2020

CAB230 Stocks API – Server Side

CAB230

Stocks API – Server Side Application

<student name/s>

<student number/s>

4/23/2020

Contents

This template is adapted from one created for a more elaborate application. The original author spends most of his professional life talking to clients and producing architecture and services reports. You may find this a bit more elaborate than you are used to, but it is there to help you get a better mark

This report will probably be around 5 pages or so including screenshots

Introduction

Purpose & description

This is written in a high-level professional tone. Tell is in about a paragraph or so the overall function of the API. Yes, you need to do this, and you need to do it without listing the endpoints in detail. That will come below. This should be in ***your*** words, though you should feel absolutely free to steal some of it from the assignment specification and adapt it.

Unlike the client side, we don't really need screenshots, but it might help to capture some results from one of the Swagger trial queries.

Completeness and Limitations

Here we want you to tell us in a couple of sentences what works and what doesn't. ***Make a claim against the standards we laid out in the assignment specification (see below) and briefly justify that claim.*** Don't give us deep details of the bugs here. Putting a positive spin on what you have achieved is fine – by all means focus on the stuff that works. But be realistic in your claim. As for the client side, the text below is stolen straight from the assignment specification:

- **[Grade of 4 level]:** Successful deployment of an Express based API which supports ***some*** of the endpoints and interacts successfully with the database. Most likely people will complete the basic *Queries* routes, but may not have proper filtering or completed *Users* routes, or have managed the authenticated query route. Swagger docs may not be deployed, and there may be significant gaps in the security requirements.
- **[Grade of 5 level]:** Successful implementation of ***all*** of the query endpoints at a basic level. Time-based filtering should basically work, even if the route is not properly authorised. Registration and login and JWT token handling must be attempted, though there may be issues with the authentication. At the grade of 4 or 5 level there may be a number of incorrect responses and codes even if the basic application works.
- **[Grade of 6 or 7 level]:** The grade of 6 and 7 levels require successful completion of ***all of the routes***. The distinction between 6 and 7 level grades for functionality then relates to problems in the valid and error responses on the routes, and in the successful use of middleware security, database connectivity and deployment of the Swagger docs. There is no 'killer' requirement here that makes the difference between a 6 and a 7. These requirements are best seen as a set that together, and done very well, give you a 7 standard mark. If you miss some of them, but still do a good job of the others, then you are likely to get a 6 standard mark.

You may find it helpful to work with the list of endpoints below, telling us of any limitations. If the endpoint is fully functional, just say 'fully functional' after the route:

/stocks/symbols
/stocks/{symbol}
/stocks/authed/{symbol}
/user/register
/user/login

Modules used

This is just a list of the external modules that you have used. You need not specify anything found in the server-side pracs or the JWT server-side worksheet. For many people, the simple statement:

No additional modules used

will be sufficient. If you did use others then in each case, ***we want the name, a brief description, and a link to the docs at npm or github or wherever.*** Please follow the style of the ag-grid-react example below, though we will be rather surprised if this was used in Assignment 2:

Ag-grid-react

Module to provide fully-featured table components, including sorting and filtering.

https://www.ag-grid.com/react-grid/

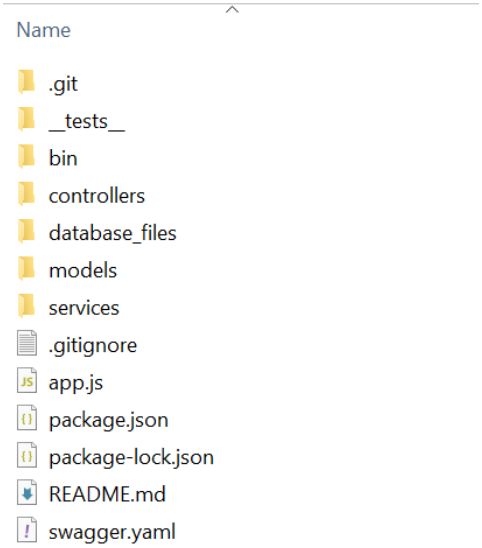
Module 1

Module n

Technical Description

Architecture

Briefly describe the overall architecture of your application at a source code level. We expect that the application will closely follow the model imposed by Express-generator, but there are many aspects of this application on which the generator is silent, and we want to know more about your choices. Most of this text should talk about the organization of the routing and the authentication and database connection services. You should use one or more screenshots of your directory structure. Here is one of ours, and yours may be markedly different and still fine:



But you should dig down into some of the subdirectories and tell us how you have organized things. This helps to convince us that you know what you are doing and makes it easier for us to navigate your work when we come to inspect the code.

Overall, including the images, this section will be be under a page in length. We should get some sense of how the application works and how the data flows around. You may also find it helpful to show us screen grabs of code if that makes your points clearer. Tell us anything you think we need to know about how you have structured the application and made it work, but there also a section below to describe problems.

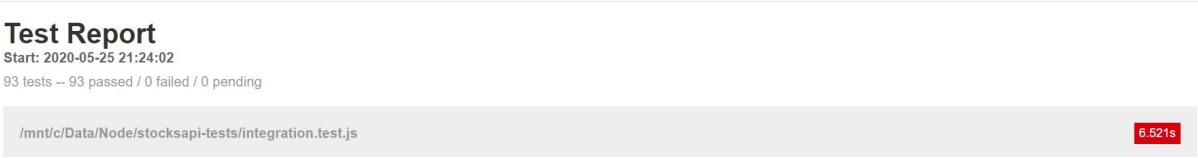
Security

In this section we expect a brief description of the security features of your application. This is essentially a bulleted checklist based on the specification. We do not need a full description of helmet or any of the other standard approaches. The list from the specification is given below. You should indicate whether or not your application includes these features and give extra details where necessary – such as additional helmet settings and any additional details of the password handling.

- Use of `knex` or other query builder without raw SQL
- Use of `helmet` with at least the default settings enabled
- Use of `morgan` with a logging level similar to that used in the pracs.
- Appropriate handling of user passwords as described in the JWT Server-Side worksheet.

Testing

In this section we expect you to screenshot the html test report:



If all of the tests pass, this is sufficient. If there are still failures, then please sceenshot the red areas and include them below. You should include your full test report at the time of submission, and we will of course have our own to look at. Please do not try to make this look better than it is. We have the tests to run ourselves.

Difficulties / Exclusions / unresolved & persistent errors /

Topics to include here could be:

- What were your major roadblocks / how did you resolve them?
- Any functionality you didn't or couldn't finish and the technical issues encountered
- Are there any outstanding bugs?

If you are describing bugs or unresolved issues please be quite specific and feel free to show screen grabs of code which will assist us in assessing your work.

Extensions (Optional)

Where could you take this:

tell us about Potential future extensions / improvements for the API

Installation guide

Tell us how to install the application – this should be at the same sort of level as most github sites.

Use screenshots as needed.

References

Use a standard approach to referencing – see the guidance at https://www.citewrite.qut.edu.au/cite/.

These references are not for the node modules but rather refer to any blogs or tutorials or whatever else you have used in the introduction or the Technical Description of the application.

Appendices as you require them

Anything you think should be included but is better left to the end.