rs.getString("content"))
63          .unitId(rs.getInt("unitId"))
64          .build();
65      });
66  }
67
68  //disassociates lessons from days of the week - DELETE
69  @Override
70  public void clearWeeksLessons() {
71      String sql = "DELETE FROM daysLessons";
72
73      Map<String, Object> params = new HashMap<>();
74      jdbcTemplate.update(sql, params);
75  }
76
77  //unassigns lesson from day - DELETE
78  @Override
79  public int deleteLessonFromDay(int lessonId) {
80      String sql = "DELETE FROM daysLessons WHERE lessonId = :lessonId";
81
82      Map<String, Object> params = new HashMap<>();
83      params.put("lessonId", lessonId);
84      if (jdbcTemplate.update(sql, params) == 0) {
85          throw new NoSuchElementException();
86      };
87      return jdbcTemplate.update(sql, params);
88  }
89
90  //assigns lessons to day of the week for planbook - CREATE
91  @Override
92  public void assignLessonsToDay(int dayId, int lessonId) {
93
94      String sqlCreate = "INSERT INTO daysLessons (dayId, lessonId) VALUES (:dayId, :lessonId)";
95
96      Map<String, Object> params = new HashMap<>();
97      params.put("dayId", dayId);

```

```

98     params.put("lessonId", lessonId);
99
100     jdbcTemplate.update(sqlCreate, params);
101 }
102
103 //change lesson day - UPDATE
104 @Override
105 public int updateDayOfLesson(int lessonId, int newDay) {
106
107     String sqlUpdate = "UPDATE daysLessons SET dayId = :newDay WHERE lessonId = :lessonId";
108
109     Map<String, Object> params = new HashMap<>();
110     params.put("newDay", newDay);
111     params.put("lessonId", lessonId);
112
113     if (jdbcTemplate.update(sqlUpdate, params) == 0) {
114         throw new NoSuchElementException();
115     };
116     return jdbcTemplate.update(sqlUpdate, params);
117 }
118
119
120 }
121

```

*DefaultLessonDao.java

```
1 package staples.heather.planbook.dao;
2
3 import java.sql.ResultSet;
4
5
6
7
8
9
10
11
12
13
14
15
16
17 @Component
18 @Slf4j
19 public class DefaultLessonDao implements LessonDao {
20
21     @Autowired
22     private NamedParameterJdbcTemplate jdbcTemplate;
23
24     public List<Lesson> listAllLessons() {
25
26         String sqlFetch = "SELECT * FROM lesson";
27
28         Map<String, Object> params = new HashMap<>();
29
30         return jdbcTemplate.query(sqlFetch, params, new RowMapper<>() {
31
32             @Override
33             public Lesson mapRow(ResultSet rs, int rowNum) throws SQLException {
34                 return Lesson.builder()
35                     .lessonId(rs.getInt("lessonId"))
36                     .lessonName(rs.getString("lessonName"))
37                     .objective(rs.getString("objective"))
38                     .content(rs.getString("content"))
39                     .unitId(rs.getInt("unitId"))
40                     .build();
41             }
42         });
43     }
44
45     @Override
46     public List<Lesson> fetchLessonById(int lessonId) {
47
48         String sqlFetch = "SELECT * FROM lesson WHERE lessonId = :lessonId";
49
50         Map<String, Object> params = new HashMap<>();
51         params.put("lessonId", lessonId);
52
53         return jdbcTemplate.query(sqlFetch, params, new RowMapper<>() {
54
55             @Override
```

```

56     public Lesson mapRow(ResultSet rs, int rowNum) throws SQLException {
57         return Lesson.builder()
58             .lessonId(rs.getInt("lessonId"))
59             .lessonName(rs.getString("lessonName"))
60             .objective(rs.getString("objective"))
61             .content(rs.getString("content"))
62             .unitId(rs.getInt("unitId"))
63             .build();
64     });
65 }
66
67 @Override
68 public List<Lesson> fetchLessonsByUnit(int unitId) {
69
70     String sqlFetch = "SELECT * FROM lesson WHERE unitId = :unitId";
71
72     Map<String, Object> params = new HashMap<>();
73     params.put("unitId", unitId);
74
75     return jdbcTemplate.query(sqlFetch, params, new RowMapper<>() {
76
77         @Override
78         public Lesson mapRow(ResultSet rs, int rowNum) throws SQLException {
79             return Lesson.builder()
80                 .lessonId(rs.getInt("lessonId"))
81                 .lessonName(rs.getString("lessonName"))
82                 .objective(rs.getString("objective"))
83                 .content(rs.getString("content"))
84                 .unitId(rs.getInt("unitId"))
85                 .build();
86         });
87     }
88
89
90 public int deleteLesson(int lessonId) {
91     // @formatter:off
92     String sql = ""
93         + "DELETE FROM lesson "
94         + "WHERE lessonId = :lessonId;";
95     // @formatter:on
96
97     Map<String, Object> params = new HashMap<>();
98 }

```



```

98     params.put("lessonId", lessonId);
99     if (jdbcTemplate.update(sql, params) == 0) {
100         throw new NoSuchElementException();
101     };
102     return jdbcTemplate.update(sql, params);
103 }
104
105
106
107
108 public void createLesson(String lessonName, String objective, String content, int unitId) {
109     String sqlCreate = "INSERT INTO lesson (lessonName, objective, content, unitId) VALUES (:lessonName, :objective, :content, :unitId)";
110     Map<String, Object> params = new HashMap<>();
111     params.put("lessonName", lessonName);
112     params.put("objective", objective);
113     params.put("content", content);
114     params.put("unitId", unitId);
115     jdbcTemplate.update(sqlCreate, params);
116 }
117
118
119
120
121
122 public int updateLesson(int lessonId, String newContent) {
123     String sqlUpdate = "UPDATE lesson SET content = :newContent WHERE lessonId = :lessonId";
124     Map<String, Object> params = new HashMap<>();
125     params.put("lessonId", lessonId);
126     params.put("newContent", newContent);
127     if (jdbcTemplate.update(sqlUpdate, params) == 0) {
128         throw new NoSuchElementException();
129     };
130     return jdbcTemplate.update(sqlUpdate, params);
131 }
132
133
134
135
136

```

```
*DefaultSubjectDao.java
1 package staples.heather.planbook.dao;
2
3 import java.sql.ResultSet;
4
14
15 @Component
16 @Slf4j
17 public class DefaultSubjectDao implements SubjectDao {
18
19     @Autowired
20     private NamedParameterJdbcTemplate jdbcTemplate;
21
22
23     @Override
24     public List<Subject> listAllSubjects() {
25
26         String sqlFetch = "SELECT * FROM subject";
27
28         Map<String, Object> params = new HashMap<>();
29
30         return jdbcTemplate.query(sqlFetch, params, new RowMapper<>() {
31
32             @Override
33             public Subject mapRow(ResultSet rs, int rowNum) throws SQLException {
34                 return Subject.builder()
35                     .subjectId(rs.getInt("subjectId"))
36                     .subjectName(rs.getString("subjectName"))
37                     .build();
38             }
39         });
40
41     }
42 }
```

```

1 package staples.heather.planbook.dao;
2
3 import java.sql.ResultSet;
4
5 @Component
6 @Slf4j
7 public class DefaultUnitDao implements UnitDao {
8
9     @Autowired
10     private NamedParameterJdbcTemplate jdbcTemplate;
11
12     @Override
13     public List<Unit> fetchUnitsBySubjectByGrade(int subjectId, int gradeNumber) {
14
15         String sqlFetch = "SELECT * FROM unit WHERE subjectId = :subjectId AND gradeNumber = :gradeNumber";
16
17         Map<String, Object> params = new HashMap<>();
18         params.put("subjectId", subjectId);
19         params.put("gradeNumber", gradeNumber);
20
21         return jdbcTemplate.query(sqlFetch, params, new RowMapper<>() {
22
23             @Override
24             public Unit mapRow(ResultSet rs, int rowNum) throws SQLException {
25                 return Unit.builder()
26                     .unitId(rs.getInt("unitId"))
27                     .unitName(rs.getString("unitName"))
28                     .subjectId(rs.getInt("subjectId"))
29                     .gradeNumber(rs.getInt("gradeNumber"))
30                     .build();
31             }
32         });
33     }
34 }

```

```

1 package staples.heather.planbook.dao;
2
3 import java.util.List;
4
5 public interface LessonDao {
6
7     List<Lesson> fetchLessonById(int lessonId);
8
9     List<Lesson> fetchLessonsByUnit(int unitId);
10
11     List<Lesson> listAllLessons();
12
13     void createLesson(String lessonName, String objective, String content, int unitId);
14
15     int updateLesson(int lessonId, String newContent);
16
17     int deleteLesson(int lessonId);
18 }

```

SubjectDao.java

```
1 package staples.heather.planbook.dao;
2
3 import java.util.List;
4
5
6
7 public interface SubjectDao {
8
9     // List<Subject> fetchSubject(String subjectName);
10
11     List<Subject> listAllSubjects();
12
13     // void createSubject(String subjectName);
14
15     // void updateSubject(String oldName, String newName);
16
17     // void deleteSubject(String subjectName);
18
19 }
20
```

UnitDao.java

Maximize package staples.heather.planbook.dao;

```
2
3 import java.util.List;
4
5
6 public interface UnitDao {
7
8     List<Unit> fetchUnitsBySubjectByGrade(int subjectId, int gradeNumber);
9
10 // List<Unit> listAllUnits();
11 //
12 // void deleteUnit(int id);
13 //
14 // void createUnit(String unitName, int subjectId, int gradeNumber);
15 //
16 // void updateUnit(String oldName, String newName);
17 }
18
```

```

Day.java
1 package staples.heather.planbook.entity;
2
3 import java.util.List;
4
5
6
7
8
9
10 @Data
11 @Builder
12 @NoArgsConstructor
13 @AllArgsConstructor
14 public class Day {
15     private int dayId;
16     private String dayOfTheWeek;
17     private List<Lesson> lessons;
18
19     @JsonIgnore
20     public int getId() {
21         return dayId;
22     }
23
24 }

```

```

Day.java DaysLessons.java
1 package staples.heather.planbook.entity;
2
3 import com.fasterxml.jackson.annotation.JsonIgnore;
4
5
6
7
8
9 @Data
10 @Builder
11 @NoArgsConstructor
12 @AllArgsConstructor
13 public class DaysLessons {
14     private int daysLessonsId;
15     private int dayId;
16     private int lessonId;
17
18 // @JsonIgnore
19 // public int getId() {
20 //     return daysLessonsId;
21 // }
22 }
23

```

```
Day.java DaysLessons.java Grade.java ✕
1 package staples.heather.planbook.entity;
2
3 import com.fasterxml.jackson.annotation.JsonIgnore;
4
5
6
7
8
9 @Data
10 @Builder
11 @NoArgsConstructor
12 @AllArgsConstructor
13 public class Grade {
14     private int gradeNumber;
15     private String gradeName;
16
17
18 @JsonIgnore
19     public int getId() {
20         return gradeNumber;
21     }
22 }
23
24
```

Lesson.java

```
1 package staples.heather.planbook.entity;
2
3 import org.springframework.data.annotation.Transient;
10
11 @Data
12 @Builder
13 @NoArgsConstructor
14 @AllArgsConstructor
15 public class Lesson {
16     private int lessonId;
17
18     @Column(value="lessonName")
19     private String lessonName;
20     private String objective;
21     private String content;
22     private int unitId;
23
24     @JsonIgnore
25     public int getId() {
26         return lessonId;
27     }
28 }
29
30
```

Lesson.java Subject.java

```
1 package staples.heather.planbook.entity;
2
3 import com.fasterxml.jackson.annotation.JsonIgnore;
4
5
6
7
8
9 @Data
10 @Builder
11 @NoArgsConstructor
12 @AllArgsConstructor
13 public class Subject {
14     private int subjectId;
15     private String subjectName;
16
17     @JsonIgnore
18     public int getId() {
19         return subjectId;
20     }
21 }
22
```

Unit.java

```
1 package staples.heather.planbook.entity;
2
3 import com.fasterxml.jackson.annotation.JsonIgnore;
4
5
6
7
8
9 @Data
10 @Builder
11 @NoArgsConstructor
12 @AllArgsConstructor
13 public class Unit {
14     private int unitId;
15     private String unitName;
16     private int subjectId;
17     private int gradeNumber;
18
19     @JsonIgnore
20     public int getId() {
21         return unitId;
22     }
23 }
24
```

Planbook/src/main/java/staples/heather/planbook/entity/Unit.java

GlobalExceptionHandler.java

```
1 package staples.heather.planbook.errorHandler;
2
3 import java.time.ZonedDateTime;
4
5 @RestControllerAdvice
6 @Slf4j
7 public class GlobalExceptionHandler {
8     private enum LogStatus {
9         STACK_TRACE, MESSAGE_ONLY
10    }
11
12    @ExceptionHandler({ConstraintViolationException.class})
13    @ResponseStatus(code = HttpStatus.BAD_REQUEST)
14    public Map<String, Object> handleConstraintViolationException(ConstraintViolationException e, WebRequest webRequest) {
15        return createExceptionMessage(e, HttpStatus.BAD_REQUEST, webRequest, LogStatus.MESSAGE_ONLY);
16    }
17
18    @ExceptionHandler({DataIntegrityViolationException.class})
19    @ResponseStatus(code = HttpStatus.BAD_REQUEST)
20    public Map<String, Object> handleConstraintViolationException(DataIntegrityViolationException e, WebRequest webRequest) {
21        return createExceptionMessage(e, HttpStatus.BAD_REQUEST, webRequest, LogStatus.MESSAGE_ONLY);
22    }
23
24    @ExceptionHandler({Exception.class})
25    @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
26    public Map<String, Object> handleException(Exception e, WebRequest webRequest) {
27        return createExceptionMessage(e, HttpStatus.INTERNAL_SERVER_ERROR, webRequest, LogStatus.MESSAGE_ONLY);
28    }
29
30    @ExceptionHandler({NoSuchElementException.class})
31    @ResponseStatus(code = HttpStatus.NOT_FOUND)
32    public Map<String, Object> handleNoSuchElementException(
33        NoSuchElementException e, WebRequest webRequest) {
34        return createExceptionMessage(e, HttpStatus.NOT_FOUND, webRequest, LogStatus.STACK_TRACE);
35    }
36
37    private Map<String, Object> createExceptionMessage(Exception e, HttpStatus status, WebRequest webRequest, LogStatus logStatus) {
38        Map<String, Object> error = new HashMap<>();
39        String timestamp = ZonedDateTime.now().format(DateTimeFormatter.RFC_1123_DATE_TIME);
40
41        if (webRequest instanceof ServletWebRequest) {
42
43            error.put("uri", ((ServletWebRequest) webRequest).getRequest().getRequestURI());
44        }
45
46        error.put("message", e.toString());
47        error.put("status code", status.value());
48        error.put("timestamp", timestamp);
49        error.put("reason", status.getReasonPhrase());
50
51        if (logStatus == LogStatus.MESSAGE_ONLY) {
52            Log.error("Exception: {}", e.toString());
53        }
54        else {
55            Log.error("Exception: {}", e);
56        }
57
58        return error;
59    }
60 }
61 }
```

DaysLessonsService.java

```
1 package staples.heather.planbook.service;
2
3 import java.util.List;
4
5
6
7
8 public interface DaysLessonsService {
9
10     List<Lesson> fetchLessonsByDay(String dayOfTheWeek);
11
12     List<Day> listWeeksLessons();
13
14     int updateDayOfLesson(int lessonId, int newDay);
15
16     void clearWeeksLessons();
17
18     int deleteLessonFromDay(int lessonId);
19
20     void assignLessonsToDay(int dayId, int lessonId);
21
22
23
24
25 }
```

DaysLessonsService.java

DefaultDaysLessonsService.java ✕

```
1 package staples.heather.planbook.service;
2
3 import java.util.List;
4
11
12 @Service
13 @Slf4j
14 public class DefaultDaysLessonsService implements DaysLessonsService{
15
16     @Autowired
17     private DaysLessonsDao daysLessonsDao;
18
19
20     @Override
21     public List<Lesson> fetchLessonsByDay(String dayOfTheWeek) {
22         List<Lesson> lessonsByDay = daysLessonsDao.fetchLessonsByDay(dayOfTheWeek);
23         return lessonsByDay;
24     }
25
26     @Override
27     public List<Day> listWeeksLessons() {
28         List<Day> lessonsForWeek = daysLessonsDao.listWeeksLessons();
29         return lessonsForWeek;
30     }
31
32     @Override
33     public void assignLessonsToDay(int dayId, int lessonId) {
34         daysLessonsDao.assignLessonsToDay(dayId, lessonId);
35     }
36
37     @Override
38     public int updateDayOfLesson(int lessonId, int newDay) {
39         return daysLessonsDao.updateDayOfLesson(lessonId, newDay);
40     }
41
42     public int deleteLessonFromDay(int lessonId) {
43         return daysLessonsDao.deleteLessonFromDay(lessonId);
44     }
45
46     public void clearWeeksLessons() {
47         daysLessonsDao.clearWeeksLessons();
48     }
49
50 }
```

```
DaysLessonsService.java  DefaultDaysLessonsService.java  DefaultLessonService.java  ⌵
1 package staples.heather.planbook.service;
2
3+ import java.util.List;
9
10 @Service
11 @Slf4j
12 public class DefaultLessonService implements LessonService {
13
14+ @Autowired
15     private LessonDao lessonDao;
16
17+ @Override
18     public List<Lesson> fetchLessonById(int lessonId) {
19         List<Lesson> lesson = lessonDao.fetchLessonById(lessonId);
20         return lesson;
21     }
22
23+ @Override
24     public List<Lesson> fetchLessonsByUnit(int unitId) {
25         List<Lesson> lessons = lessonDao.fetchLessonsByUnit(unitId);
26         return lessons;
27     }
28
29+ @Override
30     public List<Lesson> listAllLessons() {
31         List<Lesson> lessons = lessonDao.listAllLessons();
32         return lessons;
33     }
34
35+ @Override
36     public void createLesson(String lessonName, String objective, String content, int unitId) {
37         lessonDao.createLesson(lessonName, objective, content, unitId);
38     }
39
40+ @Override
41     public int updateLesson(int id, String newContent) {
42         return lessonDao.updateLesson(id, newContent);
43     }
44
45+ @Override
46     public int deleteLesson(int id) {
47         return lessonDao.deleteLesson(id);
48     }
}
```

DaysLessonsService.java DefaultDaysLessonsService.java DefaultLessonService.java *DefaultSubjectService.java

```
1 package staples.heather.planbook.service;
2
3 import java.util.List;
4
5
6
7
8
9
10 @Service
11 @Slf4j
12 public class DefaultSubjectService implements SubjectService{
13
14
15     @Autowired
16     private SubjectDao subjectDao;
17
18     // @Override
19     // public List<Subject> fetchSubject(String subjectName) {
20     //     List<Subject> subject = subjectDao.fetchSubject(subjectName);
21     //     return subject;
22     // }
23
24     @Override
25     public List<Subject> listAllSubjects() {
26         List<Subject> subjects = subjectDao.listAllSubjects();
27         return subjects;
28     }
29
30     // @Override
31     // public void createSubject(String subjectName) {
32     //     subjectDao.createSubject(subjectName);
33     // }
34
35     // @Override
36     // public void updateSubject(String oldName, String newName) {
37     //     subjectDao.updateSubject(oldName, newName);
38     // }
39
40     // @Override
41     // public void deleteSubject(String subjectName) {
42     //     subjectDao.deleteSubject(subjectName);
43     // }
44 // }
45 }
46
```

```
Console  DefaultUnitService.java x
1 package staples.heather.planbook.service;
2
3+ import java.util.List;
9
10 @Service
11 @Slf4j
12 public class DefaultUnitService implements UnitService{
13
14
15 @Autowired
16 private UnitDao unitDao;
17
18 @Override
19 public List<Unit> fetchUnitsBySubjectByGrade(int subjectId, int gradeNumber) {
20     List<Unit> units = unitDao.fetchUnitsBySubjectByGrade(subjectId, gradeNumber);
21     return units;
22 }
23 //
24 // @Override
25 // public List<Unit> listAllUnits() {
26 //     List<Unit> units = unitDao.listAllUnits();
27 //     return units;
28 // }
29 //
30 // @Override
31 // public void createUnit(String unitName, int subjectId, int gradeNumber) {
32 //     unitDao.createUnit(unitName, subjectId, gradeNumber);
33 // }
34 //
35 // @Override
36 // public void updateUnit(String oldName, String newName) {
37 //     unitDao.updateUnit(oldName, newName);
38 // }
39 //
40 // @Override
41 // public void deleteUnit(int id) {
42 //     unitDao.deleteUnit(id);
43 // }
44 //
45 }
```

LessonService.java

```
1 package staples.heather.planbook.service;
2
3 import java.util.List;
4
5
6 public interface LessonService {
7
8     List<Lesson> fetchLessonById(int lessonId);
9
10    List<Lesson> fetchLessonsByUnit(int unitId);
11
12    List<Lesson> listAllLessons();
13
14    void createLesson(String lessonName, String objective, String content, int unitId);
15
16    int updateLesson(int lessonId, String newContent);
17
18    int deleteLesson(int lessonId);
19 }
20
21
```

LessonService.java

SubjectService.java

Planbook/src/main/java/staples/heather/planbook/service/LessonService.java

```
2
3 import java.util.List;
4
5
6 public interface SubjectService {
7
8     // List<Subject> fetchSubject(String subjectName);
9
10    List<Subject> listAllSubjects();
11
12    // void createSubject(String subjectName);
13
14    // void updateSubject(String oldName, String newName);
15
16    // void deleteSubject(String subjectName);
17
18 }
19
```

UnitService.java ✕

```
1 package staples.heather.planbook.service;
2
3 import java.util.List;
4
5
6 public interface UnitService {
7
8     List<Unit> fetchUnitsBySubjectByGrade(int subjectId, int gradeNumber);
9
10    // List<Unit> listAllUnits();
11    //
12    // void deleteUnit(int id);
13    //
14    // void createUnit(String unitName, int subjectId, int gradeNumber);
15    //
16    // void updateUnit(String oldName, String newName);
17    //
18 }
19
```