Az űrlap teteje

Az űrlap alja

Az űrlap teteje

Az űrlap alja

**Pallida Orientation Exam - Java focus**

**0. Initialize a Spring Boot project in this folder**

* create a proper .gitignore file
* use dependencies: web, thymeleaf, mysql

**1. Frontend**

* You should create only one template for the exercise

**The form**

* You should display a simple form to query a database, *don't waste time with design*
  + Create an input field in which the user can enter a licence plate, like HMZ-140 or any part of the license plate (like MZ-1)
  + Create a button with the text Search
    - Clicking on the button should send a get request to the backend
    - The frontend should display the results from the DB.
  + Add two links next to the form:
    - Police cars only - Filter for cars only starting with the RB prefix
    - Diplomats only - Filter for cars only staring with DT prefix
    - if any of them is clicked list only the cars with the defined prefixed license plate

**Search results**

* Display the results from the database in a HTML <table> the following format for the query HMZ

| Licence plate | Brand | Model | Color | Year |

| HMZ-140 | [Audi] | A8 | red | 2016 |

| HMZ-555 | [BMW] | Z4 | pink | 2011 |

* The app can display multiple rows of results since it should match partial queries as well
* Clicking on the brand field should display all cars with the same type
  + So clicking on "Audi" should restart the search and list all cars from the database regardless of the original query that resulted with this list
* If the user submits an invalid licence plate, like HMZ-@#5 then display an error message instead of the table
  + Display message: Sorry, the submitted licence plate is not valid

**2. Backend**

**Checking input data**

You should check all input submitted by the user (through the search form)!

* Only allow alphanumeric characters from the user: [A-Z], [0-9] and the - dash symbol

For the licence plate lookup only:

* Maximum length of the query is 7 characters

If the submitted data doesn't match any above criteria, display the error message detailed above (in the fronend part).

**Endpoints**

You should create three endpoints:

**GET /search**

The licence plate data and the limiting filter options should be passed via query string.

Example query:

http://localhost:8080/search?q=HMZ-140 http://localhost:8080/search?police=1

* use the police parameter to filter for police cars
* use the diplomat parameter to filter for diplomat cars

The endpoint should render an HTML displaying the results.

**GET /search/{brand}**

This endpoint should return all the cars with the provided brand

Example query:

http://localhost:8080/search/audi

The endpoint should render an HTML displaying the results.

**GET /api/search/{brand}**

This endpoint should return all the cars with the provided brand

Example query:

http://localhost:8080/api/search/audi

**Responses**

The endpoint should return this data structure:

{

"result": "ok",

"data": [

{

"plate": "HMZ-140",

"car\_brand": "Audi",

"car\_model": "A8",

"year": 2016,

"color": "red"

},

{

"plate": "HMZ-555",

"car\_brand": "BMZ",

"car\_model": "Z4",

"year": 2011,

"color": "pink"

}

]

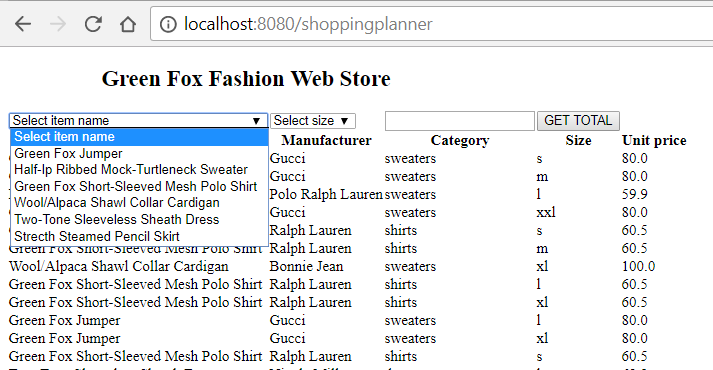
}

**Pallida Orientation Exam Retake**

**1) Green Fox Fashion Web Store**

Build a webshop that people can use to order some merchandise from the Greenfox store.

**Main page**

[](https://github.com/Kamokamocsai/pallida-orientation-exam-retake/blob/master/assets/main2.png)

* the **frontend** should have
  + a heading with the title of the site
  + dropdown list with all unique name options
  + dropdown list with all unique size options
  + input field to set the quantity of clothes
  + a GET TOTAL button which redirects to the Summary page
  + list the clothing items of the database in table

**Backend**

* when GET TOTAL button is submitted the backend should
  + summarize the total price based on the quantity that was given in the inputfield and send these information to the Summary page

**Endpoints**

* you should create these endpoints:

**GET /warehouse**

* the endpoint should render an HTML displaying the full list of clothes(explained above)

**POST /warehouse/summary**

* this endpoint should receive all the information of the selected item
* this endpoint should render an other HTML after receiving the data, which should display the extract of the selected item in a table:
  + item name
  + manufacturer
  + category
  + size
  + amount
  + sub-total price of the item (based on nr. of selected item)

**GET /warehouse/query**

* this endpoint should return all the clothes which has higher/lower/equal price than the given parameter

Example query:

http://localhost:8080/warehouse/query?price=50&type=lower

* this should return a json with every cloth where the unit price is lower than 50

{

"result": "ok",

"clothes":

[

{

"id": "21",

"item\_name": "Strecth Steamed Pencil Skirt",

"manufacturer": "Calvin Klein",

"category": "skirts",

"size": "s",

"unit\_price": 39.0,

},

{

"id": "24",

"item\_name": "Strecth Steamed Pencil Skirt",

"manufacturer": "Calvin Klein",

"category": "skirts",

"size": "m",

"unit\_price": 39.0,

},

]

}

**Summary page**

[](https://github.com/Kamokamocsai/pallida-orientation-exam-retake/blob/master/assets/summarypage.png)

* the **frontend** should have:
  + a heading with the title of the page
  + a link back to the main page
  + the list of selected items with the properties defined above

**2) Endpoint TESTing**

* create 2 endpoint tests all together, you can choose on which endpoint/endpoints

**3) Question time**

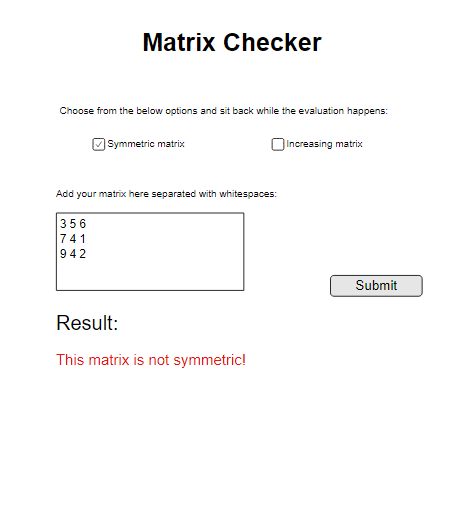
Why would you declare your class as an abstract class?

**Fulvipes Orientation Exam**

**1) Symmetric Matrix**

You will create a web page which can check if the matrix is symmetric or not.

**Matrix Checker**

[](https://github.com/mzsolt68/greenfox-orientation-exam/blob/cf8277859ca787fc12b046d4c80e669ba371e591/assets/index1.png)

**Frontend**

* the **frontend** should have:
  + a heading with the title
  + nice intro text
  + radio buttons for the matrix validation options
  + input text area for the matrix
  + submit button for sending the matrix to backend to check

**Backend**

* only a square matrix should be submitted in the input field,
  + otherwise the app should send an error message
* user can choose which validation type will the matrix ride through
* do the validation methods in backend
* in case the matrix is valid send a success message in 'Result' section
* in case the matrix is not valid send an error
* if success, save the matrix into a database with the following fields:
  + id, type, date, matrixNumbers

**Endpoints**

* you should create these endpoints:

**GET /**

* the endpoint should render an HTML displaying the main page

**POST /matrix?validtype=[validationType]**

* this endpoint should be responsible for sending the matrix with the need of relevant validation type (symmetric or increasing)

**GET /matrixes**

* this REST endpoint should query every matrix that were successfully saved and valid

**2) Question time**

**What is the difference between primary key and foreign key in database?**

**What is the difference between unit and integration test?**