Task By Task Guide

TASK 1 - Set up your local Docker environment

As a first step, you will need to install an infrastructure in order to test, and later deliver, your application. For this project, you're going to use NodeJS to initialize and set up a basic application. Your application will also need to include express (to start your web server), dotenv (to configure your NodeJS local port) and mongoose (for mongoDB object modeling) as dependencies. As in this project you will package your application using Docker, you will also need to install Docker and the official images for NodeJS and MongoDB. Your application will run in NodeJS, and its data is going to be saved in MongoDB. Finally, towards the end of the project, you will set up a Dockerfile and a docker-compose file in order to make your CRUD application available to others. You will find further guidance about packaging the application in Docker at Task 5 of this task by task guide, however you can perform this operation at the initial stage of this project as well and continuously test throughout the project.

Hint

If you're not familiar with this process, you can find an example guide here. This guide will instruct you to create a basic setup that includes a user model/route to manage different level of user access. User management is not part of this project scope, and you don't need to add it to your application, however if you wish to use the guide to create and test the infrastructure, you can later delete the user management component, or choose to go the extra mile and use it as a basis to add user management to your own project.

TASK 2: Connect to the database and create models

It is fundamental to carefully think about the business structure needs of your company or client, and create data models according to what the application will need to achieve. Read the business outline that has been provided to you carefully, so as to design the data models accordingly. For example, if your application must work with employees and company departments, probably you will need to create an employee MongoDB collection, and a department collection. Once you have built your database structure, you will connect the application to the database. You can use the facilities provided by mongoose both for connecting to the database, and creating models.

Hint

Consider using mongoose connect() to connect your application to the database. Explore the documentation for the MongoDB Docker image to know how to connect to it. Review how to model the data in MongoDB, considering, for example, the number of entities that are needed and what are the relationships between them (1-to-1, 1-to-many, many-to-many). Consider also the indices your queries might need in order to be faster. Think about what fields are required and which might be optional.

TASK 3: Create the required routes

Once you have created your models, it's time to think about the business logic of your application, its internal API (managing internal interactions between app components) and its exposed REST API, which communicates to the app client. What type of data that you expect as an input? Which type of data will your server, and your application's functions, return? What errors might be thrown? What are the possible vulnerabilities or security risks? While it is not expected in this project to implement solid security at production level, it is always good practice to think about these elements.

Hint

Be sure to consider all 4 CRUD operations. It's particularly useful to abstract as many layers as possible, for good code reuse. Oftentimes you may want to create separate functions for particular operations that are called by your route, and let the route take care only of the input from, and the output to, the client. For example, a route may need to return the list of employees: retrieving the employees list with a separate function, instead of querying the database directly inside the route itself, will make retrieving the employee list readily available to any other route or app component that will need it.
When you're querying the data, be mindful of its asynchronous nature, so please review the roles of async/await and how promises work in general.
Since the server is always going to receive data input from the app client, when working in production always be wary of data quality. You need to ensure the server uses clean and consistent data. Consider implementing data validation measures at back-end and front-end level. As the old saying goes: never trust the client!

TASK 4: Create the required views

At this point, you can begin thinking of your application interface. What is the workflow of your typical user going to be? How many pages need to be created? Consider the types of form elements that are needed in your application, how the application is going to update the data displayed or sent to the server, if it's going to be asynchronous or not, and how to display errors in data validation, both by the client and by the server. Be mindful of usability and accessibility guidelines, whether to integrate a template engine or not. While this project does not require full user interface (UI) research, design and development, you will implement the fundamental structure of the UI (such as forms and fields). As an extra, feel free in all cases to further develop your UI in HTML, CSS and JS for a better experience.

Hint

You can use jQuery to manipulate the DOM and communicate through Ajax, or you can use more complex Javascript frameworks such as AngularJS and ReactJS. You can also integrate template engines such as Handlebars for the HTML returned by your server.
When creating your interface and the links between the pages, always put yourself in the final user's shoes. Try to analyze the optimal workflow for your users.

TASK 5: Putting it all together

Finally, you will write the app.js file that is going to start your application. The app.js file is responsible to connect to the database and start the server. The file needs to have your routes as dependencies. It will also expose all the URLs that are going to be available to the browser. You also now want to use Docker in order to package and start your application nicely.

Hint

Consider using express' methods use(), get() and listen(). You want to write a Dockerfile that creates an image with all your application files and starts the web server exposing the port that you set up in your .dotenv file. To put everything together nicely, you can use docker compose to start and build your application, and a mongo container that communicates with it to store the application data.