Lab 4: Web Development using Node, NPM & Jade

Objective:

- To develop web pages using Node, NPM and Jade.

Submission:

- Checkpoints that need to be shown to the course teacher during the lab.

Introduction:

In this lab, you will learn how to develop web pages using Node, NPM and Jade.

Node.js (node in short, nodejs.org) is a JavaScript runtime built on Chrome's V8 JavaScript engine. It is essentially a server side scripting language, much like PHP, but completely using JavaScript. JavaScript is the de-facto client-side scripting language for web development. A server side JavaScript engine, like Node, allows a web developer to exploit the JavaScript concept in a server environment. What it means is that you can code in JavaScript for your back-end without needing to learn any new language such as PHP.

NPM (Node Package Manager) is used to fetch any packages (JavaScript libraries) that an application needs for development, testing, and/or production, and may also be used to run tests and tools used in the development process. It is similar to apt in Ubuntu which is used to install, update and uninstall any software (package) in the OS.

Jade is a template engine which translates its template into an HTML page. With Jade, you can create web pages using its specific commands without writing any HTML code! It improves the readability, maintainability and productivity of your code.

There are five tasks that you will need to complete in this lab. Each task has a checkpoint. ***Complete all checkpoints and show it to your teacher.***

Task-1: Install Node/NPM (3 marks)

At first, check if you node already installed on your machine, using the following command:

- `$ node -v`

If node is installed, its version number will be printed. If you do not see the version number printer, you will need to install node. You can do this using the following commands:

- `$ sudo apt-get update`
- `$ curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash –`
- `$ sudo apt-get install -y nodejs`

Ensure that node has been properly installed by checking its version. Also, check if npm has been installed as well using the following command:

- `$ npm -v`

Now, we are ready to start programming with node and npm. In our first task, we will create a web server using node using the following commands:

- `$ cd`

- `$ mkdir nodeFirst`
- `$ cd nodeFirst`
- `$ nano hello.js`

Copy the following contents into your hello.js file and save it.

```
//Load HTTP module
const http = require("http");
const hostname = '127.0.0.1';
const port = 3000;

//Create HTTP server and listen on port 3000 for requests
const server = http.createServer((req, res) => {

  //Set the response HTTP header with HTTP status and Content type
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

//listen for request on port 3000, and as a callback function have the port listened on logged
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

In this code, you at first declare an http constant by “requiring” an HTTP package. Node has a few built-in packages for you to use. HTTP package is one of them. This package allows you to create a web-server using just a few lines of code. Then, you instantiate a server utilising the http constant and then set a listener for the server at the specified port. Once an HTTP request is received by this server, it returns a plain response (check the response header) with the ‘Hello World\n’ text in it.

While in *nodeFirst* directory, issue the following command:

- `$ node hello.js`

It should show a message stating the server is running at a URL. Check the URL in your browser. If you can see the Hello World message in the browser, then your first task and checkpoint are completed. **But do not show this to your teacher yet.**

Task-2: Node with Express (3 marks)

In this task, you will use Node Express to develop your web page. Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates the rapid development of Node based Web applications.

Express is the most popular *Node* web framework, and is the underlying library for a number of other popular Node web frameworks. It provides mechanisms to:

- Write handlers for requests with different HTTP verbs at different URL paths (routes).
- Integrate with "view" rendering engines in order to generate responses by inserting data into templates.

- Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
- Add additional request processing "middleware" at any point within the request handling pipeline.

While *Express* itself is fairly minimalist, developers have created compatible middleware packages to address almost any web development problem. There are libraries to work with cookies, sessions, user logins, URL parameters, POST data, security headers, and *many* more.

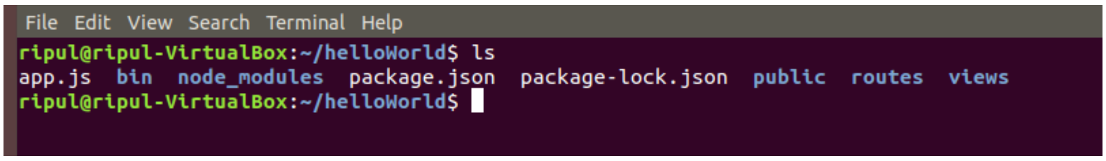
At first, install the express framework using the following commands:

- `$ cd`
- `$ sudo npm install express-generator -g`

Check if express has been properly installed using this command: `$ express –version`. The issue the following commands to create a node application using express.

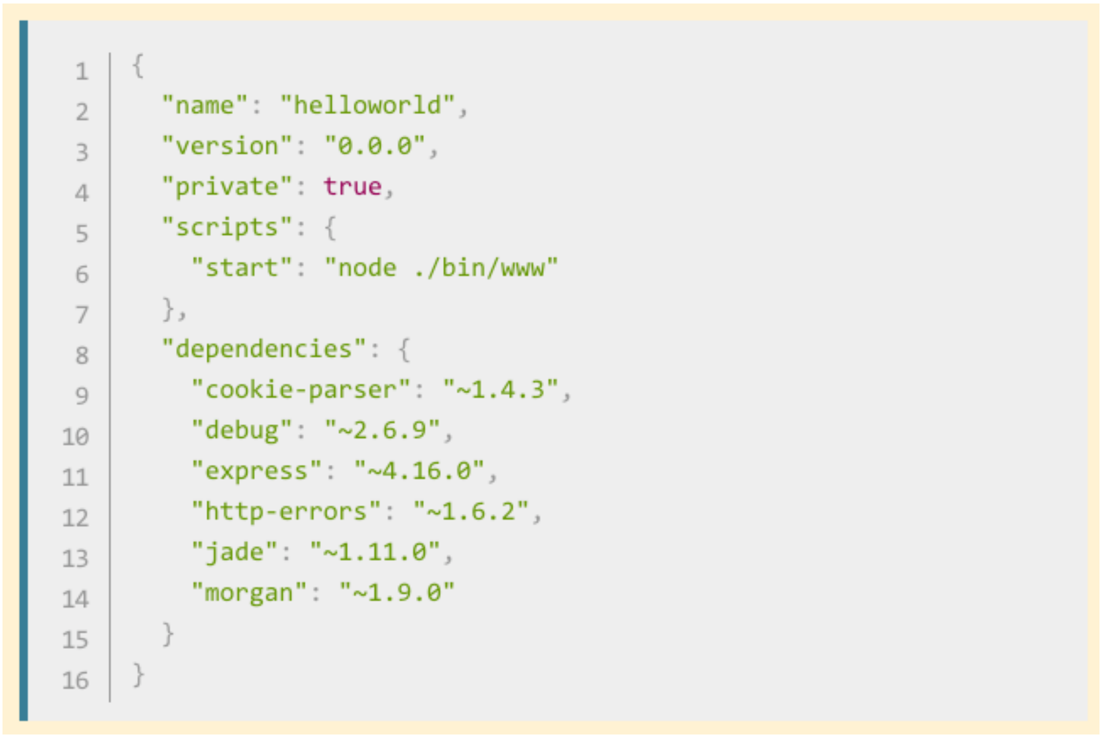
- `$ express expressApp`

Here, express will create the new *expressApp* application in the *expressApp* sub folder of your current location, displaying build progress on the console. On completion, the tool will display the commands you need to enter to install the Node dependencies and start the app. cd into expressApp folder and you will see contents similar to the following figure.



```
ripul@ripul-VirtualBox:~/helloWorld$ ls
app.js  bin  node_modules  package.json  package-lock.json  public  routes  views
ripul@ripul-VirtualBox:~/helloWorld$
```

The new app will have a package.json file in its root directory. Open the package.json file. It will have contents similar to the following figure:



```
{
  "name": "helloworld",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "cookie-parser": "~1.4.3",
    "debug": "~2.6.9",
    "express": "~4.16.0",
    "http-errors": "~1.6.2",
    "jade": "~1.11.0",
    "morgan": "~1.9.0"
  }
}
```

Among other things, the package.json lists the dependencies for your express app. These dependencies are nothing but different packages that your will express app will use. To use these dependencies, you will need to install them using the following command (while inside expressApp folder):

- `$ npm install`

Once installed, execute the following command to start your express web:

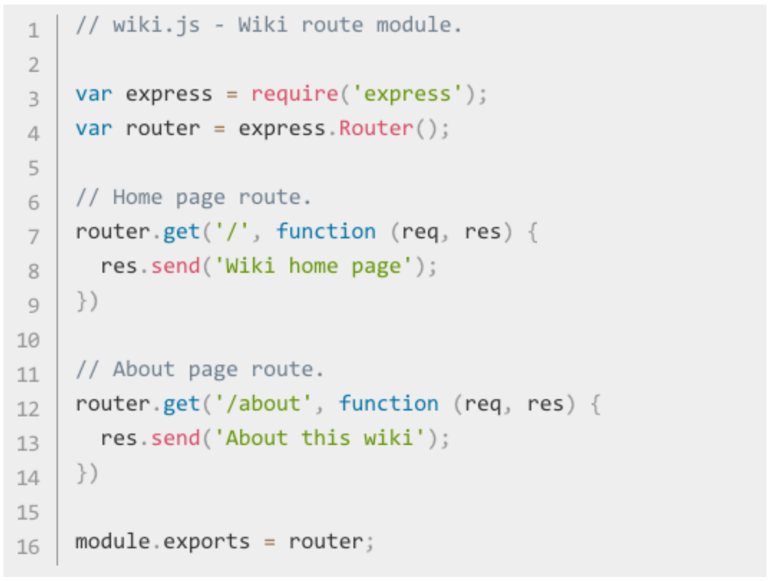
- `$ DEBUG=expressApp:* npm start`

It will display the port in the console from where you can access your application. If you can see Welcome to Express in your browser this means your app is working fine. Now, explore app.js and other contents within the expressApp folder to understand the importance of different files. Once you are happy, you have completed this task and the checkpoint. If your node application from the first Task is still running, you might need to change the port in your first to make your new expressApp run. **But do not show this to your teacher yet.**

Task-3: Node routes (4 marks)

A route is a section of Express code that associates an HTTP verb (GET, POST, PUT, DELETE, etc.), an URL path/pattern, and a function that is called to handle that pattern. There are several ways you can handle routes in Node. Two mechanisms are explained below.

Look at your app.js file to understand if it has utilised router middleware to handle routes. If so, you can handle routes (different URLs) similar to figure below:



```
// wiki.js - Wiki route module.

var express = require('express');
var router = express.Router();

// Home page route.
router.get('/', function (req, res) {
  res.send('Wiki home page');
})

// About page route.
router.get('/about', function (req, res) {
  res.send('About this wiki');
})

module.exports = router;
```

Otherwise, you need to inspect the JavaScript files in your route folder of your express web and the app.js file to understand how the routes can be handled.

Then, handle the following routes in your application:

- /about: having a line of text which contains what an About page should contain.
- /students: having the registration number(s) of your group
- /time: showing the current date dd/mm/yyyy format

Complete this task and show all three checkpoints to your teacher.

Task-4: Use Jade (5 marks)

In this task we will use Jade, Stylus and Nib to create an express application. Jade was already discussed in the beginning of this manual. It is to be noted that Jade has been renamed to **Pug** due to trademark issue. However, I am keeping this name in case you are looking for tutorial on the Internet. If you do so, you can use Jade or Pug as the keywords.

A brief introduction to Stylus and Nib are given below:

- **Stylus** is a CSS-preprocessor. If you're familiar with LESS or SASS, Stylus does very much the same job (if not, Stylus is a language which compiles down to CSS for the browser; it adds new features that make CSS easier to work with).
- **Nib** is a set of utilities for Stylus. Among other things, it enables going vendor-prefix free, which in my eyes makes it worthwhile alone.

Now, follow the steps given below to complete the task.

At first create a new project and prepare it using the following commands:

- `$ cd`
- `$ mkdir jadeApp`
- `$ cd jadeApp`
- `$ nano package.json`

Copy the following contents to package.json and save it.

```
{
  "name": "MySite",
  "version": "0.0.1",
  "private": "true",
  "dependencies": {
          "express": "3.0.0alpha4",
          "jade": "^1.0.4",
          "stylus": "^0.49.1",
          "nib": "^1.0.4"
   }
}
```

Next, execute this command: `$ npm install`. Once installed, create a file called app.js and fill the file with the contents of the supplied app.js file. Please take time to understand the contents of app.js file. A brief explanation follows.

The app.set() directives tell Express that we want to use Jade, and where we will keep our views.

The app.use() calls pass 'middleware' functions for Express to use. Middleware are simply functions that have the signature fn(req, res, next). They take the request object, the response object and next, which will call the next piece of middleware. Express executes them in the order in which they are passed, and any of them may send a repines, preventing the execution of any remaining in the series (by not calling next()).

The first piece of middleware we apply is the Express logger in 'dev' mode. This will simply log incoming requests to the console. Next, the Stylus middleware is applied, which will compile our .styl files to CSS. In order to use nib, we pass in a custom compile function to the Stylus middleware. After that it's the Express static middleware, which is used for serving static files. You might be wondering why we need this. Unlike Apache, the Express server doesn't mimic the filesystem to the visitor. This allows great flexibility for the url structure of your site, but it's quite a useful feature for serving static assets, so the static middleware does exactly this on the directory that we pass. A file 'pic.jpg' in a folder 'images' within 'public' will be available to the client at '/images/pic.jpg'.

Next, we create two jade files to populate the contents of our application. Execute the following commands while in jadeApp folder:

- `$ mkdir views`
- `$ cd views`
- `$ nano layout.jade`

Copy the following contents in this file and save it.

```
Doctype
html
  head
    title #{title} - My Site
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    header
      h1 My Site
    .container
      .main-content
        block content
      .sidebar
        block sidebar
    footer
      p Running on node with Express, Jade and Stylus
```

Even if you're not familiar with Jade, hopefully you'll be able to get an idea of what's going on here. Each word beginning a line represents a tag. Indentations after that line denote the tag's content, so rather than having to close a tag, you just stop indenting lines. Within the head tag we output the contents of a title variable with #{title} -- you'll see later on where this gets defined. We are also linking to the place we expect our compiled Stylus to be. The only thing left to explain here is block. Since we won't be filling in this file with our page contents, we can denote blocks. Templates which inherit from this one can define their own content to be output within these blocks.

Next, create our block jade from where data will be populated to this jade file. Execute the following commands:

- `$ nano index.jade`

Copy the following contents in this file and save it.

```
extend layout
block content
  p
    | Vivamus hendrerit arcu sed erat molestie
    | vehicula. Sed auctor neque eu tellus
    | rhoncus ut eleifend nibh porttitor. Ut in.
  p
    | Donec congue lacinia dui, a porttitor
    | lectus condimentum laoreet. Nunc eu
    | ullamcorper orci. Quisque eget odio ac
    | lectus vestibulum faucibus eget in metus.
    | In pellentesque faucibus vestibulum. Nulla
    | at nulla justo, eget luctus tortor.
block sidebar
  .widget
    h1 Widget
    p
      | Sed auctor neque eu tellus rhoncus ut
      | eleifend nibh porttitor. Ut in nulla enim.
    p
      | Vivamus hendrerit arcu sed erat molestie
      | vehicula.
```

You can see that the first thing we are doing is extending layout. This will output all of the content of layout.jade, and allow us to define content to be output in place of the blocks that we defined.

Next, we need to define our style in our Stylus file. While in the root of your jadeApp folder, execute the following commands:

- `$ mkdir -p public/stylesheets`
- `$ cd public/stylesheets`
- `$ nano style.styl`

Fill the file with the contents of the supplied style.styl file and save it.

Now, while in the root of your jadeApp folder, execute the following command:

- `$ node app.js`

Navigate to the correct URL and if everything works fine, you should be able see a nice little web page. This completes the task. **Show this to your teacher to tick-off your checkpoint.**

Task-5: Handling Form (5 marks)

In this task, you will need to handle HTML forms using Express and Jade. For this, follow the tutorial from this URL: https://www.tutorialspoint.com/expressjs/expressjs_form_data.htm

You will need to update the task 4 to in order to handle form data according to the tutorial. You will need to handle the form data and then display them in another Jade page.

Complete this task and show this to your teacher to tick-off the last checkpoint.

Reference:

- https://www.clock.co.uk/insight/a-simple-website-in-nodejs-with-express-jade-and-stylus