

Project Management System

SOFT355 | Aden Webb | 10574860

<https://github.com/adenwebb21/ProjectManagement-SOFT355>

Running instructions

1. db_insert_tasks.js will just insert a set of predefined data into the database, not necessary for most use cases
2. Run the server using express-tasks.js
3. Load the HTML file called tasks-test.html

Functionality	3
What Does it Do	3
Interaction	4
Technologies	4
Requirements	5
Target User	5
Feature List	5
Design	6
System Architecture	6
Data & Code Structure	7
Testing	8
Unit Testing	8
Usability Testing	9
DevOps Pipeline	9
Development Environment	9
Continuous Integration Pipeline	9
Personal Reflection	9

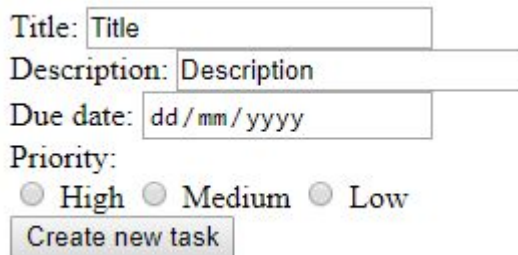
Functionality

What Does it Do

I am producing an application which will help users organise tasks and keep track of what it is they should be working on within a given project. It will assist with managing the Kanban style of workflow, with tasks moving through lanes which are commonly labelled “To-Do”, “Doing” and “Done”.

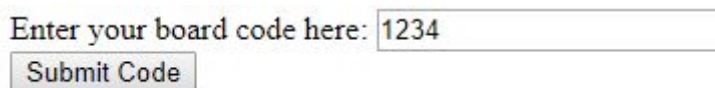
To-Do	Doing	Done
<div><div>ID: 0</div><div>Title: Task 0</div><div>Description: This is the description for task 0</div><div>Priority: 5</div><div>Due Date: 2020-01-12T00:00:00.000Z</div><div>Remove</div><div>Move Right</div><div>Move Left</div></div>	<div><div>ID: 4</div><div>Title: Task 4</div><div>Description: This is the description for task 4</div><div>Priority: 5</div><div>Due Date: 2020-01-12T00:00:00.000Z</div><div>Remove</div><div>Move Right</div><div>Move Left</div></div>	<div><div>ID: 8</div><div>Title: Task 8</div><div>Description: This is the description for task 8</div><div>Priority: 5</div><div>Due Date: 2020-01-12T00:00:00.000Z</div><div>Remove</div><div>Move Right</div><div>Move Left</div></div>
<div><div>ID: 1</div><div>Title: Task 1</div><div>Description: This is the description for task 1</div><div>Priority: 5</div><div>Due Date: 2020-01-12T00:00:00.000Z</div><div>Remove</div><div>Move Right</div><div>Move Left</div></div>	<div><div>ID: 7</div><div>Title: Task 7</div><div>Description: This is the description for task 7</div><div>Priority: 5</div><div>Due Date: 2020-01-12T00:00:00.000Z</div><div>Remove</div><div>Move Right</div><div>Move Left</div></div>	<div><div>ID: 9</div><div>Title: Task 9</div><div>Description: This is the description for task 9</div><div>Priority: 5</div><div>Due Date: 2020-01-12T00:00:00.000Z</div><div>Remove</div><div>Move Right</div><div>Move Left</div></div>

The user will be able to access a board - this board contains a number of columns, either predefined or created by the user. Each column has a number of tasks contained within it which can be created by the user and then moved around the board as required. The tasks can be created with a title, a description, a due date and a priority.



A form for creating a new task. It includes four input fields: 'Title' with placeholder text 'Title', 'Description' with placeholder text 'Description', 'Due date' with placeholder text 'dd/mm/yyyy', and 'Priority' with three radio button options: 'High', 'Medium', and 'Low'. Below the inputs is a button labeled 'Create new task'.

The method for accessing a board is shown below - the user needs to enter a pass code associated with the required board.



A form for accessing a board. It includes a text input field with placeholder text 'Enter your board code here:' and the value '1234'. Below the input field is a button labeled 'Submit Code'.

Interaction

The interaction for this tool will happen with the mouse and keyboard. The keyboard will be used for entering data into the tasks and also entering the code which will allow access to the board. The mouse will be used for selecting the various buttons used to interact with the tasks. For example, there will be buttons for removing tasks and moving tasks to the left or to the right.

Technologies

The visuals of the project are done using HTML and CSS to produce a web page capable of presenting all relevant data to the user. The HTML is what provides all the structure and the CSS makes the content a little easier to decipher at a glance. The visuals on the screen are updated using an associated JQuery file. This is in charge of taking inputs from the user on the HTML layout and translating these into commands for the database. The JQuery file is using Express to call a locally hosted server and retrieve data objects from the database.

The server is run via Node, locally, which serves as the base for Express to contact the database. The database itself is using MongoDB with the Mongoose library for easier management of the data.

I am using Mocha with Chai to write the unit tests.

Requirements

Target User

This application is aimed at anyone who has a small contained project that they would like to get organised. This would work well for hobbyist solo developers or students who need to keep track of a variety of different small tasks.

Feature List

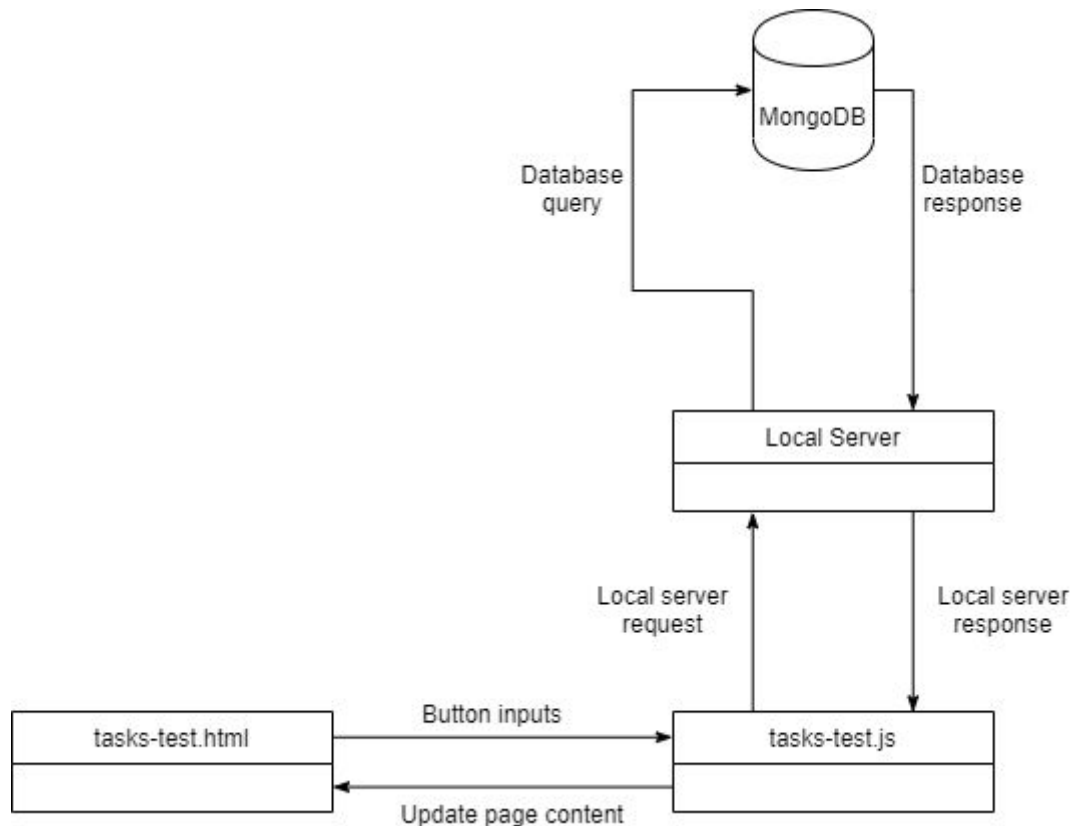
After examining similar tools, most versions of this type of system are fairly similar bar one or two features. I wanted to make a version which would be simple to use for everyone. Having the boards not just be locally stored on the machine was a big feature that needed to be present. For students it is especially uncommon to work at the same machine every time - your tasks need to be accessible wherever you are.

With a tool which is so suited to working around the users own project, I needed to provide users with enough inputs to be able to make the tasks meaningful to them. Title and description were fairly obvious, as was a due date. I also added a priority option to help users more easily see which tasks need to be done sooner.

Being able to move these tasks around emulates the standard kanban agile development model. You would make a task, add it to the to-do list, then progress it through the columns based on its current status. This meant that two more features need to be moving the tasks left and right.

Design

System Architecture



The client consists of the requested visual information. This is the columns currently within the board and the tasks within each column. The client does not really ever hold any pertinent information - merely retrieving it as necessary and using this to influence the visuals displayed.

When a button is pressed on the web page which requires the retrieval of information, an express request is made to the Node server which then uses mongoose to access the database and return the requested data in the response. This response can then be used to update the webpage.

In some cases data needs to not only be retrieved from the database, but also sent. In these cases the same procedure is used but the data to be sent is passed using parameters within the url used to access the local server. Express can then extract these elements and do with them what is necessary. An example of this would be creating a new task. The user has to enter their own details, so these are sent to the server via parameters. Once the express request has been fulfilled, the new task is then sent back to the client and used to update the new task visually.

Data & Code Structure

How are the data and code structured?

Why are these structures appropriate?

On the database the data is structured into 3 collections: Boards, Columns and Tasks. Examples of these, as well as their properties, are given below.

Board

```
_id: ObjectId("5e160c69bf85a856f825d9dc")
columns: Array
  0: 0
  1: 1
  2: 2
id: 0
title: "Board 1"
code: 1234
__v: 0
```

The board has an id for identification, a title, a passcode for accessing and an array of numbers which are the ids of the columns currently in this board. This makes sense in terms of object relations because the relationship here is aggregation - and the database tables needed to reflect that. The board contains columns, but does not require them whereas the columns cannot exist without a board.

Column

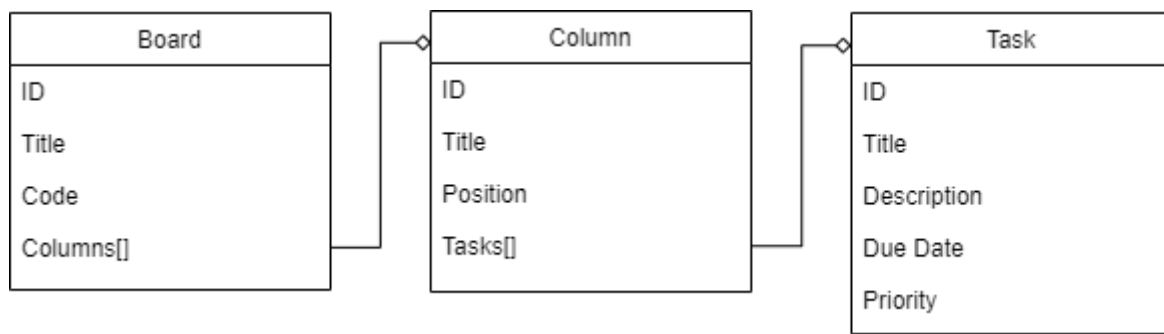
```
_id: ObjectId("5e160c69bf85a856f825d9db")
tasks: Array
  0: 8
  1: 9
id: 2
title: "Done"
position: 2
__v: 0
```

The column has an id for identification, a title to appear at the top of the column, a position on the board and an array of task indexes. Similar to the board this is correct to the actual relationship of these items. The columns don't need tasks to exist, but they need to know about the tasks within themselves. The tasks do not need to know what column they are in.

Task

```
_id: ObjectId("5e160c69bf85a856f825d9cf")
id: 0
title: "Task 0"
desc: "This is the description for task 0"
due: 2020-01-12T00:00:00.000+00:00
priority: 5
__v: 0
```

The task has an id for identification, a title to display at the top, a description to provide the user more information, a due date and a priority. The task contains no arrays since there are no smaller items than a task. It also does not need to know about the board or the columns since they already know about what is one level below themselves. This information is purely descriptive.



Testing

Unit Testing

I have developed a series of unit tests to check whether or not my server paths are returning what I expect them to. I am using Mocha with Chai for this purpose, with the assert library. A lot of my checks are related to whether or not the returned page status is correct, as well as if the response from the server contains the properties I am expecting. For example - checking for a task which does not exist should return a 404 code. An example of checking the properties of a certain response is where I check what retrieving the first task in the list returns. Since I do not necessarily know exactly what the content will be, I am checking for the data type of each parameter.

This method of unit testing is appropriate as it encourages a test driven development style, it makes sense for me to think about what I need to happen in the project, write a test to see if works and then implement the actual functionality. Doing this keeps the goal in mind at all times - the simple short term objective for this piece of code, right now. Testing returns from databases can be a little awkward when it comes to not wanting to add new entries all the time, but the method I am using (just checking the

data types of the response) means that whatever is returned will work, without needing to get exact values.

Usability Testing

For usability testing, I mainly used verbal feedback from my peers to see what they thought about the various decisions I was making about the tool - since they are part of the target audience I was aiming for with the project. The general consensus with this feedback was that it was quite a simple and limited tool, but one that felt as though it's functionality could be expanded on easily and serves as a good base product for now. This is a fairly minimal method of usability testing, but I feel it is appropriate for what I am making. It is the little tweaks and off hand mentions that wouldn't be caught by a unit test but would be noticed by someone actually using the product.

Integration Testing

I used ESLint to undertake continual checks on the javascript I was writing. This allowed me to identify any major problems in my code when certain functionalities were combined together.

DevOps Pipeline

Development Environment

The development environment I am using involves Google Chrome for loading the HTML pages, Atom (by GitHub) for code editing, GitHub desktop for managing my repository and finally the MongoDB website for accessing my database for diagnostics and connection settings.

Continuous Integration Pipeline

The CI pipeline for this project is not as in depth as it might be if it was for a team project, but I have tried to keep my commits regular for the duration of the development period and also try to commit cleanly around feature implementations. This allows me to easily roll back features should it be required. The ESLint tool also helped with the pipeline as I could run it after any major integration and see if anything significant had broken.

Personal Reflection

Overall, I am happy with the outcome of this project. It was my first real look into distributed web systems and it definitely took me a while to get my head around all the different technologies and their purposes. But, despite that, I still managed to work

through and make this project which I am quite proud of. I felt that all the technologies I used were good fits for the position I was implementing them in and I had no real issues with any of them not fulfilling the role they were supposed to.

If I were to do the project again, I think I would like to have taken a little bit more time to work out how the systems worked initially - allowing me to get into the meat of the project a little faster. I know that my file organisation is not the best and abstracting this out into proper files would be a good place to start. For example I should keep the schemas in a separate file rather than in the server file.

In general I was happy with the layout of my database tables, but one change I would like to make in the future is to make the arrays of items arrays of the actual objects instead of the id. This would have saved me a lot of time if I had done this from the beginning and as it stands I had to hack my way around it multiple times.