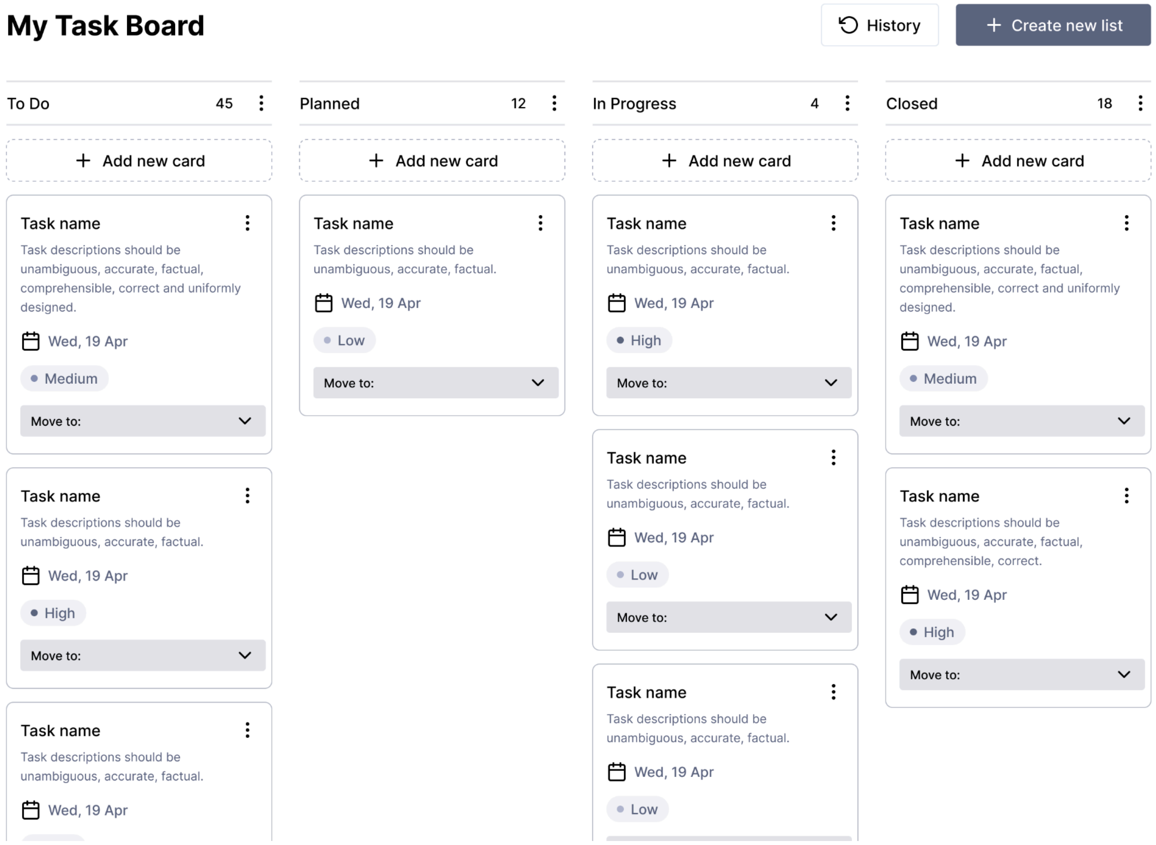
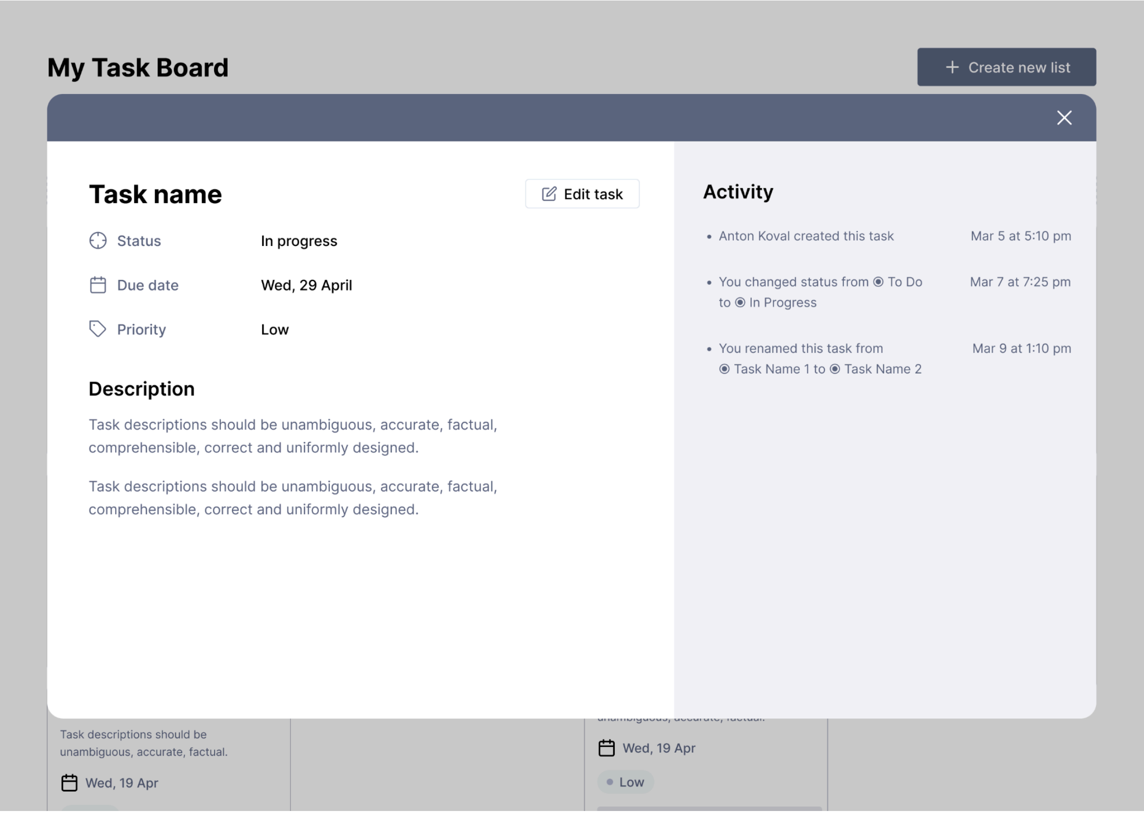
Jana Dantyszka St, Warsaw, Poland
Khimichna St., Lviv, Ukraine

RADENCY

Personal Project | Stage #1

Your task is to develop a Proof Of Concept (PoC) of a Personal Project Management Tool (Task Board). Below are the specific features and requirements.


My Task Board
History
Create new list
To Do 45
Planned 12
In Progress 4
Closed 18
Add new card
Task name
Task descriptions should be unambiguous, accurate, factual, comprehensible, correct and uniformly designed.
Task descriptions should be unambiguous, accurate, factual.
Task descriptions should be unambiguous, accurate, factual, comprehensible, correct.
Wed, 19 Apr
Low
Medium
High
Move to:



Task name

Status: In progress

Due date: Wed, 29 April

Priority: Low

Description

Task descriptions should be unambiguous, accurate, factual, comprehensible, correct and uniformly designed.

Task descriptions should be unambiguous, accurate, factual, comprehensible, correct and uniformly designed.

Activity

- Anton Koval created this task — Mar 5 at 5:10 pm
- You changed status from ◉ To Do to ◉ In Progress — Mar 7 at 7:25 pm
- You renamed this task from ◉ Task Name 1 to ◉ Task Name 2 — Mar 9 at 1:10 pm

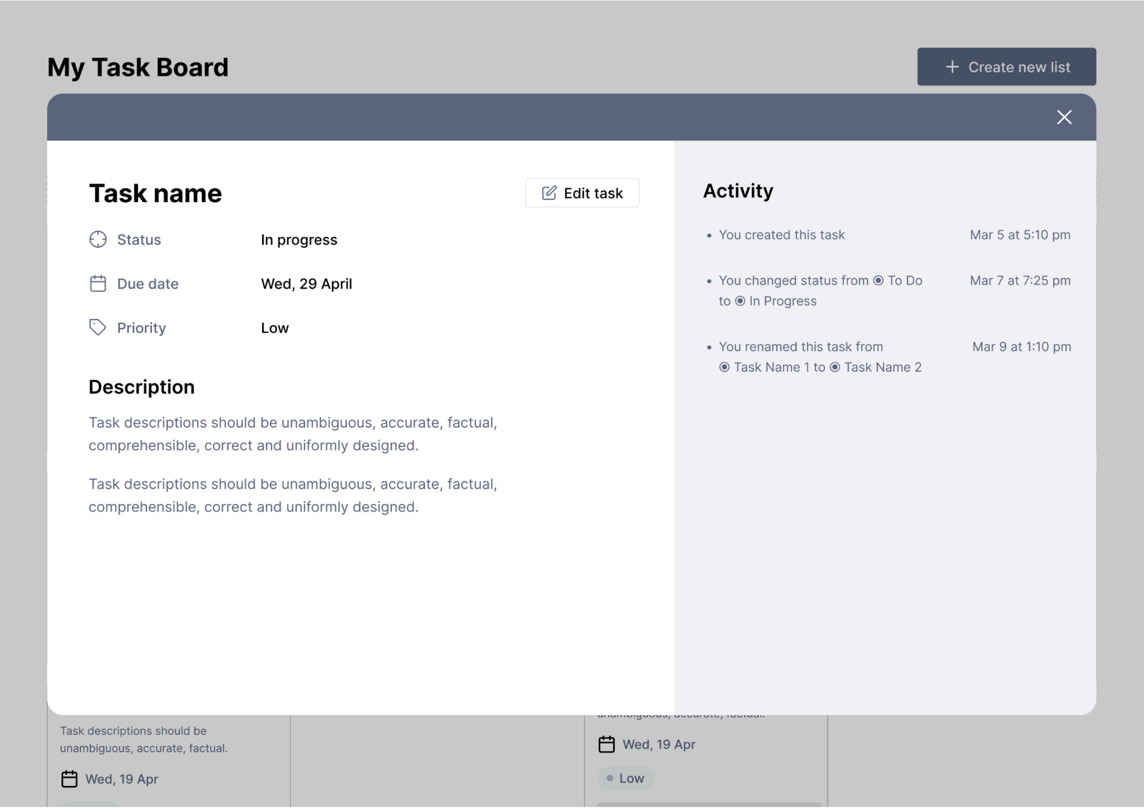
Task descriptions should be unambiguous, accurate, factual.

Wed, 19 Apr

unambiguous, accurate, factual.

Wed, 19 Apr

● Low



Task name

Status: In progress

Due date: Wed, 29 April

Priority: Low

Description

Task descriptions should be unambiguous, accurate, factual, comprehensible, correct and uniformly designed.

Task descriptions should be unambiguous, accurate, factual, comprehensible, correct and uniformly designed.

Activity

- You created this task — Mar 5 at 5:10 pm
- You changed status from ◉ To Do to ◉ In Progress — Mar 7 at 7:25 pm
- You renamed this task from ◉ Task Name 1 to ◉ Task Name 2 — Mar 9 at 1:10 pm

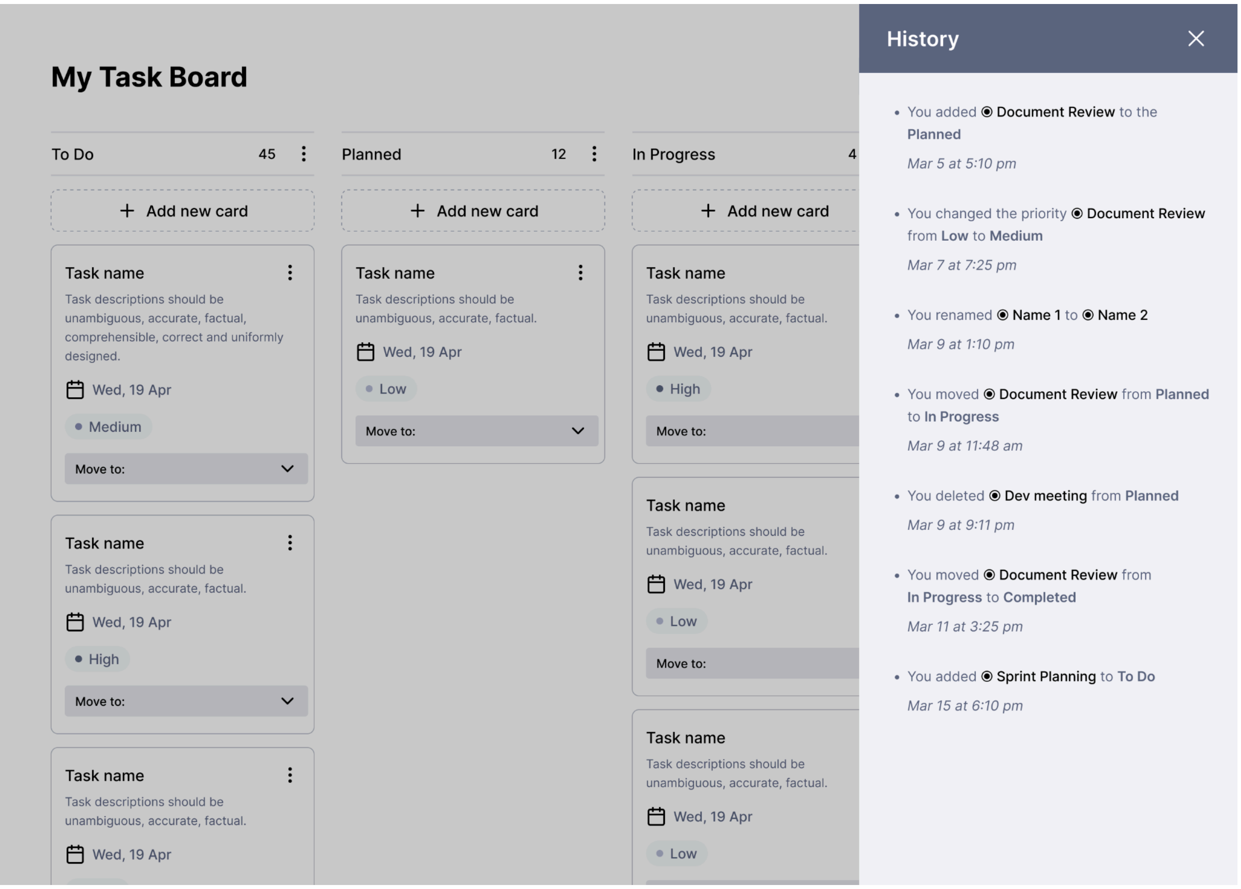
Task descriptions should be unambiguous, accurate, factual.

Wed, 19 Apr

unambiguous, accurate, factual.

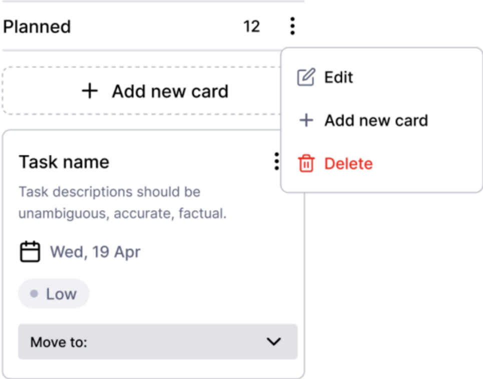
Wed, 19 Apr

● Low

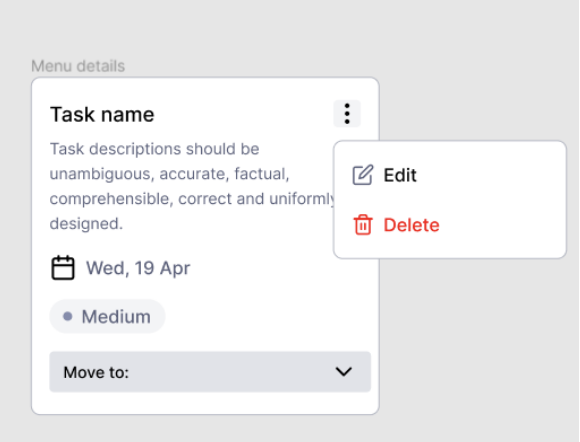


Task descriptions should be unambiguous, accurate, factual, comprehensible, correct and uniformly designed.

- You added ⦿ Document Review to the Planned
  Mar 5 at 5:10 pm
- You changed the priority ⦿ Document Review from Low to Medium
  Mar 7 at 7:25 pm
- You renamed ⦿ Name 1 to ⦿ Name 2
  Mar 9 at 1:10 pm
- You moved ⦿ Document Review from Planned to In Progress
  Mar 9 at 11:48 am
- You deleted ⦿ Dev meeting from Planned
  Mar 9 at 9:11 pm
- You moved ⦿ Document Review from In Progress to Completed
  Mar 11 at 3:25 pm
- You added ⦿ Sprint Planning to To Do
  Mar 15 at 6:10 pm



Task descriptions should be unambiguous, accurate, factual.



Task descriptions should be unambiguous, accurate, factual, comprehensible, correct and uniformly designed.

Functional requirements

Mandatory

1. Task Management:
   - Users should be able to create a **task list.** Each task **list has a name.**
   - Users should be able to add **tasks** to the **task list**.
   - Each **task** should have a **name**, **description**, **due date**, **priority**, and assigned team member(s).
   - Tasks should be movable between different **columns** (**task lists**) within a board (e.g., To Do, In Progress, Done). Use **“Move to”** for moving tasks.
2. Activity Log:
   - Maintain an **activity log** to track changes made to tasks (create, rename, add/change description, add/change due date, move, delete.
   - Users should be able to view the **history** of actions taken within the application for all tickets.
3. User Interface:
   - Design a user-friendly interface with **intuitive navigation** and **responsive design**.
   - Ensure **consistency in layout** and styling across different screens and devices.
4. Testing and Quality Assurance:
   - Perform thorough testing to identify and fix **bugs** or **issues**.

Technical requirements

Frontend

For the front-end part it is necessary to use the following stack: **Typescript**, **React**, **Redux/Redux-toolkit**. Any other React libraries, store solutions (React Context, Zustand) or style libraries could be used as well.

References:

- Use Create React App or React Documentation: Getting Started
- Follow Redux-Toolkit documentation and Redux Style Guide

Backend

For backend development, it is mandatory to use the **NestJS** framework and create **REST** endpoints that cover all required functionality. These endpoints should mirror the tasks performed on the frontend side, such as adding, editing, and removing tasks, creating task lists, and managing task movements within the lists.

Example of endpoints for the **task** entity:

Query type	**Endpoint**	**Action**
POST	/tasks	Create a task object.
DELETE	/tasks/:id	Remove an item.
PATCH	/tasks/:id	Edit item.
GET	/tasks/:id	Retrieve an item.
GET	/tasks	Get all tasks.

References:

- Follow official NestJS documentation

Database

All data must be stored in a separate database. It is recommended to use **PostgreSQL** along with **TypeORM** (you could choose Prisma ORM or Sequelize) for database management. NestJS has profound documentation about integrating ORM and connecting Database to the application.

It is recommended to utilize a Docker image for **PostgreSQL** to simplify the setup process. Optionally, provide initial data using database seeds.

References:

- Integrate ORM and connect Database
- Install Docker
- Docker Image for Postgres

Other tools

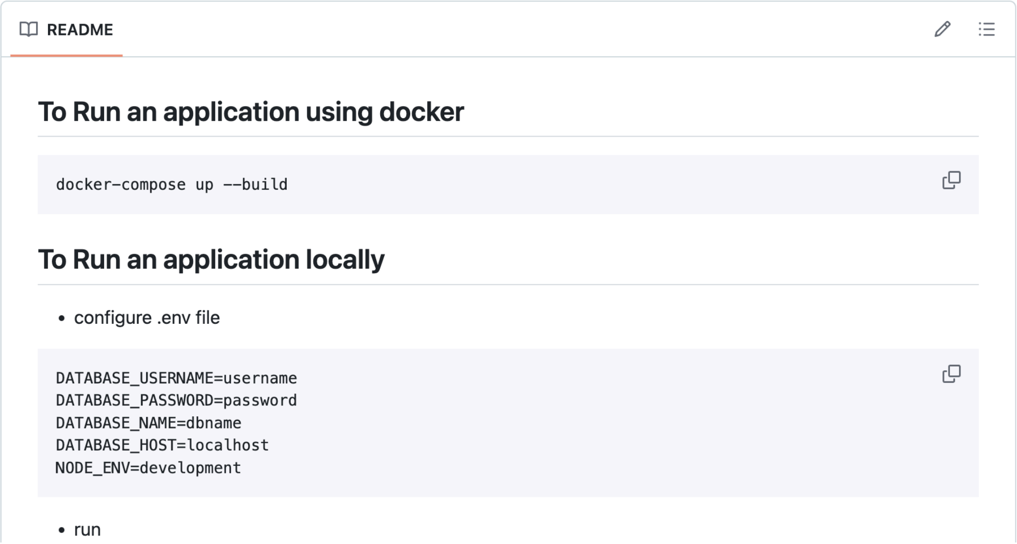
All other tools, which are not specified in the requirements, are allowed to be used, if they do not contradict the main criteria of the task. That is, React cannot be replaced by Angular, or Potsgres by MongoDB. At the same time, you can choose Vite or Webpack, use a library for working with dates, or add drag and drop.

Deployment

While deployment is not mandatory for this stage, having a deployed application is appreciated as it simplifies the evaluation process of the homework task.

Documentation

Provide a Readme file with detailed tutorial on how to set up and run a project. For example:



To Run an application using docker

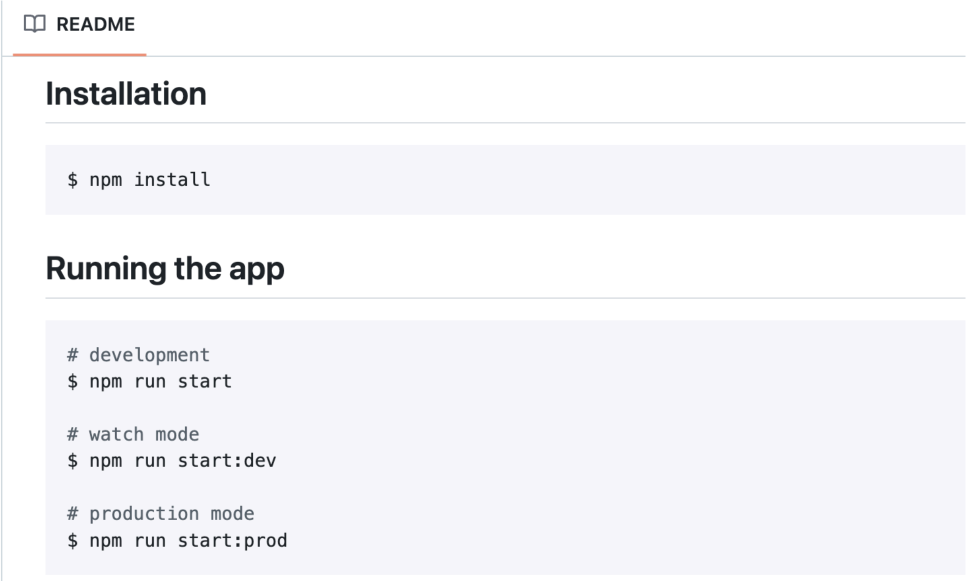
```
docker-compose up --build
```

To Run an application locally

- configure .env file

```
DATABASE_USERNAME=username
DATABASE_PASSWORD=password
DATABASE_NAME=dbname
DATABASE_HOST=localhost
NODE_ENV=development
```

- run



Installation

```
$ npm install
```

Running the app

```
# development
$ npm run start

# watch mode
$ npm run start:dev

# production mode
$ npm run start:prod
```

Try to automate the project launch as much as possible so that it can be done with a single command. Add env files or write default credentials so you can start the project faster. Add a link to the deployed project.

Code Requirements

The goal of the task is to let you get better acquainted with the modern JavaScript fullstack ecosystem, React and Redux technologies, NestJS framework and database like Postgres. If you don't know some of the APIs needed for the task, you might use the resources as references below.

While working on your task you should utilize all of the following:

1) JavaScript
   - Import / export
   - Array methods: map, reduce, filter (some of them)
   - Array spread operator
   - Regular expressions
   - Try / catch
2) React/Redux
   - Functional (a.k.a. stateless) component
   - React Hooks
   - Use Action Creators, write meaningful action names
   - Keep the immutability and readability of your reducers

   - Do not transform your store data in the components
   - Use the Redux DevTools Extension for Debugging
3) NestJS
   - Modular Architecture. Use a modular approach to structuring your application. Breaking down your code into reusable modules helps maintain a clean and organized codebase. Each module should encapsulate a specific feature or functionality.
   - Dependency Injection. DI makes it easy to manage and inject dependencies into your classes and components, Reducing tight coupling and improving code maintainability. By using decorators like @Injectable()and @Inject(), you can create loosely coupled components that are easier to test and extend.
   - Use Decorators for Metadata.
   - Middleware and Guards
   - Error Handling and Logging
4) Clean Code
   - Write small functions, pure functions.
   - Adhere to the single responsibility principle. Separate the logic of rendering and data processing, ideally to separate files.
   - Give variables and functions descriptive names.

Git and Github

Another skill you should practice is working with **Git and Github**. Implement the following git requirements while working on the task:

1. Make at least 3 commits
2. Push commits to the develop branch to your Github repository
3. When finished, create a pull request to the master branch
4. Try several git commands
   - See commit log
   - See diff between commits
   - Make some code changes and see git status
   - Perform reset --hard

References:

- https://git-scm.com/docs
- https://guides.github.com/introduction/flow/