

# TechHunt 2020 Take Home Assessment

## Overview

You're on a mission to help our HR dept build an MVP employee salary management web application to manage employees' salaries.

The app contains 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 that is  $\geq 0.00$ .

This is an MVP for us to gather more feedback. As a result, we have omitted a login feature as we are not sure how we are going to control usage nor what kind of authentication mechanism we will be using.

We have done our initial backlog review and came up with this prioritized backlog of 5 user stories. The PO has kickstarted the first sprint and has **prioritized user stories 1 and 2 as ones that must be completed**. It would be a bonus if you can complete the remaining user stories as well. For user stories 3, 4, and 5, you may choose to prioritize based on your strengths and preferences.

Please make sure that your implementation is backed by a robust and complete set of unit tests.

You must use Git and host your repo on GitHub or other similar services. Please refer to the last page on more information on how to submit your assignment

The repo must have a top-level Readme.md that briefly explains how to start up the application locally.

No constraints on tech stack other than the abovementioned. Pick what you are comfortable with.

You should expect to spend around 22 hours to complete this assignment.

## USER STORY 1: Upload Users (Prioritized)

I want to be able to upload a CSV file conforming to the following rules:

- File should be in UTF-8 encoding to allow for non-English characters.
- It contains 4 columns in the following order:
  - id - employee ID
  - login - employee login
  - name - non-unique name.
  - salary - decimal that is  $\geq 0.0$ .
- First row is for heading information that is ignored.
- All 4 columns must be filled.
- Any row starting with “#” is considered a comment and ignored.
- id and login must be unique. They cannot be repeated in another row.

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 it does not exist, then a new entry is created. It is possible to replace a login in the database as a result. However, this must not conflict with an existing login mapped to a different ID. For a complex use case of swapping logins between 2 IDs. You can accomplish this with 3 uploads: change first ID's login to temporary one, change 2nd ID's login to first ID's, change the first ID to 2nd ID's previous login.

The whole upload is considered a single transaction. If one or more of the rows fails validation, the entire file is rejected.

It is optional to support concurrent upload of files, ie. allow more than one user to upload a file at the same time. If you choose to support it, each file is uploaded in a single transaction such that the 2nd file overwrites any updates to the first file as one block. If you choose not to support it, you need to check and provide an error if a 2nd upload is initiated while a 1st upload is taking place.

I should be able to get confirmation of whether the upload has succeeded or failed.

To facilitate automated testing, the upload function should be exposed through an API as follows:

- HTTP POST /users/upload
- HTTP multipart form.
- CSV data in form field “file”, with content encoding “text/csv”.
- If upload is successful, HTTP 200 status is returned.
- If there is any error, return appropriate HTTP status code.

### Out of scope

Do not worry about large files that take a long time to upload and process. This will be handled by [User Story 4](#).

## Acceptance Criteria

- Successful uploading our good test data file with comments, both new and existing records.
- Unsuccessful upload of an empty file.
- Unsuccessful upload of files with partial number of incorrect number of columns (**both** too many **and** too few).
- Unsuccessful upload of files with some but not all rows with incorrectly formatted salaries.
- Unsuccessful upload of files with some but not all rows with salary < 0.0.
- Upload 2 files concurrently and receive expected results, or failure.

## USER STORY 2: Employee Dashboard Feature (Prioritized)

I want to be able to view a list of employee information on a web user interface. I should be able to do the following in the user interface:

- To view only 30 employees' details at one time
- To filter employees based on salary range
- To order the list by id, employee name, login or salary (ascending or descending)

The max amount of employee data you can fetch per API call is 30.

To facilitate automated testing, the API should be exposed through an restful API as follows:

- HTTP Get /users
- RequestParams: One example as follows
  - /users?minSalary=0&maxSalary=4000&offset=0&limit=30&sort=+name
  - offset is the position of the starting point to retrieve in the SQL query
  - limit is the total number of items to retrieve, should be fixed as 30
  - + means ascending, - means descending.
  - The sort order columns are id, name, login, salary
  - The sort request param takes in one sort key at a time
- If any of request params is missing, HTTP 400 status is returned
- If any of request params data format is invalid, HTTP 400 status is returned
- If successful, return data in json format as follows
  - { results: [{id:e0001, name: John, login: john, salary: 1000.00}, ...] }

## Out of scope

Don't worry about the data security of the data stored in the database. This will be handled in future user stories.

## Acceptance Criteria

- When I click on next page, it should be able to display records, with no overlapped records in page 1 and page 2
- When performing a search for min salary of 0 and max salary of 4000, it should only return records with salary between  $0 \leq \text{salary} \leq 4000$
- When sorting by name in ascending order, it should display in ascending order by name
- When sorting by salary in ascending order, it should display in ascending order by salary
- When sorting by login descending order, it should display in descending order by login

## USER STORY 3: CRUD Feature (Bonus)

I want to be able to modify name, login, or salary by selecting an existing employee from the dashboard. I also want the web app to be responsive and usable on both desktop and mobile devices. The Web Application should have the following functions:

- 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

To facilitate automated testing, the API should be exposed through an restful API as follows:

- HTTP /users/{id}
  - HTTP POST for create
  - HTTP PATCH for update
  - HTTP GET for retrieval
    - { id: e0001, name: John, login: [john@abc.com](mailto:john@abc.com), salary: 1000.00 }
  - HTTP DELETE for deletion
- Content-Type: application/json
- If json payload is invalid, HTTP 400 status is returned
- If successful for any operation, HTTP 200 status is returned

## Out of scope

Do not worry about 201, 204 http status. You can keep it simple and use HTTP 200 for this user story.

## Acceptance Criteria

- When I resize my web browser into phone form, the layout should be automatically changed to the appropriate form factor layout.
- When I click on edit function on an employee on USER story #2, I should be able to edit the employee details on the user interface.
  - When clicked on the save button, the modification should be persisted on the backend. At the employee dashboard page, you should be able to see the modified information.
- When I click on the delete button, a confirmation prompt should ask me if I really want to delete this employee.
  - When confirmed, the deletion should be persisted on the backend.
  - At the employee dashboard page, you should not be able to see this employee anymore.

## USER STORY 4: Better UX When Uploading Large CSV Files (Bonus)

It is possible that as the company grows larger, the CSV files to be uploaded become larger, maybe to the order of thousands of rows. It is also possible that I am not the only person uploading the files. I want to be able to upload large CSV files, and sometimes when someone else is uploading another file, that I have better and more instantaneous feedback from the UI.

I keep the improvements open-ended. One suggestion: UI should immediately return feedback that a file is being uploaded and notify me when upload has completed with success or failure immediately. It is ok if the user chooses to move to another page and we lose context of it.

Also take note that this would be more difficult to implement if you allow more than one uploads to take place. Hence, it may be easier if you disable that.

## Out of scope

I understand that web browsers and possibly the underlying cloud infrastructure may impose limits on how long an HTTP request can last. For this user story, I am looking for our application to be designed to provide for better UX, but without having to address any browser or infrastructure restrictions.

## Acceptance Criteria

- I want to be able to upload a good file with 5000 entries with success.
- I want to be able to start two concurrent uploads of files with 5000 entries in two different tabs and check that I obtain proper feedback.

## USER STORY 5: UI Localization (Bonus)

We are a global organization with employees in different countries. It is important in the longer term to be able to localize our UI to improve the UX for them.

I want to be able to view the UI using the locale settings of my browser environment. For example, if my browser environment is set to China, the UI should be in Chinese. If my browser environment is set to France, the UI should be in French. However, if my browser environment is set to Nepal, and no Nepali translation is available, then the UI defaults to English.

Given that this is a prototype, we only need the UI to be in English. We just need the application to be structured such that it can accept new language translations as and when we make them available.

### Out of scope

The salary is assumed to be in S\$ only. You do not need to display the currency using the local currency format.

### Acceptance Criteria

- If I change my desktop environment's language to Chinese, I should be viewing it in English still because there are no Chinese translations. I would like to be able to get indicators that it acknowledges that the language is to be in Chinese, but Chinese is not available.

# Appendix A: Sample Data

Non-exhaustive datasets meant to seed your imagination

```
e0001,hpotter,Harry Potter,1234.00
e0002,rwesley,Ron Weasley,19234.50
e0003,ssnape,Severus Snape,4000.0
e0004,rhagrid,Rubeus Hagrid,3999.999
e0005,voldemort,Lord Voldemort,523.4
e0006,gwesley,Ginny Weasley,4000.004
e0007,hgranger,Hermione Granger,0.0
e0008,adumbledore,Albus Dumbledore,34.23
e0009,dmalfoy,Draco Malfoy,34234.5
e0010,basilisk,Basilisk,-23.43
```


# Appendix B: UI Wireframe

This wireframe is meant to guide your imagination, to show how features come together. You are **encouraged** to enhance or change. Prioritize as you see fit.

Note: This wireframe **is not exhaustive**. There are other screens or modifications needed to flesh out all the user stories. You are free to design and implement these as they see fit.

For those taking up user story #3, We only have the wireframe for the desktop, but the user interface for both dashboard and CRUD features should be responsive and resize accordingly to suit different form factors of the screen size. One possible way is to convert the left navigation function panel into a side navigation drawer when resized into a mobile form factor.

## User Story #2 UI Wireframe



Long user name


Function 1

Function 2

Function 3

Function 4

Function 5




Minimum salary

Enter amount

\$

–





















Maximum salary

Enter amount

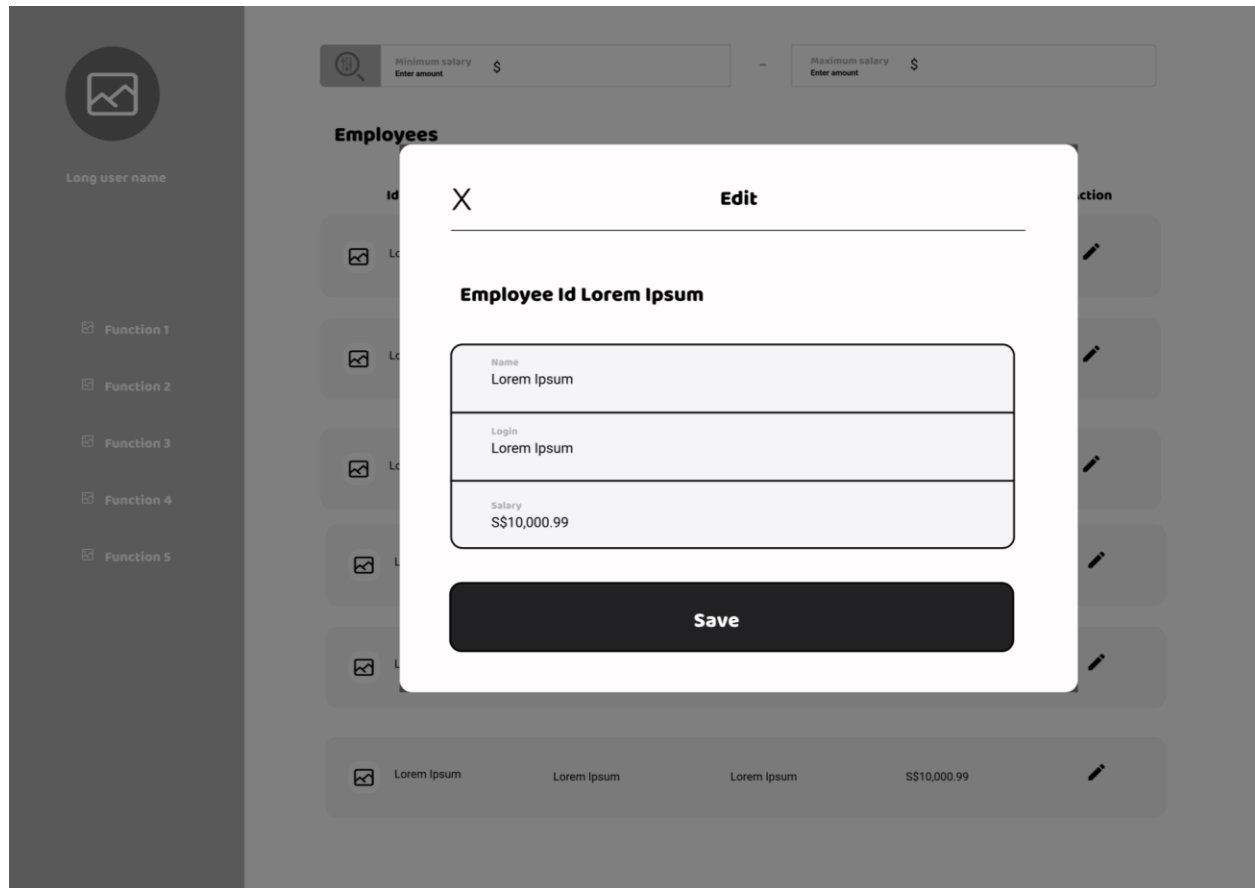
\$

Employees

Id	Name	Login	Salary	Action
 Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	SS\$10,000.99	 
 Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	SS\$10,000.99	 
 Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	SS\$10,000.99	 
 Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	SS\$10,000.99	 
 Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	SS\$10,000.99	 
 Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	SS\$10,000.99	 



## User Story #3 UI Wireframe



# Assessment Criteria

## Metrics

1. Completeness of solution **(30%)**
  - a. Each user story must have all its Assessment Criteria (ACs) met before it's considered complete.
  - b. Simplistically take a percentage as the score under this category
    - i. E.g. Of 2 compulsory user stories, 1 met all their ACs. 1 of 2 means you scored 1/2 of the total weightage for this metric which is 30%. So, the final score is 15%.
2. SWE best practices: **(25%)**
  - a. SW architecture (modular, extensible)
  - b. DB schema design
  - c. Testing (What test cases?)
  - d. Version Control
3. Code quality **(25%)**
  - a. Clean code
  - b. Consistent conventions
4. Bonus questions **(20%)**
  - a. You will get up to 20% depending on how well you attempt it.
  - b. It is not about how many bonus questions done, but how well each one is attempted.

We don't want you to kill yourself over the bonus questions. We want to emphasize quality over quantity. Make sure whatever you do, you spend enough time so you can do as well as you can.

## Submitting your assignment

- Your code must be hosted on Github, or any other similar service, in a publicly accessible repository (eg. GitHub / BitBucket / GitLab).
- You may include a section with the assumptions, interpretations you have made about the requirements above or notes on your architecture decisions.
- Do show the progress of your work with atomic git commits.
- Do not host your app on a public server.
- Please provide instructions on how to run your source code locally on our laptop in the README file.
- Containers are acceptable - make sure the Dockerfile is included in the repository.
- Please send us a link to your repository ([careers@tech.gov.sg](mailto:careers@tech.gov.sg)) when you completed your assessment.