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).

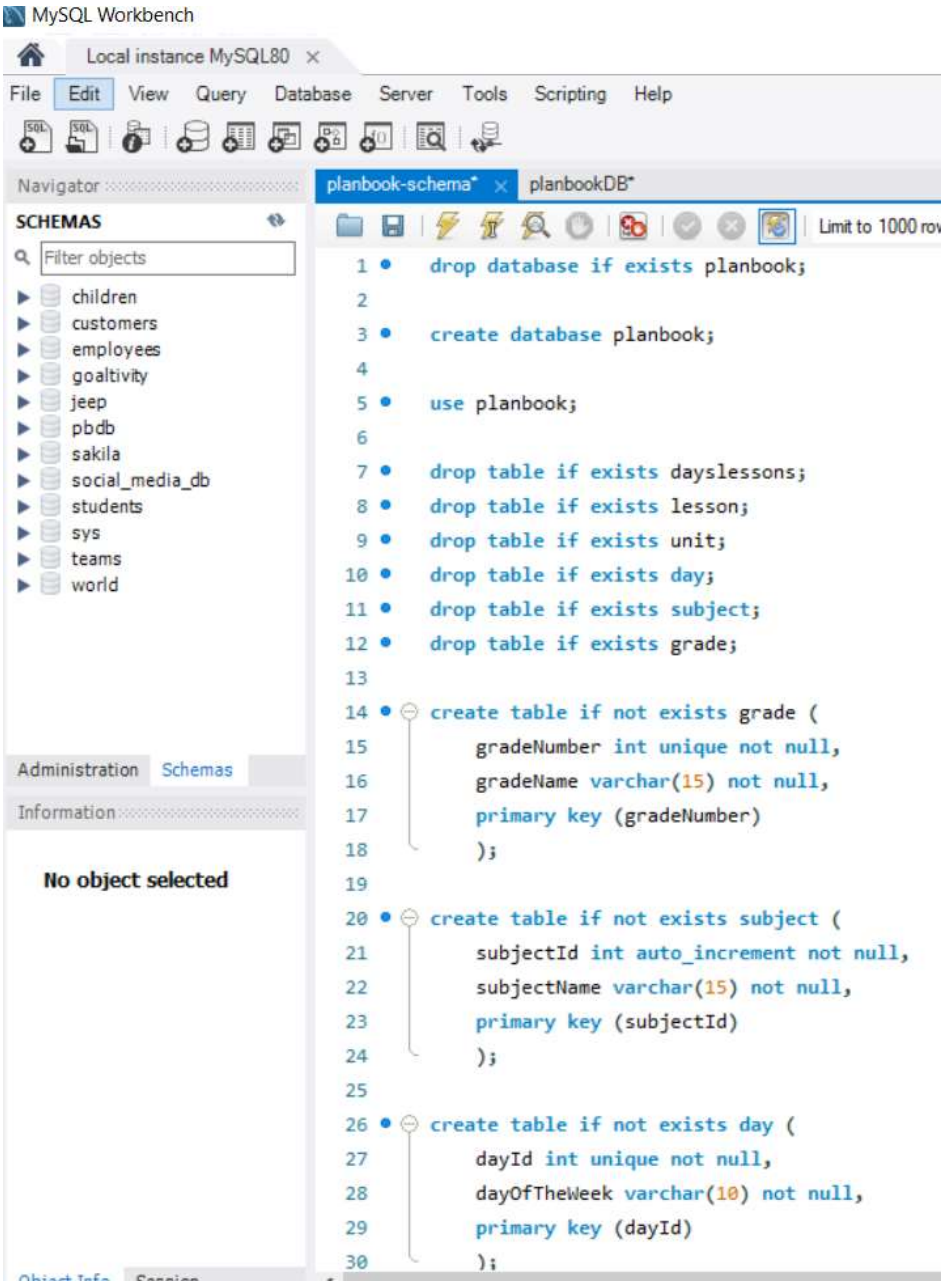
User can view their lesson by day (READ).

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

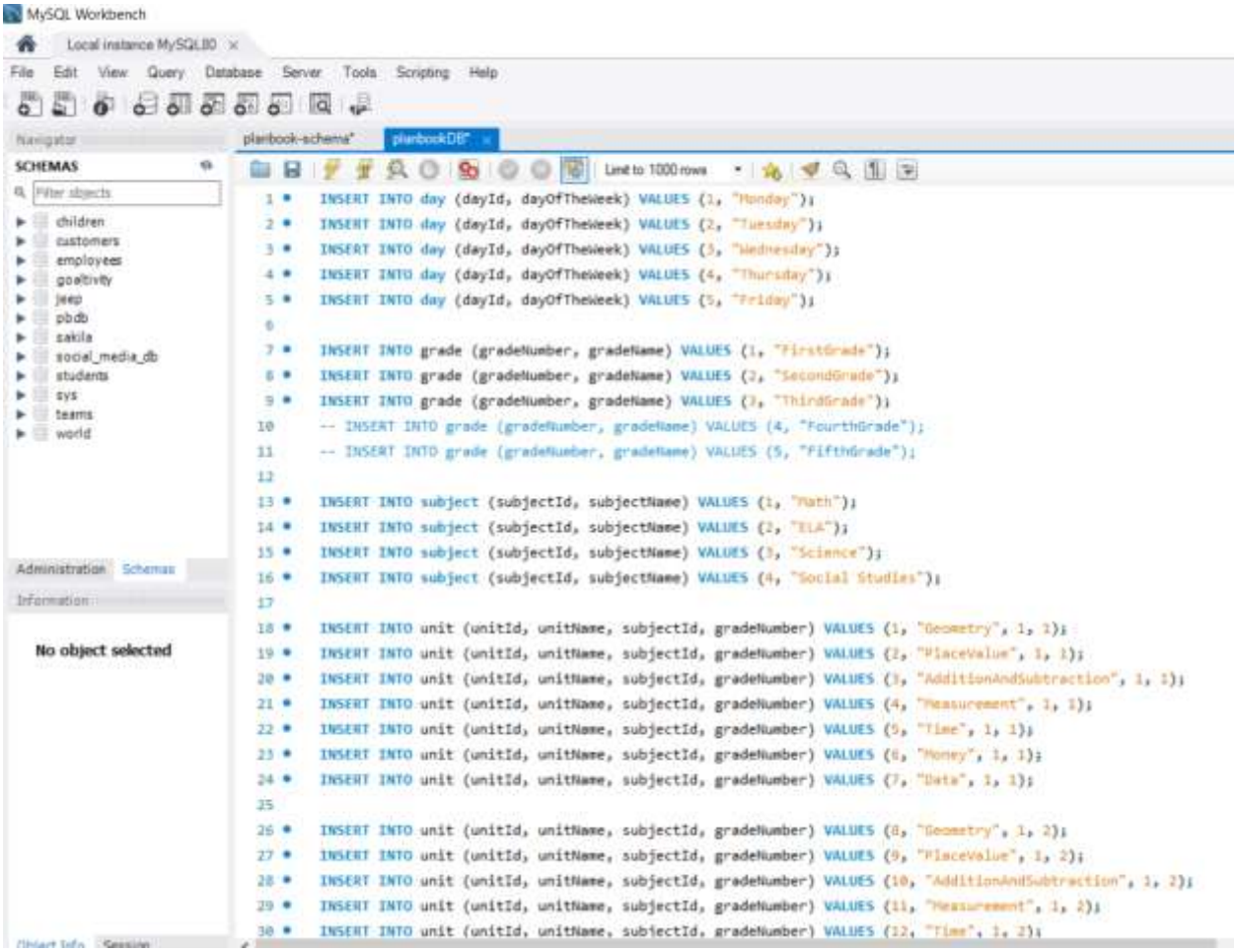
User can reschedule a lesson for another day (UPDATE).

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

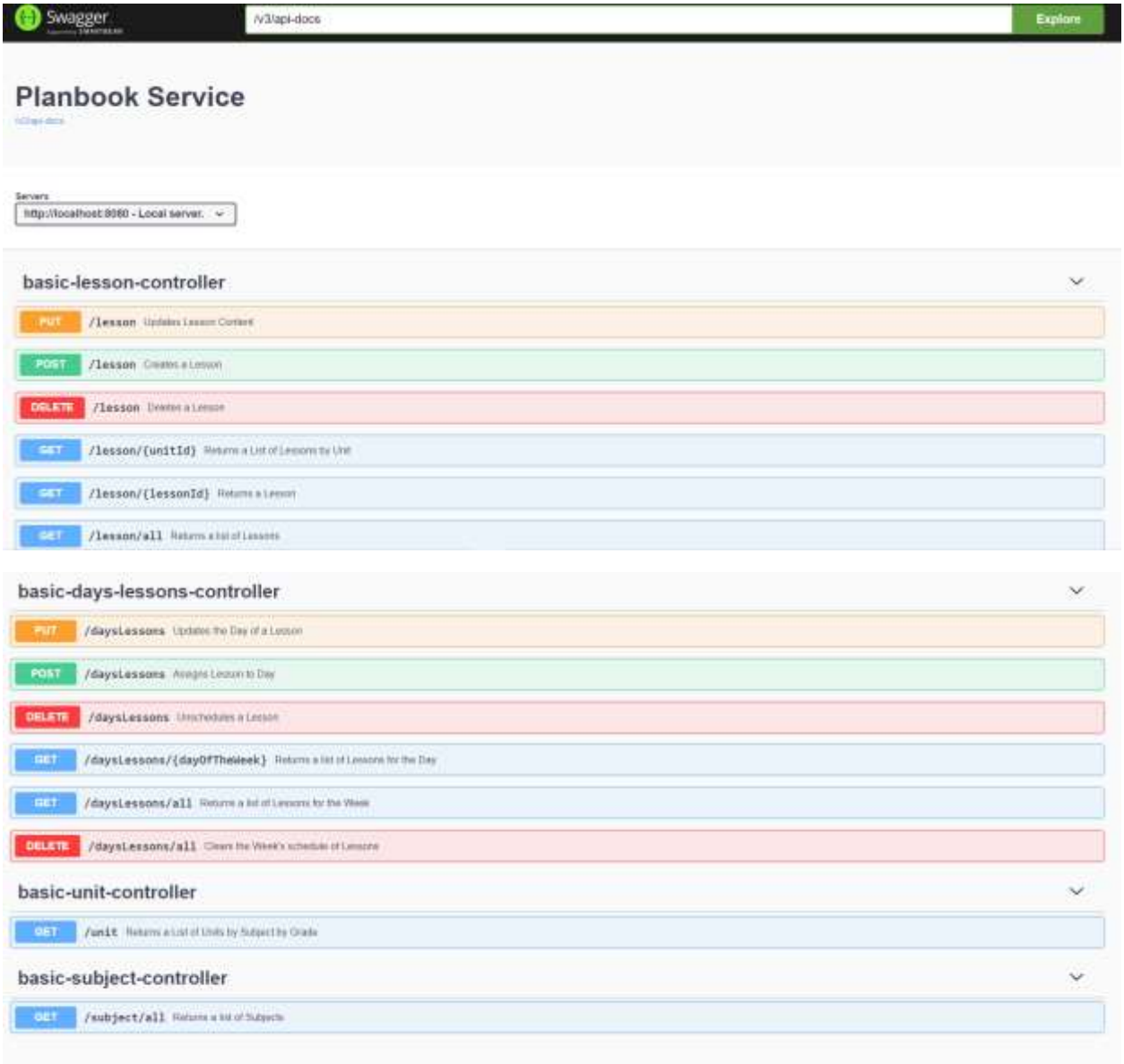
Database:


MySQL Workbench
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
Filter objects
children
customers
employees
goaltivity
jeep
pbdb
sakila
social_media_db
students
sys
teams
world
Administration Schemas
Information
No object selected
planbook-schema*
planbookDB*
Limit to 1000 rows
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);

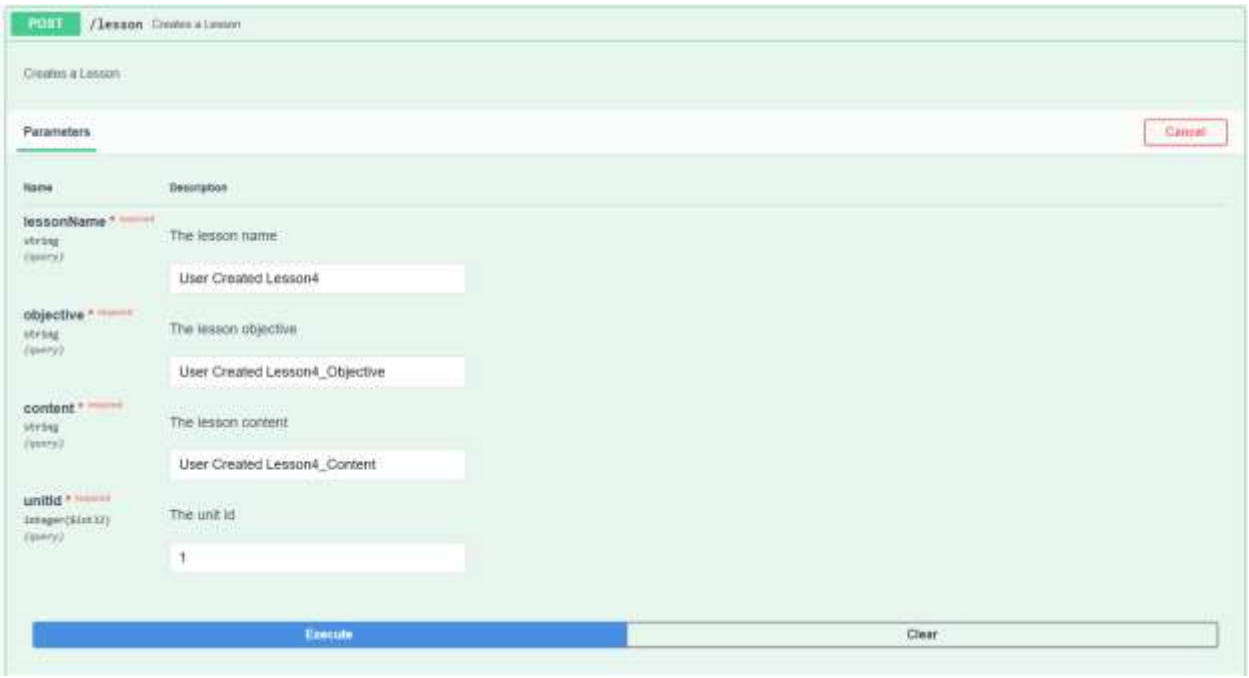
```sql

create table if not exists unit (
    unitId int auto_increment not null,
    unitName varchar(25) not null,
    subjectId int not null,
    gradeNumber int not null,
    primary key (unitId),
    foreign key (subjectId) references subject(subjectId),
    foreign key (gradeNumber) references grade(gradeNumber)
    );

create table lesson (
    lessonId int not null auto_increment,
    lessonName varchar(75) not null,
    objective varchar(255) not null,
    content text not null,
    unitId int not null,
    primary key (lessonId),
    foreign key (unitId) references unit(unitId)
    );

create table DaysLessons (
    dayslessonsId int not null auto_increment,
    dayId int not null,
    lessonId int not null,
    primary key (dayslessonsId),
    foreign key (dayId) references day(dayId),
    foreign key (lessonId) references lesson(lessonID)
    );

```


MySQL Workbench
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
Filter objects
children
customers
employees
goeltivity
jeep
pbdb
sakila
social_media_db
students
sys
teams
world
Administration Schemas
Information
No object selected
planbook-schema*
planbookDB*
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 Lesson 3_Objective", "User Created Lesson3_Content", 1);

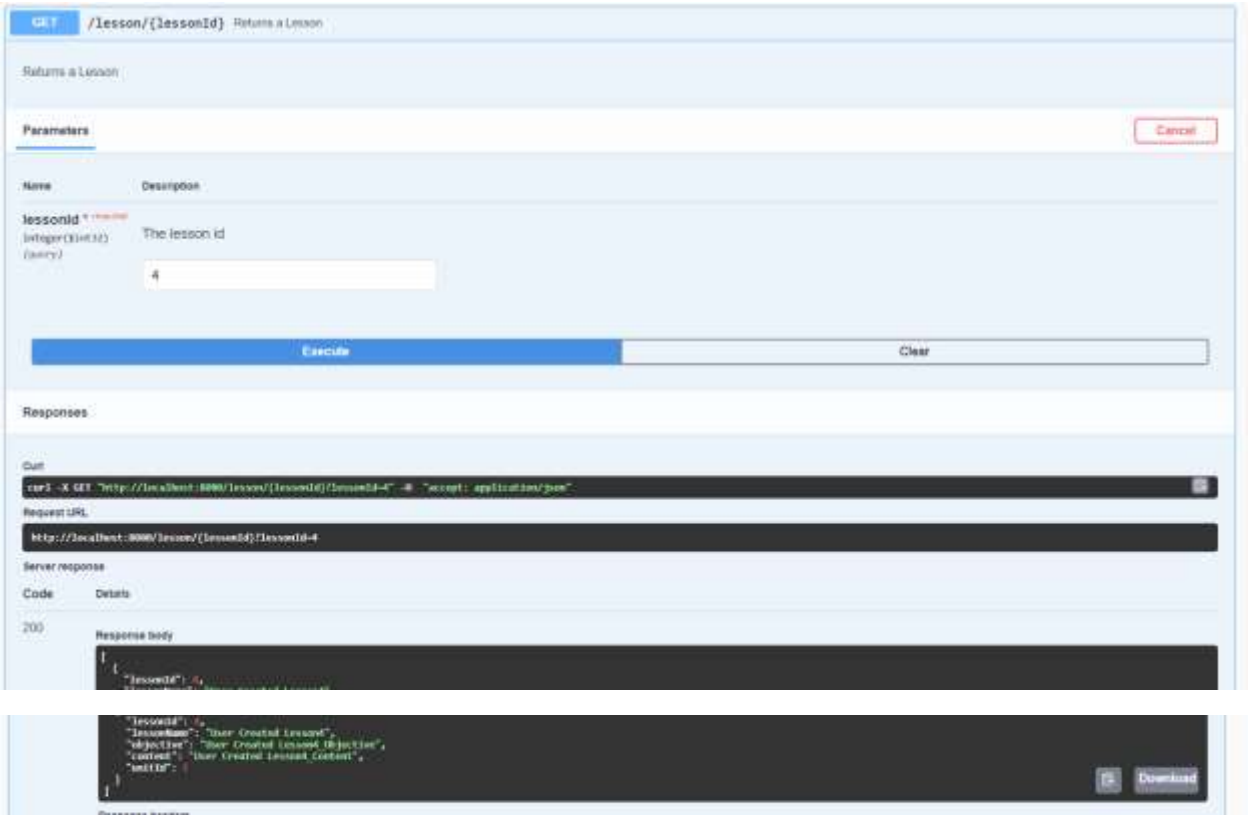
Swagger:


Swagger
/v3/api-docs
Explore
Planbook Service
Servers
http://localhost:8080 - Local server
basic-lesson-controller
PUT /lesson
POST /lesson
DELETE /lesson
GET /lesson/{unitId}
GET /lesson/{lessonId}
GET /lesson/all
basic-days-lessons-controller
PUT /daysLessons
POST /daysLessons
DELETE /daysLessons
GET /daysLessons/{dayOfTheWeek}
GET /daysLessons/all
DELETE /daysLessons/all
basic-unit-controller
GET /unit
basic-subject-controller
GET /subject/all

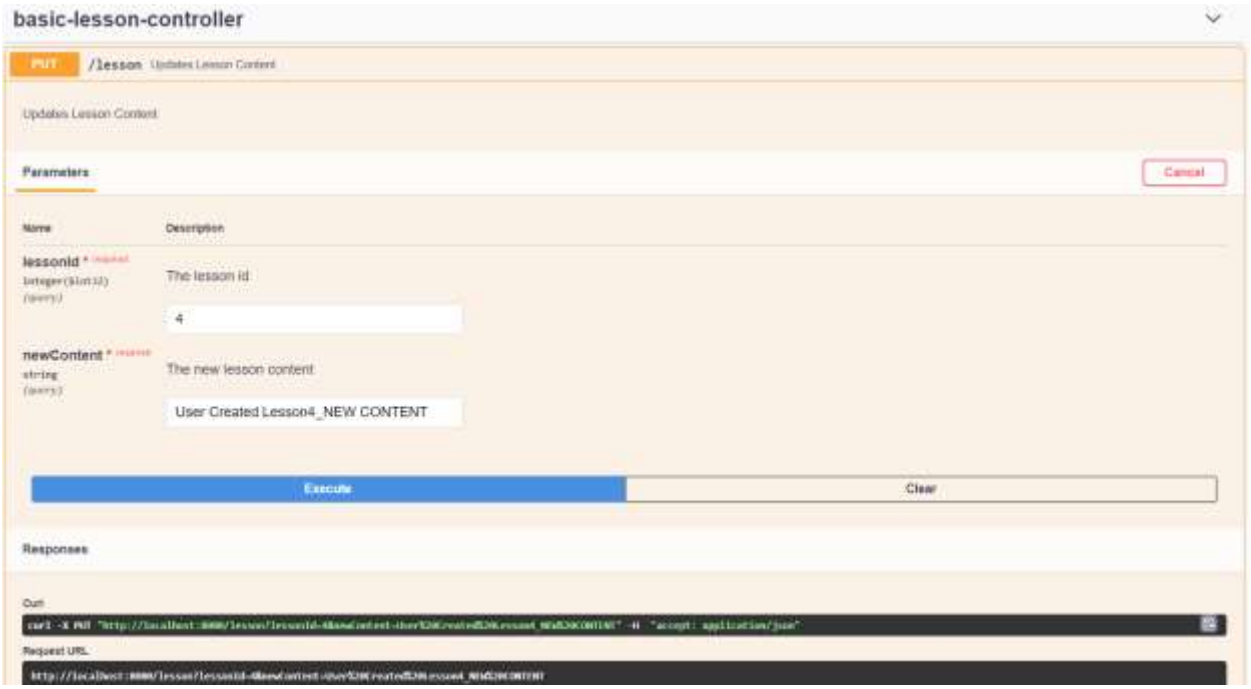
Create Lesson



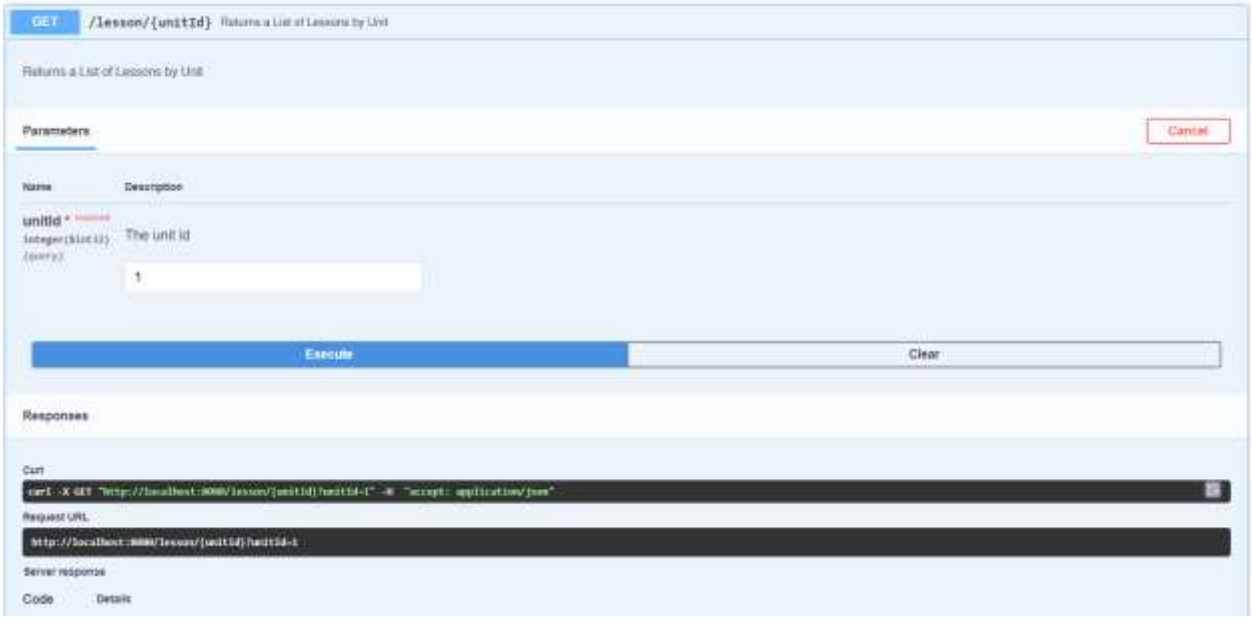
Get Lesson By ID / shows lesson 4 just created

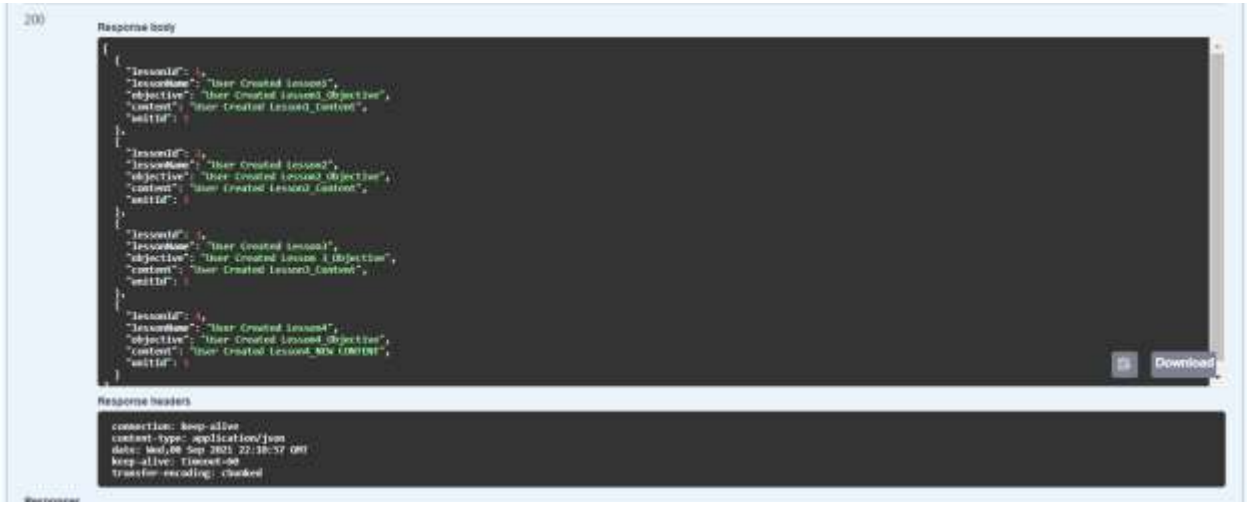


Update Lesson

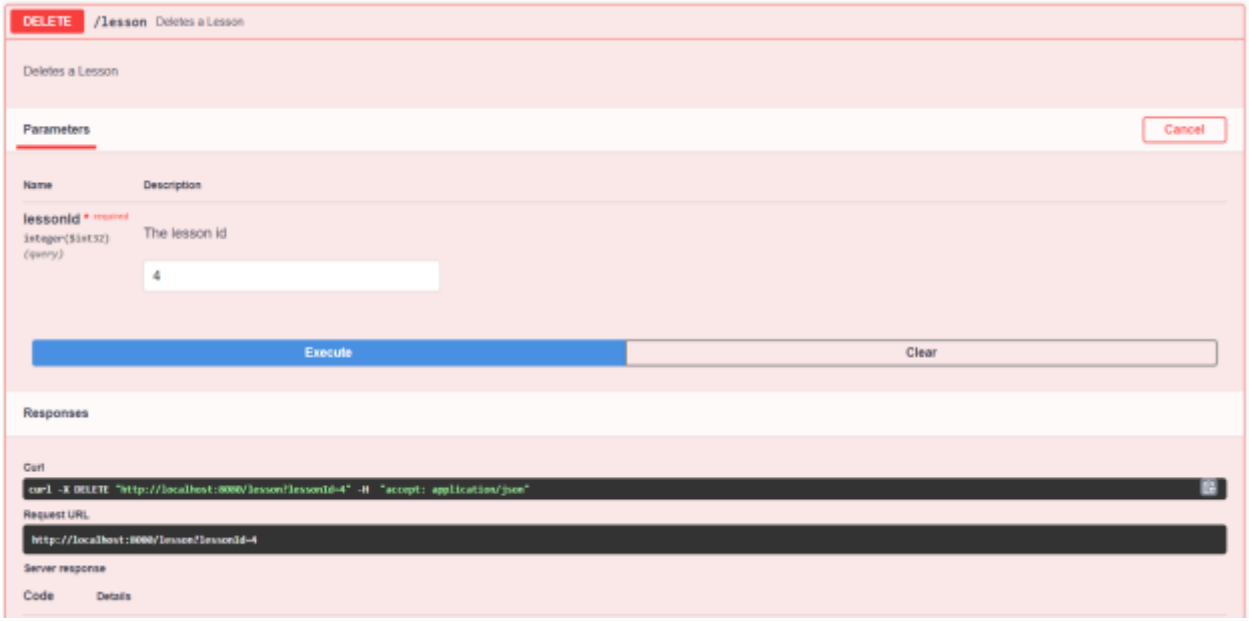


List Lessons By Unit / shows updated lesson 4

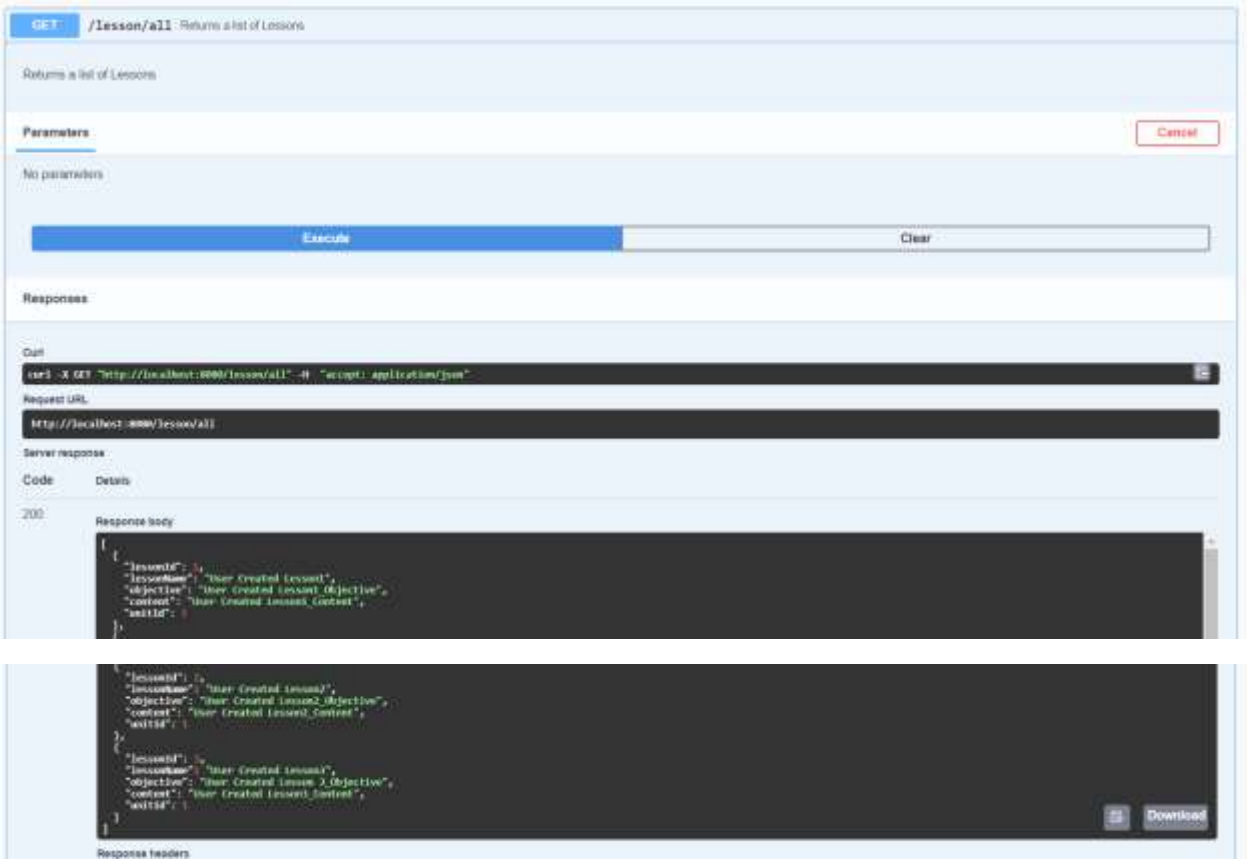




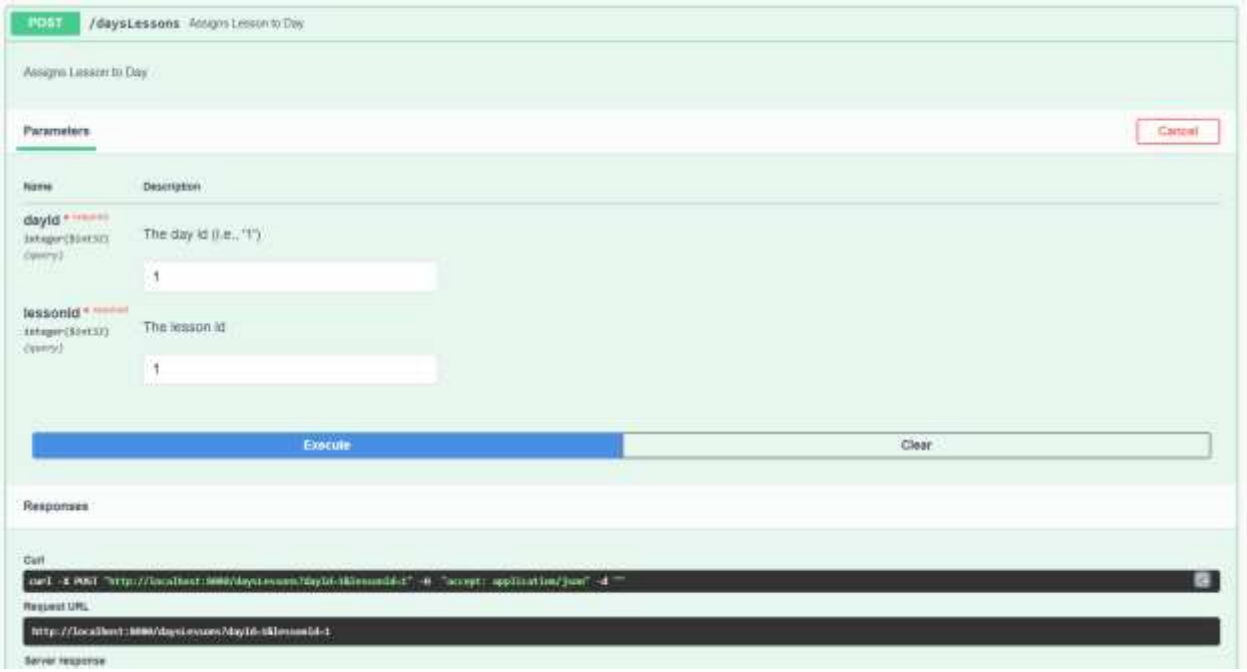
Delete a Lesson



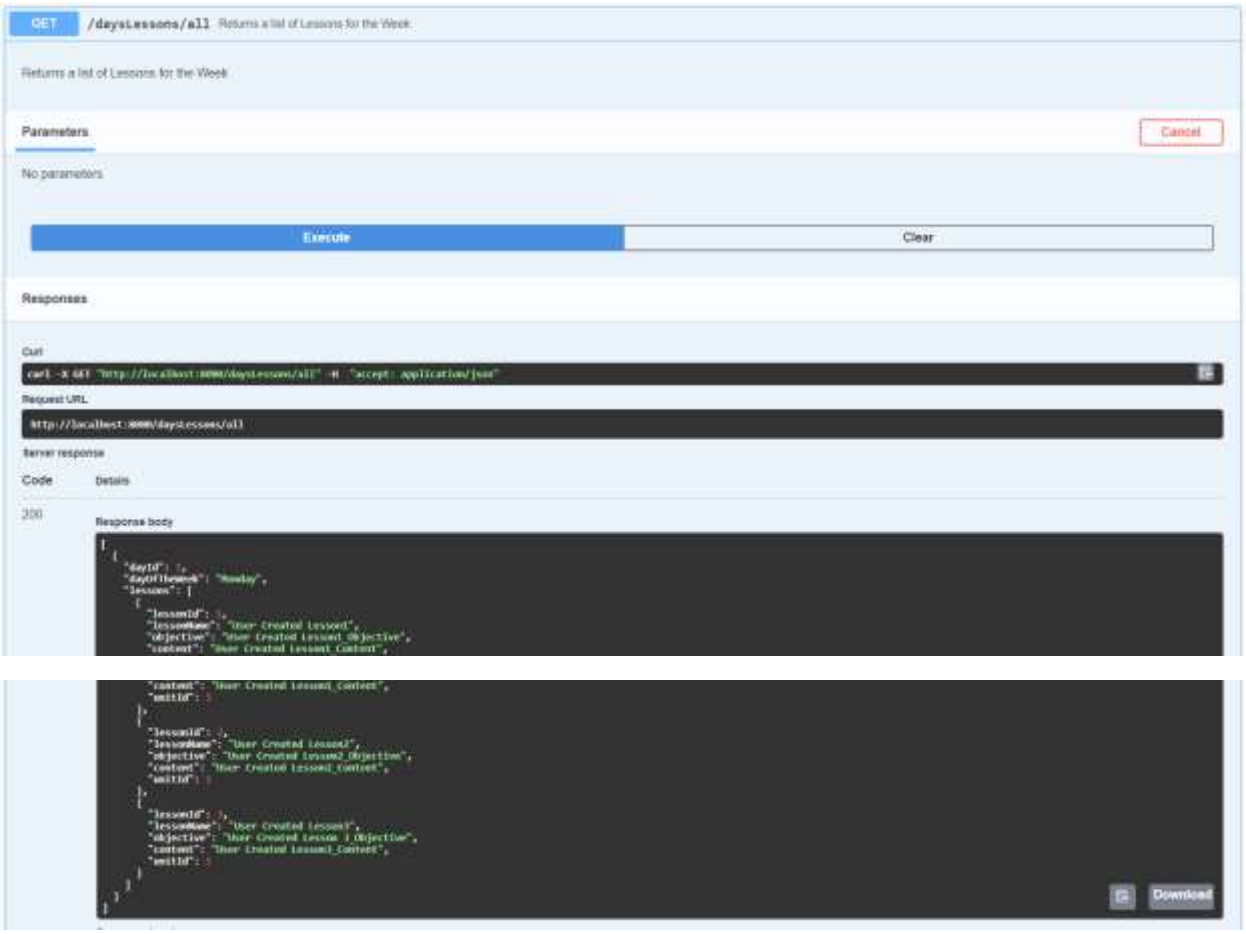
List All Lessons / shows lesson 4 has been deleted



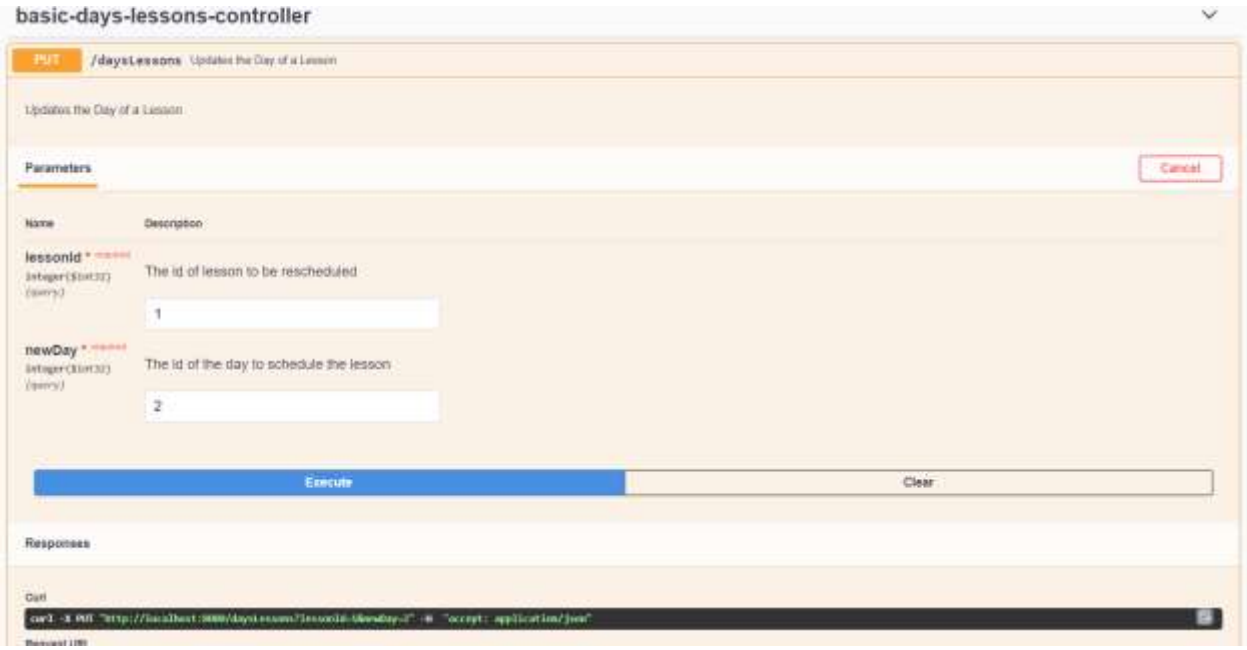
Assign a Lesson to a Day



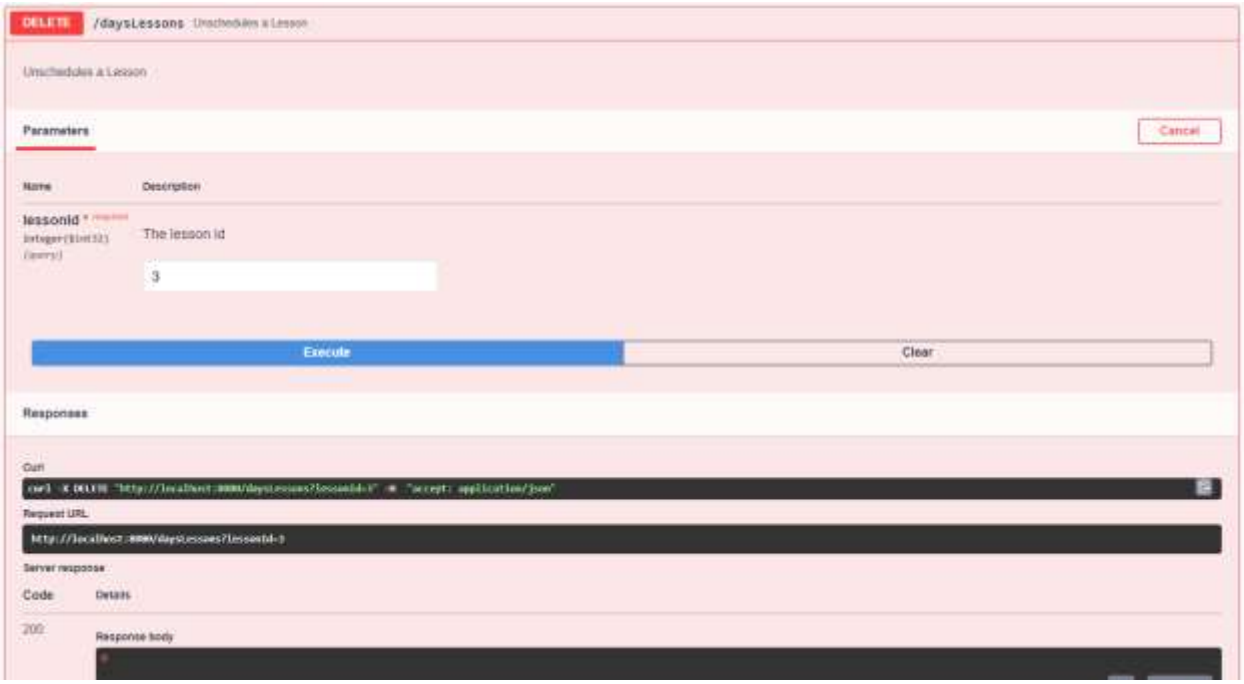
List Lessons Schedule for the Week / shows Lesson 1 (and 2 &3) assigned



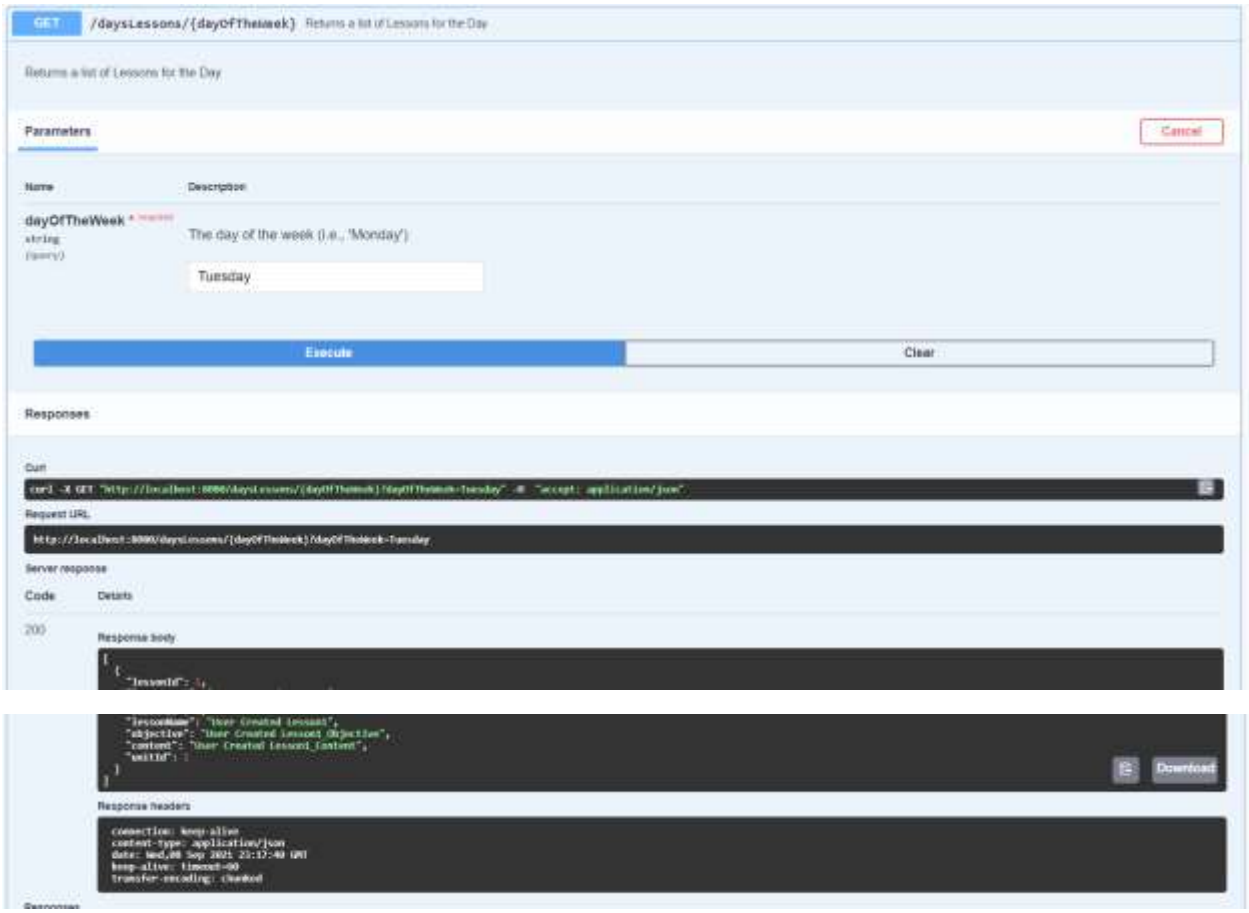
Reschedule a day’s Lessons



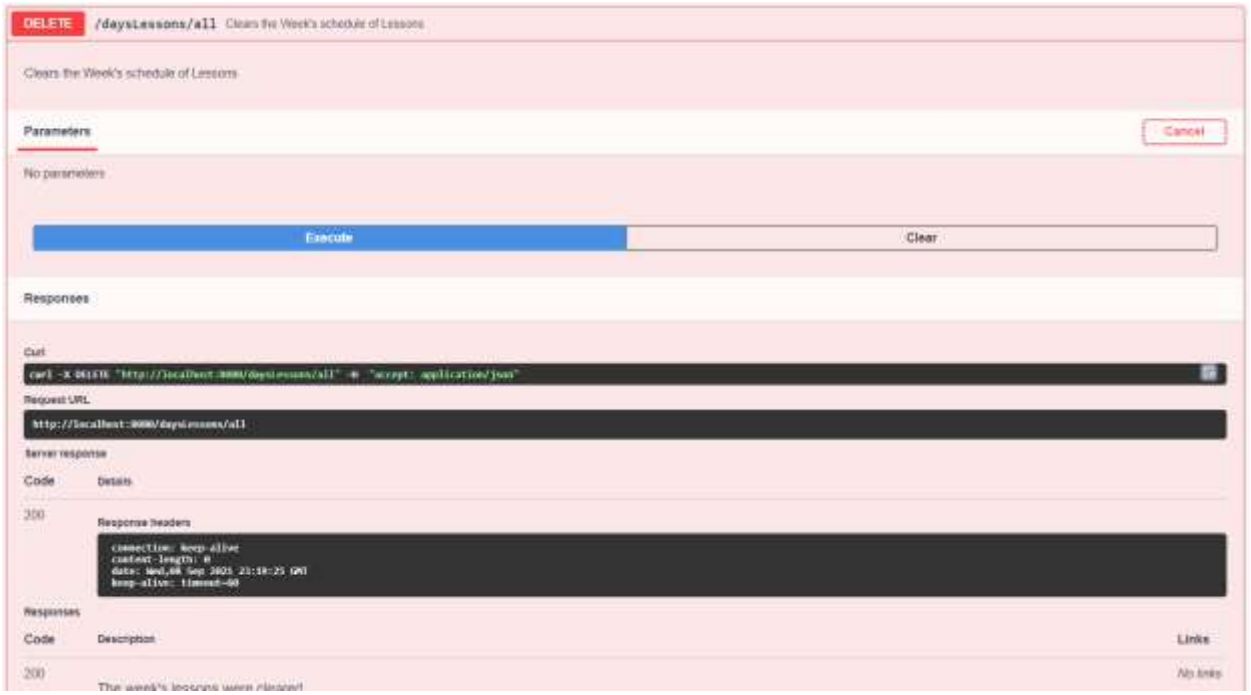
Remove a Lesson from a Day



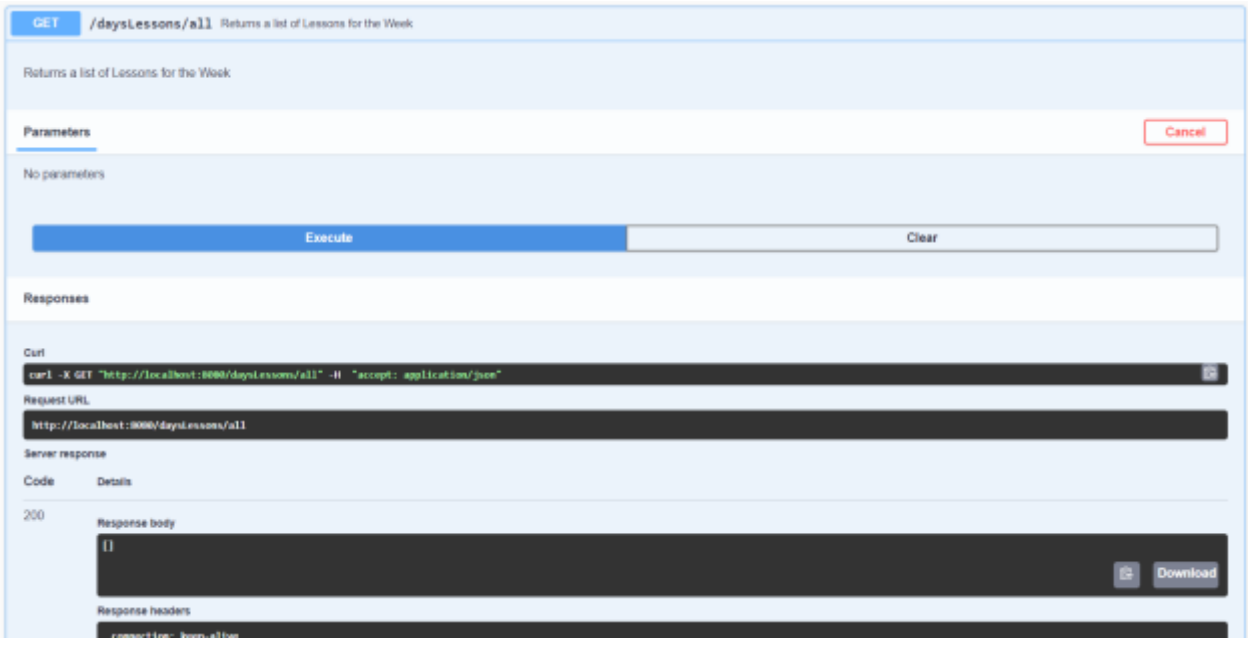
Show a Day's Lessons / shows lesson 2 rescheduled from Monday (dayId 1) to Tuesday (dayId 2)



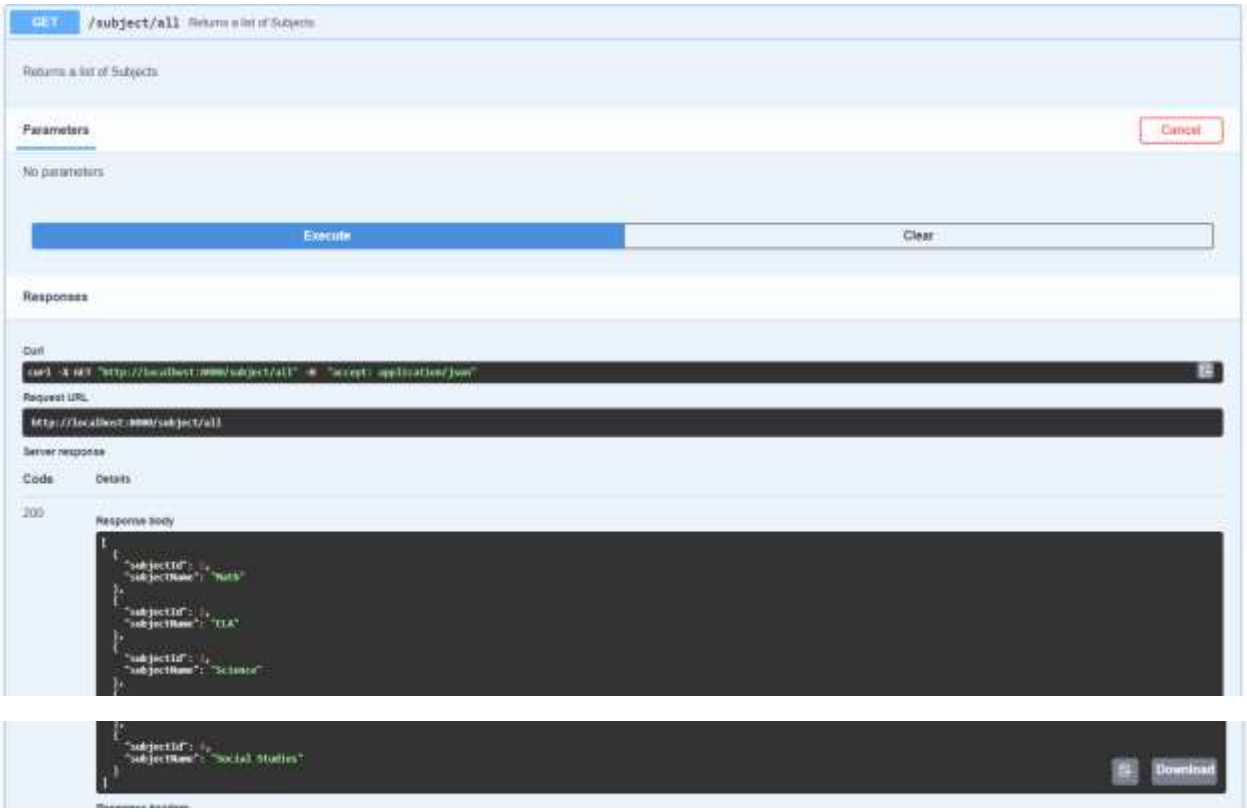
Cleared All Lessons from the Week



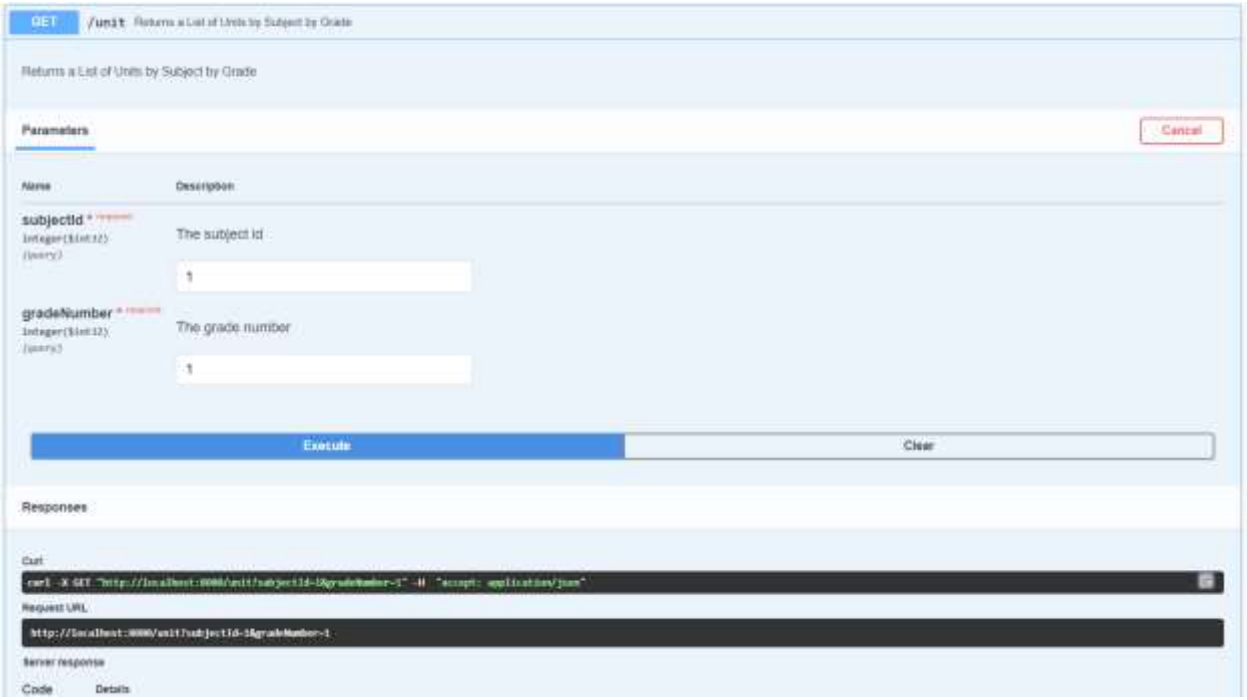
Cleared of lessons



View Subjects



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
The subject id
1
gradeNumber
The grade number
1
Execute
Clear
Responses
Curl
Request URL
Server response
Code
Details


200
Response body
Download
Response headers

ComponentScanMarker.java

```java
package staples.heather;

public interface ComponentScanMarker {

}

```

application.yaml

```yaml
spring:
  datasource:
    password: Jinghy6713
    username: root
    url: jdbc:mysql://localhost:3306/planbook

logging:
  level:
    root: warn
    '[staples.heather]': debug
```

GlobalErrorHandler.java

```java
package staples.heather.planbook.errorHandler;

import java.time.ZonedDateTime;

@RestControllerAdvice
@Slf4j
public class GlobalErrorHandler {
  private enum LogStatus {
    STACK_TRACE, MESSAGE_ONLY
    }

  @ExceptionHandler(ConstraintViolationException.class)
  @ResponseStatus(code = HttpStatus.BAD_REQUEST)
  public Map<String, Object> handleConstraintViolationException(ConstraintViolationException e, WebRequest webRequest) {
    return createExceptionMessage(e, HttpStatus.BAD_REQUEST, webRequest, LogStatus.MESSAGE_ONLY);
  }

  @ExceptionHandler(DataIntegrityViolationException.class)
  @ResponseStatus(code = HttpStatus.BAD_REQUEST)
  public Map<String, Object> handleConstraintViolationException(DataIntegrityViolationException e, WebRequest webRequest) {
    return createExceptionMessage(e, HttpStatus.BAD_REQUEST, webRequest, LogStatus.MESSAGE_ONLY);
  }


  @ExceptionHandler(Exception.class)
  @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
  public Map<String, Object> handleException(Exception e, WebRequest webRequest) {
    return createExceptionMessage(e, HttpStatus.INTERNAL_SERVER_ERROR, webRequest, LogStatus.MESSAGE_ONLY);
  }

  @ExceptionHandler(NoSuchElementException.class)
  @ResponseStatus(code = HttpStatus.NOT_FOUND)
  public Map<String, Object> handleNoSuchElementException(
    NoSuchElementException e, WebRequest webRequest) {
      return createExceptionMessage(e, HttpStatus.NOT_FOUND, webRequest, LogStatus.STACK_TRACE);
    }


  private Map<String, Object> createExceptionMessage(Exception e, HttpStatus status, WebRequest webRequest, LogStatus logStatus) {
    Map<String, Object> error = new HashMap<>();
    String timestamp = ZonedDateTime.now().format(DateTimeFormatter.RFC_1123_DATE_TIME);

    if(webRequest instanceof ServletWebRequest) {
```

```java
      error.put("uri",  ((ServletWebRequest) webRequest).getRequest().getRequestURI());
    }

    error.put("message", e.toString());
    error.put("status code", status.value());
    error.put("timestamp", timestamp);
    error.put("reason", status.getReasonPhrase());

    if(logStatus == LogStatus.MESSAGE_ONLY) {
      log.error("Exception: {}", e.toString());
    }
    else {
      log.error("Exception: {}", e);
    }

    return error;
  }
}
```

Planbook.java

```java
package staples.heather.planbook;

import org.springframework.boot.SpringApplication;

@SpringBootApplication(scanBasePackageClasses = { ComponentScanMarker.class })
public class Planbook {

  public static void main(String[] args) {
    SpringApplication.run(Planbook.class, args);
  }
}
```

BasicDaysLessonsController.java

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

import java.util.List;

  @RestController
  @Slf4j
  public class BasicDaysLessonsController implements DaysLessonsController{

    @Autowired
    private DaysLessonsService daysLessonsService;


    public List<Lesson> fetchLessonsByDay(String dayOfTheWeek) {
      return daysLessonsService.fetchLessonsByDay(dayOfTheWeek);
    }

    public List<Day> listWeeksLessons() {
      return daysLessonsService.listWeeksLessons();
    }

    public void assignLessonsToDay(int dayId, int lessonId) {
      daysLessonsService.assignLessonsToDay(dayId, lessonId);
    }

    public int updateDayOfLesson(int lessonId, int newDay) {
      return daysLessonsService.updateDayOfLesson(lessonId, newDay);
    }

    public void clearWeeksLessons() {
      daysLessonsService.clearWeeksLessons();
    }

    public int deleteLessonFromDay(int lessonId) {
      return daysLessonsService.deleteLessonFromDay(lessonId);
    }

  }
```

*BasicLessonController.java

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

import java.util.List;

@RestController
@Slf4j
public class BasicLessonController implements LessonController{

  @Autowired
  private LessonService lessonService;

  @Override
  public List<Lesson> fetchLessonById(int lessonId) {
    return lessonService.fetchLessonById(lessonId);
  }

  @Override
  public List<Lesson> fetchLessonsByUnit(int unitId) {
    return lessonService.fetchLessonsByUnit(unitId);
  }

  @Override
  public void createLesson(String lessonName, String objective, String content, int unitId) {
    lessonService.createLesson(lessonName, objective, content, unitId);
  }

  @Override
  public int updateLesson(int lessonId, String newContent) {
    return lessonService.updateLesson(lessonId, newContent);
  }

  @Override
  public int deleteLesson(int lessonId) {
    return lessonService.deleteLesson(lessonId);

    }

  @Override
  public List<Lesson> listAllLessons() {
    return lessonService.listAllLessons();
  }
}
```

BasicSubjectController.java

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

import java.util.List;

  @RestController
  @Slf4j
  public class BasicSubjectController implements SubjectController{

    @Autowired
    private SubjectService subjectService;

//     @Override
//     public List<Subject> fetchSubject(String subjectName) {
//       return subjectService.fetchSubject(subjectName);
//     }
     @Override
     public List<Subject> listAllSubjects() {
       return subjectService.listAllSubjects();
     }

//    @Override
//    public void createSubject(String subjectName) {
//      subjectService.createSubject(subjectName);
//    }

//    @Override
//    public void updateSubject(String oldName, String newName) {
//      // TODO Auto-generated method stub
//      subjectService.updateSubject(oldName, newName);
//    }

//    @Override
//    public void deleteSubject(String subjectName) {
//      subjectService.deleteSubject(subjectName);
//
//    }

  }
```

*BasicUnitController.java

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

import java.util.List;


@RestController
@Slf4j
public class BasicUnitController implements UnitController{

    @Autowired
    private UnitService unitService;

     @Override
     public List<Unit> fetchUnitsBySubjectByGrade(int subjectId, int gradeNumber) {
       return unitService.fetchUnitsBySubjectByGrade(subjectId, gradeNumber);
     }
//
//     @Override
//      public List<Unit> listAllUnits() {
//        return unitService.listAllUnits();
//      }
//
//     @Override
//     public void createUnit(String unitName, int subjectId, int gradeNumber) {
//       unitService.createUnit(unitName, subjectId, gradeNumber);
//     }
//
//     @Override
//     public void updateUnit(String oldName, String newName) {
//       unitService.updateUnit(oldName, newName);
//     }
//
//     public void deleteUnit(int id) {
//       unitService.deleteUnit(id);
//     }

}
```

DaysLessonsController.java

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

```java

import java.util.List;
import javax.validation.constraints.Pattern;
import org.hibernate.validator.constraints.Length;
import org.springframework.http.HttpStatus;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseStatus;
import io.swagger.v3.oas.annotations.OpenAPIDefinition;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.info.Info;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.servers.Server;
import staples.heather.planbook.Planbook;
import staples.heather.planbook.entity.Day;
import staples.heather.planbook.entity.Grade;
import staples.heather.planbook.entity.Lesson;

@Validated
@RequestMapping("/daysLessons")
@OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
    @Server(url = "http://localhost:8080", description = "Local server.")})

public interface DaysLessonsController {

  //@formatter:off
  @Operation(
      summary = "Returns a list of Lessons for the Day",
      description = "Returns a list of Lessons for the Day",
      responses = {
          @ApiResponse(
              responseCode = "200",
              description = "A list of lessons was returned",
              content = @Content(
                  mediaType = "application/json"
```

```
                    mediaType = "application/json",
                    schema = @Schema(implementation = Planbook.class))),
            @ApiResponse(
                responseCode = "400",
                description = "The requested parameters were invalid",
                content = @Content(mediaType = "application/json")),
            @ApiResponse(
                responseCode = "404",
                description = "No lessons were found with the input criteria",
                content = @Content(mediaType = "application/json")),
            @ApiResponse(
                responseCode = "500",
                description = "An unplanned error occurred",
                content = @Content(mediaType = "application/json"))
        },
        parameters = {
            @Parameter(
                name = "dayOfTheWeek",
                allowEmptyValue = false,
                required = true,
                description = "The day of the week (i.e., 'Monday')"),
        }
     )

    @GetMapping("/{dayOfTheWeek}")
    @ResponseStatus(code = HttpStatus.OK)
    List<Lesson> fetchLessonsByDay(
      @RequestParam(required = true)
      String dayOfTheWeek);


    @Operation(
         summary = "Returns a list of Lessons for the Week",
         description = "Returns a list of Lessons for the Week",
         responses = {
             @ApiResponse(
                 responseCode = "200",
                 description = "A list of lessons was returned",
                 content = @Content(
                     mediaType = "application/json",
                     schema = @Schema(implementation = Planbook.class))),
             @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 = "Unschedules a Lesson",
206         description = "Unschedules 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

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

import java.util.List;

@Validated
@RequestMapping("/lesson")
@OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
    @Server(url = "http://localhost:8080", description = "Local server.")})

public interface LessonController {

  //@formatter:off
  @Operation(
      summary = "Returns a Lesson",
      description = "Returns a Lesson",
      responses = {
          @ApiResponse(
              responseCode = "200",
              description = "A lesson was returned",
              content = @Content(
                  mediaType = "application/json",
                  schema = @Schema(implementation = Planbook.class))),
//          @ApiResponse(
//              responseCode = "400",
//              description = "The requested parameters were invalid",
//              content = @Content(mediaType = "application/json")),

          @ApiResponse(
              responseCode = "404",
              description = "No lesson was found with the input criteria",
              content = @Content(mediaType = "application/json")),
          @ApiResponse(
              responseCode = "500",
              description = "An unplanned error occurred",
              content = @Content(mediaType = "application/json"))
      },
      parameters = {
          @Parameter(
              name = "lessonId",
              allowEmptyValue = false,
              required = true,
              description = "The lesson id"),
      }
```

```
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
88              @ApiResponse(
89                  responseCode = "404",
90                  description = "No lessons were found with the input criteria",
91                  content = @Content(mediaType = "application/json")),
92              @ApiResponse(
93                  responseCode = "500",
94                  description = "An unplanned error occurred",
95                  content = @Content(mediaType = "application/json"))
96          },
97          parameters = {
98              @Parameter(
99                  name = "unitId",
100                 allowEmptyValue = false,
101                 required = true,
102                 description = "The unit id"),
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

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

import java.util.List;

@Validated
@RequestMapping("/subject")
@OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
    @Server(url = "http://localhost:8080", description = "Local server.")})

public interface SubjectController {
  public static final int NAME_MAX_LENGTH = 25;

  //@formatter:off

  @Operation(
       summary = "Returns a list of Subjects",
       description = "Returns a list of Subjects",
       responses = {
           @ApiResponse(
               responseCode = "200",
               description = "A list of subjects was returned",
               content = @Content(
                   mediaType = "application/json",
                   schema = @Schema(implementation = Planbook.class))),
           @ApiResponse(
               responseCode = "400",
               description = "The requested parameters were invalid",
               content = @Content(mediaType = "application/json")),
           @ApiResponse(
               responseCode = "404",
               description = "No subjects were found with the input criteria",
               content = @Content(mediaType = "application/json")),
           @ApiResponse(
               responseCode = "500",
               description = "An unplanned error occurred",
               content = @Content(mediaType = "application/json"))
       }
    )

  @ResponseStatus(code = HttpStatus.OK)
  @GetMapping("/all")
  List<Subject> listAllSubjects();

  //@formatter:on
}
```

*UnitController.java

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

import java.util.List;

@Validated
@RequestMapping("/unit")
@OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
    @Server(url = "http://localhost:8080", description = "Local server.")})

public interface UnitController {

  //@formatter:off
  @Operation(
      summary = "Returns a List of Units by Subject by Grade",
      description = "Returns a List of Units by Subject by Grade",
      responses = {
          @ApiResponse(
              responseCode = "200",
              description = "A list of units was returned",
              content = @Content(
                  mediaType = "application/json",
                  schema = @Schema(implementation = Planbook.class))),
          @ApiResponse(
              responseCode = "400",
              description = "The requested parameters were invalid",
              content = @Content(mediaType = "application/json")),
          @ApiResponse(
              responseCode = "404",
              description = "No unit was found with the input criteria",
              content = @Content(mediaType = "application/json")),
          @ApiResponse(
              responseCode = "500",
              description = "An unplanned error occurred",
              content = @Content(mediaType = "application/json"))
      },
      parameters = {
          @Parameter(
              name = "subjectId",
              allowEmptyValue = false,
              required = true,
              description = "The subject id"),
          @Parameter(
              name = "gradeNumber",
              allowEmptyValue = false,
              required = true,
              description = "The grade number"),
      }
  )

  @GetMapping
  @ResponseStatus(code = HttpStatus.OK)
  List<Unit> fetchUnitsBySubjectByGrade(@RequestParam(required = true) int subjectId, @RequestParam(required = true) int gradeNumber);

  //@formatter:on
}
```

DaysLessonsDao.java

```java
package staples.heather.planbook.dao;

import java.util.List;

public interface DaysLessonsDao {

  List<Day> listWeeksLessons();

  List<Lesson> fetchLessonsByDay(String dayOfTheWeek);

  void assignLessonsToDay(int dayId, int lessonId);

  void clearWeeksLessons();

  int deleteLessonFromDay(int lessonId);

  int updateDayOfLesson(int lessonId, int newDay);

  //void deleteLessonByDay(int dayId);
}

```

*DefaultDaysLessonsDao.java

```java
package staples.heather.planbook.dao;

import java.sql.ResultSet;

@Component
@Slf4j
public class DefaultDaysLessonsDao implements DaysLessonsDao {

  @Autowired
  private NamedParameterJdbcTemplate jdbcTemplate;

  //displays lessons for the week - READ
  @Override
  public List<Day> listWeeksLessons() {

    String sqlFetch = "SELECT * from day inner join daysLessons on day.dayId = daysLessons.dayId "
        + "INNER JOIN lesson on lesson.lessonId = daysLessons.lessonId group by day.dayId order by day.dayId";

    Map<String,Object> params = new HashMap<>();

    return jdbcTemplate.query(sqlFetch,  params, new RowMapper<>() {

    @Override
    public Day mapRow(ResultSet rs, int rowNum) throws SQLException {
      return Day.builder()
          .dayId(rs.getInt("dayId"))
          .dayOfTheWeek(rs.getString("dayOfTheWeek"))
          .lessons(fetchLessonsByDay(rs.getString("dayOfTheWeek")))
          .build();
    }});
   }

  //displays lessons for the day - READ
  @Override
  public List<Lesson> fetchLessonsByDay(String dayOfTheWeek) {
    String sqlFetch = "SELECT * FROM lesson INNER JOIN daysLessons on lesson.lessonId = daysLessons.lessonId "
        + "INNER JOIN day on day.dayId = daysLessons.dayId WHERE dayOfTheWeek = :dayOfTheWeek";

    Map<String,Object> params = new HashMap<>();
    params.put("dayOfTheWeek", dayOfTheWeek);

    return jdbcTemplate.query(sqlFetch,  params, new RowMapper<>() {
```

```
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 }
```

BasicSubjectController.java

```
1  package staples.heather.planbook.controller;
2
3  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
14
15
16
17                                                          Controller{
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  }
```

BasicUnitController.java

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

import java.util.List;


@RestController
@Slf4j
public class BasicUnitController implements UnitController{

    @Autowired
    private UnitService unitService;

     @Override
     public List<Unit> fetchUnitsBySubjectByGrade(int subjectId, int gradeNumber) {
       return unitService.fetchUnitsBySubjectByGrade(subjectId, gradeNumber);
     }
//
//    @Override
//     public List<Unit> listAllUnits() {
//       return unitService.listAllUnits();
//     }
//
//    @Override
//    public void createUnit(String unitName, int subjectId, int gradeNumber) {
//      unitService.createUnit(unitName, subjectId, gradeNumber);
//    }
//
//    @Override
//    public void updateUnit(String oldName, String newName) {
//      unitService.updateUnit(oldName, newName);
//    }
//
//    public void deleteUnit(int id) {
//      unitService.deleteUnit(id);
//    }

}
```

DaysLessonsController.java

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

import java.util.List;

@Validated
@RequestMapping("/daysLessons")
@OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
    @Server(url = "http://localhost:8080", description = "Local server.")})

public interface DaysLessonsController {

  //@formatter:off
  @Operation(
      summary = "Returns a list of Lessons for the Day",
      description = "Returns a list of Lessons for the Day",
      responses = {
          @ApiResponse(
              responseCode = "200",
              description = "A list of lessons was returned",
              content = @Content(
                  mediaType = "application/json",
                  schema = @Schema(implementation = Planbook.class))),
          @ApiResponse(
              responseCode = "400",
              description = "The requested parameters were invalid",
              content = @Content(mediaType = "application/json")),
          @ApiResponse(
              responseCode = "404",
              description = "No lessons were found with the input criteria",
              content = @Content(mediaType = "application/json")),
          @ApiResponse(
              responseCode = "500",
              description = "An unplanned error occurred",
              content = @Content(mediaType = "application/json"))
      },
      parameters = {
          @Parameter(
              name = "dayOfTheWeek",
              allowEmptyValue = false,
              required = true,
              description = "The day of the week (i.e., 'Monday')"),
      }
   )
```

```
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 = "Unschedules a Lesson",
206         description = "Unschedules 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

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

import java.util.List;

@Validated
@RequestMapping("/lesson")
@OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
    @Server(url = "http://localhost:8080", description = "Local server.")})

public interface LessonController {

  //@formatter:off
  @Operation(
      summary = "Returns a Lesson",
      description = "Returns a Lesson",
      responses = {
          @ApiResponse(
              responseCode = "200",
              description = "A lesson was returned",
              content = @Content(
                  mediaType = "application/json",
                  schema = @Schema(implementation = Planbook.class))),
          @ApiResponse(
              responseCode = "404",
              description = "No lesson was found with the input criteria",
              content = @Content(mediaType = "application/json")),
          @ApiResponse(
              responseCode = "500",
              description = "An unplanned error occurred",
              content = @Content(mediaType = "application/json"))
      },
      parameters = {
          @Parameter(
              name = "lessonId",
              allowEmptyValue = false,
              required = true,
              description = "The lesson id"),
      }
   )

  @GetMapping("/{lessonId}")
  @ResponseStatus(code = HttpStatus.OK)
  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
83              @ApiResponse(
84                  responseCode = "404",
85                  description = "No lessons were found with the input criteria",
86                  content = @Content(mediaType = "application/json")),
87              @ApiResponse(
88                  responseCode = "500",
89                  description = "An unplanned error occurred",
90                  content = @Content(mediaType = "application/json"))
91          },
92          parameters = {
93              @Parameter(
94                  name = "unitId",
95                  allowEmptyValue = false,
96                  required = true,
97                  description = "The unit id"),
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",
```

```
            allowEmptyValue = false,
            required = true,
            description = "The lesson name"),
        @Parameter(
            name = "objective",
            allowEmptyValue = false,
            required = true,
            description = "The lesson objective"),
        @Parameter(
            name = "content",
            allowEmptyValue = false,
            required = true,
            description = "The lesson content"),
        @Parameter(
            name = "unitId",
            allowEmptyValue = false,
            required = true,
            description = "The unit id")
    }
 )
@ResponseStatus(code = HttpStatus.CREATED)
@PostMapping
void createLesson(@RequestParam(required = true) String lessonName, @RequestParam(required = true) String objective,
    @RequestParam(required = true) String content, @RequestParam(required = true) int unitId);

@Operation(
    summary = "Updates Lesson Content",
    description = "Updates Lesson Content",
    responses = {
        @ApiResponse(
            responseCode = "200",
            description = "A lesson's content was updated",
            content = @Content(
                mediaType = "application/json",
                schema = @Schema(implementation = Planbook.class))),
        @ApiResponse(
            responseCode = "400",
            description = "The requested parameters were invalid",
            content = @Content(mediaType = "application/json")),
        @ApiResponse(
            responseCode = "404",
            description = "No lesson was found with the input criteria",
```

```
            content = @Content(mediaType = "application/json")),
        @ApiResponse(
            responseCode = "500",
            description = "An unplanned error occurred",
            content = @Content(mediaType = "application/json"))
    },
    parameters = {
        @Parameter(
            name = "lessonId",
            allowEmptyValue = false,
            required = true,
            description = "The lesson id"),
        @Parameter(
            name = "newContent",
            allowEmptyValue = false,
            required = true,
            description = "The new lesson content")
    }
)
@ResponseStatus(code = HttpStatus.OK)
@PutMapping
int updateLesson(@RequestParam(required = true) int lessonId, @RequestParam(required = true) String newContent);

@Operation(
    summary = "Deletes a Lesson",
    description = "Deletes a Lesson",
    responses = {
        @ApiResponse(
            responseCode = "200",
            description = "A lesson was deleted",
            content = @Content(
                mediaType = "application/json",
                schema = @Schema(implementation = Planbook.class))),
        @ApiResponse(
            responseCode = "400",
            description = "The requested parameters were invalid",
            content = @Content(mediaType = "application/json")),
        @ApiResponse(
            responseCode = "404",
            description = "No lesson was found with the input criteria",
            content = @Content(mediaType = "application/json")),
        @ApiResponse(
```

```
            responseCode = "500",
            description = "An unplanned error occurred",
            content = @Content(mediaType = "application/json"))
    },
    parameters = {
        @Parameter(
            name = "lessonId",
            allowEmptyValue = false,
            required = true,
            description = "The lesson id"),
    }
 )
@ResponseStatus(code = HttpStatus.OK)
@DeleteMapping
int deleteLesson(@RequestParam(required = true) int lessonId);

}


//@formatter:on

```

*SubjectController.java

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

import java.util.List;

@Validated
@RequestMapping("/subject")
@OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
    @Server(url = "http://localhost:8080", description = "Local server.")})

public interface SubjectController {

  //@formatter:off
  @Operation(
       summary = "Returns a list of Subjects",
       description = "Returns a list of Subjects",
       responses = {
           @ApiResponse(
               responseCode = "200",
               description = "A list of subjects was returned",
               content = @Content(
                   mediaType = "application/json",
                   schema = @Schema(implementation = Planbook.class))),
           @ApiResponse(
               responseCode = "400",
               description = "The requested parameters were invalid",
               content = @Content(mediaType = "application/json")),
           @ApiResponse(
               responseCode = "404",
               description = "No subjects were found with the input criteria",
               content = @Content(mediaType = "application/json")),
           @ApiResponse(
               responseCode = "500",
               description = "An unplanned error occurred",
               content = @Content(mediaType = "application/json"))
       }
    )

  @ResponseStatus(code = HttpStatus.OK)
  @GetMapping("/all")
  List<Subject> listAllSubjects();

  //@formatter:on
}
```

UnitController.java

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

import java.util.List;

@Validated
@RequestMapping("/unit")
@OpenAPIDefinition(info = @Info(title = "Planbook Service"), servers = {
    @Server(url = "http://localhost:8080", description = "Local server.")})

public interface UnitController {

  //@formatter:off
  @Operation(
      summary = "Returns a List of Units by Subject by Grade",
      description = "Returns a List of Units by Subject by Grade",
      responses = {
          @ApiResponse(
              responseCode = "200",
              description = "A list of units was returned",
              content = @Content(
                  mediaType = "application/json",
                  schema = @Schema(implementation = Planbook.class))),
          @ApiResponse(
              responseCode = "400",
              description = "The requested parameters were invalid",
              content = @Content(mediaType = "application/json")),
          @ApiResponse(
              responseCode = "404",
              description = "No unit was found with the input criteria",
              content = @Content(mediaType = "application/json")),
          @ApiResponse(
              responseCode = "500",
              description = "An unplanned error occurred",
              content = @Content(mediaType = "application/json"))
      },
      parameters = {
          @Parameter(
              name = "subjectId",
              allowEmptyValue = false,
              required = true,
              description = "The subject id"),
          @Parameter(
              name = "gradeNumber",
```

```
            allowEmptyValue = false,
            required = true,
            description = "The grade number"),
    }
  )

  @GetMapping
  @ResponseStatus(code = HttpStatus.OK)
  List<Unit> fetchUnitsBySubjectByGrade(@RequestParam(required = true) int subjectId, @RequestParam(required = true) int gradeNumber);

  //@formatter:on
}
```

DaysLessonsDao.java

```java
package staples.heather.planbook.dao;

import java.util.List;

public interface DaysLessonsDao {

  List<Day> listWeeksLessons();

  List<Lesson> fetchLessonsByDay(String dayOfTheWeek);

  void assignLessonsToDay(int dayId, int lessonId);

  void clearWeeksLessons();

  int deleteLessonFromDay(int lessonId);

  int updateDayOfLesson(int lessonId, int newDay);

  //void deleteLessonByDay(int dayId);
}

```

DefaultDaysLessonsDao.java

```java
package staples.heather.planbook.dao;

import java.sql.ResultSet;

@Component
@Slf4j
public class DefaultDaysLessonsDao implements DaysLessonsDao {

  @Autowired
  private NamedParameterJdbcTemplate jdbcTemplate;

  //displays lessons for the week - READ
  @Override
  public List<Day> listWeeksLessons() {

    String sqlFetch = "SELECT * from day inner join daysLessons on day.dayId = daysLessons.dayId "
        + "INNER JOIN lesson on lesson.lessonId = daysLessons.lessonId group by day.dayId order by day.dayId";

    Map<String,Object> params = new HashMap<>();

    return jdbcTemplate.query(sqlFetch,  params, new RowMapper<>() {

    @Override
    public Day mapRow(ResultSet rs, int rowNum) throws SQLException {
      return Day.builder()
          .dayId(rs.getInt("dayId"))
          .dayOfTheWeek(rs.getString("dayOfTheWeek"))
          .lessons(fetchLessonsByDay(rs.getString("dayOfTheWeek")))
          .build();
    }});
   }

  //displays lessons for the day - READ
  @Override
  public List<Lesson> fetchLessonsByDay(String dayOfTheWeek) {
    String sqlFetch = "SELECT * FROM lesson INNER JOIN daysLessons on lesson.lessonId = daysLessons.lessonId "
        + "INNER JOIN day on day.dayId = daysLessons.dayId WHERE dayOfTheWeek = :dayOfTheWeek";

    Map<String,Object> params = new HashMap<>();
    params.put("dayOfTheWeek", dayOfTheWeek);

    return jdbcTemplate.query(sqlFetch,  params, new RowMapper<>() {

```

```
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

```java
package staples.heather.planbook.dao;

import java.sql.ResultSet;

@Component
@Slf4j
public class DefaultLessonDao implements LessonDao {

  @Autowired
  private NamedParameterJdbcTemplate jdbcTemplate;

  public List<Lesson> listAllLessons() {

    String sqlFetch = "SELECT * FROM lesson";

    Map<String,Object> params = new HashMap<>();

    return jdbcTemplate.query(sqlFetch,  params, new RowMapper<>() {

    @Override
    public Lesson mapRow(ResultSet rs, int rowNum) throws SQLException {
      return Lesson.builder()
          .lessonId(rs.getInt("lessonId"))
          .lessonName(rs.getString("lessonName"))
          .objective(rs.getString("objective"))
          .content(rs.getString("content"))
          .unitId(rs.getInt("unitId"))
          .build();
    }});

    }

  @Override
  public List<Lesson> fetchLessonById(int lessonId) {

    String sqlFetch = "SELECT * FROM lesson WHERE lessonId = :lessonId";

    Map<String,Object> params = new HashMap<>();
    params.put("lessonId", lessonId);

    return jdbcTemplate.query(sqlFetch,  params, new RowMapper<>() {

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

*DefaultSubjectDao.java

```java
package staples.heather.planbook.dao;

import java.sql.ResultSet;

@Component
@Slf4j
public class DefaultSubjectDao implements SubjectDao {

  @Autowired
  private NamedParameterJdbcTemplate jdbcTemplate;


  @Override
  public List<Subject> listAllSubjects() {

    String sqlFetch = "SELECT * FROM subject";

    Map<String,Object> params = new HashMap<>();

    return jdbcTemplate.query(sqlFetch,  params, new RowMapper<>() {

      @Override
      public Subject mapRow(ResultSet rs, int rowNum) throws SQLException {
        return Subject.builder()
            .subjectId(rs.getInt("subjectId"))
            .subjectName(rs.getString("subjectName"))
            .build();
      }});
    }

}
```

*DefaultUnitDao.java

```java
package staples.heather.planbook.dao;

import java.sql.ResultSet;

@Component
@Slf4j
public class DefaultUnitDao implements UnitDao {

  @Autowired
  private NamedParameterJdbcTemplate jdbcTemplate;

  @Override
  public List<Unit> fetchUnitsBySubjectByGrade(int subjectId, int gradeNumber) {

    String sqlFetch = "SELECT * FROM unit WHERE subjectId = :subjectId AND gradeNumber = :gradeNumber";

    Map<String,Object> params = new HashMap<>();
    params.put("subjectId", subjectId);
    params.put("gradeNumber", gradeNumber);

    return jdbcTemplate.query(sqlFetch,  params, new RowMapper<>() {

      @Override
      public Unit mapRow(ResultSet rs, int rowNum) throws SQLException {
        return Unit.builder()
            .unitId(rs.getInt("unitId"))
            .unitName(rs.getString("unitName"))
            .subjectId(rs.getInt("subjectId"))
            .gradeNumber(rs.getInt("gradeNumber"))
            .build();
      }});
    }
}
```

LessonDao.java

```java
package staples.heather.planbook.dao;

import java.util.List;

public interface LessonDao {

  List<Lesson> fetchLessonById(int lessonId);

  List<Lesson> fetchLessonsByUnit(int unitId);

  List<Lesson> listAllLessons();

  void createLesson(String lessonName, String objective, String content, int unitId);

  int updateLesson(int lessonId, String newContent);

  int deleteLesson(int lessonId);
}
```

SubjectDao.java

```java
package staples.heather.planbook.dao;

import java.util.List;

public interface SubjectDao {

//  List<Subject> fetchSubject(String subjectName);

  List<Subject> listAllSubjects();

//  void createSubject(String subjectName);

//  void updateSubject(String oldName, String newName);

//  void deleteSubject(String subjectName);

}
```

UnitDao.java

Maximize

```java
kage staples.heather.planbook.dao;

import java.util.List;

public interface UnitDao {

  List<Unit> fetchUnitsBySubjectByGrade(int subjectId, int gradeNumber);

//  List<Unit> listAllUnits();
//
//  void deleteUnit(int id);
//
//  void createUnit(String unitName, int subjectId, int gradeNumber);
//
//  void updateUnit(String oldName, String newName);
}
```

Day.java

```java
package staples.heather.planbook.entity;

import java.util.List;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Day {
  private int dayId;
  private String dayOfTheWeek;
  private List<Lesson> lessons;

  @JsonIgnore
  public int getId() {
    return dayId;
  }

}
```

Day.java | DaysLessons.java

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

```java
import com.fasterxml.jackson.annotation.JsonIgnore;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class DaysLessons {
  private int daysLessonsId;
  private int dayId;
  private int lessonId;

//  @JsonIgnore
//  public int getId() {
//    return daysLessonsId;
//  }
}
```

Day.java DaysLessons.java Grade.java

```java
package staples.heather.planbook.entity;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Grade {
  private int gradeNumber;
  private String gradeName;


  @JsonIgnore
  public int getId() {
    return gradeNumber;
  }
}
```

Lesson.java

```java
package staples.heather.planbook.entity;

import org.springframework.data.annotation.Transient;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Lesson {
  private int lessonId;

  @Column(value="lessonName")
  private String lessonName;
  private String objective;
  private String content;
  private int unitId;

  @JsonIgnore
  public int getId() {
    return lessonId;
  }
}
```

Lesson.java Subject.java

```java
package staples.heather.planbook.entity;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Subject {
  private int subjectId;
  private String subjectName;

  @JsonIgnore
  public int getId() {
    return subjectId;
  }
}

```

Unit.java

```java
package staples.heather.planbook.entity;
```

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

```java
import com.fasterxml.jackson.annotation.JsonIgnore;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Unit {
  private int unitId;
  private String unitName;
  private int subjectId;
  private int gradeNumber;

  @JsonIgnore
  public int getId() {
    return unitId;
  }
}

```

GlobalErrorHandler.java

```java
package staples.heather.planbook.errorHandler;

import java.time.ZonedDateTime;

@RestControllerAdvice
@Slf4j
public class GlobalErrorHandler {
  private enum LogStatus {
    STACK_TRACE, MESSAGE_ONLY
    }

  @ExceptionHandler(ConstraintViolationException.class)
  @ResponseStatus(code = HttpStatus.BAD_REQUEST)
  public Map<String, Object> handleConstraintViolationException(ConstraintViolationException e, WebRequest webRequest) {
    return createExceptionMessage(e, HttpStatus.BAD_REQUEST, webRequest, LogStatus.MESSAGE_ONLY);
  }

  @ExceptionHandler(DataIntegrityViolationException.class)
  @ResponseStatus(code = HttpStatus.BAD_REQUEST)
  public Map<String, Object> handleConstraintViolationException(DataIntegrityViolationException e, WebRequest webRequest) {
    return createExceptionMessage(e, HttpStatus.BAD_REQUEST, webRequest, LogStatus.MESSAGE_ONLY);
  }


  @ExceptionHandler(Exception.class)
  @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
  public Map<String, Object> handleException(Exception e, WebRequest webRequest) {
    return createExceptionMessage(e, HttpStatus.INTERNAL_SERVER_ERROR, webRequest, LogStatus.MESSAGE_ONLY);
  }

  @ExceptionHandler(NoSuchElementException.class)
  @ResponseStatus(code = HttpStatus.NOT_FOUND)
  public Map<String, Object> handleNoSuchElementException(
    NoSuchElementException e, WebRequest webRequest) {
      return createExceptionMessage(e, HttpStatus.NOT_FOUND, webRequest, LogStatus.STACK_TRACE);
    }


  private Map<String, Object> createExceptionMessage(Exception e, HttpStatus status, WebRequest webRequest, LogStatus logStatus) {
    Map<String, Object> error = new HashMap<>();
    String timestamp = ZonedDateTime.now().format(DateTimeFormatter.RFC_1123_DATE_TIME);

    if(webRequest instanceof ServletWebRequest) {
      error.put("uri",  ((ServletWebRequest) webRequest).getRequest().getRequestURI());
    }

    error.put("message", e.toString());
    error.put("status code", status.value());
    error.put("timestamp", timestamp);
    error.put("reason", status.getReasonPhrase());

    if(logStatus == LogStatus.MESSAGE_ONLY) {
      log.error("Exception: {}", e.toString());
    }
    else {
      log.error("Exception: {}", e);
    }

    return error;
  }
}
```

DaysLessonsService.java

```java
package staples.heather.planbook.service;

import java.util.List;

public interface DaysLessonsService {

  List<Lesson> fetchLessonsByDay(String dayOfTheWeek);

  List<Day> listWeeksLessons();

  int updateDayOfLesson(int lessonId, int newDay);

  void clearWeeksLessons();

  int deleteLessonFromDay(int lessonId);

  void assignLessonsToDay(int dayId, int lessonId);



}
```

DaysLessonsService.java | DefaultDaysLessonsService.java

```java
package staples.heather.planbook.service;

import java.util.List;

@Service
@Slf4j
public class DefaultDaysLessonsService implements DaysLessonsService{

  @Autowired
  private DaysLessonsDao daysLessonsDao;


  @Override
  public List<Lesson> fetchLessonsByDay(String dayOfTheWeek) {
    List<Lesson> lessonsByDay = daysLessonsDao.fetchLessonsByDay(dayOfTheWeek);
    return lessonsByDay;
  }

  @Override
  public List<Day> listWeeksLessons() {
    List<Day> lessonsForWeek = daysLessonsDao.listWeeksLessons();
    return lessonsForWeek;
  }

  @Override
  public void assignLessonsToDay(int dayId, int lessonId) {
    daysLessonsDao.assignLessonsToDay(dayId, lessonId);
  }

  @Override
  public int updateDayOfLesson(int lessonId, int newDay) {
    return daysLessonsDao.updateDayOfLesson(lessonId, newDay);
  }

  public int deleteLessonFromDay(int lessonId) {
    return daysLessonsDao.deleteLessonFromDay(lessonId);
  }

  public void clearWeeksLessons() {
    daysLessonsDao.clearWeeksLessons();
  }

}
```

DaysLessonsService.java | DefaultDaysLessonsService.java | DefaultLessonService.java

```java
package staples.heather.planbook.service;

import java.util.List;

@Service
@Slf4j
public class DefaultLessonService implements LessonService {

  @Autowired
  private LessonDao lessonDao;

  @Override
  public List<Lesson> fetchLessonById(int lessonId) {
    List<Lesson> lesson = lessonDao.fetchLessonById(lessonId);
    return lesson;
  }

  @Override
  public List<Lesson> fetchLessonsByUnit(int unitId) {
    List<Lesson> lessons = lessonDao.fetchLessonsByUnit(unitId);
    return lessons;
  }

  @Override
  public List<Lesson> listAllLessons() {
    List<Lesson> lessons = lessonDao.listAllLessons();
    return lessons;
  }

  @Override
  public void createLesson(String lessonName, String objective, String content, int unitId) {
    lessonDao.createLesson(lessonName,  objective,  content, unitId);
  }

  @Override
  public int updateLesson(int id, String newContent) {
    return lessonDao.updateLesson(id, newContent);
  }

  public int deleteLesson(int id) {
    return lessonDao.deleteLesson(id);
  }
}
```

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

```java
package staples.heather.planbook.service;

import java.util.List;

@Service
@Slf4j
public class DefaultSubjectService implements SubjectService{


  @Autowired
  private SubjectDao subjectDao;

//  @Override
//  public List<Subject> fetchSubject(String subjectName) {
//    List<Subject> subject = subjectDao.fetchSubject(subjectName);
//    return subject;
//  }

  @Override
  public List<Subject> listAllSubjects() {
    List<Subject> subjects = subjectDao.listAllSubjects();
    return subjects;
  }

//  @Override
//  public void createSubject(String subjectName) {
//    subjectDao.createSubject(subjectName);
//  }

//  @Override
//  public void updateSubject(String oldName, String newName) {
//    subjectDao.updateSubject(oldName, newName);
//  }

//  @Override
//  public void deleteSubject(String subjectName) {
//    subjectDao.deleteSubject(subjectName);
//
//  }
}

```

Console | DefaultUnitService.java

```java
package staples.heather.planbook.service;

import java.util.List;

@Service
@Slf4j
public class DefaultUnitService implements UnitService{


  @Autowired
  private UnitDao unitDao;

  @Override
  public List<Unit> fetchUnitsBySubjectByGrade(int subjectId, int gradeNumber) {
    List<Unit> units = unitDao.fetchUnitsBySubjectByGrade(subjectId, gradeNumber);
    return units;
  }
//
//  @Override
//  public List<Unit> listAllUnits() {
//    List<Unit> units = unitDao.listAllUnits();
//    return units;
//  }
//
//  @Override
//  public void createUnit(String unitName, int subjectId, int gradeNumber) {
//    unitDao.createUnit(unitName,  subjectId, gradeNumber);
//  }
//
//  @Override
//  public void updateUnit(String oldName, String newName) {
//    unitDao.updateUnit(oldName,  newName);
//  }
//
//  @Override
//  public void deleteUnit(int id) {
//    unitDao.deleteUnit(id);
//  }
//
}
```

LessonService.java

```java
package staples.heather.planbook.service;

import java.util.List;

public interface LessonService {

  List<Lesson> fetchLessonById(int lessonId);

  List<Lesson> fetchLessonsByUnit(int unitId);

  List<Lesson> listAllLessons();

  void createLesson(String lessonName, String objective, String content, int unitId);

  int updateLesson(int lessonId, String newContent);

  int deleteLesson(int lessonId);
}
```

LessonService.java | SubjectService.java

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

```java
import java.util.List;

public interface SubjectService {

//  List<Subject> fetchSubject(String subjectName);

  List<Subject> listAllSubjects();

//  void createSubject(String subjectName);

//  void updateSubject(String oldName, String newName);

//  void deleteSubject(String subjectName);

}
```

UnitService.java

```java
package staples.heather.planbook.service;

import java.util.List;

public interface UnitService {

  List<Unit> fetchUnitsBySubjectByGrade(int subjectId, int gradeNumber);

//  List<Unit> listAllUnits();
//
//  void deleteUnit(int id);
//
//  void createUnit(String unitName, int subjectId, int gradeNumber);
//
//  void updateUnit(String oldName, String newName);
//
}

```