Thinix Programming Challenge - Angular

This Angular Thinix Programming Challenge is designed to test the applicant's ability to develop a simple task management application with Angular. The applicant will be provided with a basic skeleton for creating the task application.

This challenge is meant to be a test of the applicant's programming ability and is designed to take an hour or two to complete the basic functionality. The use of Twitter Bootstrap 4 is encouraged and you will already find it being loaded in the skeleton code.

A realistic scenario is that one is given an API with some documentation, and is expected to make something from it. This project has mocked version of a REST API, with some usage information (the API Documentation section of this document).

You are also encouraged to make some smart design decisions; keep in mind that this is an interview, so showing best practices will be valuable.

Using online resources, such as StackOverflow and especially the Angular Docs, are allowed, but details about the programming challenge should not be posted online. Since this is a test of the applicant's own ability, collaboration with others is strictly forbidden. Since the applicant's code will be directly compared to other applicant's, it is in the applicant's best interest to avoid working with others or helping others with their code.

Lastly, no one is perfect, including the person that created this interview project. If you find any errors with the skeleton code given, you are encouraged to send us an email.

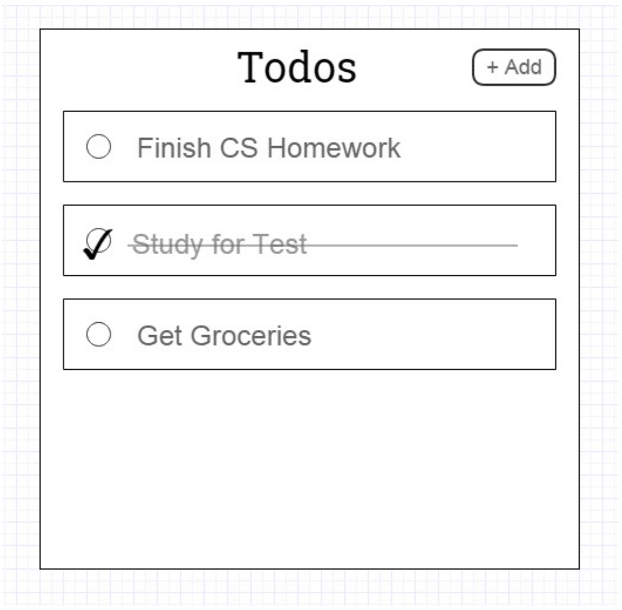
Evaluation Criteria

- The list of task items must be displayed to the user.
- It should be clear to the user if an item is marked done or not. (This is a boolean property of the task items, see Data Format documentation below).
- The user must be able to perform the following actions on task items:
    - See details about the task. This includes the title, due data and the description
    - Create a new task
    - Mark a task as complete
    - [Bonus] Delete a task
    - [Bonus] Update a task's details
    - [Bonus] Create multiple tasks at once
    - [Bonus] Delete multiple tasks at once
- [Bonus] For all of the above actions, the user should be made aware of when things succeed or fail. A simple success or failure message could do this.

** Note - Requirements marked as [Bonus] are additional requirements that would make a better project, and so we would like to see some of them attempted, although they are entirely optional.

Getting Started

Before getting started you will need to install Nodejs on your machine. Fortunately, this is fairly trivial if you have admin access. Simply go to Nodejs.org and install the recommended version. Follow the instructions on the installer and you should be good to go. To verify open a power shell and type `npm -v` this should give you a version number. If it does not, something went wrong. To get started running the project open a terminal or powershell window and change directory to this unzipped folder. You will need to install all project dependencies by running `npm install`. This could take a couple of minutes but once it is done, just run `npm start` and a browser window should open with the boilerplate programming challenge.



Data Format

A task item on the backend has the structure like the following:

```
{
  "id": 4,
  "title": "title goes here",
  "description": "An optional description may go here",
  "due": "7/20/1969",
  "done": true
}
```

The restrictions for each property is as follows:

- id - must be a number and is required for task items that have been persisted in the backend
- title - is required and must be a string
- description - is optional. If specified, it must be a string
- due - is optional If specified, it must be a string that is in a valid ISO8601 format.
- done - is a required boolean, signifying if a task has been completed.

API Documentation

The actual http backend is actually just a JSON file located in the backend folder of this project, but is hosted as the database to simulate a REST API for using Http For this reason, you must use these (or a module that in turn uses these) modules for development. By default all http methods return an observable data stream for you to subscribe to, if you more comfortable using promises, feel free to use rxjs's toPromise() function to convert the output to a promise.

The JSON Db server url is http://localhost:3000 by default.

This section of the document describes this API. For the defined routes, route parameters appear in curly braces. For example, /route/{param} suggests that {param} may be replaced by a value, e.g. /route/4. The API only responds with JSON.

Route: **GET** http://localhost:3000/tasks

Queries the API for all current task items.

Will respond with an array of task objects.

Route: **GET** http://localhost/tasks/{id}

Fetch a single task item by its id.

The response body will be the task object whose id was given in the route.

Upon failure, a 404 or 400 response may be returned.

Route: **POST** http://localhost:3000/tasks

Create a single task item.

The request body should be a valid task object. Note that for new items, the id field needs to be omitted; it is auto-generated by the backend.

The response body upon success (http status code 201) will be the object that has been persisted. It will also contain the task item's generated id.

Upon failure, a 400 Bad Request is returned, and the body will contain the errors in the request.

Route: **PUT** http://localhost:3000/tasks/{id}

Edit a single task item.

The request body must contain the full task object, this is not a PATCH request.

Upon success, a 200 OK response is returned, and the response body will contain the updated task item.

Upon failure, a 404 Not Found may be returned, or a 400 Bad Request will be returned. It will contain details about what went wrong in the response body – similar to the POST route.

Route: **DELETE** http://localhost:3000/tasks/{id}

Delete a single task item.

Upon success, a 204 http status is returned, signifying that content was deleted.

Upon failure, a 404 Not Found error is returned.

Retreiving and manipulating data

In Angular, all data is passed around via a service. For this challenge we would like you to create a service that handles all http requests to the mocked RESTapi. We have created an interface for you to implement with example method declarations. This is located in the services folder called ITaskService

Submission Instructions

The folder containing the solution and projects should be renamed "ThinixChallenge_[FirstName][LastName]" (example: "ThinixChallenge_JohnDoe"). The solution itself and projects should not be renamed.

Please remove the node_modules folder to keep the payload size low.

Zip the entire solution folder, upload to Dropbox or Google Drive and send the download link to Chris Van Oort (cvanoort@thinix.com) and Ross Reicks (rreicks@thinix.com).