/PRO

Cumulative Project: Boss Machine

Cumulative Project: Boss Machine

In this project, you will create an entire API to serve information to a Boss Machine, a unique management application for today's most accomplished (evil) entrepreneurs.

Boss Machine

Project Overview

In this project, you will create an entire API to serve information to a Boss Machine, a unique management application for today's most accomplished (evil) entrepreneurs. You will create routes to manage your 'minions', your brilliant 'million dollar ideas', and to handle all the annoying meetings that keep getting added to your busy schedule.

You can view a video demonstration of the final app here:

Next

Get Help

0:00 / 2:23

How to Begin

To start, download the starting code for this project here. After downloading the zip folder, double click it to uncompress it and access the contents.

Once you have the project downloaded, you'll need to run some terminal commands to get the application started. First, open the root project directory in your terminal. Run `npm install` to install the dependencies of this project and build the front-end application. Once it has finished installing, you can run `npm run start` to begin your server. You'll see `Server listening on port 4001` in the terminal. The `npm run start` script will automatically restart your server whenever you make changes to the **server.js** file or **server/** folder. If you want to turn this off, simply start your server with the `node server.js` command. You can kill either process with the `Ctrl + C` command.

Next

Get Help

To see the application in its initial, non-working state, simply open **index.html** in a web browser. You should use Google Chrome (at least version 60) or Firefox (at least version 55). The links above will let you download the latest release of either browser if you do not have it or are unsure of which version you're running.

Implementation Details

To complete the project, you will need to complete code in a few sections of the project. Generally, you will not have to touch anything inside the **browser**, **public**, or **node_modules** folders unless you know some React and HTML/CSS and want to customize the look of the Boss Machine. Before doing any of that, however, let's focus on getting the API server up and running:

Server Boilerplate

In **server.js**, you will see some boilerplate code, but the server is missing key functionality to allow it to run. You must:

- Set up body-parsing middleware with the `body-parser` packagae.
- Set up CORS middleware with the `cors` package. You can use the default settings.
- Mount the existing `apiRouter` at `/api`. This router will serve as the starting point for all your API routes.
- Start the server listening on the provided `PORT`. Make sure to use the `PORT` constant and not a hard-coded number, as this is required for tests to run.

Take note of the comments in **server.js**, as your code needs to fit into specific places around the existing boilerplate.

API Routes

Next Get Help

intended, not how you nest your code!

- Your ‘database’ exists in **server/db.js**. The beginning database will be seeded every time the server is restarted. There is more information on working with the database and the helper functions it exports below.

Routes Required

- `/api/minions`
  - GET /api/minions to get an array of all minions.
  - POST /api/minions to create a new minion and save it to the database.
  - GET /api/minions/:minionId to get a single minion by id.
  - PUT /api/minions/:minionId to update a single minion by id.
  - DELETE /api/minions/:minionId to delete a single minion by id.
- `/api/ideas`
  - GET /api/ideas to get an array of all ideas.
  - POST /api/ideas to create a new idea and save it to the database.
  - GET /api/ideas/:ideaId to get a single idea by id.
  - PUT /api/ideas/:ideaId to update a single idea by id.
  - DELETE /api/ideas/:ideaId to delete a single idea by id.
- `/api/meetings`
  - GET /api/meetings to get an array of all meetings.
  - POST /api/meetings to create a new meeting and save it to the database.
  - DELETE /api/meetings to delete *all* meetings from the database.

Next Get Help

property, you will have to set it based on the next id in sequence.

For `/api/meetings` POST route, no request body is necessary, as meetings are generated automatically by the server upon request. Use the provided `createMeeting` function exported from **db.js** to create a new meeting object.

Working with the 'Database'

The **server/db.js** file exports helper functions for working with the database arrays. The goal of this project is for you to focus on Express routes and not worry about how the database works under the hood. These functions always take at least one argument, and the first argument is always a string representing the name of the database model: `'minions'`, `'ideas'`, `'meetings'`, or `'work'`.

`getAllFromDatabase`:

- Takes only the single argument for model name. Returns the array of elements in the database or `null` if an invalid argument is supplied

`getFromDatabaseById`:

- Takes the model name argument and a second string argument representing the unique ID of the element. Returns the instance with valid inputs and `null` with an invalid id.

`addToDatabase`:

- Takes the model name argument and a second argument which is an object with the key-value pairs of the new instance. `addToDatabase` handles assigning `.id` properties to the instances. It does not check to make sure that valid inputs are supplied, so you will have to add those checks to your routes if necessary. `addToDatabase` will return the newly-created instance from the database. This function will validate the schema of the instance to create and throw an error if it

Next Get Help

`updateInstanceInDatabase` :

- Takes the model name argument and a second argument which is an object representing an updated instance. The instance provided must have a valid `.id` property which will be used to match. `updateInstanceInDatabase` will return the updated instance in the database or `null` with invalid inputs. This function will validate the schema of the updated instance and throw an error if it is invalid.

`deleteFromDatabasebyId` :

- Takes the model name argument and a second string argument representing the unique ID of the element to delete. Returns `true` if the delete occurs properly and `false` if the element is not found.

`deleteAllFromDatabase` :

- Takes only the single argument for model name. Deletes all elements from the proper model and returns a new, empty array. You will only need to use this function for a /api/meetings route.

Schemas

- Minion:
  - id: string
  - name: string
  - title: string
  - salary: number
- Idea
  - id: string
  - name: string

Next Get Help

- numWeeks: number
- weeklyRevenue: number

- Meeting
  - time: string
  - date: JS `Date` object
  - day: string
  - note: string

Take note that many values that could be numbers are in fact strings. Since we are writing an API, we can't trust that data is always provided by a client. You may need to transform between String and Number JavaScript types in order to provide full functionality in your API.

Custom Middleware

- You will create a custom middleware function `checkMillionDollarIdea` that will come in handy in some /api/ideas routes. Write this function in the **server/checkMillionDollarIdea.js** file. This function will make sure that any new or updated ideas are still worth at least one million dollars! The total value of an idea is the product of its `numWeeks` and `weeklyRevenue` properties.

Bonus

As a bonus, you may implement routes to allow bosses to add and remove work from their minions' backlogs.

Schema:

- Work:

Next Get Help

- description: string
- hours: number
- minionId: string

Routes required:

- GET /api/minions/:minionId/work to get an array of all work for the specified minon.
- POST /api/minions/:minionId/work to create a new work object and save it to the database.
- PUT /api/minions/:minionId/work/:workId to update a single work by id.
- DELETE /api/minions/:minionId/work/:workId to delete a single work by id.

To work on the bonus with tests, you will need to remove their pending status. Open the **test/test.js** and edit that begins the /api/minions/:minionId/work routes tests. It should start with `xdescribe(` around line 689 of the test file. If you delete the `x` (so that the line simply starts with `describe(` and save the test file before re-running, your bonus tests will now be active.

In order to fully implement these routes, the database helper functions may not provide all the functionality that you need, and you may need to use router parameters or other methods to attach the `minionId` properties correctly and handle the edge cases property. Good luck!

Testing

A testing suite has been provided for you, checking for all essential functionality and edge cases.

To run these tests, first open the root project directory in your terminal. Then run `npm install` to install all necessary testing dependencies (you will only need to do this step

Next Get Help

Finally, run `npm run test`. You will see a list of tests that ran with information about whether or not each test passed. After this list, you will see more specific output about why each failing test failed. While they are open in a terminal window, these tests will re-run every time you save server files. If you want to quit the testing loop, use `Ctrl + C`. If you only want to run the tests once, you can run the `mocha` command in the terminal from your project root directory.

As you implement functionality, run the tests to ensure you are implementing your routes and middleware correctly. The tests will additionally help you identify edge cases that you may not have anticipated when first writing your routes. You should also test the functionality on the frontend to make sure that things are working as intended. Feel free to add logging middleware to your server, it will help with debugging!

Next Get Help