

Welcome to the Final Project for the OTC St. Paul Class.

Project Name: Block Buster Movies

Project Members:

Charlee Thao

Adam Fite

Andrew Cham

Project Contributions:

Group:

Database – Schema SQL File, Data Entry SQL File

Pom File

Component Scanner

Global Error Handler

Main Application

Final Screen Shot Documentation

Charlee Thao:

Movies: Controller, Dao, Entity, Service

Concessions: Controller, Dao, Entity, Service

Adam Fite:

Employees: Controller, Dao, Entity, Service

User Roles: Controller, Dao, Entity, Service

User Validation: Controller, Dao, Entity, Service

Transaction: Controller, Dao, Entity, Service

Transactions Test: Controller, Test Body, Base Test

Andrew Cham :

Stores: Controller, Dao, Entity, Service

Customers: Controller, Dao, Entity, Service

Thanks to everyone who may also have provided help trouble shooting when we ran into trouble.

Now On With The Show!

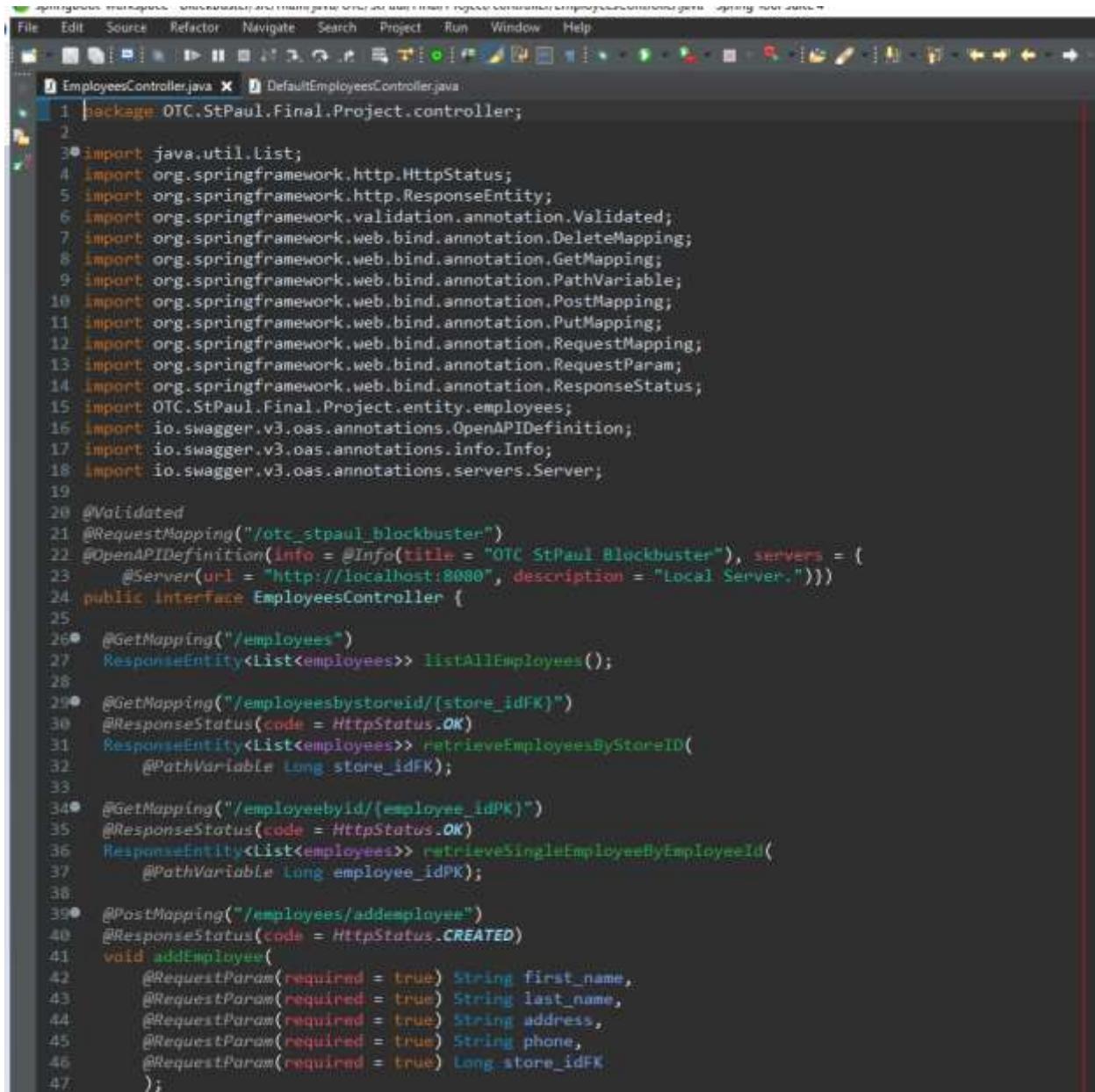
GitHub URL:

<https://github.com/cthao0502/Promineo/tree/main/Homework/WeekSpringBoot6>

Raw Code:

Controller:

Employees-



The screenshot shows a Java IDE interface with the title bar "SpringToolSuite - OTC.StPaul.Final.Project.controller" and a menu bar including File, Edit, Sources, Refactor, Navigate, Search, Project, Run, Window, Help. The main window displays the code for EmployeesController.java. The code is annotated with Swagger annotations for API documentation.

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4 import org.springframework.http.HttpStatus;
5 import org.springframework.http.ResponseEntity;
6 import org.springframework.validation.annotation.Validated;
7 import org.springframework.web.bind.annotation.DeleteMapping;
8 import org.springframework.web.bind.annotation.GetMapping;
9 import org.springframework.web.bind.annotation.PathVariable;
10 import org.springframework.web.bind.annotation.PostMapping;
11 import org.springframework.web.bind.annotation.PutMapping;
12 import org.springframework.web.bind.annotation.RequestMapping;
13 import org.springframework.web.bind.annotation.RequestParam;
14 import org.springframework.web.bind.annotation.ResponseStatus;
15 import OTC.StPaul.Final.Project.entity.employees;
16 import io.swagger.v3.oas.annotations.OpenAPIDefinition;
17 import io.swagger.v3.oas.annotations.info.Info;
18 import io.swagger.v3.oas.annotations.servers.Server;
19
20 @Validated
21 @RequestMapping("/otc_stpaul_blockbuster")
22 @OpenAPIDefinition(info = @Info(title = "OTC StPaul Blockbuster"), servers = {
23     @Server(url = "http://localhost:8080", description = "Local Server.")})
24 public interface EmployeesController {
25
26     @GetMapping("/employees")
27     ResponseEntity<List<employees>> listAllEmployees();
28
29     @GetMapping("/employeesbystoreid/{store_idFK}")
30     @ResponseStatus(code = HttpStatus.OK)
31     ResponseEntity<List<employees>> retrieveEmployeesByStoreID(
32         @PathVariable Long store_idFK);
33
34     @GetMapping("/employeebyid/{employee_idPK}")
35     @ResponseStatus(code = HttpStatus.OK)
36     ResponseEntity<List<employees>> retrieveSingleEmployeeByEmployeeId(
37         @PathVariable Long employee_idPK);
38
39     @PostMapping("/employees/addemployee")
40     @ResponseStatus(code = HttpStatus.CREATED)
41     void addEmployee(
42         @RequestParam(required = true) String first_name,
43         @RequestParam(required = true) String last_name,
44         @RequestParam(required = true) String address,
45         @RequestParam(required = true) String phone,
46         @RequestParam(required = true) Long store_idFK
47     );

```

```
48     @DeleteMapping("/deleteemployee/{employee_idPK}")
49●  @ResponseStatus(code = HttpStatus.OK)
50     void deleteEmployeeById(
51         @PathVariable Long employee_idPK);
53
54●  @PutMapping("/updateemployees/{first_name}")
55     @ResponseStatus(code = HttpStatus.OK)
56     void updateEmployeeFirstNameById(
57         @RequestParam(required = true) String first_name,
58         @RequestParam(required = true) Long employee_idPK);
59
60●  @PutMapping("/updateemployees/{last_name}")
61     @ResponseStatus(code = HttpStatus.OK)
62     void updateEmployeeLastNameById(
63         @RequestParam(required = true) String last_name,
64         @RequestParam(required = true) Long employee_idPK);
65
66●  @PutMapping("/updateemployees/{address}")
67     @ResponseStatus(code = HttpStatus.OK)
68     void updateEmployeeAddressById(
69         @RequestParam(required = true) String address,
70         @RequestParam(required = true) Long employee_idPK);
71
72●  @PutMapping("/updateemployees/{phone}")
73     @ResponseStatus(code = HttpStatus.OK)
74     void updateEmployeePhoneById(
75         @RequestParam(required = true) String phone,
76         @RequestParam(required = true) Long employee_idPK);
77
78 } // last bracket
79
```



springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/controller/DefaultEmployeesController.java - Spring Tool Suite 4

```
File Edit Source Refactor Navigate Search Project Run Window Help
```

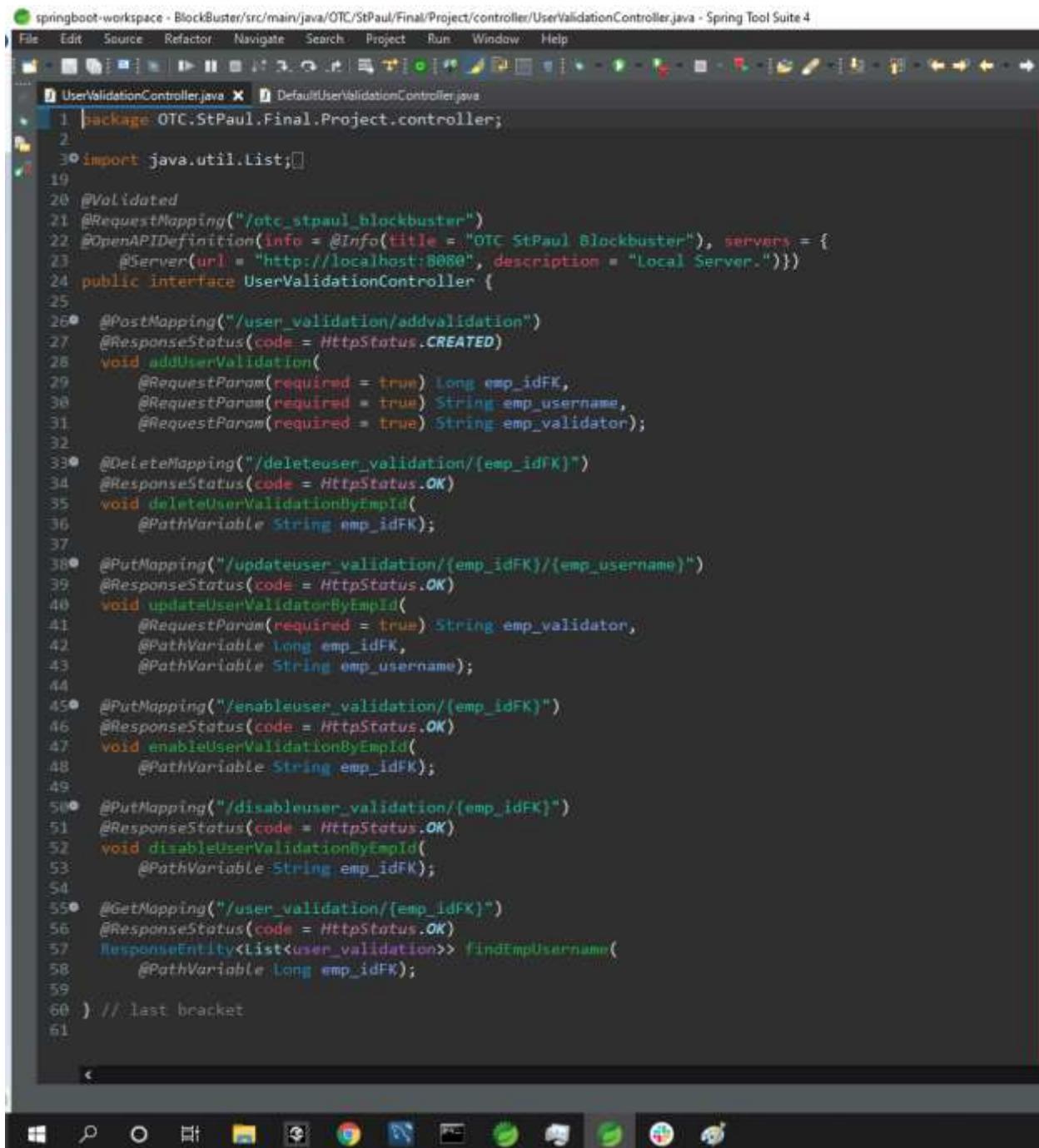
EmployeesController.java DefaultEmployeesController.java

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.http.HttpStatus;
6 import org.springframework.http.ResponseEntity;
7 import org.springframework.web.bind.annotation.RestController;
8 import OTC.StPaul.Final.Project.entity.employees;
9 import OTC.StPaul.Final.Project.service.EmployeesService;
10 import lombok.extern.slf4j.Slf4j;
11
12 @RestController
13 @Slf4j
14 public class DefaultEmployeesController implements EmployeesController {
15
16     @Autowired
17     private EmployeesService employeesService;
18
19     @Override
20     public ResponseEntity<List<employees>> listAllEmployees() {
21         log.info("A Request for the complete employees list was made");
22         List<employees> employees = employeesService.listAllEmployees();
23         return new ResponseEntity<List<employees>>(employees, HttpStatus.OK);
24     }
25
26     @Override
27     public ResponseEntity<List<employees>> retrieveEmployeesByStoreID(long store_idPK) {
28         log.info("A Request for a stores=() employees was made", store_idPK);
29         List<employees> storeEmployees = employeesService.retrieveEmployeesByStoreID(store_idPK);
30         return new ResponseEntity<List<employees>>(storeEmployees, HttpStatus.OK);
31     }
32
33     @Override
34     public ResponseEntity<List<employees>> retrieveSingleEmployeeByEmployeeId(long employee_idPK) {
35         log.info("A Request for a single employees=() information was made", employee_idPK);
36         List<employees> singleEmployee = employeesService.retrieveSingleEmployeeByEmployeeId(employee_idPK);
37         return new ResponseEntity<List<employees>>(singleEmployee, HttpStatus.OK);
38     }
39
40     @Override
41     public void addEmployee(String first_name, String last_name, String address, String phone,
42                             long store_idPK) {
43         log.info("Added an employee");
44         employeesService.addEmployee(first_name, last_name, address, phone, store_idPK);
45     }
```

```
45  
46    @Override  
47    public void deleteEmployeeById(Long employee_idPK) {  
48        log.info("Deleted an employee");  
49        employeesService.deleteEmployeeById(employee_idPK);  
50    }  
51  
52    @Override  
53    public void updateEmployeeFirstNameById(String first_name, Long employee_idPK) {  
54        log.info("Updated an employee's first name", employee_idPK);  
55        employeesService.updateEmployeeFirstNameById(first_name, employee_idPK);  
56    }  
57  
58    @Override  
59    public void updateEmployeeLastNameById(String last_name, Long employee_idPK) {  
60        log.info("Updated an employee's last name", employee_idPK);  
61        employeesService.updateEmployeeLastNameById(last_name, employee_idPK);  
62    }  
63  
64    @Override  
65    public void updateEmployeeAddressById(String address, Long employee_idPK) {  
66        log.info("Updated an employee's address", employee_idPK);  
67        employeesService.updateEmployeeAddressById(address, employee_idPK);  
68    }  
69  
70    @Override  
71    public void updateEmployeePhoneById(String phone, Long employee_idPK) {  
72        log.info("Updated an employee's phone number", employee_idPK);  
73        employeesService.updateEmployeePhoneById(phone, employee_idPK);  
74    }  
75  
76  
77 } // last bracket
```



User Validation-

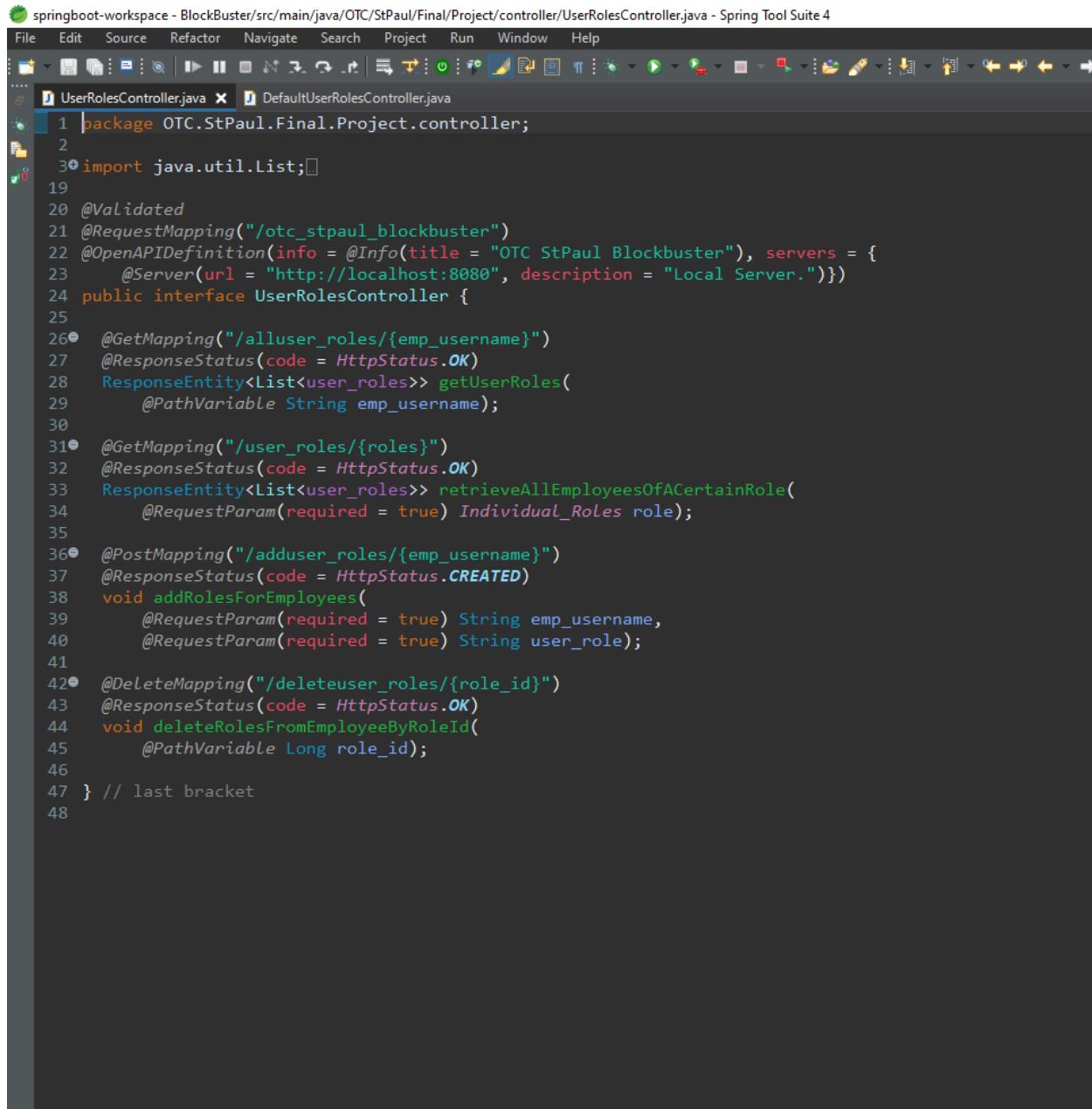


The screenshot shows the Spring Tool Suite 4 interface with the UserValidationController.java file open in the editor. The code implements a RESTful API for managing user validation records.

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4
5 @Validated
6 @RequestMapping("/otc_stpaul_blockbuster")
7 @OpenAPIDefinition(info = @Info(title = "OTC StPaul Blockbuster"), servers = {
8     @Server(url = "http://localhost:8080", description = "Local Server.")})
9 public interface UserValidationController {
10
11     @PostMapping("/user_validation/addvalidation")
12     @ResponseStatus(code = HttpStatus.CREATED)
13     void addUserValidation(
14         @RequestParam(required = true) Long emp_idFK,
15         @RequestParam(required = true) String emp_username,
16         @RequestParam(required = true) String emp_validator);
17
18     @DeleteMapping("/deleteuser_validation/{emp_idFK}")
19     @ResponseStatus(code = HttpStatus.OK)
20     void deleteUserValidationByEmpId(
21         @PathVariable String emp_idFK);
22
23     @PutMapping("/updateuser_validation/{emp_idFK}/{emp_username}")
24     @ResponseStatus(code = HttpStatus.OK)
25     void updateUserValidatorByEmpId(
26         @RequestParam(required = true) String emp_validator,
27         @PathVariable Long emp_idFK,
28         @PathVariable String emp_username);
29
30     @PutMapping("/enableuser_validation/{emp_idFK}")
31     @ResponseStatus(code = HttpStatus.OK)
32     void enableUserValidationByEmpId(
33         @PathVariable String emp_idFK);
34
35     @PutMapping("/disableuser_validation/{emp_idFK}")
36     @ResponseStatus(code = HttpStatus.OK)
37     void disableUserValidationByEmpId(
38         @PathVariable String emp_idFK);
39
40     @GetMapping("/user_validation/{emp_idFK}")
41     @ResponseStatus(code = HttpStatus.OK)
42     ResponseEntity<List<user_validation>> findEmpUsername(
43         @PathVariable Long emp_idFK);
44
45 } // last bracket
```

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.http.HttpStatus;
6 import org.springframework.http.ResponseEntity;
7 import org.springframework.web.bind.annotation.RestController;
8 import OTC.StPaul.Final.Project.entity.user_validation;
9 import OTC.StPaul.Final.Project.service.UserValidationService;
10 import lombok.extern.slf4j.Slf4j;
11
12 @RestController
13 @Slf4j
14 public class DefaultUserValidationController implements UserValidationController {
15
16     @Autowired
17     private UserValidationService userValidationService;
18
19     @Override
20     public void addUserValidation(Long emp_idFK, String emp_username, String emp_validator) {
21         log.info("Added a validation for an employee={}", emp_idFK);
22         userValidationService.addUserValidation(emp_idFK, emp_username, emp_validator);
23     }
24
25     @Override
26     public void deleteUserValidationByEmpId(String emp_idFK) {
27         log.info("Deleted the validation of an employee={}", emp_idFK);
28         userValidationService.deleteUserValidationByEmpId(emp_idFK);
29     }
30
31     @Override
32     public void updateUserValidatorByEmpId(String emp_validator, Long emp_idFK, String emp_username) {
33         log.info("Updated and employees={} validator", emp_idFK);
34         userValidationService.updateUserValidatorByEmpId(emp_validator, emp_idFK, emp_username);
35     }
36
37     @Override
38     public void enableUserValidationByEmpId(String emp_idFK) {
39         log.info("Enabled an employees={} validation", emp_idFK);
40         userValidationService.enableUserValidationByEmpId(emp_idFK);
41     }
42
43     @Override
44     public void disableUserValidationByEmpId(String emp_idFK) {
45         log.info("Disabled and employees={} validation", emp_idFK);
46         userValidationService.disableUserValidationByEmpId(emp_idFK);
47     }
48
49     @Override
50     public ResponseEntity<List<user_validation>> findEmpUsername(Long emp_idFK) {
51         log.info("A Request for an employees={} username was made", emp_idFK);
52         List<user_validation> userValidation = userValidationService.findEmpUsername(emp_idFK);
53         return new ResponseEntity<List<user_validation>>(userValidation, HttpStatus.OK);
54     }
55
56 } // last bracket
57
```

User Roles –



The screenshot shows the Spring Tool Suite 4 interface with the file `UserRolesController.java` open in the editor. The code defines a controller for managing user roles in a Spring Boot application.

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4
5 @Validated
6 @RequestMapping("/otc_stpaul_blockbuster")
7 @OpenAPIDefinition(info = @Info(title = "OTC StPaul Blockbuster"), servers = {
8     @Server(url = "http://localhost:8080", description = "Local Server.")})
9 public interface UserRolesController {
10
11     @GetMapping("/alluser_roles/{emp_username}")
12     @ResponseStatus(code = HttpStatus.OK)
13     ResponseEntity<List<user_roles>> getUserRoles(
14         @PathVariable String emp_username);
15
16     @GetMapping("/user_roles/{roles}")
17     @ResponseStatus(code = HttpStatus.OK)
18     ResponseEntity<List<user_roles>> retrieveAllEmployeesOfACertainRole(
19         @RequestParam(required = true) Individual_Roles role);
20
21     @PostMapping("/adduser_roles/{emp_username}")
22     @ResponseStatus(code = HttpStatus.CREATED)
23     void addRolesForEmployees(
24         @RequestParam(required = true) String emp_username,
25         @RequestParam(required = true) String user_role);
26
27     @DeleteMapping("/deleteuser_roles/{role_id}")
28     @ResponseStatus(code = HttpStatus.OK)
29     void deleteRolesFromEmployeeByRoleId(
30         @PathVariable Long role_id);
31
32 } // last bracket
```

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4
5 @RestController
6 @Slf4j
7 public class DefaultUserRolesController implements UserRolesController{
8
9     @Autowired
10    private UserRolesService userRolesService;
11
12    @Override
13    public ResponseEntity<List<user_roles>> getUserRoles(String emp_username) {
14        log.info("A Request for an employees={}" + emp_username);
15        List<user_roles> userRoles = userRolesService.getUserRoles(emp_username);
16        return new ResponseEntity<List<user_roles>>(userRoles, HttpStatus.OK);
17    }
18
19    @Override
20    public ResponseEntity<List<user_roles>> retrieveAllEmployeesOfACertainRole(
21        Individual_Roles role) {
22        log.info("A Request for an employees={} role was made", role);
23        List<user_roles> userRoles = userRolesService.retrieveAllEmployeesOfACertainRole(role);
24        return new ResponseEntity<List<user_roles>>(userRoles, HttpStatus.OK);
25    }
26
27    @Override
28    public void addRolesForEmployees(String emp_username, String user_role) {
29        log.info("Added a user role for an employee={}", emp_username);
30        userRolesService.addRolesForEmployees(emp_username, user_role);
31    }
32
33    @Override
34    public void deleteRolesFromEmployeeByRoleId(Long role_id) {
35        log.info("Delete a user role from an employee");
36        userRolesService.deleteRolesFromEmployeeByRoleId(role_id);
37    }
38}
```

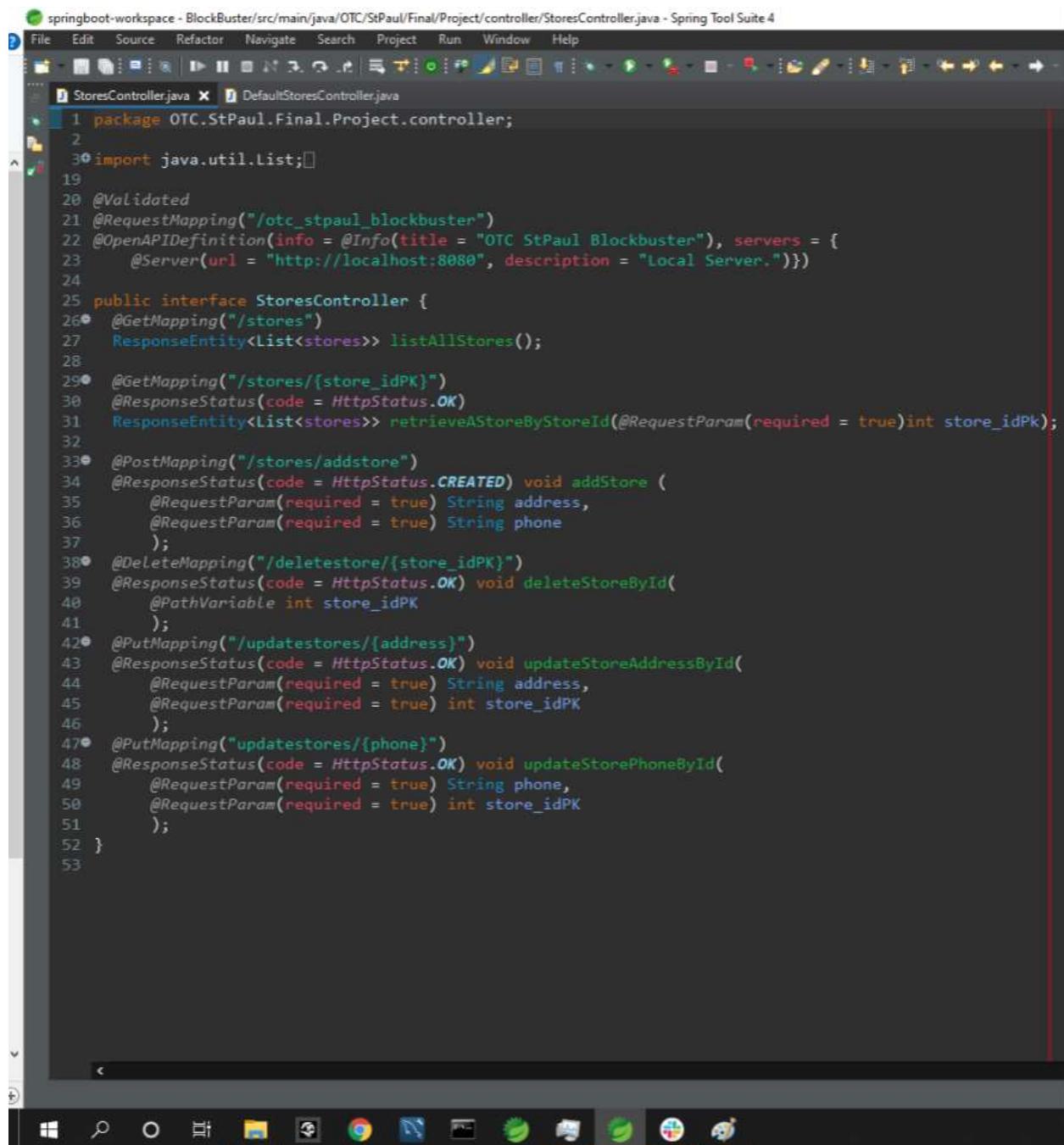
Customers –

The screenshot shows the Spring Tool Suite IDE with the file 'CustomerController.java' open. The code implements a RESTful API for managing customers. It includes methods for getting all customers, retrieving a customer by ID, adding a new customer, deleting a customer by ID, updating a customer's first name, updating a customer's last name, updating a customer's address, and updating a customer's phone number. The code uses annotations like @RequestMapping and @ResponseStatus to define the API endpoints and their behaviors.

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4
5 import org.springframework.web.bind.annotation.*;
6
7 @RestController
8 @RequestMapping("/etc_stpaul_blockbuster")
9 @ServerDefinition(info = @Info(title = "OTC StPaul Blockbuster"), servers = {
10     @Server(url = "http://localhost:8080", description = "Local Server.")})
11
12 public interface CustomersController {
13     @GetMapping("/customers") @ResponseStatus(code = HttpStatus.OK) List<Customer> list();
14
15     @GetMapping("/{customer_id}")
16     @ResponseStatus(code = HttpStatus.OK)
17     @ResponseBody List<Customer> retrieveCustomerByIdCustomer(@RequestParam(required = true) int customer_id);
18
19     @GetMapping("/{customer(first_name, last_name)}")
20     @ResponseStatus(code = HttpStatus.OK)
21     @ResponseBody List<Customer> retrieveCustomerByCustomerName(@RequestParam(required = true) String first_name, String last_name);
22
23     @PostMapping("/addCustomer")
24     @ResponseStatus(code = HttpStatus.CREATED) void addCustomer(
25         @RequestParam(required = true) String first_name,
26         @RequestParam(required = true) String last_name,
27         @RequestParam(required = true) String address,
28         @RequestParam(required = true) String phone
29     );
30
31     @DeleteMapping("/deleteCustomer/{customer_id}")
32     @ResponseStatus(code = HttpStatus.OK) void deleteCustomerById(
33         @RequestParam(required = true) int customer_id
34     );
35
36     @PutMapping("/updateCustomer/{first_name}")
37     @ResponseStatus(code = HttpStatus.OK) void updateCustomerFirstName(
38         @RequestParam(required = true) String first_name,
39         @RequestParam(required = true) int customer_id
40     );
41
42     @PutMapping("/updateCustomer/{last_name}")
43     @ResponseStatus(code = HttpStatus.OK) void updateCustomerLastName(
44         @RequestParam(required = true) String last_name,
45         @RequestParam(required = true) int customer_id
46     );
47
48     @PutMapping("/updateCustomer/{address}")
49     @ResponseStatus(code = HttpStatus.OK) void updateCustomerAddress(
50         @RequestParam(required = true) String address,
51         @RequestParam(required = true) int customer_id
52     );
53
54     @PutMapping("/updateCustomer/{phone}")
55     @ResponseStatus(code = HttpStatus.OK) void updateCustomerPhoneById(
56         @RequestParam(required = true) String phone,
57         @RequestParam(required = true) int customer_id);
58
59
60 } // last bracket
61
62
63
64
65
66 } // last bracket
67
```

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4
5 @RestController
6 @Slf4j
7 public class DefaultUserRolesController implements UserRolesController{
8
9     @Autowired
10    private UserRolesService userRolesService;
11
12    @Override
13    public ResponseEntity<List<user_roles>> getUserRoles(String emp_username) {
14        log.info("A Request for an employees={}" + emp_username);
15        List<user_roles> userRoles = userRolesService.getUserRoles(emp_username);
16        return new ResponseEntity<List<user_roles>>(userRoles, HttpStatus.OK);
17    }
18
19    @Override
20    public ResponseEntity<List<user_roles>> retrieveAllEmployeesOfACertainRole(
21        Individual_Roles role) {
22        log.info("A Request for an employees={} role was made", role);
23        List<user_roles> userRoles = userRolesService.retrieveAllEmployeesOfACertainRole(role);
24        return new ResponseEntity<List<user_roles>>(userRoles, HttpStatus.OK);
25    }
26
27    @Override
28    public void addRolesForEmployees(String emp_username, String user_role) {
29        log.info("Added a user role for an employee={}", emp_username);
30        userRolesService.addRolesForEmployees(emp_username, user_role);
31    }
32
33    @Override
34    public void deleteRolesFromEmployeeByRoleId(Long role_id) {
35        log.info("Delete a user role from an employee");
36        userRolesService.deleteRolesFromEmployeeByRoleId(role_id);
37    }
38}
```

Stores-



The screenshot shows the Spring Tool Suite 4 interface with the file `StoresController.java` open in the editor. The code defines a REST controller for managing stores. It includes methods for listing all stores, retrieving a store by ID, adding a new store, deleting a store, updating a store's address, and updating a store's phone number. The code uses annotations like `@Validated`, `@RequestMapping`, `@GetMapping`, `@PostMapping`, `@DeleteMapping`, `@PutMapping`, and `@ResponseStatus` to define the API endpoints and their behaviors.

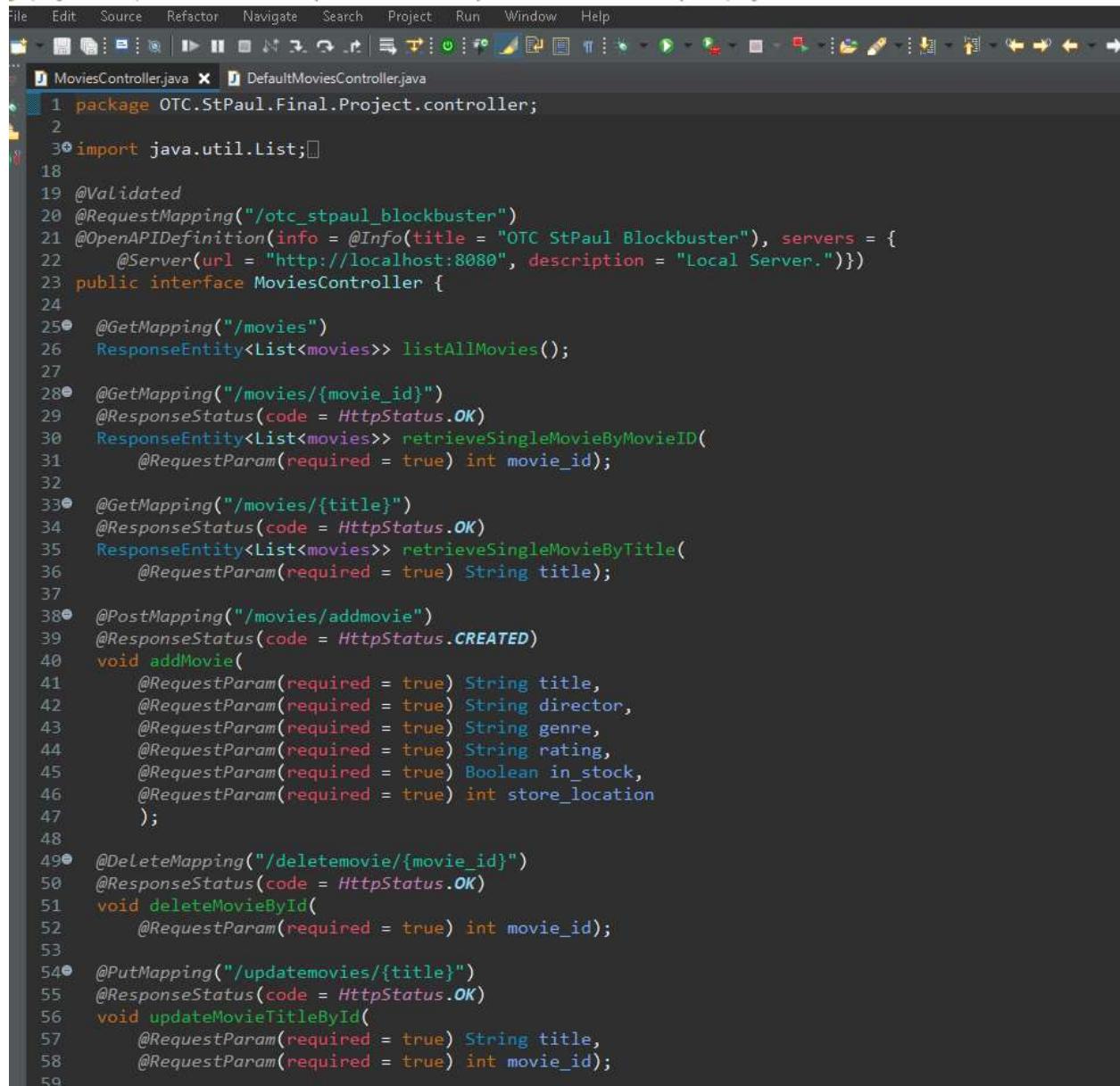
```
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4
5 @Validated
6 @RequestMapping("/otc_stpaul_blockbuster")
7 @OpenAPIDefinition(info = @Info(title = "OTC StPaul Blockbuster"), servers = {
8     @Server(url = "http://localhost:8080", description = "Local Server.")})
9
10 public interface StoresController {
11     @GetMapping("/stores")
12     ResponseEntity<List<stores>> listAllStores();
13
14     @GetMapping("/stores/{store_idPK}")
15     @ResponseStatus(code = HttpStatus.OK)
16     ResponseEntity<List<stores>> retrieveAStoreByStoreId(@RequestParam(required = true)int store_idPK);
17
18     @PostMapping("/stores/addstore")
19     @ResponseStatus(code = HttpStatus.CREATED) void addStore(
20         @RequestParam(required = true) String address,
21         @RequestParam(required = true) String phone
22     );
23
24     @DeleteMapping("/deletestore/{store_idPK}")
25     @ResponseStatus(code = HttpStatus.OK) void deleteStoreById(
26         @PathVariable int store_idPK
27     );
28
29     @PutMapping("/updatestores/{address}")
30     @ResponseStatus(code = HttpStatus.OK) void updateStoreAddressById(
31         @RequestParam(required = true) String address,
32         @RequestParam(required = true) int store_idPK
33     );
34
35     @PutMapping("/updatestores/{phone}")
36     @ResponseStatus(code = HttpStatus.OK) void updateStorePhoneById(
37         @RequestParam(required = true) String phone,
38         @RequestParam(required = true) int store_idPK
39     );
40 }
41
42 }
```

The screenshot shows a Java IDE interface with the following details:

- File Menu:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Java development toolbar with icons for file operations, search, and navigation.
- Project Explorer:** Shows two files: StoresController.java and DefaultStoresController.java.
- Code Editor:** Displays the content of DefaultStoresController.java. The code implements the StoresController interface, using @RestController and @Slf4j annotations. It contains methods for listing all stores, retrieving a store by ID, adding a store, deleting a store, updating store address, and updating store phone number. Each method includes logging via Log.info and interacts with a StoresService.

```
File Edit Source Refactor Navigate Search Project Run Window Help
StoresController.java DefaultStoresController.java
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4
5 @RestController
6 @Slf4j
7 public class DefaultStoresController implements StoresController {
8
9     @Autowired
10    private StoresService storesService;
11
12    @Override
13    public ResponseEntity<List<stores>> listAllStores() {
14        Log.info("A Request was made for all stores");
15        List<stores> stores = storesService.listAllStores();
16        return new ResponseEntity<List<stores>>(stores, HttpStatus.OK);
17    }
18
19    @Override
20    public ResponseEntity<List<stores>> retrieveAStoreByStoreId(int store_idPk) {
21        Log.info("A Request was made to list a store", store_idPk);
22        List<stores> singleStore = storesService.retrieveAStoreByStoreId(store_idPk);
23        return new ResponseEntity<List<stores>>(singleStore, HttpStatus.OK);
24    }
25
26    @Override
27    public void addStore(String address, String phone) {
28        Log.info("Added a Store");
29        storesService.addStore(address, phone);
30    }
31
32    @Override
33    public void deleteStoreById(int store_idPK) {
34        Log.info("Deleted a store");
35        storesService.deleteStoreById(store_idPK);
36    }
37
38    @Override
39    public void updateStoreAddressById(String address, int store_idPK) {
40        Log.info("Updated an Address");
41        storesService.updateAddressByStoreId(address, store_idPK);
42    }
43
44    @Override
45    public void updateStorePhoneById(String phone, int store_idPK) {
46        Log.info("Updated phone");
47        storesService.updatePhoneByStoreId(phone, store_idPK);
48    }
49
50 } // last bracket
51
52
53
54
55
56
57
58
59
60
61
62
```

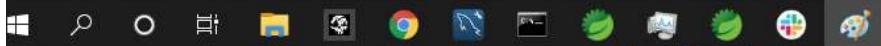
Movies-



The screenshot shows a Java code editor with the title bar "Movies-". Below the title bar, there is a toolbar with various icons. The main area displays the code for the `MoviesController.java` file. The code is annotated with numbers on the left side, likely indicating line numbers or code segments. The code itself is a Spring REST controller for managing movies.

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4
5 @Validated
6 @RequestMapping("/otc_stpaul_blockbuster")
7 @OpenAPIDefinition(info = @Info(title = "OTC StPaul Blockbuster"), servers = {
8     @Server(url = "http://localhost:8080", description = "Local Server.")})
9 public interface MoviesController {
10
11     @GetMapping("/movies")
12     ResponseEntity<List<movies>> listAllMovies();
13
14     @GetMapping("/movies/{movie_id}")
15     @ResponseStatus(code = HttpStatus.OK)
16     ResponseEntity<List<movies>> retrieveSingleMovieByMovieID(
17         @RequestParam(required = true) int movie_id);
18
19     @GetMapping("/movies/{title}")
20     @ResponseStatus(code = HttpStatus.OK)
21     ResponseEntity<List<movies>> retrieveSingleMovieByTitle(
22         @RequestParam(required = true) String title);
23
24     @PostMapping("/movies/addmovie")
25     @ResponseStatus(code = HttpStatus.CREATED)
26     void addMovie(
27         @RequestParam(required = true) String title,
28         @RequestParam(required = true) String director,
29         @RequestParam(required = true) String genre,
30         @RequestParam(required = true) String rating,
31         @RequestParam(required = true) Boolean in_stock,
32         @RequestParam(required = true) int store_location
33     );
34
35     @DeleteMapping("/deletemovie/{movie_id}")
36     @ResponseStatus(code = HttpStatus.OK)
37     void deleteMovieById(
38         @RequestParam(required = true) int movie_id);
39
40     @PutMapping("/updatemovies/{title}")
41     @ResponseStatus(code = HttpStatus.OK)
42     void updateMovieTitleById(
43         @RequestParam(required = true) String title,
44         @RequestParam(required = true) int movie_id);
45
```

```
60@  @PutMapping("/updatemovies/{director}")
61  @ResponseStatus(code = HttpStatus.OK)
62  void updateMovieDirectorById(
63      @RequestParam(required = true) String director,
64      @RequestParam(required = true) int movie_id);
65
66@  @PutMapping("/updatemovies/{genre}")
67  @ResponseStatus(code = HttpStatus.OK)
68  void updateMovieGenreById(
69      @RequestParam(required = true) String genre,
70      @RequestParam(required = true) int movie_id);
71
72@  @PutMapping("/updatemovies/{rating}")
73  @ResponseStatus(code = HttpStatus.OK)
74  void updateMovieRatingById(
75      @RequestParam(required = true) String rating,
76      @RequestParam(required = true) int movie_id);
77
78@  @PutMapping("/updatemovies/{in_stock}")
79  @ResponseStatus(code = HttpStatus.OK)
80  void updateMovieStockById(
81      @RequestParam(required = true) Boolean in_stock,
82      @RequestParam(required = true) int movie_id);
83
84@  @PutMapping("/updatemovies/{store_location}")
85  @ResponseStatus(code = HttpStatus.OK)
86  void updateMovieLocationById(
87      @RequestParam(required = true) int store_location,
88      @RequestParam(required = true) int movie_id);
89
90 } // last bracket
91 <
```



springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/controller/DefaultMoviesController.java - Spring Tool Suite 4

```
File Edit Source Refactor Navigate Search Project Run Window Help
```

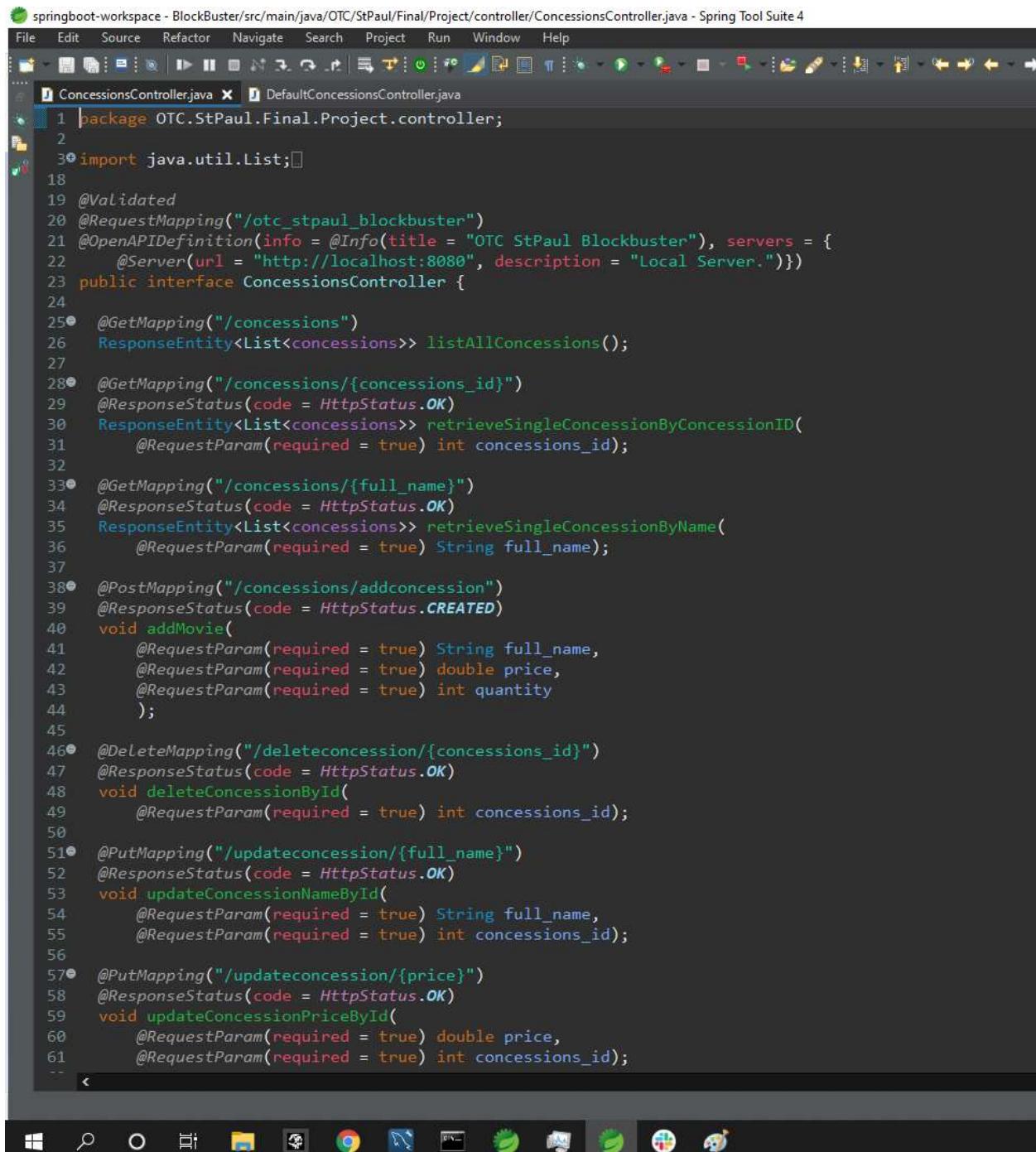
MoviesController.java DefaultMoviesController.java X

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4
5 @RestController
6 @Slf4j
7 public class DefaultMoviesController implements MoviesController {
8
9     @Autowired
10    private MoviesService moviesService;
11
12    @Override
13    public ResponseEntity<List<movies>> listAllMovies() {
14        log.info("A Request for the complete movies list was made");
15        List<movies> movies = moviesService.listAllMovies();
16        return new ResponseEntity<List<movies>>(movies, HttpStatus.OK);
17    }
18
19    @Override
20    public ResponseEntity<List<movies>> retrieveSingleMovieByMovieID(int movie_id) {
21        log.info("A Request for a single movies={} information was made", movie_id);
22        List<movies> singleMovie = moviesService.retrieveSingleMovieByMovieID(movie_id);
23        return new ResponseEntity<List<movies>>(singleMovie, HttpStatus.OK);
24    }
25
26    @Override
27    public ResponseEntity<List<movies>> retrieveSingleMovieByTitle(String title) {
28        log.info("A Request for a single movies={} title information was made", title);
29        List<movies> singleMovieTitle = moviesService.retrieveSingleMovieByTitle(title);
30        return new ResponseEntity<List<movies>>(singleMovieTitle, HttpStatus.OK);
31    }
32
33    @Override
34    public void addMovie(String title, String director, String genre, String rating, Boolean in_stock,
35        int store_location) {
36        log.info("Added a movie");
37        moviesService.addMovie(title, director, genre, rating, in_stock, store_location);
38    }
39
40    @Override
41    public void deleteMovieById(int movie_id) {
42        log.info("Deleted a movie");
43        moviesService.deleteMovieById(movie_id);
44    }
45}
```

```
53● @Override
54  public void updateMovieTitleById(String title, int movie_id) {
55      Log.info("Update a movies={} title", movie_id);
56      moviesService.updateMovieTitleById(title, movie_id);
57  }
58
59● @Override
60  public void updateMovieDirectorById(String director, int movie_id) {
61      Log.info("Update a movies={} director", movie_id);
62      moviesService.updateMovieDirectorById(director, movie_id);
63  }
64
65● @Override
66  public void updateMovieGenreById(String genre, int movie_id) {
67      Log.info("Update a movies={} genre", movie_id);
68      moviesService.updateMovieGenreById(genre, movie_id);
69  }
70
71● @Override
72  public void updateMovieRatingById(String rating, int movie_id) {
73      Log.info("Update a movies={} rating", movie_id);
74      moviesService.updateMovieRatingById(rating, movie_id);
75  }
76
77● @Override
78  public void updateMovieStockById(Boolean in_stock, int movie_id) {
79      Log.info("Update a movies={} stock", movie_id);
80      moviesService.updateMovieStockById(in_stock, movie_id);
81  }
82
83● @Override
84  public void updateMovieLocationById(int store_location, int movie_id) {
85      Log.info("Update a movies={} store location number", movie_id);
86      moviesService.updateMovieLocationById(store_location, movie_id);
87  }
88
89 } // last bracket
90 <
```



Concessions-



The screenshot shows the Spring Tool Suite 4 interface with the file `ConcessionsController.java` open. The code defines a REST controller for concessions. It includes methods for listing all concessions, retrieving a single concession by ID, retrieving a concession by full name, adding a new concession, deleting a concession, updating a concession's name, and updating a concession's price.

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4
5 @Validated
6 @RequestMapping("/otc_stpaul_blockbuster")
7 @OpenAPIDefinition(info = @Info(title = "OTC StPaul Blockbuster"), servers = {
8     @Server(url = "http://localhost:8080", description = "Local Server.")})
9 public interface ConcessionsController {
10
11     @GetMapping("/concessions")
12     ResponseEntity<List<concessions>> listAllConcessions();
13
14     @GetMapping("/concessions/{concessions_id}")
15     @ResponseStatus(code = HttpStatus.OK)
16     ResponseEntity<List<concessions>> retrieveSingleConcessionByConcessionID(
17         @RequestParam(required = true) int concessions_id);
18
19     @GetMapping("/concessions/{full_name}")
20     @ResponseStatus(code = HttpStatus.OK)
21     ResponseEntity<List<concessions>> retrieveSingleConcessionByName(
22         @RequestParam(required = true) String full_name);
23
24     @PostMapping("/concessions/addconcession")
25     @ResponseStatus(code = HttpStatus.CREATED)
26     void addMovie(
27         @RequestParam(required = true) String full_name,
28         @RequestParam(required = true) double price,
29         @RequestParam(required = true) int quantity
30     );
31
32     @DeleteMapping("/deleteconcession/{concessions_id}")
33     @ResponseStatus(code = HttpStatus.OK)
34     void deleteConcessionById(
35         @RequestParam(required = true) int concessions_id);
36
37     @PutMapping("/updateconcession/{full_name}")
38     @ResponseStatus(code = HttpStatus.OK)
39     void updateConcessionNameById(
40         @RequestParam(required = true) String full_name,
41         @RequestParam(required = true) int concessions_id);
42
43     @PutMapping("/updateconcession/{price}")
44     @ResponseStatus(code = HttpStatus.OK)
45     void updateConcessionPriceById(
46         @RequestParam(required = true) double price,
47         @RequestParam(required = true) int concessions_id);
48 }
```

```
62     @PutMapping("/updateconcession/{quantity}")
63     @ResponseStatus(code = HttpStatus.OK)
64     void updateConcessionQuantityById(
65         @RequestParam(required = true) int quantity,
66         @RequestParam(required = true) int concessions_id);
67
68 }
69 } // last bracket
70
```

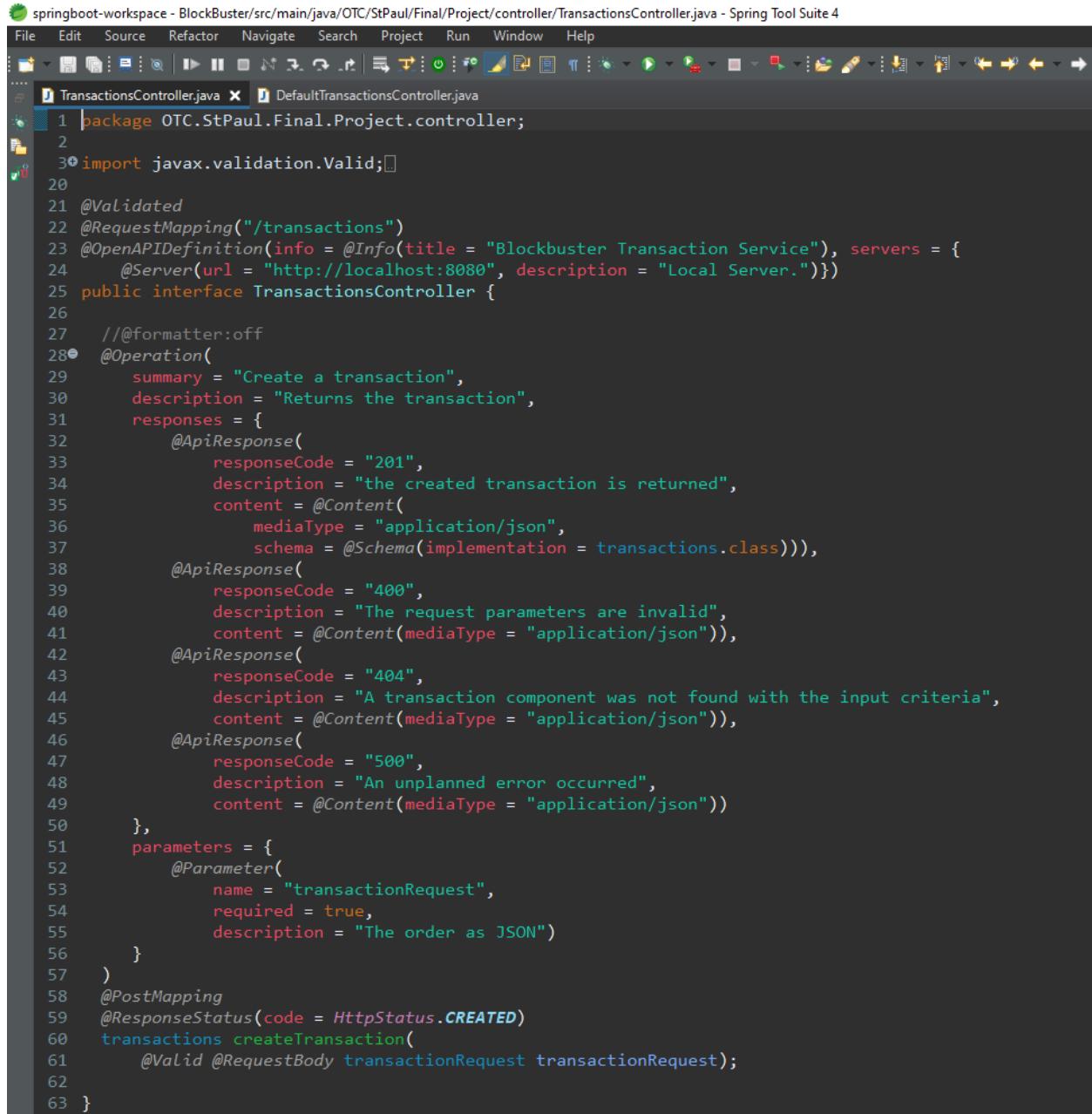
The screenshot shows a Java IDE interface with two tabs open: 'ConcessionsController.java' and 'DefaultConcessionsController.java'. The 'DefaultConcessionsController.java' tab is active, displaying the following code:

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import java.util.List;
4
5 @RestController
6 @Slf4j
7 public class DefaultConcessionsController implements ConcessionsController {
8
9     @Autowired
10    private ConcessionsService concessionsService;
11
12    @Override
13    public ResponseEntity<List<concessions>> listAllConcessions() {
14        Log.info("A Request for the complete concessions list was made");
15        List<concessions> concessions = concessionsService.listAllConcessions();
16        return new ResponseEntity<List<concessions>>(concessions, HttpStatus.OK);
17    }
18
19    @Override
20    public ResponseEntity<List<concessions>> retrieveSingleConcessionByConcessionID(
21        int concessions_id) {
22        Log.info("A Request for a single concessions={} information was made", concessions_id);
23        List<concessions> singleConcession = concessionsService
24            .retrieveSingleConcessionByConcessionID(concessions_id);
25        return new ResponseEntity<List<concessions>>(singleConcession, HttpStatus.OK);
26    }
27
28    @Override
29    public ResponseEntity<List<concessions>> retrieveSingleConcessionByName(String full_name) {
30        Log.info("A Request for a single concessions={} name information was made", full_name);
31        List<concessions> singleConcessionName = concessionsService
32            .retrieveSingleConcessionByName(full_name);
33        return new ResponseEntity<List<concessions>>(singleConcessionName, HttpStatus.OK);
34    }
35
36    @Override
37    public void addMovie(String full_name, double price, int quantity) {
38        Log.info("Added a concession");
39        concessionsService.addConcession(full_name, price, quantity);
40    }
41
42    @Override
43    public void deleteConcessionById(int concessions_id) {
44        Log.info("Deleted a concession");
45        concessionsService.deleteConcessionById(concessions_id);
46    }
47
48
49}
```

```
54
55● @Override
▲56 public void updateConcessionNameById(String full_name, int concessions_id) {
57     log.info("Update a concessions={} full_name", concessions_id);
58     concessionsService.updateConcessionNameById(full_name, concessions_id);
59 }
60
61● @Override
▲62 public void updateConcessionPriceById(double price, int concessions_id) {
63     log.info("Update a concessions={} price", concessions_id);
64     concessionsService.updateConcessionPriceById(price, concessions_id);
65 }
66
67● @Override
▲68 public void updateConcessionQuantityById(int quantity, int concessions_id) {
69     log.info("Update a concessions={} quantity", concessions_id);
70     concessionsService.updateConcessionQuantityById(quantity, concessions_id);
71 }
72
73 } // last bracket
74
```



Transactions-



The screenshot shows the Spring Tool Suite 4 interface with the TransactionsController.java file open in the editor. The code defines a REST endpoint for creating transactions using the OpenAPI specification.

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import javax.validation.Valid;
4
5 @Validated
6 @RequestMapping("/transactions")
7 @OpenAPIDefinition(info = @Info(title = "Blockbuster Transaction Service"), servers = {
8     @Server(url = "http://localhost:8080", description = "Local Server.")})
9 public interface TransactionsController {
10
11     //@@formatter:off
12     @Operation(
13         summary = "Create a transaction",
14         description = "Returns the transaction",
15         responses = {
16             @ApiResponse(
17                 responseCode = "201",
18                 description = "the created transaction is returned",
19                 content = @Content(
20                     mediaType = "application/json",
21                     schema = @Schema(implementation = transactions.class))),
22             @ApiResponse(
23                 responseCode = "400",
24                 description = "The request parameters are invalid",
25                 content = @Content(mediaType = "application/json")),
26             @ApiResponse(
27                 responseCode = "404",
28                 description = "A transaction component was not found with the input criteria",
29                 content = @Content(mediaType = "application/json")),
30             @ApiResponse(
31                 responseCode = "500",
32                 description = "An unplanned error occurred",
33                 content = @Content(mediaType = "application/json"))
34         },
35         parameters = {
36             @Parameter(
37                 name = "transactionRequest",
38                 required = true,
39                 description = "The order as JSON")
40         }
41     )
42     @PostMapping
43     @ResponseStatus(code = HttpStatus.CREATED)
44     transactions createTransaction(
45         @Valid @RequestBody transactionRequest transactionRequest);
46
47 }
```

The screenshot shows a Java development environment with the following details:

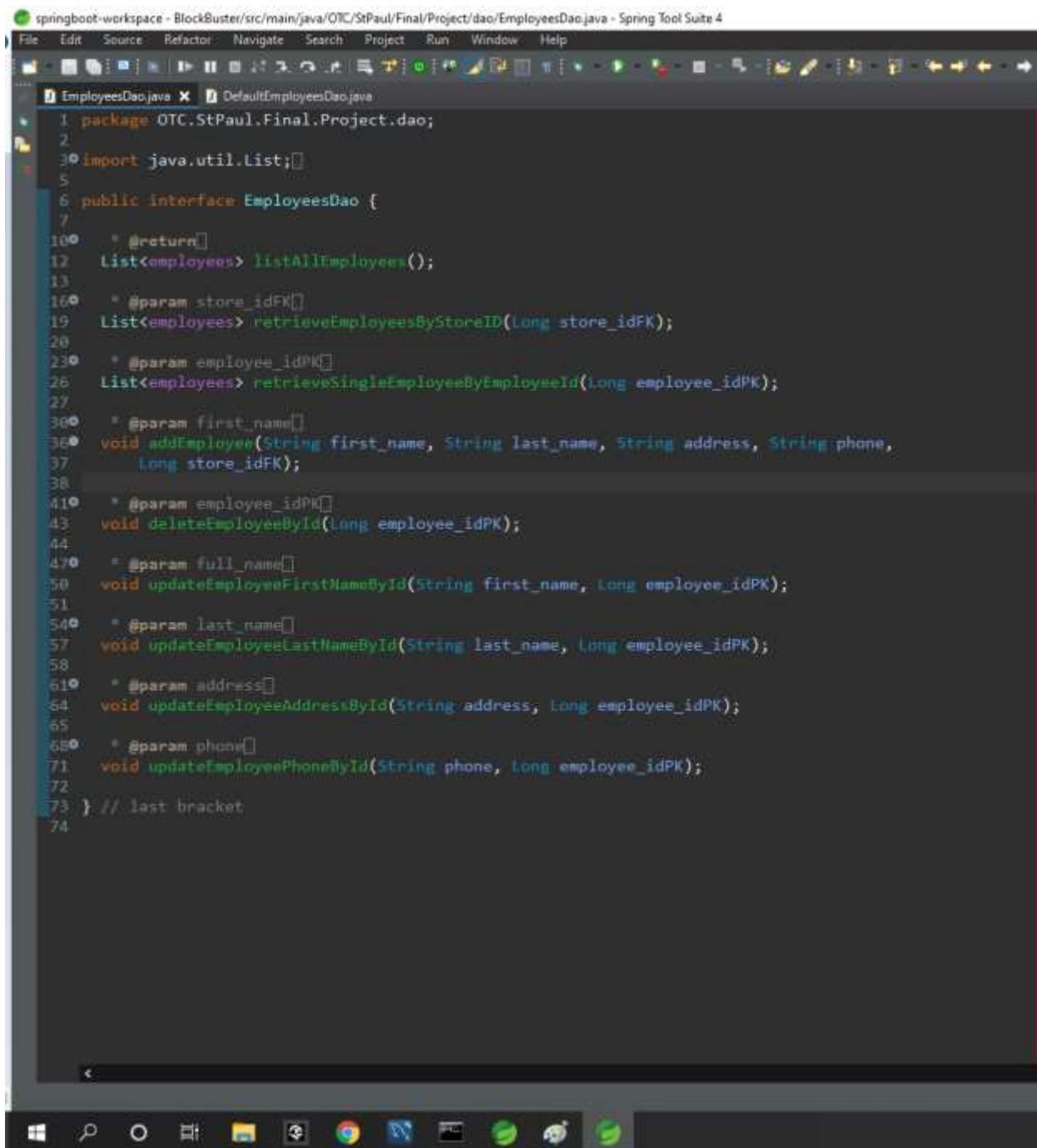
- File Menu:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Java IDE toolbar with icons for file operations, search, and project navigation.
- Project Explorer:** Shows two files: `TransactionsController.java` and `DefaultTransactionsController.java`.
- Code Editor:** Displays the content of `DefaultTransactionsController.java`. The code is as follows:

```
1 package OTC.StPaul.Final.Project.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @RestController
6 @Slf4j
7 public class DefaultTransactionsController implements TransactionsController {
8
9     @Autowired
10    private TransactionsService transactionsService;
11
12    public transactions createTransaction(transactionRequest transactionRequest) {
13        log.debug("Transaction={}", transactionRequest);
14        return transactionsService.createTransaction(transactionRequest);
15    }
16
17 } // last bracket
18
19
20 }
```

The code uses Java 8 features like `RestController`, `@Slf4j`, and `transactionRequest` which is likely a typo for `TransactionRequest`. It also uses the `transactions` type, which is another likely typo for `Transaction`.

Dao:

Employees-



The screenshot shows the Spring Tool Suite 4 interface with the EmployeesDao.java file open in the editor. The code defines a public interface EmployeesDao with various methods for managing employees. The methods include listing all employees, retrieving employees by store ID, retrieving a single employee by employee ID, adding a new employee, deleting an employee by ID, updating an employee's first name by ID, updating an employee's last name by ID, updating an employee's address by ID, and updating an employee's phone number by ID. The code uses annotations such as @param and @return to describe the parameters and return types of the methods.

```
1 package OTC.StPaul.Final.Project.dao;
2
3 import java.util.List;
4
5 public interface EmployeesDao {
6
7     * @return
8     List<employees> listAllEmployees();
9
10    * @param store_idPK
11    List<employees> retrieveEmployeesByStoreID(Long store_idPK);
12
13    * @param employee_idPK
14    List<employees> retrieveSingleEmployeeByEmployeeId(Long employee_idPK);
15
16    * @param first_name
17    void addEmployee(String first_name, String last_name, String address, String phone,
18                      Long store_idPK);
19
20    * @param employee_idPK
21    void deleteEmployeeById(Long employee_idPK);
22
23    * @param full_name
24    void updateEmployeeFirstNameById(String first_name, long employee_idPK);
25
26    * @param last_name
27    void updateEmployeeLastNameById(String last_name, long employee_idPK);
28
29    * @param address
30    void updateEmployeeAddressById(String address, long employee_idPK);
31
32    * @param phone
33    void updateEmployeePhoneById(String phone, long employee_idPK);
34
35 } // last bracket
36
```

The screenshot shows the Spring Tool Suite 4 interface with the code editor open. The title bar reads "springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/dao/DefaultEmployeesDao.java - Spring Tool Suite 4". The menu bar includes Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Find, and Run. The code editor displays Java code for a DAO class:

```
1 package OTC.StPaul.Final.Project.dao;
2
3 import java.sql.ResultSet;
4
5
6 @Component
7 @Service
8 @Transactional
9 public class DefaultEmployeesDao implements EmployeesDao {
10
11     @Autowired
12     private NamedParameterJdbcTemplate jdbcTemplate;
13
14     // returns a list of all employees in database (from every store)
15     @Override
16     public List<Employees> listAllEmployees() {
17
18         // @formatter:off
19         String sql = ""
20             + "SELECT * "
21             + "FROM employees";
22         // @formatter:on
23
24         return jdbcTemplate.query(sql, new ResultSetExtractor<>() {
25             @Override
26             public List<Employees> extractData(ResultSet rs) throws SQLException, DataAccessException {
27                 if (rs.next()) {
28                     // @formatter:off
29                     return List.of(employees.builder()
30                         .employee_idPK(rs.getLong("employee_idPK"))
31                         .first_name(rs.getString("first_name"))
32                         .last_name(rs.getString("last_name"))
33                         .address(rs.getString("address"))
34                         .phone(rs.getString("phone"))
35                         .store_idFK(rs.getLong("store_idFK"))
36                         .build());
37                     // @formatter:on
38                 }
39                 return extractData(null);
40             }
41         });
42     }
43 }
```

```
58     // returns all employees for a specific store using the store id number
59     @Override
60     public List<employees> retrieveEmployeesByStoreID(Long store_idFK) {
61
62         // @formatter:off
63         String sql = ""
64             + "SELECT * "
65             + "FROM employees "
66             + "WHERE store_idFK = :store_idFK";
67         // @formatter:on
68
69         Map<String, Object> params = new HashMap<>();
70         params.put("store_idFK", store_idFK);
71
72     return jdbcTemplate.query(sql, params, new ResultSetExtractor<>() {
73         @Override
74         public List<employees> extractData(ResultSet rs) throws SQLException, DataAccessException {
75             if (rs.next()) {
76                 // @formatter:off
77                 return List.of(employees.builder()
78                     .employee_idPK(rs.getLong("employee_idPK"))
79                     .first_name(rs.getString("first_name"))
80                     .last_name(rs.getString("last_name"))
81                     .address(rs.getString("address"))
82                     .phone(rs.getString("phone"))
83                     .store_idFK(rs.getLong("store_idFK"))
84                     .build());
85                 // @formatter:on
86             }
87
88             return extractData(null);
89         }
90     });
91 }
92
93     // returns a single employee by by employee id number
94     @Override
95     public List<employees> retrieveSingleEmployeeByEmployeeId(Long employee_idPK) {
96
97         // @formatter:off
98         String sql = ""
99             + "SELECT * "
100            + "FROM employees "
101            + "WHERE employee_idPK = :employee_idPK";
102        // @formatter:on
```

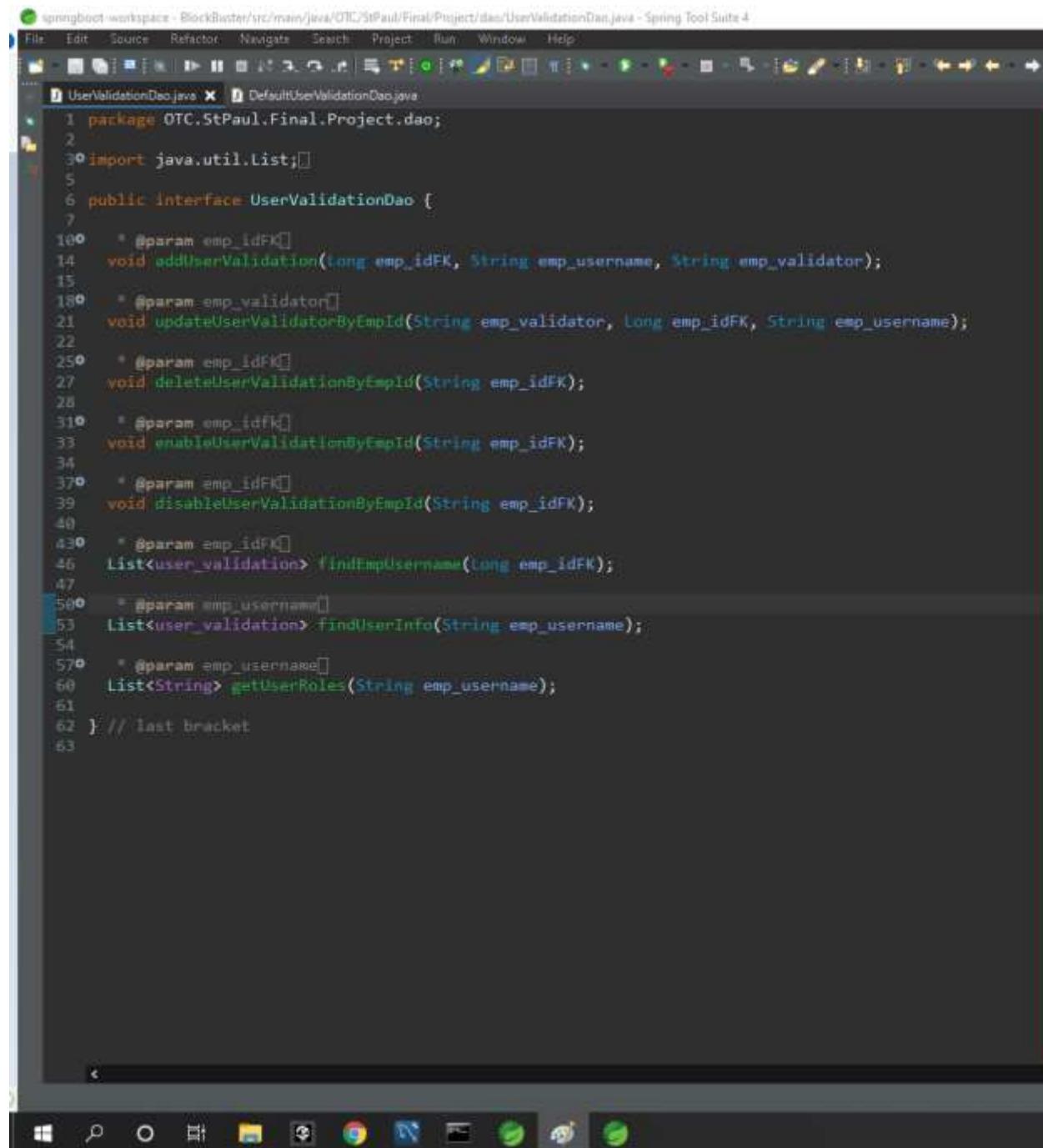
```
103
104     Map<String, Object> params = new HashMap<>();
105     params.put("employee_idPK", employee_idPK);
106
107     return jdbcTemplate.query(sql, params, new RowMapper<>() {
108
109         @Override
110         public employees mapRow(ResultSet rs, int rowNum) throws SQLException {
111
112             // @formatter:off
113             return employees.builder()
114                 .employee_idPK(rs.getLong("employee_idPK"))
115                 .first_name(rs.getString("first_name"))
116                 .last_name(rs.getString("last_name"))
117                 .address(rs.getString("address"))
118                 .phone(rs.getString("phone"))
119                 .store_idFK(rs.getLong("store_idFK"))
120                 .build();
121             // @formatter:on
122         }
123     });
124 }
125
126 // adds an employee to the database
127 @Override
128 public void addEmployee(String first_name, String last_name, String address, String phone,
129     Long store_idFK) {
130     // @formatter:off
131     String sql = ""
132         + "INSERT INTO employees (" +
133         + "first_name, last_name, address, phone, store_idFK"
134         + ") VALUES (" +
135         + ":first_name, :last_name, :address, :phone, :store_idFK"
136         + ")";
137     // @formatter:on
138     Map<String, Object> params = new HashMap<>();
139
140     params.put("first_name", first_name);
141     params.put("last_name", last_name);
142     params.put("address", address);
143     params.put("phone", phone);
144     params.put("store_idFK", store_idFK);
145
146     jdbcTemplate.update(sql, params);
147 }
```

```
148 // deletes an employee by employee id number
▲150● public void deleteEmployeeById(Long employee_idPK) {
151
152     // @formatter:off
153     String sql = ""
154         + "DELETE FROM employees "
155         + "WHERE employee_idPK = :employee_idPK";
156     // @formatter:on
157
158     Map<String, Object> params = new HashMap<>();
159     params.put("employee_idPK", employee_idPK);
160
161     jdbcTemplate.update(sql, params);
162 }
163
164 // updates an employees first name by employee id number
▲165● public void updateEmployeeFirstNameById(String first_name, Long employee_idPK) {
166
167     // @formatter:off
168     String sql = ""
169         + "UPDATE employees "
170         + "SET first_name = :first_name "
171         + "WHERE employee_idPK = :employee_idPK";
172     // @formatter:on
173
174     Map<String, Object> params = new HashMap<>();
175     params.put("first_name", first_name);
176     params.put("employee_idPK", employee_idPK);
177
178     jdbcTemplate.update(sql, params);
179 }
180
181 // updates an employees last name by employee id number
▲182● public void updateEmployeeLastNameById(String last_name, Long employee_idPK) {
183
184     // @formatter:off
185     String sql = ""
186         + "UPDATE employees "
187         + "SET last_name = :last_name "
188         + "WHERE employee_idPK = :employee_idPK";
189     // @formatter:on
190
191     Map<String, Object> params = new HashMap<>();
192     params.put("last_name", last_name);
```

```
193     params.put("employee_idPK", employee_idPK);
194
195     jdbcTemplate.update(sql, params);
196 }
197
198 // updates an employees address by employee id number
199 public void updateEmployeeAddressById(String address, Long employee_idPK) {
200
201     // @formatter:off
202     String sql = ""
203         + "UPDATE employees "
204         + "SET address = :address "
205         + "WHERE employee_idPK = :employee_idPK";
206     // @formatter:on
207
208     Map<String, Object> params = new HashMap<>();
209     params.put("address", address);
210     params.put("employee_idPK", employee_idPK);
211
212     jdbcTemplate.update(sql, params);
213 }
214
215 // updates an employees phone number by employee id number
216 public void updateEmployeePhoneById(String phone, Long employee_idPK) {
217
218     // @formatter:off
219     String sql = ""
220         + "UPDATE employees "
221         + "SET phone = :phone "
222         + "WHERE employee_idPK = :employee_idPK";
223     // @formatter:on
224
225     Map<String, Object> params = new HashMap<>();
226     params.put("phone", phone);
227     params.put("employee_idPK", employee_idPK);
228
229     jdbcTemplate.update(sql, params);
230 }
231
232 } // last bracket
```



User Validation-



The screenshot shows the Spring Tool Suite 4 interface with the file `UserValidationDao.java` open in the editor. The code defines a DAO interface for user validation operations.

```
1 package OTC.StPaul.Final.Project.dao;
2
3 import java.util.List;
4
5 public interface UserValidationDao {
6
7     * @param emp_idFK
8     void addUserValidation(Long emp_idFK, String emp_username, String emp_validator);
9
10    * @param emp_validator
11    void updateUserValidatorByEmpId(String emp_validator, Long emp_idFK, String emp_username);
12
13    * @param emp_idFK
14    void deleteUserValidationByEmpId(String emp_idFK);
15
16    * @param emp_idFK
17    void enableUserValidationByEmpId(String emp_idFK);
18
19    * @param emp_idFK
20    void disableUserValidationByEmpId(String emp_idFK);
21
22    * @param emp_idFK
23    List<user_validation> findEmpUsername(Long emp_idFK);
24
25    * @param emp_username
26    List<user_validation> findUserInfo(String emp_username);
27
28    * @param emp_username
29    List<String> getUserRoles(String emp_username);
30
31 }
32 // last bracket
33
```

springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/dao/DefaultUserValidationDao.java - Spring Tool Suite 4

```
File Edit Source Refactor Navigate Search Project Run Window Help

UserValidationDao.java DefaultUserValidationDao.java

1 package OTC.StPaul.Final.Project.dao;
2
3 import java.sql.ResultSet;
4
5
6 @Component
7 @Service
8 @Transactional
9 public class DefaultUserValidationDao extends JdbcDaoSupport implements UserValidationDao {
10
11     @Autowired
12     private NamedParameterJdbcTemplate jdbcTemplate;
13
14     public DefaultUserValidationDao(DataSource dataSource) {
15         this.setDataSource(dataSource);
16     }
17
18     // adds login credentials for a user
19     @Override
20     public void addUserValidation(long emp_idFK, String emp_username, String emp_validator) {
21
22         // @formatter:off
23         String sql = "";
24         + "INSERT INTO user_validation (" +
25             + "emp_idFK, emp_username, emp_validator"
26             + ") VALUES (" +
27             + ":emp_idFK, :emp_username, :emp_validator"
28             + ")";
29         // @formatter:on
30
31         Map<String, Object> params = new HashMap<>();
32         params.put("emp_idFK", emp_idFK);
33         params.put("emp_username", emp_username);
34         params.put("emp_validator", emp_validator);
35
36         jdbcTemplate.update(sql, params);
37     }
38
39     // delete a users login validation by employee id number
40     @Override
41     public void deleteUserValidationByEmpId(String emp_idFK) {
42
43         // @formatter:off
44         String sql = "";
45         + "DELETE FROM user_validation "
46         + "WHERE emp_idFK = :emp_idFK";
47         // @formatter:on
48
49         Map<String, Object> params = new HashMap<>();
50     }
51 }
```

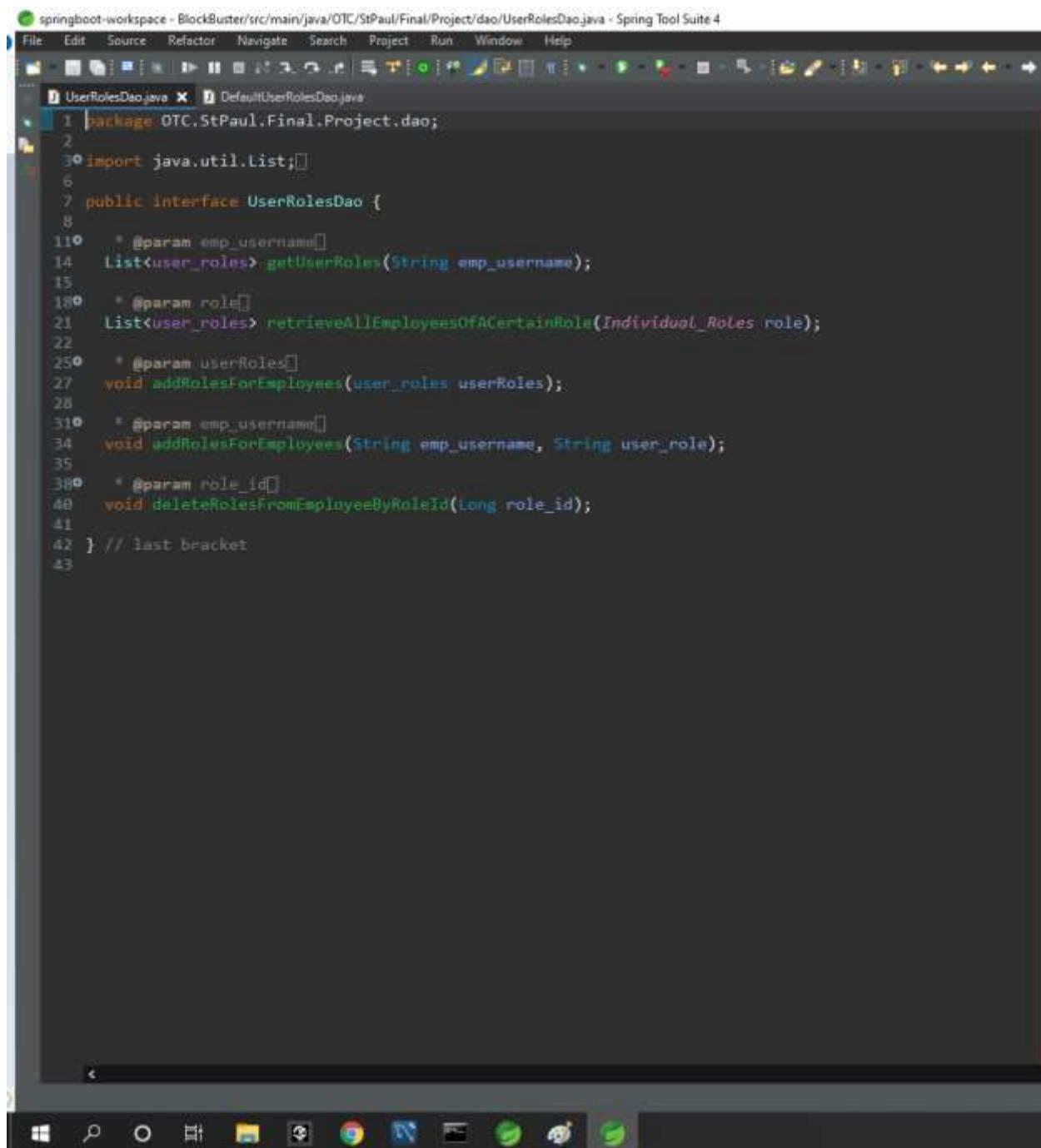
```
62     params.put("emp_idFK", emp_idFK);
63
64     jdbcTemplate.update(sql, params);
65 }
66
67 // updates an employees emp_validator (password) by employee id number & username
68 @Override
▲ 69 public void updateUserValidatorByEmpId(String emp_validator, Long emp_idFK, String emp_username) {
70
71     // @formatter:off
72     String sql = ""
73         + "UPDATE user_validation "
74         + "SET emp_validator = :emp_validator "
75         + "WHERE emp_idFK = :emp_idFK "
76         + "AND emp_username = :emp_username";
77     // @formatter:on
78
79     Map<String, Object> params = new HashMap<>();
80     params.put("emp_validator", emp_validator);
81     params.put("emp_idFK", emp_idFK);
82     params.put("emp_username", emp_username);
83
84     jdbcTemplate.update(sql, params);
85 }
86
87 // Enables a Users Validation
88 @Override
▲ 89 public void enableUserValidationByEmpId(String emp_idFK) {
90
91     // @formatter:off
92     String sql = ""
93         + "Update user_validation "
94         + "SET ENABLED = true "
95         + "WHERE emp_idFK = :emp_idFK";
96     // @formatter:on
97
98     Map<String, Object> params = new HashMap<>();
99     params.put("emp_idFK", emp_idFK);
100
101     jdbcTemplate.update(sql, params);
102 }
103
104 // Disables a Users Validation
105 @Override
▲ 106 public void disableUserValidationByEmpId(String emp_idFK) {
```

```
107     // @formatter:off
108     String sql = ""
109         + "Update user_validation "
110         + "SET ENABLED = false"
111         + "WHERE emp_idFK = :emp_idFK";
112     // @formatter:on
113
114     Map<String, Object> params = new HashMap<>();
115     params.put("emp_idFK", emp_idFK);
116
117     jdbcTemplate.update(sql, params);
118 }
119
120
121 // finds an employees emp_username by emp_idFK
122● @Override
123 public List<user_validation> findEmpUsername(Long emp_idFK) {
124
125     // @formatter:off
126     String sql = ""
127         + "SELECT emp_idFK, emp_username"
128         + "FROM user_validation"
129         + "WHERE emp_idFK = :emp_idFK";
130     // @formatter:on
131
132     Map<String, Object> params = new HashMap<>();
133     params.put("emp_idFK", emp_idFK);
134
135● return jdbcTemplate.query(sql, params, new ResultSetExtractor<>() {
136●     @Override
137     public List<user_validation> extractData(ResultSet rs) throws SQLException, DataAccessException {
138         if (rs.next()) {
139             // @formatter:off
140             return List.of(user_validation.builder()
141                 .emp_idFK(rs.getLong("emp_idFK"))
142                 .emp_username(rs.getString("emp_username"))
143                 .build());
144             // @formatter:on
145         }
146
147         return extractData(null);
148     }
149 });
150 }
151 }
```

```
152 // finds a users password for verification purposes
153 // @formatter:off
154 • @Override
155 public List<user_validation> findUserInfo(String emp_username) {
156
157     // @formatter:off
158     String sql = ""
159         + "SELECT emp_username, emp_validator "
160         + "FROM user_validation "
161         + "WHERE emp_username = :emp_username";
162     // @formatter:on
163
164     Map<String, Object> params = new HashMap<>();
165     params.put("emp_username", emp_username);
166
167• return jdbcTemplate.query(sql, params, new ResultSetExtractor<>() {
168    @Override
169    public List<user_validation> extractData(ResultSet rs) throws SQLException, DataAccessException {
170        if (rs.next()) {
171            // @formatter:off
172            return List.of(user_validation.builder()
173                .emp_username(rs.getString("emp_username"))
174                .emp_validator(rs.getString("emp_validator"))
175                .build());
176            // @formatter:on
177        }
178
179        return extractData(null);
180    }
181 });
182 }
183
184 // gets users roles for login type
185• @SuppressWarnings("deprecation")
186 @Override
187 public List<String> getUserRoles(String emp_username) {
188
189     // @formatter:off
190     String sql = ""
191         + "SELECT user_role "
192         + "FROM user_roles "
193         + "WHERE emp_username = ?";
194     // @formatter:on
195
196     Object[] params = new Object[] {emp_username};
197
198     List<String> roles = this.getJdbcTemplate().queryForList(sql, params, String.class);
199
200     return roles;
201 }
202
203 } // last bracket
204
```



User Roles –



The screenshot shows the Spring Tool Suite 4 interface with the file `UserRolesDao.java` open in the editor. The code defines a DAO interface for managing user roles. It includes methods for getting user roles by employee username, retrieving employees by role, adding roles to employees, adding roles for employees by username, and deleting roles from an employee by role ID.

```
1 package OTC.StPaul.Final.Project.dao;
2
3 import java.util.List;
4
5 public interface UserRolesDao {
6
7     * @param emp_username
8     List<user_roles> getUserRoles(String emp_username);
9
10    * @param role
11    List<user_roles> retrieveAllEmployeesOfACertainRole(Individual_Roles role);
12
13    * @param userRoles
14    void addRolesForEmployees(user_roles userRoles);
15
16    * @param emp_username
17    void addRolesForEmployees(String emp_username, String user_role);
18
19    * @param role_id
20    void deleteRolesFromEmployeeByRoleId(Long role_id);
21
22 } // last bracket
23
```

The screenshot shows a Java IDE interface with two tabs open: "UserRolesDao.java" and "DefaultUserRolesDao.java". The "UserRolesDao.java" tab is active, displaying the following code:

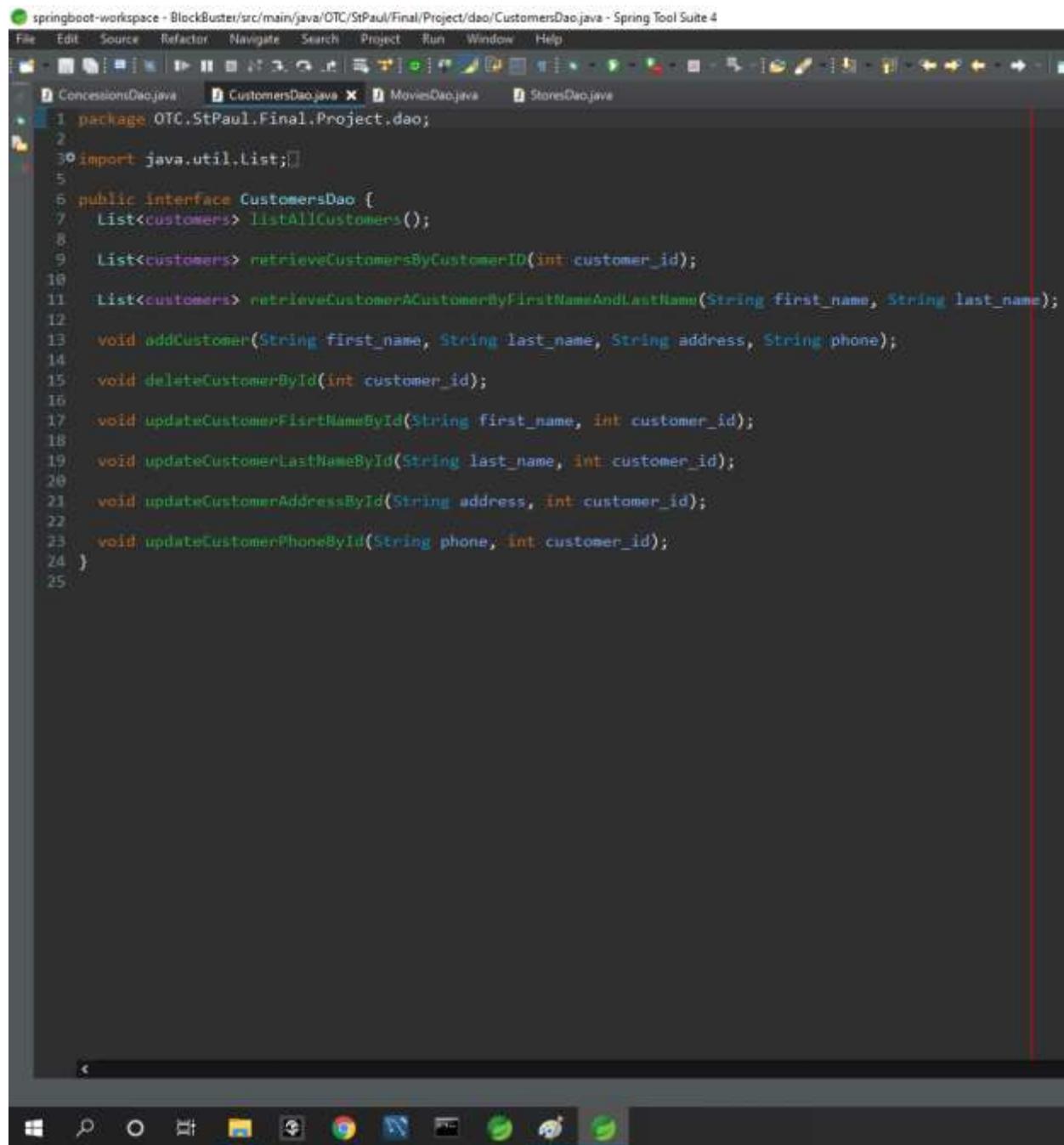
```
File Edit Source Refactor Navigate Search Project Run Window Help
UserRolesDao.java DefaultUserRolesDao.java
1 package OTC.StPaul.Final.Project.dao;
2
3 import java.sql.ResultSet;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
```

The code is annotated with line numbers and includes annotations like @Component, @Service, @Transactional, @Autowired, and @Override. It defines methods for retrieving user roles based on employee username and role.

```
62 // @formatter:off
63 String sql = ""
64     + "SELECT * "
65     + "FROM user_roles "
66     + "WHERE user_role = :user_role";
67 // @formatter:on
68
69 Map<String, Object> params = new HashMap<>();
70 params.put("user_role", role.toString());
71
72 return jdbcTemplate.query(sql, params, new RowMapper<>() {
73
74     @Override
75     public user_roles mapRow(ResultSet rs, int rowNum) throws SQLException {
76
77         // @formatter:off
78         return user_roles.builder()
79             .role_id(rs.getLong("role_id"))
80             .emp_username(rs.getString("emp_username"))
81             .user_role(rs.getString("user_role"))
82             .build();
83
84         // @formatter:on
85     }
86 });
87 }
88
89 // add a role for an employee
90 @Override
91 public void addRolesForEmployees(user_roles userRoles) {
92
93     // @formatter:off
94     String sql = ""
95         + "INSERT INTO user_roles ("
96         + "emp_username, user_role"
97         + ") VALUES ("
98         + ":emp_username, :user_role"
99         + ")";
100    // @formatter:on
101
102    Map<String, Object> params = new HashMap<>();
103    params.put("emp_username", userRoles.getEmp_username());
104    params.put("user_role", userRoles.getUser_role());
105
106    jdbcTemplate.update(sql, params);
```

```
106     jdbcTemplate.update(sql, params);
107 }
108
109 // delete a role for an employee
110 @Override
111 public void deleteRolesFromEmployeeByRoleId(Long role_id) {
112
113     // @formatter:off
114     String sql = ""
115         + "DELETE FROM user_roles "
116         + "WHERE role_id = :role_id";
117     // @formatter:on
118
119     Map<String, Object> params = new HashMap<>();
120     params.put("role_id", role_id);
121
122     jdbcTemplate.update(sql, params);
123 }
124
125 @Override
126 public void addRolesForEmployees(String emp_username, String user_role) {
127
128     // @formatter:off
129     String sql = ""
130         + "INSERT INTO user_roles (" +
131             "emp_username, user_role"
132         + ") VALUES (" +
133             ":emp_username, :user_role"
134         + ")";
135     // @formatter:on
136
137     Map<String, Object> params = new HashMap<>();
138     params.put("emp_username", emp_username);
139     params.put("user_role", user_role);
140
141     jdbcTemplate.update(sql, params);
142 }
143 } // last bracket
```

Customers –



The screenshot shows the Spring Tool Suite 4 interface with the title bar "springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/dao/CustomersDao.java - Spring Tool Suite 4". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, and Build. The code editor displays the following Java interface:

```
1 package OTC.StPaul.Final.Project.dao;
2
3 import java.util.List;
4
5 public interface CustomersDao {
6     List<customers> listAllCustomers();
7
8     List<customers> retrieveCustomersByCustomerID(int customer_id);
9
10    List<customers> retrieveCustomerACustomerByFirstNameAndLastName(String First_name, String last_name);
11
12    void addCustomer(String first_name, String last_name, String address, String phone);
13
14    void deleteCustomerById(int customer_id);
15
16    void updateCustomerFirstNameById(String first_name, int customer_id);
17
18    void updateCustomerLastNameById(String last_name, int customer_id);
19
20    void updateCustomerAddressById(String address, int customer_id);
21
22    void updateCustomerPhoneById(String phone, int customer_id);
23
24 }
25
```

springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/dao/DefaultCustomersDao.java - Spring Tool Suite 4

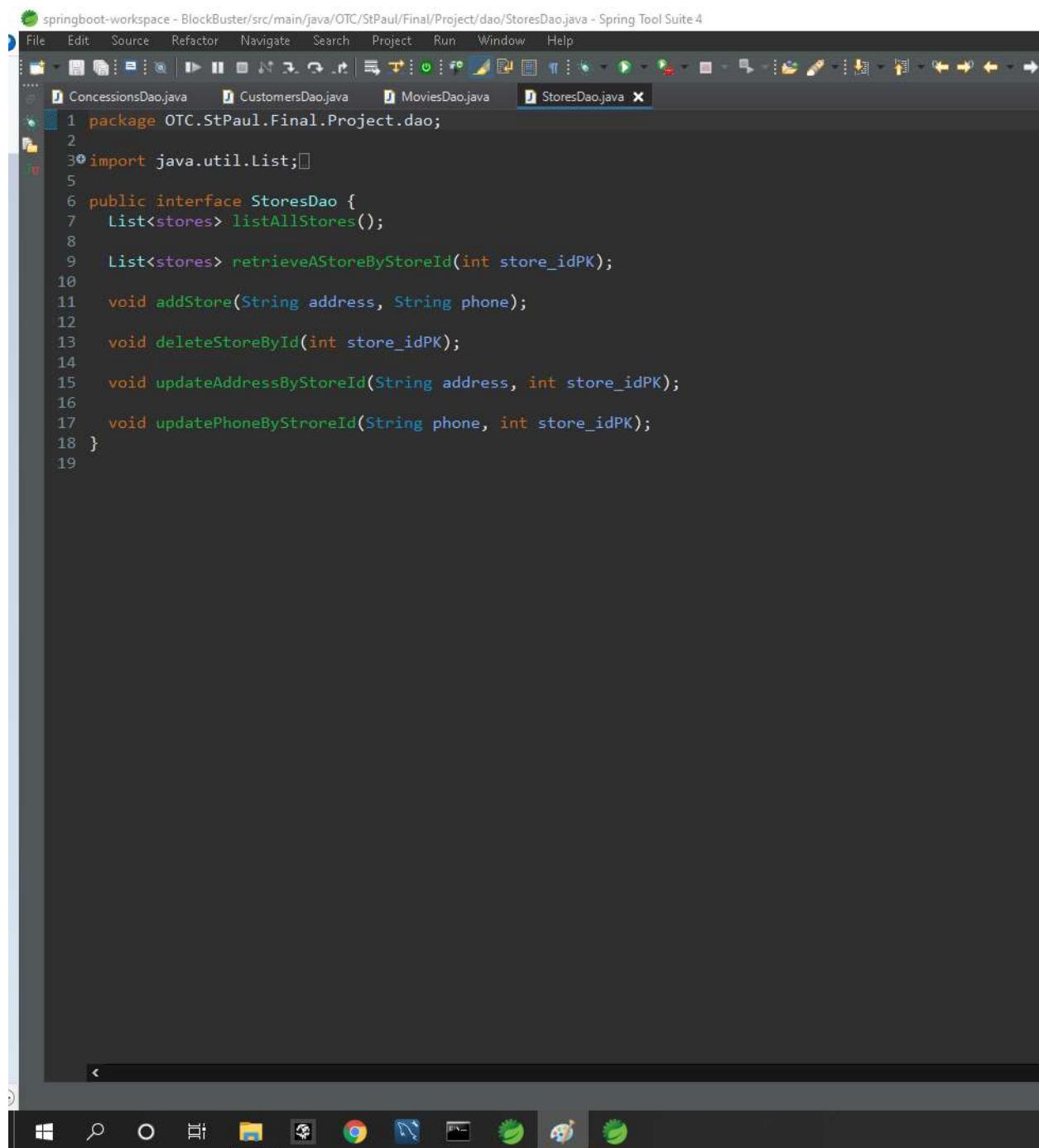
```
1 package OTC.StPaul.Final.Project.dao;
2
3+ import java.sql.ResultSet;[]
17
18 @Component
19 @Service
20 @Transactional
21 public class DefaultCustomersDao implements CustomersDao {
22
23@ Autowired
24 private NamedParameterJdbcTemplate jdbcTemplate;
25
26@ Override
27 public List<customers> listAllCustomers() {
28 // @formatter:off
29 String sql = "" + "SELECT * " + "FROM customers";
30 // @formatter:on
31@ return jdbcTemplate.query(sql, new ResultSetExtractor<>() {
32@     public List<customers> extractData(ResultSet rs) throws SQLException, DataAccessException {
33         if (rs.next()) {
34             // @formatter:off
35             return List.of(customers.builder()
36                 .customer_idPK(rs.getInt("customer_id"))
37                 .first_name(rs.getString("first_name"))
38                 .last_name(rs.getString("last_name"))
39                 .address(rs.getString("address"))
40                 .phone(rs.getString("phone"))
41                 .build());
42             // @formatter:on
43         }
44         return extractData(null);
45     }
46 });
47 }
48
49@ Override
50 public List<customers> retrieveCustomersByCustomerID(int customer_idPK) {
51
52 // @formatter:off
53 String sql = "" + "SELECT * " + "FROM customers " + "WHERE customer_idPK = :customer_idPK";
54 // @formatter:on
55
56 Map<String, Object> params = new HashMap<>();
57 params.put("customer_idPK", customer_idPK);
58@ return jdbcTemplate.query(sql, params, new RowMapper<>() {
59@     @Override
60     public customers mapRow(ResultSet rs, int rowNum) throws SQLException {
```

```
61     // #formatter:off
62     return customers.builder()
63         .customer_idPK(rs.getInt("customer_idPK"))
64         .first_name(rs.getString("first_name"))
65         .last_name(rs.getString("last_name"))
66         .address(rs.getString("address"))
67         .phone(rs.getString("phone"))
68         .build();
69     // #formatter:on
70 }
71 });
72 }
73 }
74
75● @Override
76 public List<customers> retrieveCustomerCustomerByFirstNameAndLastName(String first_name,
77     String last_name) {
78     // #formatter:off
79     String sql = "" + "SELECT * " + "FROM customers " + "WHERE first_name AND last_name = :first_name AND last_name";
80     // #formatter:on
81
82     Map<String, Object> params = new HashMap<>();
83     params.put("first_name", first_name);
84     params.put("last_name", last_name);
85● return jdbcTemplate.query(sql, params, new RowMapper<>() {
86     @Override
87     public customers mapRow(ResultSet rs, int rownum) throws SQLException {
88         // #formatter:off
89         return customers.builder()
90             .customer_idPK(rs.getInt("customer_idPK"))
91             .first_name(rs.getString("first_name"))
92             .last_name(rs.getString("last_name"))
93             .address(rs.getString("address"))
94             .phone(rs.getString("phone"))
95             .build();
96         // #formatter:on
97     }
98 });
99 }
100
101● @Override
102 public void addCustomer(String first_name, String last_name, String address, String phone) {
103     // #formatter:off
104     String sql = "" + "INSERT INTO customers (" + "first_name, last_name, address, phone" + ") "
105         + "VALUES (" + ":first_name, :last_name, :address, :phone" + ")";
106 }
```

```
106     // @formatter:on
107     Map<String, Object> params = new HashMap<>();
108     params.put("first_name", first_name);
109     params.put("last_name", last_name);
110     params.put("address", address);
111     params.put("phone", phone);
112
113     jdbcTemplate.update(sql, params);
114 }
115
116● @Override
117 public void deleteCustomerById(int customer_idPK) {
118     // @formatter:off
119     String sql = "" + "DELETE FROM customers" + "WHERE customer_idPK = :customer_idPK";
120     // @formatter:on
121
122     Map<String, Object> params = new HashMap<>();
123     params.put("customer_idPK", customer_idPK);
124
125     jdbcTemplate.update(sql, params);
126
127 }
128
129● @Override
130 public void updateCustomerFirstNameById(String first_name, int customer_idPK) {
131     // @formatter:off
132     String sql = "" + "UPDATE customers" + "SET first_name = :first_name" + "WHERE customer_idPK = :customer_idPK";
133     // @formatter:on
134
135     Map<String, Object> params = new HashMap<>();
136     params.put("first_name", first_name);
137     params.put("customer_idPK", customer_idPK);
138
139     jdbcTemplate.update(sql, params);
140 }
141
142
143● @Override
144 public void updateCustomerLastNameById(String last_name, int customer_idPK) {
145     // @formatter:off
146     String sql = "" + "UPDATE customers" + "SET last_name = :last_name" + "WHERE customer_idPK = :customer_idPK";
147     // @formatter:on
148
149     Map<String, Object> params = new HashMap<>();
150     params.put("last_name", last_name);
```

```
150     params.put("last_name", last_name);
151     params.put("customer_idPK", customer_idPK);
152 
153     jdbcTemplate.update(sql, params);
154 }
155
156● @Override
157 public void updateCustomerAddressById(String address, int customer_idPK) {
158     // #formatter:off
159     String sql = "" + "UPDATE customers " + "SET address = :address " + "WHERE customer_idPK = :customer_idPK";
160     // #formatter:on
161
162     Map<String, Object> params = new HashMap<>();
163     params.put("address", address);
164     params.put("customer_idPK", customer_idPK);
165
166     jdbcTemplate.update(sql, params);
167 }
168
169
170● @Override
171 public void updateCustomerPhoneById(String phone, int customer_idPK) {
172     // #formatter:off
173     String sql = "" + "UPDATE customers " + "SET phone = :phone " + "WHERE customer_idPK = :customer_idPK";
174     // #formatter:on
175
176     Map<String, Object> params = new HashMap<>();
177     params.put("phone", phone);
178     params.put("customer_idPK", customer_idPK);
179
180     jdbcTemplate.update(sql, params);
181 }
182
183
184 } // last bracket
```

Stores-



The screenshot shows the Spring Tool Suite 4 interface with the title bar "springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/dao/StoresDao.java - Spring Tool Suite 4". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, and Build. The code editor displays the following Java code:

```
1 package OTC.StPaul.Final.Project.dao;
2
3 import java.util.List;
4
5 public interface StoresDao {
6     List<stores> listAllStores();
7
8     List<stores> retrieveAStoreByStoreId(int store_idPK);
9
10    void addStore(String address, String phone);
11
12    void deleteStoreById(int store_idPK);
13
14    void updateAddressByStoreId(String address, int store_idPK);
15
16    void updatePhoneByStroreId(String phone, int store_idPK);
17
18 }
19
```

springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/dao/DefaultStoresDao.java - Spring Tool Suite 4

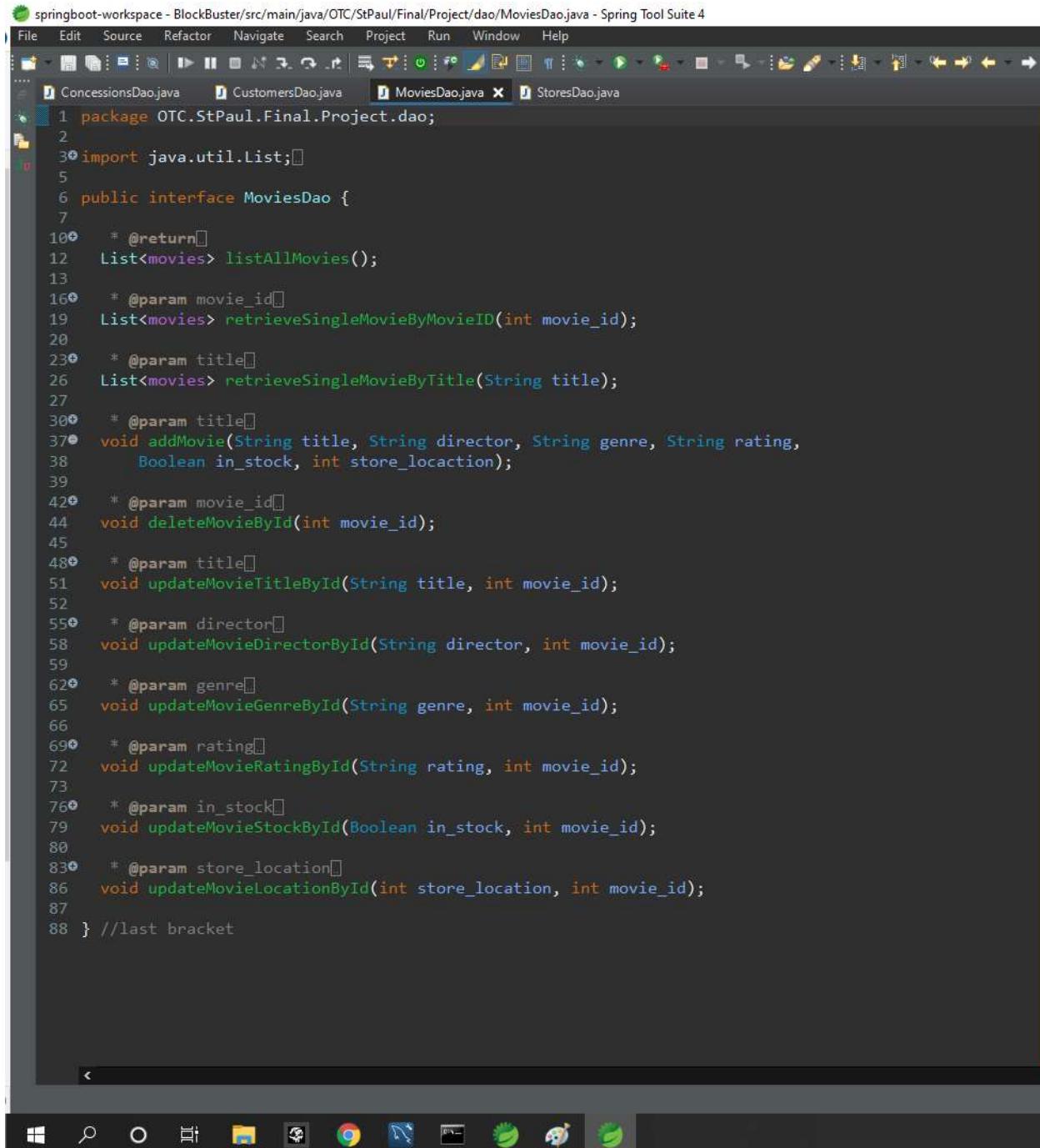
```
File Edit Source Refactor Navigate Search Project Run Window Help
DefaultStoresDao.java DefaultCustomersDao.java DefaultConcessionsDao.java
1 package OTC.StPaul.Final.Project.dao;
2
3
4 import java.sql.ResultSet;
5
6
7 @Component
8 @Service
9 @Transactional
10 public class DefaultStoresDao implements StoresDao {
11
12     @Autowired
13     private NamedParameterJdbcTemplate jdbcTemplate;
14
15     @Override
16     public List<stores> listAllStores() {
17         // @formatter:off
18         String sql = "" + "SELECT * " + "FROM stores";
19         // @formatter:on
20         return jdbcTemplate.query(sql, new ResultSetExtractor<List<stores>>() {
21
22             @Override
23             public List<stores> extractData(ResultSet rs) throws SQLException, DataAccessException {
24                 if (rs.next()) {
25                     // @formatter:off
26                     return List.of(stores.builder()
27                         .store_idPK(rs.getInt("store_idPK"))
28                         .address(rs.getString("address"))
29                         .phone(rs.getString("phone"))
30                         .build());
31                     // @formatter:on
32                 }
33                 //return extractData1(null);
34                 return null;
35             }
36         });
37     }
38
39     @Override
40     public List<stores> retrieveAStoreByStoreId(int store_idPK) {
41         // @formatter:off
42         String sql = "" + "SELECT * " + "FROM stores " + "WHERE store_idPK = :store_idPK";
43         // @formatter:on
44
45         Map<String, Object> params = new HashMap<>();
46         params.put("store_idPK", store_idPK);
47
48         return jdbcTemplate.query(sql, params, new RowMapper<> () {
49             @Override
50             public stores mapRow(ResultSet rs, int rowNum) throws SQLException {
51                 stores store = stores.builder()
52                     .store_idPK(rs.getInt("store_idPK"))
53                     .address(rs.getString("address"))
54                     .phone(rs.getString("phone"))
55                     .build();
56
57                 return store;
58             }
59         });
60     }
61 }
```

```

  62     public stores mapRow(ResultSet rs, int rowNum) throws SQLException, DataAccessException {
  63         // @formatter:off
  64         return stores.builder()
  65             .store_idPK(rs.getInt("store_idPK"))
  66             .address(rs.getString("address"))
  67             .phone(rs.getString("phone"))
  68             .build();
  69         // @formatter:on
  70     }
  71 }
  72 }
  73
 74* @Override
 75 public void addStore(String address, String phone) {
 76     // @formatter:off
 77     String sql = "" + "INSERT INTO stores (" + "address, phone" + ") "
 78     + "VALUES (" + ":address, :phone" + ")";
 79     // @formatter:on
 80
 81     Map<String, Object> params = new HashMap<>();
 82     params.put("address", address);
 83     params.put("phone", phone);
 84
 85     jdbcTemplate.update(sql, params);
 86 }
 87
 88* @Override
 89 public void deleteStoreById(int store_idPK) {
 90     // @formatter:off
 91     String sql = "" + "DELETE FROM stores "
 92     + "WHERE store_idPK = :store_idPK";
 93     // @formatter:on
 94
 95     Map<String, Object> params = new HashMap<>();
 96     params.put("store_idPK", store_idPK);
 97
 98     jdbcTemplate.update(sql, params);
 99 }
100 }
101
102* @Override
103 public void updateAddressByStoreId(String address, int store_idPK) {
104     // @formatter:off
105     String sql = "" + "UPDATE stores " + "SET address = :address " + "WHERE store_idPK = :store_idPK";
106     // @formatter:on
107
108     Map<String, Object> params = new HashMap<>();
109     params.put("address", address);
110     params.put("store_idPK", store_idPK);
111
112     jdbcTemplate.update(sql, params);
113 }
114
115
116* @Override
117 public void updatePhoneByStoreId(String phone, int store_idPK) {
118     // @formatter:off
119     String sql = "" + "UPDATE stores " + "SET phone = :phone " + "WHERE store_idPK = :store_idPK";
120     // @formatter:on
121
122     Map<String, Object> params = new HashMap<>();
123     params.put("phone", phone);
124     params.put("store_idPK", store_idPK);
125
126     jdbcTemplate.update(sql, params);
127 }
128 }
129
130 } // last bracket

```

Movies-



The screenshot shows the Spring Tool Suite 4 interface with the 'MoviesDao.java' file open in the editor. The file contains Java code for a DAO interface named 'MoviesDao'. The code includes methods for listing all movies, retrieving a single movie by ID, adding a new movie, deleting a movie by ID, updating a movie's title, director, genre, rating, stock status, and store location. The code uses annotations like @param and @return. The editor has a dark theme with syntax highlighting for Java keywords and comments.

```
springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/dao/MoviesDao.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
ConcessionsDao.java CustomersDao.java MoviesDao.java StoresDao.java
1 package OTC.StPaul.Final.Project.dao;
2
3 import java.util.List;
4
5 public interface MoviesDao {
6
7     * @return
8     List<movies> listAllMovies();
9
10    * @param movie_id
11    List<movies> retrieveSingleMovieByMovieID(int movie_id);
12
13    * @param title
14    List<movies> retrieveSingleMovieByTitle(String title);
15
16    * @param title
17    void addMovie(String title, String director, String genre, String rating,
18        Boolean in_stock, int store_locaction);
19
20    * @param movie_id
21    void deleteMovieById(int movie_id);
22
23    * @param title
24    void updateMovieTitleById(String title, int movie_id);
25
26    * @param director
27    void updateMovieDirectorById(String director, int movie_id);
28
29    * @param genre
30    void updateMovieGenreById(String genre, int movie_id);
31
32    * @param rating
33    void updateMovieRatingById(String rating, int movie_id);
34
35    * @param in_stock
36    void updateMovieStockById(Boolean in_stock, int movie_id);
37
38    * @param store_location
39    void updateMovieLocationById(int store_location, int movie_id);
40
41 } //last bracket
```

The screenshot shows the Spring Tool Suite 4 interface with the file 'DefaultMoviesDao.java' open in the editor. The code implements the MoviesDao interface, using a NamedParameterJdbcTemplate to query the 'movies' table. It includes methods for listing all movies and retrieving a single movie by ID.

```
1 package OTC.StPaul.Final.Project.dao;
2
3 import java.sql.ResultSet;
4
5
6 @Component
7 @Service
8 @Transactional
9 public class DefaultMoviesDao implements MoviesDao {
10
11     @Autowired
12     public NamedParameterJdbcTemplate jdbcTemplate;
13
14     // returns a list of all movies in database (from every store)
15     @Override
16     public List<movies> listAllMovies() {
17
18         // @formatter:off
19         String sql = ""
20             + "SELECT * "
21             + "FROM movies";
22         // @formatter:on
23
24         return jdbcTemplate.query(sql, new ResultSetExtractor<>() {
25             @Override
26             public List<movies> extractData(ResultSet rs) throws SQLException, DataAccessException {
27                 if (rs.next()) {
28                     // @formatter:off
29                     return List.of(movies.builder()
30                         .movie_id(rs.getInt("movie_id"))
31                         .title(rs.getString("title"))
32                         .director(rs.getString("director"))
33                         .genre(rs.getString("genre"))
34                         .rating(rs.getString("rating"))
35                         .in_stock(rs.getBoolean("in_stock"))
36                         .store_location(rs.getInt("store_location"))
37                         .build());
38                     // @formatter:on
39                 }
40                 return extractData(null);
41             }
42         });
43     }
44
45     // returns a single movie by movie id number
46     @Override
47     public List<movies> retrieveSingleMovieByMovieID(int movie_id) {
```

```
61 // @formatter:off
62 String sql = ""
63     + "SELECT * "
64     + "FROM movies "
65     + "WHERE movie_id = :movie_id";
66 // @formatter:on
67
68
69 Map<String, Object> params = new HashMap<>();
70 params.put("movie_id", movie_id);
71
72 return jdbcTemplate.query(sql, params, new RowMapper<>() {
73
74     @Override
75     public movies mapRow(ResultSet rs, int rowNum) throws SQLException {
76
77         // @formatter:off
78         return movies.builder()
79             .movie_id(rs.getInt("movie_id"))
80             .title(rs.getString("title"))
81             .director(rs.getString("director"))
82             .genre(rs.getString("genre"))
83             .rating(rs.getString("rating"))
84             .in_stock(rs.getBoolean("in_stock"))
85             .store_location(rs.getInt("store_location"))
86             .build();
87         // @formatter:on
88     }
89 });
90 }
91
92 // returns a single movie by title
93 @Override
94 public List<movies> retrieveSingleMovieByTitle(String title) {
95 // @formatter:off
96 String sql = ""
97     + "SELECT * "
98     + "FROM movies "
99     + "WHERE title = :title";
100 // @formatter:on
101
102 Map<String, Object> params = new HashMap<>();
103 params.put("title", title);
104
105 return jdbcTemplate.query(sql, params, new RowMapper<>() {
```

```
105     return jdbcTemplate.query(sql, params, new RowMapper<Movie>() {
106
107     @Override
108     public Movie mapRow(ResultSet rs, int rowNum) throws SQLException {
109
110         // @formatter:off
111         return movies.builder()
112             .movie_id(rs.getInt("movie_id"))
113             .title(rs.getString("title"))
114             .director(rs.getString("director"))
115             .genre(rs.getString("genre"))
116             .rating(rs.getString("rating"))
117             .in_stock(rs.getBoolean("in_stock"))
118             .store_location(rs.getInt("store_location"))
119             .build();
120         // @formatter:on
121     }
122 });
123 }
124
125     @Override
126     public void addMovie(String title, String director, String genre, String rating, Boolean in_stock,
127             int store_location) {
128         // @formatter:off
129         String sql = ""
130             + "INSERT INTO movies (
131             + "title, director, genre, rating, in_stock, store_location"
132             + ")VALUES (
133             + ":title, :director, :genre, :rating, :in_stock, :store_location"
134             + ")";
135         // @formatter:on
136         Map<String, Object> params = new HashMap<>();
137
138         params.put("title", title);
139         params.put("director", director);
140         params.put("genre", genre);
141         params.put("rating", rating);
142         params.put("in_stock", in_stock);
143         params.put("store_location", store_location);
144
145         jdbcTemplate.update(sql, params);
146     }
147
148     // deletes a movie by id number
149     @Override
150     public void deleteMovieById(int movie_id) {
151         // @formatter:off
```

```
151     String sql = ""
152         + "DELETE FROM movies "
153         + "WHERE movie_id = :movie_id";
154     // @formatter:on
155
156     Map<String, Object> params = new HashMap<>();
157     params.put("movie_id", movie_id);
158
159     jdbcTemplate.update(sql, params);
160 }
161
162 // updates a movie title by movie id number
163 public void updateMovieTitleById(String title, int movie_id) {
164 // @formatter:off
165     String sql = ""
166         + "UPDATE movies "
167         + "SET title = :title "
168         + "WHERE movie_id = :movie_id";
169     // @formatter:on
170
171     Map<String, Object> params = new HashMap<>();
172     params.put("title", title);
173     params.put("movie_id", movie_id);
174
175     jdbcTemplate.update(sql, params);
176 }
177
178 // updates a movie director by movie id number
179 public void updateMovieDirectorById(String director, int movie_id) {
180 // @formatter:off
181     String sql = ""
182         + "UPDATE movies "
183         + "SET director = :director "
184         + "WHERE movie_id = :movie_id";
185     // @formatter:on
186
187     Map<String, Object> params = new HashMap<>();
188     params.put("director", director);
189     params.put("movie_id", movie_id);
190
191     jdbcTemplate.update(sql, params);
192 }
193
194 }
```

```
196 // updates a movie genre by movie id number
▲197● public void updateMovieGenreById(String genre, int movie_id) {
198 // @formatter:off
199     String sql = ""
200     + "UPDATE movies "
201     + "SET genre = :genre "
202     + "WHERE movie_id = :movie_id";
203 // @formatter:on
204
205     Map<String, Object> params = new HashMap<>();
206     params.put("genre", genre);
207     params.put("movie_id", movie_id);
208
209     jdbcTemplate.update(sql, params);
210
211 }
212
213 // updates a movie rating by movie id number
▲214● public void updateMovieRatingById(String rating, int movie_id) {
215 // @formatter:off
216     String sql = ""
217     + "UPDATE movies "
218     + "SET rating = :rating "
219     + "WHERE movie_id = :movie_id";
220 // @formatter:on
221
222     Map<String, Object> params = new HashMap<>();
223     params.put("rating", rating);
224     params.put("movie_id", movie_id);
225
226     jdbcTemplate.update(sql, params);
227
228 }
229
230 // updates a movie stock by movie id number
▲231● public void updateMovieStockById(Boolean in_stock, int movie_id) {
232 // @formatter:off
233     String sql = ""
234     + "UPDATE movies "
235     + "SET in_stock = :in_stock "
236     + "WHERE movie_id = :movie_id";
237 // @formatter:on
238
239     Map<String, Object> params = new HashMap<>();
240     params.put("in stock", in stock);
```

```
241     params.put("movie_id", movie_id);
242     jdbcTemplate.update(sql, params);
243 }
244
245 // updates a movie's location by movie id number
246 public void updateMovieLocationById(int store_location, int movie_id) {
247     // @formatter:off
248     String sql = ""
249     + "UPDATE movies "
250     + "SET store_location = :store_location "
251     + "WHERE movie_id = :movie_id";
252     // @formatter:on
253
254     Map<String, Object> params = new HashMap<>();
255     params.put("store_location", store_location);
256     params.put("movie_id", movie_id);
257
258     jdbcTemplate.update(sql, params);
259 }
260
261 }
262
263 } // last bracket
264
```



Concessions-

The screenshot shows a Java code editor with the file `ConcessionsDao.java` open. The code defines a DAO interface for managing concessions. It includes methods for listing all concessions, retrieving a single concession by ID, adding a new concession, deleting a concession by ID, updating a concession's name by ID, updating its price by ID, and updating its quantity by ID. The code uses annotations like `@param` and `@return`. The interface is part of the package `OTC.StPaul.Final.Project.dao`.

```
1 package OTC.StPaul.Final.Project.dao;
2
3 import java.util.List;
4
5 public interface ConcessionsDao {
6     /* @return */
7     List<concessions> listAllConcessions();
8
9     /* @param concession_id */
10    List<concessions> retrieveSingleConcessionById(int concessions_id);
11
12    /* @param full_name */
13    List<concessions> retrieveSingleConcessionByName(String full_name);
14
15    /* @param full_name */
16    void addConcession(String full_name, double price, int quantity);
17
18    /* @param concession_id */
19    void deleteConcessionById(int concessions_id);
20
21    /* @param full_name */
22    void updateConcessionNameById(String full_name, int concessions_id);
23
24    /* @param price */
25    void updateConcessionPriceById(double price, int concessions_id);
26
27    /* @param quantity */
28    void updateConcessionQuantityById(int quantity, int concessions_id);
29
30 } // last bracket
31
32
33
```

The screenshot shows the Spring Tool Suite 4 interface with the file `DefaultConcessionsDao.java` open in the editor. The code implements the `ConcessionsDao` interface, using `NamedParameterJdbcTemplate` to query a database. It includes two methods: `listAllConcessions()` and `retrieveSingleConcessionByConcessionID(int concessions_id)`. The code uses Java 8's `Optional` and `Stream` API to handle the database results.

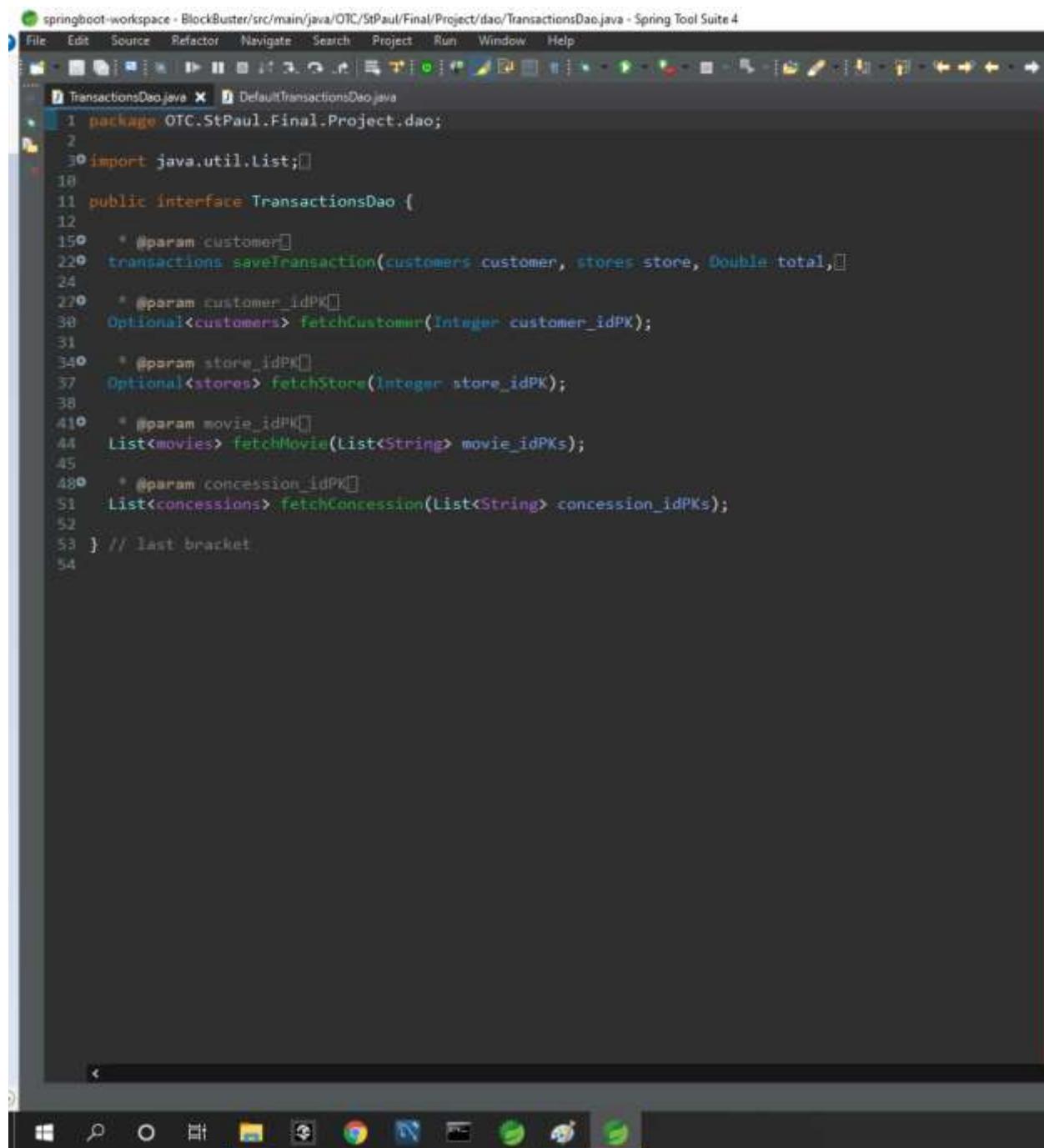
```
1 package OTC.StPaul.Final.Project.dao;
2
3 import java.sql.ResultSet;
4
5
6 @Component
7 @Service
8 @Transactional
9 public class DefaultConcessionsDao implements ConcessionsDao {
10
11     @Autowired
12     public NamedParameterJdbcTemplate jdbcTemplate;
13
14     // returns a list of all concession items
15     @Override
16     public List<Concessions> listAllConcessions() {
17
18         // @formatter:off
19         String sql = "";
20         + "SELECT * "
21         + "FROM concessions";
22         // @formatter:on
23
24         return jdbcTemplate.query(sql, new ResultSetExtractor<>() {
25             @Override
26             public List<Concessions> extractData(ResultSet rs) throws SQLException, DataAccessException {
27                 if (rs.next()) {
28                     // @formatter:off
29                     return List.of(concessions.builder()
30                         .concessions_id(rs.getInt("concessions_id"))
31                         .full_name(rs.getString("full_name"))
32                         .price(rs.getDouble("price"))
33                         .quantity(rs.getInt("quantity"))
34                         .build());
35                     // @formatter:on
36                 }
37                 return extractData(null);
38             }
39         });
40     }
41
42     // returns a single concession item by concession id
43     @Override
44     public List<Concessions> retrieveSingleConcessionByConcessionID(int concessions_id) {
45
46         // @formatter:off
47         String sql = ""
48         + "SELECT * "
49         + "FROM concessions"
50         + "WHERE concessions_id = :id"
51         + "LIMIT 1";
52         // @formatter:on
53
54         return jdbcTemplate.query(sql, new MapSqlParameterSource("id", concessions_id),
55             new ResultSetExtractor<>() {
56                 @Override
57                 public List<Concessions> extractData(ResultSet rs) throws SQLException, DataAccessException {
58                     if (rs.next()) {
59                         // @formatter:off
60                         return List.of(concessions.builder()
61                             .concessions_id(rs.getInt("concessions_id"))
62                             .full_name(rs.getString("full_name"))
63                             .price(rs.getDouble("price"))
64                             .quantity(rs.getInt("quantity"))
65                             .build());
66                         // @formatter:on
67                     }
68                     return extractData(null);
69                 }
70             });
71     }
72 }
```

```
61      + "SELECT * "
62      + "FROM concessions "
63      + "WHERE concessions_id = :concessions_id";
64  // @formatter:on
65
66  Map<String, Object> params = new HashMap<>();
67  params.put("concessions_id", concessions_id);
68
69● return jdbcTemplate.query(sql, params, new RowMapper<>() {
70
71●     @Override
72     public concessions mapRow(ResultSet rs, int columnIndex) throws SQLException {
73
74     // @formatter:off
75     return concessions.builder()
76         .concessions_id(rs.getInt("concessions_id"))
77         .full_name(rs.getString("full_name"))
78         .price(rs.getDouble("price"))
79         .quantity(rs.getInt("quantity"))
80         .build();
81     // @formatter:on
82     }
83   );
84 }
85
86 // returns a single concession item by name
87● @Override
88 public List<concessions> retrieveSingleConcessionByName(String full_name) {
89
90     // @formatter:off
91     String sql = ""
92     + "SELECT * "
93     + "FROM concessions "
94     + "WHERE full_name = :full_name";
95 // @formatter:on
96
97     Map<String, Object> params = new HashMap<>();
98     params.put("full_name", full_name);
99
100● return jdbcTemplate.query(sql, params, new RowMapper<>() {
101
102●     @Override
103     public concessions mapRow(ResultSet rs, int columnIndex) throws SQLException {
104
105     // @formatter:off
106     return concessions.builder()
107         .concessions_id(rs.getInt("concessions_id"))
108         .full_name(rs.getString("full_name"))
109         .price(rs.getDouble("price"))
110         .quantity(rs.getInt("quantity"))
111         .build();
112     // @formatter:on
113   );
114 }
```

```
106         return concessions.builder()
107             .concessions_id(rs.getInt("concessions_id"))
108             .full_name(rs.getString("full_name"))
109             .price(rs.getDouble("price"))
110             .quantity(rs.getInt("quantity"))
111             .build();
112     // @formatter:on
113 }
114 );
115 }
116
117● @Override
118 public void addConcession(String full_name, double price, int quantity) {
119 // @formatter:off
120 String sql = ""
121     + "INSERT INTO concessions ("
122     + "full_name, price, quantity"
123     + ")VALUES ("
124     + ":full_name, :price, :quantity"
125     + ")";
126 // @formatter:on
127 Map<String, Object> params = new HashMap<>();
128
129 params.put("full_name", full_name);
130 params.put("price", price);
131 params.put("quantity", quantity);
132
133 jdbcTemplate.update(sql, params);
134 }
135
136 // deletes a concessions item by id
137● @Override
138 public void deleteConcessionById(int concessions_id) {
139 // @formatter:off
140 String sql = ""
141     + "DELETE FROM concessions "
142     + "WHERE concessions_id = :concessions_id";
143 // @formatter:on
144
145 Map<String, Object> params = new HashMap<>();
146 params.put("concessions_id", concessions_id);
147
148 jdbcTemplate.update(sql, params);
149 }
150
```

```
151     // updates a concession name by concession id number
152     @Override
153     public void updateConcessionNameById(String full_name, int concessions_id) {
154         // @formatter:off
155         String sql = ""
156             + "UPDATE concessions "
157             + "SET full_name = :full_name "
158             + "WHERE concessions_id = :concessions_id";
159         // @formatter:on
160
161         Map<String, Object> params = new HashMap<>();
162         params.put("full_name", full_name);
163         params.put("concessions_id", concessions_id);
164
165         jdbcTemplate.update(sql, params);
166     }
167
168     // updates a concession price by concession id
169     @Override
170     public void updateConcessionPriceById(double price, int concessions_id) {
171         // @formatter:off
172         String sql = ""
173             + "UPDATE concessions "
174             + "SET price = :price "
175             + "WHERE concessions_id = :concessions_id";
176         // @formatter:on
177
178         Map<String, Object> params = new HashMap<>();
179         params.put("price", price);
180         params.put("concessions_id", concessions_id);
181
182         jdbcTemplate.update(sql, params);
183     }
184
185     @Override
186     public void updateConcessionQuantityById(int quantity, int concessions_id) {
187         // @formatter:off
188         String sql = ""
189             + "UPDATE concessions "
190             + "SET quantity = :quantity "
191             + "WHERE concessions_id = :concessions_id";
192         // @formatter:on
193
194         Map<String, Object> params = new HashMap<>();
195         params.put("quantity", quantity);
196
197         params.put("concessions_id", concessions_id);
198         jdbcTemplate.update(sql, params);
199     }
200 } // last bracket
```

Transactions-



The screenshot shows the Spring Tool Suite 4 interface with the file `TransactionsDao.java` open in the editor. The code defines a DAO interface for managing transactions. It includes methods for saving a transaction, fetching a customer by ID, fetching a store by ID, fetching movies by IDs, and fetching concessions by IDs. The code uses annotations like `@param` and `Optional` from the `java.util` package.

```
1 package OTC.StPaul.Final.Project.dao;
2
3 import java.util.List;
4
5 public interface TransactionsDao {
6
7     * @param customer
8     transactions saveTransaction(customers customer, stores store, double total);
9
10    * @param customer_idPK
11    Optional<customers> fetchCustomer(Integer customer_idPK);
12
13    * @param store_idPK
14    Optional<stores> fetchStore(Integer store_idPK);
15
16    * @param movie_idPK
17    List<movies> fetchMovie(List<String> movie_idPKs);
18
19    * @param concession_idPK
20    List<concessions> fetchConcession(List<String> concession_idPKs);
21
22 } // last bracket
23
```

springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/dao/DefaultTransactionsDao.java - Spring Tool Suite 4

```
File Edit Source Refactor Navigate Search Project Run Window Help
TransactionsDao.java DefaultTransactionsDao.java x
1 package OTC.StPaul.Final.Project.dao;
2
3+ import java.sql.ResultSet;[]
4
5 @Component
6 public class DefaultTransactionsDao implements TransactionsDao {
7
8     @Autowired
9     private NamedParameterJdbcTemplate jdbcTemplate;
10
11
12     @Override
13     public transactions saveTransaction(customers customer, stores store,
14             Double total, List<movies> movies_rented, List<concessions> concessions_purchased) {
15         SqlParams params = generateInsertSql(customer, store, total);
16
17         KeyHolder keyHolder = new GeneratedKeyHolder();
18         jdbcTemplate.update(params.sql, params.source, keyHolder);
19
20         Integer transactionId = keyHolder.getKey().intValue();
21         saveMovies(movies_rented, transactionId);
22         saveConcessions(concessions_purchased, transactionId);
23
24         // @formatter:off
25         return transactions.builder()
26             .transaction_id(transactionId)
27             .customer(customer)
28             .store(store)
29             .movies_rented(movies_rented)
30             .concessions_purchased(concessions_purchased)
31             .total(total)
32             .build();
33     // @formatter:on
34 }
35
36     * @param movies_rented[]
37     private void saveMovies(List<movies> movies_rented, Integer transactionId) {
38         for (movies movie : movies_rented) {
39             SqlParams params = generateInsertSql(movie, transactionId);
40             jdbcTemplate.update(params.sql, params.source);
41         }
42     }
43
44     * @param concessions_purchased[]
45     private void saveConcessions(List<concessions> concessions_purchased, Integer transactionId) {
46         for (concessions concession : concessions_purchased) {
```

```
75     SqlParams params = generateInsertSql(concession, transactionId);
76     jdbcTemplate.update(params.sql, params.source);
77 }
78 }
79 }
80 }
81 * @param customer[]
82 private SqlParams generateInsertSql(customers customer, stores store, Double total) {
83     // @formatter:off
84     String sql = ""
85         + "INSERT INTO transactions (" +
86         + "customer_idFK, store_idFK, total" +
87         + ") VALUES (" +
88         + ":customer_idFK, :store_idFK, :total" +
89         + ")";
90     // @formatter:on
91
92     SqlParams params = new SqlParams();
93
94     params.sql = sql;
95     params.source.addValue("customer_idFK", customer.getCustomer_idPK());
96     params.source.addValue("store_idFK", store.getStore_idPK());
97     params.source.addValue("total", total);
98
99     return params;
100 }
101
102 * @param movie[]
103 private SqlParams generateInsertSql(movies movie, Integer transactionId) {
104     SqlParams params = new SqlParams();
105
106     // @formatter:off
107     params.sql = ""
108         + "INSERT INTO transaction_movies (" +
109         + "movie_idFK, transaction_idFK" +
110         + ") VALUES (" +
111         + ":movie_idFK, :transaction_idFK" +
112         + ")";
113     // @formatter:on
114
115     params.source.addValue("movie_idFK", movie.getMovie_id());
116     params.source.addValue("transaction_idFK", transactionId);
117
118     return params;
119 }
```

```
131
134*  * @param concession
138● private SqlParams generateInsertSql(concessions concession, Integer transactionId) {
139     SqlParams params = new SqlParams();
140
141     // @formatter:off
142     params.sql = ""
143         + "INSERT INTO transaction_concessions (" +
144             + "concessions_idFK, transaction_idFK"
145             + ") VALUES (" +
146                 + ":concessions_idFK, :transaction_idFK"
147                 + ")";
148     // @formatter:on
149
150     params.source.addValue("concessions_idFK", concession.getConcessions_id());
151     params.source.addValue("transaction_idFK", transactionId);
152
153     return params;
154 }
155
156● @Override
157 public Optional<customers> fetchCustomer(Integer customer_idPK) {
158     // @formatter:off
159     String sql = ""
160         + "SELECT * "
161         + "FROM customers "
162         + "WHERE customer_idPK = :customer_idPK";
163     // @formatter:on
164
165     Map<String, Object> params = new HashMap<>();
166     params.put("customer_idPK", customer_idPK);
167
168     return Optional.ofNullable(jdbcTemplate.query(sql, params, new CustomerResultSetExtractor()));
169 }
170
171● public Optional<stores> fetchStore(Integer store_idPK) {
172     // @formatter:off
173     String sql = ""
174         + "SELECT * "
175         + "FROM stores "
176         + "WHERE store_idPK = :store_idPK";
177     // @formatter:on
178
179     Map<String, Object> params = new HashMap<>();
180     params.put("store_idPK", store_idPK);
```

```
181     return Optional.ofNullable(jdbcTemplate.query(sql, params, new StoreResultSetExtractor()));
182 }
183
184
185• public List<movies> fetchMovie(List<String> movie_ids) {
186     if (movie_ids.isEmpty()) {
187         return new LinkedList<>();
188     }
189
190     Map<String, Object> params = new HashMap<>();
191
192     // @formatter:off
193     String sql = ""
194         + "SELECT * "
195         + "FROM movies "
196         + "WHERE movie_id IN(";
197     // @formatter:on
198
199     for (int index = 0; index < movie_ids.size(); index++) {
200         String key = "movie_" + index;
201         sql += ":" + key + ", ";
202         params.put(key, movie_ids.get(index));
203     }
204
205     sql = sql.substring(0, sql.length() - 2);
206     sql += ")";
207
208• return jdbcTemplate.query(sql, params, new RowMapper<movies>() {
209•     @Override
210     public movies mapRow(ResultSet rs, int rowNum) throws SQLException {
211         // @formatter:off
212         return movies.builder()
213             .movie_id(rs.getInt("movie_id"))
214             .title(rs.getString("title"))
215             .build();
216         // @formatter:on
217     }
218 });
219 }
220
221• public List<concessions> fetchConcession(List<String> concessions_ids) {
222     if (concessions_ids.isEmpty()) {
223         return new LinkedList<>();
224     }
225 }
```

```
226     Map<String, Object> params = new HashMap<>();
227
228     // @formatter:off
229     String sql = ""
230         + "SELECT * "
231         + "FROM concessions "
232         + "WHERE concessions_id IN(";
233     // @formatter:on
234
235     for (int index = 0; index < concessions_ids.size(); index++) {
236         String key = "concession_" + index;
237         sql += ":" + key + ", ";
238         params.put(key, concessions_ids.get(index));
239     }
240
241     sql = sql.substring(0, sql.length() - 2);
242     sql += ")";
243
244     return jdbcTemplate.query(sql, params, new RowMapper<concessions>() {
245         @Override
246         public concessions mapRow(ResultSet rs, int rowNum) throws SQLException {
247             // @formatter:off
248             return concessions.builder()
249                 .concessions_id(rs.getInt("concessions_id"))
250                 .full_name(rs.getString("full_name"))
251                 .price(rs.getDouble("price"))
252                 .build();
253             // @formatter:on
254         }
255     });
256 }
257
258 class CustomerResultSetExtractor implements ResultSetExtractor<customers> {
259     @Override
260     public customers extractData(ResultSet rs) throws SQLException {
261         rs.next();
262
263         // @formatter:off
264         return customers.builder()
265             .customer_idPK(rs.getInt("customer_idPK"))
266             .first_name(rs.getString("first_name"))
267             .last_name(rs.getString("last_name"))
268             .build();
269         // @formatter:on
270     }
}
```

```
271     }
272
273● class StoreResultSetExtractor implements ResultSetExtractor<stores> {
274●     @Override
275     public stores extractData(ResultSet rs) throws SQLException {
276         rs.next();
277
278         // @formatter:off
279         return stores.builder()
280             .store_idPK(rs.getInt("store_idPK"))
281             .address(rs.getString("address"))
282             .phone(rs.getString("phone"))
283             .build();
284         // @formatter:on
285     }
286 }
287
288● class MovieResultSetExtractor implements ResultSetExtractor<movies> {
289●     @Override
290     public movies extractData(ResultSet rs) throws SQLException {
291         rs.next();
292
293         // @formatter:off
294         return movies.builder()
295             .title(rs.getString("title"))
296             .build();
297         // @formatter:on
298     }
299 }
300
301● class ConcessionResultSetExtractor implements ResultSetExtractor<concessions> {
302●     @Override
303     public concessions extractData(ResultSet rs) throws SQLException {
304         rs.next();
305
306         // @formatter:off
307         return concessions.builder()
308             .full_name(rs.getString("full_name"))
309             .price(rs.getDouble("price"))
310             .build();
311         // @formatter:on
312     }
313 }
314
315● class SqlParams {
```

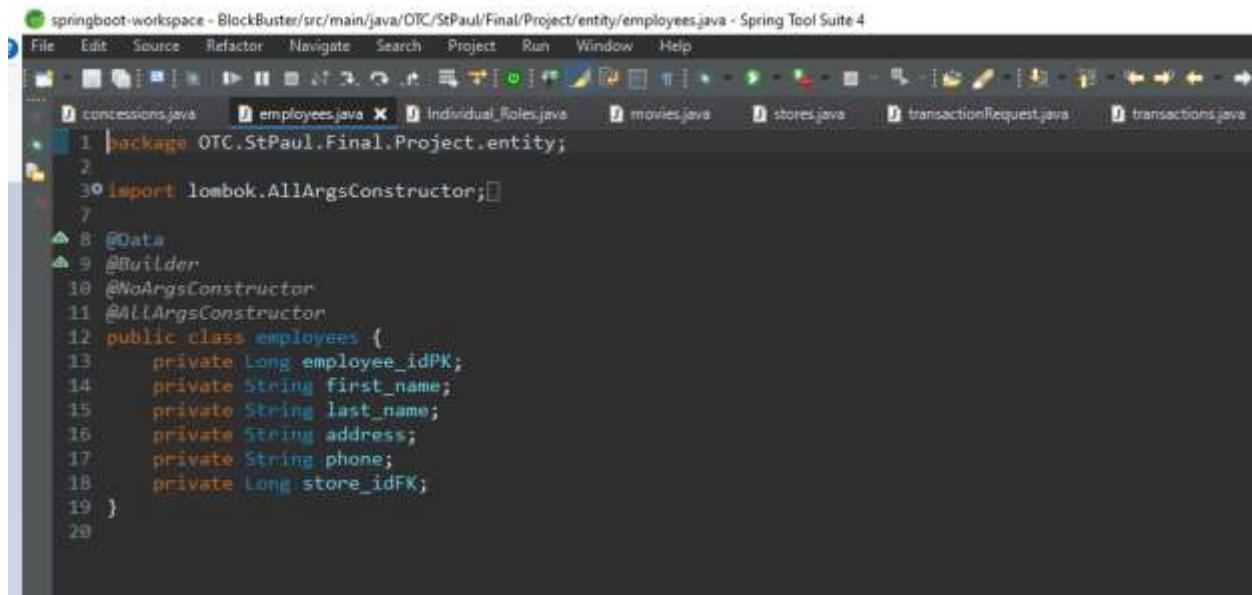


```
316     String sql;
317     MapSqlParameterSource source = new MapSqlParameterSource();
318 }
319
320 } // last bracket
`
```



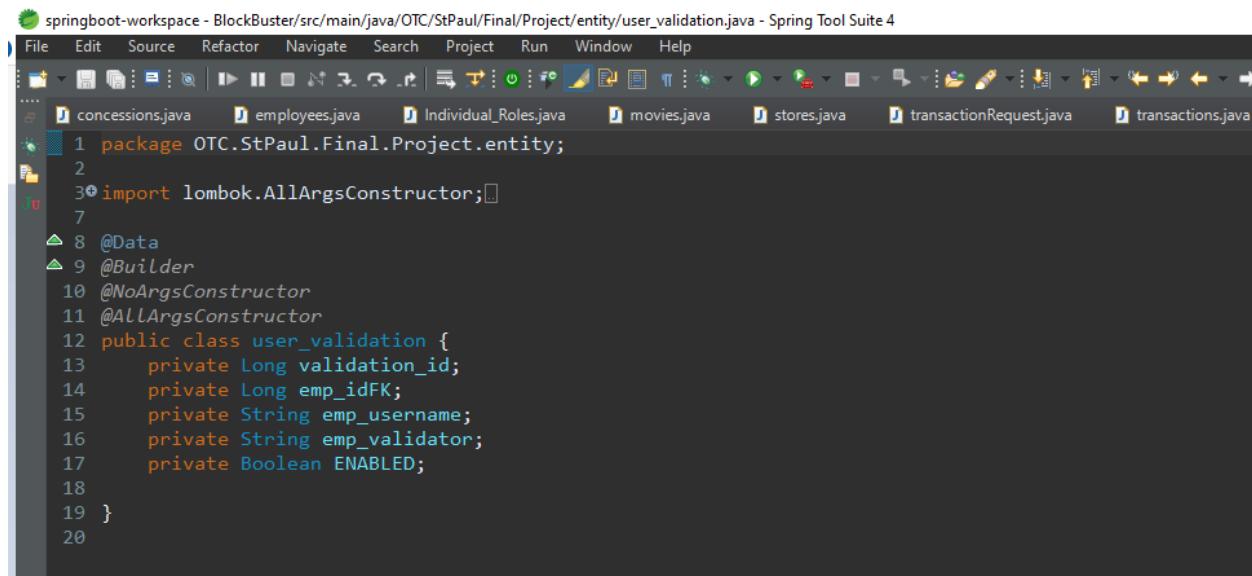
Entity:

Employees-



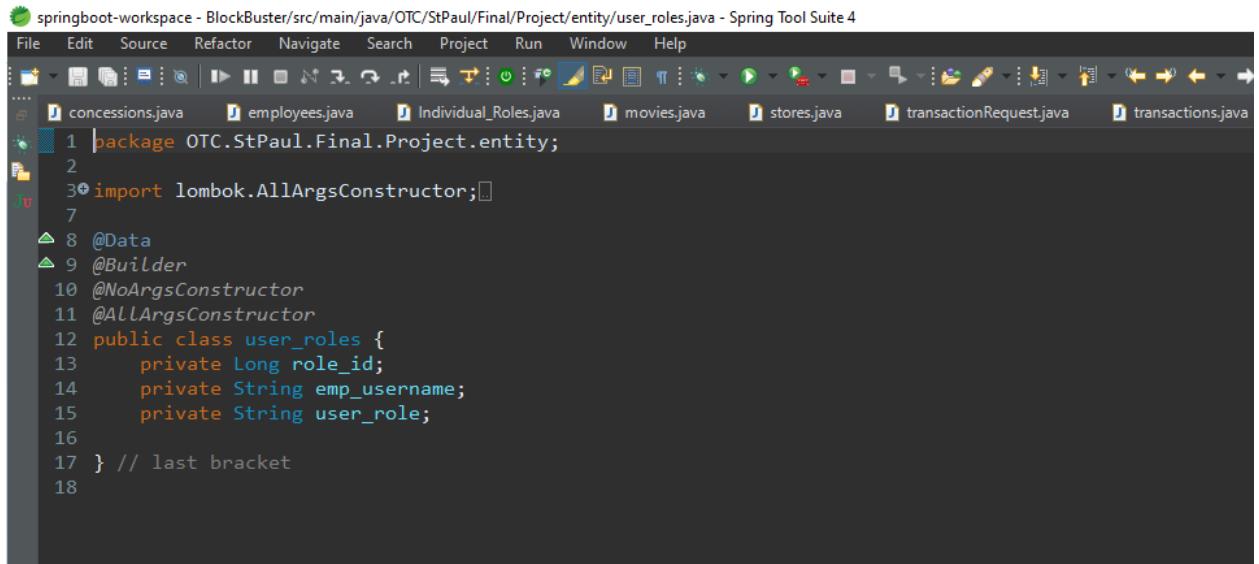
```
springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/entity/employees.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Project Run Window Help
concessions.java employees.java Individual_Roles.java movies.java stores.java transactionRequest.java transactions.java
1 package OTC.StPaul.Final.Project.entity;
2
3 import lombok.AllArgsConstructor;
4 import lombok.NoArgsConstructor;
5 import lombok.NonNull;
6 import lombok.ToString;
7
8 @Data
9 @Builder
10 @NoArgsConstructor
11 @AllArgsConstructor
12 public class employees {
13     private Long employee_idPK;
14     private String first_name;
15     private String last_name;
16     private String address;
17     private String phone;
18     private Long store_idFK;
19 }
20
```

User Validation-

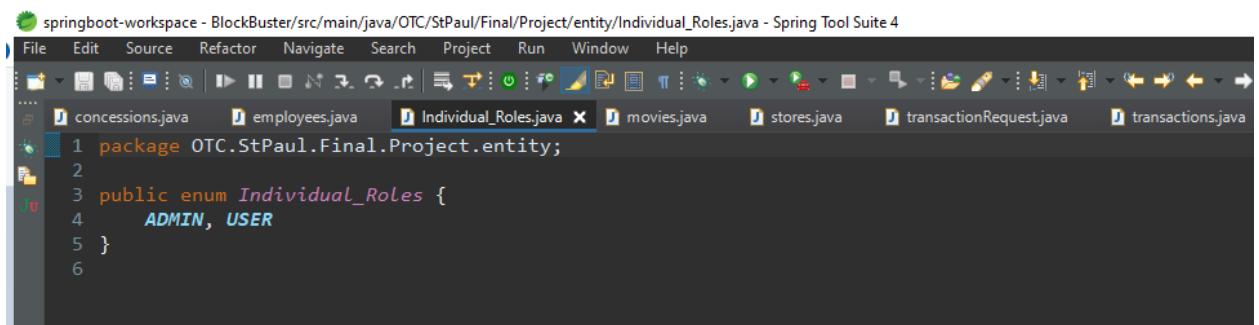


```
springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/entity/user_validation.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Project Run Window Help
concessions.java employees.java Individual_Roles.java movies.java stores.java transactionRequest.java transactions.java
1 package OTC.StPaul.Final.Project.entity;
2
3 import lombok.AllArgsConstructor;
4 import lombok.NoArgsConstructor;
5 import lombok.NonNull;
6 import lombok.ToString;
7
8 @Data
9 @Builder
10 @NoArgsConstructor
11 @AllArgsConstructor
12 public class user_validation {
13     private Long validation_id;
14     private Long emp_idFK;
15     private String emp_username;
16     private String emp_validator;
17     private Boolean ENABLED;
18 }
19
20
```

User Roles –

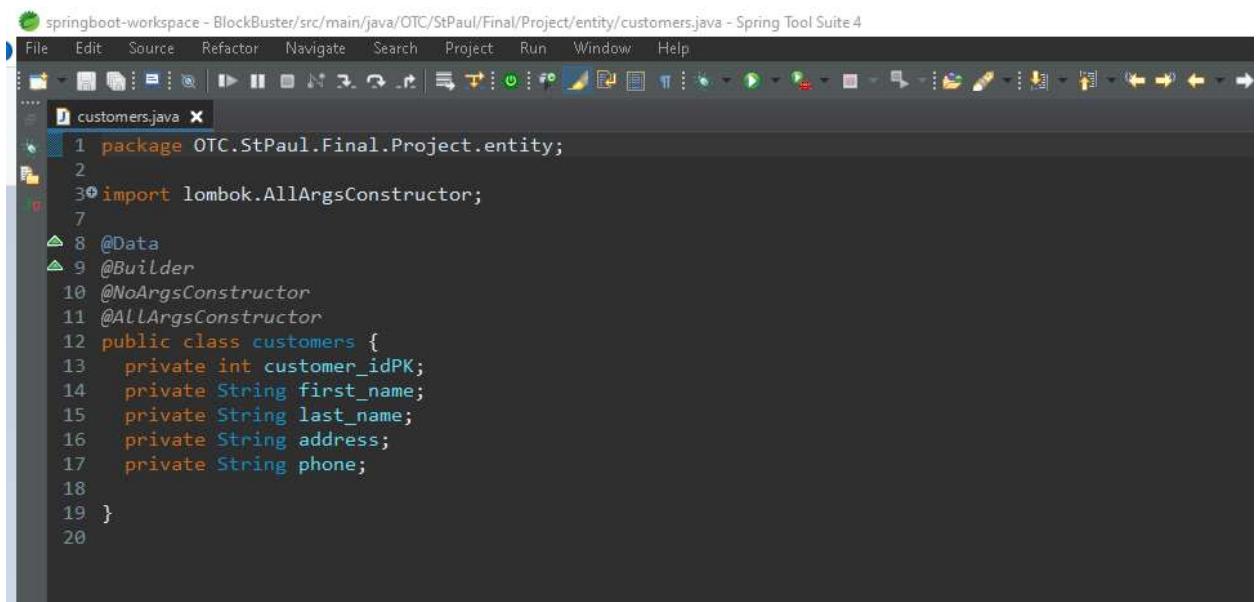


```
springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/entity/user_roles.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
concessions.java employees.java Individual_Roles.java movies.java stores.java transactionRequest.java transactions.java
1 package OTC.StPaul.Final.Project.entity;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7
8 @Data
9 @Builder
10 @NoArgsConstructor
11 @AllArgsConstructor
12 public class user_roles {
13     private Long role_id;
14     private String emp_username;
15     private String user_role;
16 }
17 // last bracket
18
```



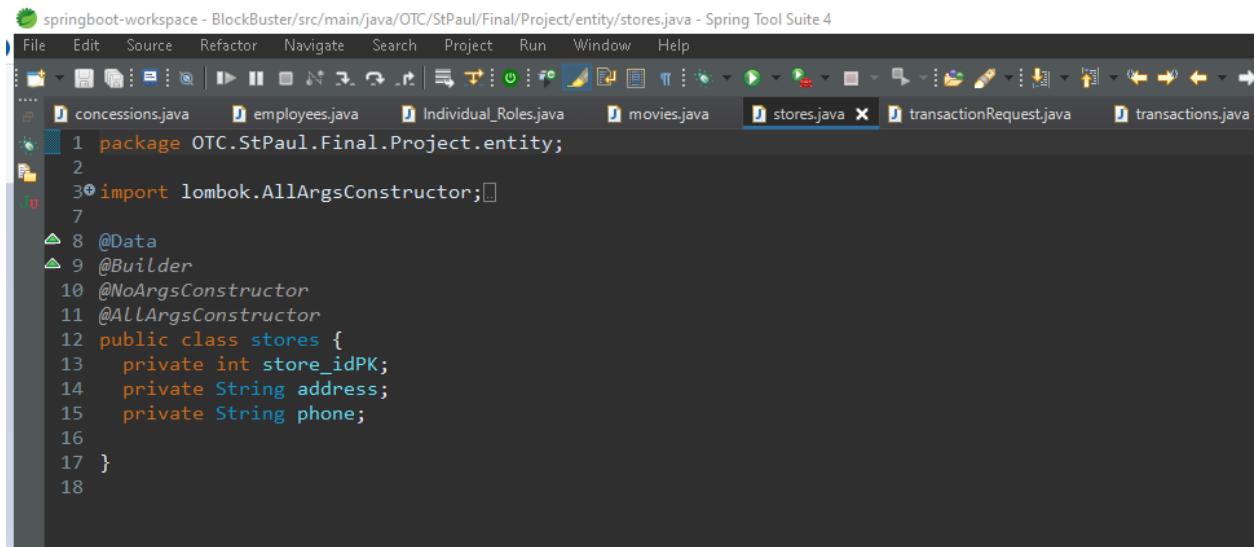
```
springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/entity/Individual_Roles.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
concessions.java employees.java Individual_Roles.java movies.java stores.java transactionRequest.java transactions.java
1 package OTC.StPaul.Final.Project.entity;
2
3 public enum Individual_Roles {
4     ADMIN, USER
5 }
6
```

Customers –



```
springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/entity/customers.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
customers.java
1 package OTC.StPaul.Final.Project.entity;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7
8 @Data
9 @Builder
10 @NoArgsConstructor
11 @AllArgsConstructor
12 public class customers {
13     private int customer_idPK;
14     private String first_name;
15     private String last_name;
16     private String address;
17     private String phone;
18 }
19
20
```

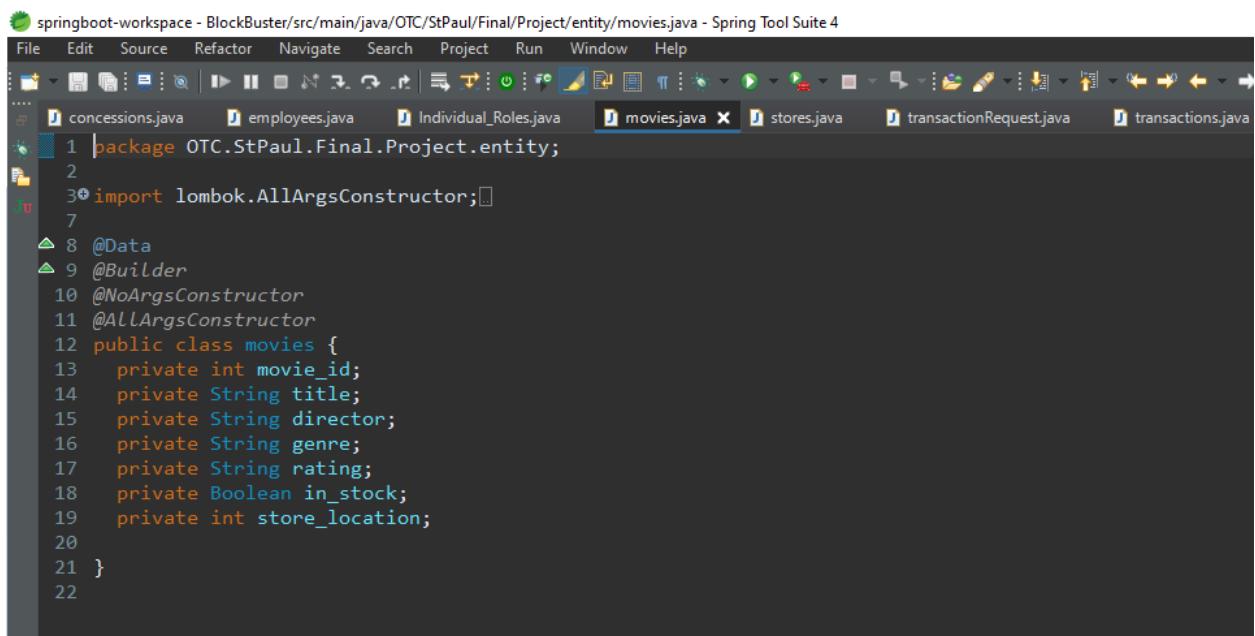
Stores-



A screenshot of the Spring Tool Suite 4 interface. The title bar says "springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/entity/stores.java - Spring Tool Suite 4". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, and Run. The code editor shows the following Java code:

```
1 package OTC.StPaul.Final.Project.entity;
2
3 import lombok.NoArgsConstructor;
4
5 import lombok.NonNull;
6
7 import lombok.Builder;
8 import lombok.Data;
9
10 import lombok.NoArgsConstructor;
11 import lombok.NonNull;
12 import lombok.Builder;
13 import lombok.Data;
14
15 import javax.persistence.Entity;
16 import javax.persistence.GeneratedValue;
17 import javax.persistence.GenerationType;
18 import javax.persistence.Id;
```

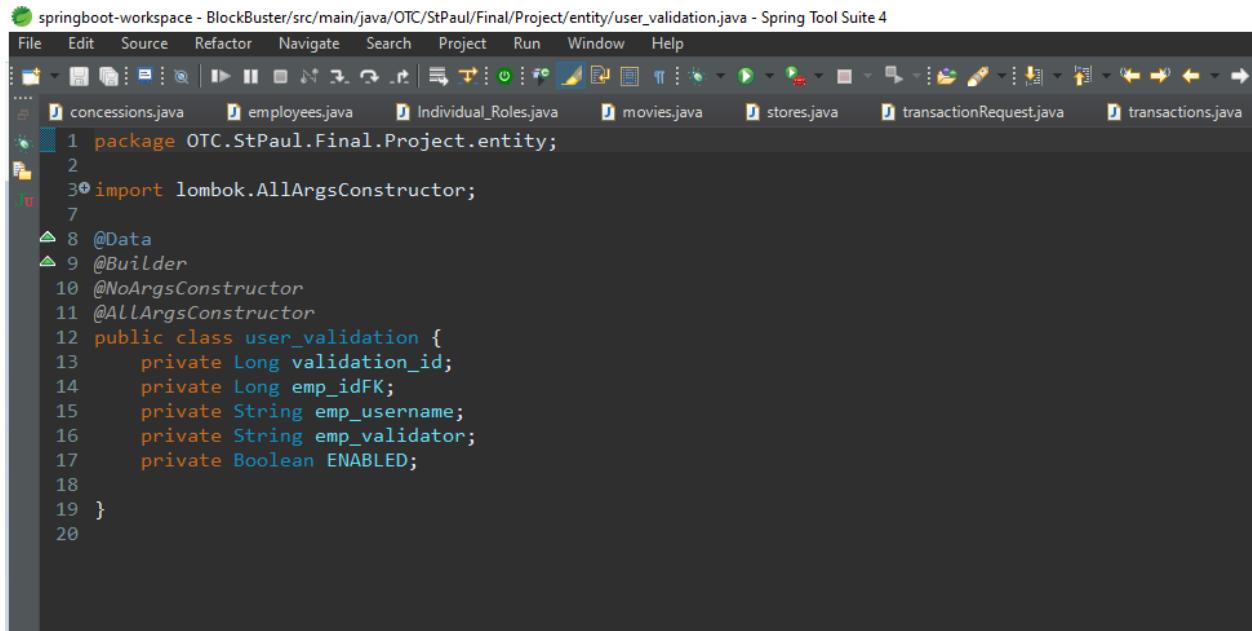
Movies-



A screenshot of the Spring Tool Suite 4 interface. The title bar says "springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/entity/movies.java - Spring Tool Suite 4". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, and Run. The code editor shows the following Java code:

```
1 package OTC.StPaul.Final.Project.entity;
2
3 import lombok.NoArgsConstructor;
4
5 import lombok.NonNull;
6
7 import lombok.Builder;
8 import lombok.Data;
9
10 import lombok.NoArgsConstructor;
11 import lombok.NonNull;
12 import lombok.Builder;
13 import lombok.Data;
14
15 import javax.persistence.Entity;
16 import javax.persistence.GeneratedValue;
17 import javax.persistence.GenerationType;
18 import javax.persistence.Id;
```

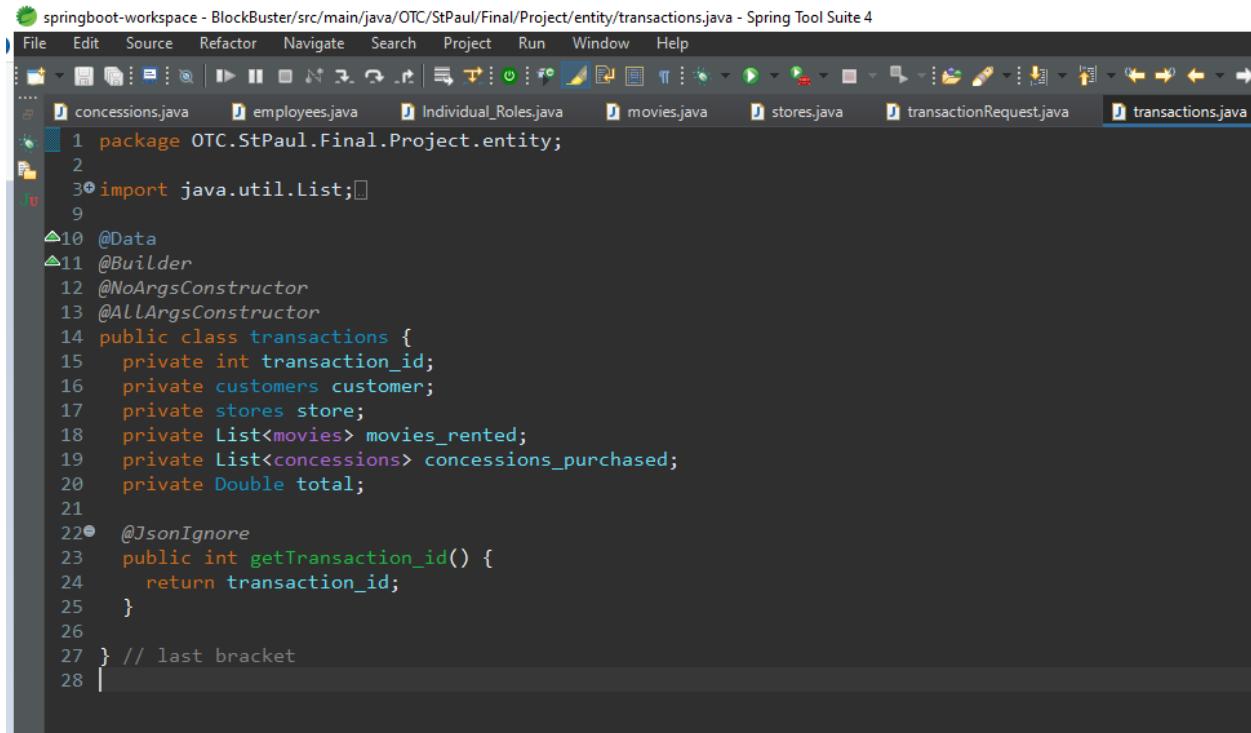
Concessions-



The screenshot shows the Spring Tool Suite 4 interface with the file `user_validation.java` open in the editor. The code defines a class `user_validation` with various fields and annotations. The code editor has a dark theme with syntax highlighting.

```
1 package OTC.StPaul.Final.Project.entity;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7
8 @Data
9 @Builder
10 @NoArgsConstructor
11 @AllArgsConstructor
12 public class user_validation {
13     private Long validation_id;
14     private Long emp_idFK;
15     private String emp_username;
16     private String emp_validator;
17     private Boolean ENABLED;
18
19 }
20
```

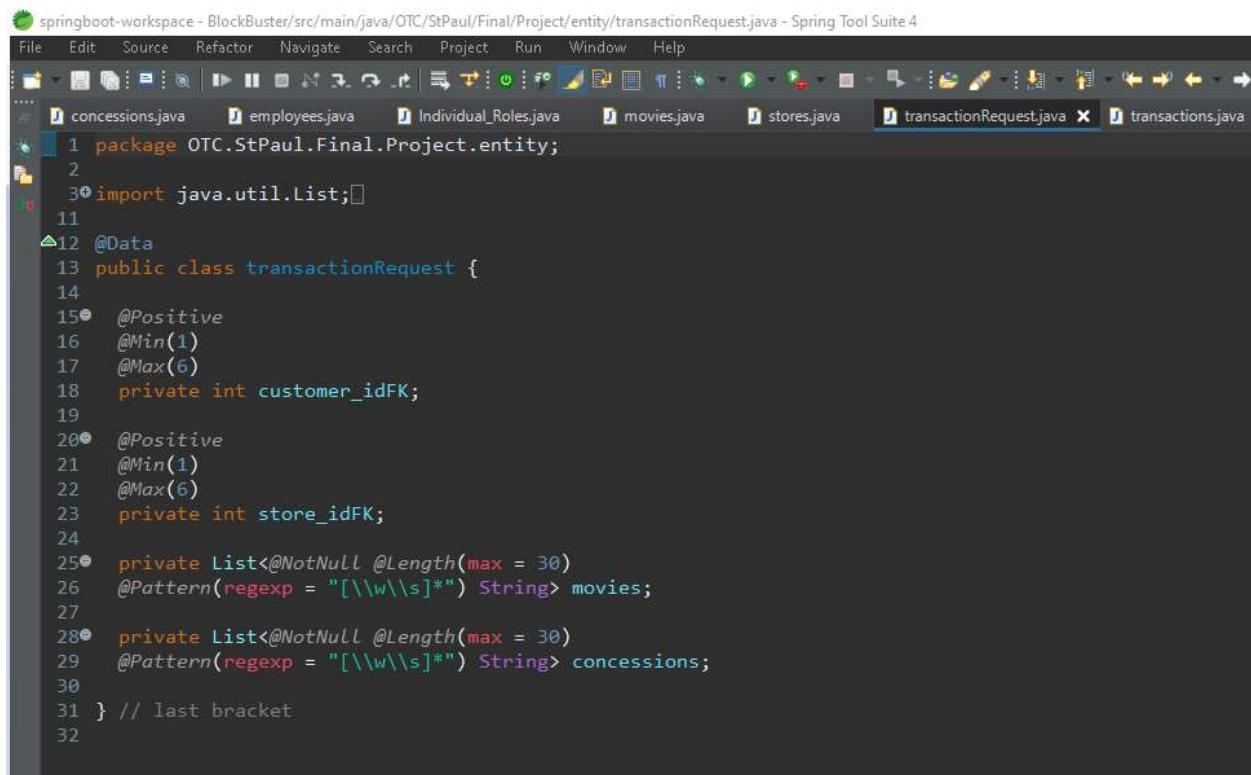
Transactions-



springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/entity/transactions.java - Spring Tool Suite 4

```
File Edit Source Refactor Navigate Search Project Run Window Help
concessions.java employees.java Individual_Roles.java movies.java stores.java transactionRequest.java transactions.java

1 package OTC.StPaul.Final.Project.entity;
2
3 import java.util.List;
4
5
6 @Data
7 @Builder
8 @NoArgsConstructor
9 @AllArgsConstructor
10 public class transactions {
11     private int transaction_id;
12     private customers customer;
13     private stores store;
14     private List<movies> movies_rented;
15     private List<concessions> concessions_purchased;
16     private Double total;
17
18     @JsonIgnore
19     public int getTransaction_id() {
20         return transaction_id;
21     }
22 }
23
24 } // last bracket
25
26
27
28 }
```



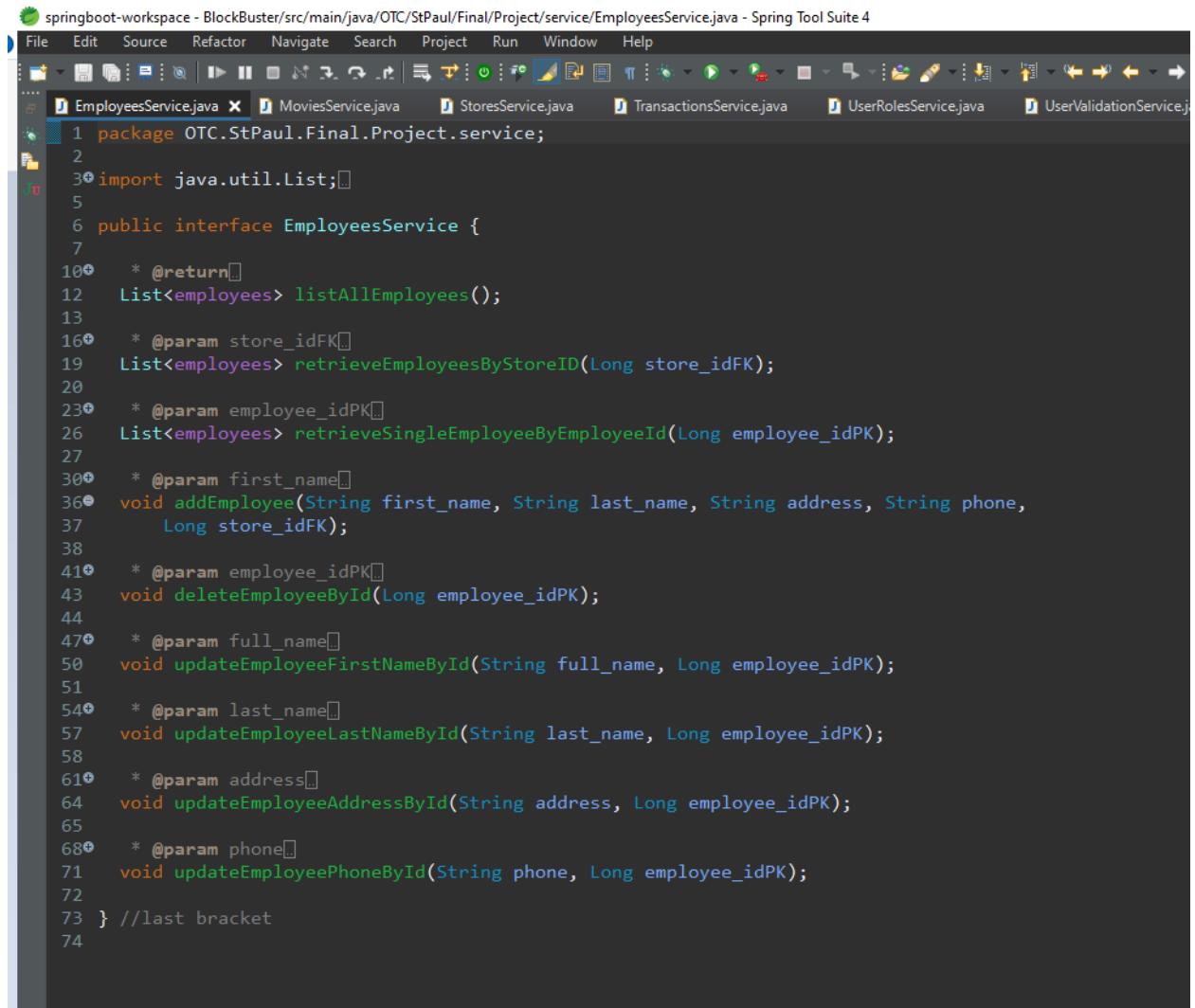
springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/entity/transactionRequest.java - Spring Tool Suite 4

```
File Edit Source Refactor Navigate Search Project Run Window Help
concessions.java employees.java Individual_Roles.java movies.java stores.java transactionRequest.java transactions.java

1 package OTC.StPaul.Final.Project.entity;
2
3 import java.util.List;
4
5
6 @Data
7 public class transactionRequest {
8     @Positive
9     @Min(1)
10    @Max(6)
11    private int customer_idFK;
12
13    @Positive
14    @Min(1)
15    @Max(6)
16    private int store_idFK;
17
18    private List<@NotNull @Length(max = 30)
19    @Pattern(regexp = "[\\w\\s]*") String> movies;
20
21    private List<@NotNull @Length(max = 30)
22    @Pattern(regexp = "[\\w\\s]*") String> concessions;
23
24
25 } // last bracket
26
27
28
29
30
31
32 }
```

Service

Employees-



The screenshot shows the Spring Tool Suite 4 interface with the EmployeesService.java file open in the editor. The code defines a public interface for managing employees, including methods for listing all employees, retrieving employees by store ID or employee ID, adding a new employee, deleting an employee, and updating employee details like first name, last name, address, and phone number.

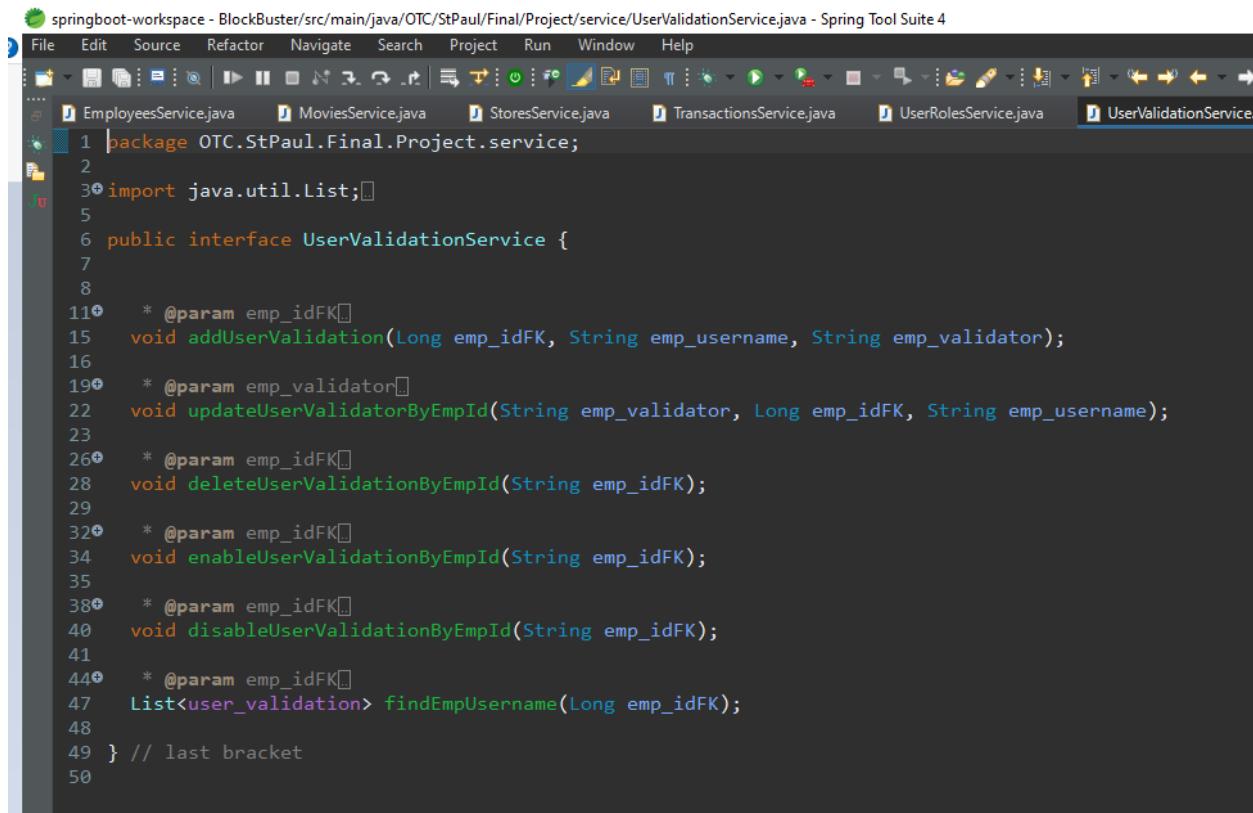
```
1 package OTC.StPaul.Final.Project.service;
2
3 import java.util.List;
4
5 public interface EmployeesService {
6
7     * @return
8     List<employees> listAllEmployees();
9
10    * @param store_idFK
11    List<employees> retrieveEmployeesByStoreID(Long store_idFK);
12
13    * @param employee_idPK
14    List<employees> retrieveSingleEmployeeByEmployeeId(Long employee_idPK);
15
16    * @param first_name
17    void addEmployee(String first_name, String last_name, String address, String phone,
18                      Long store_idFK);
19
20    * @param employee_idPK
21    void deleteEmployeeById(Long employee_idPK);
22
23    * @param full_name
24    void updateEmployeeFirstNameById(String full_name, Long employee_idPK);
25
26    * @param last_name
27    void updateEmployeeLastNameById(String last_name, Long employee_idPK);
28
29    * @param address
30    void updateEmployeeAddressById(String address, Long employee_idPK);
31
32    * @param phone
33    void updateEmployeePhoneById(String phone, Long employee_idPK);
34
35 } //last bracket
36
```

springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/service/DefaultEmployeesService.java - Spring Tool Suite 4

The screenshot shows the Spring Tool Suite 4 interface with the code editor open to the file DefaultEmployeesService.java. The file contains Java code for a service layer, specifically for managing employees. The code includes imports for java.util.List and OTC.StPaul.Final.Project.service, and defines a class DefaultEmployeesService that implements EmployeesService. It uses autowiring to inject an EmployeesDao and overrides various methods to interact with the database. The code is well-formatted with line numbers and color-coded syntax highlighting.

```
1 package OTC.StPaul.Final.Project.service;
2
3 import java.util.List;
4
5 @Service
6 public class DefaultEmployeesService implements EmployeesService{
7
8     @Autowired
9     private EmployeesDao employeesDao;
10
11     @Override
12     public List<employees> listAllEmployees() {
13         return employeesDao.listAllEmployees();
14     }
15
16     @Override
17     public List<employees> retrieveEmployeesByStoreID(Long store_idFK) {
18         return employeesDao.retrieveEmployeesByStoreID(store_idFK);
19     }
20
21     @Override
22     public List<employees> retrieveSingleEmployeeByEmployeeId(Long employee_idPK) {
23         return employeesDao.retrieveSingleEmployeeByEmployeeId(employee_idPK);
24     }
25
26     @Override
27     public void addEmployee(String first_name, String last_name, String address, String phone,
28             Long store_idFK) {
29         employeesDao.addEmployee(first_name, last_name, address, phone, store_idFK);
30     }
31
32     @Override
33     public void deleteEmployeeById(Long employee_idPK) {
34         employeesDao.deleteEmployeeById(employee_idPK);
35     }
36
37     @Override
38     public void updateEmployeeFirstNameById(String full_name, Long employee_idPK) {
39         employeesDao.updateEmployeeFirstNameById(full_name, employee_idPK);
40     }
41
42     @Override
43     public void updateEmployeeLastNameById(String last_name, Long employee_idPK) {
44         employeesDao.updateEmployeeLastNameById(last_name, employee_idPK);
45     }
46
47     @Override
48     public void updateEmployeeAddressById(String address, Long employee_idPK) {
49         employeesDao.updateEmployeeAddressById(address, employee_idPK);
50     }
51
52     @Override
53     public void updateEmployeePhoneById(String phone, Long employee_idPK) {
54         employeesDao.updateEmployeePhoneById(phone, employee_idPK);
55     }
56
57     @Override
58     public void updateEmployeeAddressById(String address, Long employee_idPK) {
59         employeesDao.updateEmployeeAddressById(address, employee_idPK);
60     }
61 } // last bracket
62
```

User Validation-



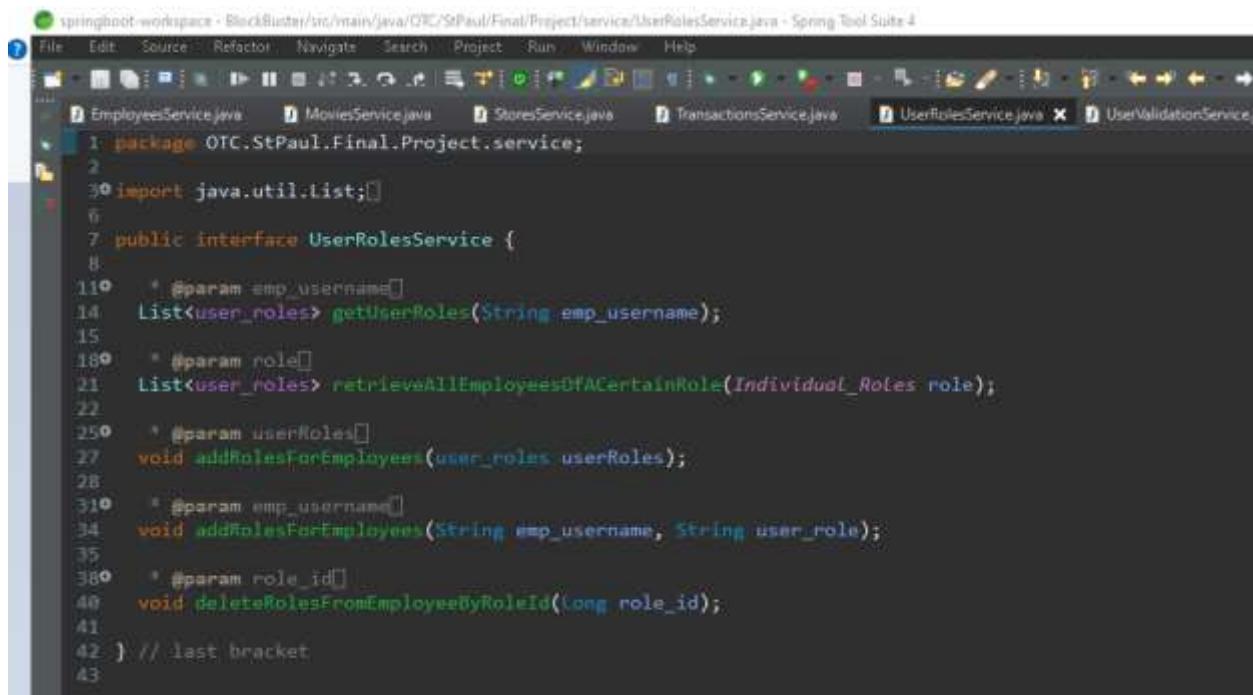
The screenshot shows the Spring Tool Suite 4 interface with the project 'springboot-workspace' open. The 'UserValidationService.java' file is the active editor. The code defines a public interface for user validation operations, including methods for adding, updating, deleting, enabling, and disabling user validations based on employee IDs and usernames.

```
1 package OTC.StPaul.Final.Project.service;
2
3 import java.util.List;
4
5 public interface UserValidationService {
6
7
8
11     * @param emp_idFK
12     void addUserValidation(Long emp_idFK, String emp_username, String emp_validator);
13
16     * @param emp_validator
17     void updateUserValidatorByEmpId(String emp_validator, Long emp_idFK, String emp_username);
18
21     * @param emp_idFK
22     void deleteUserValidationByEmpId(String emp_idFK);
23
26     * @param emp_idFK
27     void enableUserValidationByEmpId(String emp_idFK);
28
31     * @param emp_idFK
32     void disableUserValidationByEmpId(String emp_idFK);
33
36     * @param emp_idFK
37     List<user_validation> findEmpUsername(Long emp_idFK);
38
39 }
40 // last bracket
```

springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/service/DefaultUserValidationService.java - Spring Tool Suite 4

```
File Edit Source Refactor Navigate Search Project Run Window Help
DefaultConcessionsService.java DefaultCustomersService.java DefaultEmployeesService.java DefaultMoviesService.java DefaultStoresService.java
1 package OTC.StPaul.Final.Project.service;
2
3 import java.util.List;
4
5 @Service
6 public class DefaultUserValidationService implements UserValidationService{
7
8     @Autowired
9     private UserValidationDao userValidationDao;
10
11     @Override
12     public void addUserValidation(Long emp_idFK, String emp_username, String emp_validator) {
13         userValidationDao.addUserValidation(emp_idFK, emp_username, emp_validator);
14     }
15
16     @Override
17     public void updateUserValidatorByEmpId(String emp_validator, Long emp_idFK, String emp_username) {
18         userValidationDao.updateUserValidatorByEmpId(emp_validator, emp_idFK, emp_username);
19     }
20
21     @Override
22     public void deleteUserValidationByEmpId(String emp_idFK) {
23         userValidationDao.deleteUserValidationByEmpId(emp_idFK);
24     }
25
26     @Override
27     public void enableUserValidationByEmpId(String emp_idFK) {
28         userValidationDao.enableUserValidationByEmpId(emp_idFK);
29     }
30
31     @Override
32     public void disableUserValidationByEmpId(String emp_idFK) {
33         userValidationDao.disableUserValidationByEmpId(emp_idFK);
34     }
35
36     @Override
37     public List<user_validation> findEmpUsername(Long emp_idFK) {
38         return userValidationDao.findEmpUsername(emp_idFK);
39     }
40
41 } // last bracket
```

User Roles –



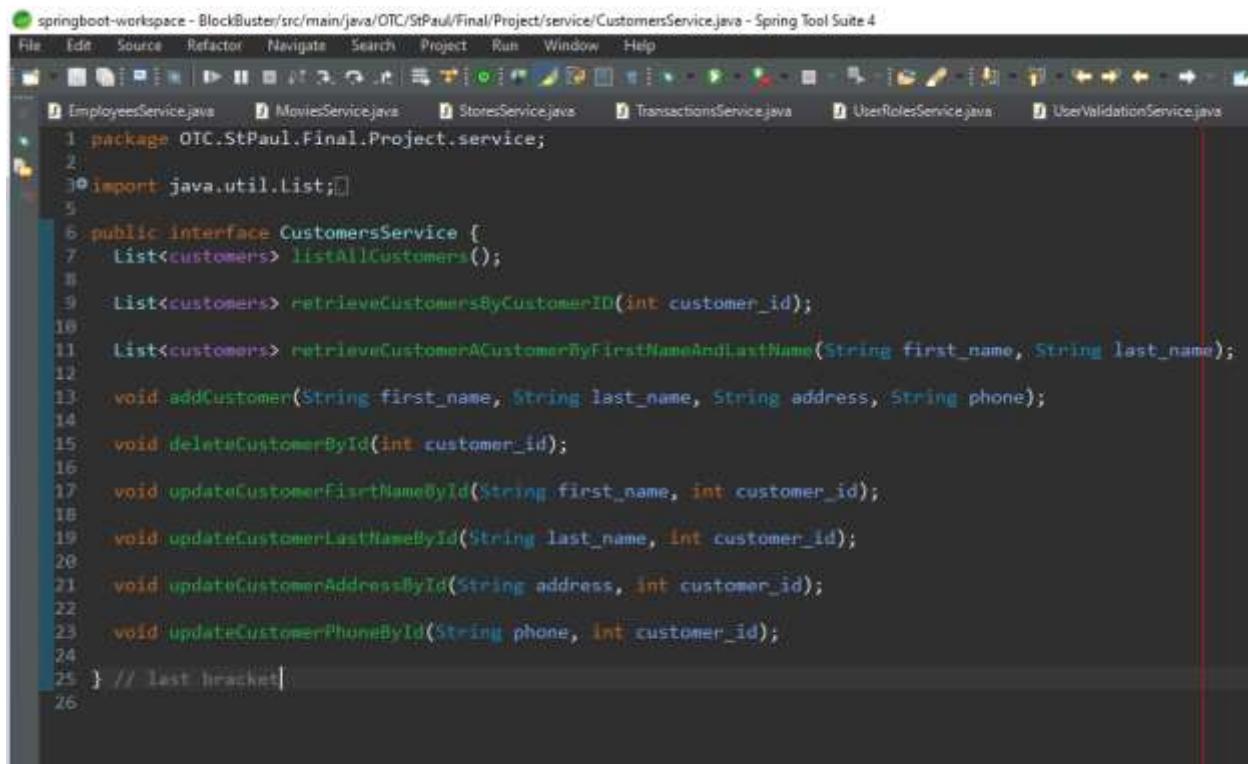
The screenshot shows the Spring Tool Suite 4 interface with the 'UserRolesService.java' file open in the editor. The code defines a public interface for managing user roles. It includes methods for getting user roles by employee username, retrieving all employees for a certain role, adding roles for employees, adding roles for an employee by username, and deleting roles from an employee by role ID.

```
1 package OTC.StPaul.Final.Project.service;
2
3 import java.util.List;
4
5 public interface UserRolesService {
6
7     * @param emp_username[]
8     List<user_roles> getUserRoles(String emp_username);
9
10    * @param role[]
11    List<user_roles> retrieveAllEmployeesOfACertainRole(Individual_Roles role);
12
13    * @param userRoles[]
14    void addRolesForEmployees(user_roles userRoles);
15
16    * @param emp_username[]
17    void addRolesForEmployee(String emp_username, String user_role);
18
19    * @param role_id[]
20    void deleteRolesFromEmployeeByRoleId(long role_id);
21
22 } // last bracket
23
```

springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/service/DefaultUserRolesService.java - Spring Tool Suite 4

```
File Edit Source Refactor Navigate Search Project Run Window Help
DefaultConcessionsService.java DefaultCustomersService.java DefaultEmployeesService.java DefaultMoviesService.java DefaultStoresService.java
1 package OTC.StPaul.Final.Project.service;
2
3+ import java.util.List;
4
5
6 @Service
7 public class DefaultUserRolesService implements UserRolesService{
8
9     @Autowired
10    private UserRolesDao userRolesDao;
11
12
13@override
14    public List<user_roles> getUserRoles(String emp_username) {
15        return userRolesDao.getUserRoles(emp_username);
16    }
17
18
19@override
20    public List<user_roles> retrieveAllEmployeesOfACertainRole(Individual_Roles role) {
21        return userRolesDao.retrieveAllEmployeesOfACertainRole(role);
22    }
23
24
25
26@override
27    public void addRolesForEmployees(user_roles userRoles) {
28        userRolesDao.addRolesForEmployees(userRoles);
29    }
30
31
32@override
33    public void addRolesForEmployees(String emp_username, String user_role) {
34        userRolesDao.addRolesForEmployees(emp_username, user_role);
35    }
36
37@override
38    public void deleteRolesFromEmployeeByRoleId(Long role_id) {
39        userRolesDao.deleteRolesFromEmployeeByRoleId(role_id);
40    }
41 } // last bracket
42
```

Customers –



The screenshot shows the Spring Tool Suite 4 interface with the title bar "springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/service/CustomersService.java - Spring Tool Suite 4". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, and Build. Below the toolbar is a tab bar with "EmployeesService.java", "MoviesService.java", "StoresService.java", "TransactionsService.java", "UserRolesService.java", and "UserValidationService.java". The main code editor area contains the following Java code:

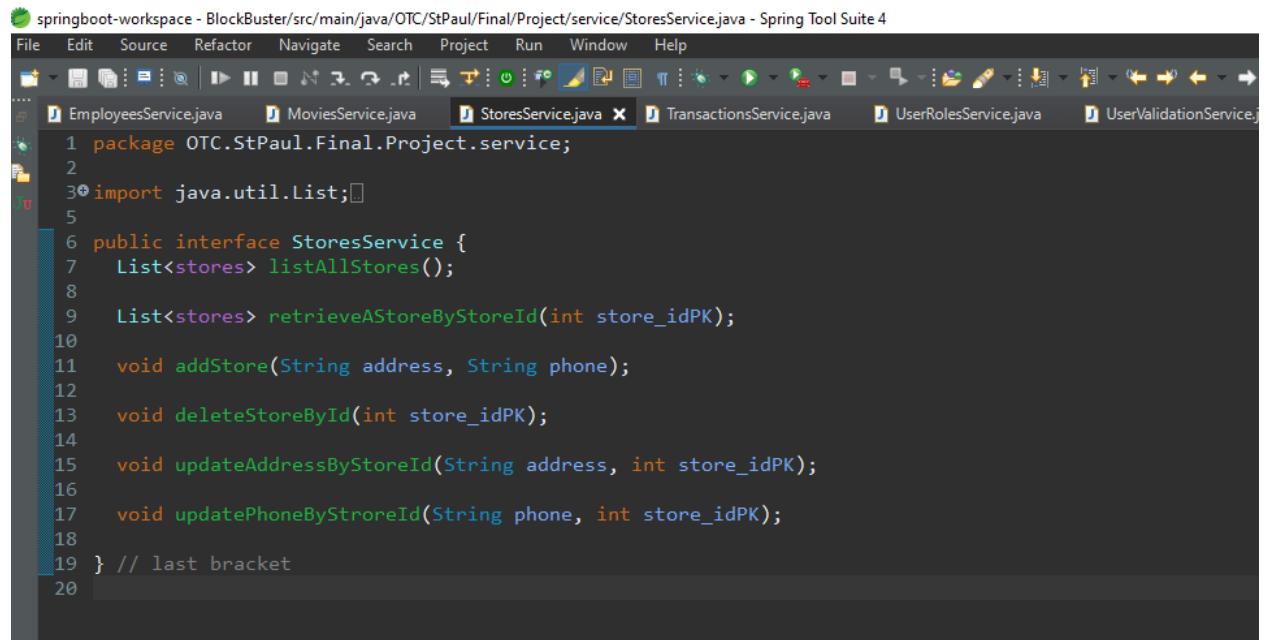
```
1 package OTC.StPaul.Final.Project.service;
2
3 import java.util.List;
4
5 public interface CustomersService {
6     List<customers> listAllCustomers();
7
8     List<customers> retrieveCustomersByCustomerId(int customer_id);
9
10    List<customers> retrieveCustomerACustomerByFirstNameAndLastName(String first_name, String last_name);
11
12    void addCustomer(String first_name, String last_name, String address, String phone);
13
14    void deleteCustomerById(int customer_id);
15
16    void updateCustomerFirstNameById(String first_name, int customer_id);
17
18    void updateCustomerLastNameById(String last_name, int customer_id);
19
20    void updateCustomerAddressById(String address, int customer_id);
21
22    void updateCustomerPhoneById(String phone, int customer_id);
23
24
25 } // last bracket
26
```

springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/service/DefaultCustomersService.java - Spring Tool Suite 4

```
File Edit Source Refactor Navigate Search Project Run Window Help
DefaultCustomersService.java X DefaultEmployeesService.java DefaultMoviesService.java DefaultTransactionsService.java
1 package OTC.StPaul.Final.Project.service;
2
3 import java.util.List;
4
5 @Service
6 public class DefaultCustomersService implements CustomersService {
7
8     @Autowired
9     private CustomersDao customersDao;
10
11     @Override
12     public List<customers> listAllCustomers() {
13         return customersDao.listAllCustomers();
14     }
15
16     @Override
17     public List<customers> retrieveCustomersByCustomerID(int customer_id) {
18         return customersDao.retrieveCustomersByCustomerID(customer_id);
19     }
20
21     @Override
22     public List<customers> retrieveCustomerACustomerByFirstNameAndLastName(String first_name,
23         String last_name) {
24         return customersDao.retrieveCustomerACustomerByFirstNameAndLastName(first_name, last_name);
25     }
26
27     @Override
28     public void addCustomer(String first_name, String last_name, String address, String phone) {
29         customersDao.addCustomer(first_name, last_name, address, phone);
30     }
31
32     @Override
33     public void deleteCustomerById(int customer_id) {
34         customersDao.deleteCustomerById(customer_id);
35     }
36
37     @Override
38     public void updateCustomerFisrtNameById(String first_name, int customer_id) {
39         customersDao.updateCustomerFisrtNameById(first_name, customer_id);
40     }
41
42     @Override
43     public void updateCustomerLastNameById(String last_name, int customer_id) {
44         customersDao.updateCustomerLastNameById(last_name, customer_id);
45     }
46
47     @Override
48     public void updateCustomerAddressById(String address, int customer_id) {
49         customersDao.updateCustomerAddressById(address, customer_id);
50     }
51 }
```

```
52
53 }
54
55● @Override
56 public void updateCustomerAddressById(String address, int customer_id) {
57     customersDao.updateCustomerAddressById(address, customer_id);
58 }
59
60
61● @Override
62 public void updateCustomerPhoneById(String phone, int customer_id) {
63     customersDao.updateCustomerPhoneById(phone, customer_id);
64 }
65
66
67 } // last bracket
68
```

Stores-



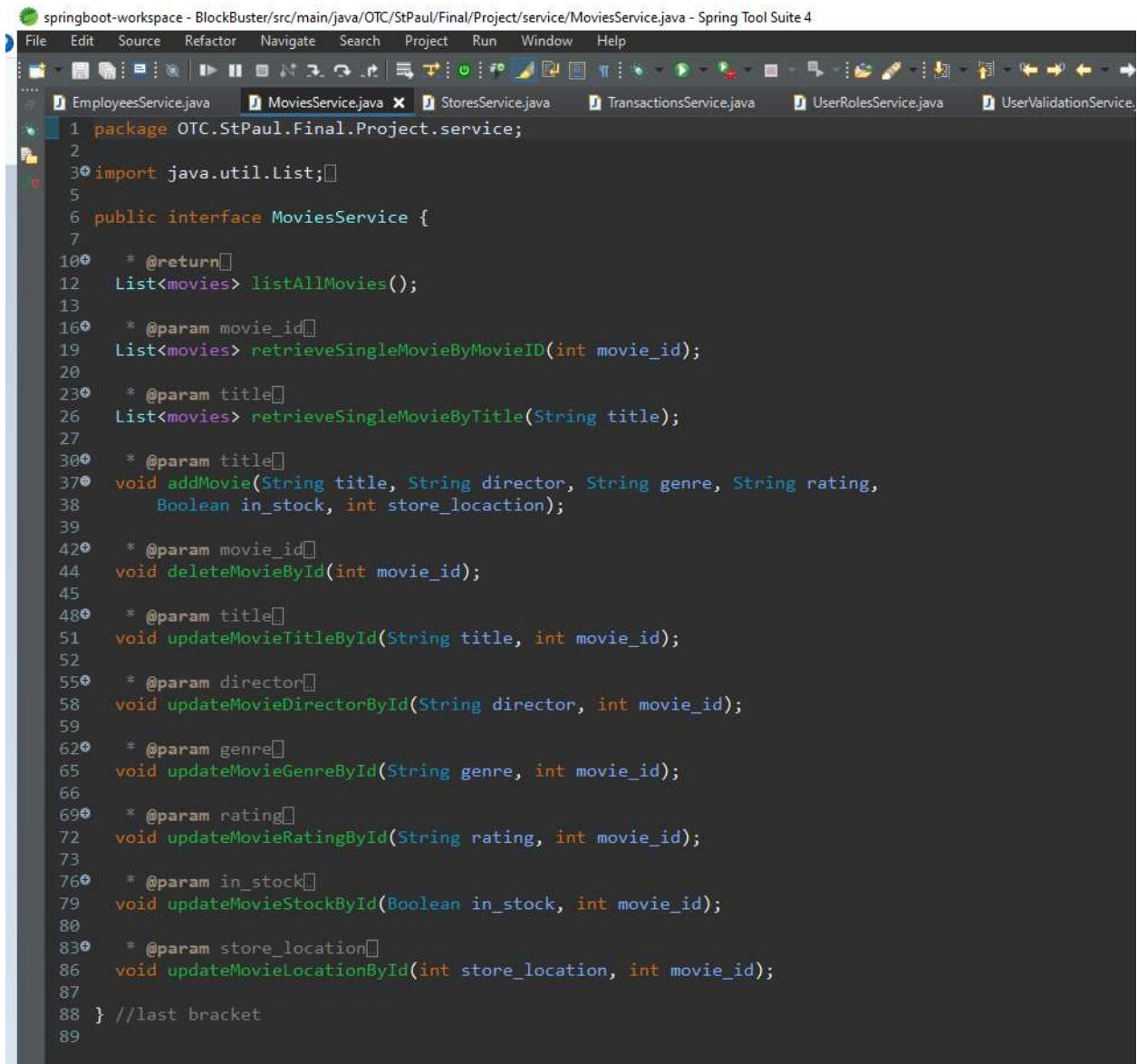
The screenshot shows the Spring Tool Suite 4 interface with the 'StoresService.java' file open in the central editor window. The file contains Java code defining a service interface for stores. The code includes imports, a package declaration, and several methods for managing store data.

```
springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/service/StoresService.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
EmployeesService.java MoviesService.java StoresService.java TransactionsService.java UserRolesService.java UserValidationService.java
1 package OTC.StPaul.Final.Project.service;
2
3 import java.util.List;
4
5 public interface StoresService {
6     List<stores> listAllStores();
7
8     List<stores> retrieveAStoreByStoreId(int store_idPK);
9
10    void addStore(String address, String phone);
11
12    void deleteStoreById(int store_idPK);
13
14    void updateAddressByStoreId(String address, int store_idPK);
15
16    void updatePhoneByStroreId(String phone, int store_idPK);
17
18 }
19 // last bracket
20
```

springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/service/DefaultStoresService.java - Spring Tool Suite 4

```
File Edit Source Refactor Navigate Search Project Run Window Help
DefaultConcessionsService.java DefaultCustomersService.java DefaultEmployeesService.java DefaultMoviesService.java DefaultStoresService.java
1 package OTC.StPaul.Final.Project.service;
2
3 import java.util.List;
4
5 @Service
6 public class DefaultStoresService implements StoresService {
7
8     @Autowired
9     private StoresDao storesDao;
10
11     @Override
12     public List<stores> listAllStores() {
13         return storesDao.listAllStores();
14     }
15
16     @Override
17     public List<stores> retrieveAStoreByStoreId(int store_idPK) {
18         return storesDao.retrieveAStoreByStoreId(store_idPK);
19     }
20
21     @Override
22     public void addStore(String address, String phone) {
23         storesDao.addStore(address, phone);
24     }
25
26     @Override
27     public void deleteStoreById(int store_idPK) {
28         storesDao.deleteStoreById(store_idPK);
29     }
30
31     @Override
32     public void updateAddressByStoreId(String address, int store_idPK) {
33         storesDao.updateAddressByStoreId(address, store_idPK);
34     }
35
36     @Override
37     public void updatePhoneByStoreId(String phone, int store_idPK) {
38         storesDao.updatePhoneByStoreId(phone, store_idPK);
39     }
40
41     @Override
42     public void updatePhoneByStoreId(String phone, int store_idPK) {
43         storesDao.updatePhoneByStoreId(phone, store_idPK);
44     }
45
46     }
47
48 }
49 } // last bracket|
```

Movies-



The screenshot shows the Spring Tool Suite 4 interface with the 'MoviesService.java' file open in the editor. The code defines a RESTful service for movies, utilizing annotations like @Param and @Return. The code is as follows:

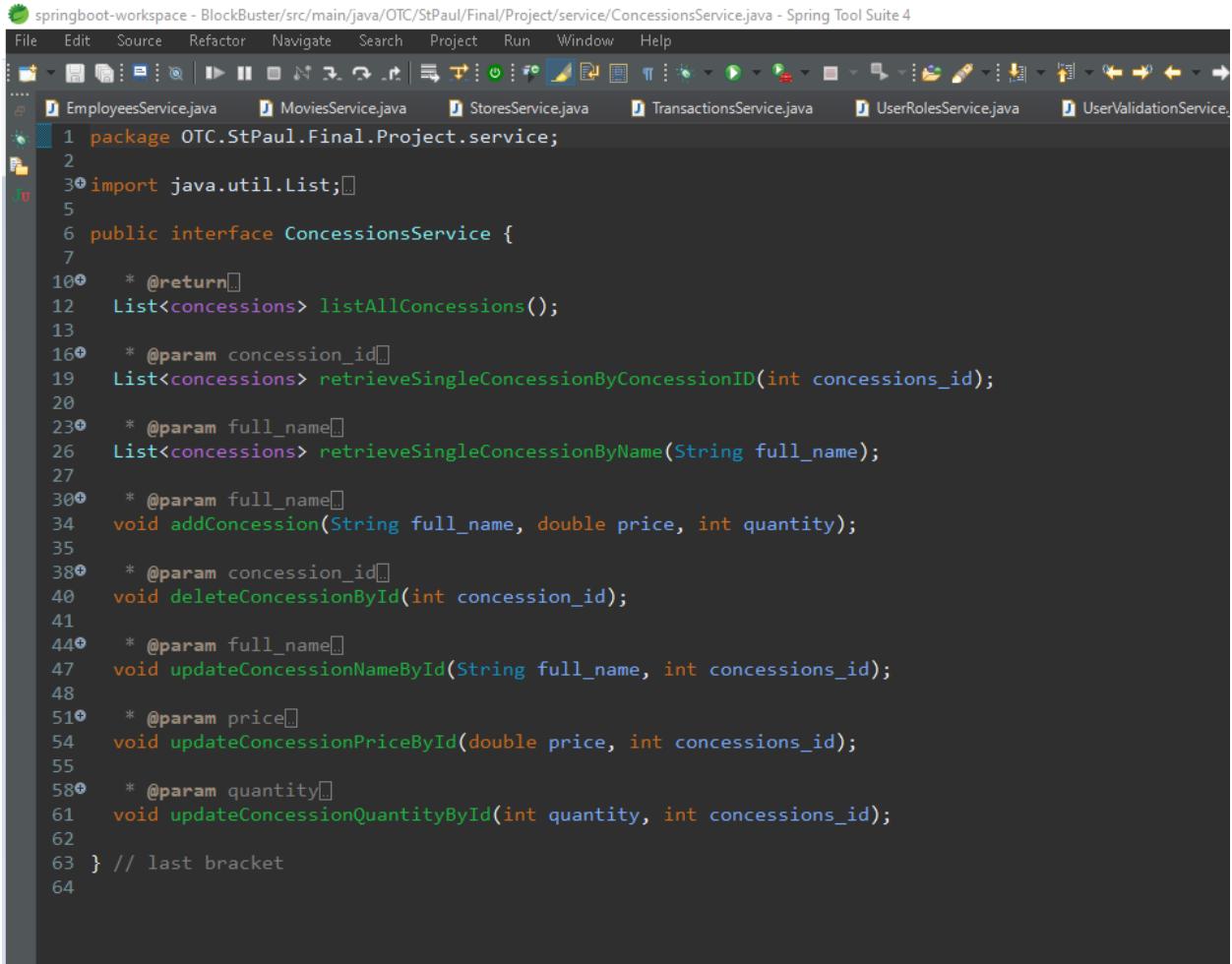
```
1 package OTC.StPaul.Final.Project.service;
2
3 import java.util.List;
4
5 public interface MoviesService {
6
7     /**
8      * @return
9     */
10    List<movies> listAllMovies();
11
12    /**
13     * @param movie_id
14     */
15    List<movies> retrieveSingleMovieByMovieID(int movie_id);
16
17    /**
18     * @param title
19     */
20    List<movies> retrieveSingleMovieByTitle(String title);
21
22    /**
23     * @param title
24     */
25    void addMovie(String title, String director, String genre, String rating,
26                  Boolean in_stock, int store_locaction);
27
28    /**
29     * @param movie_id
30     */
31    void deleteMovieById(int movie_id);
32
33    /**
34     * @param title
35     */
36    void updateMovieTitleById(String title, int movie_id);
37
38    /**
39     * @param director
40     */
41    void updateMovieDirectorById(String director, int movie_id);
42
43    /**
44     * @param genre
45     */
46    void updateMovieGenreById(String genre, int movie_id);
47
48    /**
49     * @param rating
50     */
51    void updateMovieRatingById(String rating, int movie_id);
52
53    /**
54     * @param in_stock
55     */
56    void updateMovieStockById(Boolean in_stock, int movie_id);
57
58    /**
59     * @param store_location
60     */
61    void updateMovieLocationById(int store_location, int movie_id);
62
63 }
64 //last bracket
```

springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/service/DefaultMoviesService.java - Spring Tool Suite 4

```
File Edit Source Refactor Navigate Search Project Run Window Help
DefaultMoviesService.java X DefaultTransactionsService.java
1 package OTC.StPaul.Final.Project.service;
2
3 import java.util.List;
4
5 @Service
6 public class DefaultMoviesService implements MoviesService {
7
8     @Autowired
9     private MoviesDao moviesDao;
10
11     @Override
12     public List<movies> listAllMovies() {
13         return moviesDao.listAllMovies();
14     }
15
16     @Override
17     public List<movies> retrieveSingleMovieByMovieID(int movie_id) {
18         return moviesDao.retrieveSingleMovieByMovieID(movie_id);
19     }
20
21     @Override
22     public List<movies> retrieveSingleMovieByTitle(String title) {
23         return moviesDao.retrieveSingleMovieByTitle(title);
24     }
25
26     @Override
27     public void addMovie(String title, String director, String genre, String rating, Boolean in_stock,
28             int store_location) {
29         moviesDao.addMovie(title, director, genre, rating, in_stock, store_location);
30     }
31
32     @Override
33     public void deleteMovieById(int movie_id) {
34         moviesDao.deleteMovieById(movie_id);
35     }
36
37     @Override
38     public void updateMovieTitleById(String title, int movie_id) {
39         moviesDao.updateMovieTitleById(title, movie_id);
40     }
41
42     @Override
43     public void updateMovieDirectorById(String director, int movie_id) {
44         moviesDao.updateMovieDirectorById(director, movie_id);
45     }
46
47     @Override
48     public void updateMovieRatingById(String rating, int movie_id) {
49         moviesDao.updateMovieRatingById(rating, movie_id);
50     }
51     @Override
```

```
▲52  public void updateMovieGenreById(String genre, int movie_id) {
53      moviesDao.updateMovieGenreById(genre, movie_id);
54  }
55
56● @Override
▲57  public void updateMovieRatingById(String rating, int movie_id) {
58      moviesDao.updateMovieRatingById(rating, movie_id);
59  }
60
61● @Override
▲62  public void updateMovieStockById(Boolean in_stock, int movie_id) {
63      moviesDao.updateMovieStockById(in_stock, movie_id);
64  }
65
66● @Override
▲67  public void updateMovieLocationById(int store_location, int movie_id) {
68      moviesDao.updateMovieLocationById(store_location, movie_id);
69  }
70
71 } //last bracket
72
```

Concessions-



The screenshot shows the Spring Tool Suite 4 interface with the file 'ConcessionsService.java' open in the editor. The code defines a public interface for managing concessions. It includes methods for listing all concessions, retrieving a single concession by ID, adding a new concession, deleting a concession by ID, updating a concession's name, updating its price, and updating its quantity.

```
1 package OTC.StPaul.Final.Project.service;
2
3+ import java.util.List;
4
5
6 public interface ConcessionsService {
7
8+     * @return
9     List<concessions> listAllConcessions();
10
11+    * @param concession_id
12     List<concessions> retrieveSingleConcessionByConcessionID(int concessions_id);
13
14+    * @param full_name
15     List<concessions> retrieveSingleConcessionByName(String full_name);
16
17+    * @param full_name
18     void addConcession(String full_name, double price, int quantity);
19
20+    * @param concession_id
21     void deleteConcessionById(int concession_id);
22
23+    * @param full_name
24     void updateConcessionNameById(String full_name, int concessions_id);
25
26+    * @param price
27     void updateConcessionPriceById(double price, int concessions_id);
28
29+    * @param quantity
30     void updateConcessionQuantityById(int quantity, int concessions_id);
31
32 } // last bracket
```

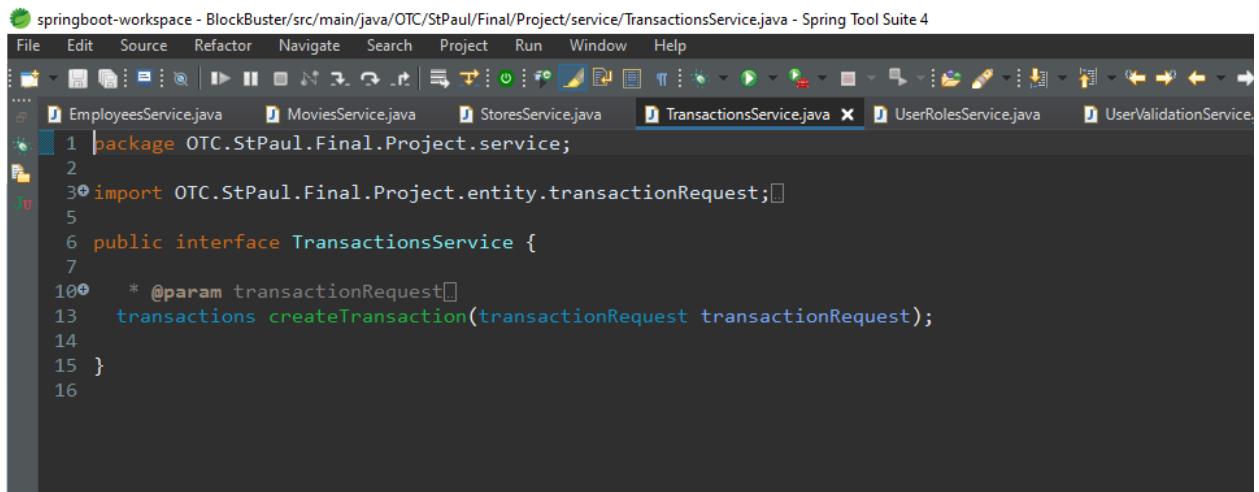
springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/service/DefaultConcessionsService.java - Spring Tool Suite 4

The screenshot shows the Spring Tool Suite 4 interface with the following details:

- Title Bar:** "springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/service/DefaultConcessionsService.java - Spring Tool Suite 4"
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Toolbar:** Includes icons for file operations like Open, Save, Find, and Run.
- Project Explorer:** Shows the project structure with files like DefaultConcessionsService.java, DefaultCustomersService.java, DefaultEmployeesService.java, DefaultMoviesService.java, and DefaultTransactions.java.
- Code Editor:** Displays the Java code for DefaultConcessionsService.java. The code implements the ConcessionsService interface and delegates to a ConcessionsDao. It includes methods for listing all concessions, retrieving a single concession by ID, adding a new concession, deleting a concession by ID, updating a concession's name by ID, updating its price by ID, and updating its quantity by ID.

```
1 package OTC.StPaul.Final.Project.service;
2
3 import java.util.List;
4
5 @Service
6 public class DefaultConcessionsService implements ConcessionsService {
7
8     @Autowired
9     private ConcessionsDao concessionsDao;
10
11     @Override
12     public List<concessions> listAllConcessions() {
13         return concessionsDao.listAllConcessions();
14     }
15
16     @Override
17     public List<concessions> retrieveSingleConcessionByConcessionID(int concessions_id) {
18         return concessionsDao.retrieveSingleConcessionByConcessionID(concessions_id);
19     }
20
21     @Override
22     public List<concessions> retrieveSingleConcessionByName(String full_name) {
23         return concessionsDao.retrieveSingleConcessionByName(full_name);
24     }
25
26     @Override
27     public void addConcession(String full_name, double price, int quantity) {
28         concessionsDao.addConcession(full_name, price, quantity);
29     }
30
31     @Override
32     public void deleteConcessionById(int concessions_id) {
33         concessionsDao.deleteConcessionById(concessions_id);
34     }
35
36     @Override
37     public void updateConcessionNameById(String full_name, int concessions_id) {
38         concessionsDao.updateConcessionNameById(full_name, concessions_id);
39     }
40
41     @Override
42     public void updateConcessionPriceById(double price, int concessions_id) {
43         concessionsDao.updateConcessionPriceById(price, concessions_id);
44     }
45
46     @Override
47     public void updateConcessionQuantityById(int quantity, int concessions_id) {
48
49     }
50
51     public void updateConcessionQuantityById(int quantity, int concessions_id) {
52         concessionsDao.updateConcessionQuantityById(quantity, concessions_id);
53     }
54
55 } // last bracket
56
```

Transactions-



The screenshot shows the Spring Tool Suite 4 interface with the title bar "springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/service/TransactionsService.java - Spring Tool Suite 4". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Find, and Run. Below the toolbar is a tab bar with several Java files: EmployeesService.java, MoviesService.java, StoresService.java, TransactionsService.java (which is currently selected), UserRolesService.java, and UserValidationService.java. The main editor area displays the code for the TransactionsService.java file:

```
1 package OTC.StPaul.Final.Project.service;
2
3 import OTC.StPaul.Final.Project.entity.transactionRequest;
4
5
6 public interface TransactionsService {
7
8     * @param transactionRequest
9     transactions createTransaction(transactionRequest transactionRequest);
10
11 }
12
13 }
```

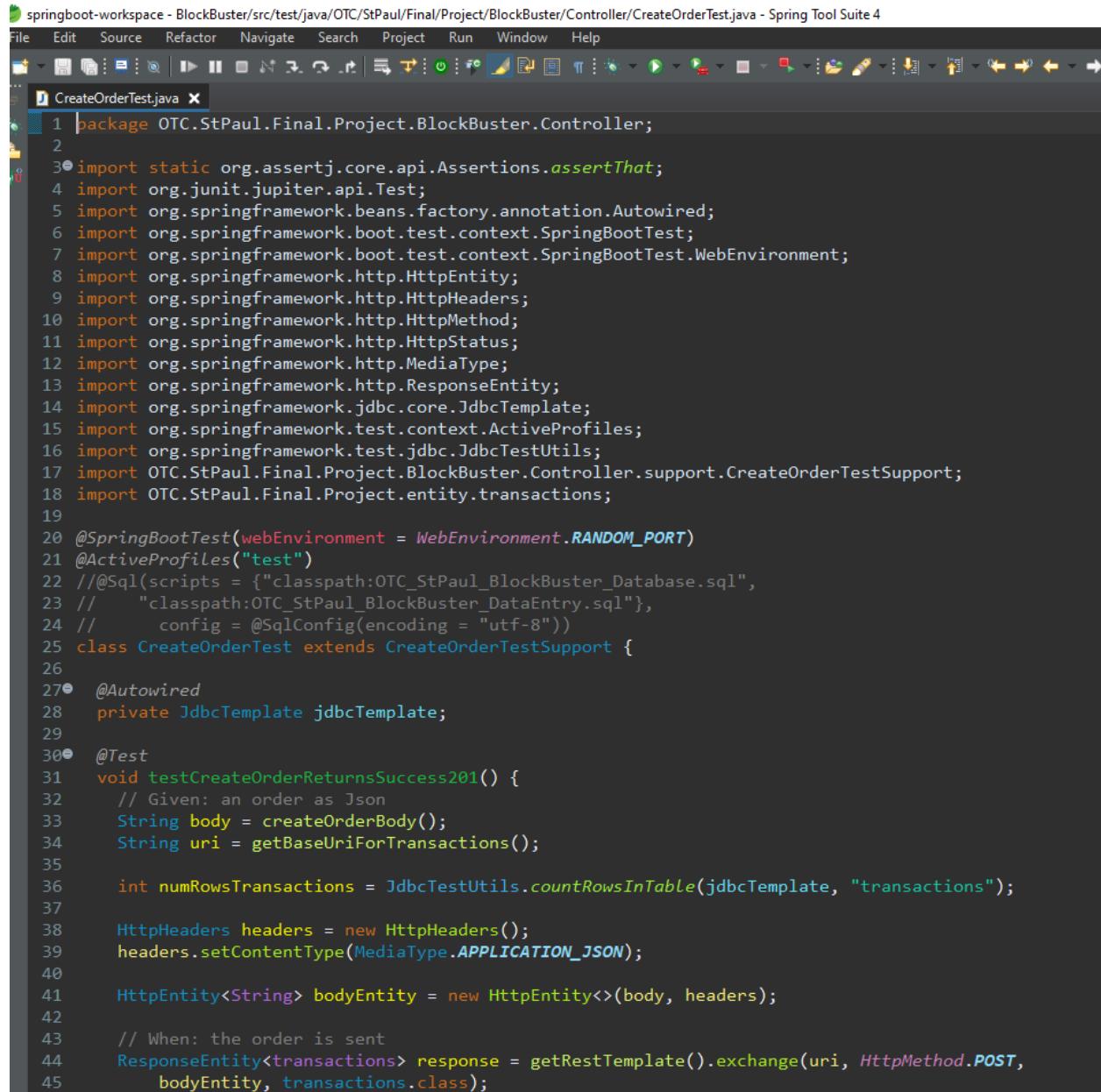
springboot-workspace - BlockBuster/src/main/java/OTC/StPaul/Final/Project/service/DefaultTransactionsService.java - Spring Tool Suite 4

The screenshot shows the Spring Tool Suite 4 interface with the code editor open. The title bar indicates the project is 'springboot-workspace' and the file is 'DefaultTransactionsService.java'. The code editor displays Java code for a service class. The code uses annotations like @Service, @Autowired, and @Transactional. It includes methods for creating transactions, fetching customers, stores, movies, and concessions from a DAO. Error markers are present on lines 41 and 44.

```
1 package OTC.StPaul.Final.Project.service;
2
3 import java.util.List;
4
5 @Service
6 public class DefaultTransactionsService implements TransactionsService {
7
8     @Autowired
9     private TransactionsDao transactionsDao;
10
11     @Transactional
12     @Override
13     public transactions createTransaction(transactionRequest transactionRequest) {
14         customers customer = getCustomer(transactionRequest);
15         stores store = getStore(transactionRequest);
16         List<movies> movies = getMovie(transactionRequest);
17         List<concessions> concessions = getConcession(transactionRequest);
18
19         Double total = ((movies.size()) * 1.99);
20
21         for(concessions concession : concessions) {
22             total = total + concession.getPrice();
23         }
24
25         return transactionsDao.saveTransaction(customer, store, total, movies, concessions);
26     }
27
28     * @param transactionRequest
29     private customers getCustomer(transactionRequest transactionRequest) {
30         return transactionsDao.fetchCustomer(transactionRequest.getCustomer_idFK())
31             .orElseThrow(() -> new NoSuchElementException("Customer with ID="
32                 + transactionRequest.getCustomer_idFK() + " was not found"));
33     }
34
35     * @param transactionRequest
36     private stores getStore(transactionRequest transactionRequest) {
37         return transactionsDao.fetchStore(transactionRequest.getStore_idFK())
38             .orElseThrow(() -> new NoSuchElementException("Store with ID="
39                 + transactionRequest.getStore_idFK() + " was not found"));
40     }
41     * @param transactionRequest
42     private List<movies> getMovie(transactionRequest transactionRequest) {
43         return transactionsDao.fetchMovie(transactionRequest.getMovies());
44     }
45
46     private List<concessions> getConcession(transactionRequest transactionRequest) {
47
48         return transactionsDao.fetchConcession(transactionRequest.getConcessions());
49     }
50
51 } // last bracket
52 }
```

Test:

Controller-



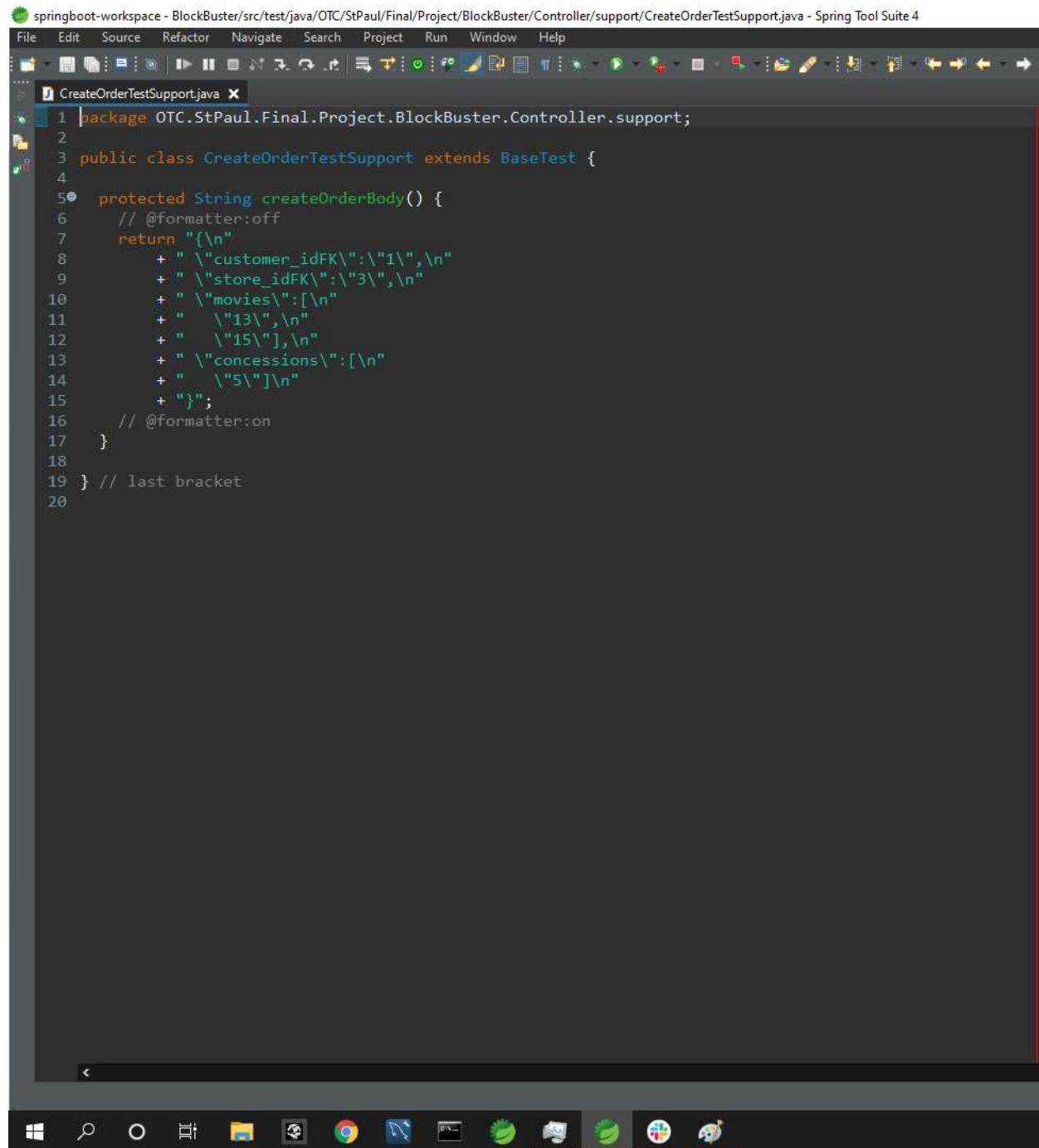
The screenshot shows the Spring Tool Suite 4 interface with the 'CreateOrderTest.java' file open in the central editor window. The code is a Spring Boot test class for a controller. It includes imports for various Spring and JUnit annotations and utilities, as well as specific classes from the project's package. The class itself extends 'CreateOrderTestSupport' and contains a single test method named 'testCreateOrderReturnsSuccess201'. This method sets up a JSON body, defines a URI, and uses a RestTemplate to send a POST request to the controller, then asserts that the response status is 201 and the 'transactions' table has the expected number of rows.

```
springboot-workspace - BlockBuster/src/test/java/OTC/StPaul/Final/Project/BlockBuster/Controller/CreateOrderTest.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
CreateOrderTest.java
1 package OTC.StPaul.Final.Project.BlockBuster.Controller;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4 import org.junit.jupiter.api.Test;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.boot.test.context.SpringBootTest;
7 import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
8 import org.springframework.http.HttpEntity;
9 import org.springframework.http.HttpHeaders;
10 import org.springframework.http.HttpMethod;
11 import org.springframework.http.HttpStatus;
12 import org.springframework.http.MediaType;
13 import org.springframework.http.ResponseEntity;
14 import org.springframework.jdbc.core.JdbcTemplate;
15 import org.springframework.test.context.ActiveProfiles;
16 import org.springframework.test.jdbc.JdbcTestUtils;
17 import OTC.StPaul.Final.Project.BlockBuster.Controller.support.CreateOrderTestSupport;
18 import OTC.StPaul.Final.Project.entity.transactions;
19
20 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
21 @ActiveProfiles("test")
22 //@Sql(scripts = {"classpath:OTC_StPaul_BlockBuster_Database.sql",
23 //      "classpath:OTC_StPaul_BlockBuster_DataEntry.sql"},
24 //      config = @SqlConfig(encoding = "utf-8"))
25 class CreateOrderTest extends CreateOrderTestSupport {
26
27     @Autowired
28     private JdbcTemplate jdbcTemplate;
29
30     @Test
31     void testCreateOrderReturnsSuccess201() {
32         // Given: an order as Json
33         String body = createOrderBody();
34         String uri = getBaseUriForTransactions();
35
36         int numRowsTransactions = JdbcTestUtils.countRowsInTable(jdbcTemplate, "transactions");
37
38         HttpHeaders headers = new HttpHeaders();
39         headers.setContentType(MediaType.APPLICATION_JSON);
40
41         HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
42
43         // When: the order is sent
44         ResponseEntity<transactions> response = getRestTemplate().exchange(uri, HttpMethod.POST,
45             bodyEntity, transactions.class);
```

```
46     // Then: a 201 Status is Returned
47     assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
48
49     // And: the return order is correct
50     assertThat(response.getBody()).isNotNull();
51
52     transactions transaction = response.getBody();
53     assertThat(transaction.getCustomer().getCustomer_idPK()).isEqualTo(1);
54     assertThat(transaction.getStore().getStore_idPK()).isEqualTo(3);
55     assertThat(transaction.getMovies_rented()).hasSize(2);
56     assertThat(transaction.getConcessions_purchased()).hasSize(1);
57
58     assertThat(JdbcTestUtils.countRowsInTable(jdbcTemplate, "transactions"))
59         .isEqualTo(numRowsTransactions + 1);
60
61 }
62
63 } // last bracket
64
65 <
```



Support-



The screenshot shows a Java code editor window in Spring Tool Suite 4. The title bar reads "springboot-workspace - BlockBuster/src/test/java/OTC/StPaul/Final/Project/BlockBuster/Controller/support/CreateOrderTestSupport.java - Spring Tool Suite 4". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Find, and Run. The code editor displays the following Java code:

```
1 package OTC.StPaul.Final.Project.BlockBuster.Controller.support;
2
3 public class CreateOrderTestSupport extends BaseTest {
4
5     protected String createOrderBody() {
6         // @formatter:off
7         return "{\n"
8             + " \"customer_idFK\": \"1\", \n"
9             + " \"store_idFK\": \"3\", \n"
10            + " \"movies\": [\n"
11                + "   \"13\", \n"
12                + "   \"15\" ], \n"
13                + " \"concessions\": [\n"
14                    + "   \"5\" ]\n"
15                + "}";
16        // @formatter:on
17    }
18
19 } // last bracket
20
```

The code defines a class named CreateOrderTestSupport that extends BaseTest. It contains a protected method createOrderBody() which returns a JSON string representing an order body. The JSON includes fields for customer_idFK (value 1), store_idFK (value 3), movies (array containing values 13 and 15), and concessions (array containing value 5). The code uses Javadoc-style annotations // @formatter:off and // @formatter:on to control code formatting.

The screenshot shows a Java IDE interface with a dark theme. The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations like Open, Save, and Print, as well as navigation and search functions. A status bar at the bottom displays system icons for network, battery, and other system status.

The main window displays the code for `BaseTest.java`:

```
1 package OTC.StPaul.Final.Project.BlockBuster.Controller.support;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 public class BaseTest {
6     @LocalServerPort
7     private int serverPort;
8
9     @Autowired
10    @Getter
11    private TestRestTemplate restTemplate;
12
13    /**
14     * @return
15     */
16    protected String getBaseUriForBlockbuster() {
17        return String.format("http://localhost:%d/otc_stpaul_blockbuster", serverPort);
18    }
19
20    /**
21     * @return
22     */
23    protected String getBaseUriForTransactions() {
24        return String.format("http://localhost:%d/transactions", serverPort);
25    }
26
27}
28} // last bracket
29
```

Swagger Running API:

Employees:

Starting Employees Table -



A screenshot of a database application showing a result grid titled "Result Grid". The grid displays data from a table with columns: employee_idPK, first_name, last_name, address, phone, and store_idPK. The data consists of 10 rows, indexed from 1 to 10. Row 10 is marked with an asterisk (*) and contains all NULL values. The application interface includes various toolbar icons for filtering, editing, and exporting.

	employee_idPK	first_name	last_name	address	phone	store_idPK
▶	1	Adam	Fite	123 Madison Street	417-555-8891	3
	2	Charlee	Thao	123 Madison Street	417-555-8891	2
	3	Andrew	Cham	123 Madison Street	417-555-8891	1
	4	Tony	Stark	2008 Malibu Rd	417-555-7910	3
	5	Frank	Castle	1989 Winchester Ln	417-555-5710	2
	6	Jean	Grey	2000 Marvel Rd	417-555-8891	1
	7	Jessica	Jone	2015 Investigation St	417-791-4845	3
	8	Ellie	Sattler	1995 Jurassic Parkway	417-555-9306	2
*	9	Wednesday	Addams	1938 New Yorker Place	417-555-1991	1
		NULL	NULL	NULL	NULL	NULL

Variables:

Creating Use: Abigail Westler 125 West Maple St 417-555-5478 store 3

Update Sample: Abby Franklin 8525 North South Rd 417-555-6369

Create Employee –

-Swagger-

The screenshot shows a web browser displaying the Swagger API documentation for creating an employee. The URL is `http://localhost:8080/swagger-ui.html#/?apis=%2Fapi%2Fv1%2Femployees&operationId=createEmployee`. The main area shows a form with the following fields:

- last_name**: Westler
- address**: 125 West Maple St
- phone**: 417-555-5478
- store_idFK**: 3

Below the form, the "Responses" section displays a successful response (201) with the following JSON content:

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Length: 160
Date: Mon, 26 Aug 2019 11:47:59 GMT
Etag: 1234567890

{
  "employee_idPK": 10,
  "first_name": "Abigail",
  "last_name": "Westler",
  "address": "125 West Maple St",
  "phone": "417-555-5478",
  "store_idFK": 3
}
```

-Database-

	employee_idPK	first_name	last_name	address	phone	store_idFK
▶	1	Adam	Fite	123 Madison Street	417-555-8891	3
	2	Charlee	Thao	123 Madison Street	417-555-8891	2
	3	Andrew	Cham	123 Madison Street	417-555-8891	1
	4	Tony	Stark	2008 Malibu Rd	417-555-7910	3
	5	Frank	Castle	1989 Winchester Ln	417-555-5710	2
	6	Jean	Grey	2000 Marvel Rd	417-555-8891	1
	7	Jessica	Jone	2015 Investigation St	417-791-4845	3
	8	Ellie	Sattler	1995 Jurassic Parkway	417-555-9306	2
	9	Wednesday	Addams	1938 New Yorker Place	417-555-1991	1
*	10	Abigail	Westler	125 West Maple St	417-555-5478	3
	NULL	NULL	NULL	NULL	NULL	NULL

Get Employee –

The screenshot shows a browser window with a REST API testing tool. The URL is `http://localhost:8080/EmployeeManagementSystem/rest/employees/10`. The method is set to `GET`. In the 'Parameters' section, there is a single parameter named `employee_idPK` with the value `10`. Below the parameters, there are tabs for 'Success' and 'Error'. Under the 'Responses' section, there is a 'Code' tab and a 'Details' tab. The 'Code' tab shows the response code `200`. The 'Details' tab shows the response body, which is a JSON object:

```
[{"id": 10, "name": "John Doe", "last_name": "Doe", "age": 30, "gender": "Male", "address": "123 Main Street", "phone": "+1234567890"}]
```

Update First Name –

-Swagger-

The screenshot shows the Swagger UI interface for an API endpoint. The URL is `/api_update_blackbox/updateemployees/{?first_name}`. The parameters section shows two fields: `first_name` set to `Abby` and `employee_idPK` set to `10`. Below the parameters, there are sections for `Request`, `Code` (HTTP status codes), and `Server response`. The server response shows a successful update with a status code of `200` and a response body indicating the update was successful.

-Database-

The screenshot shows a database result grid with the following columns: `employee_idPK`, `first_name`, `last_name`, `address`, `phone`, and `store_idPK`. The data is as follows:

	employee_idPK	first_name	last_name	address	phone	store_idPK
▶	1	Adam	Fite	123 Madison Street	417-555-8891	3
▶	2	Charlee	Thao	123 Madison Street	417-555-8891	2
▶	3	Andrew	Cham	123 Madison Street	417-555-8891	1
▶	4	Tony	Stark	2008 Malibu Rd	417-555-7910	3
▶	5	Frank	Castle	1989 Winchester Ln	417-555-5710	2
▶	6	Jean	Grey	2000 Marvel Rd	417-555-8891	1
▶	7	Jessica	Jone	2015 Investigation St	417-791-4845	3
▶	8	Ellie	Sattler	1995 Jurassic Parkway	417-555-9306	2
▶	9	Wednesday	Addams	1938 New Yorker Place	417-555-1991	1
▶	10	Abby	Westler	125 West Maple St	417-555-5478	3
*	NULL	NULL	NULL	NULL	NULL	NULL

Update Last Name –

-Swagger-

The screenshot shows the Swagger UI interface for a POST request to update an employee's last name. The URL is `/api/v1/employees/{last_name}`. The parameters section shows two fields: `last_name` (string, value: Franklin) and `employee_idPK` (integer, value: 10). The response section shows the raw JSON response from the server, which includes the updated employee information and a timestamp.

```
curl -X POST -H "Content-Type: application/json" -d "{\"last_name\": \"Franklin\", \"employee_idPK\": 10}" http://localhost:8080/api/v1/employees/{last_name} {{"last_name": "Franklin", "first_name": "Abby", "store_idPK": 3, "address": "125 West Maple St", "phone": "417-555-5478", "id": 10}, "date": "Mon, 28 Aug 2023 23:00:31 GMT", "status": 200, "title": "Employee Inserted"}
```

-Database-

The screenshot shows a database result grid with the following columns: employee_idPK, first_name, last_name, address, phone, and store_idPK. The data is as follows:

	employee_idPK	first_name	last_name	address	phone	store_idPK
▶	1	Adam	Fite	123 Madison Street	417-555-8891	3
▶	2	Charlee	Thao	123 Madison Street	417-555-8891	2
▶	3	Andrew	Cham	123 Madison Street	417-555-8891	1
▶	4	Tony	Stark	2008 Malibu Rd	417-555-7910	3
▶	5	Frank	Castle	1989 Winchester Ln	417-555-5710	2
▶	6	Jean	Grey	2000 Marvel Rd	417-555-8891	1
▶	7	Jessica	Jone	2015 Investigation St	417-791-4845	3
▶	8	Ellie	Sattler	1995 Jurassic Parkway	417-555-9306	2
▶	9	Wednesday	Addams	1938 New Yorker Place	417-555-1991	1
▶	10	Abby	Franklin	125 West Maple St	417-555-5478	3
*	NULL	NULL	NULL	NULL	NULL	NULL

Update Address –

-Swagger-

The screenshot shows the Swagger UI interface for a POST request to update an employee's address. The URL is `/v1/api/v1/employees/{address}`. The parameters section includes:

- name**: Description
- address**: `8525 North South Rd`
- store_idFK**: `10`

The response section shows a successful `200` status code with a JSON response body:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 111
Date: Mon, 19 Aug 2019 21:00:00 GMT
Server: Apache/2.4.34 (Ubuntu)
```

The response body is redacted.

-Database-

The screenshot shows a MySQL Workbench result grid displaying data from a table. The columns are `employee_idPK`, `first_name`, `last_name`, `address`, `phone`, and `store_idFK`. The data is as follows:

	employee_idPK	first_name	last_name	address	phone	store_idFK
▶	1	Adam	Fite	123 Madison Street	417-555-8891	3
	2	Charlee	Thao	123 Madison Street	417-555-8891	2
	3	Andrew	Cham	123 Madison Street	417-555-8891	1
	4	Tony	Stark	2008 Malibu Rd	417-555-7910	3
	5	Frank	Castle	1989 Winchester Ln	417-555-5710	2
	6	Jean	Grey	2000 Marvel Rd	417-555-8891	1
	7	Jessica	Jone	2015 Investigation St	417-791-4845	3
	8	Ellie	Sattler	1995 Jurassic Parkway	417-555-9306	2
	9	Wednesday	Addams	1938 New Yorker Place	417-555-1991	1
	10	Abby	Franklin	8525 North South Rd	417-555-5478	3
*	NULL	NULL	NULL	NULL	NULL	NULL

Update Phone Number –

-Swagger-

The screenshot shows the Swagger UI interface for a 'default-employees-controller'. A POST request is being made to the endpoint `/api/v1/default/employees/{id}/updateEmployee/{phone}`. The 'Parameters' section includes:

- `id`: 10440000
- `phone`: 417-555-0569
- `employee_idPK`: 10

The 'Responses' section shows a successful response (200) with the following details:

- Code:** 200
- Request URL:** `http://localhost:8080/api/v1/default/employees/10/updateEmployee/417-555-0569?employee_idPK=10`
- Server Response:**
 - Code:** 200
 - Response Headers:**
 - Content-Type: application/json
 - Date: Mon, 20 Nov 2017 11:55:48 GMT
 - Server: Apache/2.4.10 (Ubuntu)
 - ResponseBody:**

```
{"id": 10, "first_name": "Abby", "last_name": "Franklin", "address": "8525 North South Rd", "phone": "417-555-6369", "store_idFK": 3}
```

-Database-

The screenshot shows the 'employees' table in MySQL Workbench. The table has the following columns: employee_idPK, first_name, last_name, address, phone, and store_idFK. There are 10 rows of data:

	employee_idPK	first_name	last_name	address	phone	store_idFK
▶	1	Adam	Fite	123 Madison Street	417-555-8891	3
	2	Charlee	Thao	123 Madison Street	417-555-8891	2
	3	Andrew	Cham	123 Madison Street	417-555-8891	1
	4	Tony	Stark	2008 Malibu Rd	417-555-7910	3
	5	Frank	Castle	1989 Winchester Ln	417-555-5710	2
	6	Jean	Grey	2000 Marvel Rd	417-555-8891	1
	7	Jessica	Jone	2015 Investigation St	417-791-4845	3
	8	Ellie	Sattler	1995 Jurassic Parkway	417-555-9306	2
	9	Wednesday	Addams	1938 New Yorker Place	417-555-1991	1
*	10	Abby	Franklin	8525 North South Rd	417-555-6369	3
	NULL	NULL	NULL	NULL	NULL	NULL

Delete Employee –

-Swagger-

The screenshot shows the Swagger UI interface for a DELETE endpoint. The URL is `/api/v1/employees/{employee_idPK}`. The method is set to `DELETE`. The `Parameters` section contains a single parameter named `employee_idPK` with a value of `40`. Below the parameters is a large blue button labeled `Execute`. To the right of the execute button is a `Cancel` button. Under the `Responses` section, there are three entries: a `Code` entry with a dropdown menu containing `200`, `400`, and `500`; a `Response Headers` entry showing a JSON object with keys `Content-Type`, `Content-Length`, and `Date`; and a `Response Body` entry with a dropdown menu containing `OK`.

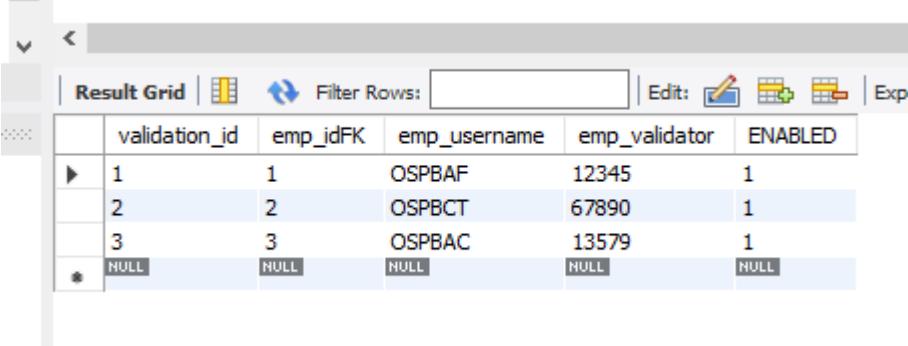
-Database-

The screenshot shows the MySQL Workbench interface with a result grid titled "Result Grid". The table structure is as follows:

	employee_idPK	first_name	last_name	address	phone	store_idFK
1	Adam	Fite	123 Madison Street	417-555-8891	3	
2	Charlee	Thao	123 Madison Street	417-555-8891	2	
3	Andrew	Cham	123 Madison Street	417-555-8891	1	
4	Tony	Stark	2008 Malibu Rd	417-555-7910	3	
5	Frank	Castle	1989 Winchester Ln	417-555-5710	2	
6	Jean	Grey	2000 Marvel Rd	417-555-8891	1	
7	Jessica	Jone	2015 Investigation St	417-791-4845	3	
8	Ellie	Sattler	1995 Jurassic Parkway	417-555-9306	2	
9	Wednesday	Addams	1938 New Yorker Place	417-555-1991	1	
*	NULL	NULL	NULL	NULL	NULL	NULL

User Validation:

Starting User Validation –



A screenshot of a database result grid window. The grid displays a table with six columns: validation_id, emp_idFK, emp_username, emp_validator, and ENABLED. There are four rows of data, with the fifth row being a header row indicated by an asterisk (*). The data is as follows:

	validation_id	emp_idFK	emp_username	emp_validator	ENABLED
▶	1	1	OSPBAF	12345	1
	2	2	OSPBCT	67890	1
	3	3	OSPBAC	13579	1
*	NULL	NULL	NULL	NULL	NULL

Variables: emp# 10 username OSPBAW password 65824

Update Variable: password

Create Validation –

-Swagger-

The screenshot shows the Swagger UI interface for a REST API. In the 'Parameters' section, three fields are defined:

- emp_idFK**: Integer (Optional) with value 10.
- emp_username**: String (Optional) with value OSPBAF.
- emp_validator**: Integer (Optional) with value 65824.

The 'Responses' section shows a successful response (201 Created) with the following JSON content:

```
HTTP/1.1 201 Created
Content-Type: application/json
Date: Fri, 09 Aug 2019 23:06:14 GMT
Server: Apache/2.4.34 (Ubuntu)
Set-Cookie: JSESSIONID=44D9A8E8A88A8A8A8A8A8A8A8A8A8A8A; Path=/; HttpOnly

{
    "validation_id": 1,
    "emp_idFK": 10,
    "emp_username": "OSPBAW",
    "emp_validator": 65824,
    "ENABLED": 1
}
```

-Database-

The screenshot shows the MySQL Workbench Result Grid displaying data from a table. The table has columns: validation_id, emp_idFK, emp_username, emp_validator, and ENABLED. The data is as follows:

	validation_id	emp_idFK	emp_username	emp_validator	ENABLED
▶	1	1	OSPBAF	12345	1
▶	2	2	OSPBCT	67890	1
▶	3	3	OSPBAC	13579	1
▶	4	10	OSPBAW	65824	1
*		NULL	NULL	NULL	NULL

Get Username –

The screenshot shows a browser window displaying a REST API endpoint for user validation. The URL is `/api/_staging/blockbuster/user_validation/{emp_id#}`. The page is divided into sections: Parameters, Responses, and Headers.

Parameters:

Name	Description
emp_id#	10

Responses:

Code:

```
curl -X 'GET' 'http://localhost:8080/api/_staging/blockbuster/user_validation/10'
```

Request URL:

```
http://localhost:8080/api/_staging/blockbuster/user_validation
```

Header response:

Code: 200
Response body:

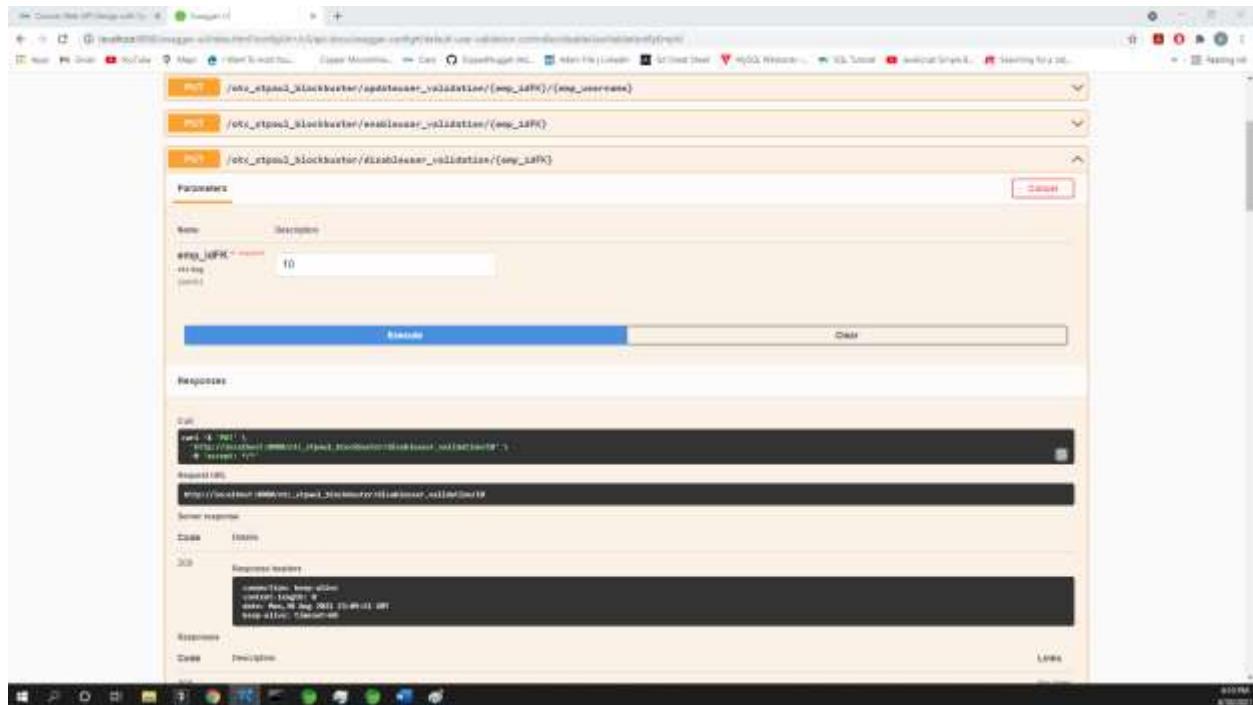
```
[{"id": "validation_id", "emp_id": "10", "username": "admin", "password": "123456"}]
```

Response headers:

```
Content-Type: application/json
Date: Wed, 26 Aug 2021 21:45:19 GMT
Server: Gunicorn/20.0.4
Transfer-Encoding: chunked
```

Disable Validation-

-Swagger-



PUT /otc_otpms3_blockbuster/disableuser_validation/{emp_idPK}

Parameters

Name	Description
emp_idPK	emp_idPK

Responses

200

```
{  "validation_id": 1,  "emp_idPK": 10,  "emp_username": "OSPBWF",  "emp_validator": 12345,  "ENABLED": 1}
```

Request URL: http://localhost:8080/v1/otc_otpms3_blockbuster/disableuser_validation/10

Server response

Code Status

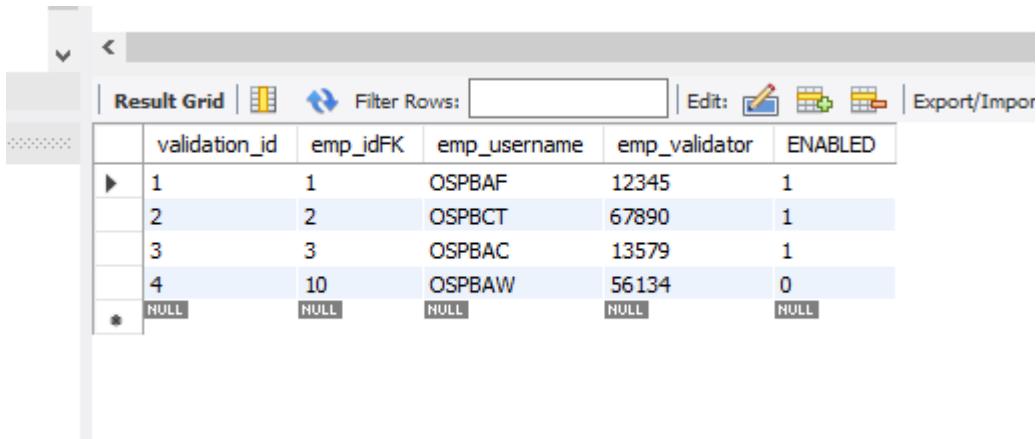
300 Response history

```
Content-type: application/json; charset=UTF-8
Date: Mon, 20 Aug 2023 23:49:11 GMT
Server: GlassFish 4.1
```

Responses

Code Description

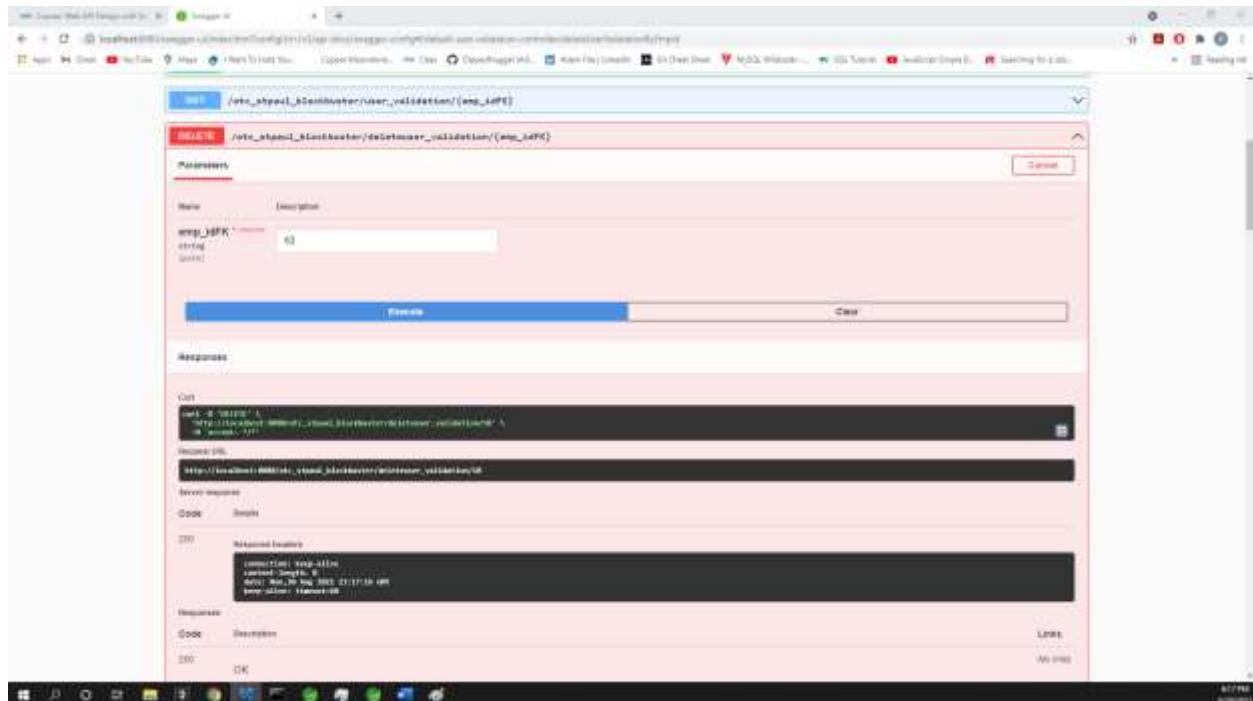
-Database-



	validation_id	emp_idPK	emp_username	emp_validator	ENABLED
▶	1	1	OSPBWF	12345	1
	2	2	OSPBCT	67890	1
	3	3	OSPBAC	13579	1
*	4	10	OSPBAW	56134	0
*	NULL	NULL	NULL	NULL	NULL

Delete Validation –

-Swagger-

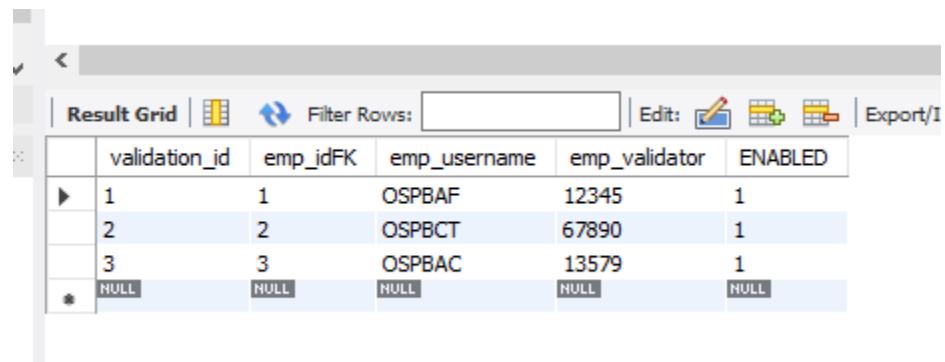


The screenshot shows the Swagger UI interface for a DELETE operation. The URL is `/v1/_schemas/_kafka/user_validation/{emp_idFK}`. The method is `DELETE`. A parameter `emp_idFK` is set to `12`. The response body is a JSON object:

```
{  "validation_id": 1,  "emp_idFK": 1,  "emp_username": "OSPBAF",  "emp_validator": "12345",  "ENABLED": 1}
```

The status code is `200 OK`.

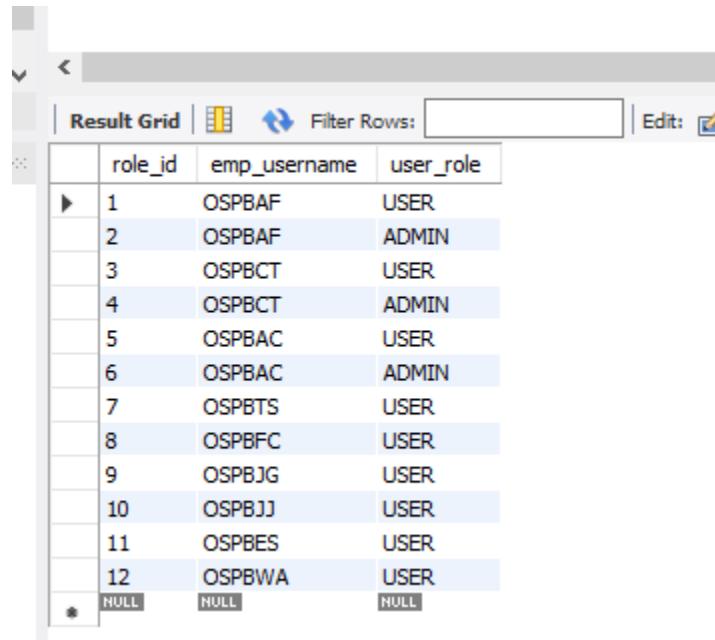
-Database-



	validation_id	emp_idFK	emp_username	emp_validator	ENABLED
▶	1	1	OSPBAF	12345	1
▶	2	2	OSPBCT	67890	1
▶	3	3	OSPBAC	13579	1
*	NULL	NULL	NULL	NULL	NULL

User Roles:

Starting Database –



A screenshot of a database management tool showing a result grid. The grid has three columns: role_id, emp_username, and user_role. The data is as follows:

	role_id	emp_username	user_role
▶	1	OSPBAF	USER
	2	OSPBAF	ADMIN
	3	OSPBCT	USER
	4	OSPBCT	ADMIN
	5	OSPBAC	USER
	6	OSPBAC	ADMIN
	7	OSPBTS	USER
	8	OSPBFC	USER
	9	OSPBJG	USER
	10	OSPBJJ	USER
	11	OSPBES	USER
	12	OSPBWA	USER
*	NULL	NULL	NULL

Giving a role –

-Swagger-

The screenshot shows the Swagger UI interface for a REST API. The URL is `/api/v1/user-roles-controller`. The method is `POST` to `/api/v1/user-roles-controller/{emp_username}`. The parameters are `emp_username` (value: OSPBAF) and `user_role` (value: USER). The response code is `201` with the message "Role created". The response body is a JSON object:

```
{  "role_id": 1,  "emp_username": "OSPBAF",  "user_role": "USER"}  
Content-Type: application/json  
Date: Mon, 26 Aug 2013 21:15:21 GMT  
Server: GlassFish 3.1.2.2
```

-Database-

	role_id	emp_username	user_role
▶	1	OSPBAF	USER
	2	OSPBAF	ADMIN
	3	OSPBCT	USER
	4	OSPBCT	ADMIN
	5	OSPBAC	USER
	6	OSPBAC	ADMIN
	7	OSPBTS	USER
	8	OSPBFC	USER
	9	OSPBFG	USER
	10	OSPBJJ	USER
	11	OSPBES	USER
	12	OSPBWA	USER
*	13	OSPBAW	USER
	NULL	NULL	NULL

Getting all employees of a certain role –

The screenshot shows a browser window with the URL <http://localhost:8080/api/v1/users/roles>. The page displays an API response for a GET request. The response code is 200 OK, and the response body contains a JSON array of employee objects, each with properties like 'user_id', 'role', 'username', and 'password'. The response headers include 'connection: keep-alive', 'content-type: application/json', and 'date: Mon, 20 Aug 2012 21:21:29 GMT'.

```
curl -X GET http://localhost:8080/api/v1/users/roles
{
    "user_id": 1,
    "role": "ROLE_ADMIN",
    "username": "OSPBAW",
    "password": "123456"
}
{
    "user_id": 2,
    "role": "ROLE_USER",
    "username": "OSPBAW",
    "password": "123456"
}
{
    "user_id": 3,
    "role": "ROLE_USER",
    "username": "OSPBAW",
    "password": "123456"
}
{
    "user_id": 4,
    "role": "ROLE_USER",
    "username": "OSPBAW",
    "password": "123456"
}
```

Response Headers:

```
connection: keep-alive
content-type: application/json
date: Mon, 20 Aug 2012 21:21:29 GMT
```

Getting an employees roles –

The screenshot shows a browser window with the URL <http://localhost:8080/api/v1/users/roles/1>. The page displays an API response for a GET request. The response code is 200 OK, and the response body contains a JSON object with properties 'user_id', 'role', 'username', and 'password'. The response headers include 'connection: keep-alive', 'content-type: application/json', and 'date: Mon, 20 Aug 2012 21:21:29 GMT'.

```
curl -X GET http://localhost:8080/api/v1/users/roles/1
{
    "user_id": 1,
    "role": "ROLE_ADMIN",
    "username": "OSPBAW",
    "password": "123456"
}
```

Response Headers:

```
connection: keep-alive
content-type: application/json
date: Mon, 20 Aug 2012 21:21:29 GMT
keep-alive: timeout=300
Transfer-Encoding: chunked
```

Deleting a role –

-Swagger-

The screenshot shows the Swagger UI interface for a REST API. The URL is `/v1/vipaul_blackberry/user_roles/{role_id}`. The method is set to `DELETE`. In the 'Parameters' section, there is a single parameter named `role_id` with a value of `15`. Below the parameters, there is a large red box containing the response body. The response body shows a JSON object with a single key `status` and a value of `"Success"`. The status code is `200 OK`. At the bottom of the red box, it says "Response Headers" and lists `Content-Type: application/json`, `Date: Mon, 29 Aug 2011 21:00:01 GMT`, and `Expires: Tue, 30 Aug 2011 21:00:01 GMT`.

-Database-

The screenshot shows the MySQL Workbench Result Grid displaying the contents of the `user_roles` table. The table has three columns: `role_id`, `emp_username`, and `user_role`. The data is as follows:

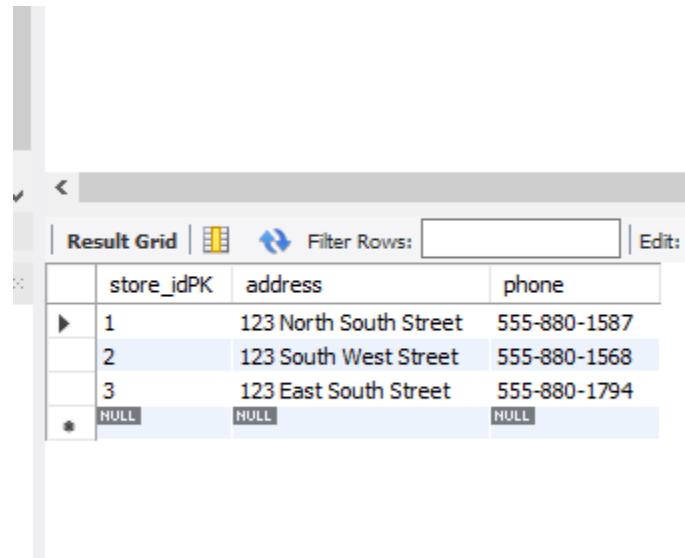
	role_id	emp_username	user_role
▶	1	OSPBAF	USER
▶	2	OSPBAF	ADMIN
▶	3	OSPBCT	USER
▶	4	OSPBCT	ADMIN
▶	5	OSPBAC	USER
▶	6	OSPBAC	ADMIN
▶	7	OSPBTS	USER
▶	8	OSPBFC	USER
▶	9	OSPBJG	USER
▶	10	OSPBJJ	USER
▶	11	OSPBES	USER
▶	12	OSPBWA	USER
*	NULL	NULL	NULL

Stores:

Variables : 698 West East St 417-999-3646

Update variables : 705 W East Street 417-999-4578

Starting Database –



A screenshot of a database result grid titled "Result Grid". The grid has four columns: "store_idPK", "address", and "phone". There are three visible rows with data and one row at the bottom labeled with an asterisk (*). The first row contains values 1, 123 North South Street, and 555-880-1587 respectively. The second row contains values 2, 123 South West Street, and 555-880-1568. The third row contains values 3, 123 East South Street, and 555-880-1794. The bottom row is entirely filled with "NULL" values.

	store_idPK	address	phone
▶	1	123 North South Street	555-880-1587
	2	123 South West Street	555-880-1568
*	3	123 East South Street	555-880-1794
	NULL	NULL	NULL

Creating a Store-

-Swagger-

The screenshot shows the Swagger UI interface for a POST request to the endpoint `/store`. The "Parameters" section contains two fields: `address` (string, value: "698-West East St") and `phone` (string, value: "417-999-3646"). The "Responses" section displays the raw JSON response and a detailed view of the response body, which includes the message "Success" and the store ID "4".

-Database-

	store_idPK	address	phone
▶	1	123 North South Street	555-880-1587
▶	2	123 South West Street	555-880-1568
▶	3	123 East South Street	555-880-1794
▶	4	698 West East St	417-999-3646
*	NUL	NUL	NUL

Getting a store –

The screenshot shows a browser window displaying a REST API endpoint for retrieving a store by its ID. The URL is `/api/_internal/_blockbuster/stores/{store_id#PK}`. The page includes sections for 'Parameters' and 'Responses'.

Parameters

Name	Description
store_id#PK	integer (min:0, max:1000)

Responses

Code: 200 OK

```
HTTP/1.1 200 OK
Date: Wed, 20 Aug 2014 20:11:52 GMT
Content-Type: application/json
Content-Length: 140

{
  "store_id": 1,
  "address": "7225 Park South Street",
  "name": "Star-Wars"
}
```

Headers received:

```
connection: keep-alive
content-type: application/json
date: Wed, 20 Aug 2014 20:11:52 GMT
keep-alive: timeout=5
transfer-encoding: chunked
```

Update a store address-

-Swagger-

The screenshot shows the Swagger UI interface for a REST API. The URL is `/v2/_openapi/blockbuster/api/stores/{address}`. The method is `POST`. The parameters section shows two fields: `address` with value `705 W East Street` and `store_idPK` with value `4`. The responses section shows a successful response (200) with a status message and headers including `Content-Type: application/json`, `Date: Tue, 12 Jul 2022 22:49:31 GMT`, and `Server: Apache/2.4.41 (Ubuntu)`.

-Database-

The screenshot shows a MySQL Workbench result grid. The columns are `store_idPK`, `address`, and `phone`. The data is as follows:

	store_idPK	address	phone
▶	1	123 North South Street	555-880-1587
▶	2	123 South West Street	555-880-1568
▶	3	123 East South Street	555-880-1794
▶	4	705 W East Street	417-999-3646
*	NULL	NULL	NULL

Update a store phone number-

-Swagger-

The screenshot shows the Swagger UI interface for a REST API. The URL is `/api/_api/v1_stores/updatePhone/{id}`. The method is `PUT`. The parameters section shows:

- `name`: `descption`
- `phone`: `417-999-4578`
- `store_idPK`: `4`

The responses section shows a successful response (200) with the following JSON content:

```
curl -X PUT "http://localhost:8000/api/_api/v1_stores/updatePhone?store_idPK=4&phone=417-999-4578" -H "Content-Type: application/json"
```

The browser taskbar at the bottom shows various open tabs and icons.

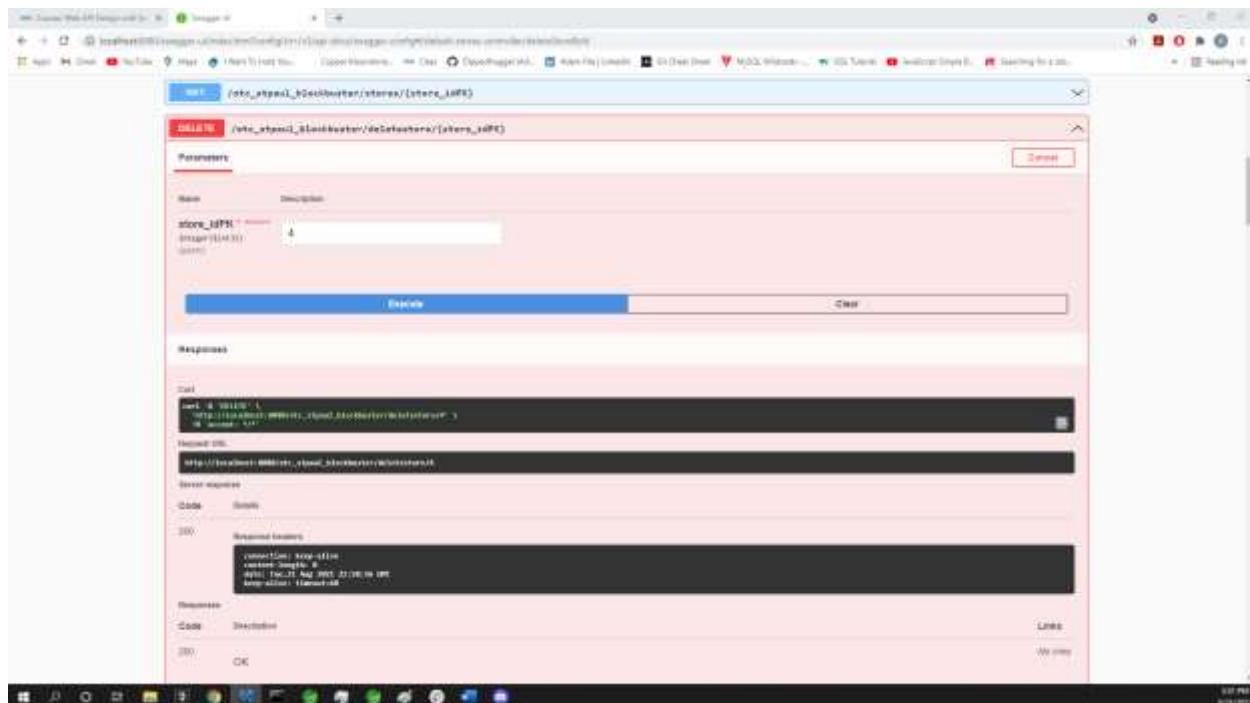
-Database-

A screenshot of a database result grid titled "Result Grid". The grid has columns: `store_idPK`, `address`, and `phone`. The data is as follows:

	store_idPK	address	phone
▶	1	123 North South Street	555-880-1587
	2	123 South West Street	555-880-1568
	3	123 East South Street	555-880-1794
▶	4	705 W East Street	417-999-4578
*	NULL	NULL	NULL

Delete a Store-

-Swagger-



The screenshot shows the Swagger UI interface for a DELETE operation. The URL is `/v1/stores/{store_idPK}`. A parameter `store_idPK` is set to `4`. The method is `DELETE`. Below the parameters, there is a `Responses` section. For the `200` status code, the response body is shown as a JSON object:

```
curl -X DELETE "http://localhost:8080/v1/stores/4"
```

Below the responses, there is a `Responses` section with a single entry for `200` with the value `OK`.

-Database-

Result Grid Filter Rows: <input type="text"/> Edit:			
	store_idPK	address	phone
▶	1	123 North South Street	555-880-1587
	2	123 South West Street	555-880-1568
	3	123 East South Street	555-880-1794
*	NULL	NULL	NULL

Customers:

Variables: William Clinton 42 Arkansas Way 555-968-7945

Update variables: Bill Clamped 42 Brooklyn Way 555-753-2542

Starting Database –

A screenshot of a database management system interface showing a result grid. The grid has columns for customer_idPK, first_name, last_name, address, phone, and join_date. The data shows six rows of historical customer information. A new row is being added at the bottom with placeholder values: * in customer_idPK, NULL in first_name, NULL in last_name, NULL in address, NULL in phone, and NULL in join_date.

	customer_idPK	first_name	last_name	address	phone	join_date
▶	1	George	Washington	01 Virigina Street	555-234-6897	2021-08-29 10:50:26
	2	Martha	Washington	01 Virigina Street	555-234-6897	2021-08-29 10:50:26
	3	John	Adams	02 Massachusetts Street	555-845-6487	2021-08-29 10:50:26
	4	Abigail	Adams	02 Massachusetts Street	555-845-6487	2021-08-29 10:50:26
	5	Thomas	Jefferson	03 Virigina Street	555-234-1478	2021-08-29 10:50:26
*	6	Martha Wayles Skelton	Jefferson	03 Virigina Street	555-234-1478	2021-08-29 10:50:26
	NULL	NULL	NULL	NULL	NULL	NULL

Create Customer-

-Swagger-

first_name: Wilson
last_name: Clinton
address: 42 Arkansas Way
phone: 555-968-7945

Raw

```
[{"customer_id": 8, "first_name": "Wilson", "last_name": "Clinton", "address": "42 Arkansas Way", "phone": "555-968-7945", "join_date": "2021-08-31T17:56:32"}]
```

-Database-

customer_idPK	first_name	last_name	address	phone	join_date
1	George	Washington	01 Virigina Street	555-234-6897	2021-08-29 10:50:26
2	Martha	Washington	01 Virigina Street	555-234-6897	2021-08-29 10:50:26
3	John	Adams	02 Massachusetts Street	555-845-6487	2021-08-29 10:50:26
4	Abigail	Adams	02 Massachusetts Street	555-845-6487	2021-08-29 10:50:26
5	Thomas	Jefferson	03 Virigina Street	555-234-1478	2021-08-29 10:50:26
6	Martha Wayles Skelton	Jefferson	03 Virigina Street	555-234-1478	2021-08-29 10:50:26
7	William	Clinton	42 Arkansas Way	555-968-7945	2021-08-31 17:56:32
*	NULL	NULL	NULL	NULL	NULL

Get Customer- -Swagger-

The screenshot shows a web browser window displaying the Swagger UI for a REST API. The URL in the address bar is `/api/v1/swagger-ui.html?apiUrl=http://localhost:8080/api/v1/customer/getCustomerById/{customer_id}`. The main content area is titled "Get Customer- -Swagger-".

Parameters:

Name	Description
<code>customer_id</code>	integer (4d:32) 7

Responses:

Call:
`http://localhost:8080/api/v1/customer/getCustomerById/{customer_id}; {customer_id: 7}`

Request Body:
`HTTP/2.0 200 OK, content-type:application/json; charset=utf-8`

Sample response:

Code: `Identity`

200 Response Body:

```
[{"customer_id": 7, "first_name": "William", "last_name": "Johnson", "address": "123 Elm Street", "city": "Austin", "state": "Texas", "zip": "78701"}]
```

Download

Response Headers:

```
connection: keep-alive
content-type: application/json
date: Mon, 10 Jul 2017 11:14:00 GMT
Content-Type: Identity
Content-Length: 163
transfer-encoding: chunked
```

Download

Update First Name-

-Swagger-

The screenshot shows the Swagger UI interface for a REST API. The URL is `/api_updateCustomer`. The method is `POST`. The path parameter is `{first_name}`. The parameters section shows two fields: `first_name` (string, required) and `customer_id` (integer (int32), required). Below the parameters is a large blue button labeled "Execute". Under the "Responses" section, there is a "Code" tab showing the response body:

```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Fri, 21 Aug 2020 22:00:44 GMT
Server: Amazon.com

{
    "customer_id": 1,
    "first_name": "George",
    "last_name": "Washington",
    "address": "01 Virigina Street",
    "phone": "555-234-6897",
    "join_date": "2021-08-29 10:50:26"
}
```

-Database-

The screenshot shows a database grid with the following columns: customer_idPK, first_name, last_name, address, phone, and join_date. The data is as follows:

	customer_idPK	first_name	last_name	address	phone	join_date
▶	1	George	Washington	01 Virigina Street	555-234-6897	2021-08-29 10:50:26
	2	Martha	Washington	01 Virigina Street	555-234-6897	2021-08-29 10:50:26
	3	John	Adams	02 Massachusetts Street	555-845-6487	2021-08-29 10:50:26
	4	Abigail	Adams	02 Massachusetts Street	555-845-6487	2021-08-29 10:50:26
	5	Thomas	Jefferson	03 Virigina Street	555-234-1478	2021-08-29 10:50:26
	6	Martha Wayles Skelton	Jefferson	03 Virigina Street	555-234-1478	2021-08-29 10:50:26
*	7	Bill	Clinton	42 Arkansas Way	555-968-7945	2021-08-31 17:56:32
*	NULL	NULL	NULL	NULL	NULL	NULL

Update Last Name –

-Swagger-

The screenshot shows the Swagger UI interface for a POST request to update a customer's last name. The URL is `/api/v1/customer/{customer_id}/last_name`. The parameters section shows `last_name` set to "Clamped" and `customer_id` set to 1. The response section shows the raw JSON response: `{ "customer_id": 1, "first_name": "George", "last_name": "Clamped", "address": "01 Virgina Street", "phone": "555-234-6897", "join_date": "2021-08-29T10:50:26Z" }`.

-Database-

	customer_idPK	first_name	last_name	address	phone	join_date
▶	1	George	Washington	01 Virgina Street	555-234-6897	2021-08-29 10:50:26
2	Martha	Washington	01 Virgina Street	555-234-6897	555-234-6897	2021-08-29 10:50:26
3	John	Adams	02 Massachusetts Street	555-845-6487	555-845-6487	2021-08-29 10:50:26
4	Abigail	Adams	02 Massachusetts Street	555-845-6487	555-845-6487	2021-08-29 10:50:26
5	Thomas	Jefferson	03 Virgina Street	555-234-1478	555-234-1478	2021-08-29 10:50:26
6	Martha Wayles Skelton	Jefferson	03 Virgina Street	555-234-1478	555-234-1478	2021-08-29 10:50:26
7	Bill	Clamped	42 Arkansas Way	555-968-7945	555-968-7945	2021-08-31 17:56:32
*	NUL	NUL	NUL	NUL	NUL	NUL

Update Address –

-Swagger-

The screenshot shows the Swagger UI interface for a REST API. The URL is `/api/v1/customer/updatecustomer/{first_name}`. The method is `PUT`. The path parameters are `first_name` (value: `Bill`) and `customer_id` (value: `7`). The body parameters are `address` (value: `42 Brooklyn Way`). The responses section shows a successful response with status code `200` and a JSON payload containing the updated customer information.

-Database-

	customer_idPK	first_name	last_name	address	phone	join_date
▶	1	George	Washington	01 Virigina Street	555-234-6897	2021-08-29 10:50:26
▶	2	Martha	Washington	01 Virigina Street	555-234-6897	2021-08-29 10:50:26
▶	3	John	Adams	02 Massachusetts Street	555-845-6487	2021-08-29 10:50:26
▶	4	Abigail	Adams	02 Massachusetts Street	555-845-6487	2021-08-29 10:50:26
▶	5	Thomas	Jefferson	03 Virigina Street	555-234-1478	2021-08-29 10:50:26
▶	6	Martha Wayles Skelton	Jefferson	03 Virigina Street	555-234-1478	2021-08-29 10:50:26
▶	7	Bill	Clamped	42 Brooklyn Way	555-968-7945	2021-08-31 17:56:32
*	NULL	NULL	NULL	NULL	NULL	NULL

Update Phone Number-

-Swagger-

The screenshot shows the Swagger UI interface for a 'default-customers-controller'. The endpoint being viewed is `PUT /api/_asynchronous/blockbuster/updatecustomer/{phone}`. The 'Parameters' section shows two required parameters: `phone` (string, value: 555-753-2542) and `customer_id` (integer (64-bit), value: 1). Below the parameters is a large blue 'Execute' button. The 'Responses' section shows a successful response (200 OK) with the following JSON content:

```
curl -X PUT -d "customer_id=1&phone=555-753-2542" http://localhost:8080/api/_asynchronous/blockbuster/updatecustomer/555-753-2542/customer_id=1
```

Below the responses, there is a 'Server response' section showing the raw HTTP response:

```
HTTP/1.1 200 OK
Date: Fri, 21 Aug 2021 21:00:01 GMT
Content-Type: application/json
Content-Length: 146
Connection: keep-alive
Last-Modified: Fri, 21 Aug 2021, 21:00:01 GMT
ETag: "1629448001000"

{"customer_id": 1, "first_name": "George", "last_name": "Washington", "address": "01 Virginia Street", "phone": "555-234-6897", "join_date": "2021-08-29T10:00:01Z"}  
Process finished with exit code 0
```

-Database-

The screenshot shows a 'Result Grid' from MySQL Workbench displaying data from a table named 'customer'. The table has columns: customer_id (PK), first_name, last_name, address, phone, and join_date. The data is as follows:

	customer_id	first_name	last_name	address	phone	join_date
▶	1	George	Washington	01 Virginia Street	555-234-6897	2021-08-29 10:00:01
	2	Martha	Washington	01 Virginia Street	555-234-6897	2021-08-29 10:00:01
	3	John	Adams	02 Massachusetts Street	555-845-6487	2021-08-29 10:00:01
	4	Abigail	Adams	02 Massachusetts Street	555-845-6487	2021-08-29 10:00:01
	5	Thomas	Jefferson	03 Virginia Street	555-234-1478	2021-08-29 10:00:01
	6	Martha Wayles Skelton	Jefferson	03 Virginia Street	555-234-1478	2021-08-29 10:00:01
*	7	Bill	Clamped	42 Brooklyn Way	555-753-2542	2021-08-31 17:00:01
*	NONE	NULL	NULL	NULL	NULL	NULL

Delete a Customer-

-Swagger-

The screenshot shows the Swagger UI interface for a REST API. The URL is `/eric_stripe_6/Customers/20DeleteCustomer/{customer_id}`. The 'Parameters' section contains a single parameter: `customer_id` (Type: String, Value: 7). The 'Responses' section shows two possible outcomes: a successful response (Status: 200, Content-Type: application/json, Body: `{"customer_id": "7", "first_name": "George", "last_name": "Washington", "address": "01 Virigina Street", "phone": "555-234-6897", "join_date": "2021-08-29 10:50:26"}`) and a default response (Status: 500, Content-Type: application/json, Body: `{"error": "Internal Server Error"}`). The bottom status bar indicates the browser is running on Windows 10.

-Database-

The screenshot shows the MySQL Workbench Result Grid displaying data from a table. The columns are `customer_idPK`, `first_name`, `last_name`, `address`, `phone`, and `join_date`. The data consists of six rows:

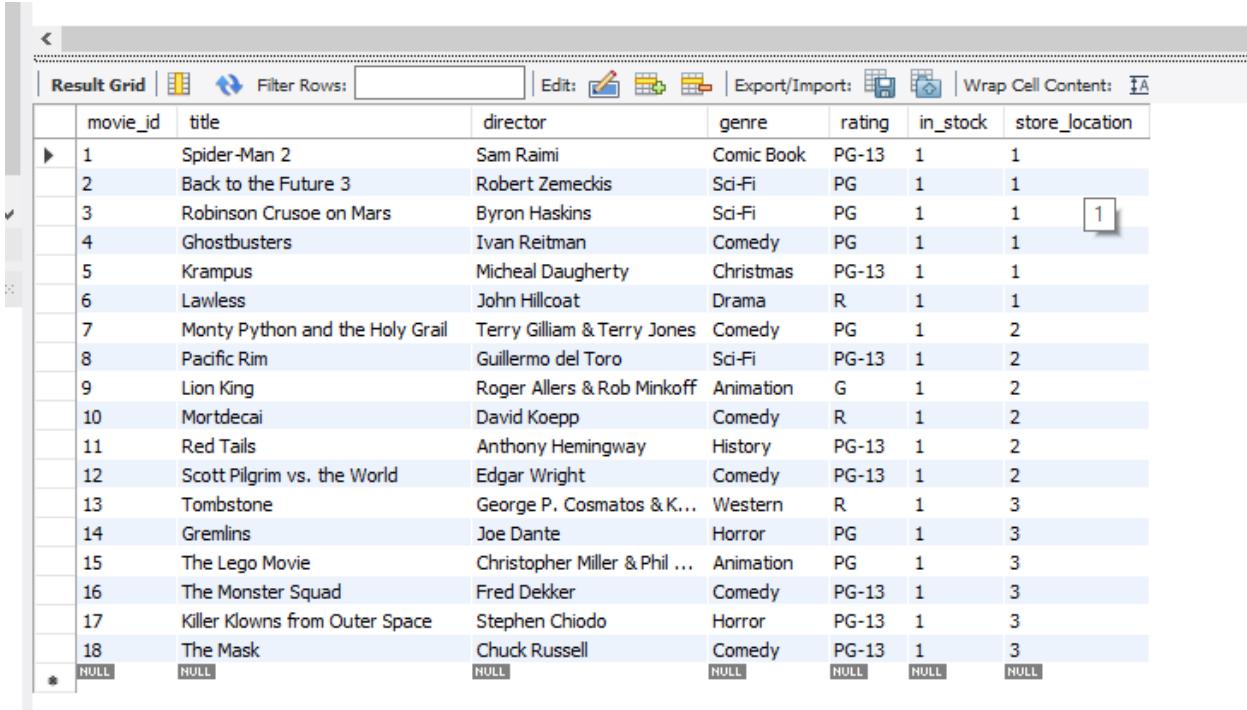
	customer_idPK	first_name	last_name	address	phone	join_date
▶	1	George	Washington	01 Virigina Street	555-234-6897	2021-08-29 10:50:26
▶	2	Martha	Washington	01 Virigina Street	555-234-6897	2021-08-29 10:50:26
▶	3	John	Adams	02 Massachusetts Street	555-845-6487	2021-08-29 10:50:26
▶	4	Abigail	Adams	02 Massachusetts Street	555-845-6487	2021-08-29 10:50:26
▶	5	Thomas	Jefferson	03 Virigina Street	555-234-1478	2021-08-29 10:50:26
*	6	Martha Wayles Skelton	Jefferson	03 Virigina Street	555-234-1478	2021-08-29 10:50:26
*	HULL	HULL	HULL	HULL	HULL	HULL

Movies:

Variable: Suicide Squad David Ayer Comic Book PG-13 03

Update variables: The Suicide Squad James Gunn Action/Adventure R 01

Starting Database –



A screenshot of a database grid titled "Result Grid". The grid displays 18 rows of movie information across seven columns: movie_id, title, director, genre, rating, in_stock, and store_location. The "store_location" column for row 18 contains the value "1", which is highlighted with a red box.

movie_id	title	director	genre	rating	in_stock	store_location
1	Spider-Man 2	Sam Raimi	Comic Book	PG-13	1	1
2	Back to the Future 3	Robert Zemeckis	Sci-Fi	PG	1	1
3	Robinson Crusoe on Mars	Byron Haskins	Sci-Fi	PG	1	1
4	Ghostbusters	Ivan Reitman	Comedy	PG	1	1
5	Krampus	Micheal Daugherty	Christmas	PG-13	1	1
6	Lawless	John Hillcoat	Drama	R	1	1
7	Monty Python and the Holy Grail	Terry Gilliam & Terry Jones	Comedy	PG	1	2
8	Pacific Rim	Guillermo del Toro	Sci-Fi	PG-13	1	2
9	Lion King	Roger Allers & Rob Minkoff	Animation	G	1	2
10	Mortdecai	David Koepp	Comedy	R	1	2
11	Red Tails	Anthony Hemingway	History	PG-13	1	2
12	Scott Pilgrim vs. the World	Edgar Wright	Comedy	PG-13	1	2
13	Tombstone	George P. Cosmatos & K...	Western	R	1	3
14	Gremlins	Joe Dante	Horror	PG	1	3
15	The Lego Movie	Christopher Miller & Phil ...	Animation	PG	1	3
16	The Monster Squad	Fred Dekker	Comedy	PG-13	1	3
17	Killer Klowns from Outer Space	Stephen Chiodo	Horror	PG-13	1	3
18	The Mask	Chuck Russell	Comedy	PG-13	1	3
*	NULL	NULL	NULL	NULL	NULL	NULL

Get a movie –

-Swagger-

The screenshot shows the Swagger UI interface for a REST API. The main title is "Get a movie – Swagger". The interface includes sections for "Parameters", "Responses", and "Code".

Parameters:

Name	Description
movie_id	Movie ID

Responses:

200:

```
Content-Type: application/json
Date: Fri, 20 Aug 2021 09:01:11 GMT
Server: Apache/2.4.41 (Ubuntu)
Transfer-Encoding: chunked

{
    "id": 1,
    "title": "The Shawshank Redemption",
    "year": 1994,
    "rating": 9.2
}
```

Request URL: http://localhost:4000/v1/api/movies/movie_id/movie_id

Code:

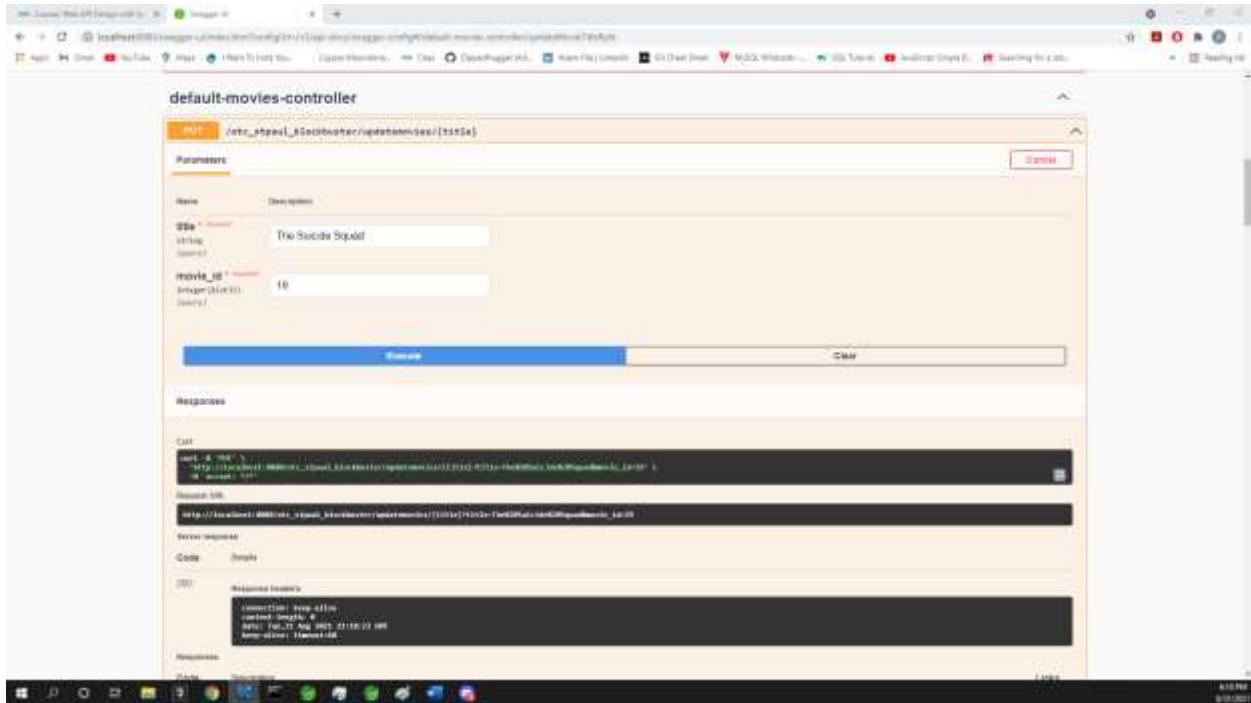
200:

```
Response-body:
{
    "id": 1,
    "title": "The Shawshank Redemption",
    "year": 1994,
    "rating": 9.2
}

Response-header:
Connection: keep-alive
Content-Type: application/json
Date: Fri, 20 Aug 2021 09:01:11 GMT
Server: Apache/2.4.41 (Ubuntu)
Transfer-Encoding: chunked
```

Update a Title-

-Swagger-

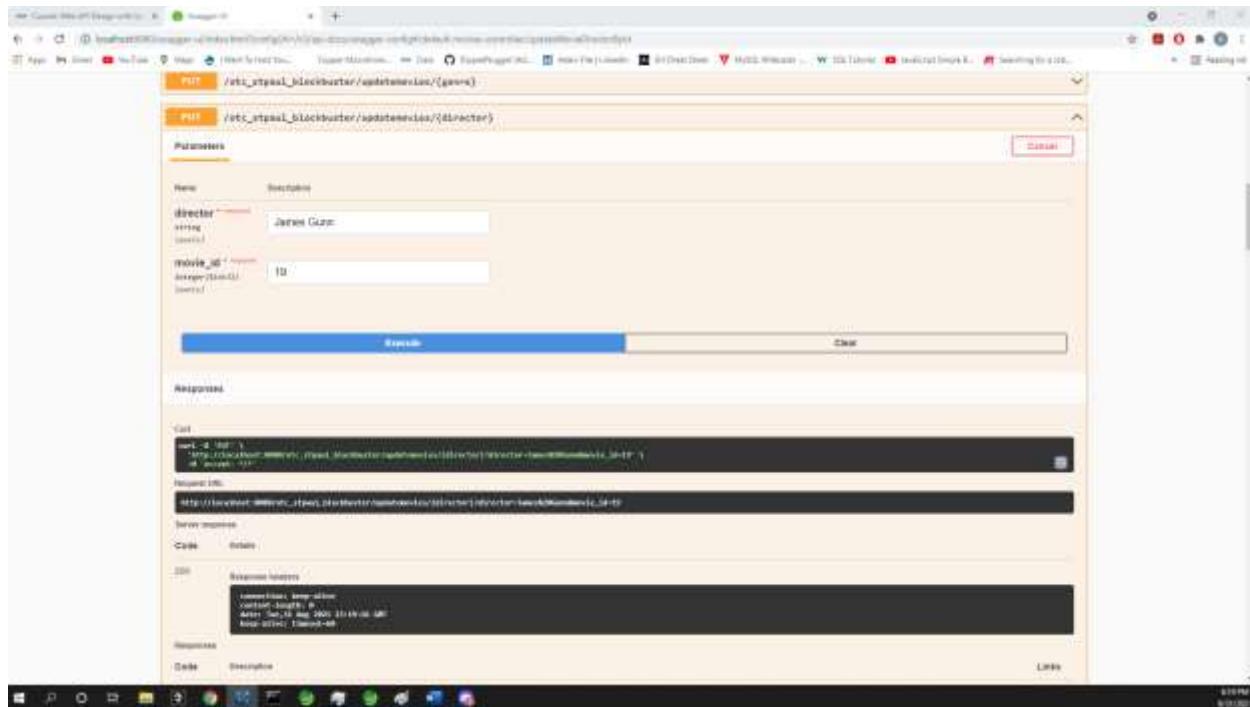


-Database-

	movie_id	title	director	genre	rating	in_stock	store_location
▶	1	Spider-Man 2	Sam Raimi	Comic Book	PG-13	1	1
	2	Back to the Future 3	Robert Zemeckis	Sci-Fi	PG	1	1
	3	Robinson Crusoe on Mars	Byron Haskins	Sci-Fi	PG	1	1
	4	Ghostbusters	Ivan Reitman	Comedy	PG	1	1
	5	Krampus	Micheal Daugherty	Christmas	PG-13	1	1
	6	Lawless	John Hillcoat	Drama	R	1	1
	7	Monty Python and the Holy Grail	Terry Gilliam & Terry Jones	Comedy	PG	1	2
	8	Pacific Rim	Guillermo del Toro	Sci-Fi	PG-13	1	2
	9	Lion King	Roger Allers & Rob Minkoff	Animation Comedy	G	1	2
	10	Mortdecai	David Koepp		R	1	2
	11	Red Tails	Anthony Hemingway	History	PG-13	1	2
	12	Scott Pilgrim vs. the World	Edgar Wright	Comedy	PG-13	1	2
	13	Tombstone	George P. Cosmatos & K...	Western	R	1	3
	14	Gremlins	Joe Dante	Horror	PG	1	3
	15	The Lego Movie	Christopher Miller & Phil ...	Animation	PG	1	3
	16	The Monster Squad	Fred Dekker	Comedy	PG-13	1	3
	17	Killer Klowns from Outer Space	Stephen Chiodo	Horror	PG-13	1	3
	18	The Mask	Chuck Russell	Comedy	PG-13	1	3
	19	The Suicide Squad	David Ayer	Comic Book	PG-13	1	3
*	HULL	NULL	HULL	NULL	NULL	NULL	NULL

Update a Director –

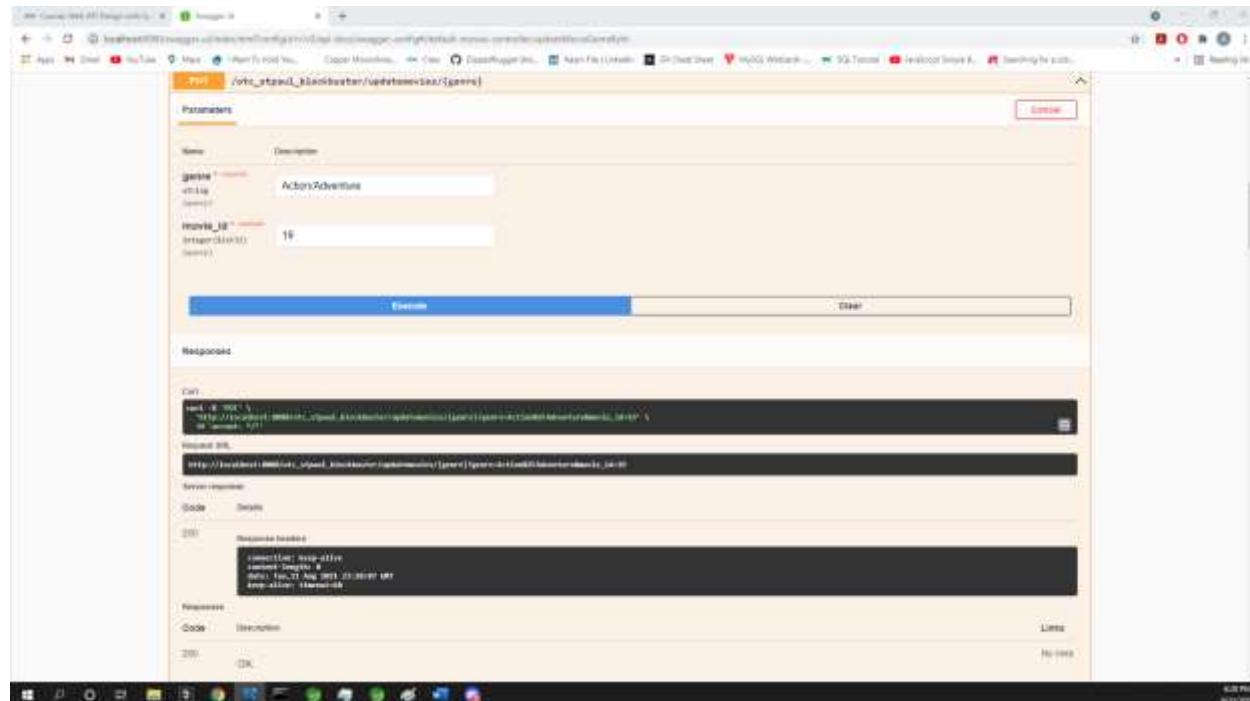
-Swagger-



-Database-

Update a Genre –

-Swagger-



-Database-

Update if In Stock-

-Swagger-

The screenshot shows a browser window with the following details:

- URL:** `http://localhost:8080/movies-admin/rest/stock/stockkeeper/updatebalance/{in_stock}`
- Method:** POST
- Request Body (Parameters):**

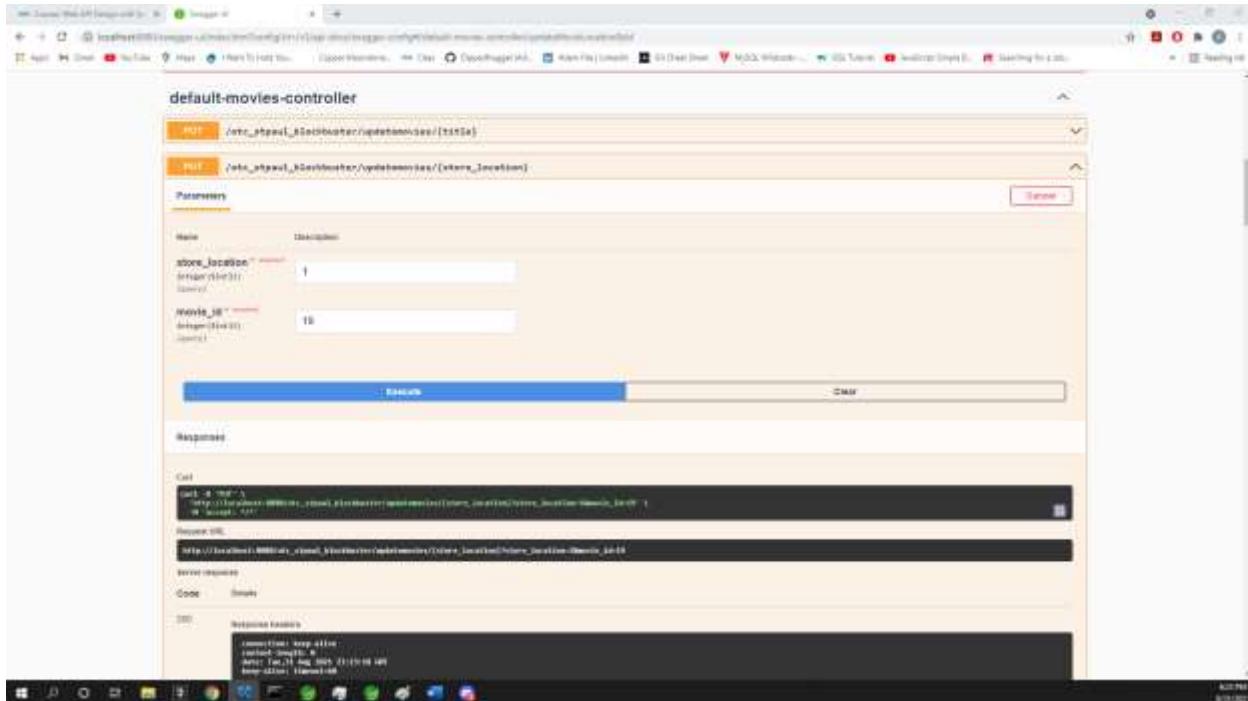
Name	Description
in_stock	10
movie_id	10
- Response Headers:**
 - Content-Type: application/json
 - Date: Tue, 28 Mar 2017 20:20:46 GMT
 - Server: GlassFish Server 4.1
- Response Body:**

```
{"message": "Stock updated successfully!", "status": 200}
```

-Database-

Update Store Location –

-Swagger-



-Database-

Result Grid		Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:		
	movie_id	title	director	genre	rating	in_stock	store_location
▶	1	Spider-Man 2	Sam Raimi	Edit current row	PG-13	1	1
	2	Back to the Future 3	Robert Zemeckis	Sci-Fi	PG	1	1
	3	Robinson Crusoe on Mars	Byron Haskins	Sci-Fi	PG	1	1
	4	Ghostbusters	Ivan Reitman	Comedy	PG	1	1
	5	Krampus	Micheal Daugherty	Christmas	PG-13	1	1
	6	Lawless	John Hillcoat	Drama	R	1	1
	7	Monty Python and the Holy Grail	Terry Gilliam & Terry Jones	Comedy	PG	1	2
	8	Pacific Rim	Guillermo del Toro	Sci-Fi	PG-13	1	2
	9	Lion King	Roger Allers & Rob Minkoff	Animation	G	1	2
	10	Mortdecai	David Koepf	Comedy	R	1	2
	11	Red Tails	Anthony Hemingway	History	PG-13	1	2
	12	Scott Pilgrim vs. the World	Edgar Wright	Comedy	PG-13	1	2
	13	Tombstone	George P. Cosmatos & K...	Western	R	1	3
	14	Gremlins	Joe Dante	Horror	PG	1	3
	15	The Lego Movie	Christopher Miller & Phil ...	Animation	PG	1	3
	16	The Monster Squad	Fred Dekker	Comedy	PG-13	1	3
	17	Killer Klowns from Outer Space	Stephen Chiodo	Horror	PG-13	1	3
	18	The Mask	Chuck Russell	Comedy	PG-13	1	3
	19	The Suicide Squad	James Gunn	Action/Adventure	R	0	1
*	NULL	NULL	NULL	NULL	HULL	NULL	NULL

Delete a Movie –

-Swagger-

The screenshot shows the JMeter interface with a test plan containing a single thread group. The thread group has one sampler named "DeleteMovie". The sampler is configured with the following details:

- Method:** DELETE
- URL:** http://192.168.1.100:8080/jetro_otpst_blockbuster/deleteMovie/{movie_id}
- Parameters:** movie_id=10

The "Advanced" tab is selected. In the "Response" section, the response code is 200 OK, and the response message is "OK". The "Raw" tab shows the raw response content.

-Database-

Concessions:

Variables: Gum Drops 1.99 12

Update Variables: Dots .99 18

Starting Database –

	concessions_id	full_name	price	quantity
▶	1	REESES PIECES	3.99	12
▼	2	JUNIOR MINTS	4.89	12
...	3	HOT TAMALES	3.99	12
...	4	Jiffy Pop Butter Popcorn	3.64	6
...	5	Twizzlers	1.99	18
...	6	Snow Caps	2.99	12
*	NULL	NULL	NULL	NULL

Create a Concession –

-Swagger-

The screenshot shows the Swagger UI interface for a REST API. The top section is titled "Create a Concession – -Swagger-". Below it, there's a form with fields for "full_name" (set to "Gum Drops"), "price" (set to "1.99"), and "quantity" (set to "12"). The "Responses" section shows a successful response (201) with a status message and headers like "Content-Type: application/json", "Content-Length: 0", and "Date: Tue, 30 Aug 2024 20:45:48 GMT". At the bottom, there's a "Responses" table with a single row for "Created".

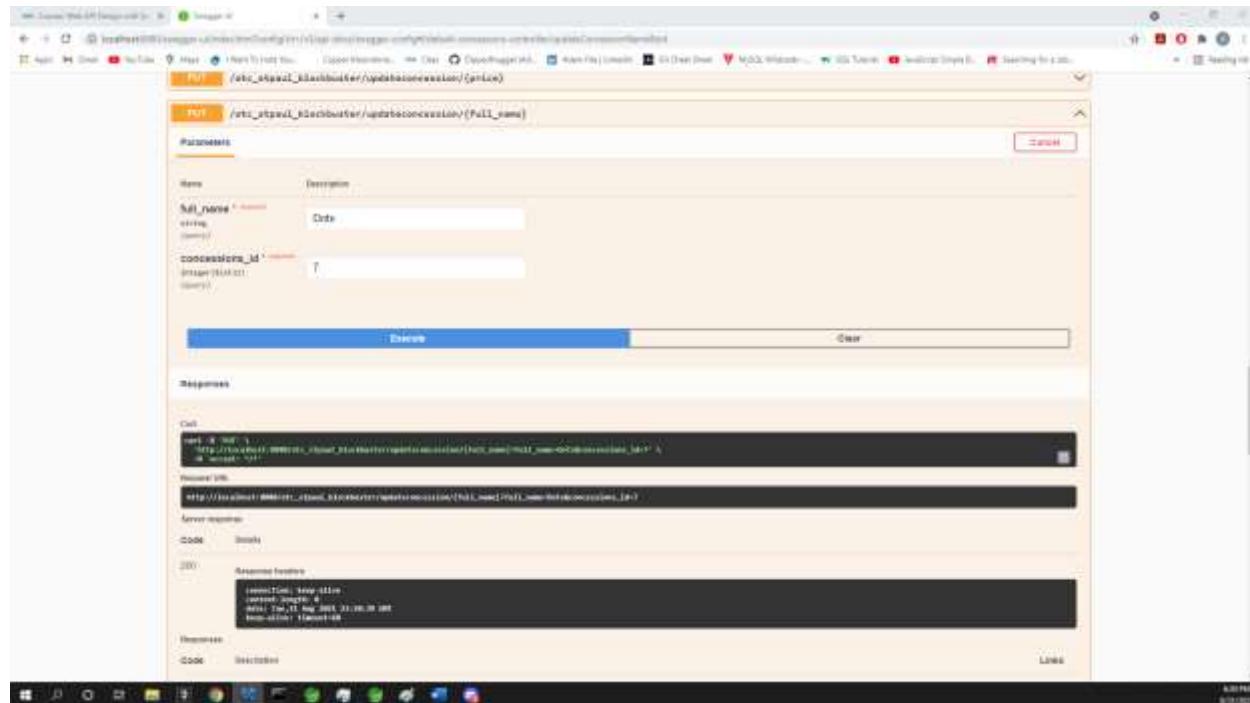
-Database-

The screenshot shows a "Result Grid" from MySQL Workbench displaying a table of concession items. The columns are "concessions_id", "full_name", "price", and "quantity". The data rows are:

	concessions_id	full_name	price	quantity
▶	1	REESES PIECES	3.99	12
▶	2	JUNIOR MINTS	4.89	12
▶	3	HOT TAMALES	3.99	12
▶	4	Jiffy Pop Butter Popcorn	3.64	6
▶	5	Twizzlers	1.99	18
▶	6	Snow Caps	2.99	12
▶	7	Gum Drops	1.99	12
*	NULL	NULL	NULL	NULL

Update Full Name –

-Swagger-



The screenshot shows the Swagger UI interface for a REST API. The URL is `/abc_ataud/classbuster/updateconcessionary/{full_name}`. The method is `PUT`. The `Parameters` section contains two fields:

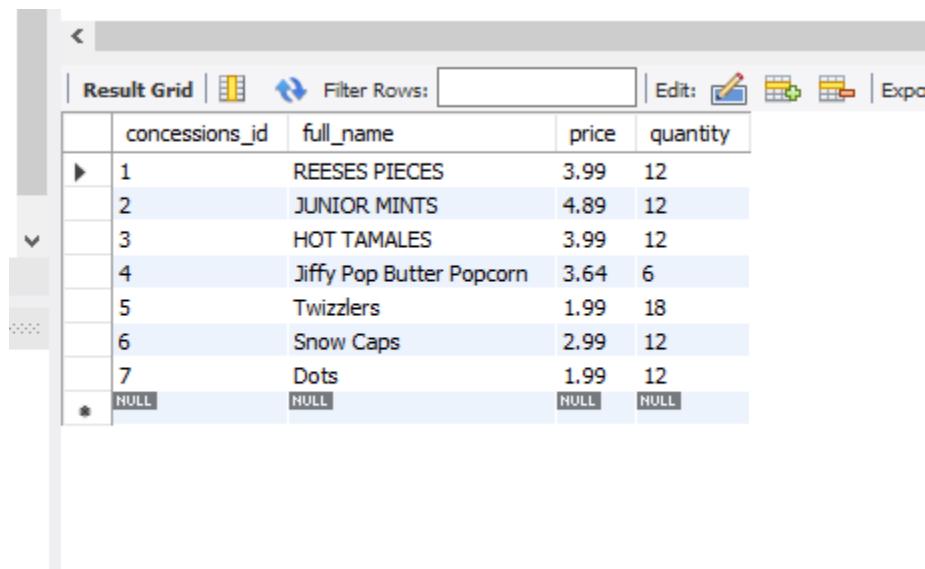
- `full_name`: `string` (required)
- `concessions_id`: `integer` (required)

The `Responses` section includes a `Code` field with the value `200`, which displays the response headers:

```
Content-Type: application/json
Date: Tue, 28 Jun 2022 11:46:38 GMT
Expires: Fri, 28 Jun 2022 11:46:38 GMT
```

Below the responses are sections for `Request URL` (`http://localhost:8080/abc_ataud/classbuster/updateconcessionary/{full_name}/{full_name}/full_name`) and `Server Response`.

-Database-



The screenshot shows a MySQL Workbench Result Grid displaying data from a table. The columns are `concessions_id`, `full_name`, `price`, and `quantity`. The data is as follows:

	concessions_id	full_name	price	quantity
▶	1	REESES PIECES	3.99	12
▼	2	JUNIOR MINTS	4.89	12
...	3	HOT TAMALES	3.99	12
...	4	Jiffy Pop Butter Popcorn	3.64	6
...	5	Twizzlers	1.99	18
...	6	Snow Caps	2.99	12
...	7	Dots	1.99	12
*	NULL	NULL	NULL	NULL

Update Price –

-Swagger-

The screenshot shows the Swagger UI interface for a REST API. The URL is `/api/v1/concessions/{concessions_id}/updateConcession/{price}`. The method is `POST`. The parameters section shows two fields: `price` (float) set to `0.99` and `concessions_id` (integer) set to `7`. Below the parameters, there are tabs for `Example` and `Data`. The responses section shows a `200` OK response with a JSON example:

```
curl -X POST "http://localhost:8000/api/v1/concessions/7/updateConcession/0.99" -H "Content-Type: application/json"
```

Raw response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Tue, 30 Aug 2024 20:41:01 GMT
Server: Caddy/0.99
```

Response headers:

Name	Description	Type	Format
Content-Type	application/json	String	application/json

Response body:

```
{"status": "Success", "message": "Concession updated successfully", "data": {"id": 7, "name": "Dots", "price": 0.99, "quantity": 12}}
```

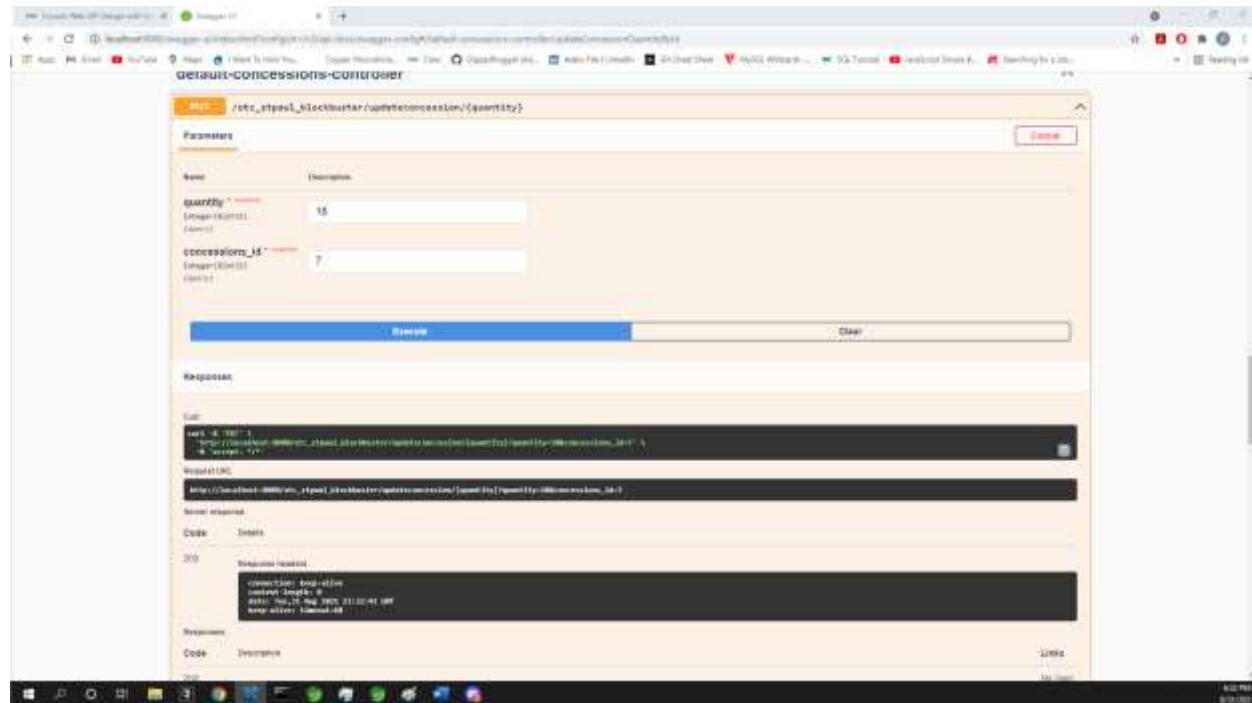
-Database-

The screenshot shows the MySQL Workbench interface with a result grid displaying data from a table named `concessions`. The columns are `concessions_id`, `full_name`, `price`, and `quantity`. The data is as follows:

	concessions_id	full_name	price	quantity
▶	1	REESES PIECES	3.99	12
▶	2	JUNIOR MINTS	4.89	12
▶	3	HOT TAMALES	3.99	12
▶	4	Jiffy Pop Butter Popcorn	3.64	6
▶	5	Twizzlers	1.99	18
▶	6	Snow Caps	2.99	12
▶	7	Dots	0.99	12
*	NULL	NULL	NULL	NULL

Update Quantity –

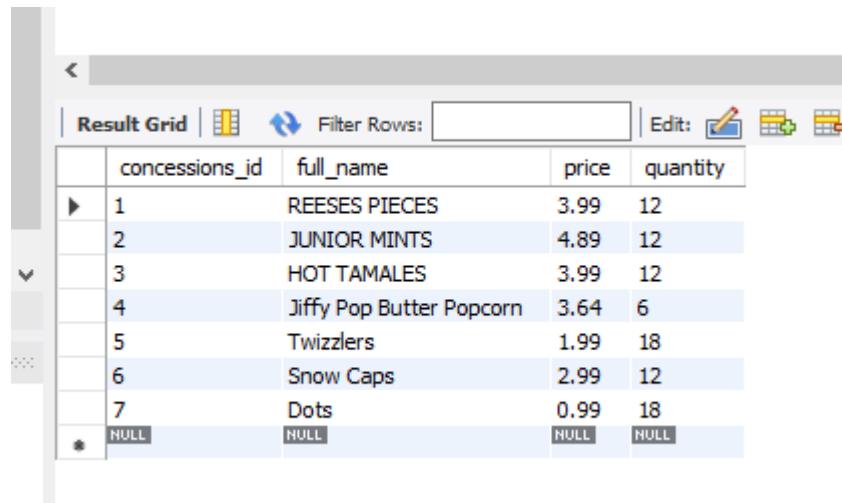
-Swagger-



The screenshot shows the Swagger UI interface for a REST API. The URL is `/v1/concessions/{concessions_id}/updateConcessionQuantity`. The parameters section shows two fields: `quantity` (set to 15) and `concessions_id` (set to 7). The response section shows a successful update with status code 200, returning a JSON object with the following data:

```
{  "concessions_id": 7,  "full_name": "Jiffy Pop Butter Popcorn",  "price": 3.64,  "quantity": 12}
```

-Database-



	concessions_id	full_name	price	quantity
▶	1	REESES PIECES	3.99	12
▼	2	JUNIOR MINTS	4.89	12
...	3	HOT TAMALES	3.99	12
	4	Jiffy Pop Butter Popcorn	3.64	6
	5	Twizzlers	1.99	18
	6	Snow Caps	2.99	12
	7	Dots	0.99	18
*	NULL	NULL	NULL	NULL

Get Concession-

-Swagger-

The screenshot shows the Swagger UI interface for a REST API. At the top, there are two tabs: 'API' and 'UI'. Below them, the URL is set to `http://localhost:8080/concessionservice/api/v1/concessions/getConcessionById/{concession_id}`. There are two main sections: 'Parameters' and 'Responses'.

Parameters section:

Name	Description
concession_id	Response body

The 'concession_id' parameter is defined with the type 'integer' and a value of '7' is entered into the input field. Below the input field is a 'Cancel' button.

Responses section:

For the 'Default' response (status code 200), the response body is shown as:

```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Mon, 11 Aug 2013 21:03:30 GMT
{
    "concession": {
        "id": 7,
        "name": "Burger King"
    }
}
```

Below the response body, the 'Headers' section shows:

```
connection: keep-alive
content-type: application/json
date: Mon, 11 Aug 2013 21:03:30 GMT
Content-Length: 16
```

At the bottom right of the responses panel is a 'Download' button.

Delete Concession –

-Swagger-

The screenshot shows the Swagger UI interface for a REST API. The main title is "Delete Concession –". Below it, the subtitle "-Swagger-" is displayed. The central part of the screen is a detailed view of the "DELETE /v1/api/blockbuster/concessions/{concessions_id}" endpoint. This section includes:

- Parameters:** A table with one row:

name: concessions_id	description: integer
----------------------	----------------------
- Responses:** A table with two rows:

Code: 200	Description: Response message
Code: 404	Description: Resource not found

Below these sections, there are examples of the API call structure, including the URL, headers, and body.

-Database-

The screenshot shows the MySQL Workbench interface with a "Result Grid" tab selected. The grid displays the contents of a table named "concessions". The columns are labeled "concessions_id", "full_name", "price", and "quantity". The data is as follows:

	concessions_id	full_name	price	quantity
▶	1	REESES PIECES	3.99	12
▼	2	JUNIOR MINTS	4.89	12
⋮	3	HOT TAMALES	3.99	12
⋮	4	Jiffy Pop Butter Popcorn	3.64	6
⋮	5	Twizzlers	1.99	18
⋮	6	Snow Caps	2.99	12
*	NULL	NULL	NULL	NULL

Transaction:

The Running of the transaction code Is done via a TEST.

That is correct we actually managed to get a TEST running.

Starting Database –

-Transactions-

A screenshot of a MySQL Workbench interface showing a table named 'Transactions'. The table has four columns: 'transaction_id', 'customer_idFK', 'store_idFK', and 'total'. There is one row with values: transaction_id is NULL, customer_idFK is NULL, store_idFK is NULL, and total is NULL. The table is displayed in a 'Result Grid' format with standard column headers and a 'Edit' button.

	transaction_id	customer_idFK	store_idFK	total
*	NULL	NULL	NULL	NULL

-Stored Transaction Movie List-

A screenshot of a MySQL Workbench interface showing a table named 'Stored Transaction Movie List'. The table has two columns: 'movie_idFK' and 'transaction_idFK'. There is one row with values: movie_idFK is NULL and transaction_idFK is NULL. The table is displayed in a 'Result Grid' format with standard column headers and a 'Wrap Cell Content' button.

	movie_idFK	transaction_idFK
...	NULL	NULL

-Stored Transaction Concession List-

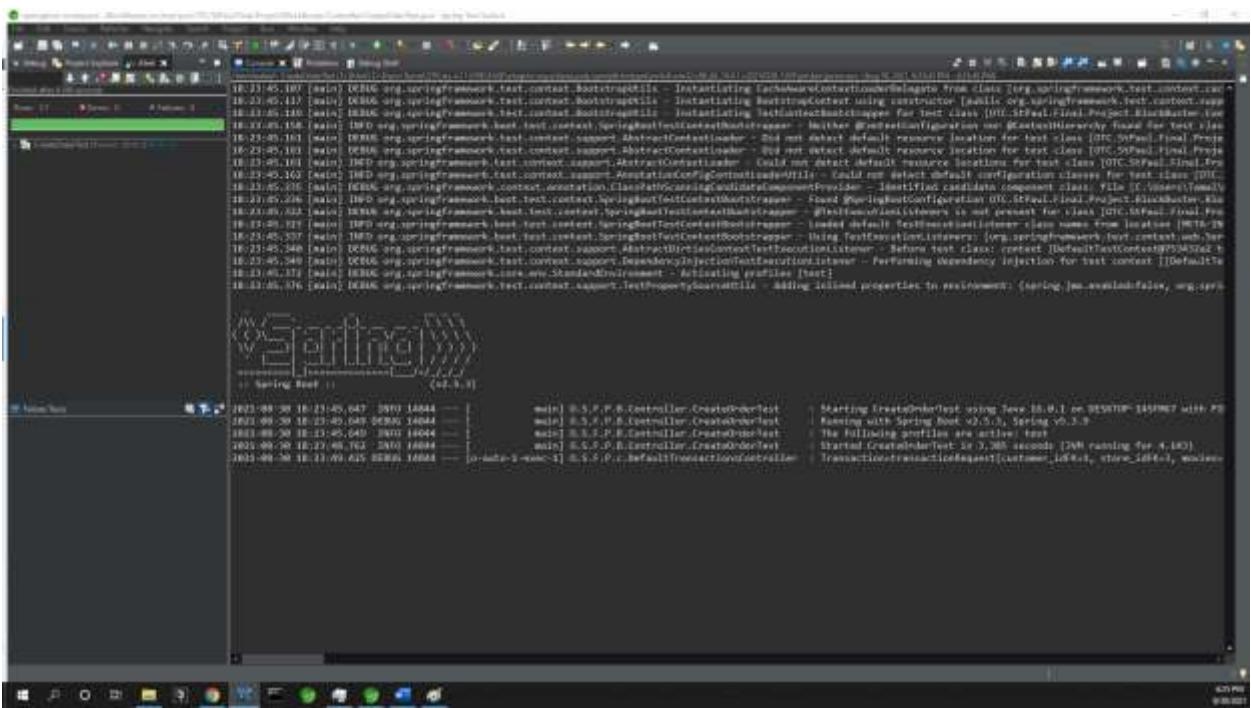
A screenshot of a MySQL Workbench interface showing a table named 'Stored Transaction Concession List'. The table has two columns: 'concessions_idFK' and 'transaction_idFK'. There is one row with values: concessions_idFK is NULL and transaction_idFK is NULL. The table is displayed in a 'Result Grid' format with standard column headers and a 'Wrap Cell Content' button.

	concessions_idFK	transaction_idFK
...	NULL	NULL

Test Name-

```
@Test  
void testCreateOrderReturnsSuccess201() {
```

Green Bar –



Created Database –

-Transactions-

A screenshot of a MySQL Workbench result grid. The grid has four columns: transaction_id, customer_idFK, store_idFK, and total. The data row contains values 6, 1, 3, and 5.97 respectively. There are also 'NULL' entries in the first three columns.

	transaction_id	customer_idFK	store_idFK	total
▶	6	1	3	5.97
*	NULL	NULL	NULL	NULL

-Stored Transaction Movie List-

A screenshot of a MySQL Workbench result grid. The grid has two columns: movie_idFK and transaction_idFK. The data rows contain (13, 6) and (15, 6).

	movie_idFK	transaction_idFK
▶	13	6
	15	6

-Stored Transaction Concession List-

A screenshot of a MySQL Workbench result grid. The grid has two columns: concessions_idFK and transaction_idFK. The data row contains (5, 6).

	concessions_idFK	transaction_idFK
▶	5	6

THE END!

“Lets all go to the lobby, lets all go to the lobby, lets all go to the lobby, AND grab ourselves a SNACK!”