Novelus

Challenge 1 Documentation

Bus Drivers Shifts

Mortada Ghanem
September 30, 2021

Table of Contents

1. Service

The application is called Bus Drivers Shifts, designed to manage the schedules of buses and drivers.

1.1. Features

The provided features by the application are:

- Manage Drivers (CRUD operations)
- Manage Buses (CRUD operations)
- Manage Schedules (CRUD operations)
- View and filter Schedules by Driver and Week
- View and filter Schedules by Bus and Week

1.2. Roles

In this application, we defined 2 user roles:

Manager Role: a manager can manage Drivers/Buses/Schedules.

Employee Role: an employee can view and filter schedules by Driver/Bus and Date.

These roles are used in front-end after login, so the manager is navigated to the ManagerView react component and the employee is navigated to the EmployeeView react component.

1.3. Used Frameworks

1.3.1. Back-end Development

Java Spring Boot web framework has been used to develop the back-end part of the application, it's been used to create the database and to create REST APIs. It is the link between the database and the front-end part.

1.3.2. Front-end Development

React JS has been used to develop the front-end part of the applications, it's been used to create web pages and forms and to manipulate data by invoking APIs created in the back-end part to access database records.

1.3.3. Database

H2-Database has been used as our application's database, it is an in-memory database that will be created when we start the server.

2. How to Build/Run the Application

In order to build the application, the following requirements should be satisfied:

- Java v1.8
- Java IDE (Eclipse or IntelliJ)
- NodeJS v17

To run the back-end part using IDE, you have to do the following steps:

1. Open your Java IDE
2. Import Existing Maven Project
3. Enter the directory path of the project
4. Select the project and press OK
5. When it loads, Update Maven Project to download Maven required dependencies that are set in *pom.xml* file
6. Run Spring Boot Application

To run the back-end part using command line, you have to do the following steps:

1. Open command line in Spring Boot project folder directory
2. Run this command: `./mvnw spring-boot:run`

Note: The application can be accessed on localhost:8080/bus-drivers-shifts

To run the front-end part:

1. Open command line in React JS project folder directory
2. Run this command: `./npm start`

Note: The application can be accessed by default on localhost:3000/

Here's a list of pre-defined users that can be used to login:

- Username: "**manager**", Password: "**manager**"
- Username: "**employee**", Password: "**employee**"
- Username: "**admin**", Password: "**admin**"

3. API Documentation

The application is accessed on localhost:8080/bus-drivers-shifts/

3.1. User

3.1.1. Login

Method	URL
POST	api/login

Request:

Parameters	Type	Values	Required
username	Request Body	string	Yes
Password	Request Body	string	Yes

Response:

Case Success:

Code: 200 OK

{

“user”: {

“username”: <username>,

“roles”: [] <roles>

}

}

Case Authentication Failed:

Code: 401 UNAUTHORIZED

{

“code”: 401,

“message”: “Unauthorized”

}

3.2. Driver

3.2.1. Get All Drivers

Method	URL
GET	api/driver/all

Request:

Parameters	Type	Values	Required
No Parameters			

Response:

Case Success:

Code: 200 OK

[

<list of driver objects>

]

3.2.2. Get Driver

Method	URL
GET	api/driver/{ssn}

Request:

Parameters	Type	Values	Required
ssn	Path Variable	string	Yes

Response:

Case Success:

Code: 200 OK

{

“ssn”: <ssn>,

“firstName”: <firstName>,

“lastName”: <lastName>

}

Case Driver Not Found:

Code: 400 BAD REQUEST

{

“identifier”: “Driver with SSN: <ssn> not found”

}

3.2.3. Save/Update Driver

Method	URL
POST	api/driver

Request:

Parameters	Type	Values	Required
ssn	Request Body	string	Yes
firstName	Request Body	string	Yes
lastName	Request Body	string	Yes

Response:

Case Success:

Code: 200 OK

{

“ssn”: <ssn>,

“firstName”: <firstName>,

“lastName”: <lastName>

}

Case Data Not Provided:

Code: 400 BAD REQUEST

{

“ssn”: “Social Security Number (SSN) is required”,

“firstName”: “First name is required”,

“lastName”: ”Last name is required”

}

3.2.4. Delete Driver

Method	URL
DELETE	api/driver/{ssn}

Request:

Parameters	Type	Values	Required

ssn	Path Variable	string	Yes

Response:

Case Success:

Code: 200 OK

“Driver with SSN: <ssn> was deleted”

Case Driver Not Found:

Code: 400 BAD REQUEST

{

“identifier”: “Driver with SSN: <ssn> not found”

}

3.3. Bus

3.3.1. Get All Buses

Method	URL
GET	api/bus/all

Request:

Parameters	Type	Values	Required
No Parameters			

Response:

Case Success:

Code: 200 OK

[

<list of bus objects>

]

3.3.2. Get Bus

Method	URL
GET	api/driver/{id}

Request:

Parameters	Type	Values	Required
id	Path Variable	integer	Yes

Response:

Case Success:

Code: 200 OK

{

“id”: <id>,

“capacity”: <capacity>,

“model”: <model>,

“make”: <make>,

“driverSSN”: <driverSSN>,

“associatedDriver”: { <driver object> }

}

Case Bus Not Found:

Code: 400 BAD REQUEST

{

“identifier”: “Bus with ID: <id> not found”

}

3.3.3. Save/Update Bus

Method	URL
POST	api/bus

Request:

Parameters	Type	Values	Required
Id	Request Body	integer	No
capacity	Request Body	string	Yes
model	Request Body	string	Yes
make	Request Body	string	Yes
driverSSN	Request Body	string	No

Response:

Case Success:

Code: 200 OK

{

“id”: <id>,

“capacity”: <capacity>,

“model”: <model>,

“make”: <make>,

“driverSSN”: <driverSSN>,

“associatedDriver”: { <driver object> } or ”null”

}

Case Data Not Provided:

Code: 400 BAD REQUEST

{

“model”: “Model is required”,

“make”: ”Make is required”

}

3.3.4. Delete Bus

Method	URL
DELETE	api/bus/{id}

Request:

Parameters	Type	Values	Required
id	Path Variable	integer	Yes

Response:

Case Success:

Code: 200 OK

“Bus with ID: <id> was deleted”

Case Bus Not Found:

Code: 400 BAD REQUEST

{

“identifier”: “Bus with ID: <id> not found”

}

3.4. Schedule

3.4.1. Get All Schedules

Method	URL
GET	api/schedule/all

Request:

Parameters	Type	Values	Required
No Parameters			

Response:

Case Success:

Code: 200 OK

[

<list of schedule objects>

]

3.4.2. Get Schedule

Method	URL
GET	api/schedule/{id}

Request:

Parameters	Type	Values	Required
id	Path Variable	integer	Yes

Response:

Case Success:

Code: 200 OK

{

“id”: <id>,

“busID”: <busID>,

“driverSSN”: <driverSSN>,

“day”: <day format(“yyyy-mm-dd”)>,

“timeFrom”: <timeFrom format(“hh:mm:ss”)>,

“timeTo”: <timeTo format(“hh:mm:ss”)>

}

Case Schedule Not Found:

Code: 400 BAD REQUEST

{

“identifier”: “Schedule with ID: <id> not found”

}

3.4.3. Save/Update Schedule

Method	URL
POST	api/schedule

Request:

Parameters	Type	Values	Required
Id	Request Body	integer	No
busID	Request Body	integer	No
driverSSN	Request Body	string	No
day	Request Body	date	No
timeFrom	Request Body	time	No
timeTo	Request Body	time	No

Response:

Case Success:

Code: 200 OK

{

“id”: <id>,

“busID”: <busID>,

“driverSSN”: <driverSSN>,

“day”: <day format(“yyyy-mm-dd”)>,

“timeFrom”: <timeFrom format(“hh:mm:ss”)>,

“timeTo”: <timeTo format(“hh:mm:ss”)>

}

3.4.4. Delete Schedule

Method	URL
DELETE	api/schedule/{id}

Request:

Parameters	Type	Values	Required
id	Path Variable	integer	Yes

Response:

Case Success:

Code: 200 OK

"Schedule with ID: <id> was deleted"

Case Schedule Not Found:

Code: 400 BAD REQUEST

{

"identifier": "Schedule with ID: <id> not found"

}

3.4.5. Get Schedules by Bus ID

Method	URL
GET	api/schedule/busID?busID={busID}

Request:

Parameters	Type	Values	Required
busID	Request Parameter	integer	Yes

Response:

Case Success:

Code: 200 OK

[

<list of schedule objects having busID=<busID>>

]

3.4.6. Get Schedules by Driver SSN

Method	URL
GET	api/schedule/driverSSN?driverSSN={driverSSN}

Request:

Parameters	Type	Values	Required
driverSSN	Request Parameter	string	Yes

Response:

Case Success:

Code: 200 OK

[

<list of schedule objects having driverSSN =<driverSSN>>

]

3.4.7. Get Weekly Schedules by Bus ID

Method	URL
GET	api/schedule/busID/Weekly?busID={busID}&day={day}

Request:

Parameters	Type	Values	Required
busID	Request Parameter	integer	Yes
day	Request Parameter	date	Yes

Response:

Case Success:

Code: 200 OK

[

<list of schedule objects having busID=<busID> and day between <day> and <day+6>>

]

3.4.8. Get Weekly Schedules by Driver SSN

Method	URL
GET	api/schedule/driverSSN/Weekly?driverSSN={driverSSN}&day={day}

Request:

Parameters	Type	Values	Required
driverSSN	Request Parameter	string	Yes
day	Request Parameter	date	Yes

Response:

Case Success:

Code: 200 OK

[

<list of schedule objects having driverSSN =< driverSSN> and day between <day> and <day+6>>

]

3.4.9. Get Weekly Schedules by Date

Method	URL
GET	api/schedule/date/Weekly? date={date}

Request:

Parameters	Type	Values	Required
date	Request Parameter	date	Yes

Response:

Case Success:

Code: 200 OK

[

<list of schedule objects day between <date> and <date+6>

]

4. Not Implemented Parts

These parts not implemented for time reasons.

4.1. Unit Tests

To test REST controllers in our Spring Boot application, I wanted to create Junit test classes in *src/test/java*, and by using injected MockMvc using @Autowired, I can perform HTTP requests into my API and set the expected results, then run the test to check if it succeeded or not.

4.2. API Security

To secure the API, I wanted to configure Spring Security with JWT authentication so each request sent should contain JWT token sent in the request header to make sure this user is authenticated and if he's authorized to access some actions or controllers in our Spring Boot application.