Welcome to Module 5 of the IOD Software Engineering course!

This is a handy guide to help students make sure their lab work is easily understood and completed. Make sure to ask your trainers if you have any questions about how to complete these exercises.

Trainers will release answers once everyone has had a chance to complete them, but often there are many possible answers so don’t worry if yours are different. The extensions are there for students who have extra time after completing the main exercises - they are not required.

Create a new folder for **Module 5** that you can include in a GitHub repository. You may wish to re-use the same private repo from previous modules and include a new folder in there, or create a separate private repo. Exercises 2, 3, 5, 6, 7, 8 all build on each other so you can reuse the same code in the same folder, adding in the new features for each exercise. Exercises 1 and 4, and the second part of Exercise 8, should go in separate folders. Ensure your trainers have access.

| **Lab Exercise 1:** | **Slide 18** |
| --- | --- |

**Instructions**: Follow the instructions on slides 11-17 to create a basic Express app, and extend it to create multiple web servers (at least 2) running on different ports.

**Goal**: To learn the basics of creating and starting a backend application using Express.

**Submission**: Create your Express app in a folder for Exercise 1 and submit it into your GitHub lab exercises repository once changes are complete.

**Extension**: Read through the links below to understand more about creating back-ends with Express.

**Resources & Extra Learning**:

* <https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server> .
* <https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction>
* <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction>

| **Lab Exercise 2:** | **Slide 36** |
| --- | --- |

**Instructions**: Create a new Express app to act as a server-side calculator with multiple routes, by re-following the setup instructions on slides 11-17 and adding custom routes as demonstrated on slides 25, 26 and 32. The routes should implement the remaining core mathematical operations (subtract, multiply, divide) based on the add example given in the slides.

**Goal**: To learn how to set up multiple different routes in a back-end Express server.

**Submission**: Create a new **calculator-app** folder in your **Module 5** folder/repo. Your Express app should be created here including all necessary sub-folders. Commit your changes when done.

**Extension**: Add another **extraRoutes.js** file in the **routes** folder and map it in index.js using a different prefix (see slide 32). Include a route that will accept a query parameter called **max** and return a random number between 0 and the given value. Extend it to take another query parameter called **min** and return a random number between min and max.

**Resources & Extra Learning**:

* <https://expressjs.com/en/starter/basic-routing.html>
* <https://expressjs.com/en/guide/routing.html>
* <https://medium.com/@JavaScript-World/handling-routes-with-express-js-a-comprehensive-guide-for-traffic-management-f54f16256525>
* <https://medium.com/nerd-for-tech/understanding-routing-in-express-js-as-simple-as-possible-45b4a852c4a0>

| **Lab Exercise 3:** | **Slide 37** |
| --- | --- |

**Instructions**: Use the same calculator app as in Exercise 2, and create a basic front-end UI for your server-side calculator which can support all 4 mathematical options, using the code on pages 34 & 35 to kickstart your exercise.

**Goal**: To learn how to connect back-end API routes to a front-end interface.

**Submission**: Commit your changes to the same **calculator-app** folder you used for Exercise 2.

**Extension**: Try to split out your JS into a separate file in public/js and include it via a script tag. Review it to reduce any duplication and improve efficiency, and try to use the **axios** library instead of **fetch**. Also create an external CSS file to style your calculator.

**Resources & Extra Learning**:

* <https://apidog.com/blog/nodejs-express-get-query-params/>
* <https://axios-http.com/docs/intro>
* <https://www.freecodecamp.org/news/make-api-calls-in-javascript/>

| **Lab Exercise 4:** | **Slide 44** |
| --- | --- |

**Instructions**: Copy the Express App template from **Exercise4/m5lab4\_expressapp** from Google Drive into your existing GitHub repo for Module 5. Make your changes to that code to complete the instructions on the slide (these are also in the comments).

**Goal**: To learn how to handle different types of data available in the request for each route.

**Submission**: Commit your changes to the **m5lab4\_expressapp** folder in your existing GitHub repo for Module 5.

**Extension**: Add another route to handle deleting a friend with a given ID. Create a new array of objects similar to friends but for movies, and a new set of routes to handle creating. reading, updating and deleting movies from this list.

**Resources & Extra Learning**:

* <https://flaviocopes.com/express-request-parameters/>
* <https://masteringjs.io/tutorials/express/req>
* <https://www.digitalocean.com/community/tutorials/use-expressjs-to-get-url-and-post-parameters>

| **Lab Exercise 5:** | **Slide 49** |
| --- | --- |

**Instructions**: Use the same calculator app as in Exercise 2 & 3, and move all logic from the routes file into the controller file, following the example on slide 47.

**Goal**: To learn how to implement and follow an MVC-style framework in a back-end application.

**Submission**: Commit your changes to the same **calculator-app** folder you used for Exercise 2 & 3.

**Extension**: Create a separate controller file for the routes in Exercise 4 and move logic there.

**Resources & Extra Learning**:

* <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes>
* <https://dev.to/ericchapman/nodejs-express-part-5-routes-and-controllers-55d3>

| **Lab Exercise 6:** | **Slide 63** |
| --- | --- |

**Instructions**: Use the same calculator app as in Exercise 2, 3 & 5, and add automated unit testing for all routes by following the example on slide 59.

**Goal**: To learn the basics of unit testing using Jest and how to implement it for an Express app using Supertest.

**Submission**: Commit your changes to the same **calculator-app** folder you used for Exercise 2, 3 & 5.

**Extension**: Create a separate suite of unit tests for the routes in Exercise 4.

**Resources & Extra Learning**:

* <https://jestjs.io/docs/getting-started>
* <https://jestjs.io/docs/using-matchers>
* <https://www.testim.io/blog/api-testing/>

| **Lab Exercise 7:** | **Slide 75** |
| --- | --- |

**Instructions**: Use the same calculator app as in Exercise 2, 3, 5 & 6, and include libraries to handle the calculations and logging logic.

Part 1: Expand the Calculator library from slide 71 to include functions for the other operations, and modify your controller to use the library.  
Part 2: Change the Calculator library to use a randomly generated ID instead of a timestamp  
Part 3: Create a second Logger library to handle logging, and modify the Calculator library to use this Logger to log all requests.

**Goal**: To learn how to refactor code to make use of object-oriented libraries to improve maintainability.

**Submission**: Commit your changes to the same **calculator-app** folder you used for Exercise 2, 3, 5 & 6.

**Extension**: Take the OOP coding challenge at <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Test_your_skills:_Object-oriented_JavaScript>

**Resources & Extra Learning**:

* <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_programming>
* <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Classes_in_JavaScript>

| **Lab Exercise 8:** | **Slide 86** |
| --- | --- |

**Instructions**: Finalise your calculator application by creating some documentation, and then practice implementing a backend using the skills and techniques learned in this module.

Part 1: Use the same calculator app as in Exercise 2, 3, 5, 6 & 7, and create interactive API documentation using Swagger, following the starter code on slides 78-82.  
Part 2: Follow the instructions on the slide to set up an Express back-end for your Fake Store exercise from Module 4. Create a new Express app for this exercise, using routes and controller functions to handle fetching the Fake Store data, and your front end should fetch from your back-end route/s instead of directly from the Fake Store API.

**Goal**: To learn how to document an Express back-end API and practice creating one from scratch.

**Submission**: Commit your changes for Part 1 to the same **calculator-app** folder you used for Exercise 2, 3, 5, 6 & 7. Create a new folder called **fake-store-app** or similar for Part 2 and include it in your Module 5 LabExercises folder.

**Extension**: Add unit tests for all routes in Part 2, and try creating a library that caches the Fake Store data to avoid re-fetching it on every request.

**Resources & Extra Learning**:

* <https://github.com/swagger-api/swagger-ui>
* <https://learn.openapis.org/specification/structure.html>
* <https://quickstarts.postman.com/guide/express/index.html?index=..%2F..index#0>