NPHC SWE Take Home Challenge

Overview

You're on a mission to help our HR department build an MVP employee salary management web service to facilitate management and analyse employees' salaries.

The web service maintains an employee list with the following information:

- id - unique alphanumeric ID assigned by the company.
- login - unique alphanumeric login assigned by the company.
- name - possibly non-unique name. May not be in English, so please use UTF-8 encoding.
- salary - decimal.
- startDate - date of start of employment.

As this is a backend challenge, the solution would not include any frontend element even though you will be building a backend service to support a frontend.

It is critical that you conform to the specifications precisely. We would evaluate this based on conformity to the specs.

USER STORY 1: Upload Users

I want to be able to upload employee data using a CSV file.

CSV Format

- File should be in UTF-8 encoding to allow for non-English characters.
- 1st row is the header row with 4 columns in the following order:

Column	Description
id	Internal employee ID, String, primary key
login	Employee login ID, String
name	Full name of employee
salary	Numeric decimal
startDate	Date of start of employment in one of two formats ● "yyyy-mm-dd", ie. "2020-11-23")

	● “dd-mmm-yy”, ie. “01-Jan-11”) Both are in local timezone only.

Validation rules:

- Salary in decimal that is >= 0.0
- All 5 columns **must be filled**.
- “id” is the primary key that uniquely identifies each record.
- “id", “login” must be unique. “name” can be non-unique.
- Any row starting with “#” is considered a comment and ignored.

Upload API

Request URL	/users/upload
Request Method	POST
Request Content Type	multipart/form-data
Request Params	● file - File to upload
Response Status	● 200 - Success but no data updated. ● 201 - Data created or uploaded. ● 400 - Bad input - parsing error, duplicate row, invalid salary etc.
Response Content Type	application/json
Response Body	`{ "message": … }` Leave it to your discretion on error messages.

- If an entry in the CSV contains an employee ID that already exists in the database, the existing entry in the database is updated. If no entry exists, a new entry is created. Hence, we can keep uploading the same file without noticeable impact on the data.
- There should not be more than one row with the same employee ID in each file. Otherwise, error is returned.
- While “login” is not the primary key, it must be unique.
- If an error exists during the upload, the entire upload is discarded with an error.

Acceptance Criteria

- To be able to successfully upload a set of CSV files provided by us to test various validation requirements.
- Clear error messages to help diagnose errors with input files.

USER STORY 2: Fetch List of Employees

I want to have the building block for a mechanism to view a list of employee information. I should be able to do the following in the user interface through an API call:

- To view only n employees details at one time.
- Able to filter employees based on salary range.
- Results should be ordered by employee ID unless you are supporting custom ordering.
- Bonus: Able to order the list by id, employee name, login or salary ascending or descending.
- Bonus: Able to add additional filter of your choice (ie. match wildcard name etc).

Fetch API

Request URL	/users
Method	GET
Request Params	Optional parameters: ● minSalary - return only if salary >= minimum salary. Default is 0. ● maxSalary - return only if salary < maximum salary. Default is 4000.00. ● offset - starting offset of results to return. Default is 0 => start. ● limit - max number of results to return. Default is 0 => no limit. Bonus - add parameters of your own choice to support one or more of the following: ● filtering (define your filters) ● sorting
Response Status	● 200 - Success but no data updated ● 400 - Bad input, ie. bad parameters
Response Content Type	application/json

Response Body	<pre>{
	"results": [{
	"id": "emp0001",
	"name": "Harry Potter",
	"login": "hpotter",
	"salary": 1234.00,
	"startDate": "2001-11-16"
	},
	{
	"id": "emp0002",</pre>

	<pre>"name": "Severus Snape",
	"login": "ssnape",
	"salary": 1234.56,
	"startDate": "2001-11-16"
	},
	...
	]
	}</pre>

Acceptance Criteria

- When performing a search for min salary of 0 and max salary of 4000, it should only return records with salary between 0 <= salary < 4000
- Results should be sorted by id unless otherwise specified.

USER STORY 3: CRUD

I want to be able to modify name, login, or salary by selecting an existing employee from the dashboard, and I want the web app to be responsive and usable on both desktop and mobile devices.

- Able to serve a responsive web app. Should be able to support both desktop and mobile layout. Applicable also for user story #1 and #2
- Able to view name, login and salary given id
- Able to modify name, login, salary on the backend
- Able to see name, login and salary given id

CRUD APIs

Get

Request URL	/users/{$id}
Method	GET
Response Status	● 200 - Success but no data updated ● 400 - Bad input - no such employee
Response Content Type	application/json

Response Body	<pre>{
	"id": "e0001",
	"name": "Harry Potter",
	"login": "hpotter",
	"salary": 1234.00,
	"startDate": "2001-11-16"
	
	}</pre>

Create

Request URL	/users
Method	POST
Request Params	None
Request Content Type	application/json

Request Body	<pre>{
	"id": "emp0001",
	"name": "Harry Potter",
	"login": "hpotter",
	"salary": 1234.00,
	"startDate": "2001-11-16"
	}</pre>

Response Status	● 201 - New employee record created. ● 400 - Bad input - see error cases below.
Response Content Type	application/json
Response Body	● <pre>{"message": "Successfully created"}</pre> ● <pre>{"message": "Employee ID already exists"}</pre> ● <pre>{"message": "Employee login not unique"}</pre> ● <pre>{"message": "Invalid salary"}</pre> ● <pre>{"message": "Invalid date"}</pre>

Update

Request URL	/users/{id}
Method	PUT/PATCH
Response Status	● 200 - Successfully updated, ● 400 - Bad input - no such employee, login not unique etc.

Response Content Type	application/json
Response Body	• `{"message": "Successfully updated"}` • `{"message": "No such employee"}` • `{"message": "Employee login not unique"}` • `{"message": "Invalid salary"}` • `{"message": "Invalid date"}`

Delete

Request URL	/users/{id}
METHOD	DELETE
Response Status	• 200 - Employee deleted. • 400 - Bad input - no such employee.
Response Content Type	application/json
Response Body	• `{"message": "Successfully deleted"}` • `{"message": "No such employee"}`

Acceptance Criteria

- HTTP status codes, output encoding, and messages conform to specifications.
- Deleted records should not appear in subsequent operations.

Appendix A: Sample Data

CSV file with the following content.

```
id,login,name,salary,startDate
e0001,hpotter,Harry Potter,1234.00,16-Nov-01
e0002,rwesley,Ron Weasley,19234.50,2001-11-16
e0003,ssnape,Severus Snape,4000.0,2001-11-16
e0004,rhagrid,Rubeus Hagrid,3999.999,16-Nov-01
e0005,voldemort,Lord Voldemort,523.4,17-Nov-01
e0006,gwesley,Ginny Weasley,4000.004,18-Nov-01
e0007,hgranger,Hermione Granger,0.0,2001-11-18
e0008,adumbledore,Albus Dumbledore,34.23,2001-11-19
```

```
e0009,dmalfoy,Draco Malfoy,34234.5,2001-11-20
e0010,basilisk,Basilisk,23.43,21-Nov-01
```

Assessment Criteria

Submitting your assignment

- Your code must be in **Java or Kotlin** and recent version of **Spring Boot**, and backed by a **database** (embedded is sufficient).
- Your code must be hosted on **publicly-accessible repository** (eg. GitHub / BitBucket / GitLab). Please send us a link to your repository back to us.
- Your solution must listen on **port 8080**.
- Your solution **must conform** fully to the specifications.
- Your solution **must include** a complete set of unit tests.
- You should have a top-level README on how to build, test, and run your solution, as well as key assumptions and design decisions made.
- Do not host your app on a public server.
- Containers are acceptable . Make sure the Dockerfile is included in the repository.
- We are strict about having buildable and runnable code for which we can apply our automated testing to. Please make sure that your solution runs
- successfully and conforms fully to our specifications.

Metrics

1. Completeness of solution and consistency with the specs.
2. SWE best practices.
    a. SW architecture (modular, extensible)
    b. DB schema design
    c. Testing
    d. Version Control
3. Code quality.
    a. Clean code
    b. Consistent conventions
4. Bonus functionalities.
5. Good suggestions on how the solution can be improved.