Part 4 & Part 5

Introduction

The blog list is deployed to:

https://guarded-anchorage-04684.herokuapp.com/

Username: RenSherbourne

Password: secret

Self Evaluation:	Mastery / Progressing / Needs Improvement
Due date:	2021-10-01
Completion:	2021-09-30
GitHub:	https://github.com/ZDGharst/FullStackOpen

Part 4 – Testing Express servers, user administration

In order to start this part about testing, I have to modularize the application into smaller units that are then testable. I covered project structure, separate of concerns, and best practices within Node.js. Within *Part4a* I used jest to test JavaScript applications – Jest is developed by Facebook, and it's great for testing Node and React projects. It provides the features you'd expect from a modern testing library: suites, parallelism, mocking, etc. Within this section, I create a few helper functions (most liked authors, most prolific author, total likes, etc.) for a bloglist app and write tests for them.

Part4b: In order to test the database operations, I set up the test environment variable to signify which environment the program is running in. If this is the test environment, all logged is disabled and the database config (URL, here) changes. A library called supertest is used to expedite testing the express application. supertest allows for the app module to be wrapped into a superagent and tests ran against it without it actually running as a server on a port. I initialize the database for each test, clear it out for each test, and also close the connection after all tests. I start this unit with my program using the promise and chaining syntax; however, once tests are written in this part and pass, I rewrite the backend to use the async and await syntax using good practice from tests to control for output. A lot of tests are written here in the material and the exercises, and they're refactored multiple times as well.

User administration is the focus of *Part4c*. I set up schema for creation of users for apps. The course teaches me about how to reference documents in one collection in another collection – within a relational database, this would be using a foreign key in a certain place depending on the relationship type, then using join queries once the schema is set up. However, when using MongoDB, common convention with relational databases isn't always the best idea. A reference may be placed in either document or even both. You may also place the entire document rather than a reference into the referring document (example: store notes in their entirely within a user) and populate (join for relational databases) from there. I also hashed and salted the passwords using the Node.js implementation of bcrypt.

After users are created in *Part4c* and their passwords hashed, login needs to be handled. Best practices today dictate that a token is created on a successful login – then, all operations that require login request the user's token and verify it. The only way to generate a token is to have the same string the server uses when it encrypts the token. Later, when it comes to the frontend side, the token can be

saved within LocalStorage or a cookie. I also handled errors and security issues with tokens in this issue, and I implemented it within the bloglist app.

Additional notes can be found in the Part 4 README.

Part 5 – Testing React apps

Within *Part 4* I created users, token handling, and authentication. Within *Part5a* I learned how to incorporate the backend authentication into the frontend such that a user can have their own data to operate on. I saved the token to user's local storage and retrieve it from there when the application loads. The frontend service handler has to check the user's token for authenticity, and alert the user for errors.

Now that I have a better handle on working in JavaScript, *Part5b* gives a bit more detail in best practices. I learned how to create child elements of components and render them within the component. I use references between components to call functions and states. Lastly, I also learned how to enforce props to be required and/or of a certain data type. The exercises purposefully have me modularizing components, then using references to use functions/data between them.

While Part4 was about testing Node.js apps, *Part5c* is all about testing React apps. First, components have to be rendered in order to test. There are tool chains here to make this simple. You may also choose to run tests using snapshots, which records a component render as an expected value, then runs it again in the future to check if it has changed. I simulate button clicks and form changes for the tests. I also search components according to content. Lastly, the course instructor explains that integration tests are difficult with React due to mocking the backend – instead, I will be testing the endpoints in the next section.

I learn in *Part4d* how to do E2E (end to end) testing. The library used for this is Cypress. Cypress opens the webpage, and then you can search for content on the page, sort of like a web scraper. I also write to a form, submit it, then check the new data and the database to see if the note is added (or, conversely, if there is an intentional error and it doesn't post as appropriate). I also create a router only used in testing to control the database.

Additional notes can be found in the Part 5 README.

Deliverables

Exercise	Commit / Link	Diff	Notes
4.1	f4f539b	diff	
4.2	432534f	diff	
4.3	7d6042a	diff	
4.4	678bafa	diff	
4.5*	4c2cefe	diff	
4.6*	0611825	diff	
4.7*	102d77c	diff	screenshot
4.8	1ebe407	diff	
4.9*	7a205f0	diff	
4.10	5d6e8b2	diff	
4.11*	254647d	diff	
4.12*	84feb70	diff	
4.13	7c9a891	diff	
4.14	f4c08cf	diff	screenshot
4.15	f36434a	diff	
4.16*	7a187f1	diff	
4.17	6a0e25c	diff	
4.18	96069b9	diff	
4.19	077e12b	diff	
4.20*	fbfe249	diff	
4.21*	d0bfe13	diff	
4.22*	b70a730	diff	
4.23*	2049e05	diff	screenshot
5.1	bed07d9	diff	
5.2	cc86697	diff	
5.3	b33b7a9	diff	
5.4*	935dcbd	diff	
5.5	2463b92	diff	
5.6	1600f8e	diff	
5.7*	a170b75	diff	

5.8*	2d8c49c	diff	
5.9*	87e703c	diff	
5.10*	8d75279	diff	
5.11	cc9547a	diff	
5.12	2d333dd	diff	
5.13	83f9ca0	diff	
5.14*	9767929	diff	
5.15*	3790d4b	diff	
5.16*	a1b9905	diff	
5.17	3a67bd6	diff	
5.18	8a54951	diff	
5.19	185abeb	diff	
5.20	86de00b	diff	
5.21	515ff75	diff	
5.22	fe1b412	diff	

Supplementary Material

No supplementary material within these modules was used.