# IAT – Practical Lab exercise 2 – Intro to DevOps

|  |  |
| --- | --- |
| **IAT industry area:** | **Digital** |
| **Signature discipline:** | Software development |
| **Micro Credential / Micro Skill title:** | Intro to DevOps |
| **Expected duration** (hours): | 3 hours |
| **Product Lead Organisation:** | **Macquarie University** |
| **Topic number/ week** | **Topic 2 – Practical Lab 2** |

## Practical Lab Details

Practical exercise Week 2 –

Performed by students before the online facilitated session with 1 hour synchronous class time to review and assist students’ work.

## Learning Outcomes

|  |  |
| --- | --- |
| **Learning Outcomes** | **Details** |
| LO3 | Demonstrate an understanding of DevOps workflows. |
| LO4 | Define and implement a deployment automation solution. |

|  |  |
| --- | --- |
| **Description:** | *In this practical task you'll be introduced to the basic usage of Git, GitHub and Jest testing for JavaScript through some hands-on tasks.*  *You will complete this practical at home on your own computer prior to your weekly class, with the opportunity to review the practical content during the class.*  *This task will provide you with the knowledge required to complete the final project assessment.*  *Detailed instructions will be found in the LMS but your educator may assist you with any questions or issues you encounter during this week’s class.* |

## Practical instructions Step-by-step (in table below)

|  |  |
| --- | --- |
| **Task /question** | **Details  (insert images as necessary here)** |
|  | In last week’s practical you cloned a Git repository that you will be working on for each of these practical exercises. Git repositories are often used for sharing work between a team of developers, meaning that multiple people may be making changes to the repository at the same time. Therefore, whenever starting a new session working on a shared repository it is best to ensure that your local copy of the repository is up to date with any changes other people may have made on the “remote” copy of the repository.  While no one else is working on your practical repository, it is still a good idea to get into this practice. You can check for and download any updates to the repository using the command *git pull.* Read more about how *git pull* works here: <https://www.atlassian.com/git/tutorials/syncing/git-pull>.  Once you have read this article, open your practical project code in VS Code and open a terminal if you do not already have one open (**ctrl+shift+`**). Inside this terminal use the *git pull* command to check for updates to the project. You should see a message that says *Already up to date.* You are now ready to start on this week’s practical. |
|  | When writing code, it is always important to test your code to check that it works correctly. You did this many times during last week’s practical by manually running your scripts, providing input and comparing the output with what you expected. However, doing this many times for a big project becomes very tedious and time-consuming. This is one of the key reasons that many programming languages have testing libraries for automating this sort of menial testing.  For this course you will be using Jest, a popular testing framework for JavaScript. It provides many advanced testing mechanisms for checking your code but for this course you will only be using some of the simpler ones.  The type of testing you will be performing with Jest is called unit testing.Unit testing involves splitting your program into smaller parts or “units” and testing each of those individually, which makes testing a bit more straightforward. Read <https://aws.amazon.com/what-is/unit-testing/> for some more information about unit testing and common practices involved in it. |
|  | In order to test a program you generally need to have some understanding of what the code for the program does. In the case of these practicals you will be working with a simple inventory management program, such as what a store might use to track stock, which runs in the command line. In your VS Code terminal you can start the program by running the command *npm start.* Spend some time running the program and experimenting with its functionality. Once you feel like you understand what it does, move onto the next step. |
|  | The code for this program is contained in three files inside the *src* folder (which can be found in the **Explorer** panel on the left of your screen). *index.js* contains the main logic for the program, while *utilities.js* and *inventoryDisplay.js*contain extra functions that are used by *index.js*. Feel free to have a brief look at each of these files but you will not need to fully understand the code to continue.  For this week’s practical you will be focusing on the *isValidDateString* function in *utilities.js*. While experimenting with the program, you hopefully noticed that there are a few situations in which the user is asked to enter a date in a particular format. The *isValidDateString* function is used to validate these inputs, ensuring the user provides an actual valid date. Take a few minutes to look through the code for the function and read the comments to better understand how it works. |
|  | The tests for the project are housed in the *tests* folder. You will see that there are two files, *utilities.test.js* and *inventoryDisplay.test.js*, which correspond to two of the three code files in *src*. In Jest terminology, each of these files constitutes a test suite, which is a collection of tests for a similar purpose. In this case, the purpose for the tests in each suite is to test one particular file.  Open *utilities.test.js* and take a look at its contents. You will see there are three tests, where each test is a function with two parameters: a string that provides a descriptive name for the test and an arrow function that contains the functionality of the test. It will be helpful for this exercise to know a bit more about functions in JavaScript. Read <https://www.w3schools.com/js/js_functions.asp> and <https://www.w3schools.com/js/js_arrow_function.asp> before continuing. |
|  | You may also have noticed that below the tests in *utilities.test.js* there are a series of comments that describe some tests that need to be added for *isValidDateString*. Based on your understanding of the *isValidDateString* function can you spot any that are missing? Add comments to describe any additional tests that you think may need to be added to thoroughly test *isValidDateString* (the aim is to ensure that there is a test representing every possible case that could be encountered). |
|  | Add a new empty test under the *// Test for invalid string* comment. Give it a descriptive name and provide an empty arrow function for now (reference the format of the existing tests if you need a guide).  When creating the functionality of a test you must provide an expectation (denoted by *expect(<value>)*), which is the value produced by the code you want to test, and a matcher, which describes a condition or value to match your expectation against. Each test should contain only one expectation and matcher pair.  Jest provides a wide variety of matchers to suit the needs of various test types but for this course you will only need to use the *toBe(<value>)* matcher which checks if the value provided to the expectation **exactly matches** the value provided to the matcher.  Fill in the arrow function of the empty test you created with an expectation that you give the value *1* and a *toBe* matcher that you give the value *0*. This code that you have just written will expect 1 to be 0, which you know is not true. This test will therefore always fail. |
|  | To run your tests for this project, enter the command *npm test* in your terminal. You will receive a report that tells you about the various tests that ran, including how many suites and individual tests passed/failed. In this case, you should see a prominent indicator that the test suite *utilities.test.js*, and more specifically the test you added, has failed.  The results will also give you details about what was wrong. For your test you should see that the expected value was *0* but the value that was given was *1*, which lines up with the code you wrote. If you change the value passed to *toBe* to a *1* and run the tests again you will see that 2 test suites and 10 tests pass. |
|  | Good work, you have now written and run your first Jest test! Time to make this test do what it is actually supposed to do. Swap out the values in *expect* and *toBe* to test if *isValidDateString* returns *false* for a string that is clearly not a valid date (e.g. *“thisisnotadate”*). Don’t forget to make sure that your test has a meaningful and informative name. Save your code and run the tests to check if your test passes. |
|  | Now create tests to check each of the cases described in the remaining comments (including those that you added earlier), inserting them under the appropriate comments and giving them meaningful names. You may wish to run your tests after adding each new test to check that it passes.  Once all your tests are passing move onto the next step. |
|  | While all the changes that you have made are saved onto your local machine, they are not yet part of the Git repository that you are working on. Whenever you add a major feature or block of code to the project you should always make sure you also save it to the repository.  Luckily, this is a relatively straightforward three-step process. Firstly, you must add the modified files to your repository’s staging area with *git add* command. Read <https://www.atlassian.com/git/tutorials/saving-changes> to find out about this command. While the command can be provided with specific files to add to your staging area, it is often easier to add all the files modified in one go by providing the character “.” instead of a file name.  Give this a go now by running *git add .* in your terminal.  If you now run the *git status* command to display the status of your repository you should see *Changes to be committed:* followed by a list of files that you have modified. Your modified files are now ready for the next step. |
|  | The second step is to move the changes from the staging area to your local copy of the repository in an action called committing. This is achieved using *git commit* which you can read about at <https://www.atlassian.com/git/tutorials/saving-changes/git-commit>. Use the command with the –*m “<commit message>”* option to commit your staged changes (make sure you provide a meaningful commit message describing your changes). Note that if this is your first time using Git on this computer you may be asked to enter some credentials before you can commit; Git will tell you how to do this.  You are now ready for the final step of saving your changes. |
|  | Now that you have committed your changes to your local repository, you need to add them to the central remote repository (the one on GitHub that you looked at briefly in Week 1). You can do this with the *git push* command. <https://www.atlassian.com/git/tutorials/syncing/git-push> explains more about this command. Use the *git push* command to push your committed changes to your remote repository. Your repository is now up to date with your new changes. |
|  | This brings you to the end of this week’s practical exercise. In this practical you have learnt the basics of testing JavaScript with Jest and using Git to synchronise your work with a repository. For more information on Jest testing you can visit their docs at <https://jestjs.io/docs/getting-started>. To learn more about using Git you can check out Atlassian’s Git tutorials at <https://www.atlassian.com/git>. |