Code Assignment - EveryHero

everyhero

Summary

The mayor of Gotham has a huge problem: Nasty villains. In the past, the problem could be solved to some extent, because Batman took care of peace in the city. But Batman is now old and frail and lives in a retirement home. The mayor was desperately looking for a replacement, but unfortunately superheroes are hard to find. He therefore had an idea: EveryHero. A digital platform where superheroes can register and showcase their specific skills. This way, the city administration can easily search for their heroes and contact them.

He asks us to implement his idea. The team has already implemented a rough version of the platform. It's now up to you to finish the application.

Getting started

1. Checkout the NX monorepo and install the package

```
git clone https://github.com/EveryS2020/code-assignment.git
cd everyhero
git checkout -b results-{YOUR NAME}
npm install
```

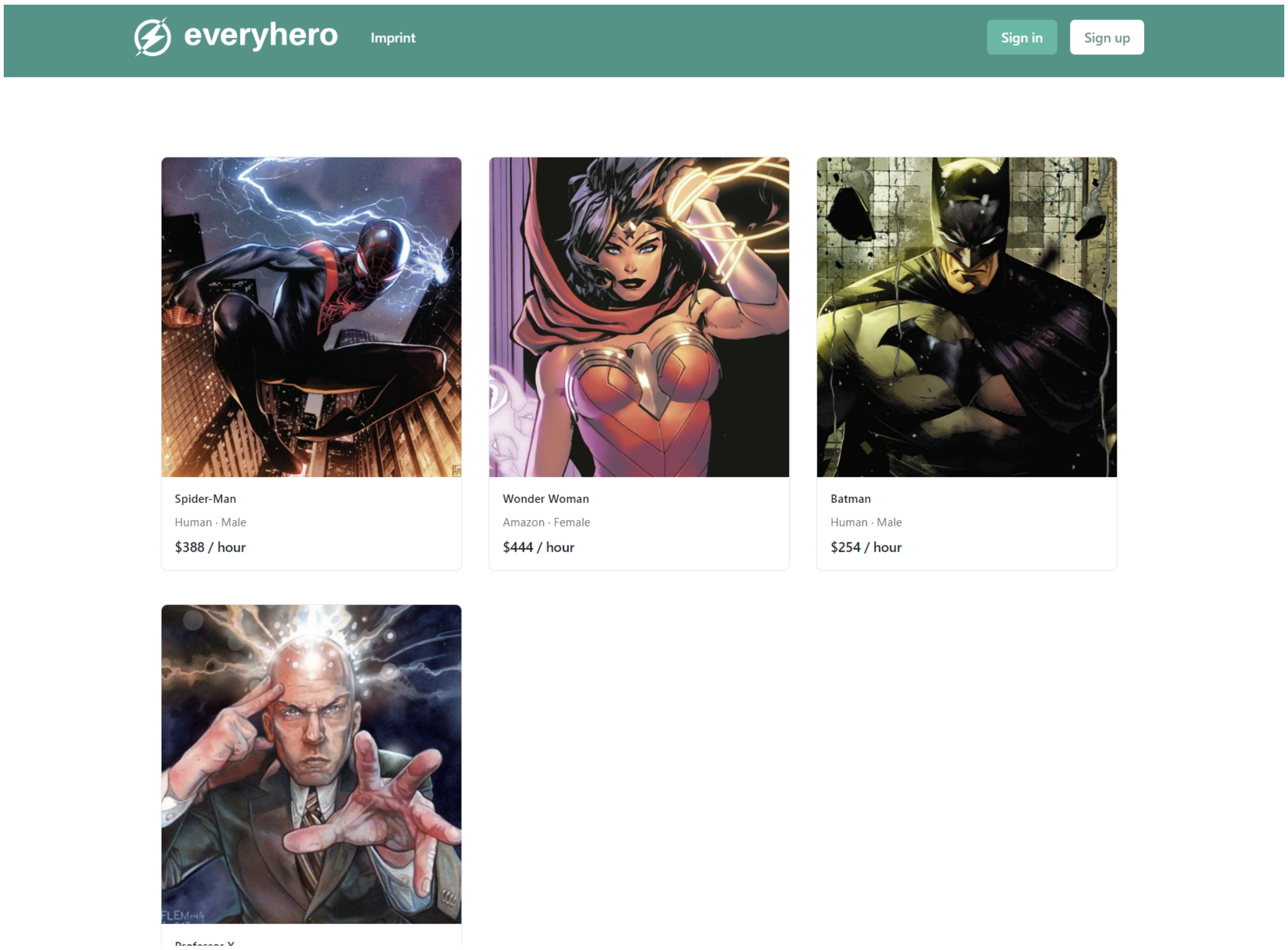
2. Start a combined (i.e. for both the frontend and backend) hot-reloading development server

```
nx run-many --target=serve --projects=frontend,backend
```

3. Verify that you are able to access the frontend by navigating to

🌎 http://localhost:4200

in your favorite browser. You'll see a list of 4 superheros on the front page (i.e. the HeroListPage). When clicking on a specific hero, your browser navigates to the detail page, which is not yet implemented at the moment.



Spider-Man
Human · Male
$388 / hour

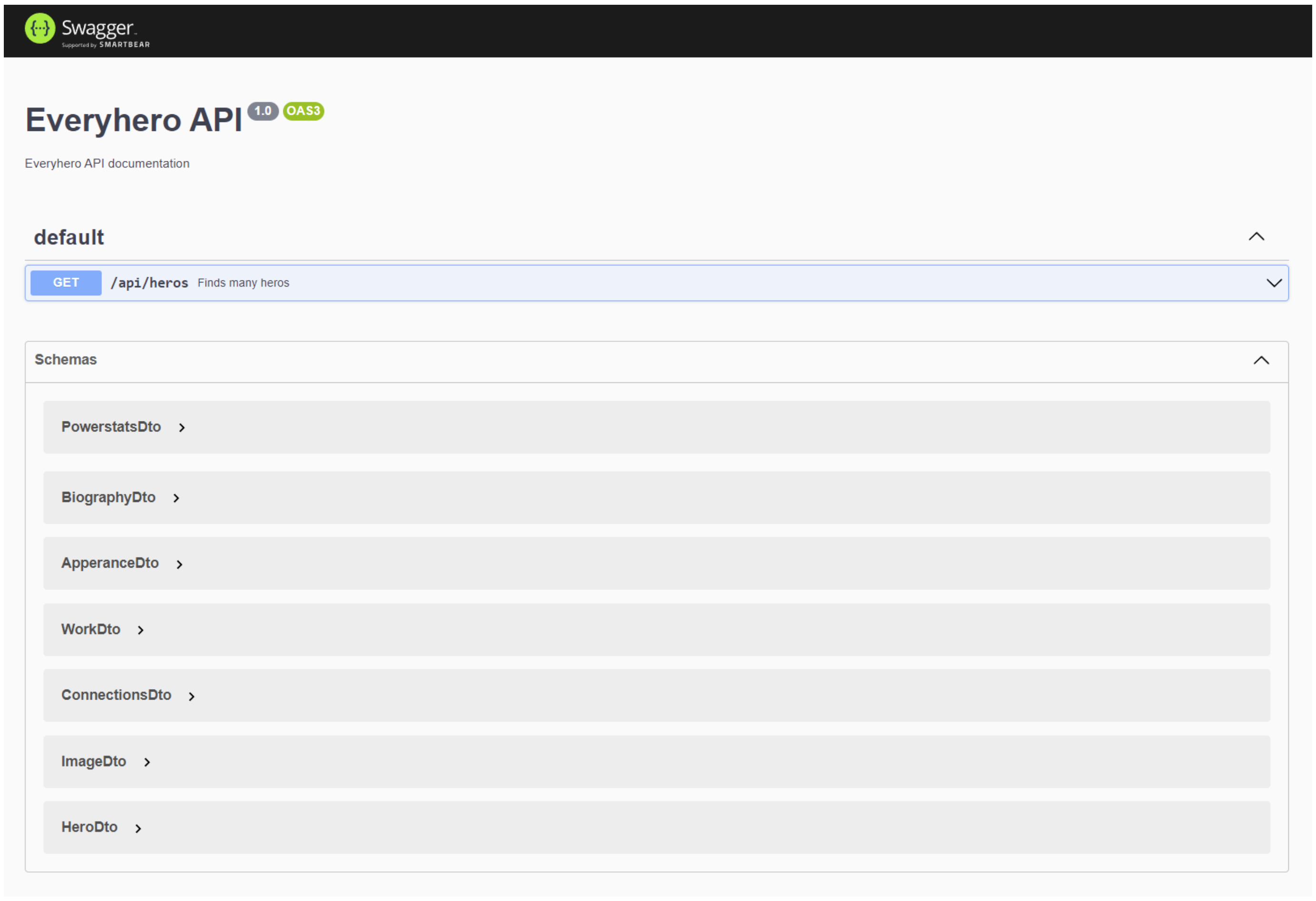
Wonder Woman
Amazon · Female
$444 / hour

Batman
Human · Male
$254 / hour

4. Next, head over to the swagger API documentation that the backend application offers for you:

🌎 http://localhost:3333/api

Feel free to play around with the operations listed, getting to know the models and mechanics of the endpoints will greatly help you in the to be tackled tasks.


Everyhero API 1.0 OAS3

Everyhero API documentation

default

GET /api/heros Finds many heros

Schemas

- PowerstatsDto
- BiographyDto
- ApperanceDto
- WorkDto
- ConnectionsDto
- ImageDto
- HeroDto

Architecture

This repository holds a nx monorepo consisting of two ***applications***:

- **frontend** (apps/frontend) - a NextJS application providing the frontend pages and components that the city administration personel will access.
- **backend** (apps/backend) - a NestJS application featuring RestAPI endpoints to query for superheros

Additionally, two ***libraries*** are already present:

- **lib-types** (libs/types) - a TS library holding type declarations for the system entities (e.g. *Hero*, *Query*, etc.)
- **lib-ui** (libs/ui) - a React libary offering basic view components that can be used to compose frontend views.

Frontend

Pages (`apps/frontend/pages`)

Filename	URL	Name	Purpose
index.tsx	/	HeroList	displays one card holding basic information for each superhero that matches current filtering criteria
[hero].tsx	/{hero-id}	HeroDetails	displays detailed information about the superhero whose ID matches the query parameter {hero-id}
imprint.tsx	/imprint	ImprintPage	(Not relevant)

Usage

Run

- `nx serve frontend` to start a hot reloading dev server

- `nx run frontend:lint` to lint the frontend project

Backend

Controller, Service, Models and DTOs (`apps/backend/src/app`)

Scope	Directory	Purpose
HeroController	app/hero.controller.ts	Rest API controller that provides an endpoint to query for hero records.
HeroService	app/hero.service.ts	Simple injectable service that has access to the Sequelize Model and provides a method to query hero records.
Models	app/modles/**.model.ts	Sequelize Models. The data is held by a sqlite file residing in /db /database.sqlite.
Dtos	app/dtos/**.dto.ts	Data Transfer Objects that are returned by the API. Annotated with OpenAPI/Swagger metadata to enrich documentation information on the Swagger Editor.

Usage

Run

- `nx serve backend` to start a hot reloading dev server
- `nx run backend:lint` to lint the frontend project

Tasks

A - Fetch and Populate the Hero List

So far, the 4 heros on the front page are mocked, in terms of the frontend just yielding the contents of a static JSON file. You'll find this file in `/apps/frontend/src/mocks/heros.json`.

Obviously, this is not optimal: As you might have already noticed, the `/api/heros` endpoint at the backend allows to properly fetch such data via the backend.

Start by understanding how the pagination parameters offered by the `/api/heros` endpoint work, feel free to play around with them. Then, think about ways on how the heros can be fetched (examples might include simple browser fetching, SWR, ReactQuery, etc.) and efficiently presented. You're completely free to chose whatever strategy you think solves the problem best.

Implement your solution by reusing the existing view components such as the HeroList, HeroItem components. Commit your results and include your reasoning behind your selection of the data fetching strategy in one of the commit messages.

B - Show Hero Details on the Hero Detail Page

As the mayor of Gotham City is now able to skim through basic information about the superheros he might engage, he wishes to be able to get detailed information about potential candidates when clicking on one of the heros on the HeroListPage. At this moment, the page exists, however, it is mostly empty.

Think about how the detail page might look like. You might start by creating a wireframe and iteratively approach your solution proposal by thinking about how to achieve high UX quality, meaningful grouping of information and neat visuals. Again, you're free to use whatever tools, libraries (if any), etc. you use.

Implement your solution by creating new components and finally composing the detail page. Commit your results in a separate commit.

C - Add Filters to the Hero List

By now, the heros are just listed and the mayor is forced to read through all ~ 700 heros whenever he decides to hire hero personell. As this is quite time consuming, he asks you to add a way on filtering the data. You and the city administration agree on implementing a tiny proof of concept that consists of

- Adding a Select Dropdown that allows to chose between the Options All / Males only / Females only on the Front Page. Whenever the selected option is changed, the result list shall update.
- The actual data filtering shall happen on the backend endpoint you already worked with (i.e. /api/heros). Add a query parameter for your filtering context and modify the current query to support your new argument.

Implement your solution by creating new frontend components and modifications on the backend. Again, commit your changes. Finally push your branch to the origin.