Node.JS Bootcamp – Labs



Labs Summary

[Module 1: Node.JS 4](#_Toc35455357)

[Lab 1: Installing VS Code 4](#_Toc35455358)

[Lab 1 A: Downloading Visual Studio Code 4](#_Toc35455359)

[Lab 2: Installing Node.JS 6](#_Toc35455360)

[Lab 2 A: Installing Node.JS on Windows 6](#_Toc35455361)

[Lab 2 B: Installing Node.JS on Linux 6](#_Toc35455362)

[Lab 2 C: Installing Node.JS on Container 6](#_Toc35455363)

[Lab 3: Installing Git 7](#_Toc35455364)

[Lab 3 A: Installing Git on Windows 7](#_Toc35455365)

[Lab 4: Installing Yarn 8](#_Toc35455366)

[Lab 4 A: Installing Yarn on Windows 8](#_Toc35455367)

[Module 2: Working with Node.JS 9](#_Toc35455368)

[Lab 1: Creating a Hello World app 9](#_Toc35455369)

[Lab 1 A: Creating a Hello World app 9](#_Toc35455370)

[Lab 2: Creating a Hello World app 12](#_Toc35455371)

[Lab 2 A: Creating a simple API app 12](#_Toc35455372)

[Lab 2 B: Modifying Express App 14](#_Toc35455373)

[Lab 3: Working with Front-End frameworks 16](#_Toc35455374)

[Lab 3 A: Creating a simple Angular app 16](#_Toc35455375)

[Lab 3 B: Creating a simple React app 23](#_Toc35455376)

[Lab 3 C: Creating a simple Vue app 27](#_Toc35455377)

[Module 3: Deployment to Azure WebApp 32](#_Toc35455378)

[Lab 1: Deploying Express API to Azure WebApp 32](#_Toc35455379)

[Lab 1 A: Install Azure CLI 32](#_Toc35455380)

[Lab 1 B: Create Azure Resource Group 32](#_Toc35455381)

[Lab 1 C: Create App Service Plan and Azure WebApp 33](#_Toc35455382)

[Lab 1 D: Deploy Express API using Local Git to Azure WebApp 34](#_Toc35455383)

[Lab 2: Deploy Angular/React/Vue app to Azure WebApp 36](#_Toc35455384)

[Lab 2 A: Updating Back-End endpoint to Azure WebApp Back-End URL 36](#_Toc35455385)

[Lab 2 B: Create Azure WebApp to host Front-End Application 39](#_Toc35455386)

[Lab 2 C: Deploy Angular/Reach/Vue Front-End using Local Git to Azure WebApp 39](#_Toc35455387)

[Module 4: Troubleshooting 43](#_Toc35455388)

[Lab 1: Deploy and Troubleshoot an application from a Git repository 43](#_Toc35455389)

[Lab 1 A: Tasks to be accomplished 43](#_Toc35455390)

# Module 1: Node.JS

In this module you will learn about:

* Installing Visual Studio Code
* Installing Node.JS
* Installing Git
* Installing Yarn

## Lab 1: Installing VS Code

In this lab you will learn how to download and install Visual Studio Code

### Lab 1 A: Downloading Visual Studio Code

Visual Studio Code is a lightweight code editor redefined and optimized for building and debugging modern web and cloud applications.

In this lab you will learn how to download and install Visual Studio Code

Please use one of the installation methods bellow.

Download Visual Studio Code:

1. Using your Internet browser go to <https://code.visualstudio.com/Download> and select the appropriate operating system version.
2. Execute the installer and follow the wizard.
3. Open Visual Studio Code to verify installation.

Installing using PowerShell:

Installing using PowerShell is useful for automation and quick install.

1. Copy the following script and execute in your PowerShell console.

##########################

# Installing VSCode

##########################

$url = "https://vscode-update.azurewebsites.net/latest/win32-x64/stable"

$Downloaddir = ($home+"\Downloads")

Invoke-WebRequest -Uri $url -OutFile ($Downloaddir+"\VSCodeSetup.exe")

Unblock-File ($Downloaddir+"\VSCodeSetup.exe")

$VSCodeInstallResult = (Start-Process ($Downloaddir+"\VSCodeSetup.exe") '/verysilent /suppressmsgboxes /mergetasks=!runcode,addcontextmenufiles,addcontextmenufolders,associatewithfiles,addtopath,desktopicon,quicklaunchicon' -Wait -Passthru).ExitCode

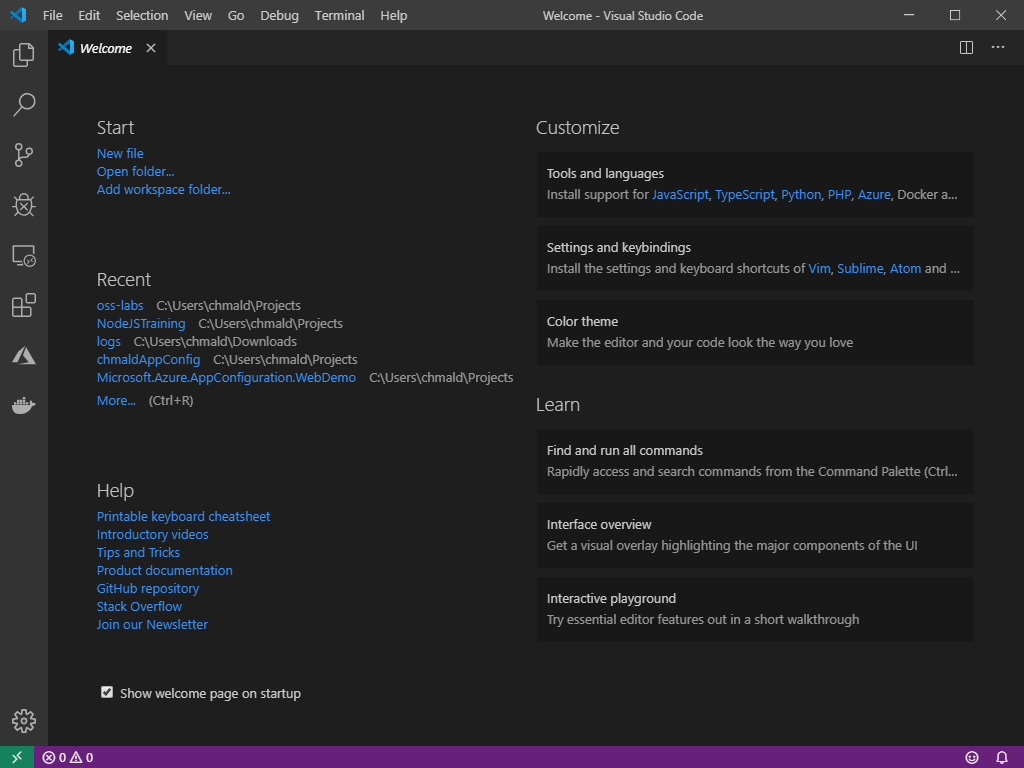
if ($VSCodeInstallResult -eq 0) {

    Log("Install VSCode Success")

}

1. Open Visual Studio Code to verify installation. You will have a icon in the Desktop.

Here is how Visual Studio Code looks like:



## Lab 2: Installing Node.JS

In this lab you will learn how to install Node.JS on Windows, Linux and Containers.

**Note:** It’s expected that you follow the instructions only for Windows. Keep in mind it’s important to understand how the installation steps works for Linux and Containers.

### Lab 2 A: Installing Node.JS on Windows

1. Using your Internet browser go to <https://nodejs.org/en/>
2. Download the current Node.JS LTS version.
3. Execute the installer and follow the wizard.

### Lab 2 B: Installing Node.JS on Linux

1. Using the Linux shell type the following commands:

curl -sL https://deb.nodesource.com/setup\_10.x | sudo -E bash -

sudo apt-get install -y nodejs

1. You can define which version will be installed by changing setup\_10.x to the desired version like setup\_12.x.

### Lab 2 C: Installing Node.JS on Container

1. In case you need to install Node.JS in a container you can use the same Linux command line in the Dockerfile. This scenario would be very specific since Node.JS offer an official image to be used.
2. Node.JS has a official Docker container images available at Docker HUB: <https://hub.docker.com/_/node/>
3. Those images can be referenced in the Dockerfile in the FROM statement like this:

FROM node:10

WORKDIR /usr/src/app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 8080

CMD [ "node", "server.js" ]

1. Those images will have Node.JS pre-installed with all prerequisites.

## Lab 3: Installing Git

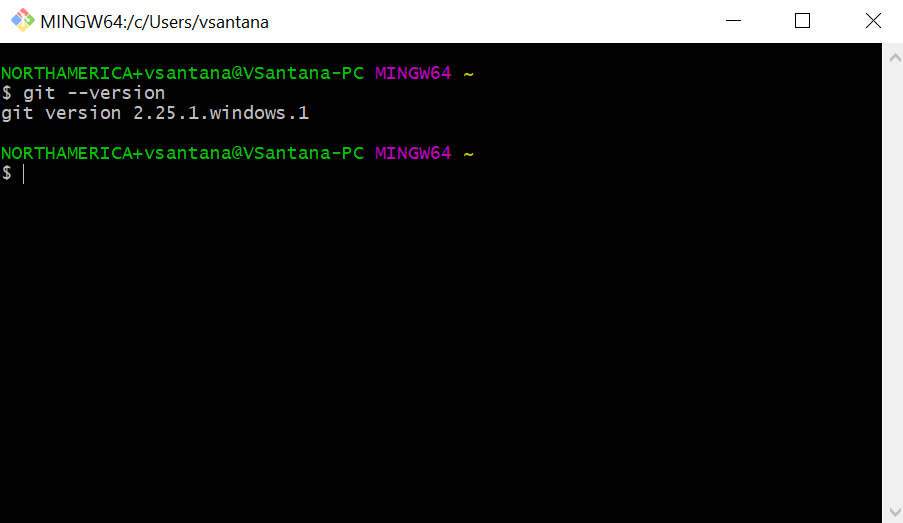
In this lab you will learn how to install Git on Windows.

**Note:** Most of Linux distributions come with Git pre-installed. You can start using it by just typing git in Linux Shell.

### Lab 3 A: Installing Git on Windows

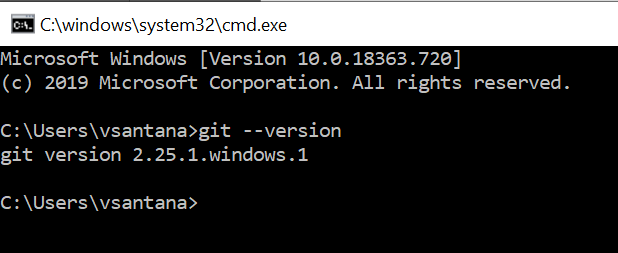
1. Using your Internet browser go to <https://git-scm.com/downloads> and select the appropriate operating system version.
2. Execute the installer and follow the wizard.
3. Following the recommended settings during the installation wizard, Git for Windows will install **Git Bash** and **Git for Windows command** **prompt**.
4. Open Git Bash Shell and type the following command:

**git --version**



1. Open Windows Command prompt and type the following command:

**git --version**



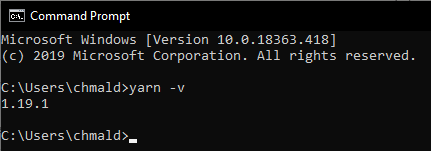
## Lab 4: Installing Yarn

In this lab you will learn how to install Yarn on Windows.

### Lab 4 A: Installing Yarn on Windows

1. Using your Internet browser go to https://yarnpkg.com/en/docs/install#windows-stable and select the appropriate operating system version.
2. Execute the installer and follow the wizard.
3. Using Windows Command prompt type:

**yarn -v**



# Module 2: Working with Node.JS

In this module you will learn about:

* Create a hello world app
* Create a simple API app
* Create an Angular/React/Vue App
* Installing Yarn

***Node:*** *Before you start it’s recommended that you create a Project folder named NodeJSTraining where you will add all projects for this training.*

## Lab 1: Creating a Hello World app

In this lab you will create a Hello World app using Node.JS.

### Lab 1 A: Creating a Hello World app

1. Open a new command prompt. (It’s important to re-open command prompt after you have installed Node.JS to reload the PATH environment variable)
2. Create a new directory under your preferred location and call it hello\_world by using the following command:

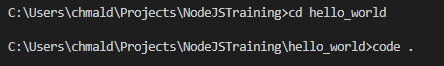
**mkdir hello\_world**



1. Open Visual Studio Code from this directory using the following commands:

**cd hello\_world**

**code .**

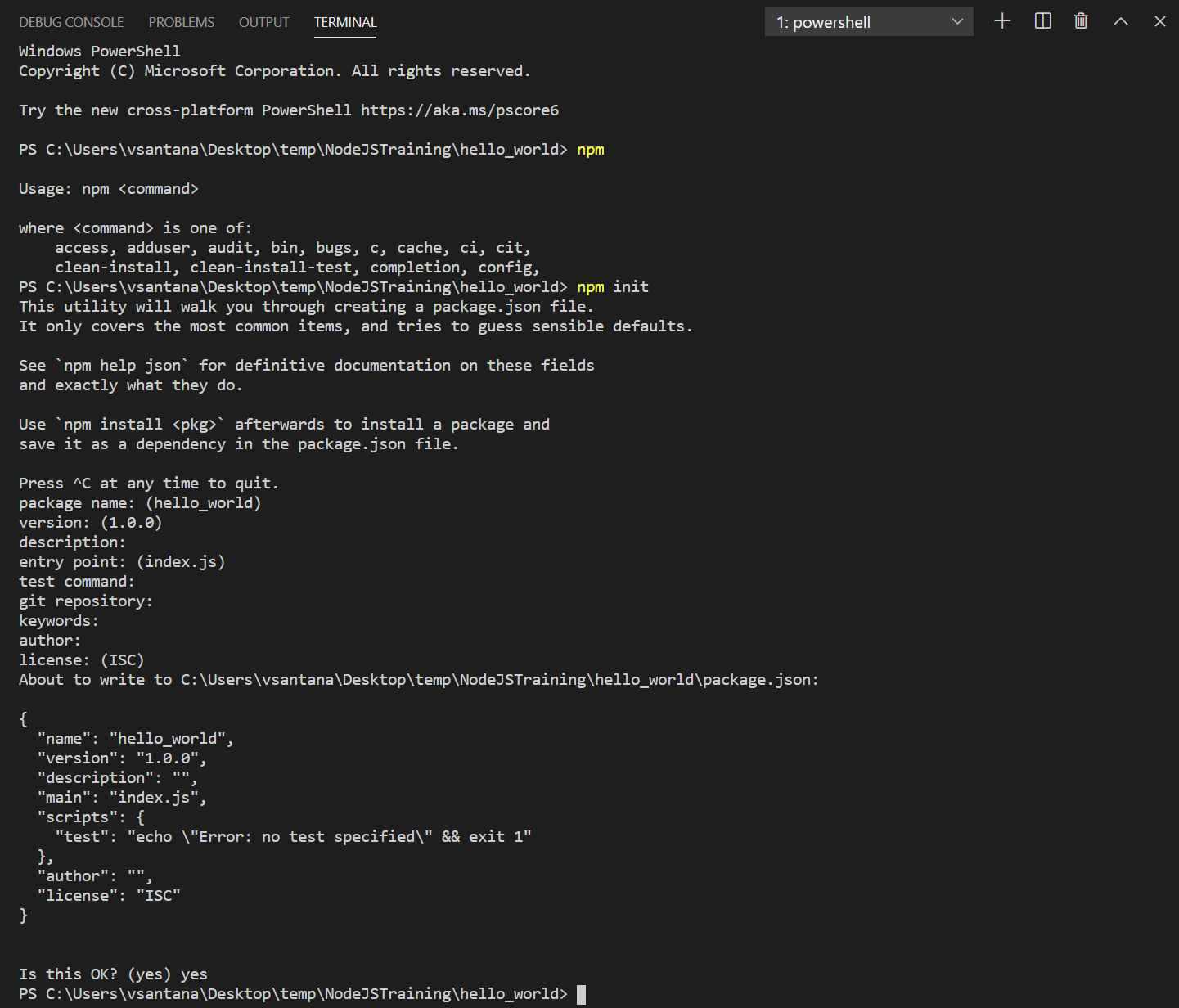


1. Inside Visual Studio code open a new terminal by clicking in **menu Terminal, New Terminal**.
2. Using the Visual Studio code terminal initialize a new project by typing:

**npm init**

It will create a new package.json file.

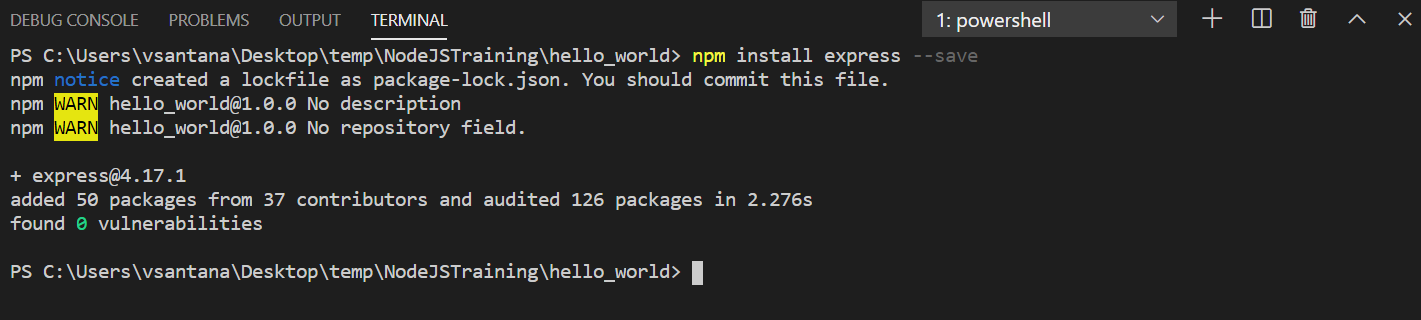
During the prompt use all default information.



1. In this lab, we are going to use Node.JS Express framework to create a simple app with a simple response.
2. Run the following command to add Express framework to your project:

It will create a node\_modules folder and add Express to package.json as a dependency.

**npm install express --save**



1. Using Visual Studio Code create a new file named **index.js**
2. Add the following code to the file:

const express = require('express')

const app = express()

const port = 3000

app.get('/', (req, res) => {

    res.send('Hello World!')

})

app.listen(port, () => {

    console.log(`Hello World app listening on port ${port}!`)

})

1. Line by line explanation:

Line 1: We are importing the express module into the application using the variable express.

Line 2: We create the app object by calling and creating an express instance.

Line 3: We are defining out application’s port.

Line 5-7: We add a new route for ‘/’ which will send back our response of Hello World!

Line 9-11: We set our application to start listening on this specified port defined on line 3.

1. At this point, we will run our application using the following command:

**node index.js**

1. Once started you should see the following message:

Hello World app listening on port 3000!

1. Using internet browser navigate to <http://localhost:3000>
2. You have now created your first Node.JS hello world app.

## Lab 2: Creating a Hello World app

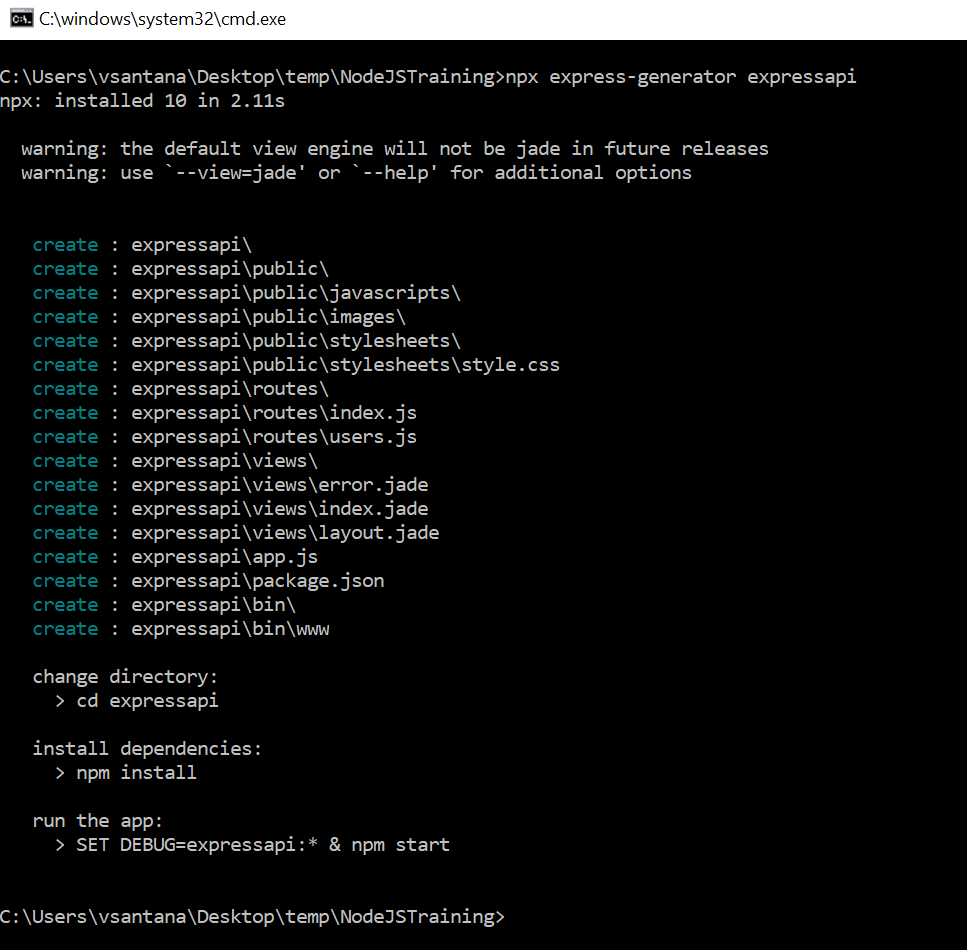
In this lab you will create a Hello World app using Node.JS.

### Lab 2 A: Creating a simple API app

In this lab, you will use express generator to create a base API.

1. Open Windows command prompt
2. Go under your preferred location where the base API will be created and execute the following command:

**npx express-generator expressapi**

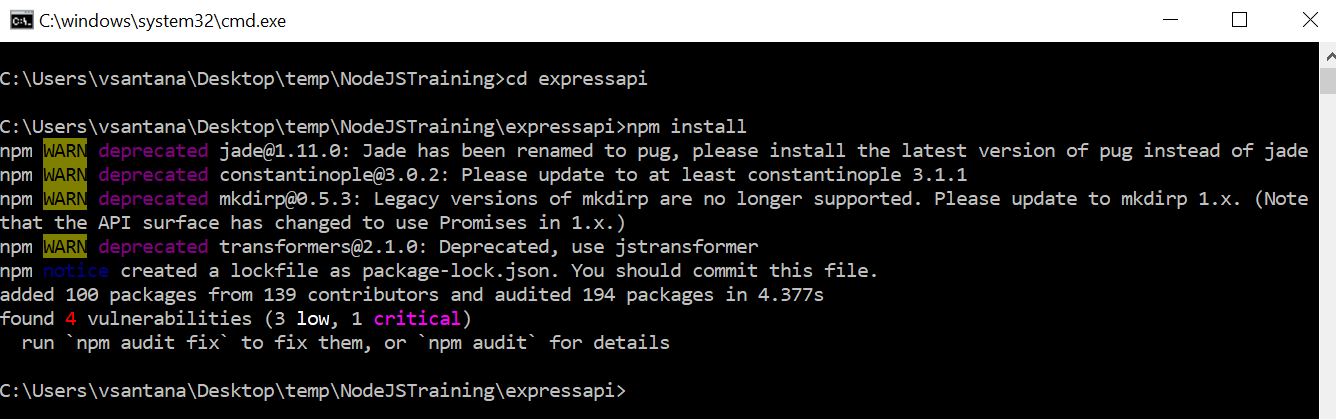


1. This will create a new base express app project under the directory **expressapi**.
2. Once it has been created, change your directory to your new express API by typing:

**cd expressapi**

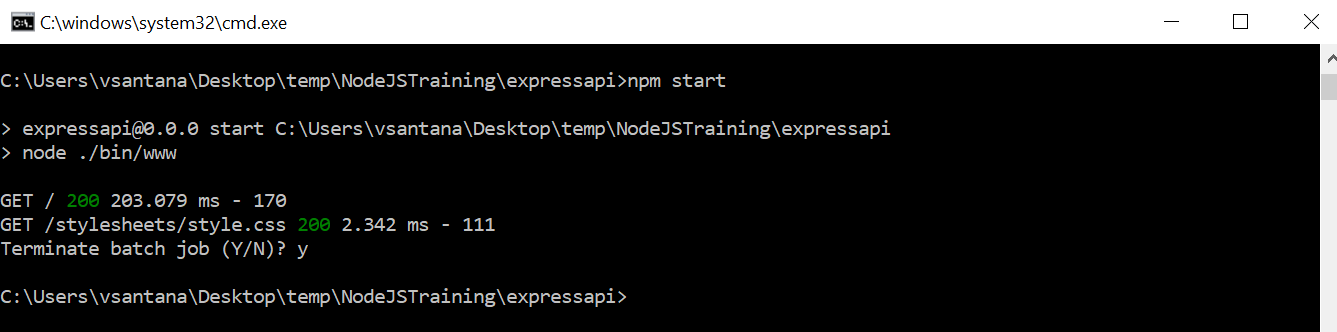
1. This new project template will have a files and folders for express framework. It also include a new **package.json** with all dependencies.
2. Install all dependencies using npm by typing:

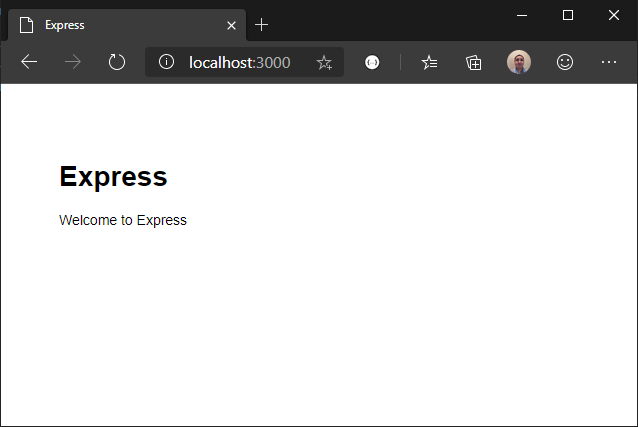
npm install



1. The previous command will create a node\_modules folder inside the project folder with all dependencies for this project.
2. Once it’s finished, initiate your new express app by typing:

npm start



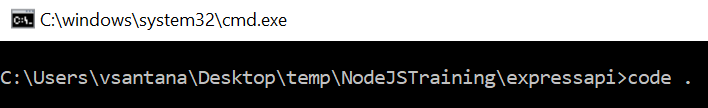
1. Open your internet browser and access <http://localhost:3000> to view your express app
2. 
3. Stop the app by pressing **CTRL+C**

### Lab 2 B: Modifying Express App

In this lab, you will modify the Express App to create an API response.

1. Using the Windows command prompt that is already open from the previous lab launch Visual Studio Code by typing:

code .

1. 
2. Open the file **app.js**
3. You need to add some CORS settings to be able to call this API from other applications.
4. Keep in mind this is for testing purpose for production environment it should be handled in a different way.
5. Add the following code right after the express app definition in line 10:

app.use(function(req, res, next) {

  res.header("Access-Control-Allow-Origin", "\*");

  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

  next();

});

1. It should look like this:
2. 
3. Open the file: **routes/users.js**
4. You will modify this file to make the API send an array of json simulating some users in your application.
5. In this file you will see the response for when someone navigates to **/users**.
6. We will be modifying line 6 to send a response of json (**res.json**) instead of a regular response (**res.send**).
7. Replace lines 5 to 7 with the following content:

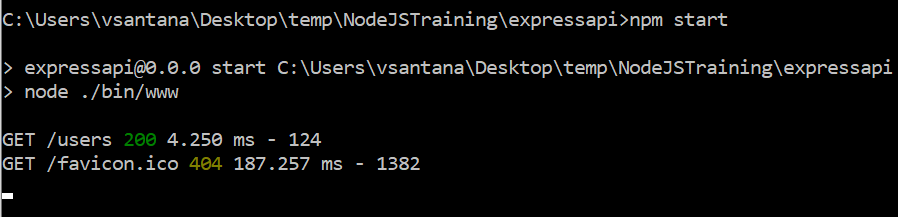
router.get('/', function(req, res, next) {

  res.json([{user: 'Adam', job: 'Global Security'}, {user: 'Chris', job: 'Blesser of Images'}, {user: 'yourname', job: 'whatdoyoudo'}]);

});

1. It should look like this:
2. 
3. Now that you have some data to query from this API, launch the application again by typing:

npm start

1. 
2. Test the change by using your internet browser and accessing the application using this link: <http://localhost:3000/users>
3. Let’s move along to the next lab to connect this further!!

## Lab 3: Working with Front-End frameworks

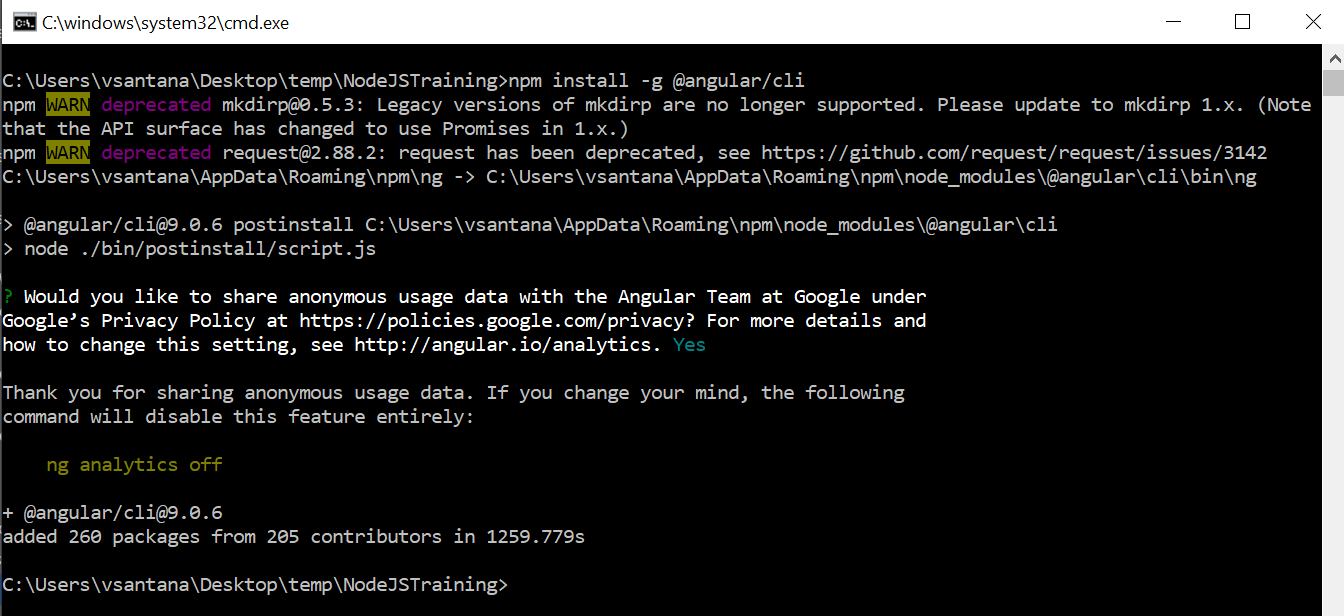
In this lab, you can choose from three different types of frontend frameworks to create to connect with your simple API app. The options are: Angular, React, or Vue. Each are different in their own way but serve the same purpose.

### Lab 3 A: Creating a simple Angular app

In this lab, you will use Angular CLI to create an Angular app from a default template.

1. To start with Angular, you will need to install the Angular CLI via NPM.
2. Npm can install packages local to your project by reading the package.json file but it can also install global packages which will be available to all projects. In the next step you will install @angular/cli and make it available globally by using the parameter **-g** in the npm cli.
3. Open Windows command prompt go under your preferred location and type:

npm install -g @angular/cli

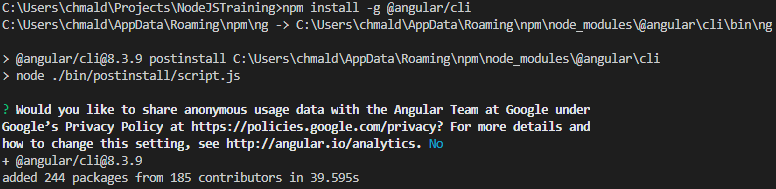


1. After the CLI is installed, you can use **ng** for any Angular commands.
2. Make sure you are under your preferred location/path in Windows command prompt.
3. The next command will create Angular app project folder.
4. For this example, we will not need angular routing and we can use CSS as the stylesheet format.
5. Create a new Angular app using the following command:

ng new angularfrontend

When prompted for Angular routing answer no.

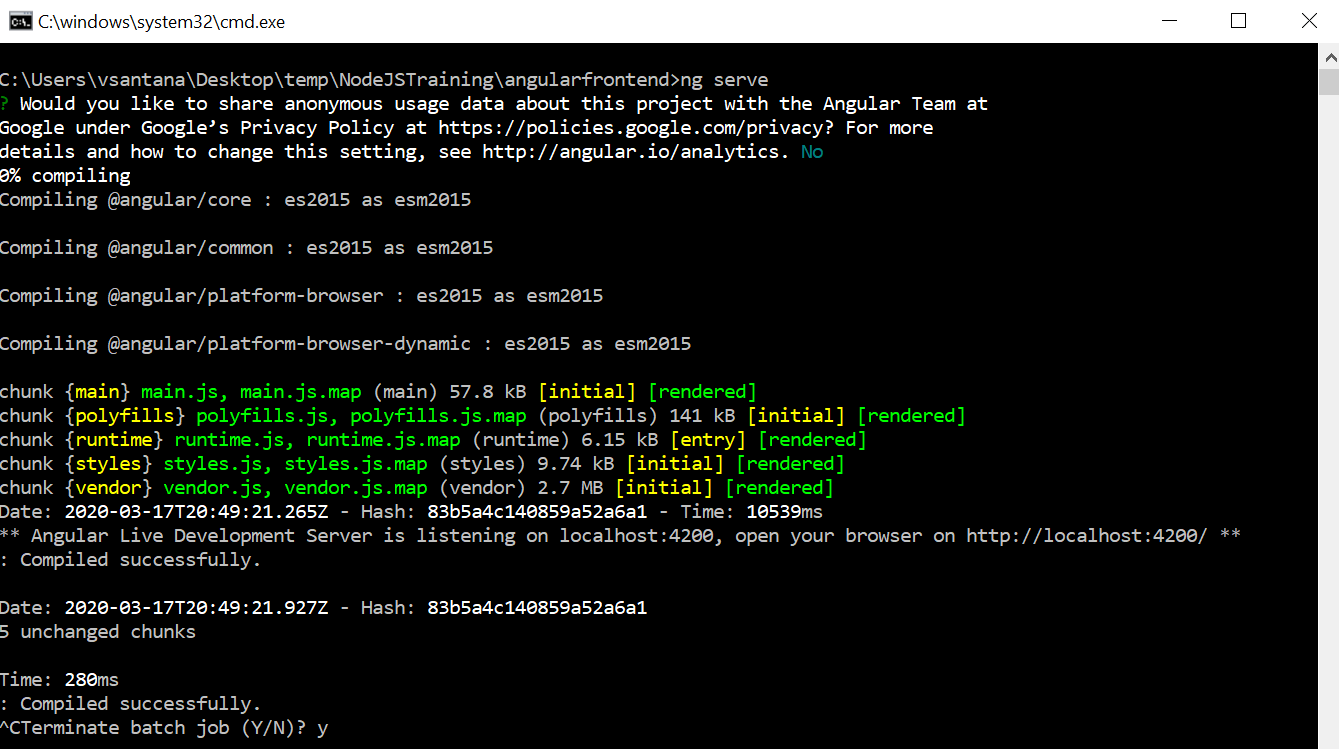
Next, chose CSS stylesheet.

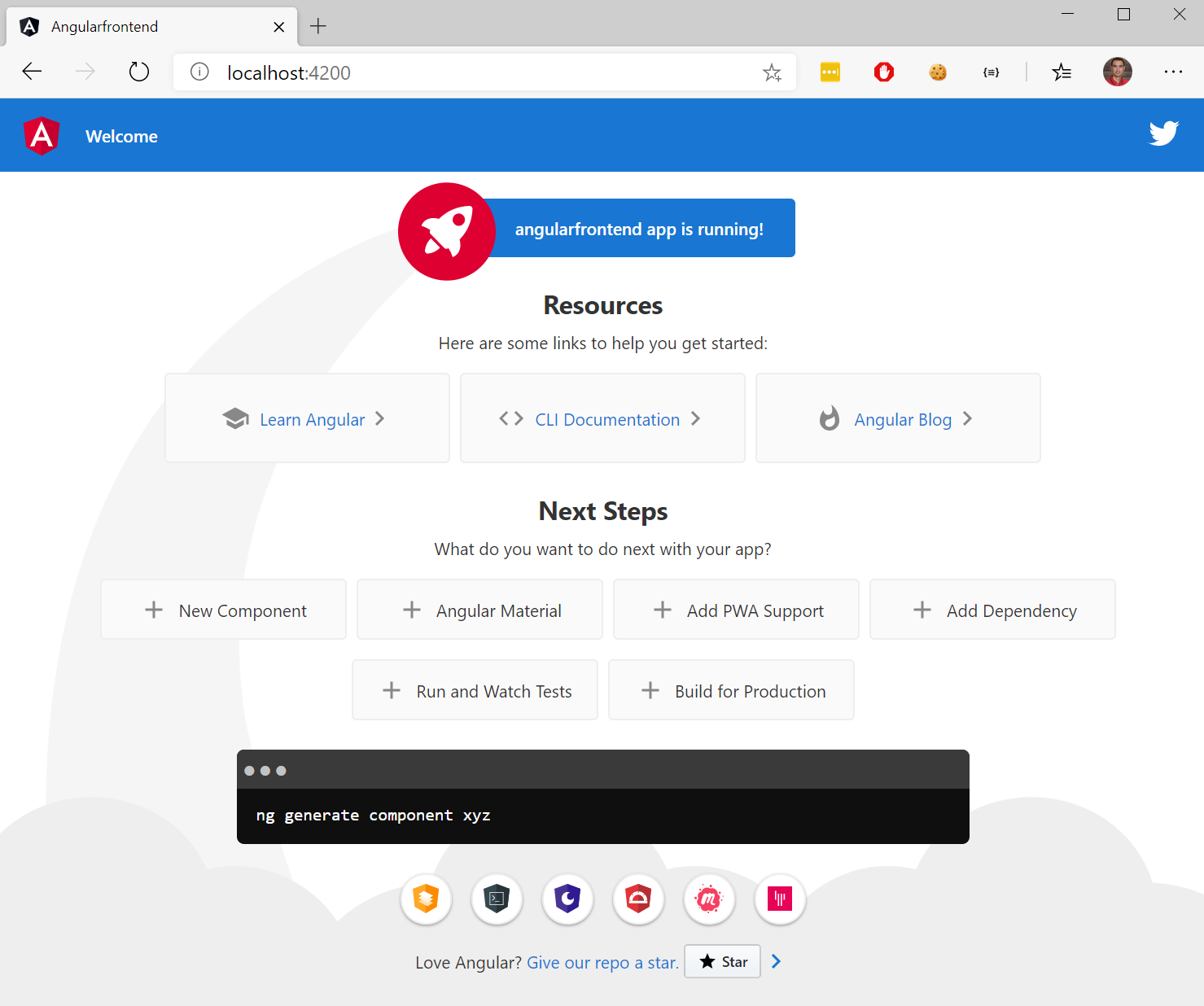
1. 
2. Once it has been created, change your directory to your new frontend app by typing:

cd angularfrontend

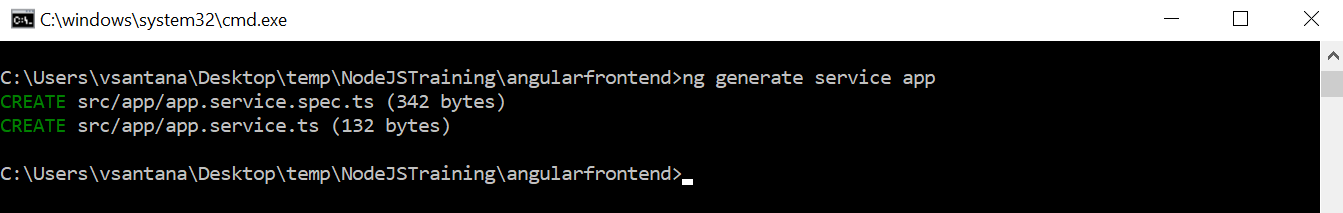
1. Start the new Angular app typing:

ng serve



1. Open the internet browser and navigate to <http://localhsot:4200> and the default Angular app should be running.
2. 
3. Stop the server pressing **CTRL+C**
4. You need to create an Angular service to call the API previously created to gather the json data.
5. Run the following command to create the service:

ng generate service app

1. 
2. It will create two files under the Angular project **src/app/app.service.spec** and **src/app/app.service.ts.**
3. Open the new project folder with Visual Studio Code.
4. Open the file **/src/app/app.service.ts.**
5. Replace the whole content with the following code:

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

@Injectable({

  providedIn: 'root'

})

export class AppService {

  getUsers() {

    return this.http.get('http://localhost:3000/users');

  }

  constructor(

    private http: HttpClient

  ) {}

}

1. In this service (the code you just replaced) there is a function called getUsers which will call the API and return the array of users. Please note there is also a **HTTPClient** added as well. For this to work you need to enable it in another file.
2. Save all the changes.
3. Open the file **/src/app/app.module.ts**
4. You need to import the **HttpClientModule** and set the **HttpClientModule** under the imports of the NgModule.
5. Replace the whole file content with this peace of code(it will add the import line and the module under the imports array):

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import { HttpClientModule } from '@angular/common/http';

import { AppComponent } from './app.component';

@NgModule({

  declarations: [

    AppComponent

  ],

  imports: [

    BrowserModule,

    HttpClientModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

1. Now that service and module are configured for HTTP requests, you need put it together inside the component for this app.
2. Open the following file: **src/app/app.component.ts**

import { Component, OnInit } from '@angular/core';

import { AppService } from './app.service';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit{

  title = 'angularfrontend';

  users;

  constructor(

    private appService: AppService

  ) {};

  ngOnInit() {

    this.users = this.appService.getUsers();

  }

}

1. Next you need to set the view to visualize the users.
2. Open the following file: src/app/app.component.html
3. This file is the public facing HTML and CSS for your component.
4. Override the whole content of this file with the following code:

<div style="text-align: center;">

  <div class="toolbar" role="banner">

    <img

        width="200"

        alt="Angular Logo"

        src="data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZpZXdCb3g9IjAgMCAyNTAgMjUwIj4KICAgIDxwYXRoIGZpbGw9IiNERDAwMzEiIGQ9Ik0xMjUgMzBMMzEuOSA2My4ybDE0LjIgMTIzLjFMMTI1IDIzMGw3OC45LTQzLjcgMTQuMi0xMjMuMXoiIC8+CiAgICA8cGF0aCBmaWxsPSIjQzMwMDJGIiBkPSJNMTI1IDMwdjIyLjItLjFWMjMwbDc4LjktNDMuNyAxNC4yLTEyMy4xTDEyNSAzMHoiIC8+CiAgICA8cGF0aCAgZmlsbD0iI0ZGRkZGRiIgZD0iTTEyNSA1Mi4xTDY2LjggMTgyLjZoMjEuN2wxMS43LTI5LjJoNDkuNGwxMS43IDI5LjJIMTgzTDEyNSA1Mi4xem0xNyA4My4zaC0zNGwxNy00MC45IDE3IDQwLjl6IiAvPgogIDwvc3ZnPg=="

      />

  </div>

  <div class="content" role="main">

    <div \*ngFor="let user of users | async">

      <p>{{ user.user }} - {{ user.job }}</p>

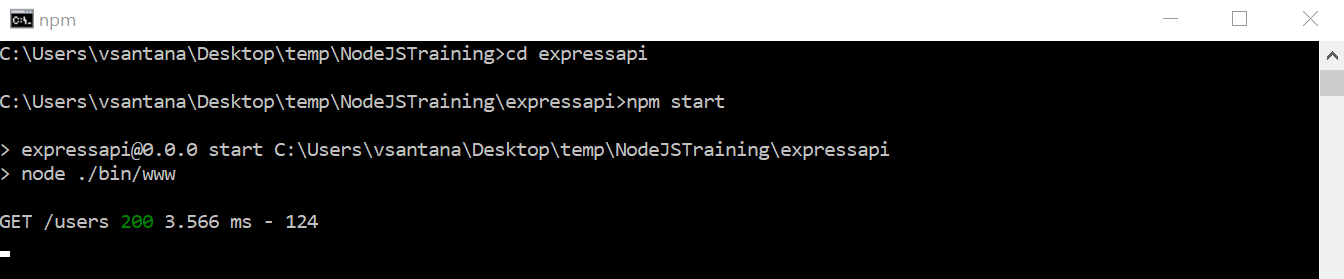
    </div>

  </div>

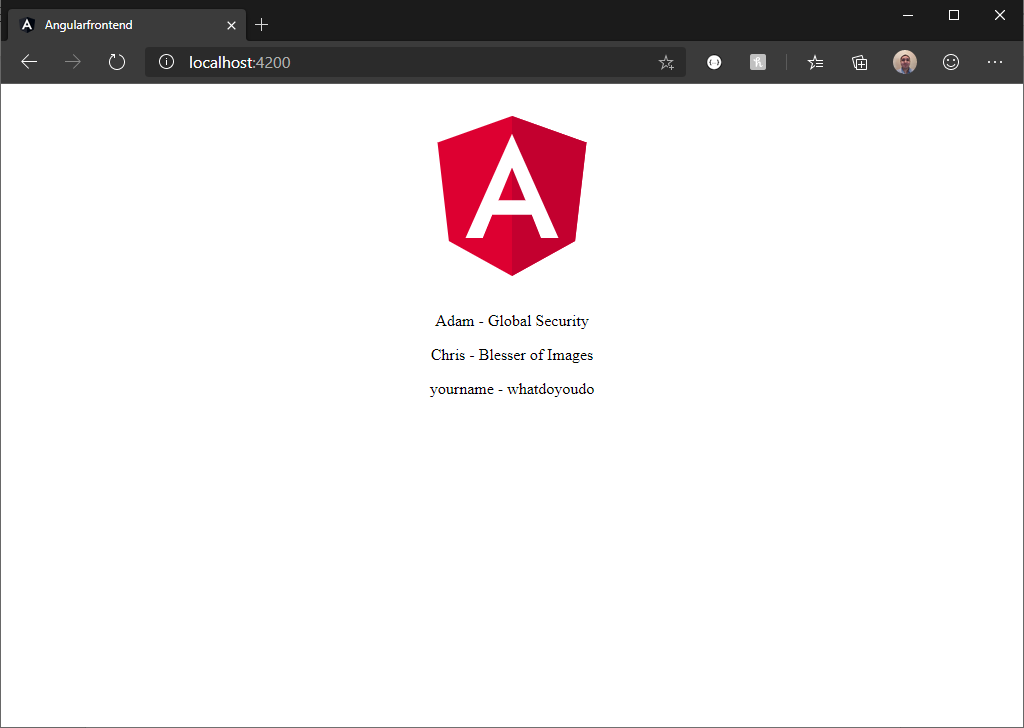
</div>

1. It’s time to initialize the application Angular App working as the Front-End and the ExpressAPI working as the Back-End.
2. Open a new Windows Command prompt and initialize the ExpressAPI by typing:

npm start

1. 
2. Initialize Angular App in the previous command prompt by typing:

ng serve

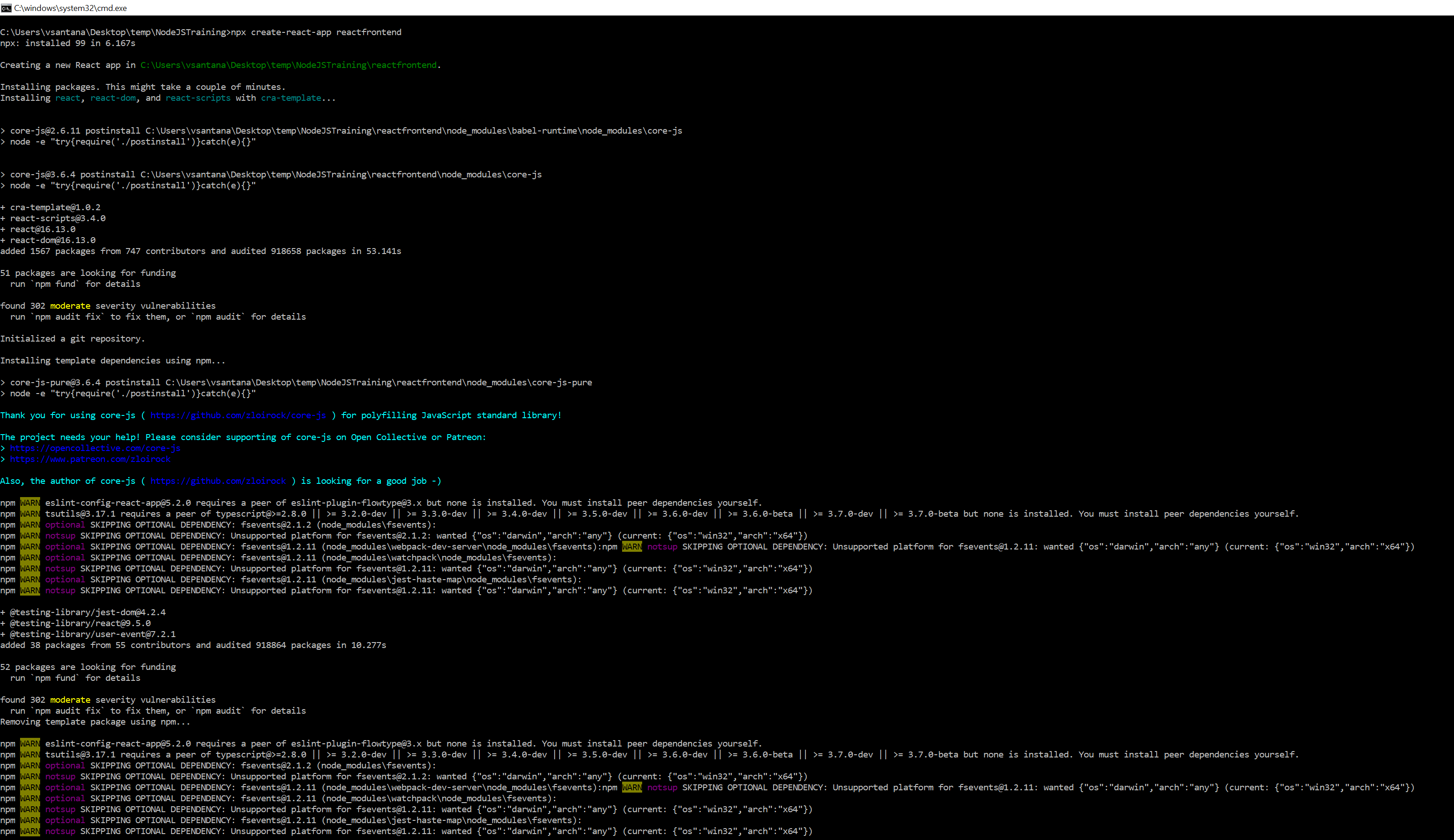
1. Open internet browser and navigate to <http://localhost:4200>
2. 
3. It’s expected to see the users you have defined in the ExpressAPI app.

### Lab 3 B: Creating a simple React app

In this lab, you will create a Reat app from a default template.

1. To start with React, you will need to create the app from a default template.
2. Open Windows command prompt go under your preferred location and type:

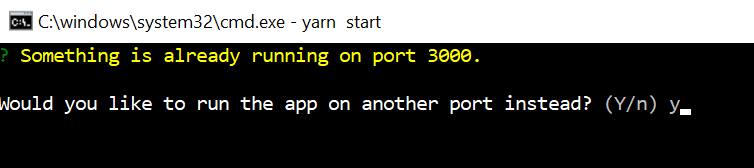
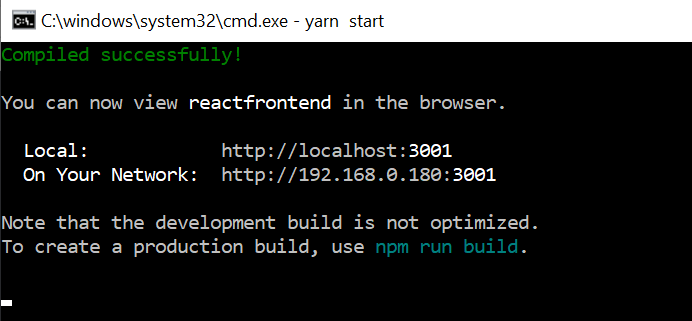
npx create-react-app reactfrontend

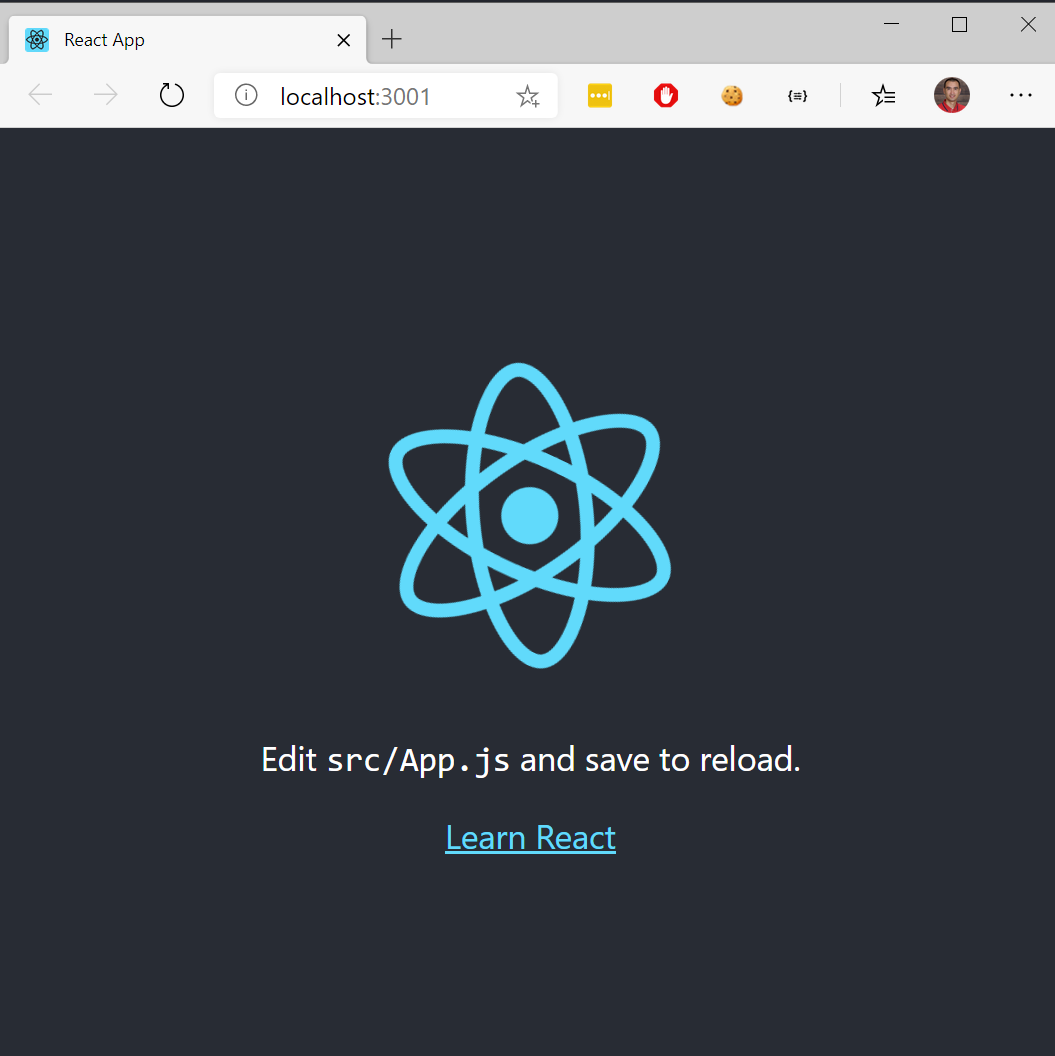
1. 
2. Once it has been created, change your directory to your new frontend app typing:

cd reactfrontend

1. React uses **yarn** instead of **npm** for installing and running commands.
2. Initialize React project typing:

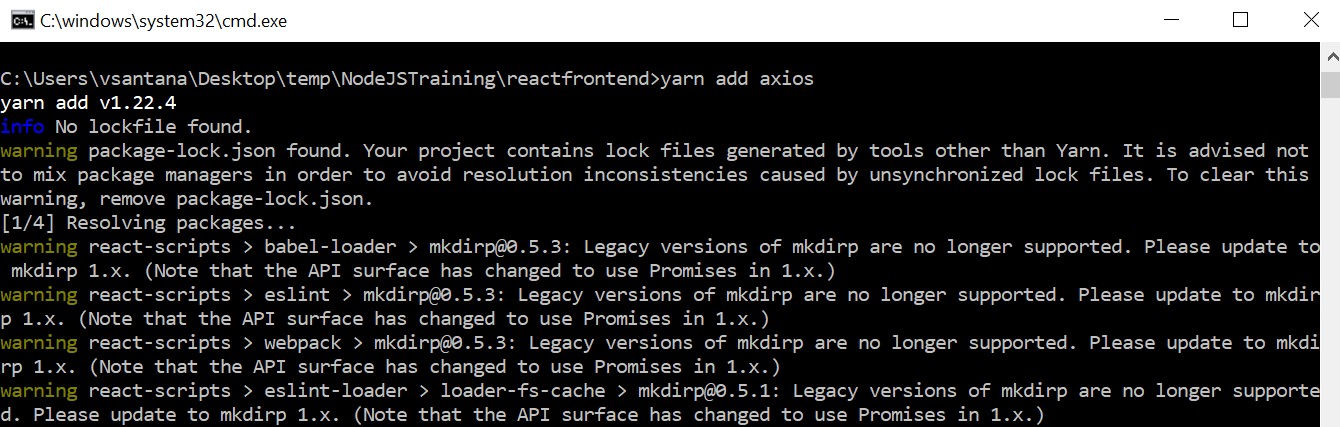
yarn start

1. Keep in mind that ExpressAPI is configured to use the TCP port 3000. React use the TCP port 3000 by default too. If you start the ExpressAPI first React will prompt you asking to run in another port. It’s important to have both of them running at the same time.
2. 
3. 
4. Open your internet browser and navigate to <http://localhost:3001/> and your default React app should be running.



1. Stop the server pressing **CTRL+C**
2. In React is necessary to install a new node module for HTTP requests: **axios**.
3. To install axios run the following command:

yarn add axios

1. 
2. Open Visual Studio Code and open the project folder.
3. Open the following file: **src/App.js**
4. This is the main React app’s component. In this file you will add the needed code to make an HTTP request to ExpressAPI and render the data.
5. By default, React creates it’s render data into a Function and exports this. We are going to convert this to a React.Component Class.
6. Replace the content of src/App.js with the following code:

import React from 'react';

import logo from './logo.svg';

import './App.css';

export default class App extends React.Component {

  render() {

    return (

      <div className="App">

        <header className="App-header">

          <img src={logo} className="App-logo" alt="logo" />

          <p>

            Edit <code>src/App.js</code> and save to reload.

          </p>

          <a

            className="App-link"

            href="https://reactjs.org"

            target="\_blank"

            rel="noopener noreferrer"

          >

            Learn React

          </a>

        </header>

      </div>

    );

  }

}

1. Now that you have the traditional React Component class, you will start making the changes in the project.

The intention is to make React access the ExpressAPI loading/fetching user API data. You need to import the previous installed module **axios** to make HTTP request to ExpressAPI.

1. To work with the data coming from ExpressAPI you need an array to store the data.
2. Add a constructor to the component React App and create a state object to store the users data coming from ExpressAPI.
3. Additionally, you need to call a lifecycle method to gather this data and set the users data.
4. You will use **componentDidMount()** to fetch the users data. Using **axios.get** to call ExpressAPI and set users as the response data (**res.data**) of the axios request.
5. Axios also supports other methods like **POST, DELETE, PATCH**, etc.
6. Once you have all the methods of gathering the data and storing it in a variable, you need to modify the render so that it displays the gathered info.
7. In the render function, list out all the user info.
8. You can map the users to a user object creating a **<li>** for each object. This is similar to **foreach** or **for** loops.
9. To accomplish all the above steps, replace the file **src/App.js** with the following code:

import React from 'react';

import logo from './logo.svg';

import './App.css';

import axios from 'axios';

export default class App extends React.Component {

  constructor(props) {

    super(props)

    this.state = {

      users: []

    }

  }

  componentDidMount() {

    axios.get('http://localhost:3000/users').then(

      res => {

        const users = res.data;

        this.setState({ users });

      });

  }

  render() {

    return (

      <div className="App">

        <header className="App-header">

          <img src={logo} className="App-logo" alt="logo" />

          <h3>Our Users</h3>

          <p>

            {this.state.users.map(user => <li>{user.user} - {user.job}</li>)}

          </p>

        </header>

      </div>

    );

  }

}

1. Please carefully review all changes to understand how all those steps was accomplished in this code.

### Lab 3 C: Creating a simple Vue app

In this lab, you will create a Vue app from a default template.

1. To start with Vue, you will need to install the Vue CLI via NPM.
2. Npm can install packages local to your project by reading the package.json file but it can also install global packages which will be available to all projects. In the next step you will install **@vue/cli** and make it available globally by using the parameter **-g** in the npm cli.
3. Open Windows command prompt go under your preferred location and type:

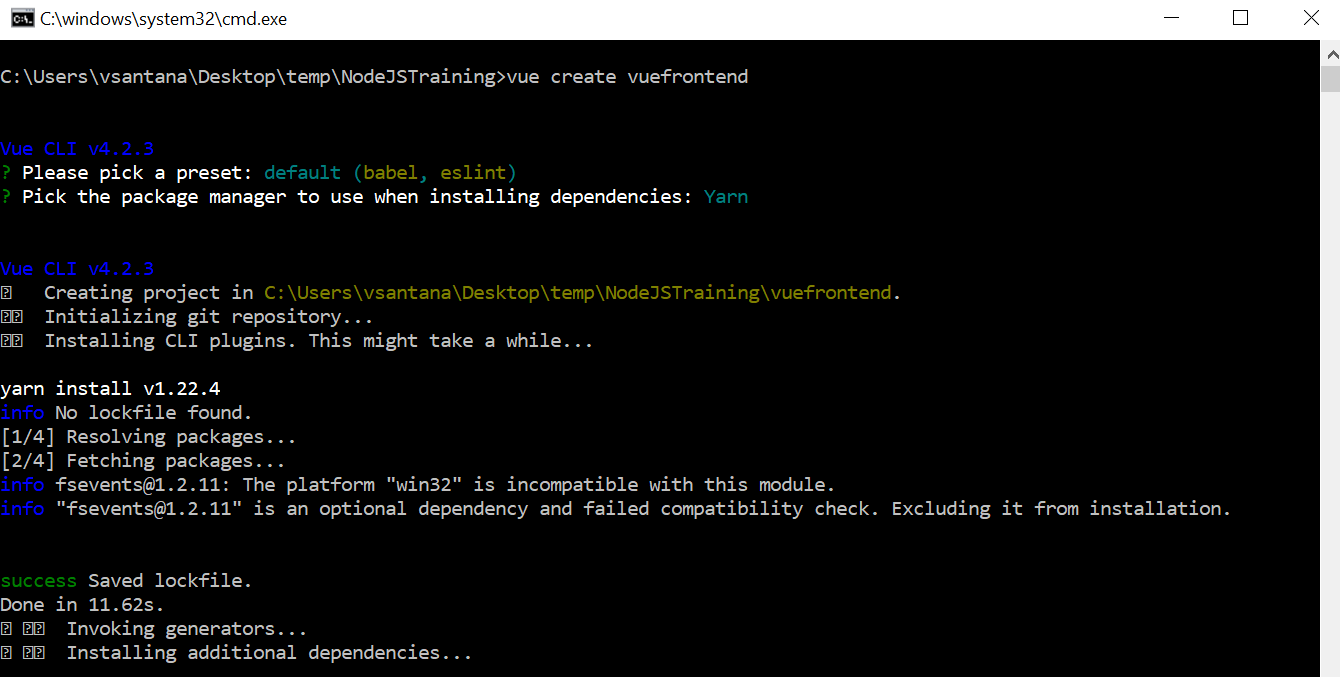
npm install -g @vue/cli



1. After the CLI is installed, you can use **vue** for any **Vue** commands.
2. Make sure you are under your preferred location/path in Windows command prompt.
3. The next command will create Vue app project folder.
4. Use the **default settings** and **yarn** for package management.
5. Create a new **Vue** app using the following command:

vue create vuefrontend

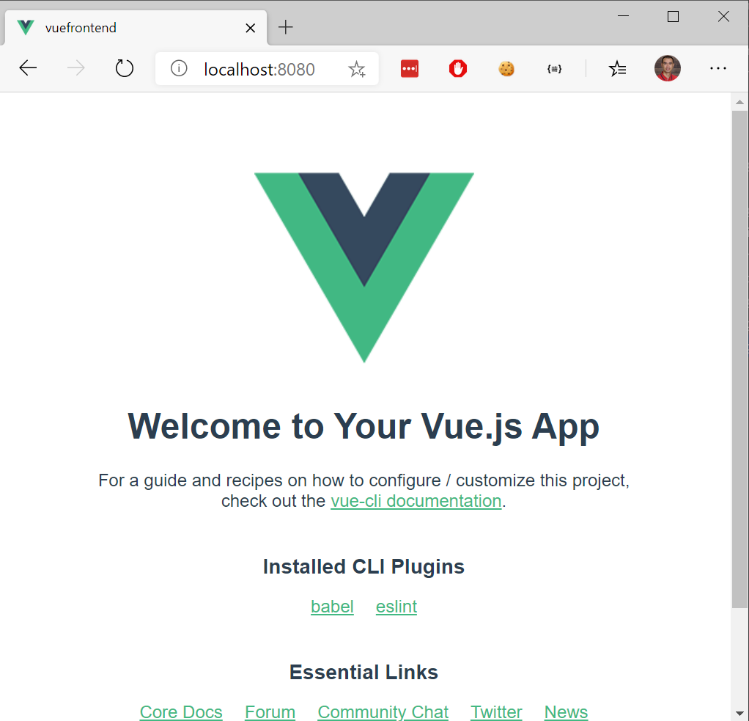
When prompted select **default settings** and yarn as package management.

1. 
2. Once it has been created, change your directory to your new frontend app by typing:

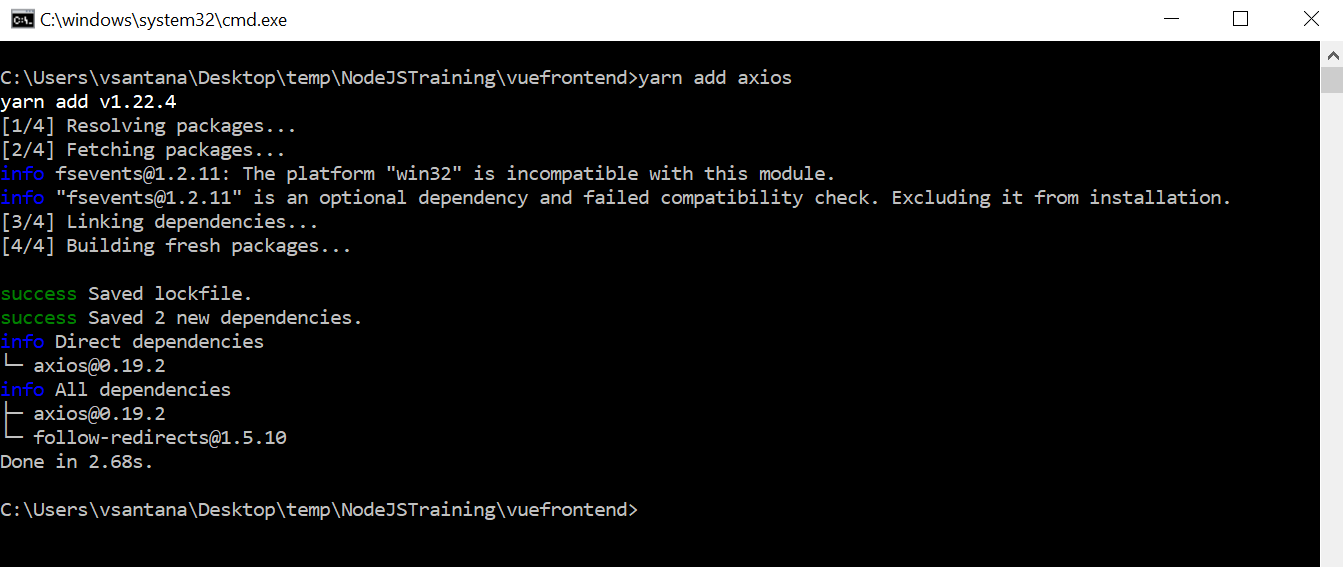
cd vuefrontend

1. Start the new Vue app typing:

yarn serve

1. Open the internet browser and navigate to <http://localhost:8080> and the default Vue app should be running.
2. 
3. Stop the server pressing **CTRL+C**
4. In Vue we will need to install a new node module for HTTP requests: **axios**. To do this, run the following command:

yarn add axios

1. 
2. Open Visual Studio Code and open the project folder.
3. Open the following file: **src/App.vue**
4. This is your main Vue’s starting file. Typically, we use this file to display a Vue component.
5. **Vue files** have three parts: **template, script, and style**. This handles everything inside one file as compared to other frameworks.
6. Create a new file called **Users.vue** inside of **src/components**
7. Add the following code to **Users.vue**:

<template>

    <div>

        <li :key="user.user" v-for="user in users">

            {{ user.user }} - {{ user.job }}

        </li>

    </div>

</template>

<script>

    import axios from 'axios';

    export default {

        data() {

            return {

                users: []

            }

        },

        mounted() {

            axios.get('http://localhost:3000/users').then(

                res => {

                    this.users = res.data;

            });

        }

    }

</script>

<style>

</style>

1. Code explanation:

**Script** section:

* First **axios** (add previously) is imported. **Axios** is used to make HTTP requests.
* Inside **export default** there are two functions **data** and **mounted.**
  + **Data** is the Vue instance variables for this component only.
  + **Mounted,** will execute the code specified when the DOM is mounted.

**Template** section:

* In this section a list item for each user inside users array is created.
* Additionally with Vue, is necessary to add a **:key** value so that we know what item to show.

1. Now is necessary modify main **App.vue** file to show the correct component.
2. Replace the whole content of **App.vue** with the following code:

<template>

  <div id="app">

    <img alt="Vue logo" src="./assets/logo.png">

    <Users/>

  </div>

</template>

<script>

import Users from './components/Users.vue'

export default {

  name: 'App',

  components: {

    Users

  }

}

</script>

<style>

#app {

  font-family: Avenir, Helvetica, Arial, sans-serif;

  -webkit-font-smoothing: antialiased;

  -moz-osx-font-smoothing: grayscale;

  text-align: center;

  color: #2c3e50;

  margin-top: 60px;

}

</style>

1. Code explanation:

**Script** section:

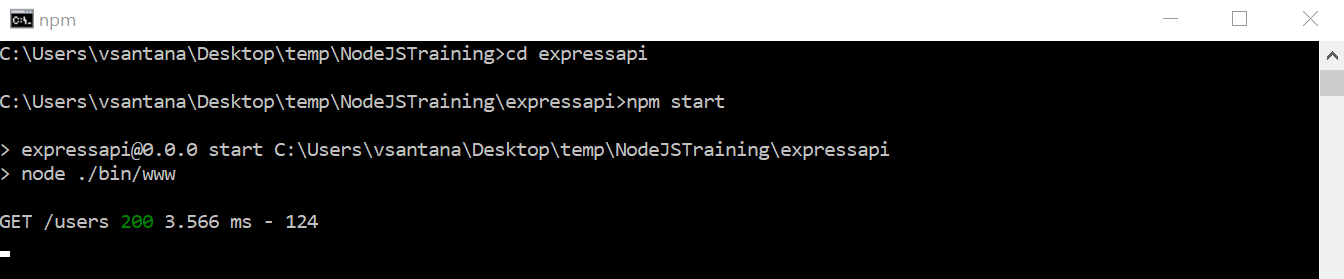
* You are importing the Users component.
* Exporting the Users component in the export default too.

**Template** section:

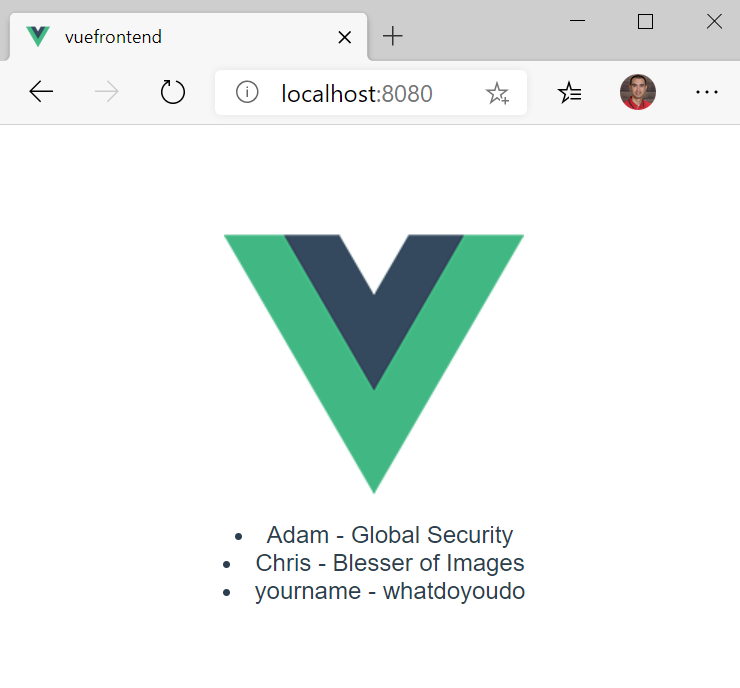
* You are loading Users.

1. It’s time to initialize the application Vue App working as the Front-End and the ExpressAPI working as the Back-End.
2. Open a new Windows Command prompt and initialize the ExpressAPI by typing:

npm start

1. 
2. Initialize Vue App in the previous command prompt by typing:

yarn serve

1. Open internet browser and navigate to <http://localhost:8080>
2. 
3. It’s expected to see the users you have defined in the ExpressAPI app.

# Module 3: Deployment to Azure WebApp

In this module you will learn about:

* Deploy the simple Express API to Azure Webapp
* Deploy Angular/React/Vue app to Azure WebApp

## Lab 1: Deploying Express API to Azure WebApp

In this lab you will deploy Express API previously created in **Module 2, Lab 2**.

You will accomplish the following steps:

* Install Azure CLI
* Create an Azure Resource Group
* Create a new App Service Plan and WebApp
* Enable Local Git deployment for this WebApp
* Deploy Express API using Local Git to Azure WebApp

All those steps will be done using Azure CLI

### Lab 1 A: Install Azure CLI

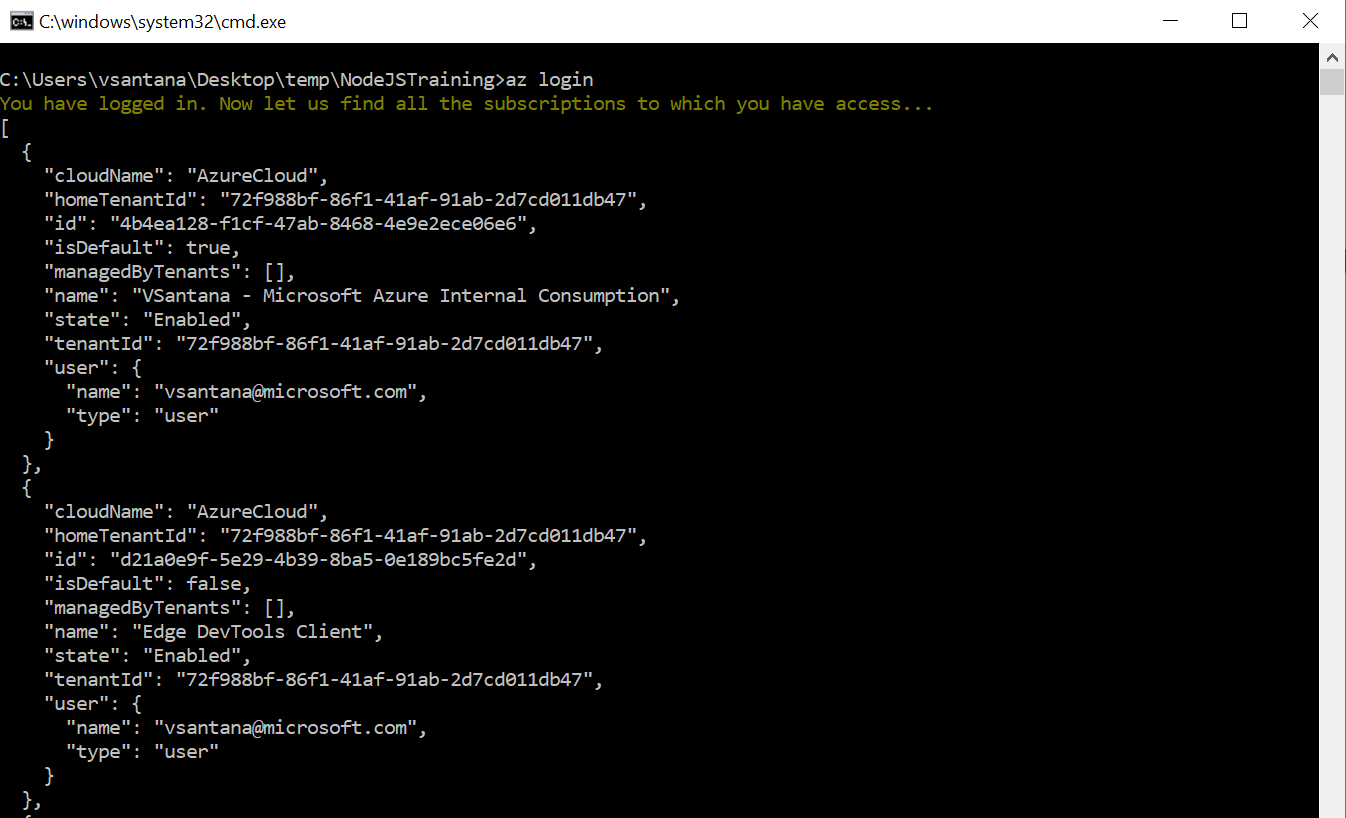
1. Open PowerShell as Administrator and execute the following command line:
2. Invoke-WebRequest -Uri https://aka.ms/installazurecliwindows -OutFile .\AzureCLI.msi; Start-Process msiexec.exe -Wait -ArgumentList '/I AzureCLI.msi /quiet'
3. It will silently download and install the latest Azure CLI.
4. Reference:

<https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest>

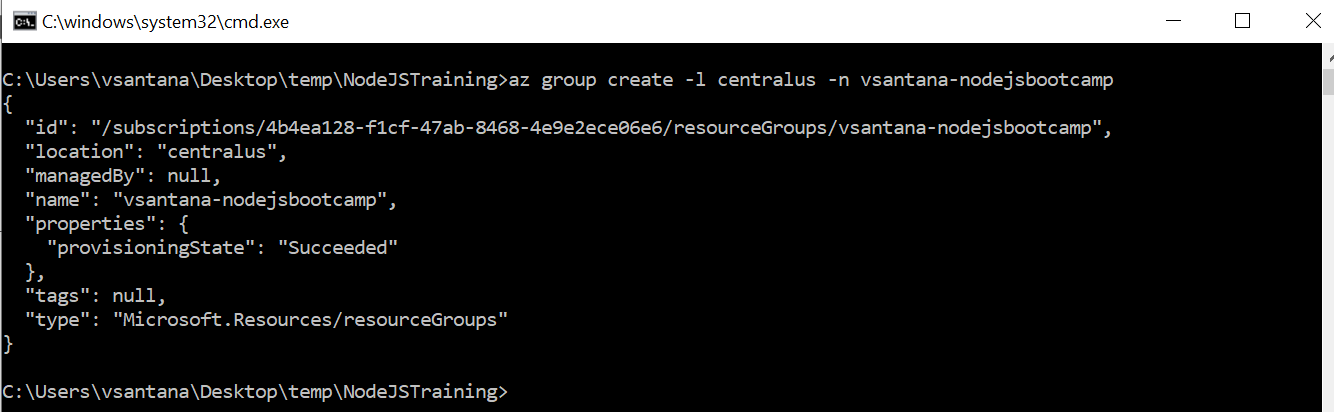
### Lab 1 B: Create Azure Resource Group

1. Open a new Windows command prompt and login to Azure using Azure CLI typing:

az login

1. It will open internet browser and ask your Microsoft credentials.
2. Once it’s done you will see an output like this:
3. 
4. Create Resource Group using the following command:

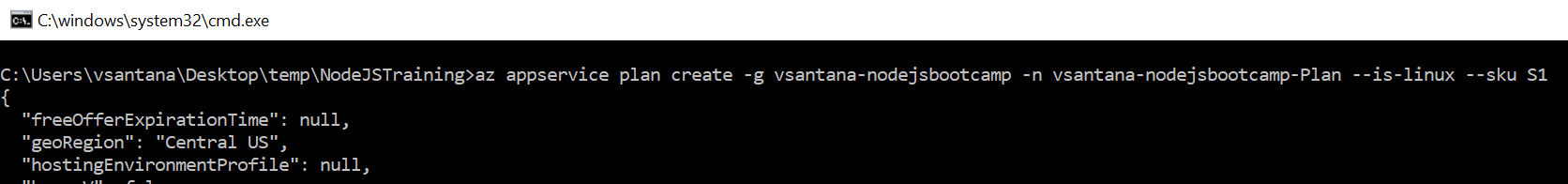
az group create -l centralus -n <alias>-nodejsbootcamp



### Lab 1 C: Create App Service Plan and Azure WebApp

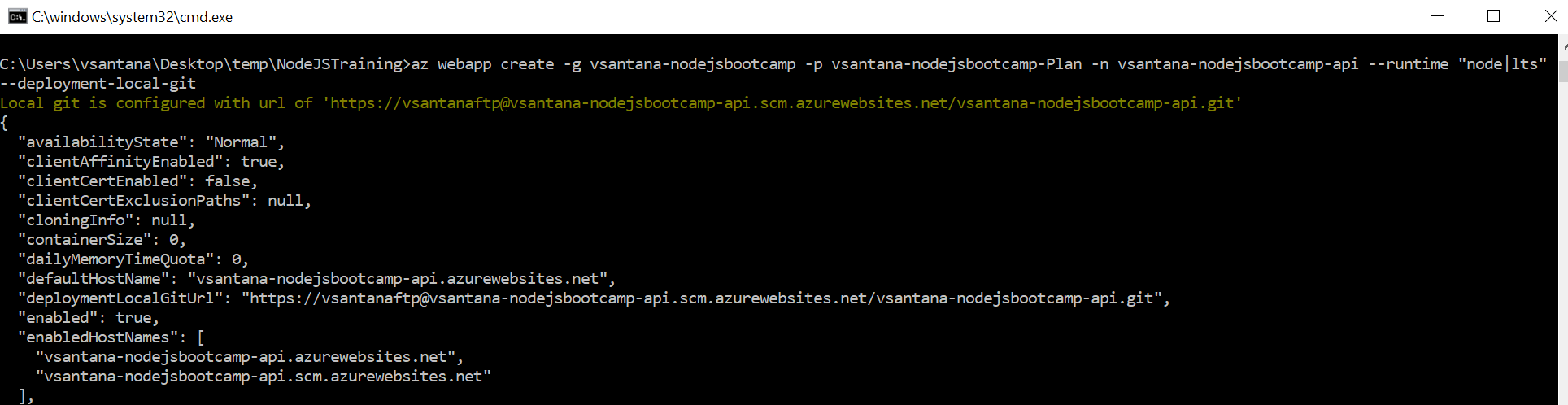
1. Using the previous Windows command prompt (already signed in) type the following command to create an Azure App Service Plan using a Linux instance.

az appservice plan create -g <alias>-nodejsbootcamp -n <alias>-nodejsbootcamp-Plan --is-linux --sku P1V2



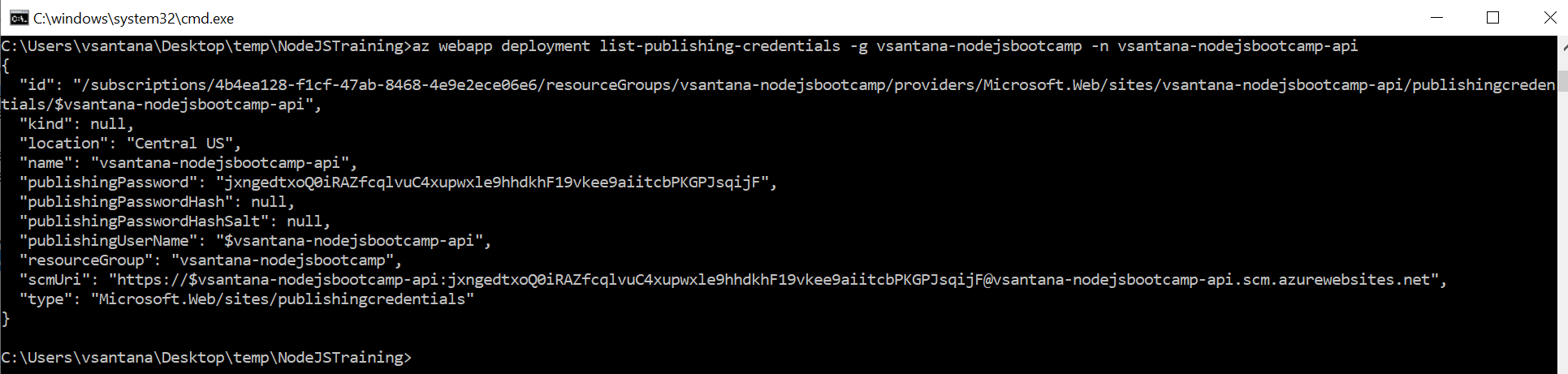
1. Type the following command to create an Azure WebApp using Local Git as the deployment option.

az webapp create -g <alias>-nodejsbootcamp -p <alias>-nodejsbootcamp-Plan -n <alias>-nodejsbootcamp-api --runtime "node|lts" --deployment-local-git



1. Now you need to get Local Git deployment Credentials and URL. Type the following command to list the URL that will be used in the deployment lab:

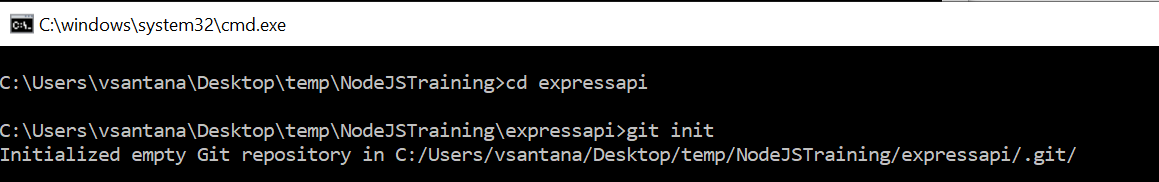
az webapp deployment list-publishing-credentials -g <alias>-nodejsbootcamp -n <alias>-nodejsbootcamp-api

1. 
2. In the previous output check the scmUri. In this Uri you have the **user name, password and Local Git url.**
3. Save the output.

### Lab 1 D: Deploy Express API using Local Git to Azure WebApp

1. Using Windows command prompt make sure you are in the directory of ExpressAPI project. If not use the **cd** to move to the right location.
2. You will now initiate git and add a remote endpoint to push the project content.
3. Type the following commands to deploy Express API to Azure WebApp using Local Git:

git init

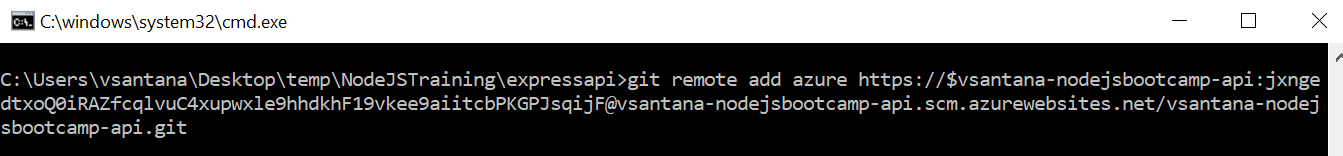


**Now it’s time to use the saved Uri from previous lab and add the app name at the end.**

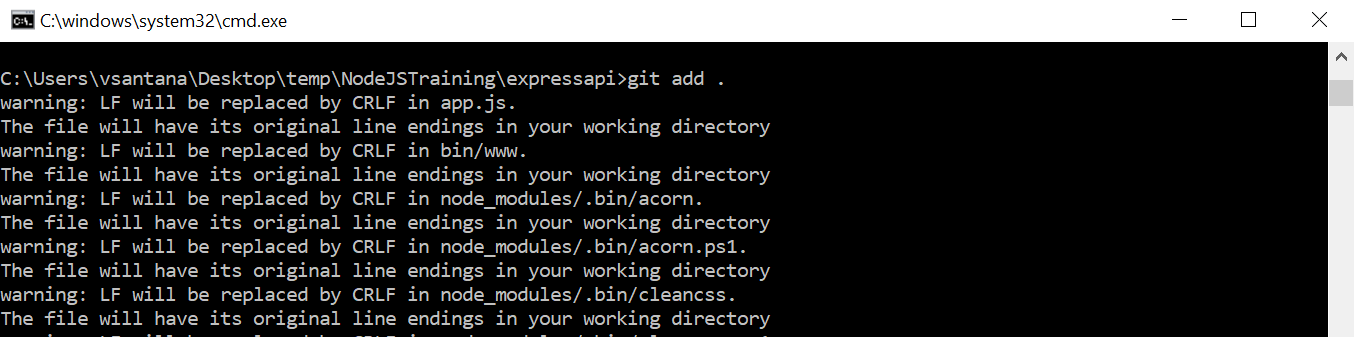
**Example:**

**https://$<alias>-nodejsbootcamp-api:<password>@<alias>-nodejsbootcamp-api.scm.azurewebsites.net/<alias>-nodejsbootcamp-api.git**

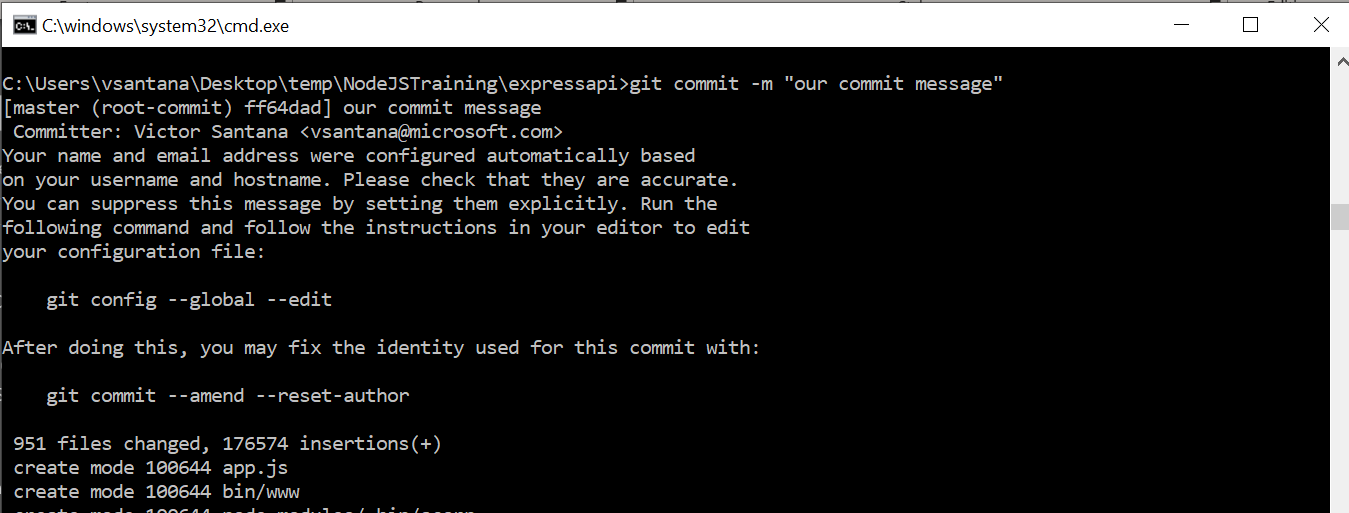
git remote add azure *https://$<alias>-nodejsbootcamp-api:<password>@<alias>-nodejsbootcamp-api.scm.azurewebsites.net/<alias>-nodejsbootcamp-api.git*



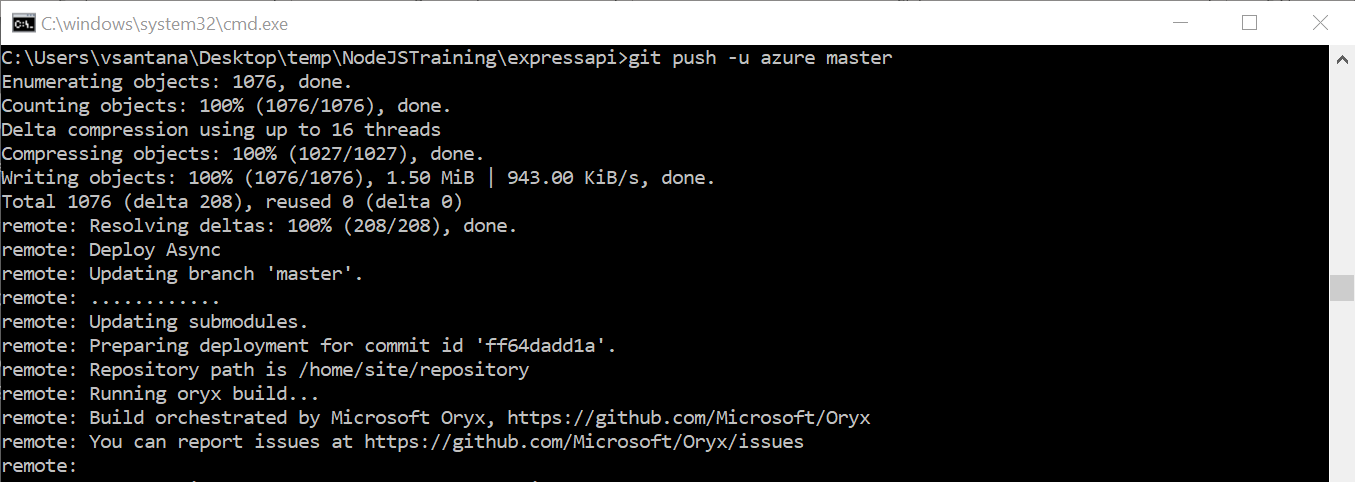
git add .



git commit -m "our commit message"



git push -u azure master



1. Command summary:

First command Git was initialized in the project.

Second added our remote endpoint labeled as azure.

Third add all the files into the staged changes.

Fourth create a commit to all staged changes and add a commit message.

The last command you are pushing the commit and setting the upstream to azure/master since this is the first commit. All other future commits can be done by just using git push.

1. Once the deployment process has finished. Open internet browser and navigate to your web app at: https://<alias>-nodejsbootcamp-api.azurewebsites.net
2. Hit Express API endpoint for your Users: https://<alias>-nodejsbootcamp-api.azurewebsites.net/users

## Lab 2: Deploy Angular/React/Vue app to Azure WebApp

In this lab you will deploy one of the Front-End Web Application previously created in **Module 2, Lab 3**.

In this lab you will accomplish the following tasks:

* Replace the Express API backend endpoint in the desired Front-End application you choose.
* Create a new WebApp to host the Front-End Web Application.
* Enable Local Git deployment for this WebApp
* Deploy Front-End Web Application using Local Git to Azure WebApp

All those steps will be done using Azure CLI

### Lab 2 A: Updating Back-End endpoint to Azure WebApp Back-End URL

1. Before start deploying, you need to update the backend endpoints from localhost to the Express API app deployed in the previous lab to Azure App Service.

For Angular:

Navigate and update /src/app/app.service.ts

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

@Injectable({

  providedIn: 'root'

})

export class AppService {

  getUsers() {

    return this.http.get('http://<alias>-nodejsbootcamp-api.azurewebsites.net/users');

  }

  constructor(

    private http: HttpClient

  ) {}

}

For React:

Navigate and update /src/App.js

import React from 'react';

import logo from './logo.svg';

import './App.css';

import axios from 'axios';

export default class App extends React.Component {

  constructor(props) {

    super(props)

    this.state = {

      users: []

    }

  }

  componentDidMount() {

    axios.get('http://<alias>-nodejsbootcamp-api.azurewebsites.net/users').then(

      res => {

        const users = res.data;

        this.setState({ users });

      });

  }

  render() {

    return (

      <div className="App">

        <header className="App-header">

          <img src={logo} className="App-logo" alt="logo" />

          <h3>Our Users</h3>

          <p>

            {this.state.users.map(user => <li>{user.user} - {user.job}</li>)}

          </p>

        </header>

      </div>

    );

  }

}

For Vue:

Navigate and update /src/components/Users.vue

<template>

    <div>

        <li :key="user.user" v-for="user in users">

            {{ user.user }} - {{ user.job }}

        </li>

    </div>

</template>

<script>

    import axios from 'axios';

    export default {

        data() {

            return {

                users: []

            }

        },

        mounted() {

            axios.get('http://<alias>-nodejsbootcamp-api.azurewebsites.net/users').then(

                res => {

                    this.users = res.data;

            });

        }

    }

</script>

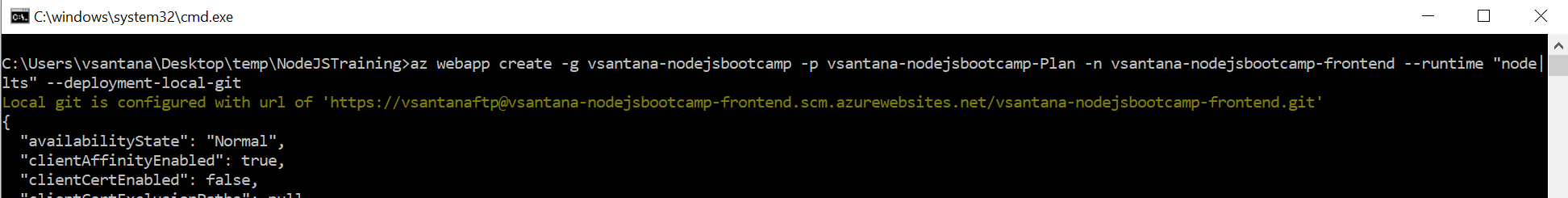
<style>

</style>

### Lab 2 B: Create Azure WebApp to host Front-End Application

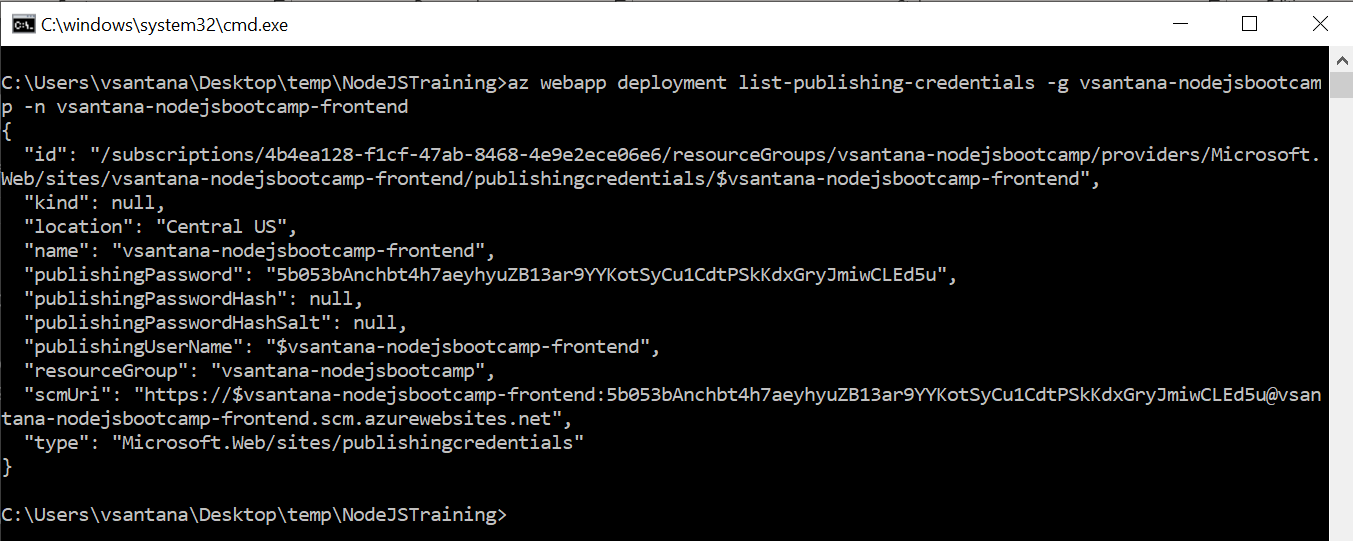
1. Type the following command to create an Azure WebApp using Local Git as the deployment option.

az webapp create -g <alias>-nodejsbootcamp -p <alias>-nodejsbootcamp-Plan -n <alias>-nodejsbootcamp-frontend --runtime "node|lts" --deployment-local-git



1. Now you need to get Local Git deployment Credentials and URL. Type the following command to list the URL that will be used in the deployment lab:

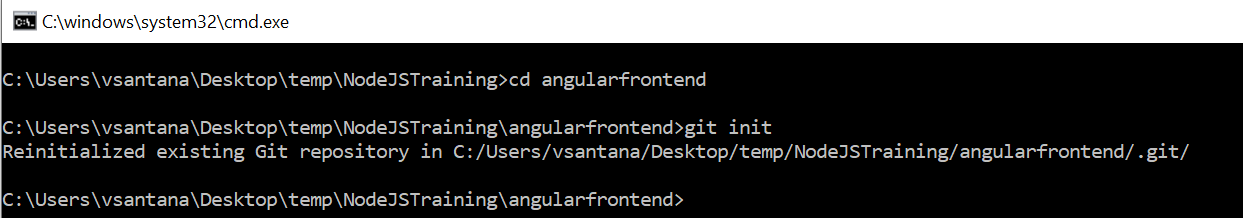
az webapp deployment list-publishing-credentials -g <alias>-nodejsbootcamp -n <alias>-nodejsbootcamp-frontend

1. 
2. In the previous output check the scmUri. In this Uri you have the **user name, password and Local Git url.**
3. Save the output.

### Lab 2 C: Deploy Angular/Reach/Vue Front-End using Local Git to Azure WebApp

1. Using Windows command prompt make sure you are in the desired directory of the Front-End project like **Angular/React/Vue**. If not use the **cd** to move to the right location.
2. You will now initiate git and add a remote endpoint to push the project content.
3. Type the following commands to deploy Express API to Azure WebApp using Local Git:

git init

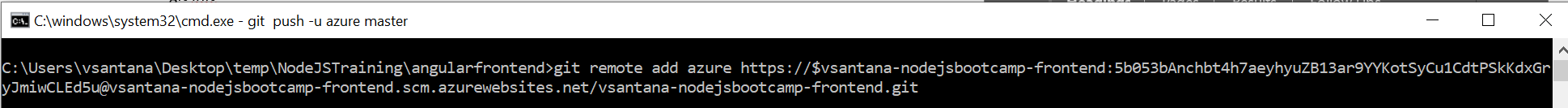


**Now it’s time to use the saved Uri from previous lab and add the app name at the end.**

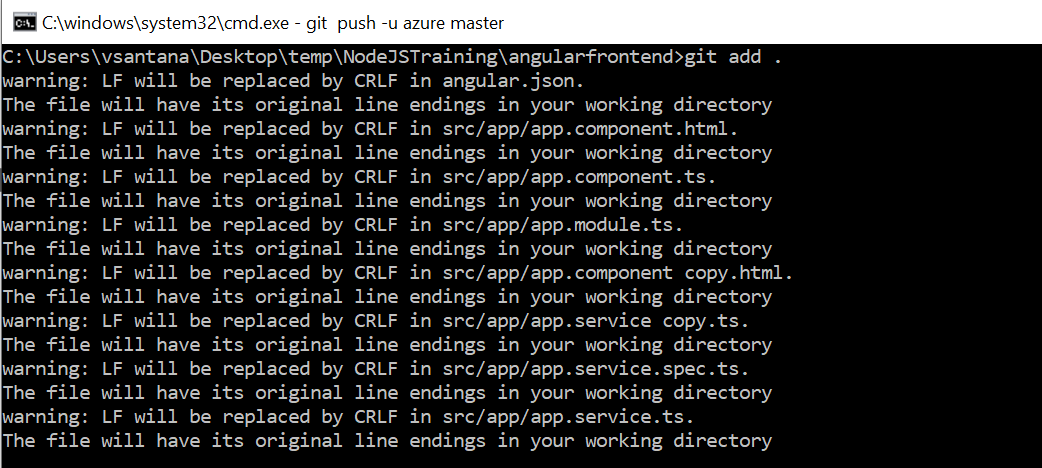
**Example:**

**https://$<alias>-nodejsbootcamp-frontend:<password>@<alias>-nodejsbootcamp-api.scm.azurewebsites.net/<alias>-nodejsbootcamp-frontend.git**

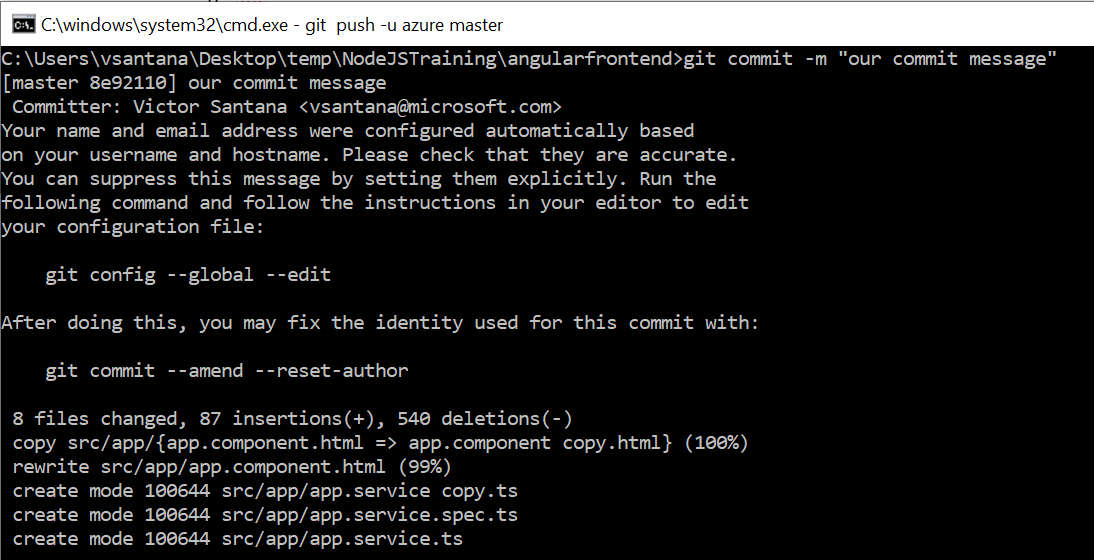
git remote add azure *https://$<alias>-nodejsbootcamp-api:<password>@<alias>-nodejsbootcamp-frontend.scm.azurewebsites.net/<alias>-nodejsbootcamp-frontend.git*



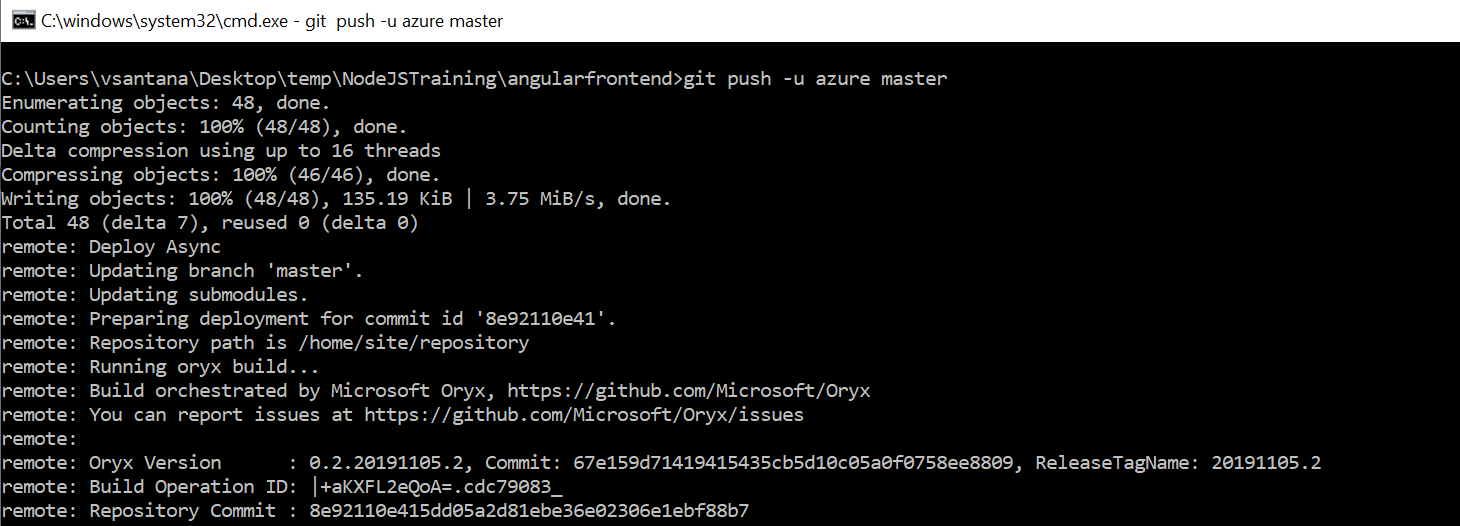
git add .



git commit -m "our commit message"



git push -u azure master



1. Command summary:

First command Git was initialized in the project.

Second added our remote endpoint labeled as azure.

Third add all the files into the staged changes.

Fourth create a commit to all staged changes and add a commit message.

The last command you are pushing the commit and setting the upstream to azure/master since this is the first commit. All other future commits can be done by just using git push.

1. During the deployment process all the npm modules are installed and the build process is completed.

***Note:*** *Azure WebApp uses Oryx to build the uploaded code during all deployments. Oryx will evaluate the deployed code if there is a* ***package.json*** *if so it will run npm build and prepare the initialization script based in the start entry in the package.json file. Otherwise it will look for* ***bin/www, server.js, app.js, index.js, hostingstart.js*** *files or if the* ***Startup Command*** *is setup and if found it will prepare the initialization script based in those. For the launch process it also generates a dynamic script with all the steps that needs to be executed to start the application. All Front-End frameworks has it own CLI and compilation tools that is not available in the final Docker Container in Azure WebApp but Oryx build has a build container with all compilation tools for all runtimes which also install all* ***devDependencies*** *from package.json (.Net Core, Node.JS, Python, Java).*

*For more information:*

* [*https://github.com/microsoft/Oryx/blob/master/src/startupscriptgenerator/src/node/scriptgenerator.go*](https://github.com/microsoft/Oryx/blob/master/src/startupscriptgenerator/src/node/scriptgenerator.go)*.*
* [*https://github.com/microsoft/Oryx/blob/master/doc/architecture.md*](https://github.com/microsoft/Oryx/blob/master/doc/architecture.md)
* <https://github.com/microsoft/Oryx/blob/master/doc/runtimes/nodejs.md>

1. Front-End frameworks just generate static content that are served. Once compiled it does not include a Web Server to serve the data. For that reason, you need to make a change in the Start-up command of the WebApp to setup PM2 to work as a Web Server to serve the static content.

For more information:

* <https://pm2.keymetrics.io/docs/usage/expose/>

1. Execute the following command to change the Startup command according to the Front-End framework you choose:

Angular:

az webapp config set -g <alias>-nodejsbootcamp -n <alias>-nodejsbootcamp-frontend --startup-file "pm2 serve /home/site/wwwroot/dist/angularfrontend --no-daemon"

React:

az webapp config set -g <alias>-nodejsbootcamp -n <alias>-nodejsbootcamp-frontend --startup-file "pm2 serve /home/site/wwwroot/build --no-daemon"

Vue:

az webapp config set -g <alias>-nodejsbootcamp -n <alias>-nodejsbootcamp-frontend --startup-file "pm2 serve /home/site/wwwroot/dist --no-daemon"

1. Once the deployment process has finished. Open internet browser and navigate to your web app at: http://<alias>-nodejsbootcamp-frontend.azurewebsites.net
2. You now have a full application Front-End and Back-End up and running on Azure App Services.

# Module 4: Troubleshooting

In this module you will practice what you have learned:

In this lab you will go through some troubleshooting to cover a few common things you will face when customers try and deploy and run an application build with Node.JS.

## Lab 1: Deploy and Troubleshoot an application from a Git repository

In this lab you will clone a Git repository from GitHub and deploy to a Azure WebApp.

The intention of this is for you practice what you have learned without a complete guidance.

### Lab 1 A: Tasks to be accomplished

1. Clone the Git repository from GitHub. You can do it using the following command line with the provided repository:

git clone <https://github.com/chmald/nodejsbootcamp-module4>

1. Create an Azure WebApp using Node LTS on Linux using Local git as a deployment type:

az webapp create -g <alias>-nodejsbootcamp -p <alias>-nodejsbootcamp-Plan -n <alias>-nodejsbootcamp-module4 --runtime "node|lts" --deployment-local-git

1. Now you need to get Local Git deployment Credentials and URL. Type the following command to list the URL that will be used in the deployment lab:

az webapp deployment list-publishing-credentials -g <alias>-nodejsbootcamp -n <alias>-nodejsbootcamp-module4

1. In the previous output check the scmUri. In this Uri you have the **user name, password and Local Git url.**
2. Save the output.
3. Now it’s time to use the saved Uri from previous step and add the app name at the end.

Example:

https://$<alias>-nodejsbootcamp-module4:<password>@<alias>-nodejsbootcamp-module4.scm.azurewebsites.net/<alias>-nodejsbootcamp-module4.git

git remote add azure https://$<alias>-nodejsbootcamp-module4:<password>@<alias>-nodejsbootcamp-module4.scm.azurewebsites.net/<alias>-nodejsbootcamp-module4.git

1. Deploy the cloned repository to your Azure WebApp.

git push -u azure master

1. Change the start up command:

az webapp config set -g <alias>-nodejsbootcamp -n <alias>-nodejsbootcamp-module4 --startup-file "pm2 serve /home/site/wwwroot/dist --no-daemon"

1. Once it’s deployed try to access it.
2. Were you able to see the users coming from Express API?
3. Now use all available tools to troubleshooting:

* Observer;
* AppLens;
* Logs;
* Etc…

1. Try to identify the error using all available troubleshooting tools.
2. There are two problems in this code.
   1. The axios module is not been imported at /src/components/Users.vue at line 10:

import axios from 'axios';

* 1. The endpoint in /src/components/Users.vue at line 18 is point to a wrong place:

axios.get('http://<alias>-nodejsbootcamp-api.azurewebsites.net/users').then(

1. Replace both lines and re-deploy.

git add .

git commit -m "Fix axios and endpoint"

git push -u azure master

1. Open internet browser and access the WebApp again.

## Lab 2: Deploy a Flatris game

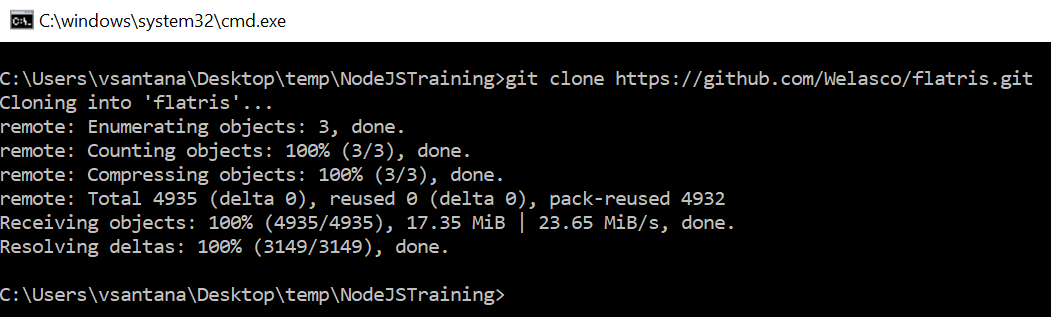
In this lab you will clone a Git repository from GitHub and deploy to an Azure WebApp.

This is a real-world case where a project doesn’t work as it’s in Azure WebApp. You must figure-out why this project doesn’t work in Azure WebApp and fix it.

### Lab 2 A: Tasks to be accomplished

1. Test the project executing it locally first. That’s one of the basics steps you should always try.
2. Clone the Git repository from GitHub. You can do it using the following command line with the

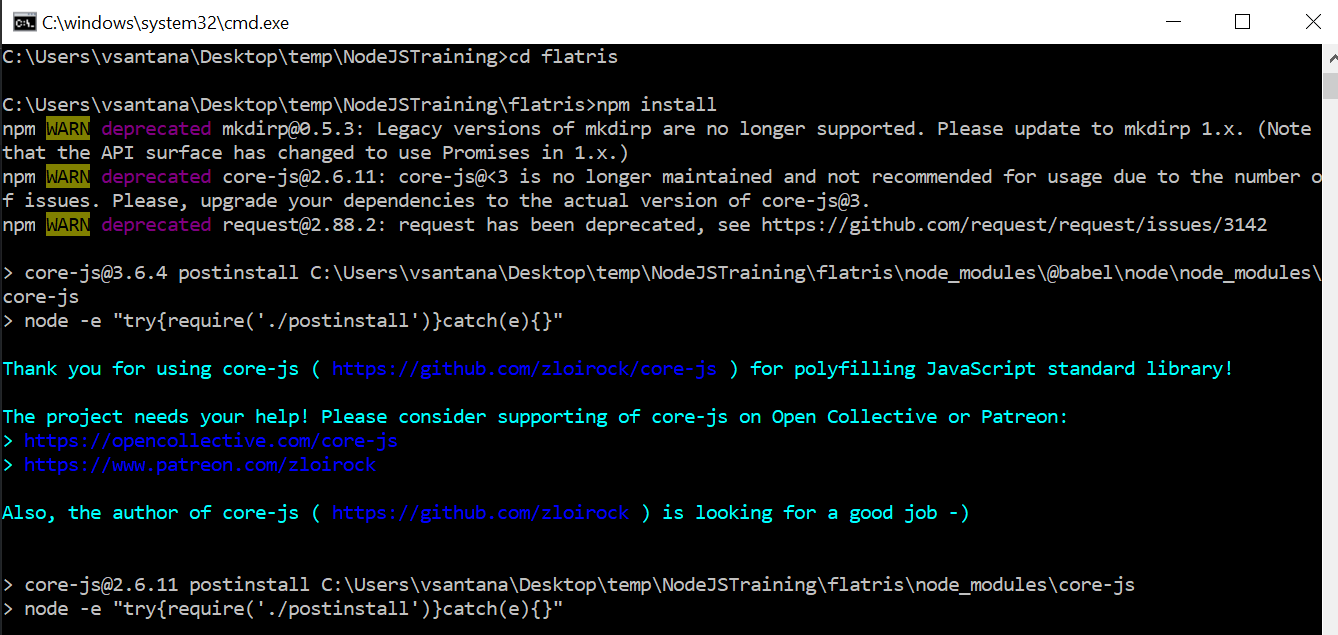
git clone https://github.com/Welasco/flatris.git

1. 
2. Once it has been cloned, change your directory to the projects folder:

cd flatris

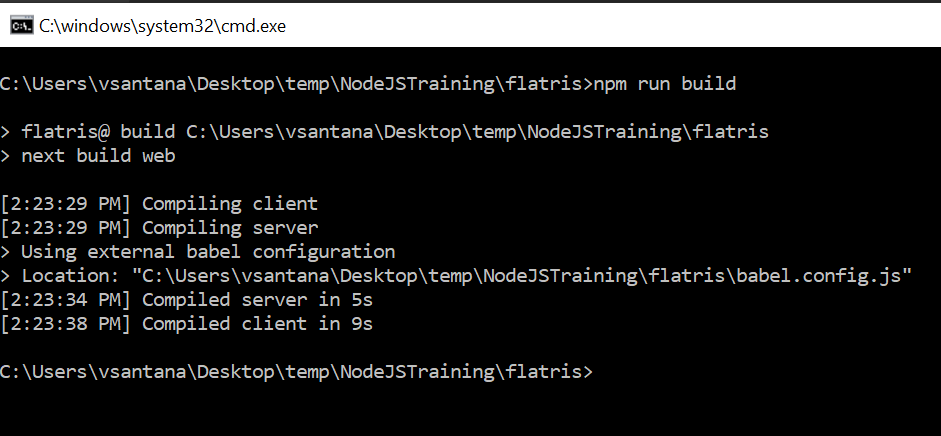
1. Install all the dependencies typing:

npm install

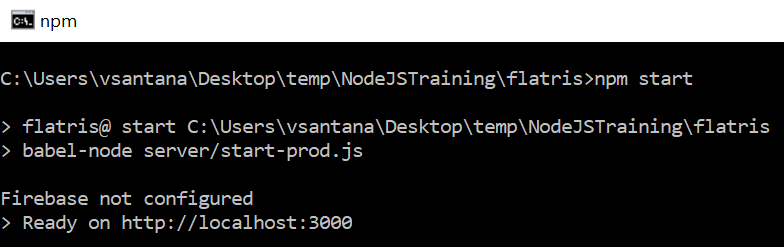
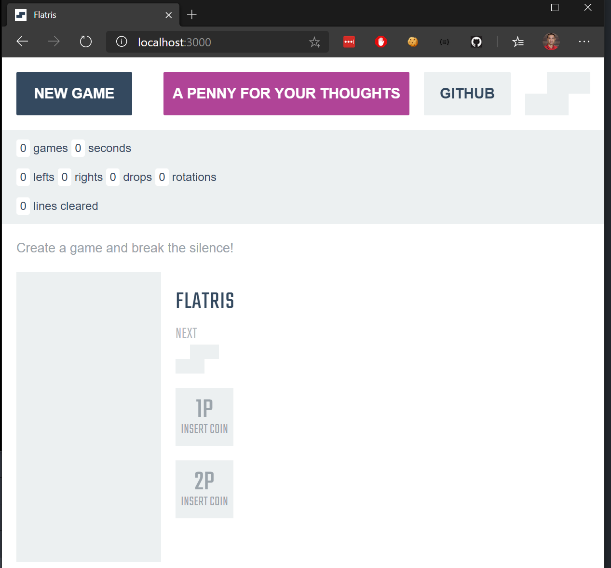


1. Build the project running the following command:

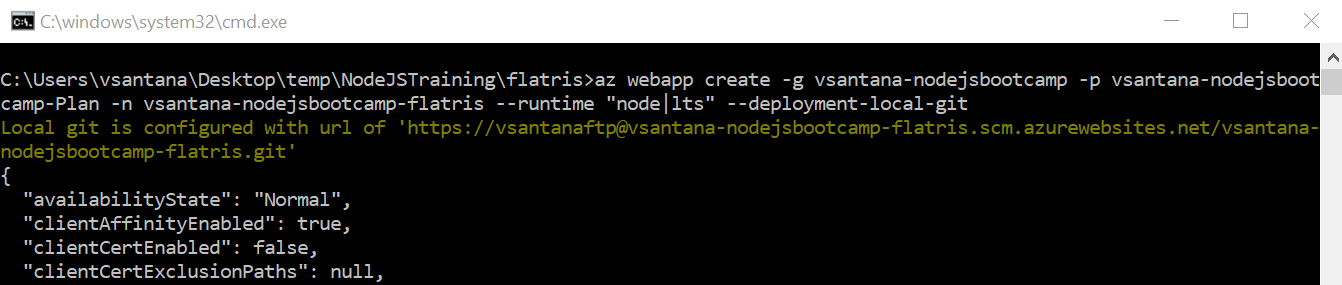
npm run build

1. 
2. Start the project using following command:

npm start

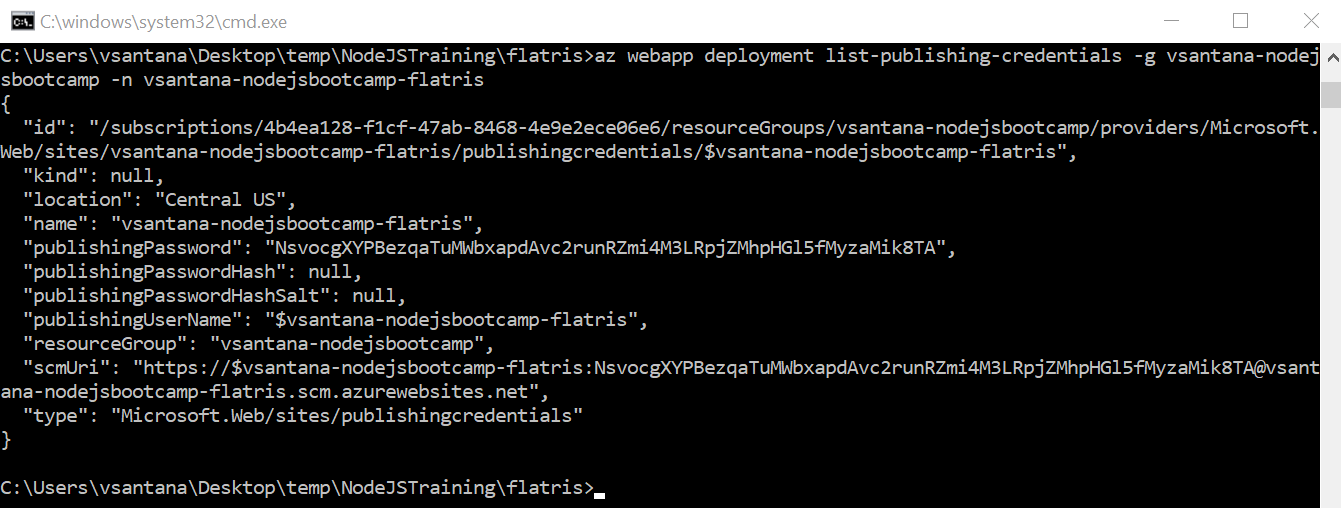
1. 
2. Open internet browser and navigate to <http://localhost:3000>
3. 
4. Stop the server pressing **CTRL+C**
5. Now that you have cloned the repository deploy it to Azure WebApp.
6. Create a new Azure WebApp using the following command:

az webapp create -g <alias>-nodejsbootcamp -p <alias>-nodejsbootcamp-Plan -n <alias>-nodejsbootcamp-flatris --runtime "node|lts" --deployment-local-git



1. Now you need to get Local Git deployment Credentials and URL. Type the following command to list the URL that will be used in the deployment lab:

az webapp deployment list-publishing-credentials -g <alias>-nodejsbootcamp -n <alias>-nodejsbootcamp-flatris

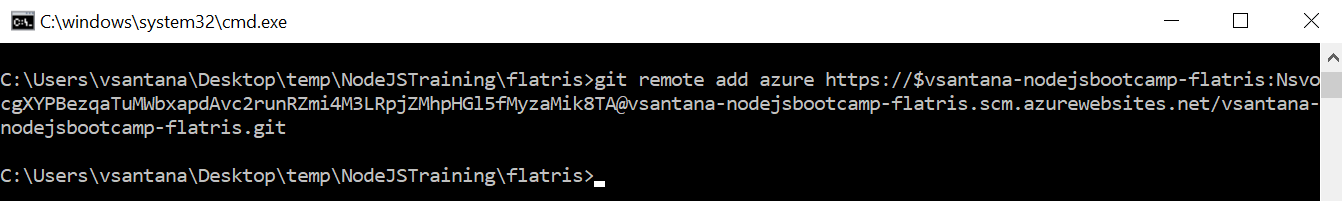


1. In the previous output check the scmUri. In this Uri you have the **user name, password and Local Git url.**
2. Save the output.
3. Now it’s time to use the saved Uri from previous step and add the app name at the end.

Example:

https://$<alias>-nodejsbootcamp-flatris:<password>@<alias>-nodejsbootcamp-flatris.scm.azurewebsites.net/<alias>-nodejsbootcamp-flatris.git

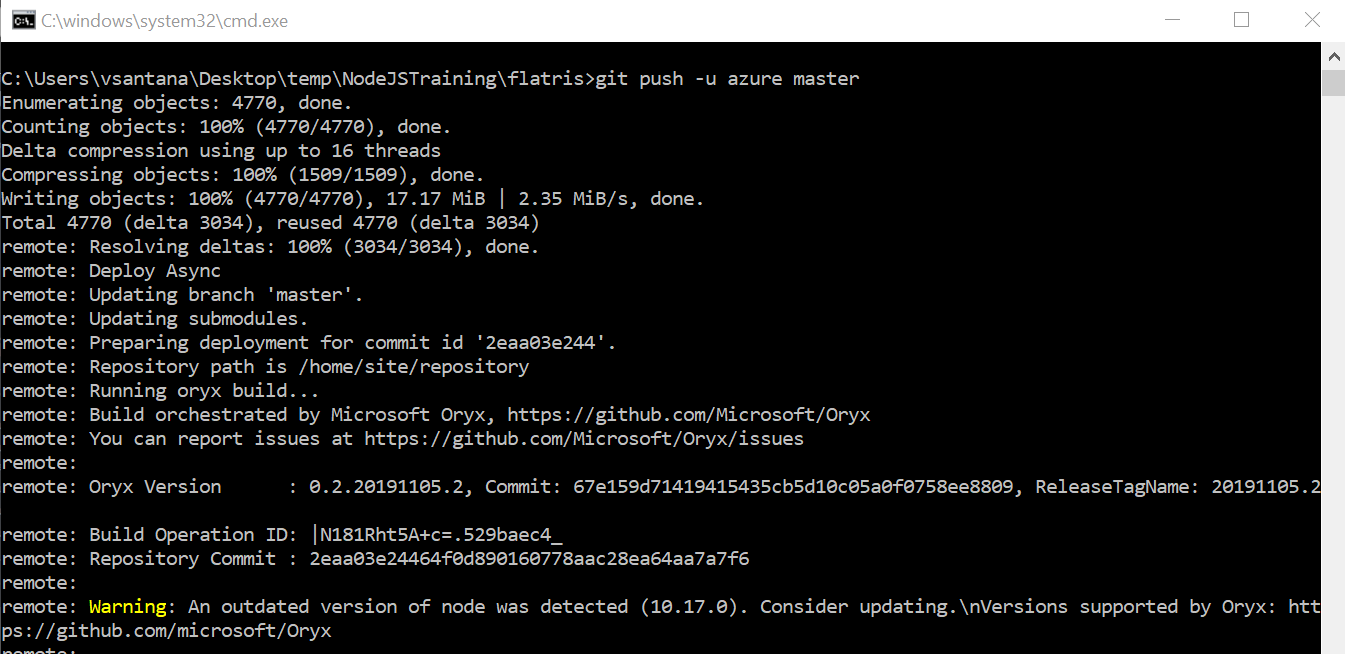
git remote add azure https://$<alias>-nodejsbootcamp-flatris:<password>@<alias>-nodejsbootcamp-flatris.scm.azurewebsites.net/<alias>-nodejsbootcamp-flatris.git



1. Deploy the code to Azure WebApp using the following command:

***Note:*** *This is the step that trigger Orix to prepare the code running building steps.*

git push -u azure master

1. 

***Note:*** *Check the output very carefully to understand the steps that Oryx is taking.*

1. Once it’s done the Azure WebApp will restarted.
2. Open internet browser and try to access the Azure WebApp navigating to:

http://<alias>-nodejsbootcamp-flatris.azurewebsites.net

1. Were you able to access the WebApp?

There is a problem that is preventing the WebApp to work.

1. It’s time to Troubleshoot:
2. Once it’s deployed access it and use all available tools to troubleshoot it:

* Observer;
* AppLens;
* Logs;
* Kudo Logs;
* Etc…

1. Please take your time to review the logs carefully.
2. **In the next page you see what is relevant for this issue.**
3. Open internet browser and access Kudo Console for your Azure WebApp:

https://<alias>-nodejsbootcamp-flatris.scm.azurewebsites.net/

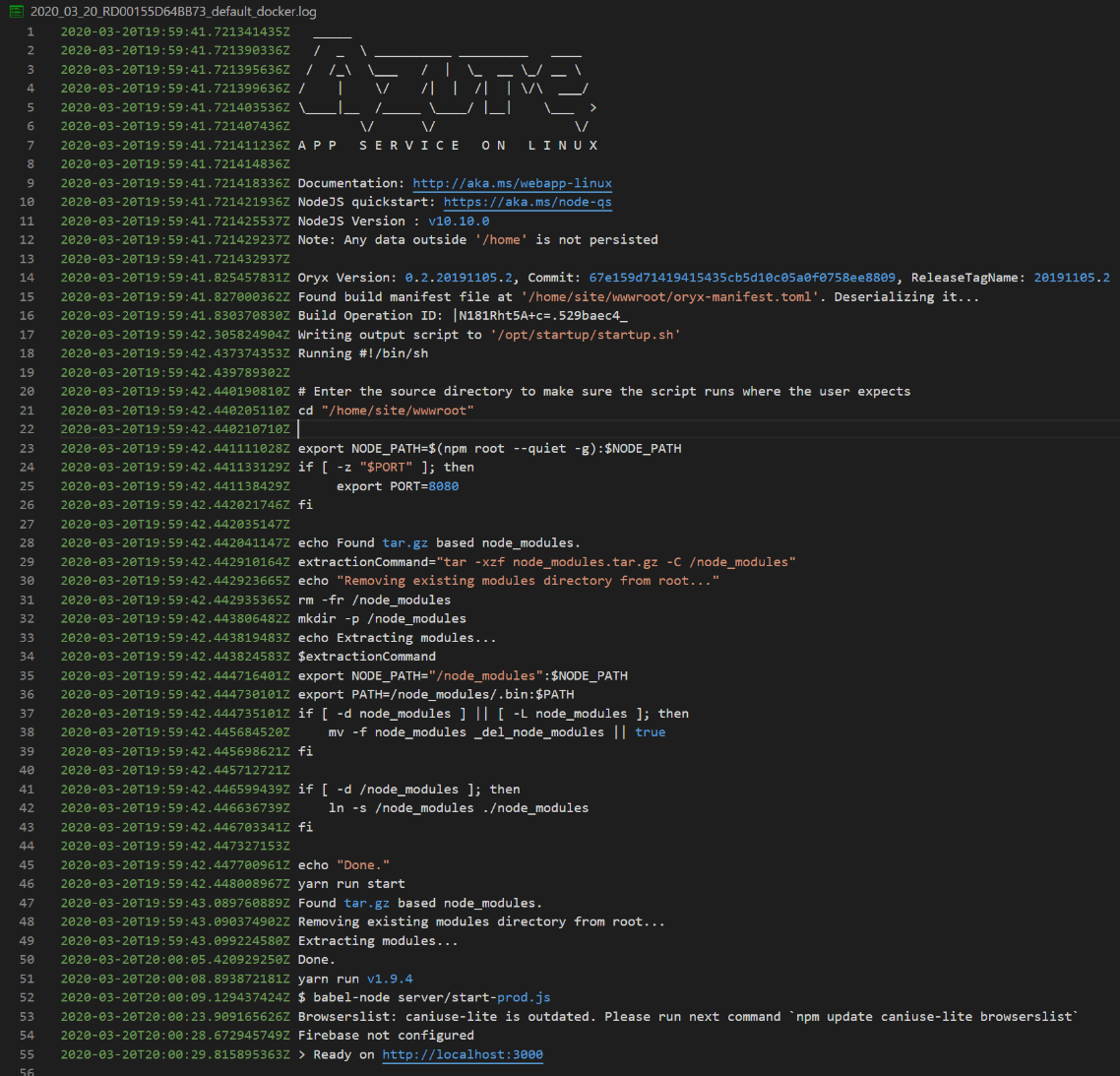
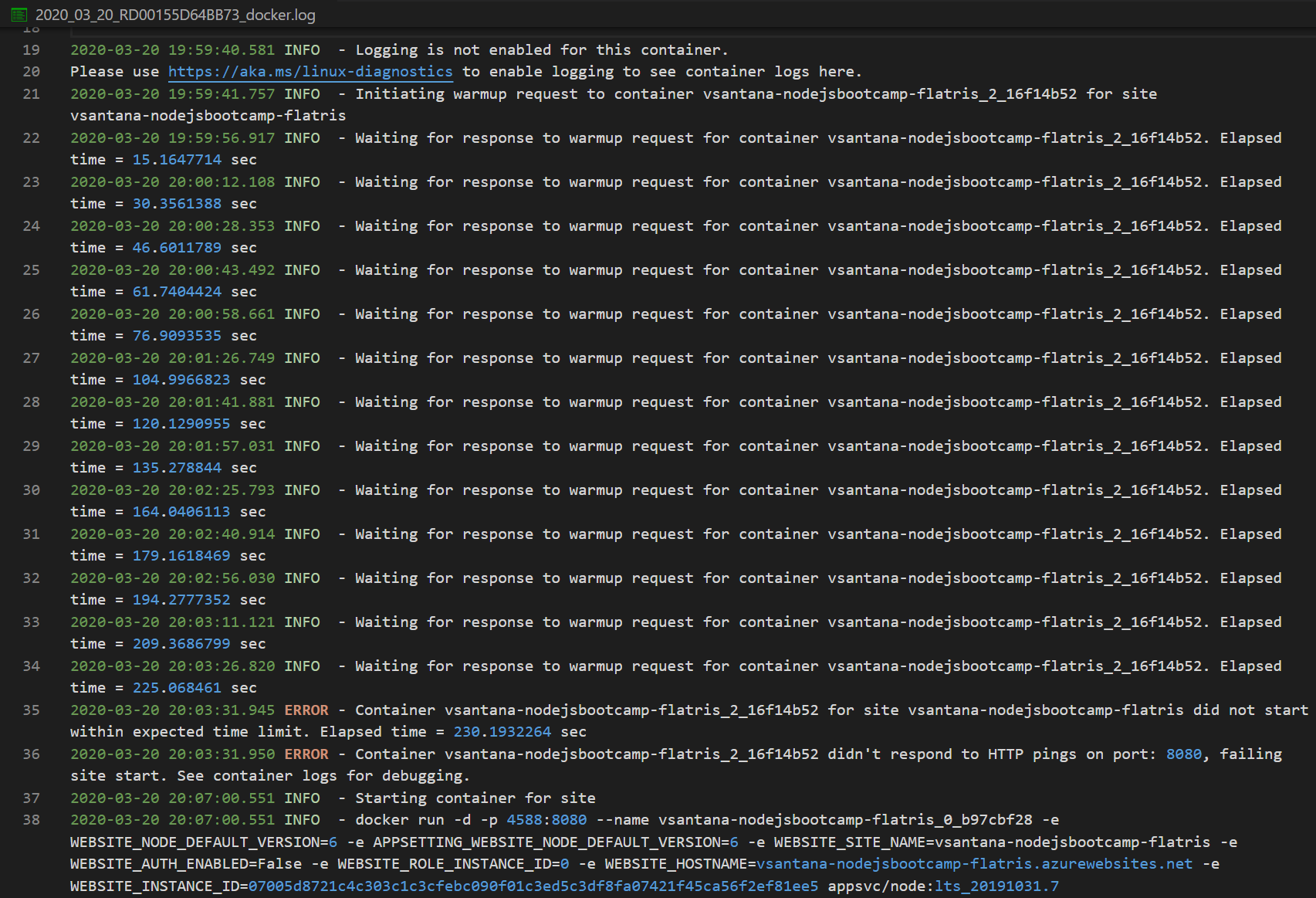
1. Click download current Docker logs (Download as zip):

Direct Link:

https://<alias>-nodejsbootcamp-flatris.scm.azurewebsites.net/api/logs/docker/zip

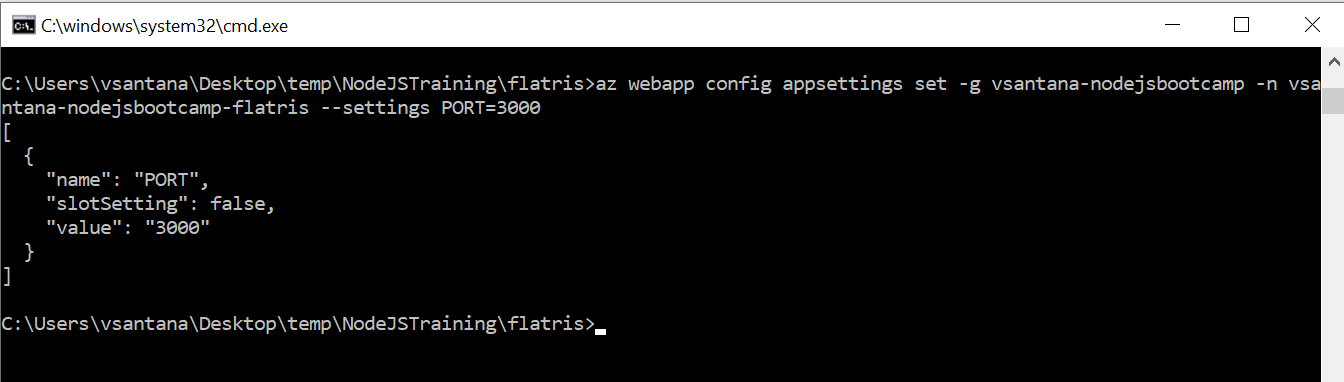
1. In this log you both the customer container log and the Docker Engine log:

* <date>\_<HostName>\_default\_docker à This is the Customer Log STDOUT
* <date>\_<HostName>\_docker à This is the Docker Engine Log

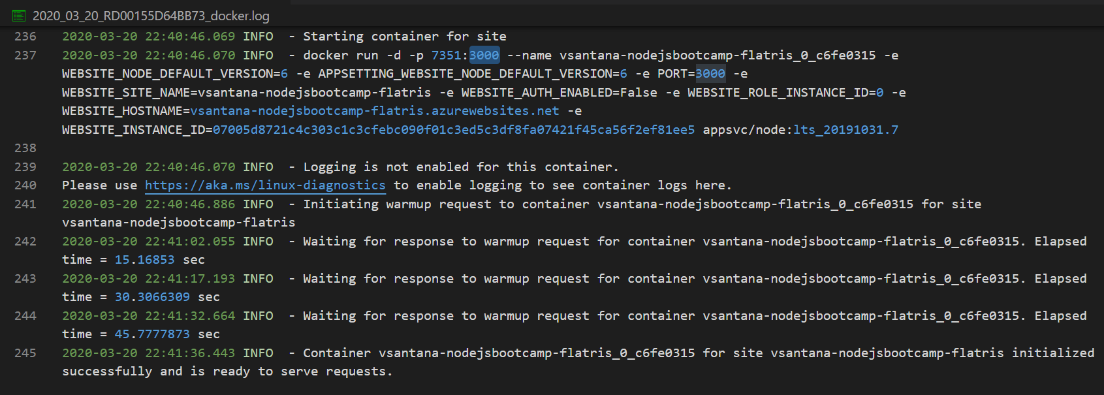
1. Open the log **<date>\_<HostName>\_default\_docker**  using Visual Studio Code:
2. 
3. Carefully review the output.
4. Open the log **<date>\_<HostName>\_ docker**  using Visual Studio Code:
5. 
6. Carefully review the output.
7. From both logs you can compare what the Azure WebApp is trying to do VS what the **“customer code”** is doing.
8. **“Customer code”** is listening under TCP port 3000:
9. 
10. Azure WebApp health check is trying to reach the App under TCP Port 8080:
11. 
12. There are a couple of ways in how to work around this scenario:

* By default, Azure WebApp using a blessed image uses TCP 8080. Changing the code to listen on 8080 instead of 3000 would fix the issue.
* Defining an Azure WebApp application setting change default listening PORT to 3000 would also fix the issue. You can achieve that by running the following command line:

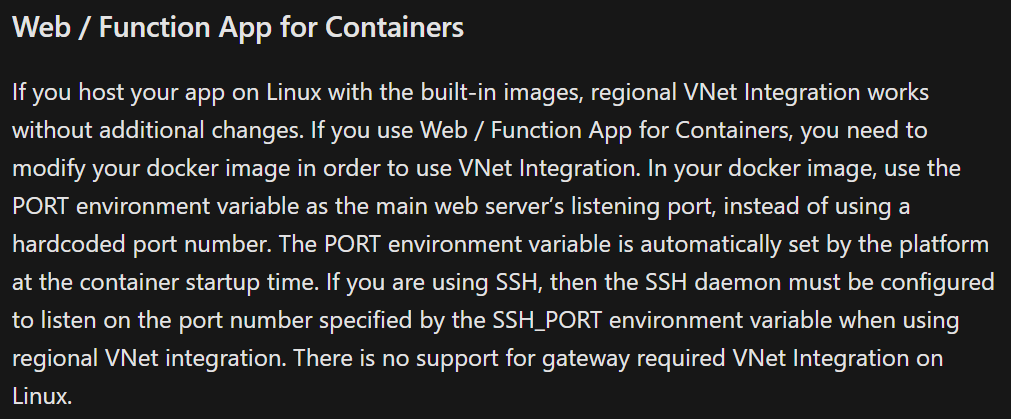
az webapp config appsettings set -g <alias>-nodejsbootcamp -n <alias>-nodejsbootcamp-flatris --settings PORT=3000



That’s how the log looks like if you change the Application setting defining a PORT to 3000 in the log.



* The expected way in how it should be fixed is changing the code to make it use an environment variable named PORT. It’s important to use this method because the Azure WebApp platform can dynamically change the environment variable when the container is initialized. There are some features like Regional VNet integration that depends on this approach to work:



Reference:

<https://docs.microsoft.com/en-us/azure/app-service/web-sites-integrate-with-vnet>

Change the code to fix the issue:

***Note:*** *If you have used the last method for testing you need to delete the App Setting. You can use the following command to delete it:*

az webapp config appsettings delete -g <alias>-nodejsbootcamp -n <alias>-nodejsbootcamp-flatris --setting PORT

Open the file server**../flatris/server/next.js** and replace **line 31** with:

startServer(server, process.env.PORT || '3000');

Open the file server**../flatris/web/utils/api.js** and replace **line 87** with:

return typeof window === 'undefined' ? 'http://localhost:' + process.env.PORT || '3000' : '';

This change will check if there is a PORT environment variable set if not will launch the code using the port 3000.

Now save both files, add to staging, commit and push the change.

git add .

git commit -m "Fix dynamic PORT"

git push -u azure master

Once it’s done the problem will be fixed.