

# Front-End Web Development with React

(by The Hong Kong University of Science and Technology)

Part II of IV - Front-end web development from Coursera



## About this course:

This course explores Javascript based front-end application development, and in particular the React library (Currently Ver. 16.3).

This course will use JavaScript ES6 for developing React applications.

You will;

- Get an introduction to the use of **Reactstrap** for Bootstrap 4-based responsive UI design.
- Be introduced to various aspects of React **components**.
- Learn about React **router** and its use in developing single-page applications.
- Learn about designing **controlled forms**.
- Introduced to the **Flux** architecture and **Redux**.
- Explore various aspects of Redux and use it to develop React-Redux powered applications.
- Learn to use **Fetch** for client-server communication and the use of **REST** API on the server side.
- A quick tour through React animation support and testing rounds off the course.

You must have preferably completed the previous course in the specialization on Bootstrap 4, or have a working knowledge of Bootstrap 4 to be able to navigate this course.

Also, a good working knowledge of JavaScript, especially ES 5 is strongly recommended.

## **At the end of this course, you will:**

- Be familiar with client-side Javascript application development and the React library
- Be able to implement single page applications in React
- Be able to use various React features including components and forms
- Be able to implement a functional front-end web application using React
- Be able to use Reactstrap for designing responsive React applications
- Be able to use Redux to design the architecture for a React-Redux application

## **Week 1 of 4**

### **Introduction to React**

In this module we get a quick introduction to front-end JavaScript frameworks and libraries, followed by an introduction to React. We will also learn about React components and JSX.

### **01-Welcome to Front-End Web Development with React**

Welcome to Front-End Web Development with React.

This is the second course in the specialization on Full Stack Web Development with React.

We have already covered Bootstrap 4, in the first course of this specialization.

In this course, we will concentrate on learning about the React library and also Redux for structuring our web application.

Before you get started on this course, let me make sure that you have sufficient background to be able to learn the topics covered in this course.

I expect that you will have a very good working knowledge of HTML and CSS.

I will expect that you would have a very good working knowledge of JavaScript and be able to write JavaScript programs and applications. Specifically, the ES5 version of JavaScript.

In this course itself, we will make use of the ES2015 or the ES6 version of JavaScript.

We will use some of the new features that have been introduced in the ES6 version of JavaScript including arrow functions and classes.

If you are not familiar with these, then you can learn these concepts along the way, as we do the exercises in this course.

I will leave with the message saying your mileage may vary.

Depending upon how dedicated you are, you could easily pick up these new concepts of ES6 as we go through this course.

If not, I will urge you to quickly brush up on your knowledge on ES6, especially, the new features of ES6 before proceeding ahead with this course.

Also, we will be extensively using Bootstrap as a way of styling the views in our React application.

I will prefer that you would have completed the first course on Front- End Web UI Frameworks and tools, Bootstrap 4, which is the first course in this specialization before you proceed ahead with this course.

Also, at the outset, let me clarify my view of Web design and development.

I view Web Design and Web development as two separate parts of the entire Web landscape.

The design aspect, concentrates on the user interface or the user experience design, the visual design, the prototyping, the colors, graphics, and animation aspect of web application development.

In this course, we are concentrating more on the development building and deployment of web applications. In particular, we are concentrating on hard skills, like the use of the JavaScript for developing our web application.

In the first course in the specialization, we covered Bootstrap 4.

In this course, we will look at React.

In the next course, we will look at React Native, as a way of developing mobile applications, and then the final course, we'll look at Server- side Development using Node Express and MongoDB.

In this course, we are concentrating more on building up our hard web development skills.

Design aspect is not something that we are concentrating in this specialization.

If you are approaching this course with the design aspect and view, then this course may not be suited for you.

Instead, we are looking at developing hardcore JavaScript skills for developing React applications.

Also, in this context, you may be wondering about Full Stack Web Development.

If you have taken the previous course in the specialization, I've already introduced you to Full Stack Web Development.

If not, the very first lecture in this course will introduce you to Full Stack Web development.

In this course, specifically, we are looking at the JavaScript framework library approach for developing web application.

In particular, we are looking at React.

Now, we will also look at React Native, for mobile application development at the next course.

Then, we'll look at the Server-side development including Business Logic Layer, and then the Data Layer, in the last course of this specialization.

Again, to emphasize, this course concentrates on the React library and the React ecosystem.

We will introduce you to React and look at the various aspects of developing a React application.

We'll also introduce you to React strap, as a way of styling the views in your React components.

We will look at the use of Redux as a way of structuring the architecture of your React plus Redux application, and we'll look at the use of Fetch for interacting with the server side from our React application.

So, these are the basic topics that will be covered in this course. Let's look at the detailed structure.

This course itself is organized into four modules.

Each module, typically, corresponding to one week of work.

Now, let that not be a constraint on your ability to learn.

If you need more time, than a week, for covering the topics in each module, then by all means please take sufficient time to understand the concepts and also understand the technology that we cover in each of the lectures.

Don't be in a hurry to complete the materials by the deadline for the assignments.

If you are unable to finish within this session, you are most welcome to move to the next session of this course.

This course will be repeated every 15 days and a new session will start every 15 days.

So, when you move to a new session, all the work that you have completed in the current session will be automatically carried over to the next session.

Now, with that in mind, let me cover what each module covers in this course.

The **first module** will introduce you to Full Stack Web Development. The big picture view of Full Stack Web development.

If you don't know Git and Node, then we will have a quick introduction to both Git and Node in this course.

If you have already done this as part of the first course of the specialization, then you can skip over that lesson.

Then, we will introduce you to the React, and we will explain how you can set up a React application using Create React App.

Then, we'll look at React Components, how we can develop React Components, and that should take you all the way to the very first assignment.

The **second module** concentrates on the React Router and Single Page Applications.

We will introduce you to various kinds of React Components and what role they play in your overall React Application.

We'll look at React Router, as a way of navigating between various views in your React Application, and then that will take you on to the Single Page Application concept.

We'll look at how we can develop Single Page Applications using React.

That should take you all the way to the second assignment of this course, where we will cover the React Components, the React Router, and Single Page Applications.

The **third module** in this course, concentrates mainly on Forms, as a way of user input. We'll look at Controlled Components in React and how Controlled Forms can be developed inside Controlled Components.

Then, we'll look at Uncontrolled Forms which are hosted in Uncontrolled Components in a React Application.

Then, we will introduce you to Redux as a way of structuring your React Application plus the architecture for your React application.

We'll briefly introduce you to Flux Architecture, and we'll look at Redux as a variant of the Flux architecture, and how you can structure your React plus Redux and application.

We'll then introduce you to React Redux Forms, as a way of leveraging the presence of Redux in your React application to persist your Form data.

That should take you all the way to the third assignment.

Where we will examine various kinds of React Forms and also briefly Redux.

The *fourth and final module*, we'll examine Redux in more detail.

We'll look at Redux actions, how we can set up Redux actions to change the state of our Redux tool.

Again, I'm throwing a lot of ideas at you. As you go through the course, you will learn more about these.

Don't worry too much about all these terminologies that I'm introducing in this very first lesson.

We will examine each and every one of them in detail, as we go through this course.

So, by the end of this course, you should be completely familiar with all these technologies and terminologies that we introduce here.

Then, we will go on to Redux Thunk.

We'll look at Client Server Communication.

How will your React application communicate with its server that supports REST API.

We'll look at the use of Fetch as a way of communicating with the server.

We'll briefly look at React Animations.

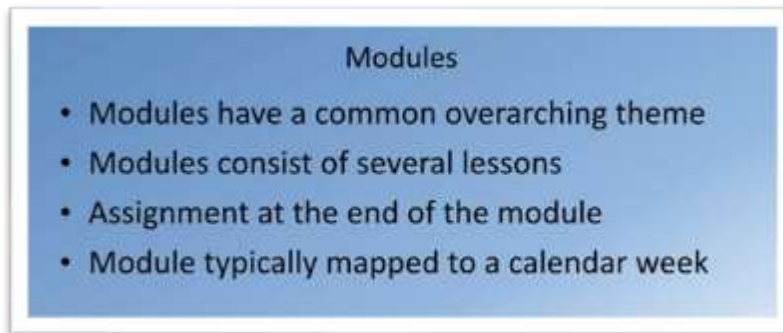
Then we'll look at how we can build and deploy your React application to a server, and look briefly at testing for your React application.

That should take you all the way to the final assignment.

I hope you have fun going through the various topics that we cover in this course, and at the end of this course, you would have a very good understanding of the react and redux ecosystem.

Once again, welcome to the Frontend Web Development using React.

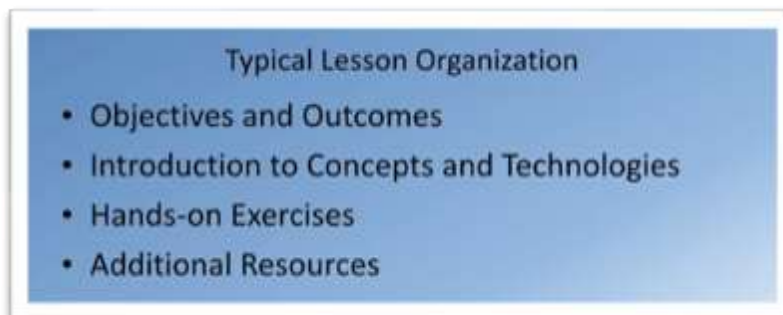
## 02-How to Use the Learning Resources



Before you embark on this course, let me take you through a quick tour of all the learning resources that are available in this course, and how best you can make use of these learning resources to maximize the benefit of doing this course.

The course itself is organized into four modules.

Each module has an overarching theme that guides the design of the module.



A module itself is divided into multiple lessons, each lesson concentrating on one specific topic.

There would be an assignment to do at the end of each module.

Each module typically corresponds to one calendar week of work, so this whole course could be completed in four weeks' time.

But, please take the time necessary for you to completely understand before you move on to the next module.

As already mentioned, each module is divided into multiple lessons. Each lesson itself is organized in this manner.

A lesson starts out with a set of objectives and outcomes being specified for that lesson.

Then following that, you will have a set of videos that introduce you to the concepts and technologies on which this particular lesson is concentrating on.

This will be followed by one or more hands-on exercises that will help you to understand the concepts and technologies by doing the exercises.

At the end of each lesson, you will also have additional resources, which will point you to resources that might be useful for the topic covered in this particular lesson.



Let's take a quick tour of your course pitch.

As you go to the course homepage, you will find a lot of details about the course there.

You'll also notice that the course itself is organized into weeks, each week corresponding to one module.

Let's visit one particular module to learn how the module itself is structured.



As you go to each module, you will see that the module will have a brief introduction on the theme of that particular module.

Then there will be a set of learning objectives for the particular module.

Following that, you will have a set of lessons. As I mentioned earlier, each lesson will be organized into a set of items.

Let's take a quick tour of this lesson on Introduction to React.



As you click and go into that lesson, you will notice that at the start of the lesson, you will have the objectives and outcomes for that lesson explicitly specified.

I would urge you to take a few moments to read through this document because this will clearly set up the goals for this lesson, and also tell you exactly what you will achieve at the end of that particular lesson.

Following that, you would have a set of videos that will explain some of the concepts and technologies relevant to that particular lesson.

I would urge you to watch these videos first. Following this set of videos, you will have a one or more hands-on exercises for that particular lesson.

The exercises themselves will be illustrated through a video, which gives you step-by-step progress of that particular exercise.

I would strongly urge you to watch this video in order to understand how to do the exercise.

Also, in the exercise, I will explain a lot of details about why each step is being performed in that particular exercise.

Now, in addition to the video illustration of the exercise, you will also have a summary of the instructions for that particular exercise given as a separate document following that video.

The objectives and outcomes for that particular exercise will be listed out, then each step will be illustrated briefly in this particular document here.

So, once you watch the video, you can also use this exercise instructions as a guide as you perform your own exercise routine.

In addition, the exercise instructions will also contain code snippets that will be useful for you to do your exercise.

I would strongly urge you not to simply cut and paste this course's snippets, instead, understand exactly what you're doing at each step and then, either typing the code yourself or copy and paste the code from the instructions here.

Furthermore, for every video, if there is a set of presentations slides, those would be available in the downloads to the right of this video there on the webpage.

Also, if you go to the additional resources that are provided for you at the end of each lesson, you will also have links to the PDFs of the presentation.

There are two places from where you can download the PDFs of the presentations about the concepts and technologies for each module.

You will also have links to React documentation about the specific concepts and technologies covered in that particular lesson.

I will also give you links to articles, blogs, or additional resources from the Internet that might be useful or relevant to this particular topic of the lesson.

I would suggest to you to spend some time looking up these additional resources so that they may aid you in also furthering your understanding of the specific topics covered in the particular lesson.

To summarize, the learning resources in each lesson is organized into a set of video lectures illustrating the concepts and technologies, then hands-on exercises, which include both step-by-step instructions in the form of a video and also written instructions for the exercises, and additional resources that might be useful for you relevant to that particular lesson.

Also, as I mentioned, at the end of each module, you will have an assignment due.

The tasks to be performed in the assignment will be illustrated through a video, where all the tasks that you need to do in the assignment will be showed to you.

Thereafter, you will have a written document that explains the details of that assignment.

This written document will give you access to the resources that may be required for this assignment.

Following that, you will have the set of objectives and outcomes for the particular assignment given to you.

Then, the requirements of the assignment will be specified in terms of the set of tasks that you need to complete, and the steps for each task will be illustrated there.

You may also be given a snapshot of the webpage to show you the state of the webpage at the completion of this particular assignment, with the specific tasks being highlighted in that snapshot.

At the end of the document, you'll also be given the review criteria, the criteria based on which your submission will be judged for this particular assignment.

As you embark on this course, let me quickly remind you of a Sanskrit Subhashita or a wise saying about learning. It goes as follows, AchAryAt pAdam Achette, pAdam shiShyaH SwamedhayA sa-brahmach Aribhyah pAdam, pAdam kAlakrameNa cha. What this means is that one-fourth of your learning comes from your teacher, one-fourth of your learning comes through your own intelligence. As you do the steps, use your own intellect to analyze and

understand what you are doing. One-fourth of your learning comes from your classmates. So this is where interacting with your colleagues that are also taking this course at the same time through the discussion forums would also help you to understand the concepts. Finally, the last fourth comes only through experience or only with time. As you learn and make use of the concepts and technologies that you learn in this course, you'll begin to understand more and more about this particular topic. Let me leave you with my best wishes for you, so that you would get the most out of your journey through this course. Wish you all the best.

## Welcome to Front-End Web Development with React: Additional Resources

### PDFs of Presentations

- 00.Course-Overview.pdf
- 00.How-to-Learn.pdf
- 01. FSWD-BigPicture.pdf

### React Resources

- [React Site](#)

### Coursera Resources

- [Coursera Learner Help](#)
- [Switching to a Different Session](#)

## Full Stack Web Development: The Big Picture: Objectives and Outcomes

### 03-What is Full-Stack Web Development?

The lecture gives you an overview of full stack web development.

At the end of this lesson, you will be able to:

- Understand what is meant by full stack in the context of web development
- Distinguish between front-end, back-end and full stack web development
- Understand the position of this course in the context of this specialization

### What is Full Stack Web Development?

Let me clarify to you a few terms so that we start with a common understanding of full stack web development as applied in this specialization.

We often hear people talking about the front end and the back end.

The front end is where we are delivering the content to the user, typically in a browser where the user accesses the information and this is where we use technologies like HTML, CSS, and Javascript to render the content for the user.

This information delivery is supported behind the scenes by a back end support, which is typically implemented these days using technologies like PHP, Java, ASP.NET, Ruby, Python, or Node.js.

We often hear people talking about the three tier architecture for web development.

In this approach, the entire web application is organized into three different layers.

The presentation layer which is concerned with delivering the information to the user, so this is usually the UI related concerns that are dealt with at the presentation layer.

The business logic layer on the other hand, is concerned more about the data, the data validation, the dynamic content processing, and generating the content to be delivered to the user.

This is backed up behind the scenes with the data persistence layer or the data access layer.

This is concerned with how we store and interact with the data, typically in the form of a database and access this data through an API.

Exploring this further, let us see what is implemented typically in the traditional web development in each of these three layers.

The business logic layer is usually implemented these days using technologies like Ruby, Python, PHP, Java, C++, or ASP.NET.

This business logic layer is interacting behind the scenes with the persistent data, typically stored in a relational database and accessed by the business logic layer.

The business logic layer is also concerned with the rendering of information to the front side, typically in the form of server-side rendering these days.

The HTML, CSS, and Javascript is generated on the server side and then sent over to the client side in the form of a web page.

In this approach, we need specialists in each of these three layers.

A front-end specialist typically would be well versed in HTML, CSS, and Javascript.

The business logic specialist would be well versed in one of the technologies that is used for implementing the business logic and then, you need a data specialist who will be well versed in the relational database management system.

There is an increasing trend towards using a single language to implement the entire stack, this being Javascript.

You could have the front end implemented, for example, as a single page application using frameworks like Angular or React.

You have the server side or the business logical layer being implemented using technologies like Node.js, which is also dependent on JavaScript and then, you have the data storage itself being implemented using technologies like MongoDB, which stores data in the form of JSON documents and the information exchange between the server side and the client side is usually done using JSON as the format and the server side supports a REST API endpoint.

We will cover all these technologies as part of this specialization.

As you go through the specialization, you will see that on the presentation layer side, we will cover Bootstrap 4 and Angular or React for implementing front-end applications.

Then at the business logic layer, we will be using Node.js and Node.js modules and Express for implementing the business logic layer.

We will also consider back end as a service and then the data support implemented using MongoDB.

## Full Stack Web Development: Additional Resources

### PDFs of Presentations

01-FSWD-BigPicture.pdf

### Useful Links

- What is a Full Stack developer?
- Wait, Wait... What is a Full-stack Web Developer After All?
- The Myth of the Full-stack Developer
- Multi-tier Architecture
- What is the 3-Tier Architecture?

## Setting up Your Development Environment: Git and Node: Objectives and Outcomes

At the end of this lesson you should have set up Git and Node.js on your computer.

You can skip this lesson if you have already completed this as part of an earlier course in the specialization.

At the end of this lesson, you will be able to:

- Set up a Git repository and perform basic Git operations
- Set up and use online Git repositories
- Use Node-based modules to perform basic operations.

## Setting up your Development Environment

### Software Requirements

1. Text editor of your choice: Any text editor that you are already familiar with can be used for editing the project files. I will be using Visual Studio Code (<https://code.visualstudio.com/>) as the editor of choice in this specialization. You may also consider other editors such as Brackets (<http://brackets.io/>), Sublime Text (<http://www.sublimetext.com/>), or Atom (<https://atom.io/>).
2. Browser of your choice: You may use your preferred browser. I will be using Chrome as the browser in all the exercises. All the exercises and assignments in this course have been tested using Chrome v. 46. Please note that not all browsers may support all the HTML5 features to the same extent. You might encounter problems when using other browsers. I strongly urge you to use the latest Chrome browser for the exercises and assignments in this course so that any problems are minimized.
3. Command line shell: Familiarity with the command-line shell will be essential for the exercises. In Windows a cmd window or power shell with admin privileges would be needed. On a Mac or in Linux, a terminal window can be used. Please get familiar with the "sudo" command in OS X and Linux.
4. Files required for the exercises: We will provide additional starter files for the exercises wherever needed. Links to download the files will be provided inline in the exercise instructions that follow each exercise video. Please download the files provided there, if any, before beginning the exercise. The links are also available through the Additional Resources of the specific lesson.

Note: Please remember to retain the folders and all the files that you create in the exercises. Further exercises will build upon the files that you create in the preceding exercises. DO NOT DELETE the files at the end of the exercises, unless

otherwise instructed. You may wish to set up your exercise folder as a Git repository and commit the files to the repository at the end of each exercise.

## 04-Exercise (Video): Setting up Git

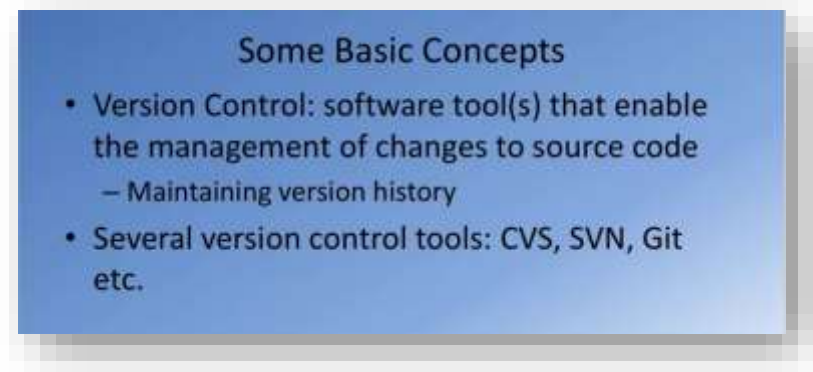
Git is a very popular version control system for software.

They need Git for working with our node ecosystem that we're going to use throughout this specialization.

It is important for you to get Git onto your computer.

Let's go through a few quick steps to set up Git on your computer and then some quick commands to enable us to make use of Git in this specialization.

I will not go into too much of details about version control and how Git works and so on, that is beyond the scope of this course.



Instead, we'll just get some basic understanding of Git.

Git is as I said, a version control system.

This is a software tool that enables us for the management of changes to source code and maintaining a version history.

As your source code evolves, you will be able to chicken the code at different piles of time so that you can always have a way of rolling back to a previous version in case your updates to the code doesn't work correctly.

There are various version control systems that are in use in the real-world including CVS, SVN and Git, being a very popular mechanism for version control these days.

## Git

- Distributed version control system
- Developed by Linus Torvalds for managing Linux kernel development
- Widely adopted now by several projects
  - The Node ecosystem thrives on it

Where did Git originate?

Git was designed by Linus Torvalds.

The person behind the Linux operating system.

Git was designed as a version control system, a distributed version control system for use in Linux kernel development.

And it has seen much wider deployment in the real world these days.

The node ecosystem is very much tied in to Git and that is the reason why we need Git for this specialization.

Let's now move on to a few hands on exercises where you will first set up Git, then you will learn a few basic Git commands, and then also understand how you can set up an online repository in places like Github or Bitbucket for synchronizing your source code from your computer to the online repository.

This is obviously not a comprehensive tutorial on Git but we will learn just enough of the commands that are necessary for use in this specialization.

## Exercises

- Setting up Git on your machine
- Using Git
- Using online Git repositories

Let's now run through a few quick steps to set up Git on your computer.



One easy way of setting up Git on your computer, is to go to this site called [git-scm.com](https://git-scm.com) and then download the Git installation files from there.

When you visit this website [git-scm.com](https://git-scm.com), you will see on the right-hand side here a button for you to download the Git.

If you want too you can go directly to the downloads page here on the site and then you'll find the downloads for various platforms and you can download the one for your specific platform for installation.

This is the easiest way of getting Git onto your computer.

There are other ways of setting up Git depending on your platform.

For example, if you install Github desktop on your computer, it'll automatically set up Git for you.

If you're using a Mac and you set up Xcode and especially the Xcode command line tools, that will also set up Git for you.



On this site [git-scm.com](https://git-scm.com), if you go to the documentation page you will notice that there is a book here called Pro Git.

You can just go on to the Getting Started link.

You will see a specific chapter here on installing Git.

Just go to this installing Git site and there you have more details about how to install Git on your specific platform.

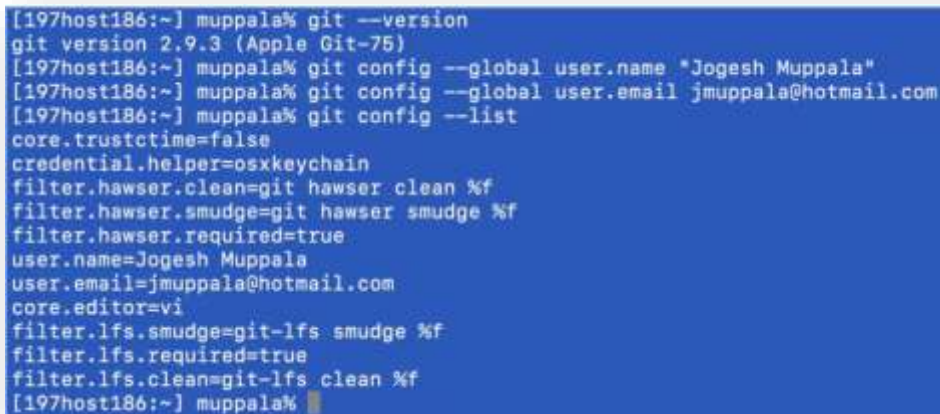
You can use any of the methods suggested here for installing Git on your platform.

This is fairly straightforward so I'm not going to go through the details of how to do the set up.

Download the installer and run through the set up to get Git onto your computer.

Once you install Git on your computer, start up a command window or PowerShell if you're using a Windows machine, or start up a terminal window if you're using a Mac or a Linux machine, and then add the prompt type in Git minus version to check what version of Git is installed on your computer.

On my machine I have this current version of Git installed and that is good enough for me to work with.

A terminal window with a blue background and white text. The text shows a series of commands and their outputs for installing and configuring Git. The commands include checking the version, setting global user name and email, and listing the configuration. The output shows Git version 2.9.3 and the configured user name 'Jogesh Muppala' and email 'jmuppala@hotmail.com'.

```
[197host186:~] muppala% git --version
git version 2.9.3 (Apple Git-75)
[197host186:~] muppala% git config --global user.name "Jogesh Muppala"
[197host186:~] muppala% git config --global user.email jmuppala@hotmail.com
[197host186:~] muppala% git config --list
core.trustctime=false
credential.helper=osxkeychain
filter.hawser.clean=git hawser clean %f
filter.hawser.smudge=git hawser smudge %f
filter.hawser.required=true
user.name=Jogesh Muppala
user.email=jmuppala@hotmail.com
core.editor=vi
filter.lfs.smudge=git-lfs smudge %f
filter.lfs.required=true
filter.lfs.clean=git-lfs clean %f
[197host186:~] muppala%
```

Once we verify that Git is installed, we will configure a couple of global identity parameters, the user name and an email address so that whenever Git does any comments as you will learn later, it will make use of this information.

To do that add the prompt type Git config minus-minus global user name.

And then you can type in your username.

The other parameter that I'm going to configure is my email.

And to ensure that this information has been configured, you can type Git config list, then it lists out a bunch of configurations that we have done.

Some of these are automatically set up for you but if you want to you can set them up as you require.

Here I have my code editor value set to VI, which is what I use to do a command line editing of files.

With this we have completed the setup of Git for use on our computer.

Once you get Git on your computer, it is time to go and learn Git bit by bit.

Next

## Exercise (Instructions): Setting up Git

## Objectives and Outcomes

In this exercise you will learn to install Git on your computer. Git is required for using all the remaining Node.js and Node based tools that we encounter in the rest of the course. At the end of this exercise, you would be able to:

- Install Git on your computer
- Ensure that Git can be used from the command-line or command-prompt on your computer
- Set up some of the basic global configuration for Git

## Downloading and Installing Git

- To install Git on your computer, go to <https://git-scm.com/downloads> to download the Git installer for your specific computing platform.
- Then, follow the installation steps as you install Git using the installer.
- You can find more details about installing Git at <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>. This document lists several ways of installing Git on various platforms.
- Installing some of the GUI tools like GitHub Desktop will also install Git on your computer.
- On a Mac, setting up XCode command-line tools also will set up Git on your computer.
- You can choose any of the methods that is most convenient for you.

## Some Global Configuration for Git

- Open a cmd window or terminal on your computer.
- Check to make sure that Git is installed and available on the command line, by typing the following at the command prompt:

```
git --version
```

- To configure your user name to be used by Git, type the following at the prompt:

```
git config --global user.name "Your Name"
```

- To configure your email to be used by Git, type the following at the prompt:

```
git config --global user.email <your email address>
```

- You can check your default Git global configuration, you can type the following at the prompt:

```
git config --list
```

## Conclusions

At the end of this exercise you should have Git available on the command-line of your computer.

This course explores Javascript based front-end application development, and in particular the React library (Currently Ver. 16.3).

This course will use JavaScript ES6 for developing React application.

- You will also get an introduction to the use of Reactstrap for Bootstrap 4-based responsive UI design.
- You will be introduced to various aspects of React components.
- You will learn about React router and its use in developing single-page applications.
- You will also learn about designing controlled forms.
- You will be introduced to the Flux architecture and Redux.
- You will explore various aspects of Redux and use it to develop React-Redux powered applications.
- You will then learn to use Fetch for client-server communication and the use of REST API on the server side.

A quick tour through React animation support and testing rounds off the course.

You must have preferably completed the previous course in the specialization on Bootstrap 4, or have a working knowledge of Bootstrap 4 to be able to navigate this course.

Also, a good working knowledge of JavaScript, especially ES 5 is strongly recommended.

At the end of this course, you will:

- Be familiar with client-side Javascript application development and the React library
- Be able to implement single page applications in React
- Be able to use various React features including components and forms
- Be able to implement a functional front-end web application using React
- Be able to use Reactstrap for designing responsive React applications
- Be able to use Redux to design the architecture for a React-Redux application.

<https://reactjs.org/>

# Module 1: Full Stack Web Development: The Big Picture: Objectives and Outcomes

This lesson gives you a big picture view of the Full Stack Web Development. The lecture gives you an overview of full stack web development. At the end of this lesson, you will be able to:

- Understand what is meant by full stack in the context of web development
- Distinguish between front-end, back-end and full stack web development
- Understand the position of this course in the context of this specialization

**WEEK**

**1**

10 hours to complete

## Introduction to React

---

In this module we get a quick introduction to front-end JavaScript frameworks and libraries, followed by an introduction to React. We will also learn about React components and JSX.

### Learning Objectives

---

- Express the general characteristics of JavaScript frameworks and libraries
- Create a new project using React
- Create React components within your React application
- Express what is meant by full-stack web development

## 01.Welcome to Front-End Web Development with React

### PDFs of Presentations

0-Course-Overview.pdf

PDF File

How to Learn.pdf

PDF File

React Resources

- [React Site](#)

Coursera Resources

- [Coursera Learner Help](#)
- [Switching to a Different Session](#)

---

## 02.How to Use the Learning Resources

---

### 03.What is Full-Stack Web Development?

Objectives and Outcomes

This lesson gives you a big picture view of the Full Stack Web Development. The lecture gives you an overview of full stack web development. At the end of this lesson, you will be able to:

- Understand what is meant by full stack in the context of web development
- Distinguish between front-end, back-end and full stack web development
- Understand the position of this course in the context of this specialization

---

### Full Stack Web Development: Additional Resources

Exercise (Video): Setting up Git

Useful Links

- [What is a Full Stack developer?](#)
  - [Wait, Wait... What is a Full-stack Web Developer After All?](#)
  - [The Myth of the Full-stack Developer](#)
  - [Multi-tier Architecture](#)
  - [What is the 3-Tier Architecture?](#)
-

## 04.Setting up Your Development Environment: Git and Node: Objectives and Outcomes

At the end of this lesson, you should have set up Git and Node.js on your computer. You can skip this lesson if you have already completed this as part of an earlier course in the specialization. At the end of this lesson, you will be able to:

- Set up a Git repository and perform basic Git operations
- Set up and use online Git repositories
- Use Node-based modules to perform basic operations.

### Setting up your Development Environment

#### Software Requirements

1. **Text editor of your choice:** Any text editor that you are already familiar with can be used for editing the project files. I will be using Visual Studio Code (<https://code.visualstudio.com/>) as the editor of choice in this specialization. You may also consider other editors such as Brackets (<http://brackets.io/>), Sublime Text (<http://www.sublimetext.com/>), or Atom (<https://atom.io/>).
2. **Browser of your choice:** You may use your preferred browser. I will be using Chrome as the browser in all the exercises. All the exercises and assignments in this course have been tested using Chrome v. 46. Please note that not all browsers may support all the HTML5 features to the same extent. You might encounter problems when using other browsers. I strongly urge you to use the latest Chrome browser for the exercises and assignments in this course so that any problems are minimized.
3. **Command line shell:** Familiarity with the command-line shell will be essential for the exercises. In Windows a cmd window or power shell with admin privileges would be needed. On a Mac or in Linux, a terminal window can be used. Please get familiar with the "sudo" command in OS X and Linux.
4. **Files required for the exercises:** We will provide additional starter files for the exercises wherever needed. Links to download the files will be provided inline in the **exercise instructions** that follow each exercise video. Please download the files provided there, if any, before beginning the exercise. The links are also available through the **Additional Resources** of the specific lesson.

Note: Please remember to retain the folders and all the files that you create in the exercises. Further exercises will build upon the files that you create in the

preceding exercises. DO NOT DELETE the files at the end of the exercises, unless otherwise instructed. You may wish to set up your exercise folder as a Git repository and commit the files to the repository at the end of each exercise.

---

## Exercise (Video): Setting up Git

## Exercise (Instructions): Setting up Git

### Objectives and Outcomes

In this exercise you will learn to install Git on your computer. Git is required for using all the remaining Node.js and Node based tools that we encounter in the rest of the course. At the end of this exercise, you would be able to:

- Install Git on your computer
- Ensure that Git can be used from the command-line or command-prompt on your computer
- Set up some of the basic global configuration for Git

### Downloading and Installing Git

- To install Git on your computer, go to <https://git-scm.com/downloads> to download the Git installer for your specific computing platform.
- Then, follow the installation steps as you install Git using the installer.
- You can find more details about installing Git at <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>. This document lists several ways of installing Git on various platforms.
- Installing some of the GUI tools like GitHub Desktop will also install Git on your computer.
- On a Mac, setting up XCode command-line tools also will set up Git on your computer.
- You can choose any of the methods that is most convenient for you.

### Some Global Configuration for Git

- Open a cmd window or terminal on your computer.
- Check to make sure that Git is installed and available on the command line, by typing the following at the command prompt:

```
git --version
```

- To configure your user name to be used by Git, type the following at the prompt:

```
git config --global user.name "Your Name"
```

- To configure your email to be used by Git, type the following at the prompt:



```
git config --global user.email <your email address>
```

- You can check your default Git global configuration, you can type the following at the prompt:

## 05.Exercise (Video): Basic Git Commands

Let us now explore some Basic Git Commands that are very useful for us in this specialization.

There is a lot more to get than what we cover in this exercise.

To get started, go to a convenient location on your computer, and then create a folder named git-test.

Then, open this folder in your favorite editor.

Here I have the GIT-TEST folder that we just created open in my Visual Studio code.

Let me add a file to this folder named index.html.

Now, you can see that I've added in some html code into this index.html file.

Let's save the changes.

Now, let's switch to our command line.

At the command line, go to the git-test folder.

And let's initialize this folder to be A Git repository.

To do that, at the command line type \$ `git init.`

Now, this folder has been initialized to be a Git repository.

This is our first Git command that we've learned, \$ `git init.`

This initializes the current folder as a Git repository.

And when it initializes the folder, it will mark that folder as a master.

This is the master branch for my git.

Now, let's not worry about branches and so on, we will not deal with that in this course.

We will only be working with the master branch in this particular specialization.

This would be marked as a master.

Now, this is the initial point of our repository.

The next command that we're going to look at is git status.

If you type \$ `git status` in the command line, it'll tell you the current status of the folder.

Let's do that at the command line, and see what it shows.

On the command line type `git status`, and read the information that is tapped out on the command Window.

You see that it says on branch master, so that is the master branch that we are on.

And it says untracked files, and then shows `index.html` in red.

On your specific computer, it may be using different colors to represent this differently, but this is what it shows on my Mac.

Now, this `index.html` file that we have just created in this folder is now not been added to our git repository.

Let's go ahead and add that file to the git repository.

To do that, we say \$ `git add`, and you can simply say dot, which means that all the files in the current directory will be added to what is called the staging area of my git repository.

Now, I again type `git status`, you will see that that file `index.html` is marked in green.

And it says, changes to be committed there, and then shows the file name.

And then, so that mean that this finally is now ready to be committed to my git repository.

The next command that we saw `git add`.

By using `git add` you can add files or folders to the staging area.

Once you add it to the staging area, then you can commit that snapshot of folder status to our git repository.

That means that when you do that `git commit`, what this command will do is commit the current state of our folders into our git repository.

All the files, as they exist at the moment, once they have been staged using the `git add`, then they will be committed to our git repository when we execute the `git commit` command folder.

At this point when we execute the `git commit` command, then our initial statement now be changed to the first commit to the git repository.

Let's go ahead and do that.

Back at the Command Prompt, let's type `git commit` and then we can even add a message to our commit.

I'm going to say `git commit -m first commit` because this is the first commit to our git repository.

When I do that it says, okay, one file has been added to the git repository, and some other information will be typed out onto the command Windows.

Let's now check again, `git status`.

And now you see that it says nothing to commit, working directory is clean.

What that means is the current state of my working directory or working folder has been committed to the git repository.

A snapshot has been committed to my git repository.

Now, I can type the next command, called `git log` online, and see that it shows a number there, an eight-digit number there.

And then, also shows the message that we put into our commit, saying, first commit.

That is the log of all the commits that have been put into my git repository.

Going back to our next git command, we saw that `git log --` one line will show us a brief log of all the commits.

If you simply type `git log`, it will display a lot more detail information about all the commits.

But this is sufficient enough for obtaining information that we require.

Let's now come back to our editor here in Visual Studio Code.

I'm going to add more changes to my `index.html` file.

Now, let me add another folder under the `git-test` folder.

I will create a subfolder here named, `templates`.

And inside this `templates` folder, I'm going to create another file named `test.html`.

This is just to show you how Git can commit entire folder hierarchy into its repository.

With `test.html` now there, I'm just going to copy everything from my `index.html` into my `test.html`, and see if that changes.

Going back to the command line, let's now check out the status of our git repository and this folder.

Typing `$ git status` shows that the index.html file that we have already added earlier to the depository has now been modified.

There is a newer version of the index.html file.

Also, it shows that there are some untracked files in this folder called templates.

Let's add all these changes to our repository to the staging area.

Again, type `$ git add dot.`

And then, all these files will be added to the staging area.

Again, checking out the status you now see the changes that have been added to the staging area.

All these files have been added to the staging area.

Let's do one more commit.

I would say `$ git commit -m`, Second commit.

And then, let's check out the log.

If you check out the log, you'll now see that there are two commits in my git repository.

The first and the second commit, and note that each one of them is given a different number there.

If you want to see the full details of the log, you can type simple `$ git log` and then you will see more details in there than what you would be interested in.

Notice that where one-line comment only gives the first few characters of my commit number there, but it's sufficient enough for us to operate with.

Let me now go back again to my Visual Studio and add one more,

One more line to my index.html file.

Now, my index.html file has been modified, and let's save the changes.

Going back to the command line, doing git status shows that the index.html file has been modified.

Let's add this to the staging area, and then do a third count.

Let's say, git add, git status.

Now you see that the index.html modified version has been added.

Now, we can say git commit -m third commit.

And do `git log --oneline`, and you'll see there are three comments in our repository.

Now, our repository contains snapshots of three different points.

At the end of the first commit, at the end of the second commit, and at the end of the third commit.

Now we also can roll back changes.

We can revert the repository to a previous version.

We can pull out a file from an older commit, and then, replace the existing file in our directory from the older commit.

Let's see how we can operate with these things by learning a couple of more commands.

At this stage our `index.html5` is in the current state.

You can notice that it has an `h1` and 2 `p`'s.

Let's now look at the next git command.

The next git command that we're going to learn about is `git checkout`.

This checkout command allows us to checkout a file from a previous commit in our git repository.

If we don't like the current file that we have in our folder, and we want to go back to a previous version of the file.

We can always check out the file from a previous commit, or from the current commit, and then continue to work with that file.

Let's make use of this and see some further changes to our git repository.

Going back to our command line, we remember that between the second and the third commit, I made changes to my `index.html` file.

Suppose I want to revert back to the index from HTML5 for my second commit.

Then I can simply say `git checkout 900cfcf`, so that is the commit identifier, the number that identifies that particular commit.

And then, I can say `index.html`, and what you will notice is that older file, it checked out.

Into my current working directory.

Going to my virtual studio code, you now notice that my `index.html` file has reverted to the previous version so the change that I made before the third commit is now gone.

My index.html file has been restored to its state at the end of the second commit.

Now, at the command line, if I type, `git status`, you'll notice that this index.html file which has reverted to what it was at the end of the second commit.

It has now already been staged, so using the `git checkout` will pull out an older version of the file.

And then, replace what is in the current directory, and then it will also check it into the staging area.

Now, if I do that and then I realize that this is what I want, I can simply do another commit at this point.

And then that file can be committed as the fourth commit.

But suppose I don't like this.

I want to revert back to the index.html file at the end of the third commit, then all I can do is, say `git reset HEAD`, and index.html.

At this point what happens is that the index.html, the modified version that I have checked out, is still there.

But this file has been unstaged from the staging area.

If you go back and look at the index.html in your editor, it will still show the state at the end of the second commit, because we had pulled out that file using `checkout` for that.

Now, if you want to revert it back to what it was at the end of third commit, then we do one more `checkout` from the third commit.

Going to our, Command Window, type `$ git status` and you will notice that the index.html is marked as modified, but it also shows this particular statement here.

It says `git checkout, --` and the file name to discard the changes in the working directory.

That's one way you can discard the changes that you have made to a particular file corresponding to the previous commit.

Let me restore this index.html back to what it was at the end of the third commit.

To do that, I will simply say, `$ git checkout -- index.html`, and then if I do, `Git status`, it shows that my directory is clean, and basically my directory has been restored to the state of the end of the third commit.

Going to the File in my Visual Studio Code.

I see that the file has been restored back to what it was at the end of the third commit, so this is one way you can.

If you have made changes to a file after a commit and you want to just discard those changes, you can simply Check out the file from the last commit.

And then, all your changes that you've done after the last commit will be discarded.

These are some basic commands that are very useful for you as you go through the courses in this specialization, because you may want to commit at the end of each exercise.

And as you proceed forward, you will still have a completed version of this state of your folder at the end of the previous exercise.

That way, if you are getting out a new exercise, and you discovered that you have made mistakes, and you want to revert back to the previous commit.

You always have a way of doing that, using the commands that we have just learned.

With this basic understanding of these few get commands we will be able to proceed forward with understanding and using get in the courses of this specialization.

Now, we have reviewed, the git reset, for a specific file on git reset in general.

If you simply type git reset, it will restore your back to the last comment.

It will reset the staging area to the last commit, without disturbing the changes that you've done to your working directory.

Once you reset then you can check out the previous version of the file that you have committed in the previous commit.

This way, you can restore your folder back to where you were at the starting point of previous commit.

Sometimes when you are going to an exercise, and you realize you made a mistake.

You always have a way of reverting back to a previous version.

With these commands, I think you're all set to go ahead to use git in the courses of this specialization.

At the end of this exercise, did you git it?

```
git config --list
```

## Conclusions

At the end of this exercise you should have Git available on the command-line of your computer.

## Exercise (Instructions): Basic Git Commands

### Objectives and Outcomes

In this exercise you will get familiar with some basic Git commands. At the end of this exercise, you will be able to:

- Set up a folder as a Git repository
- Perform basic Git operations on your Git repository

### Basic Git Commands

- At a convenient location on your computer, create a folder named **git-test**.
- Open this git-test folder in your favorite editor.
- Add a file named *index.html* to this folder, and add the following HTML code to this file:

```
<!DOCTYPE html>

<html>

  <head></head>

  <body>

    <h1>This is a Header</h1>

  </body>

</html>
```

### Initializing the folder as a Git repository

- Go to the git-test folder in your cmd window/terminal and type the following at the prompt to initialize the folder as a Git repository:

```
git init
```

### Checking your Git repository status

- Type the following at the prompt to check your Git repository's status:

```
git status
```

### Adding files to the staging area

- To add files to the staging area of your Git repository, type:

```
git add .
```

### Committing to the Git repository



- To commit the current staging area to your Git repository, type:

```
git commit -m "first commit"
```

## Checking the log of Git commits

- To check the log of the commits to your Git repository, type

```
git log --oneline
```

- Now, modify the *index.html* file as follows:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head></head>
```

```
<body>
```

```
<h1>This is a Header</h1>
```

```
<p>This is a paragraph</p>
```

```
</body>
```

```
</html>
```

- Add a sub-folder named **templates** to your **git-test** folder, and then add a file named *test.html* to the templates folder. Then set the contents of this file to be the same as the *index.html* file above.
- Then check the status and add all the files to the staging area.
- Then do the second commit to your repository
- Now, modify the *index.html* file as follows:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head></head>
```

```
<body>
```

```
<h1>This is a Header</h1>
```

```
<p>This is a paragraph</p>
```

```
<p>This is a second paragraph</p>
```

```
</body>
```

```
</html>
```

- Now add the modified index.html file to the staging area and then do a third commit.

### Checking out a file from an earlier commit

- To check out the index.html from the second commit, find the number of the second commit using the git log, and then type the following at the prompt:

```
git checkout <second commit's number> index.html
```

### Resetting the Git repository

- To discard the effect of the previous operation and restore index.html to its state at the end of the third commit, type:

```
git reset HEAD index.html
```

- Then type the following at the prompt:

```
git checkout -- index.html
```

- You can also use *git reset* to reset the staging area to the last commit without disturbing the working directory.

### Conclusions

At the end of this exercise, you should have learnt some basic Git commands. Experiment with these commands until you fully understand how to use Git.

---

## 06.Exercise (Video): Online Git Repositories

Online Git repositories enable you to store a copy of your Git repository online and it can easily be shared among multiple computers and multiple users.

Let's learn about two of the online Git repositories service providers, GitHub and Bitbucket and look a bit more detail.

We'll look at how we can set up our local Git repository to be mirrored in an online Git repository.

There are several online Git repository service providers.

Two of the most popular ones are GitHub and Bitbucket.

I will go through in more detail about how you can set up your Git repository that you prepared in the previous exercise to be mirrored on an online Git repository online Bitbucket.

To setup an online Git repository, go to one of these online service providers and then sign up for an account.

Here, I have Signed up into my Bitbucket account and I have my home page open here.

I'm going to create a new repository online on my Bitbucket repository, so I just click on the Create repository.

And then this would come up with some details. Now this varies with the repository service provider.

Here I will specify the name of the repository as git-test and then I am going to mark this as a private repository.

For this course, and all the remaining courses in the specialization, I would strongly advise you to keep your repositories as private repositories.

Because you don't want somebody else to be copying the code that you might save in your online Git repository.

Please make sure that you only use private repositories in this specialization.

I will sign up as a Git private repository and I'll simply click on Create repository, and then it'll do some set up and then be ready for me, okay?

Now, what I need to do is to copy this particular URL for my repository.

You will see a URL for the repository in the home page of the repository.

Just copy that URL for the repository because we would need that in order to synchronize our local Git repository with this repository.

I'm going to copy that and since I've already created my local Git repository, I'm going to go in there and set that up to be mirrored in this online Git repository.

For that, we need to use a couple of Git commands that enable us to do this.

Going to the command prompt in my Git repository folder, I will type, `$ git remote add origin`, and then paste the URL for my Bitbucket repository that I've just created and hit the Return.

Now my local Git repository's remote origin is sent to the Bitbucket repository.

Now, I want to be able to push the entire contents of my local Git repository to my online Git repository.

Reviewing the commands for setting up the online Git repository.

The first one that we just did, `git remote add origin`, and the repository URL and so this will add the online repository as the remote repository for my local git repository.

Once I do that, then I need to push my contents to my online repository.

For that, I do \$ `git push -u origin master`.

And this command would push the local git repository to the online repository.

Once I type this at the command prompt and hit the Return, it's going to set up my Git repository.

Sometimes it may ask you to type your credentials including your password on your Bitbucket account.

You may have to do that, I had already done that previously so that's why it didn't ask me again for

the password. But once I type that in, then the contents of my local git repository will be pushed to the server side.

Now, the data on my local Git repository is now matched on my server side.

Let's go to Bitbucket to see the status of my online Git repository now.

Going to my Bitbucket and then reviewing the online Git repository you would see that there has been a branch, the master branch, that has been pushed to my online Git repository.

I can then examine the source by looking at the source here, and you can see that I have the index.html file which is exactly the same as what I have in my local Git repository.

And examine the various commits also so I can look at the state in the second commit and the first commit.

You can now see that all this information has been uploaded to the online repository including the history of all the commits.

Then going back there you can see more details about all the commits that have been sent to the server side.

I can examine each of those commits in a bit more detail and then going back to the source, I can look at more details of the source like that there.

That has now setup my local Git repository to be mirrored in Bitbucket.

The procedure is pretty much similar even on GitHub.

Let me set up an online GitHub repository.

Now obviously, you have to remember that your local repository can only be matched to one online Git repository.

I will only go through setting up the repository on GitHub, and you will notice that the same kind of commands will be required if you want to set up your local repository to be mirrored on a repository on GitHub.

On GitHub, if you log into your account, you would see something like this in your homepage. You can go to the right-hand side where there's a plus sign and then click on that to create a new repository, or just click on this button here to say New Repository.

And then when that comes up, I can simply say git-test, and then again, as a reminder, please make that a private repository.

Most of these providers allow you to now store private repositories online, so why make them public unless you really want to share it with other people.

I would strongly urge you to keep your repositories private for the moment, unless if you're working with a team. I should remind you that, at this moment, GitHub supports private repositories only for those subscribers who pay for their GitHub account.

It is not available for subscribers who are using their free account.

Simply click on Private and then create repository.

And that will create repository and then you see the GitHub also gives me a batch of commands here on how to set up the repository so far creating new repository on the command line.

The commands that we actually have done earlier or to push an existing repository that seen two kinds of commands that have to be issued.

That basically sets up your GitHub repository to mirror your local repository.

Since I have already linked repository to Bitbucket, I'm not going to use my GitHub repository the purpose.

As you notice the procedure is pretty much similar on either one of them.

Give and take a few differences in the way the information is rendered on the screen, in the user interface and a little bit of changes.

More or less they are similar in the way you're going to make use of them, in terms of mirroring your local Git repository to the online repository here.

For this, suppose somebody gives you a Git repository, can you make a copy of that on to your local one?

This is where we use another Git command called git clone.

Here, you see that I have this git-test repository.

What I am going to do,

is I'm going to go to this repository and then copy this URL.

And then I'll show you how I can create a copy of the Git repository or clone this online repository into my local computer.

Let's pretend that I am on a different computer.

And then we'll go to our command line and then create another repository with that same name.

Going to my command line, I'm just going to move up into my, and you see that I have my git-test folder there.

I'm going to create another temporary folder here just to show you that I can clone an online Git repository.

In my temporary folder you see that it is empty.

To review the command, it says git clone and that repository's URL.

Let's apply this command then clone the online Git repository.

Pretending that we are on another computer, so I'm going to say git clone.

And then paste the URL of my Bitbucket repository here.

And then you would see that that repository would now get cloned into a local folder with the same name, git-test.

Now if I do a listing of the directory, you can see that the git-test folder has been created.

Let's go to the git-test folder and then you will see that this is an exact copy of the folder structure that we had created earlier.

This is how you can clone an online Git repository.

All you need to do is to obtain the URL of the Git repository and then simply use get clone to get a copy of that onto your local computer.

With this, we complete this exercise on using online Git repositories. Did you finally get it?

## **Exercise (Instructions): Online Git Repositories**

### **Objectives and Outcomes**

In this exercise you will learn about how to set up and use an online Git repository and synchronize your local Git repository with your online repository.

At the end of this exercise, you will be able to:

- Set up the online repository as a remote repository for your local Git repository
- Push your commits to the online repository

- Clone an online Git repository to your computer

## Setting up an Online Git repository

- Sign up for an account either at Bitbucket (<https://bitbucket.org>) or GitHub (<https://github.com>).
- Then set up an online Git repository named `git-test`. Note the URL of your online Git repository. Note that private repositories on GitHub requires a paid account, and is not available for free accounts.

## Set the local Git repository to set its remote origin

- At the prompt, type the following to set up your local repository to link to your online Git repository:

```
git remote add origin <repository URL>
```

## Pushing your commits to the online repository

- At the prompt, type the following to push the commits to the online repository:

```
git push -u origin master
```

## Cloning an online repository

- To clone an online repository to your computer, type the following at the prompt:

```
git clone <repository URL>
```

## Conclusions

In this exercise you have learnt to set up an online Git repository, synchronize your local repository with the remote repository, and clone an online repository.

---

## 07.Node.js and NPM

JavaScript which was designed as a scripting language for the browser, has seen deployment far beyond the browser.

Node.js has played a significant role in this shift of JavaScript from the browser to the desktop.

Let's now learn a little bit about what node.js is and what role does NPM, the Node Package Manager, play in the context of node.js.

Node.js as I mentioned earlier, allows us to bring the power of JavaScript to the desktop.

Node.js is based on the JavaScript runtime engine that has been built for the Chrome browser.

The Chrome V8 JavaScript engine has been ported from the browser to run on the desktop and support the execution of JavaScript programs on the desktop.

Node.js is built around an event driven non-blocking I/O model which makes it very efficient to run JavaScript programs on the desktop and synchronous Javascript on the desktop.

This is where node finds its true pouch.

Right now, we will examine Node.js In the context of its use as a JavaScript runtime.

We'll look at the server-side application of Node.js In detail in the last course of this specialization.

This is the typical architecture of Node.js.

In this, the Chrome V8 engine is at the bottom layer together with libuv forms, the layer that interacts with the underlying computer system to support the execution of JavaScript programs.

On top of it, we have Node Bindings which are also implemented in C++.

At the top layer, you have the Node.js and standard library which are all implemented in JavaScript, and this is what enables us to write JavaScript programs and run them on the desktop.

Naturally, the ability to run JavaScript programs on the desktop energize the web development community to explore using JavaScript to develop a significant number of web development tools.

Tools such as Bower, Grunt, Gulp, Yeoman, and many others.

We will explore some of these in the later part of this course and in subsequent courses.

The last course in the specialization, as I mentioned, looks at the use of Node.js on the server side.

How we can develop Web server, Business logic, all implemented in JavaScript on the server site.

Together with Node, you often hear people talking about the Node Package Manager or NPM.

When you install Node on your computer, NPM automatically gets installed.



The Node Package Manager is the manager for the node ecosystem that manages all the node modules and packages that have been made publicly available by many different users.

A typical node package consists of JavaScript files together with a file called package.json which is the manifest file for this node module.

We will look at how we can use the package.json file in more detail in the subsequent exercises.

---

## **08.Exercise (Video): Setting up Node.js and NPM**

So, are you all excited and rubbing your hands eagerly to get hold of Node.js?

You don't need to wait for too long. Let's just head over to [nodejs.org](https://nodejs.org) and get hold of Node.js.

In your browser, head over to [nodejs.org](https://nodejs.org) and as you browse down you see the download buttons for Node.js for your specific platform.

In this case it shows the download buttons for Mac OS.

Click on the current version of Node.js.

Click on that to download the installer package for your platform.

Once you get hold of the installer package, double click on it to start installing Node.js.

Depending on your operating system, you will see a window like this on your computer and just follow along the instructions to install Node.js on your machine.

You may need administrator privileges to install Node.js on your machine.

Make sure that you are all logged in into an administrator account, and install Node.js to be accessible by all users that logged in on the computer.

And once the installation is complete, let's go ahead and verify that Node.js has been installed correctly.

Open up a terminal window or a command window, and at the prompt type `node -v`, to check the version of Node installed.

Similarly, check `NPM -V` to check the version of NPM installed.

I am starting with these versions of Node and NPM, and the subsequent exercise will be based on these.

Even if you install the latest version, I'm pretty sure that it would be backward compatible, so all the steps should work pretty much the same.

With this, we complete installation of Node.js.

So, let's go ahead and make use of Node.js in the next exercise.

---

## Exercise (Instructions): Setting up Node.js and NPM

Note: Make sure you have installed Git on your machine before you install Node.js. Please complete the previous Git installation exercise before proceeding with this exercise.

### Objectives and Outcomes

In this exercise, you will learn to set up the Node.js environment, a popular Javascript based server framework, and node package manager (NPM) on your machine. To learn more about NodeJS, you can visit <https://nodejs.org>. For this course, you just need to install Node.js on your machine and make use of it for running some front-end tools. You will learn more about the server-side support using Node.js in a subsequent course. At the end of this exercise, you will be able to:

- Complete the set up of Node.js and NPM on your machine
- Verify that the installation was successful and your machine is ready for using Node.js and NPM.

### Installing Node

- To install Node on your machine, go to <https://nodejs.org> and click on the Download button. Depending on your computer's platform (Windows, MacOS or Linux), the appropriate installation package is downloaded.
- As an example, on a Mac, you will see the following web page. Click on the Download button. Follow along the instructions to install Node on your machine. (Note: Now Node gives you the option of installing a mature and dependable LTS version and a more newer stable version. You should to install the LTS version. I will use this version in the course.)

Note: On Windows machines, you may need to configure your PATH environmental variable in case you forgot to turn on the add to PATH during the installation steps.

### Verifying the Node Installation

- Open a terminal window on your machine. If you are using a Windows machine, open a cmd window or PowerShell window with **admin** privileges.
- To ensure that your NodeJS setup is working correctly, type the following at the command prompt to check for the version of **Node** and **NPM**



```
node -v
```

```
npm -v
```

## Conclusions

At the end of this exercise, your machine is now ready with the Node installed for further development.

We will examine web development tools next.

## 09.Exercise (Video): Basics of Node.js and NPM

We will set up a package.json file for git-test folder that we have been working with so far, then we will set up a node module called as lite-server that will serve up contents of our git-test folder and then we can browse this index.html file and other files in a browser, and we will also see how the lite server will

enable us to automatically see updates to our browser window, as we make changes to our `index.html` file or any other files in our `git-test` folder.

The `lite-server` is something that we are going to extensively use in this and future courses to be able to see the changes in real-time in a browser window, as you edit the files of your project.

As I mentioned, we want to set up the `package.json` file.

What exactly is `package.json` file that we're going to set up?

Here I have some information from the `npmjs.org` site, which specifies what exactly is the role of the `package.json` file.

The `package.json` file serves as the documentation on what all other packages that your project is dependent upon.

For example, when you set up the `lite-server` for your project that will be recorded in the `package.json` file, so that subsequently you can also make use of that package in the future.

Also, it allows you to specify which specific version of a package that your project is dependent on.

Even if the package that you depend on changes in the future, you may insist that you want the user to install only a specific version of the package for use within your node application, and also it makes your builds reproducible.

Which means that when you share your code with others, then they can also do installation of all the node modules, as we will see later in this exercise on their own computer.

Obviously, your next question would be how do we create the `package.json` file?

If you are starting a new project where you want to initialize the `package.json` file then simply type `npm init` at the prompt in the project folder and that will take you through a set of steps, which will enable you to configure your `package.json` file.

Let's proceed with that for our `git-test` project.

Open a terminal window on a command window and go to the `git-test` folder, and at the prompt type `npm init`, then follow along the questions that are asked.

For the name of the project, we will just leave it as the default `git-test`; for version, we'll just leave it as one point zero point zero, we can edit that later; for description, "`this is a test directory to learn git and node.`", and then at the entry point, I would just say `index.html`.

Usually, if it is a node or package, the entry point would be `index.js`.

Now, this folder that we have set up is a `index.html` based folder; so that's why I just typed in `index.html`; test command, nothing; git repository, if we had already set up the git repository in the previous exercise, it'll automatically prompt that for you.

If not, this would be empty and give you an option to type in the git repository URL in case you're using an online repository; some keywords for your project, I'm going to leave blank; author, type your name, let's be narcissistic; and license; and then it'll show you the configuration of the `package.json` file in json format.

If you are familiar with json, this will look very familiar.

If this looks all good, let's just say okay and then that results in the creation of the `package.json` file.

Now, if you list the folder contents, you will see the `package.json` file in the folder contents.

Open the `git-test` folder in your favorite editor and then take a look at the contents of `package.json` file in your editor.

As the next step, we will learn how we can install a node module using `npm`, the node package manager.

We're going to install this node module called `lite-server`.

The `lite-server` will serve up the contents of this `git-test` folder in a server that it starts up, so that you can view the contents in a browser.

Given that we have an `index.html` file, if we sum up this folder then it will be a website and you can view the `index.html` in a browser.

Let's set up the `lite-server` and then we will see how we can make use of the `lite-server`, to serve up the contents of this folder.

This is very useful because if you are working on a web development project, you want to see the live version of your web development project, so that as you make changes to your project, you can see the changes immediately reflected in the browser.

This is a very good node package that is very useful for this purpose.

Let's setup the `lite server`.

To do that at the prompt type in `npm install`.

Notice, if you want `npm` to install a node package, this is how you're going to invoke it, and then you'd say `lite-server` and then, we also want to save the fact that our project is using the `lite-server`.

We will save this information in the `package.json` file.

To do that, you're going to type in `minus minus save-def`. Now, the `save-dev` option specifies that this `lite-server` is used for development dependency for our project.

If you are installing a node module on which your project is directly dependent on, then you would install it by simply saying `minus minus save-option`.

Let's go ahead and install it. And you wait patiently for the installation to take place. It'll take all of a few minutes for that to complete its installation.

Once that is installed, then you would immediately notice when you look at the contents of your folder, you'll immediately notice that there is a folder there created named `node_modules`.

Now, if you're going to the `node_modules` folder, you will see a whole bunch of other subfolders in there which contain node modules, which are necessary for the `lite-server` node module and so on.

Let's take a quick tour of the `node_modules` folder to see what the contents of these are.

Going to my `git-test` folder, if you're going to the `node_modules` folder, you would see, as I said, a whole bunch of subfolders there.

Normally, you don't need to be venturing into the `node_modules` folder.

They just exist there because they are needed for the `lite-server`.

As you browse through, you should notice a folder named `lite-server` here.

When you go into the `lite-server` folder, note in particular the presence of the `index.js` file and then `package.json` and several other things.

The contents of this folder comprise the `lite-server` node module.

But, this `lite-server` node module is dependent on other node modules to provide it with some additional functionality.

That's the reason when you install the `lite-server` node module, it will in-turn install many other node modules on which the `lite-server` itself is dependent on.

That's the reason why you see that explosion of those folders inside the `node_modules` folder. Don't be too concerned about it. The sum total of all these will not be more than a few tens of megabytes. So, it is not going to fill up your directory with junk. This is all essential for node to be able to help you.

In case you're curious about the `lite-server` and how it works and so on, you can always go down to this GitHub site where the `lite-server` is hosted and then look up the documentation for `lite-server`. I will introduce you to whatever you need to know about `lite-server` as we go through this course and the remaining

courses. So, you don't need to worry too much about it. But just in case you're curious, you can always go to this site to find out more details about lite-server. The link is provided in your exercise instructions and the additional resources are part of this lesson.

Once you have completed that, then head over to the editor, where you have the git-test folder open and then view the contents of the package.json file. So, you would see that the package.json file contains exactly the information that you are configured with your NPM init. You would see the name, version, repository, author, and in particular, note this information here says devDependencies, and then it specifies the lite-server and also a notice it says that 2.2.2, which means that this particular project depends upon lite-server that is at least version 2.2.2 or higher. This is very useful for us. Now, why do we need this information here?

Later on, when you go to the other exercises, you will notice that when you store this on an online repository, you don't want to be storing everything in your node modules folder. We would only be storing information of all the files that we have created. The node modules folder can always be recreated by typing npm install at our command prompt and then based upon the devDependencies and dependencies that are listed in the package.json file, all the node modules that your project depends on will automatically be installed. We will see that later on on how to use npm installed in this course. Now that we are at package.json file, let's make a couple of edits so that we will be able to make use of the lite-server to serve up their content. Right here, in this option called scripts, let's add in one more here; so, we will say start. So, start is a command that npm supports, which enables you to specify a bunch of things that will be started. So, later on, we will see how we make use of this. So, here, I'm going to say npm run lite. And after the test, I'm going to add in one more entry called lite, which I will configure as lite-server. Okay. With these changes, let's save that package.json file, and now our project is configured; so that now if you're tied to the lite-server, the contents of your folder will be now served up in your favorite browser. Heading back to our command prompt, at the prompt, if I type npm start, now you see why I put that entry called start into my package.json file. So if I say npm start, whatever that start is configured as in the package.json file, we specified that as npm run lite and the lite was specified as lite-server. So, essentially, we are saying start the lite-server. So, once I type npm start, it will start the lite-server and it'll serve up the contents of this folder. Now, how do you access the contents of this folder? If you want to access locally, you will access it by specifying the url as localhost:3000. This is the default settings for the lite-server. Furthermore, this should automatically open the browser window of your default browser and then show the contents of index.html in that browser window. Here, you can see that I have opened my editor and my browser window

directed at localhost 3,000 simultaneously side by side, so that we can see how the browser window will immediately reflect any changes that we make to our files in the Git test folder. So, let me go to index.html. And then for the sake of space, I'm going to turn that over. So here, you can see that this is the contents of this. And then now, let me add in one more paragraph. And save the changes. And then you would immediately notice that the change that I made to my index.html file is reflected into my browser. This provides a very nice way of being able to observe in real time the changes that you make to your code being reflected into your browser. So, when you are working on a project, it will be very appropriate for you to be able to see the changes immediately. When you make a change and then save the file, the modified code is immediately loaded into your browser, so you can immediately see the change being reflected in your browser window. This is a very useful tool while you are doing development of your project. That is the reason why I introduce you to the light server and set it up so that we can make use of it as we develop the website in this course. If you recall, we had already set up our Git test folder to be a Git repository. So checking again, we will see that we already have three commits in our Git repository. And those Git repository is already Midhurst to our online Git repository which we have set up in the previous exercise, either at Bitbucket or GitHub. My Git test folder is synced to my Bitbucket repository in this particular exercise. So, what I'm going to do now is to show you how you can exclude some folders from your project folder and then make sure that they are not synchronized to your online repository. Now, as I said, the node modules folder can always be recreated by typing NPM install at the prompt. So, that's why when you upload the contents of your folder to an online Git repository or when you do a commit of the folder to your Git repository, you don't want the node modules folder or all the subfolders under it to be included in the commit. So, how do we exclude some folders or some files from our folder from being checked in into our Git repository? So to do that, we will set up a file named dot Git ignore. So that's the name of the file, dot gitignore. To create this dot gitignore file, we will go to our editor. So in the editor, in the Git test folder, I'm going to create a new file and I will name it dot gitignore. Note that the name begins with a dot and then the rest of the name is G I T I G N O R E. So this is very very important that you're set up the file with exactly that name, dot gitignore. Let's create this file called dot gitignore. And the first line of that file, we will type as node modules. What this means is that the nodes modules folder is going to be excluded from our git commit. Once I create the dot gitignore file, and then add node modules into the dot gitignore file. Let's save the changes. And then, we will now do a commit of the current state of our project into our Git repository. I hope you remember your Git commands. Let's do a Git status. And then when you do that, you will immediately notice that you have the index.html file marked as modified. And



then the two new files, Git ignore and package.json. So, we do a Git add dot and then do a Git status. And then you'll see that all these new files have been checked into your commit. Let's do a Git commit, Git commit minus M fourth commit. And the files are committed. Let's push the new commit to our online repository. So to do that, Git push minus U origin master. And wait for it to be pushed to our servers. Now, if you go to your online Git repository, you'll see that the package.json file and dot gitignore would have been checked in into your Git repository. Going to my Bitbucket repository for the Git test, you will see that when I look at the source, you will see that the package.json file has been added. The dot Gitignore has been added. And the new index.html file has been checked in. So that completes this exercise. In this exercise, we have learned how to set up a package.json file using NPM init. We have learned how to install an NPM module. And we have learnt how to use the light server, NPM module to serve up the contents of our project folder so that it can be viewed in a browser. So, this is a nice way of serving up your web contents, your web application or your website, so that you can see changes in real time being reflected to your browser window. And then, we also saw how we can set up the dot gitignore so that some folders can be excluded from being checked into our Git repository. This completes this exercise. So with this, I'm sure you would have gotten a good handle on the use of both Git and then also node and node modules. Don't worry. We will be using node extensively in various ways as you go through the courses of this specialization. This is just a start.

## Exercise (Instructions): Basics of Node.js and NPM

### Objectives and Outcomes

In this exercise you will learn the basics of Node and NPM. At the end of this exercise, you will be able to:

- Set up package.json file in the project folder for configuring your Node and NPM for this project
- Install an NPM module and make use of it within your project

### Initializing package.json

- At the command prompt in your **git-test** folder, type

```
npm init
```

- Follow along the prompts and answer the questions as follows: accept the default values for most of the entries, except set the entry point to index.html
- This should create a *package.json* file in your **git-test** folder.

## Installing an NPM Module

- Install an NPM module, `lite-server`, that allows you to run a Node.js based development web server and serve up your project files. To do this, type the following at the prompt:

```
npm install lite-server --save-dev
```

- You can check out more documentation on `lite-server` [here](#).
- Next, open `package.json` in your editor and modify it as shown below. Note the addition of two lines, line 7 and line 9.

```
{  
  "name": "git-test",  
  "version": "1.0.0",  
  "description": "This is the Git and Node basic learning project",  
  "main": "index.html",  
  "scripts": {  
    "start": "npm run lite",  
    "test": "echo \"Error: no test specified\" && exit 1",  
    "lite": "lite-server"  
  },  
  "repository": {  
    "type": "git",  
    "url": "git+https://jogesh k muppala@bitbucket.org/jogesh k muppala/git-test.git"  
  },  
  "author": "",  
  "license": "ISC",  
  "homepage": "https://bitbucket.org/jogesh k muppala/git-test#readme",  
  "devDependencies": {  
    "lite-server": "^2.2.2"  
  }  
}
```

- Next, start the development server by typing the following at the prompt:

```
npm start
```

- This should open your *index.html* page in your default browser.

- If you now open the *index.html* page in an editor and make changes and save, the browser should immediately refresh to reflect the changes.

## Setting up .gitignore

- Next, create a file in your project directory named *.gitignore* (Note: the name starts with a period) Then, add the following to the .gitignore file

```
node_modules
```

- Then do a git commit and push the changes to the online repository. You will note that the node\_modules folder will not be added to the commit, and will not be uploaded to the repository.

## Conclusions

In this exercise you learnt to set up package.json, install a npm package and start a development server.

---

# Setting up your Development Environment: Git and Node: Additional Resources

## PDFs of Presentations

Git.pdf

Git-Exercises.pdf

NodeJS.pdf

Exercises-Node-NPM.pdf

## Additional Resources (Git)

- Git site <http://git-scm.com>.
- [Installing Git](#) chapter from Pro Git
- [Git reference manual](#)
- Quick reference guides: [GitHub Cheat Sheet](#) (PDF) | [Visual Git Cheat Sheet](#) (SVG | PNG)
- [Atlassian comprehensive Git tutorial](#)

## Additional Resources (Node.js and NPM)

- [Nodejs.org](https://nodejs.org)
  - [Npmjs.com](https://npmjs.com)
  - [Node API Documentation](https://nodejs.org/en/docs/guides/node-api/documentation/)
  - [NPM Documentation](https://docs.npmjs.com/)
  - [lite-server](https://github.com/vercel/lite-server)
- 

## Module 4: Introduction to React: Objectives and Outcomes

In this lesson you will be given a quick overview of JavaScript frameworks and libraries and then introduced to React in particular. We will learn some basics of React and how to configure a React application using the `create-react-app` the command line tool. At the end of this lesson, you will be able to:

- Get a basic overview of JavaScript frameworks and libraries
  - Understand the architecture of an React application
  - Scaffold out a starter React application using *create-react-app*, the command line tool
- 

## 10.Front-end JavaScript Frameworks and Libraries Overview

Before we begin examining React in more detail, let's take a step backwards and ask ourselves a couple of fundamental questions. What are JavaScript frameworks and libraries? Why do we need them? And what do they help us accomplish that we cannot do with standard Vanilla JavaScript? Let's examine a few of these questions in more detail in this lecture first. Now the first question, why JavaScript frameworks or libraries? Now, as you've seen in the previous course, and from your prior experience with HTML CSS is in JavaScript, you can easily accomplish a lot of things using plain HTML CSS in JavaScript or a front-end web UI framework like Bootstrap together with jQuery. Indeed, many complex websites can easily be implemented by using this. Now, then the question comes, why JavaScript libraries or frameworks? Now, to answer this question, the first thing that we need to realize is that when you need complex manipulations of the DOM, especially by fetching data from a server and then update the DOM, it gets fairly complicated using tools like jQuery and plain Vanilla JavaScript. So, that is where the JavaScript libraries and frameworks shine. In the web world, you might have heard people mention about application architectures like the Model-View-Controller approach, or the Model-View-ViewModel, or the Model-View-Whatever approach. And the binding of the model

and the view with the use of controllers and the view models. Now, if you haven't heard of these, don't worry, we will examine some of these in a bit more detail as we go along in this course. Also, in the context of React, you might have heard people mention about the flux architecture or a redux way of implementing Reactor applications. Now, what exactly is the role that this plays in developing a full fledged Reactor application? We will examine some of these as we go along in this course. But we realize that for a full-fledged web application, we will need a structured way of approaching this, and so that is where software engineering paradigms like the Model-View-Controller, Model-View-ViewModel, help us to structure our web application. As you implement more and more complex applications, you'll begin to soon realize that there are some standard approaches or standard patterns that you use when you implement your applications, and standard set of functions that you frequently reuse within your application. So, capturing this standard set of functionality into a collection of well-defined implementations of these behaviors or functionalities is what leads you into developing a software library. So, a software library gives you a well-defined collection of implementations or repeated behaviors that you can use to quickly implement complicated applications. And also, this is where the reuse of behavior is of paramount importance for us. And the fact that you can implement your applications in a model or fashion, leveraging the functionality provided by the software library. And indeed, examples of such software library in the web world include jQuery. Now, the next question, what is a software framework and how is it different from a software library? A software framework is an abstraction in which the framework provides a set of generic functionality that you can then customize by implementing it on user defined code. Angular, being an example of a software framework. So, the generic framework defines a standard set of behaviors that you can then customize to implement your specific application. So, this is where the framework provides a universal reusable environment with a specific functionality and then you can customize that by adding your own specific code. So, again, there are several examples of frameworks, Angular being one of the most popular, you have Ember and Backbone. So with a framework, a framework clearly defines how the application should be implemented and very often, it is highly opinionated in the way the application needs to be implemented. So, you are little bit constrained by the dictates of the specific framework and the way you have to implement your application in order to leverage the generic functionality that is provided by the framework. Let's ask ourselves a few more questions to distinguish a framework from a library next. Now that we have examined a software library and a software framework, you're probably curious about the difference between the two. So, to help highlight the difference, let me draw upon the information in AngularJS documentation in order to distinguish between a library and a

framework. So, a library, as we see, is a collection of functions that make it easy for us to implement web applications. So, when you leverage a library for implementing web applications, your code is in charge and then simply calls upon the functions provided by the library in order to accomplish some repeated common behavior. A framework, on the other hand, is a particular implementation of a web application where the framework provides generic functionality and your code fills in the details in order to customize that framework for accomplishing the specific application functionality that you want to implement. So, the framework, in this case, takes charge of the overall functionality of your application, and the framework, in turn, calls into your specific code that you implement in order to customize the behavior to accomplish the specific application functionality that you're trying to implement. Again, Angular, Ember, Backbone are all examples of frameworks. Now, depending on which one you use, the amount of opinionated approach that they take is different, Ember being the most opinionated among them, and Backbone being the least opinionated among these frameworks, and Angular is in between the two. Now, of course, let's ask ourselves about React as we go along in this course. Drawing upon a few salient features of the framework, we see one of the principles of how a framework is implemented, what I refer to as the Hollywood Principle. If you know how Hollywood operates, when actors or actresses want to find work, they approach an agent. The agent typically tells the actor saying, "Don't call us, we'll call you when we need you." So here, the agent takes over the role of what a framework does in the software world. So the framework will call upon their specific functionality that you implement as a web application designer in order to customize the generic functionality that the framework provides. This is where we often hear about the inversion of control. So, as you can notice from the Hollywood Principle, the framework takes control of how the application works and then the framework is the one that decides when and where to call your customized code in order to accomplish whatever needs to be done. So this is where we also distinguish between two different approaches to implementing applications, the imperative approach as opposed to the declarative programming approach. In an imperative approach, the application designer clearly specifies how the work needs to be accomplished or how the application needs to be executed. Whereas, in the declarative approach, the application designer simply specifies what needs to be accomplished and leaves it up to the framework to decide how the work is going to be accomplished. So, this is two different ways of achieving the same end goal. The imperative approach, when you specify clearly step by step, how it needs to be accomplished, as opposed to the declarative approach where you specify what you want to accomplish, but then leave the details up to the framework to decide how it is going to be accomplished. Again, don't confuse yourself too much into

trying to read all the details about this. These are some generic principles that govern how these various frameworks and libraries are implemented. But as we examine React in more detail, it'll become more and more clearer how this is actually impacting the way implement our React application. Also, within the JavaScript frameworks world, you often hear people talking about single page applications or rich internet applications. Now, we will look at how we implement a single page application using React as we go along in this course and we'll also understand what exactly is meant by some single page applications at that point. Similarly, we will often hear people, as I said, mentioning about architectures like the Model-View-Controller and the Model-View-ViewModel or the Model-View-Whatever. We'll also examine a little bit more about some of these approaches as we examine architectures that we can use for our React applications, especially the flux architecture redux towards the second half of this course. And we'll also examine how we achieve scalable, reusable, and maintainable JavaScript code and briefly touch upon test driven development towards the end of this course. There are many JavaScript frameworks and libraries that are in use in the real world. Here is an incomplete list of these and you often hear more and more such frameworks being created and being made available for the JavaScript work. So you are pretty soon bombarded with so many different JavaScript frameworks and libraries. And so, how do we navigate through this sea of frameworks and libraries and choose one framework or library that we want to use within our web application development? Again, depending on what is it that you're trying to accomplish, one or the other of these frameworks maybe more suitable for your specific use case. Among these, some of the most popular include React and Angular, Ember, Backbone, and Vue. Having briefly examined JavaScript frameworks and libraries, you are left with the obvious question in your mind, is React a library or is it a framework? Now, we will defer this question to the next lecture in this lesson.

## 11.Introduction to React

We ended the previous lecture with a question, is React a library or is it a framework and what exactly is React? Let's examine these questions in a bit more detail in this lecture. Let's start out with the very first question. What exactly is React and how is it different from the other frameworks or libraries that we have mentioned in the previous lecture? When you visit React website, you see it clearly specified right on the front page that React is a JavaScript library for building user interfaces. Now again, depending on who you ask some people tend to call React a library and others tend to call it a framework. Now, let's not bother ourselves too much in splitting our hairs over whether it is a framework, or a library but let's concentrate more on what it actually helps us accomplish. It is more important for us to understand that rather than worrying



about whether it is a framework or a library. The React approach to implementing them applications is what we are after in this course. So, React also states that it uses a declarative approach. Now that leaves you in a confused state because we saw that frameworks generally tend to use the declarative approach. But in React, the declarative approach used by React as specified on its website says that, it makes it easy to create interactive UIs with simple views for each state within your application. And also React takes care of automatically updating the UI and then rendering any changes to their specific components as required on your page. You just heard me mentioning the term Component. React Indeed is a component based approach. In a Component based approach, we encapsulate behaviors into small units called Components. We will examine Components in more detail in the next lesson. There it will become more clear to you how, and why a component based approach is useful for implementing our Web applications in React. Furthermore, React makes no assumptions about the entire technology stack that you're going to use for implementing your Web applications. React plays well with any technology stack that you can use behind the seats. React itself concentrates only on the user interface side of the story, and that leaves it up to the application designer to decide how they want to implement the architecture and how they want your application to interact with the back-end server. So, as we go through this course, we will examine one approach that we use for implementing the entire technology stack which includes the Flux architecture approach, and in specifically the use of Redux for implementing a state based storage for our Web application and also the use of Fetch for interacting with our back-end server. Again I've mentioned a few terms like Flux, Redux, and Fetch. We will examine these towards the second half of this course. It's obviously useful to examine the history behind the React approach. So, to understand where React originated, and how it came about to the state it is today. React was first designed by Jordan Walke who was part of the Facebook team. It was first deployed for Facebook's news feed around 2011. Subsequently in 2013, React was open sourced at this JS conference. React took off as an approach for implementing Web applications from then onwards. React is designed for speed, speed of implementing the application, simplicity, and scalability. So, the three essence of React, and why it has become so popular in the real value. So, as we examine React more in this course, you'll become more and more familiar with why this approach is very suited for implementing Web applications. As you enter the React world you will be bombarded with a lot of vocabulary that is used in the React world. So, you will hear people talking about One-way data flow especially in the context of the Flux Architecture. You will often hear people mentioning about JSX, we will examine JSX in the very next lecture and understand what role it plays in developing a React application. We'll hear about Components which we will



examine in the next lesson and also in the second module of this course in more detail and we'll hear about the state and how a React Component interacts with the state of your application and the way you store the state of your application, or do you store the state in a specific component. We'll also hear about Props, a way of passing data between the various components. Also we'll hear about Virtual Dom, and how is it different from Real DOM. Why React manipulate the Virtual DOM, and how the Virtual DOM eventually gets incorporated, or rendered onto the Real DOM. And Element, the React Element, which is the smallest unit of building up a React application. A component being a collection of React elements. Then we hear about the Flux and Redux architectures in a bit more detail. Again, as we go along this course we will examine these concepts and these terminology in more detail. Again, don't get overwhelmed with the vocabulary that you hear in the React world. If you learn step by step, you'll begin to pretty soon get a very good handle on all this vocabulary and you can easily go ahead and impress people by throwing these words at them, and trying to impress them or how much you know about React. So, this is the jargon that you will end up learning also at the end of this course. Enough of the jargon. Let's go ahead and get our hands dirty by starting to build a full fledged React application, which will form part of all the exercises as you go through the rest of this course. We will start with our first exercise where we'll install the create-react-app, which we will use to scaffold out our very first React application in the first exercise, and then we will start building upon this application throughout the remaining exercises of this course.

## **12.Exercise (Video): Getting Started with React**

Now that we have examined React briefly in the previous two lectures, I'm sure you're curious about getting started with React, getting your hands dirty with starting out on a React application. So in this exercise, we will look at how we will get started with React. I'm sure by now your computer is already configured with Node and you have access to NPM, the package manager that comes with Node.

Just like NPM, Yarn is yet another package manager that is very useful, especially in the React. In this course I'm going to use Yarn for installing and downloading all my Node modules for my React application. If you choose to instead stay with NPM that is perfectly fine. You can use NPM to accomplish exactly the same thing.

Whenever I use Yarn, I will also remind you about how you can use NPM to do exactly the same set of commands that you do with Yarn.

If you choose to use Yarn, it is important to get started first installing Yarn on your computer. So where do we get ahold of Yarn? If you choose to use Yarn

instead of NPM for your React application, then I would suggest you go to this site called [yarnpackage.com](https://yarnpackage.com) and then you Instructions there on how to install Yarn on your computer.

So, going to the getting started page you will see the installation instructions right there for your specific operating system. And so please follow along the instructions if you choose to use Yarn instead of using NPM. If you choose to use NPM that is perfectly fine. As I said, I will tell you both approaches as we go along in the exercises of this course, but I find that Yarn is much better suited when I am working with the React application. So that's why I choose to use Yarn.

So, go ahead and install Yarn on your computer as specified in the installation instructions here. If you go to the React site and then look up the documentation and especially in the installation, you would notice that there is a link to saying add React to a new app. And when you click on that the approach that is suggested by React is to use this create React app. Which they say is the best way of starting out building a new React single page application. In this course that is indeed the approach that I am going to take in order to make use of React with in this particular course. To get started open a terminal window or a command window on your computer and at the prompt type `npm install -g create-react-app@1.5.2`. As we go through this course, I will specify exact versions of the various NPM packages that I will use in this course. This is to ensure that when you go through this course, you will install the exact same versions of the NPM packages that I use in my course so that your journey through this course will be painless. If you are doing this installation on a Mac or a Linux machine, don't forget to put `sudo` in front of the NPM install because we are installing this as a global package here. And wait for the installation to be completed. So once this is completed as you can see, I am using create-react-app version 1.5.2 in this course. After you install create-react-app you may need to restart your terminal or your command window in order for the create-react-app command to be made available at your prompt.

Once you get restarted with your terminal or your command window at the prompt type `create-react-app --help` and that will print out a set of instructions on how we can use the create react app to create a React application.

So, once we understand this then we will get started creating our first React application. So to create our first React application let me go to a convenient location on my computer. So I will go into my documents Coursera folder. And in this folder I will create a new subfolder named React.

So, let me create a new subfolder named React. Again, if you are using a Windows machine, you can appropriately create the react some folder and then move into the React subfolder. And as you notice, this is empty at this moment.

To scaffold out your very first React application at the prompt type `create-react-app` and then the name of your application as `confusion`. It will become clearer to you why I named this application as `confusion` as you go through the rest of this course, and wait for the `create-react-app` to create your React application. It will take some time for this to complete its work.

Once it is completed doing its work you will notice that there will be a sub folder named `confusion`.

So, move into this subfolder to get started on your React application.

You can just type `yarn start` or if you are using NPM you just type `npm start` and your React application will start up, and be served using that built-in server that is part of the `create-react-app` ecosystem. And once the react application is compiled then it'll become available at this address.

So, go to a browser and type this address into the address bar and then you will be able to view your React application. Going to my browser I have typed in that address into my address bar. And then you see the resulting React application here. We will examine the details of this React application in the next lecture. If you are going to initialize a git repository for your React application, which I would strongly suggest you do. So then, go to the `confusion` folder in another terminal tab or another command window. And then at the prompt type `git init` to initialize the git repository and then we can do a `git status` to look at the states of our folder and then do a `git add .` to add all the files from the git repository into my git staging area. And then again, if I do `git status`, you'll see all the files that have been checked into the staging area of my git. And then let's do a `git commit`, so at the prompt type `git commit -m "Initial Setup"`.

So, this is the starting point for our React application.

And so, then when you say `git status` you will see that your commit has been saved into the git repository. And then you can do `git log` and note that your very first commit has been put into the git repository there. Thereafter you can synchronize this git repository with an online git repository either on GitHub or on bitbucket. Make sure that your git repository is a private repository. With this we complete this very first exercise where we have used `create-react-app` to scaffold out our very first React application. We will examine what has been scaffolded out in the next lecture and thereafter we will start working on further developing our React application.

## **Exercise (Instructions): Getting Started with React**

### **Objectives and Outcomes**

In this first React exercise, you will first install *create-react-app*, the command line tool for scaffolding React applications. You will then use the tool to scaffold out a basic React application. We will thereafter develop this application into a full-fledged React application in the process of doing the exercises in this course. At the end of this exercise you will be able to:

- Install *create-react-app*
- Scaffold out a basic React application

## Installing Yarn

- Yarn is another package manager like NPM, but is better suited and faster to work with for React applications. So let us install yarn and use it for building our React applications.
- To install Yarn, you can find the instructions for your specific platform at <https://yarnpkg.com/en/docs/install>.
- If you choose not to install Yarn, you can continue to use npm in place of yarn without any problem.

## Installing create-react-app

- From the React documentation we learn that the *create-react-app* CLI makes it easy to create an application that already works, right out of the box. It already follows the best practices suggested by the React community!
- To install *create-react-app* globally, type the following at the prompt:

```
yarn global add create-react-app@1.5.2
```

Use *sudo* on a Mac and Linux. Alternately you can use npm, by typing "npm install -g create-react-app@1.5.2".

- This will make the command line tool for creating React applications. To learn more about the various commands that this CLI provides, type at the prompt:

```
create-react-app -help
```

## Generating and Serving a React Project using create-react-app

- At a convenient location on your computer, create a folder named *React* and move into that folder.
- Then type the following at the prompt to create a new React application named *confusion*:

```
create-react-app confusion
```

- This should create a new folder named *confusion* within your *React* folder and create the React application in that folder.

- Move to the *confusion* folder and type the following at the prompt:

```
yarn start
```

- This will compile the project and then open a tab in your default browser at the address <http://<Your Computer's Name>:3000>.
- You can initialize your project to be a Git repository by typing the following commands at the prompt:

```
git init
```

```
git add .
```

```
git commit -m "Initial Setup"
```

- Thereafter you can set up an online Git repository and synchronize your project to the online repository. Make sure that the online Git repository is a *private* repository.

## Conclusions

In this exercise you installed the create-react-app CLI tool and created a basic React project and served up the compiled project to your browser.

---

## 13.React App Overview

Now that we have scaffolded out our first React Application using create React App, you're obviously, curious about how this application is actually constructed, and how does it correspond to what we see in the browser as we have seen in the previous exercise. So, let's pay a quick visit to the folder, the confusion folder where the application has been scaffolded out, and then take a look at several files that have been created there by the create React App in order to understand the structure of our React application. Taking a closer look at our React Application, you'll notice, what is contained in the view that we see here in the browser. So, you can notice that, there seems to be a logo, that is continuously rotating. There seems to be some text here and then a comment here saying to get started, edit source/app.js to C and save to reload. Now, having seen the structure of that page as rendered in the browser, let's now go to see how this is actually created, by our React Application. So here, I have my React Application open in Visual Studio Code, and in the view here, you can see that in the confusion folder, there are 2 some folders here, public and source. And then we have the node modules folder here. From your previous course, you understand what the node modules folder should contain. And let's pay a visit to some of these files in order to understand the structure of our react application. So, as you would normally do, If you look at a web page, you would be immediately drawn towards looking at what is inside the index.html page. So when you open

the index.html page, you can see a few things here where it says link manifest here, and then link shortcut. So this two refer to the manifest, and the favorite icon here, favorite icon here. And then down below here, you don't see anything else being important in the head of the file here, and you see the title of the page as React App. And then as you go down below here, you'll notice that this html page contains nothing except, for this body which contains this div id root here. And then there's no script here. Nothing else here to indicate that this is a React Application. So, that leaves us with very few clues. Let's pay a visit to a few more files, before we come back to see why this index.html page, is structured as seen here. So, after this, let's go into the source folder in the index.js file, and in the index.js file, also you'll see a few imports here and some line here saying ReactDOM render app and so on. That's about it. And this again leaves you with no clues about how your React Application is structured. But recall, we saw a comment in the browser window that we just saw a short while ago, which says, edit app.js file. So, maybe that's where all the action is. So let's pay a quick visit to app.js file. And when we come into the app.js file, you immediately notice some things that are familiar from what you have seen in the browser. You can see here this words, welcome to React, and which you obviously saw in the browser window there. And also you saw this to get started edit source app.js save to reload. So you seem to notice that something here indicates that this is sort of giving me the structure of the React Application that I see rendered in the page. And if you're thinking in that direction, you all getting there, but before we dive into the details here, let me introduce a couple of things about the structure of what you see here. Now let me draw your attention to this particular line here. It says h1 classname, app title, welcome to react and then slash h1. This looks like html code. Is it? That something that I will answer in the next lecture. But for the moment, let's go back and see what this represents in React. You saw me drawing attention to this particular line in the code there, which is h1 plus name app title, welcome to React there. Now this is the smallest building block for a React application, what we refer to as a React Element. In this React Element, you're noticing this h1 tag here, which seems to remind you of the html h1 tag. What this represents in React, is plain Javascript object. And so in React, all these elements are plain JavaScript objects, that are very cheap to create and so you can create a lot of them for use within your React Application. Now how exactly do we make use of them, that we will come to as we go through this course. But also, you have heard me using the term component in one of the previous lectures. A component in React is made up of elements like this. And a component in React as you saw in the code earlier, is defined like this. Class App extends Component. Now, having seen this, let's go back and revisit the code and take a closer look. Going back to our code, you'll notice that, in the app.js file, you have a ES 6 class definition here, saying class app extends component. So

obviously from your knowledge of ES 6, you'll know that component is some kind of ES 6 class which you are extending to create the app class. What is this component class? So if you look up from here, you notice that, in the very first line, it says import React and within braces component from React. So we are importing this from this React module here. Then component, class from there. And that is what we are using to create this subclass here called app. And then inside this App class, you'll see that there is a single method here called render here. The render method contains, as you can see React Elements that are grouped together. Innovate to create the view that you saw in the browser a little bit earlier. So, this is returned by this render method of this render function of this class component here. So, this is the typical structure of the React component. We'll deal with the React components a bit more in the next lesson where we will look at how you can create React components and so on. But this has already been scaffolded out for us, so that's why I'm drawing your attention to some of these details here. And also, you'll see the header in div and p being defined here, but notice that unlike HTML where we would specify a class, as in class equal to and then we'll specify the CSS class here. Instead, here, we are saying class name. The reason for using class name here is because since we are including this inside my JavaScript code here, if you don't use class name, instead if you just use class, this could be easily confused with this word, the reserve keyboard class that is used by es6 here. So, that is why in React, you would see that we will express any CSS classes that we attach to any element by saying, class name, and then this is of course CSS class here. Now, where are the CCS classes defined? You can open this app.CSS file here and then this is where the corresponding CSS code for all those CSS classes are defined here. So, that will show you how we are making use of this CSS classes inside here and applying them to these various React elements that we use within our JavaScript component here. Again, I have drawn your attention to a few things will come back to, why we define these things this way? And is this real HTML or something else? That, I will answer to you in the next lesson. But for the moment, you can now begin to see why you are viewing the browser is rendering exactly what we are seeing here. Similarly, here, you can see that there is an image with the source as specified as logo and so on. So, this looks very much like HTML code, but it is composed of React elements. Now, having seen the app.js file, let's now go to index.js file. Now, in index.js file, again, let me draw your attention to a couple of things here. So, this is again a JavaScript file here which contains imports of various items from the JavaScript modules here. So, you can see that you're importing React from React, and then you're importing ReactDOM from react-dom, and then you're importing the CSS file here, and then you're importing App from.App. So now, again, going back to App.js, you'll notice that if you scroll down here, it says export default App; So, this app is an export from



this App.js file. Also, notice that we are importing App.css here before applying it to our application. Again, switching to index.js, you can now see that your App is being imported from the App that you've just defined here. Let's leave this aside. We'll not discuss this at the moment, but notice how I am including a statement here, it says ReactDOM.render. So, this ReactDOM, as you see, is imported from react-dom and then you're calling the render method of ReactDOM and then for that, you are supplying the first parameter as App. So this is a tag here, the App here refers to this App that we are just importing, and so that is a self closing tag that we have enclosed here, and then the second parameter here that says document.getElementById('root'). Now from your knowledge of JavaScript, document refers to the HTML document that we have seen and then says getElementById('root') here. And so, you can interpret the statement saying ReactDOM.render. This particular thing, and then attach it to this DOM element in my index.html page. Now that we have interpreted this appropriately, let's now go to index.html and page and in the index.html page, let me draw your attention to this particular element in there written div id root here. That now completes the story. So what you can see here is that in the index.html page, I am rendering that particular DOM element, onto this element in my index.html page. So this is going to be replaced by what is rendered by index.js, and put in there in place of that, and which is nothing but what is included inside this app component, which is nothing but what we just saw here which is being rendered by the app component, and this is exactly what we saw being rendered to the browser window that we've seen. So now, you see the connection between all the three items in my React application code that we have seen here, being scaffolded around by Create React App. Again, coming back and summarizing what we have just seen here, how do we render the view to the DOM? So, to render the view to the DOM, as you noticed, we have used this particular statement which is in the index.js file that says ReactDOM.render(<App/>, document.getElementById('root')). And so, that is how we render that into the index.html page. And furthermore, where is it rendered? As we have seen, this is the div which is going to be used as the Root DOM node onto which that dom elements created by my app.js is rendered in my page. And so, that is how the view of your React application gets created. So, now that we have seen all these different parts, you'll see how the whole thing comes together to render our React application. With this quick examination of what we have just scaffolded out, the React application, we'll now move onto the next lecture, where we'll look at more details about this curious code HTML-like core that we saw in the app.js file, and then understand why we write the code that way, and what is the HTML-like code doing inside my JavaScript file.

---



## 14.Introduction to JSX

In the previous lecture, we had seen some HTML like syntax in the app.js file. At that point, I deferred further discussion to a later lecture. Now, I will introduce you to JSX, or the JavaScript extensions, which is what you actually saw in the code in app.js file. Let's learn a little more details about JSX. Now, I will give you a brief introduction to JSX in this lecture, but as we go along in the rest of the lessons of this course, we will examine more and more details about JSX, as and when we encounter them, and then learn them in the context in which we encounter more about JSX. As we saw in the previous lecture, we introduced a react element which looks like this. And obviously, one look at it and you are probably wondering, what is this HTML doing inside my JavaScript code here? You have never seen something like this before where HTML code gets engulfed inside my JavaScript code. And so, this is the special syntax that react uses when it expresses the various react elements. Let's examine this and a bit more detail. So, continuing with the discussion about JSX, JSX is syntactic extensions to JavaScript which enables us to express react elements using it HTML like syntax. So, it is a shorthand notation from JavaScript functions that are called, which evaluate the JavaScript object. So, as we will see, the reason for approaching it this way as opposed to having the HTML separate from the JavaScript code as done in other frameworks used to avoid the artificial separation of the rendering logic from the other UI logic. Very often, what is rendered in the screen is dependent upon the rest of the UI logic controlling the rendering on the screen. In other frameworks like angular, we separate out the template code from the JavaScript code, and in some way control the templates from within our JavaScript through various expressions and so on. In react, the two are combined together, so you would see a mixture of both the UI logic and the rendering logic written in single file as you saw in the app.js file earlier. So again, illustrating this with an example, you might see a react element being defined like this in the code. Now, then you'll see this, you obviously notice immediately some HTML like syntax in the code, and some specification of attributes for the HTML like elements and so on. But in reality, this actually gets mapped in react into a corresponding JavaScript object here. So as you can see, this is mapped into a `react.createElement`. This call to the `react.createElement` with the three parameters as shown here, will result in a JavaScript object being created. So, that corresponding JavaScript object looks like this. So, your element becomes a JavaScript object with the type specified there, and the props specified. Many of the standard HTML CACs have corresponding react components that are already defined, predefined react components. And hence, we are able to use them freely within our react code. Not only that, there are facts that you are mixing UI logic with rendering logic means that you should be able

to easily embed JavaScript like expressions into your JSX code. And indeed that is quite possible with JSX. So, for example, you could easily declare a JavaScript object like this here, and then correspondingly, you can incorporate their properties from this JavaScript object into the code that you render here. So as you can see here, for a div, I'm giving a key attribute as dish.id, which has actually derived from the JavaScript object there. And then correspondingly, you can see the use of the media heading. And then in there written braces, you'll see dish.name being declared here, and so on. And so, this code shows you how JavaScript expressions can be embedded into JSX code, as and when required. Now, we will encounter more of such examples as we double up our react application later, and then at that point, you will see how versatile JSX is for specifying various react element.

---

## Setting up your Development Environment

---

### Software Requirements

1. Text editor of your choice: Any text editor that you are already familiar with can be used for editing the project files. I will be using Visual Studio Code (<https://code.visualstudio.com/>) as the editor of choice in this specialization. You may also consider other editors such as Brackets (<http://brackets.io/>), Sublime Text (<http://www.sublimetext.com/>), or Atom (<https://atom.io/>).
2. Browser of your choice: You may use your preferred browser. I will be using Chrome as the browser in all the exercises. All the exercises and assignments in this course have been tested using Chrome v. 46. Please note that not all browsers may support all the HTML5 features to the same extent. You might encounter problems when using other browsers. I strongly urge you to use the latest Chrome browser for the exercises and assignments in this course so that any problems are minimized.
3. Command line shell: Familiarity with the command-line shell will be essential for the exercises. In Windows a cmd window or power shell with admin privileges would be needed. On a Mac or in Linux, a terminal window can be used. Please get familiar with the "sudo" command in OS X and Linux.
4. Files required for the exercises: We will provide additional starter files for the exercises wherever needed. Links to download the files will be provided inline in the exercise instructions that follow each exercise video. Please download the files provided there, if any, before beginning the

exercise. The links are also available through the Additional Resources of the specific lesson.

## 15. Exercise (Instructions): Configuring your React Application

In the previous exercise, we have scaffolded out a starter React Application using create a React app. Now we want to develop that React application into our own React application throughout the exercises in this course. To get started, we will first configure our React application to make use of Bootstrap. Since we have already learned Bootstrap in the previous course, we might as well use Bootstrap for styling our React components or at least the Bootstrap CSS classes. Furthermore, the JavaScript part of Bootstrap cannot be directly used together with React. Instead, what we would do in this exercise is configure our React application to use a Bootstrap-based package called as Reactstrap. What Reactstrap supplies is Bootstrap components re-implement using React components. And so we can make use of them. Like for example, on the JavaScript-based components and Bootstrap have been re-implemented using React components in Reactstrap. And that is what we will use together with Bootstrap in order to style our various components within our React application. So in this exercise, that is what we are going to be configuring our React application with. To configure our React application to make use of Bootstrap classes, at the prompt type `yarn add Bootstrap 4.0.0`. Here, I'm using yarn to fetch in Bootstrap. If you choose to use NPM, you can just say NPM, install Bootstrap at 4.0.0 and space minus minus save, to install Bootstrap into your React project. Now once that is installed, then I'm going to pull in Reactstrap. So, I prompt type `yarn add Reactstrap at 5.0.0` and then also `react popper at 0.9.2`. Now these two together will enable us to make use of React components based re-implementation of the various Bootstrap Javascript components. So let me go ahead and install these two into my application. Again, if you use NPM, do NPM install and then these two and minus minus save, instead. Once these are installing into our React project, then the next step is to go ahead and configure our React application to make use of Bootstrap. To configure our React application to make use of Bootstrap, let's go and first open up the `index.js` file. And in the `index.js` file, we will import the Bootstrap CSS code here. So, right before the `index.css`, I'll import `Bootstrap/dist/css/bootstrap.min.css`. So we are importing this from the node modules folder where Bootstrap has been downloaded, and so we are importing Bootstrap into our application to make use of that within our application. Just like, we imported `index.css`. Now, note that I am importing Bootstrap first and then the `index.css` later. So if I add my own customization CSS classes, that can override the default Bootstrap classes if

required. So that's the reason why I import Bootstrap CSS first and then index.css later. Now once I do this, then I'm going to go into the app.js file and then reconfigure my React application to make use of the Bootstrap component. Now within my React application, I don't want to make use of these, so I'm just going to get rid of these and then introduce my own Reactstrap based components here. So let me first import Navbar from Reactstrap. And so this is our first Bootstrap component that we import from Reactstrap. React component based re-implementation of the Bootstrap Navbar component that Reactstrap makes available to us. Now this Navbar I'm going to configure within my React applications. So here I'll say Navbar. And if you want to use the Navbar with the dark theme, I just say dark and color primary. So I will use the dark color, with the primary color as the darker color. And then close the Navbar. So this immediately puts in the Navbar Reactstrap component into my React application. Inside there if you'll recall from your knowledge of Bootstrap, I can say, `div ClassName "Container"`. So you know what a div, what a container class does to the Bootstrap. But notice that whenever I use my Bootstrap classes, I should say `ClassName`, with a capital N here. So that is how we apply CSS classes to the `jsx` tags here. So here `div ClassNames`. So I introduce a container here and then in there I will say `NavbarBrand` and `href`. And then, this should be quite familiar to you from your previous Javascript or rather the Bootstrap course why we have this `NavbarBrand` here. So notice that Reactstrap's re-implementation of the Navbar and the `NavbarBrand` here. So let me import `NavbarBrand` also from `reactstrap`. So these two components, I am importing, the actual components I'm putting from `reactstrap` and then making use of them to implement the Navbar into my `app.js` file. So with this, let's save the changes and then go and take a look at our updated react application in the browser. Now when you save the changes, then since our `yarn start` command that we executed in the previous exercise, will keep our server up and running and setting up our React application, so it would automatically recompile whenever I make any changes to my React application and then the update of the React application will be served up by my server there. So I'll go to the browser. In the browser, you now see the updated version of my React application. And it contains just the Navbar here with the `NavbarBrand` being declared there. Exactly what we wanted in our application. So with this my React application is now updated to use the Navbar and in the next set of exercises we'll further continue to develop our React application. With this, we complete this exercise. In this exercise, we have configured our React application to make use of both Bootstrap CSS classes and also Reactstrap, which is the React, the components based re-implementation of Bootstrap's JavaScript components. And so we are making use of that within our application to construct the various views that we will

render in our browser. This is also a good time for you to do a Git commit with the message configuring React.

---

## Exercise (Instructions): Configuring your React Application

### Objectives and Outcomes

In this exercise we will set up our project to use Reactstrap (a package supporting easy to use React based Bootstrap 4 components). We will then introduce our first reactstrap component into our application. At the end of this exercise you will be able to:

- Configure your React project to use reactstrap.
- Start using reactstrap components in your application.

### Configure your React Project to use Reactstrap

- To configure your project to use reactstrap, type the following at the prompt to install reactstrap, and Bootstrap 4:

```
yarn add bootstrap@4.0.0
yarn add reactstrap@5.0.0
yarn add react-popover@0.9.2
```

**Note:** You can also install the same using npm using the "npm install <package> -save" option if you are using npm instead of yarn.

### Configure to use Bootstrap 4

- Next, open index.js file in the src folder and add the following line into the imports:

```
. . .
import 'bootstrap/dist/css/bootstrap.min.css';
. . .
```

### Adding a Navigation Bar:

- Open App.js in the src folder and update it as follows:

```
. . .
import { Navbar, NavbarBrand } from 'reactstrap';
class App extends Component {
```

```

render() {
  return (
    <div className="App">
      <Navbar dark color="primary">
        <div className="container">
          <NavbarBrand href="/">Ristorante Con Fusion</NavbarBrand>
        </div>
      </Navbar>
    </div>
  );
}
}

```

...

- Do a Git commit with the message "Configuring React"

## Conclusions

In this exercise we learnt to configure our React application to use Reactstrap.

---

# Introduction to React: Additional Resources

## PDFs of Presentations

1-JavaScript-Frameworks.pdf

2-Intro-React.pdf

3-React-App-Overview.pdf

4-Intro-JSX.pdf

## React Resources

- [Reactjs.org](https://reactjs.org)
- [create-react-app](https://create-react-app.dev)
- [reactstrap](https://reactstrap.github.io)

- [reactstrap Navbar](#)
- [Introducing JSX](#)
- [Convert JSX using Online Babel Compiler](#)

## Definitions

- [Framework](#)
- [Hollywood Principle](#)
- [Inversion of Control](#)
- [Imperative vs Declarative Programming](#)
- [Imperative vs Declarative](#)

## Blog Articles

- [5 Best JavaScript Frameworks in 2017](#)
- [Top JavaScript Frameworks & Topics to Learn in 2017](#)
- [Declarative vs. Imperative Programming for the Web](#)
- [Is React library or a framework?](#)
- [Is React a library or a framework and why?](#)
- [An Introduction to the React Framework](#)
- [React is a framework](#)
- [Why isn't React called framework? What does it lack to be a framework?](#)

## 16. React Components: Objectives and Outcomes

In this lesson you will learn about React components and how we construct an React component and design its views. At the end of this lesson you will be able to:

- Create a React component
- Construct the React component code and the view for your component using JSX and JavaScript

## 17. Exercise (Video): React Components Part 1

[MUSIC] We have already informally encountered React components in the previous lesson. When we scaffolded out our React application using the Create React App, we created our very first component, which was an app.js. Now, in this lesson, we'll explore components in a bit more detail, and understand how components can be created, how we can add our own components to a React application, and how we make use of components within the React application. In the first lesson of the next module, we'll explore various kinds of components that can be created and used in a React application. We can look at a component

as a unit that returns a group of React elements that together render it part of the screen. So the components acts as a unit for gathering together a bunch of React elements with a common purpose. Now the use of react components enables us to split the user interface into multiple independent, reusable pieces. So when you define a component, which renders a particular kind of user interface part, then that component can easily be reused in your UI wherever you require similar kind of behavior or similar kind of rendering of the part of the user interface. So components enable us to break down the entire UI in to smaller reusable pieces. Furthermore, you can easily control what a component renders by supplying inputs to the component. We'll explore this issue a little more in the next lecture after the exercise. And also, components in React can be of different kinds. A few things that you need to note when you create components in React is that when you define your own components and add them to React, you should always start the name of the component with a capital letter. This way the React compiler recognizes that that should be mapped into a corresponding `React.createElement`. So as you re-explore GSX in the previous lesson, we saw how the component defined in HTML syntax is mapped into a `React.createElement` function call. And tags that usually start with a small letter, or lower case letter, is typically integrated as a DOM tag. And then this triggers React to use some of the built in components corresponding to these HTML tags within React. With this quick overview of components, we will move onto the next exercise where we will create our own component and add it to the React application. We will also see how we can use some of the React strap components to design the UI for our specific React component that we are adding in. And we will also explore the concept of state in the exercise. I will come back to explain about state and props in the lecture that follows the exercise.

## 18. React Components State & Props Part 1

In this exercise, we will create a new React Component and package to our React Application. We'll also see how we can make use of it and React struck components to design the view of our React Component and then, we'll see how we can make use of the react component in our React Application. Will also see the concept of state in a react component, when we design this new component and add it to our application. To get started with this exercise, first, go to the exercise instructions and then right up top, you'll see this image.zip file. Download this image.zip file to your computer and unzip it. And then you will get a folder named images. Then, within your project, go to the public folder and then create a new sub-folder under that named assets. And move the images folder into this assets folder. So, let me go ahead and do that and show you the resulting project state. Once you move the images folder into the assets folder,



you would see that in the images folder, you will see a bunch of PNG files that have supplied to you, which will make use of, in creating the view for our new React Component that we are going to add to this application. Next, go to the source for a time in the source folder. Let me create a new folder there named components. The reason for this is that, I was store all my React Components in this component's folder. There's just the convention that I make use of, when I write my React Applications and is just a convenient way of organizing your components into one single location. So, within this components folder, let's create a new file and name this as, my new component.js and in this my new component.js, I'm going to add in my new component. To add in the new component in here, let me first import React and then also import Component from React. Because that is what will allow me to create a React Component. And once we do this, let's create our new component here as a class component and so I will type class Menu extends Component. As you can see, this is how you would add in a new component to React Application and the name of this new component is Menu. And within this menu component, I'm going to first define the constructor for this component. So, we'll define the constructor and this constructor gets a parameter called props. We look at props in more detail in the next exercise, but with this constructor props, I need to supply this props to my super class. So, I say super props here, and this is required whenever you define a class component. Furthermore, any component in React, a class component should implement this method called render() which will turn the corresponding view for this component. So, inside this render method, I need to return the view for this component. So, I'm going to create this return value here and you can see the red squiggly line, telling me that this is empty at the moment so I need to fill that in appropriately. Also when you create the component, don't forget to export that component from this file, because we would need to import this component wherever we want to make use of it within our application. So, there you go. We have completed the basic structure for a component. So, a class component would be defined like this. So, you first extend the component, you define the constructor for it, and then also implement a function called render(), inside this component which will return what needs to be displayed on the UI by this component. And don't forget to export the component from this file. That's it. So, any time you need to create a new component, this is how you would go about creating that new component. Now that we have created the new component, let's decide, what we want to return from this component. So, inside this component, I'm going to first define div with the class name as container and from your knowledge of bootstrap, you understand why we define the container and inside this container, I will define a row div here. Again, from your knowledge of bootstrap, you understand what a row is meant for and the moment I defined this, you can see the red squiggly line has disappeared

because I'm returning something from this component. Now, obviously there is nothing inside this div here. So, if you render this component it will just render a blank screen. Now inside here, I'm going to construct what my component is going to return. So, in this component, since it is called, menu. So, this menu component is going to construct a list of dishes for my restaurant. So, as you saw in the previous exercise, we created this restaurant confusion. So, this is going to be the web application for my restaurant that is going to display a list of dishes in my restaurant. So, that's what I'm going to construct inside this menu component. Now to help me to construct this menu component, I'm going to import a media component from reactstrap and I'm going to make use of that in order to construct my menu for my restaurant. So, inside here, I will say import media and then in here, I'm going to say media and let's say list. So, this is going to display a list of items for my restaurant. And so, if you say media list here, and in here, I'm going to include a JavaScript variable called menu which I'm going to define in a short while. Now, you're saying, I can make use of JavaScript variables inside, my JSX code. Indeed that's what we learnt in the previous lesson. So, what this Menu? I'm going to define this menu as a JavaScript constant here. So, I'll say, const menu and they will leave it at this point and will come back and define this menu, a little bit later. But first, let's bring in some data into my component so that, I can make use of that in order to construct my list of dishes for my restaurant. Now, to do that, we'll go into that constructor here, and then I'm going to define a state for my component here. What does the state. The state stores in. Properties related to this component that we can make use of. So, inside this step here, I will define a property call this dishes. And inside those dishes, I'm going to define this as a JavaScript object which contains a list of dishes. Now, in order to save myself the trouble of typing in all these, what I would suggest for you to do is to go to the code snippet that I have given in the exercise instructions and simply copy the JavaScript objects that contain the details of all the dishes and then paste it in here. So, here you can see that I have copied in all the dishes information from my instructions, exercise instructions and then pasted it in here. Now, typing this one by one is going to take a lot of time so it is more easier to simply copy and paste this code in here. So, now I have this dishes property here which consists of four JavaScript objects and each JavaScript object has an ID, name, an image reference, a category, a label, and a price and a description here. Now we are going to make use of these in order to construct our list of items in my menu here. Also note that the image field here refers to this assets/images so that is why we moved on the PNG files into that folder in our public folder. So, after pasting this, let's go ahead and construct the items in the menu here. So, now all that we are left is to make use of this list of dishes that are given here and then construct the menu here. Now, this is where I'm going to make use of the array that is defined here.

So, these dishes that we have defined here is an array of JavaScript objects. So, I'm going to take that JavaScript object. So, when you're define a property like this in the state, you can refer to this within your code here as saying "this.state.dishes." So, you'll see that the dishes has been, property has been defined on the state and then I'm going to alternate over all the items in my dishes array. So, that is where I use the JavaScript map operator. And then I will map that into a list of items here. So, I will say map and then dish. So, I'm sure from your knowledge of JavaScript, you understand how the map operator works. So, in here, I am using an arrow function to define what is going to be returned by this map operator. So, this means that I will be iterating over every dish in the dishes array here. For every dish, I'm going to return. So, you see how I'd define this here. I'll say "Return". For every dish, I'm going to return a lay out for the dish here. So, I'll say `<div key={ dish.id}>`. I'm going to explain that in a minute and we'll say `className="col-12 mt-5"`. Again from your knowledge of bootstrap, you understand what col-12 is and mt-5 means give a top margin of five units to this and then inside this div. So, inside this div, I'm going to construct the view for each of the items in my dishes here. So, now, you're wondering why we need this key here. Now, whenever you construct a list of items in React, every item requires a key attribute to be specified for it. So, this is where we are taking a list of items and then rendering them here. And then so the key helps when React is rendering these items on the screen. The key helps React to recognize each one of these elements and while it is updating the screen so the keys will enable it to identify each of those elements uniquely. And so when it sees that it needs to update the user interface, in case for example the dishes we added more items into the dishes and the UI needs to be updated, then it uses this key property to uniquely identify each item that has been rendered in here. So, that's the reason why whenever you render a list of items, you always give a key property to each of those items. And in here, since each dish as you can see has a unique ID already assigned to it, I'm just going to use the dishes' ID as the key for each of these items there. So, that's how you render the list of items in the menu. Now, how do we render each item in the menu? So, this is where I'm going to use the media class that I have defined here. So, again if you read the react bootstrap documentation on how to use the media class, so this is how they make use of the media class. So, we'll say `<media tag="li">`. So, the "tag li" says that each of these is going to act as a list item here. And notice that here I have already specified this as a media list so that displays a list of items here. So, when I say "tag li" and in here I can go ahead and lay out my items. So, I'm going to make use of `<media left middle>`. So, if you remember how you will use the media objects in bootstrap in the previous course, you understand what we are doing here. So, this is react bootstrap way of specifying here. So, this particular media, element here is going to help me to display the image here. So, I'll say

`<media object src={dish.image}`. Now, if you recall how we used the media object in bootstrap, you would begin to see why we define it this way. And then we'll close this off because this is a self closing tag here. So, I'm just going to do a self closing of the tag because it is just display just this particular image here. And then below that, I'm going to display the media body with the class name, `ml-5` meaning give a left margin of five units for this. And in here, we'll go in and display the media as Media Heading and we'll see `dish.name`. Now, when you see how this is laid out, it will become more clear to you why I'm defining this object like that, `media dish image`, and then also down below, we'll say `p dish.description`. There you go. So now, you see how I have constructed this div and returning this div from inside of this menu constant. So here, you can see that in here, I am laying out each and every dish by using this media object here. And so in here, I'm showing the dish image, I'm showing the name of the dish as a heading, and the dish description down below here. Now it will become very clear to you how this is actually shown when we look at what is rendered on the UI in a minute. So this essentially ends up constructing a list of items and then returns that, and then this particular JavaScript variable I am using `m` here to define my media list. So whatever is returned by this is going to be used in the media list to render a list of dishes from my menu component. So, that completes the structuring of my menu component. Now, how do we make use of this menu component? To make use of this menu component, we'll go into `app.js`. And in `app.js`, I'm going to import menu from menu component. That's it. The app component that I've just created now becomes available within my app component. Now if I want to make use of the app component inside my menu component, all I need to do is go down here and then simply say menu with a self closing tack and that's it. Now, my menu component will be rendered below the nav bar in my app component. So now, you can see how I have subdivided my UI into two parts. There is this one menu component. And then on top, you have the Navbar component, which is enclosed inside the app component, which is then rendered by the `index.js` file here as a single component. So you can see how one component can be imported into another component and then made use of in the other component in order to construct the overall user interface. You will see me repeatedly doing this when I construct the user interface for my application. So with this, let's save all the changes and go and take a look at the resulting page in our browser. Now, going to web browser, you can now see how my menu component has allowed me to construct a list of items, list of dishes and then display them as shown here. Again, I notice that this thing is centered. The reason it is center is because we are applying some CSS classes to the `app.js` file. So I'm going to go ahead and remove those and then we'll see that the menu will be rendered more appropriately. So again, going back to the `app.css` file. In the `app.css` file, let me go to the `app.css` file. I am not going to be using any of

these CSS classes in my React applications. I'm just going to delete all of these and then remove them from my app.css file which also means that I go to app.js and then this class doesn't exist, so I'm going to remove that also from app.js and then save the changes. Returning to the browser, you can now see how my list of items in my menu is now constructed and then rendered on the user interface. So this part, this list of dishes that are displayed here is constructed using the menu component. And this, of course, is the Navbar component which we have used in the app component. Summed up together, this is now constructing my user interface, and you can see how the media object allows me to display an item in the list of items with an image on the left side and then the details in the description of the right section. So this is how the media object and bootstrap works. I basically use the React straps implementation of the media object from bootstrap in order to render each of these items in my list of items that is displayed here. That's it. So, we have now seen how we can easily construct a menu component and then make use of that within our React application. With this, we complete this exercise. This is a good time for you to do it a Git commit with the message components part one, because there is a part two to this exercise which we will do after the next lecture. In this exercise, we have seen how easy it is for us to add our own component into our React application and then construct the UI off that component using several other React elements and then several other components that we imported from React strap and then laid out the content on the screen. We have also seen how we can make use of the state of a component to store some data and then make use of that when we construct the UI for the component as we have seen in this exercise.

## Exercise (Instructions): React Components Part 1

### Exercise Resources

[images](#)

[ZIP File](#)

### Objectives and Outcomes

In this exercise you will add the first component to your React application and update its view using JSX. At the end of this exercise you will be able to:

- Add components to your React application
- Use JSX to define the views of your component.

### Adding a Menu Component

- First, download the *images.zip* file provided above and then unzip the file. Create a folder named *assets* in the *public* folder. Move the resulting *images* folder containing some PNG files to the React project's *public/assets* folder. These image files will be useful for our exercises.
- Next, add a new folder named *components* in the *src* folder, and create a new file named *MenuComponent.js* in this folder.
- Add the following code to *MenuComponent.js*:

```
dishes: [
  {
    id: 0,
    name: 'Uthappizza',
    image: 'assets/images/uthappizza.png',
    category: 'mains',
    label: 'Hot',
    price: '4.99',
    description: 'A unique combination of Indian Uthappam (pancake) and Italian pizza, topped with Cerignola olives, ripe vine cherry tomatoes, Vidalia onion, Guntur chillies and Buffalo Paneer.'
  },
  {
    id: 1,
    name: 'Zucchipakoda',
    image: 'assets/images/zucchipakoda.png',
    category: 'appetizer',
    label: '',
    price: '1.99',
    description: 'Deep fried Zucchini coated with mildly spiced Chick pea flour batter accompanied with a sweet-tangy tamarind sauce'
  },
  {
    id: 2,
    name: 'Vadonut',
    image: 'assets/images/vadonut.png',
    category: 'appetizer',
    label: 'New',
    price: '1.99',
    description: 'A quintessential ConFusion experience, is it a vada or is it a donut?'
  },
  {
```

```

        id: 3,
        name: 'ElaiCheese Cake',
        image: 'assets/images/elaicheesecake.png',
        category: 'dessert',
        label: '',
        price: '2.99',
        description: 'A delectable, semi-
sweet New York Style Cheese Cake, with Graham cracker crust and spiced with Indian
cardamoms'
      },
    ];
  }
}

```

```

render() {
  const menu = this.state.dishes.map((dish) => {
    return (
      <div key={dish.id} className="col-12 mt-5">

```

- Next, open App.js file and update it as follows:

...

```
import Menu from './components/MenuComponent';
```

...

```
<Menu />
```

...

- Open *App.css* file and delete all its contents.
- Save all changes and do a Git commit with the message "Components Part 1".

## Conclusions

In this exercise we added a new component to our React application, added data to its class, and then updated the app to show the information in the web page.

## React Components: State and Props



We have already seen the use of state in the previous exercise. Babies told the information about the dishes in that menu components state. And we use this information to render the menu items. Another way of passing information to a component is through Props. Let's discuss a little more detail next. So, as we have seen in the previous exercise, each component can store its own local information in its state quote and quote. So this information is private and fully controlled within that component. Now is state within a component can easily be passed to its children through the use of props which we will talk about a little bit later. In react, only class components can store state. So, if in a particular component you need to store the state, you need to implement that component as a class component. We will see other kinds of react components there, we don't need to store state and we can simplify the component declaration significantly. As we have seen in their previous exercise, the state information is usually declared in the constructor of the class component. So, when you declare the constructor, you would initialize a variable called this dot state which is a JavaScript object which contains all the state for this component. If you need to update the state of a component, you cannot directly go and update the state by changing the property values. Instead, any update to a state of the component has to be done through the use of that Sect State method. So, whenever you need to make any changes to the state of the component, you always say this.setState and then supply the property that you want to modify. So, when you call the this.setState with a particular property that is change, then this change will be merged into the state of the component. And again, let me remind you that you should never change the state by doing something like this. You can't say this.state.select dish is equal to dish. So, never directly manipulate the state property values. When you make use of the time component in your own component, then whatever attributes that you specify to the JSX element will be passed in as props to the child component. So these attributes will be available as props within the child component and you can easily pass in multiple attributes to the child component. Let's take a look at an example. So in these examples, you can see in the first case, we are using this menu component within your current component and then you're passing in dishes as one of the attributes. Now when you pass this in, then within the menu component, this dishes information that your passing will be available as this.props.dishes and can be used within the menu component for it to render its view. Similarly, you can pass in multiple attributes. If you choose to for a child component and the child component will get access to each of these multiple attributes, as this. props.theattributeimage. So, as you see this.props.dish or this.props. comments and so on. So, this is an easy way of a parent component passing information to its child component. So in this case, a parent component maybe storing the state and then the state information can be passed to the child component through the



use of the props in the form of attributes that you specify to the JSX statements. Just like the way you handle events in the DOM, you can also handle events similarly in react. But when you specify the events, then you use camel case to specify the events. And when you specify the event you'll pass a function as the event handler in this case. As an example, you can see in this case we are specifying the on click event handler there now, written on click even handle and passing in an arrow function as a parameter there. Now, within this arrow function as you can see, if you want to pass some information or parameter back to the the event handler, that can be easily done as shown in this example. If you don't have any parameters or data to be passed back to the event handler, then you can simply say this.onDishSelect instead of defining it as the arrow function. We will see several users of such event handlers as they go along in this course. And at that time, I will again remind you about this particular aspect of how you handle events in react. Very often, your react application will be implemented as a hierarchy of components. So in that case, you may wish to lift the state up to an ancestor of the current component. So for example, if several components have a shared state and any change to the state needs to be automatically seen by all these components, then it is better to move the state to one of the ancestors, common ancestor of all these components, and ensure that any state changes will be sent back up to that common ancestor. We will see the use of this in the exercise in one of the later modules. But again, in the very next exercise, you will see me lifting the state up from it child component to it's parent component, to make it convenient for us to store the state. Any changes to data in one component may need to be reflected to another component, and this is where by lifting the state up it makes it more convenient for you to handle this. When you have an array of items, you could easily lay them out as list in your react components. Lists and the way you handle them is very similar to JavaScript. So for example, if you have an array, and you want to iterate over all the items in the array and then converted into a list, you can easily do that using the map JavaScript function data. So in this example, you can see that I am taking the dishes which come in as an array of JavaScript objects and then I do the map on that which gives me access to each element of this array. And then for each element of the array I can then laid out as a list of item. When you do list items in react. For each of the items it is important to specify the keys for each element inside an array. So as you can see in this example, I am using an outer div array a very specific keys is equal to dish ID. So in this case since each dish will have its own unique ID, I'm using the dish ID as the key when I lay out the list of items data. The reason to specify a key in react, is that when there are changes to the number of items in the list maybe a new item is added into the list or an item is removed from the list, any such changes to the data may cause re-rendering of the list items. So, when react is doing that, it will use these keys to identify which

items have been removed and so appropriately partial re-rendering of this information is facilitated in react. As you recall, react only modifies those parts of the gum tree that need to be updated. With this quick understanding about the state and props and how you lift the state up and the application of list of keys, let's move on to the next exercise where we'll use all these concepts in reorganizing our react application, and then also use a better way of rendering the list of items within our menu in our react application.

## 19. Exercise (Video): React Components Part 2

Continuing with our explanation of React Components in this exercise, we look at how we can pass information to a child component from a parent component using props, and we'll look at how we can lift the state up to a higher-level component.

As you saw in the previous exercise, the menu component was being used by the App component in order to render the list of dishes. The App component becomes the parent for the menu component. So, if there are other components that will share the same state information as the menu component, instead of storing the state information in the menu component, we can lift the state up to the App component and make it available to all other components, that are children of the App component through props. That is what we're going to attempt in this exercise.

As a first step in this exercise, I'm going to remove these dishes from this state here, then move it into a separate file, so that my code looks more cleaner and better organized. Furthermore, I'm going to lift the state information up to the App component. So, to do that, going to the source folder, I'm going to create a new subfolder in there named, shared subfolder. Now, I'm going to use the shared subfolder to store all the information that is shared among various components in my React Application that I'm designing. So, in here, I'm going to create a new file named dishes.js file. Now, into this dishes.js file, I'm going to paste in all of the information about the dishes, in the form of an array, and then export it from this dishes.js file. So, if you do that, you can either go to the exercise instructions, and then copy the list of dishes information that I've given there, and then paste it in here. Or alternatively, I have already given you a pretty far better dishes.js file, you can simply download the dishes.js file and then move it into this shared folder. So, here you can see that I have pasted all the dish information into this dishes.js file in the shared folder, and then I'm exporting this constant named the dishes from here, which is an array that contains all the details of all the dishes including additional information. So, as you can see, if I simply paste this information into menu component or js, then, your menu component file will expand so huge. Instead, it is better to store it in a separate

file like this and then export it as a constant from here. We are leveraging JavaScript for making our code organization better. So, now, I paste this information in here into the dishes.js file. Now, what I'm going to do is to go into the App.js file and then lift the state into the App.js file. You will see that in the assignment that follows, you'll realize why it is better to store the state information in the App.js file. So, coming into the App.js file, let me first save the changes to the dishes.js file, and then coming into the App.js file, I'm going to import dishes from shared dishes file here. Then, going into the App.js file, note that we need to define the state in here. So, to do that, let me add in the constructor. So, as you'll recall, in order to store the state, you need to define the state in the constructor of the class component, and then build here `super(props)`. This is required, and then we'll define the state here, as this state equals to and a JavaScript object which is dishes, which is the dishes that I have just imported into my App.js file. So, now my state information that contains all the dishes is now lifted into the App.js file. I can make this available to the menu component through props from the App.js file. So, once I have lifted this into the App.js file, let me go down here, and then make those dishes available as props to the menu component. So, I'll say, `dishes equal to this.state.dishes`. So, this way the dishes that I have defined in the state for my App component is now made available as props to my menu component. So, I can go into my menu component, and this dishes information becomes available into my menu component. Now, let me save the changes to the App.js file. Now, going into the menu component, I no longer need the dishes in the state of the menu component, because I am receiving that as props from my parent, which is the App component. So, I'm going to clear out the state there, so the code is a lot more cleaner now. Now, in order to make this available for us to use within my menu component, notice that the props are already available for me here. So, in here, when I construct the menu item here, so, I'm constructing the menu item here. Now, when I construct the menu here, instead of saying `this.state.dishes`, I need to change that to `this.props.dishes`, and then that's it. My menu component will work just as before. So, let me save the changes to the menu component, and let's go and take a look at our application in the browser. Going to the application in the browser, I can see that my application looks exactly the same as before, no changes. I have simply replaced the use of the state in the menu component by use of the props in the menu component, and everything works just as before. Now, I'm going to go into the menu component and then render this dish information in a different way using cards. So, if you have done the bootstrap course, you know what cards look like. So, we're going to use a card component from reactstrap, and then layout the menu items in a different way, to make it look a bit more elegant. Going back to my menu component, instead of the media object, I'm going to use a card from reactstraps. So, I'll import card,

and then `cardImg`, because I'm going to display an image there, and then I'll also import `cardImg` overlay. You will see how I make use of that in a short while. Then `cardText`, and `CardBody`, and `cardTitle`. All these components I'm importing from `reactstrap`, and then I'm going to make use of them to render my dish information in a different way. So, now that I have imported all these, let's go ahead and then fix up the way we layout each dish in here. So, instead of using the `media`, I'm going to change that to a card. So, I'm going to make use of the card component to do the layout of this. Then for the `div` that encloses this here, I'm going to say `div col-12 col-md-5`, change this to `m-1`. Meaning, one unit margin all around in here. So by doing this, what I am doing to this `div` is, for the extra small to small screen sizes, I'm going to layout one card below the other in that row. For medium to extra large screens, I'm going to layout cards side by side, each card occupying five columns in my row, in my bootstrap row there. So, that's the change that I have made here. So for the card, now, we don't have the `media` anymore, so instead of the `media` object, here, I'm going to use the `cardImg` to render the image there, `card image`, and then for this card image, I will give the width as 100 percent, and the source remains as `dish.image` and the alternator is also a `dish.name`, so that part remains exactly the same as before. Then this one, I'm going to change this to `cardImg`. Overlay here. And this part also to `card image` overlay. So, the `media` that object that I was using I have changed that to `card image` overlay here. And then this `media` heading, I'm going to change that to `CardTitle` and I'm not going to show the description in the menu here. So, I'm going to remove the description from there. After these changes, let me now go down to the bottom here. And then in here, I'm going to remove this `media` list here because I am no longer using the `media` object here. So, I'm going to remove that and I'm simply going to render the menu into this row here. And the menu items are already being formatted into different columns of the row by using this `div` here. So, let's save the changes and go and take a look at the resulting application in the browser. Going to the browser, you see now how my menu items are laid out on the screen there. So as you can see, instead of displaying a list of items, I'm now showing images of each of the items and then the name of each of these items like this. Of course, the details of each of these menu items is now removed from this list, but this is how this is displayed in a medium to extra-large screen size. If I want to see the same view in extra small to small screen size, we can go to the view and developer tools and then I'm going to click on this and then see the view. As you can see, you can see the responsive view here. So, on a pixel two, what this view looks like on a pixel two device. So, you can see the nav bar there and then the items being layered one below the other because we have used `col12` there. So, this is layered one below the other in the view here. That is the layout that we have just created by using the card for my menu items. So, not only that, now the next thing that I'm going to

do is if I click on any one of these, I want to show the details of that particular dish down below here. And I'm going to use again the card in order to display the details of the dish. So now, I need a way of responding to clicks on this. So, this is where we're going to use the on click to respond to clicks of these items in the menu. So, going back to my menu component, in this state of my menu component, I'm going to introduce a new property called the selected.Dish which I will assign as null initially. So, the selected.Dish is null initially so which means that I haven't selected any dish. Now, whenever I click on any one of these dishes, then I will make the dish information become equal to the selected.Dish. To do that, let me go into the card here and then for this card, I'm going to introduce the on click method here. So, when this is clicked. So, this is how you respond to the handling of the events that are calloused on your view there. So, inside this onClick. So whenever this onClick is clicked, I'm going to call this function which I'm going to implement in a short while called this.onDishSelect, and then I will pass the dish information as a parameter to that. So, now you see how we can respond to the event that is calloused in my view there. So when I click on the card, that card information is passed into this method called onDishSelect. Now, what do I do in this onDishSelect method? So, I need to implement this onDishSelect here. So, let's go in here and say onDishSelect, and this receives the dish as the parameter. So, when it receives the dish as the parameter, now I want to update the selected dish to point to this dish which is received as the parameter here. So to do that, remember that when we need to change the state, we need to use the this.setState function call. You can't directly say selected dish is equal to dish. Instead, we have to say this.setState and then inside here we'll say, selected.Dish and this will result in whenever this function is called, then this set state will ensure that this selected dish will be set equal to the dish which is received as the parameter here. So, at that point, the selected dish will be set to the dish on which we just clicked on the user interface. So, that's how you change the state of your component here. So, we are setting the selected dish. Now, when the dish is selected, I want to also render the details of the dish. So, I'm going to add in one more method here called renderDish which will be supplied the dish as the parameter here. And then so inside this renderDish method, I will first check if dish not equal to null. So, obviously if the dish is not null, only then I will render the dish. Otherwise, I will simply return a div, an empty div here. So, if you return an empty div, nothing will be rendered on the screen. So, when the selected dish is null, I will return this empty div and nothing will be rendered on the screen. If the selected dish is not null, then at that point we will render the dish here. So, in here we'll say, return. Now, here I'm going to use the card that we have just designed there, the card component. I'll use the card component in a different way to render my dish here. So, inside this card component, we'll render the dish in this way. Let me

copy the card image. We are going to use the same approach for the card image, so I'm going to display the card image at the top, and then down below the card image, I'm going to display the CardBody. And in the CardBody, I'm going to display the CardTitle so let me copy this CardTitle from here. So, in this CardBody, I'm going to display the CardTitle and also, I will display the CardText as dish description. So, CardText. So, this is how I construct the card for the selected dish when I render it on the screen there using the card. So, now you see how the rendered dish is returning the card. So, how do I make use of the render dish here? Going down into my code here, right below this, I'm going to define another row here, and inside this row div, I'm going to display the card. So, to do that, I'll say this.renderDish, and supply the this.state.selectedDish as the parameter to that renderDish function. So, when this renderDish function is called, that will return the selected dish in the form of a card, and that will be displayed down below here. So, with these changes, let me save all the changes and then let's go and take a look at our application in the browser. Going to my application in the browser. Now, when I click on any dish, so let's say we click on the second dish here. You will see immediately that the second dish is rendered down below here with the details of the dish as shown here. So, as you can see, the details of the dish. So if I click on a different dish, then immediately that dish information is rendered down below here and so on. With this, we complete this exercise. In this exercise, we have seen several things. We have seen how we can lift the state up to a higher level component and then pass the state information down to the child component using props. We also saw how we can respond to events that are calloused on the UI. In this case, the on click event on any card in the UI, and how we render the details of the dish using the card component from reextract, but this may complete this exercise. This is a good time for you to do a get commit with the message components part two.

## **Exercise (Instructions): React Components Part 2**

### **Objectives and Outcomes**

In this exercise we will continue modifying the menu component from the previous exercise. Instead of a list, we will use a Card component from reactstrap to display the menu in a different way. Also we will use the Card component to display the details of a selected dish. At the end of this exercise you will be able to:

- Make use of the Card component to display a list of items.
- Use the Card component to display detailed information.

### **Exercise Resources**

dishes  
JS File

## Updating the Menu Component

- Open *MenuComponent.js* and update its contents as follows. Note that we have removed the `dishes` variable from the state of the component, and updated it to use the `Card`:

```
. . .

import { Card, CardImg, CardImgOverlay, CardText, CardBody,
  CardTitle } from 'reactstrap';

class Menu extends Component {

  constructor(props) {
    super(props);

    this.state = {
      selectedDish: null
    }
  }

  onDishSelect(dish) {
    this.setState({ selectedDish: dish});
  }

  renderDish(dish) {
    if (dish !== null)
      return(
        <Card>
          <CardImg top src={dish.image} alt={dish.name} />
          <CardBody>
            <CardTitle>{dish.name}</CardTitle>
            <CardText>{dish.description}</CardText>
          </CardBody>
        </Card>
      );
    else
      return(
        <div></div>
      );
  }

  render() {
```



```
const menu = this.props.dishes.map((dish) => {
  return (
    <div className="col-12 col-md-5 m-1">
```

- Add a folder named *shared* under the *src* folder.
- In the *shared* folder, create a new file named *dishes.js* and add the following content to it (**Note:** Alternately you can download the *dishes.js* file given above in the Exercise Resources and move it to the shared folder. Make sure the file is named *dishes.js*):

```
export const DISHES =
```

```
[
  {
    id: 0,
    name: 'Uthappizza',
    image: 'assets/images/uthappizza.png',
    category: 'mains',
    label: 'Hot',
    price: '4.99',
    description: 'A unique combination of Indian Uthappam (pancake) and Italian pizza, topped with Cerignola olives, ripe vine cherry tomatoes, Vidalia onion, Guntur chillies and Buffalo Paneer.',
    comments: [
      {
        id: 0,
        rating: 5,
        comment: "Imagine all the eatables, living in conFusion!",
        author: "John Lemon",
        date: "2012-10-16T17:57:28.556094Z"
      },
      {
        id: 1,
        rating: 4,
        comment: "Sends anyone to heaven, I wish I could get my mother-in-law to eat it!",
        author: "Paul McVites",
        date: "2014-09-05T17:57:28.556094Z"
      },
      {
```



```

    id: 2,
    rating: 3,
    comment: "Eat it, just eat it!",
    author: "Michael Jaikishan",
    date: "2015-02-13T17:57:28.556094Z"
  },
  {
    id: 3,
    rating: 4,
    comment: "Ultimate, Reaching for the stars!",
    author: "Ringo Starry",
    date: "2013-12-02T17:57:28.556094Z"
  },
  {

```

- Open *App.js* and update it as follows:

. . .

```
import { DISHES } from './shared/dishes';
```

. . .

```

class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      dishes: DISHES
    };
  }
}

```

. . .

```
<Menu dishes={this.state.dishes} />
```

. . .

- Save the changes and do a Git commit with the message "Components Part 2".

## Conclusions

In this exercise we used a list of Cards to display the information in the menu component. Also we used a card to display the details of a selected dish.

## 20. React Components: Lifecycle Methods Part 1

---

We have just completed learning about a React component. We have seen how we can create a React component by defining a class and sub-classing the React component class. Now, in the next module, we'll learn about other kinds of React components. Like for a class component in a React application, every class component has a life cycle associated with it. What exactly is a life cycle? Let's briefly talk about that. As we realize, a React application is made up of multiple React components that are connected together to form the entire screen of your React applications view. Now, each React component as in when it is required will be created by React and then added into the DOM of your entire application. So, every time a React component needs to be included into your applications view, then the component that hosts the specific part of the view will be created and added into the overall React component hierarchy. So, a component passes through what we call as the life cycle for the component. Now initially, the component doesn't exist, then the component gets created and then once it is created, then it can be mounted into your React application at an appropriate point in the hierarchy of components, then it will exist at that point for a period of time and then subsequently, maybe you no longer require that view, and so at the point, the component can be removed from this hierarchy. So that will be an unmounting of the component. So, as you realize, a component passes through a life cycle of creation, mounting, and then existing in the hierarchy, then unmounting, and then disappearing at the point. Now in every stage of this life cycle React provides several life cycle hooks or life cycle methods that can be invoked to perform certain operations. So again, to summarize what I have just discussed, a React component goes through the following life cycle stages: the mounting of the component after creation and in between if the view of the component needs to be updated, then it gets updated. So at that point, it will be in the updating stage, and then when that component is no longer needed, it'll be unmounted from the hierarchy so that will be the unmounting stage of the life cycle for your component. Now, as I mentioned, there are several life cycle methods that are available in each stage. The mounting stage, the updating stage, and the unmounting stage. So, these life cycle methods will be in invoked

by React as your component is passing through these stages of its life cycle. Let's examine some of these life cycle methods. Later on we will understand how we can make use of these life cycle methods to perform certain operations required when a component is passing through a particular stage of its lifecycle. Let's first talk about the life cycle methods that are available when the component is getting mounted into the overall hierarchy of the React components. Now in the mounting stage, these methods that are available, a constructor, `getDerivedStateFromProps`, `render`, and `componentDidMount` methods are invoked when an instance of a component is created and inserted into the DOM. So, at that time, when it gets mounted into the DOM at that point these methods are going to be called. The constructor will be called first, the `render` will be called then, then the `componentDidMount` will be called, and so the `componentDidMount` will be called after your component gets mounted into the DOM, and then the view is rendered into the overall view of your application. Now, if you see some of the older React documentation or some of the older examples, you might also encounter people using another method called `componentWillMount`. Now, with the React 16.3, this method has been duplicated and you're no longer advised to use this method. From 17.0 onward this method will be completely removed from React. So, we would desist from using this method in our React application. We will only use the constructor then quite the `render` method obviously has to be used every single time. The `componentDidMount` will be used when you need to execute something after the component gets added into the DOM. Then the `getDerivedStateFromProps` will be used if required. Although I don't have an example to illustrate that to you. Now, we'll look at an example of how we can make use of some of these mounting life cycle methods by modifying our application that we have been working on to see that these life cycle methods indeed get called as the component is getting mounted. At a later stage, in another module we'll come back to the methods that are used when a component is being updated and also when a component is being unmounted. To help you understand how the life cycle methods are invoked, let's go to our application that we have been working on so far and in the menu component. Now, you'll see that in the menu component that we have created in our application, the constructor is already included there. So, the constructor will be invoked when the menu component is created. So, to illustrate to you that indeed this is going to be called, I will put a simple console log statement in there saying, "Menu Components constructor is invoked". So that way, you can see in the console log that this indeed was called. Now, similarly, I will take this same console log and then we will use it in the `render` method. So in the `render` method. I'm going to also invoke that console log. So, "Menu components render is invoked", and let me add in a new method here, new life cycle method here called `componentDidMount`, and this method will be

invoked again will test, receive and this method is invoked. So, we'll say `componentDidMount` and then I'll do a console log saying, "Menu `componentDidMount` is invoked". Now, I will add this in here just to illustrate to you that these life cycle methods are indeed being called as the component is getting mounted. So, in this example, we see that when our React application gets created, the `App.GS` will mount the menu component into its view, and so at that point the menu component gets created and added into our overall DOM for our React application. So, let's execute this and then watch the JavaScript console in our browser to see when these methods are getting invoked. Going to our browser, you'll see that our application has been rendered just like before. There is no change to the view of your application, but let me open the JavaScript console. So here, I'm going to the view. Developer JavaScript console. If you're using another browser, then you would similarly be able to access the JavaScript console in the browser using a similar approach. Look for the developer tools, which are part of your specific browser that you are using. So, again going into the JavaScript console, you can see here that first statement that says, menu component constructor is invoked. Then, the next thing that is printed in the console says, menu component render is invoked, and then finally, Menu `componentDidMount` is invoked. So you see how these three methods are invoked in that sequence, as your component get mounted into your DOM by your reactor application. So, this methods will allow you to get access to the component at various stages of its creation and mounting into the DOM, and if you need to perform any operation during that period, you can include the appropriate code into that specific location, either in the constructor, or in the render, or in the `componentDidMount` methods. Now, we have already seen that in the constructor, we are initializing the state for the component, right within the constructor. So, that is one use of the life-cycle method. Similarly, if we want to do something after the component is mounted into the DOM, then we can include that in the `componentDidMount` method. Now we will see the use of that in one of the later exercises in the last module of this course. As we are working on our React application, let me also draw your attention to a developer tool that is available for React, called as `react-devtools` which is available. This is the GitHub repository for it. This is an extension that gets added to your browsers, either Chrome or Firefox, or even available as a standalone app, if you would prefer to use it. Now, in my case since I am using the Chrome browser, I'm going to install the Chrome browser extension, and then make use of it within my application. So this is an interesting tool that will be very useful as you're developing your React application, because you can easily watch things when your React application gets created and rendered, then you can see various aspects of your React application including the DOM hierarchy, which will enable you to check for any errors. So if you encounter any problems in the way the views are rendered,

then you can go and check in the React developer tools. Let me show you react-devtools for the specific application that we have been working on. Now before we do that, if you want to add the developer tools, go to the appropriate browser extension there, and then add it into your browser. So for Chrome, you go to the Chrome dev store and then from there add the extension for Chrome, similarly for Firefox. If you want to use a standalone app, then you can download that and then make use of it. Now, I find that it is a lot more easy for me to integrate the extension into my Chrome, because I'm already using the console log for watching the JavaScript console, and so the react-devtools will add one more tab to that developer tools window that I bring up at the bottom of my browser, and so it is more convenient for me to use the react-devtools that way. Now, a link to the react-devtools is available in the additional resources off this section. Now going to my React application, right there in the developer tools, I have the react-devtools installed there. So, when you open it you can see the structure of your DOM there, so you can see the app here. Now, the advantage of doing this is that you can go in and watch all the aspects and see all the properties that are being sent to each one of these components, and also for example in the menu, you can go into the menu and then see the props that have come to the menu, so you can easily find out whether your props are being correctly passed into the menu item or not, and so on, and then when you click on a particular dish as you realize, you're going to be rendering that particular dish there. So the details of the dish are shown down below here, and you would also see in the React developer tools, you can see the state here, and you can see that the selected dish is initialized to that dish that we just selected there. So, that's the advantage of react-devtools. It will allow you to go in and see the different aspects, the different parts of the application, and be able to check to see if you are encountering any errors. If the props are been correctly passed, if the step is correctly initialized, all these can be easily useful while you're developing your React application. So, take advantage of the react-devtools to help you track as you develop your React application.

## **React Components: Additional Resources**

### **PDFs of Presentations**

5-Component-Part1.pdf

PDF File

6-Component-Part2.pdf

PDF File

## React Resources

- [React Components](#)
- [React Component State](#)
- [React Component Props](#)
- [reactstrap Media Object](#)
- [React.Component and Lifecycle Methods](#)
- [reactstrap Card](#)
- [Bootstrap unstyled list](#)
- [React Dev Tools](#)
- [React Dev Tools Chrome Extension](#)

## Assignment 1 Requirements (Video): React Components

It is time for the very first assignment in this course. In this module, we had explored React; how we can create React components, the state of props of a component, and also list and keys for when we display list of items in our components. So in this exercise, we will use all these concepts. Now, in the previous exercise, we had shown the details of a chosen dish at the bottom in the menu component itself. In this assignment, you're going to separate that into its components called as the dish detailed component. So, the details of the dish will be rendered through the dish detail component, which you will import into that menu component, and passing the information about that selected dish to the dish detail component in the form of props from the menu component. So, that gives you the rough idea of what we would like to display. In addition, we will also show a list of comments about the specific dish in the dish detail component, when we display the details of the chosen dish. Let's look at the three tasks in this assignment in a bit more detail. Coming to our application, in the first task of your assignment, you will be taking the details of the selected dish that is showed at the bottom, and instead move that into its own component called as the dish digital component. Details of what you need to do are given in the assignment right up. So, when you click on any dish, then that details on that dish will be displayed at the bottom here using the dish detail component. Now,

within the dish detail component as you can see, the details of a dish are shown through one single card here. This card occupies five columns, when you are on a medium to extra large screen size. So you'll notice that, this is aligning very well with each of the items in my menu here. Now, in addition to this. So that will be your first task, separating this out into its own component. The second task is to show the details of the specific dish. Here using the card component, we have already seen the use of the card component for showing the details of the specific dish in the previous exercise. So you need to reorganize the code in to the dish detail component, so that this card is shown in the dish detail component. In addition, we are also supplying you with a list of comments that you need to display in the form of a list here. So, as you can see each comment is shown there by the details of the comment are shown, and the author of the comment is shown here, and then the date is shown as such as a string as given in that comments object that is included in each dish. So look at the details of the comments that are included in each dish to get an idea of where you can get the comment, the author of the comment, and the date for the particular comment. So, this is how you will be displaying each of the comments in your list here to the right side of this. This is the way you're displaying this information on medium to extra large screen size. Now, when you switch to a extra small or a small screen size, so as you can see here, this is a pixel two in portrait mode. So here, you'll see that all the dishes are stacked, but the chosen dish is going to be shown below the list of dishes here, and then the comments are shown in it's own column below the selected dish here as shown in this picture here. So, these are the three tasks that you'll need to complete in this assignment. First task, to move the display of the selected dish into its own component. Second, to show the details of the dish using a card component, and third, to show the list of comments about the selected dish as shown here. Have fun completing the first assignment of this course.

## Peer-graded Assignment: React Components

Deadline Feb 12, 11:59 PM PST

i

It looks like this is your first peer-graded assignment. [Learn more](#)

Ready for the assignment?

You will find instructions below to submit.

### NEW: Assignment reminders

Set an email reminder to complete this on a more convenient date.

Got it

Remind me later

1. [Instructions](#)
2. [My submission](#)
3. [Discussions](#)

In this assignment you will add a new component to the React application to show the details of a selected dish. You will use the Card component and the Bootstrap unstyled list component to prepare the view for this new component.

## Step-By-Step Assignment Instructions

less

### Objectives and Outcomes

In this assignment, you will continue to work with the React application that you have been developing in the exercises. You will add a new component named **DishdetailComponent** that will display the details of a selected dish. You will then design the view for the component using the card component. At the end of this assignment, you should have completed the following tasks:

- Created a new DishdetailComponent and added it to your React application.
- Updated the view of the DishdetailComponent to display the details of the selected dish using an reactstrap card component.
- Updated the view of the DishdetailComponent to display the list of comments about the dish using the Bootstrap unstyled list component.

### Assignment Requirements

This assignment requires you to complete the following tasks. Detailed instructions for each task are given below. The picture of the completed web page included below indicates the location within the web page that will be updated by the three tasks.

#### Task 1

In this task you will be adding a new **DishdetailComponent** to your React application and include the component into the menu component's view so that the details of a specific dish are displayed there:

- Replace the card showing the selected dish in MenuComponent's view with the DishdetailComponent, and make sure to pass the selected dish information as props to the DishdetailComponent.



- Create a new *DishDetail* class in a file named *DishdetailComponent.js* in the *components* folder
- Export the *DishDetail* class from this file so that it can be imported in *MenuComponent.js* and used to construct the view of the selected dish.
- Return a `<div>` from the `render()` function. This `<div>` should use the Bootstrap *row* class to position the content within the `<div>`. This div will display both the details of the dish in a Card and the list of comments side-by-side for medium to extra large screens, but will stack them for xs and sm screens.
- The card should be enclosed inside a `<div>` appropriate Bootstrap column classes so that it occupies the entire 12 columns for the xs and sm screen sizes, and 5 columns for md screens and above. Also apply a class of *m-1* to this div.
- The comments should be enclosed in a `<div>` to which you apply appropriate column classes so that it occupies the entire 12 columns for the xs and sm screen sizes, and 5 columns for md screens and above. Also apply a class of *m-1* to this div.
- If the dish is null then you should return an empty `<div>`

## Task 2

In this task you will be adding a card component to the *DishdetailComponent* view to display the details of the dish given above:

- Implement a function named `renderDish()` that takes the dish as a parameter and returns the JSX code for laying out the details of the dish in a reactstrap Card. You have already seen this as part of the *MenuComponent* class in the exercise earlier.
- Display the name of the dish as the Card title, and the description as the Card text.

## Task 3

In this task you will use the comments that are included in the dish object above to display a list of the comments for the dish. Please use your JavaScript knowledge to recall how you would access an inner property in a JavaScript object that itself points to an array of JavaScript objects (comments). This task involves the following steps:

- Implement a function named `renderComments()` that takes the comments array as a parameter and lays out each comment as shown in the image below. You should use the Bootstrap *list-unstyled* class on the list.
- Each comment should be displayed on two lines, the first one showing the comment, and the second line showing the comment author's name and the date.
- The comments should contain a `<h4>` header with the word "Comments".

- Remember to enclose the header and comments inside a <div> before returning the JSX code. Otherwise React will not do the layout correctly.
- If the comments are null, then you should return an empty <div>.

## Assignment 1: React Components: Additional Resources

10m

Ristorante Con Fusion





**Vadonut**

**Task 1 & 2**

A quintessential ConFusion experience, is it a vada or is it a donut?

**Comments**

Imagine all the eatables, living in conFusion!

-- John Lemon , Oct 17, 2012

Sends anyone to heaven, I wish I could get my mother-in-law to eat it!

-- Paul McVites , Sep 06, 2014

Eat it, just eat it!

-- Michael Jaikishan , Feb 14, 2015

Ultimate, Reaching for the stars!

-- Ringo Starry , Dec 03, 2013

It's your birthday, we're gonna party!

-- 25 Cent , Dec 03, 2011

**Task 3**

## Ristorante Con Fusion

### Uthappizza



## React Resources

- [React Components](#)
- [React Component State](#)
- [React Component Props](#)
- [reactstrap Media Object](#)
- [reactstrap Card](#)
- [Bootstrap unstyled list](#)
- [Bootstrap Grid](#)

## Ideation: Objectives and Outcomes 10m

The first step in your journey towards the implementation of the Capstone project begins with an idea. In this module you will develop the idea for your

project, the set of expected features, survey the market to look at similar ideas to enable you to differentiate your project from others, while at the same time drawing inspiration from them. You are required to submit a formal ideation report following the structure given in the template. This will enable your peers to provide you feedback and suggestions for your project.

Before you get started on a project, the first step is to develop the idea for the project. In this module you will explore how you develop your idea and come up with possible set of features for your project. At the end of this step you should be able to:

- Clearly express the central idea of your project, and identify the problem being addressed
- Delineate a set of features that you expect your website and app should support
- Identify other projects that might have similar features and would act as exemplars for your project

---

## **Ideation Report Template** 10m

### **Project Title**

#### **1. Introduction**

- A brief introduction to your website idea. State the goals of the project.
- The values / benefits (tangible and intangible) this application can bring to a company/organization/end-user.

#### **2. Expected List of Features**

- A brief list of features that you expect your website to support.
- Brief justifications for including these features.

#### **3. Market Survey**

- Do a survey of the Web to find about five web sites that might have similar ideas as yours.
- Briefly compare the features of these applications with your application idea.

#### **4. References**

- Give references to any material / websites / books etc. relevant to your application idea

- Give the links to the websites relevant to your idea, that you listed in the section above.

---

Ideation: Additional Resources 10m

## Honors Peer-graded Assignment: Ideation

1. **Instructions**
2. **My submission**
3. **Discussions**

In this assignment you will be submitting a written report describing the general idea of your project, the expected list of features and a survey of existing projects, websites and/or apps that are similar to your ideas and/or have some features similar to your proposed project. The structure of the written report should adhere to the report template given in this module, and emphasize the points specified in the template. The written submission needs to be no more than three standard Letter/A4 sized pages.

### Review criteria

less

Your submission will be reviewed based on the following criteria by peers in order to provide you with constructive feedback on your project idea:

1. Does the Ideation report clearly state the idea of the project and the primary aim and purpose of the proposed website ?
2. Does the Ideation report list the expected features that will be supported by the website?
3. Did the user provide a survey of related ideas/projects/websites that have some similarities to the proposed idea?
4. Does the Ideation report provide references to suitable sources in support of the project idea?

## Ideation: Additional Resources

### General Resources

- [Ideation \(creative process\)](#)

### Volunteer your Services

- [VolunteerMatch.org](#)
- [Free Code Camp](#)

## React Component Types: Objectives and Outcomes

In this lesson you will learn about various types of React components: Presentational, Container and Functional components. At the end of this lesson you will be able to:

- Identify the salient features and uses for the various types of components
- Create presentational, container and functional components in your React application

## Presentational and Container Components 6m

When you've implemented the dish detail component in the first assignment, you might have noticed that the dish detail component operates purely on the props that are passed to it from the menu component in the form of the selected dish details. So, the dish detail component itself has no local state, and is purely dependent on rendering its view based on the props that is passed on it. So, it is acting as a pure presentational component as we might classify our components. Similarly, a container component may simply store the state information and pass the state information to its children. Let's look at some details next. Some people have informally classified the various kinds of components that we use when we implement our React application into different cuts. So, for example, we can talk about presentational components that are purely responsible for rendering the view based on the props that they are passed in, and then you could have container components that simply make use of presentational components and pass props to them and they are responsible for storing the state (the container components). Similarly, we can talk about these as skinny and fat components. Skinny components mean purely responsible for rendering the view and fat components having a lot more information being tracked there in the form of state. Similarly, we can even label them as dumb and smart components, stateless and stateful components. No matter how you classify them, you begin to see that they can view components as of two kinds. Again, these are informal ways of classifying the components. There is no formal description of these kinds of components in React itself, but this helps us to structure our application when we view the components of different kinds as delineated here. Let's look at specifically presentational and the container components in a little more detail. Taking a broad view of presentational components, we can consider presentational components to be mainly concerned with rendering the view, that is how things look. So, they are mostly interested in the markup and style, so they become the repository for the markup and styles for rendering the view to the screen. So, they render the view based upon the data that is passed on to them through props. So, we have already seen how we

pass that selected dish details through props to the dish detail components. So that being one example of a presentational component. Furthermore, presentational components do not maintain their own local state. So, they won't maintain the state of the application in any way. They may maintain some basic state with respect to only the UI state. So for example, if you have a dialog box popped up or a model popped up on the screen, that state of the dialog box of the model, may be reflected in the local state, but beyond that, the presentational components do not worry about the application state itself and they simply depend upon their opinion to passing the application state that is necessary for rendering the view. So, this gives you a rough idea of how we view a presentational component. On the other hand, we can view a container component as being responsible for tracking the state or at least a part of the state of our application. So, they are responsible for making things work. So, they are more worried about fetching the data, say for example, from a server and then using the data within the application or to set up the state or maybe taking care of updating the state in response to any user interactions on the screen or updating the states based upon the data sent back from the server. So, all these would be responsibilities of container component. So, again this is just a broad way of specifying what a container component may be used for, and they make use of presentational components for actually rendering the view. So, they could wrap the information sent in by the presentational components in their own wrapping divs within the container component in order to position the views that are rendered by the presentational component on the screen, but beyond that they will not be responsible for actually constructing the views. They delegate that responsibility to the presentational component, and also they are responsible for providing the data to the presentational component in the form of props that can be passed in to the child component there. So, they maintain the state and they take care of any state updates and then they will take care of communicating with the server in order to maintain the state or reflecting the state update and so on. So that becomes the responsibility of a container component. So, you can clearly see the separation of concerns between representational component and a container component. So, if you view your React application as consisting of these two kinds, then implementing your application or at least thinking of how you implement your application becomes more easier. So, you could only think about presentational components that are purely responsible for the views, and then you could think about container components that make use of the presentational components in order to construct the overall view of your application. Again, this is just an informal way of looking at how you can organize your React application itself into a hierarchy of components. So, with this view, let's go and work on our React application and then reconstruct it to use a presentational component say for example, we can



redesign our menu component to be a pure presentational component, and we can redesign another component on top of it as a parent, which will be responsible for the state. We have already lifted the state into app.js in the previous exercise of the last module.

---

## Exercise (Video): Presentational and Container Components 21m

Now that we have seen, a little bit about Presentational and Container Components. Let's go ahead and reorganize our React application, so that we will implement the menu component as a pure presentational component, and the dish detail component is already a pure presentational component, and we will lift all state information into a container component which we will implement in the exercise. As you work on your project, make sure that yarn start or npm start that you executed in the very first exercise, keeps running continuously, so you can see that my yarn start is running continuously in my terminal here, and then it'll recompile whenever I make any changes to my application. So, you can look at what has been printed on the terminal or command window in order to get an idea, if there are any errors they will be reflected here. Now in this case as I can notice, it says that the 'logo' is defined but never used in the app.js file. So I'm going to go ahead and remove that from my app.js file and clean it up a little bit. Any time you change your code in your application, it will be recompiled automatically, because the yarn start is running, and it is running the compiler behind the scenes keeping a watchful on the files, and then recompile the files whenever any changes are made, and then start summing up the updated versions, from the building server that is running from the create react app script that we have installed early. Going to our application. In the application, in app.js, I'm just going to first remove this logo from there, and also I will delete this logo.svg file from there, because I don't need that anymore. So I will remove that also from my application. Furthermore, going into the components folder, I'm creating a new file here called MainComponent.js. So this would be acting as a container component for my application. Now what I'm going to do is, I'm going to remove much of the work from app.js into the MainComponent.js so I'm just going to copy, that code from app.js and move into the MainComponent.js and then paste it in here, because my main components structure will be very much similar to the app component, and whatever the app component was doing earlier, much of that work I will move it into the main component. So here, I'm not going to be using any styles for the main component because it is not doing any rendering of the views so I don't need any styles here. You're simply making use of the views that are exported by the various components that we already



have. Similarly, I will go into the menu component, and then this DishDetail component that I was using in the menu component, and I'm going to cut that and then move it into the main component. So the main component becomes responsible for both the menu, and the DishDetail line. It will do all the rendering on our behalf. With these changes, let's go ahead and rename this as Main component. So class Main extends Main component and the state is stored right there, and within the render function. I'm going to first render the navbar, as we saw here, and will render the menu, and also we'll render the DishDetail component also right below here and then I will pass the dish information right here. Now to pass the selected dish information obviously, I need to know the selected dish in the main component. So, going to the state I'm going to introduce a new variable here called selectedDish, or new property, called selectedDish, and then I will assign that to null here. So, as you notice the selectedDish, I'm going to move that into the main component from the menu component. So, in the menu component I can now remove this SelectedDish completely, and then this onDishselect also I'm going to cut from the menu component, and move it into the main component here. So that, the main component is responsible for everything related to the state of my application. So the selectedDish will also be tracked by the main component, and that is going to be supplied to my Dishdetail component from here. So, once we have done these updates then, you notice that in the menu component I still need to respond to the one click method. So, to help me to do that, from the main component, I'm going to pass in the onDishselect to the menu component. So, for the menu component I'm going to come in here, and then I will say onClick. So, this onClick prop I'm going to pass in from the menu component here. So, say "Menu component onClick". Within the onClick, what am going to do is, I'm going to go to the main component. I'm going to remove this whole thing from there, will fix the menu component a little bit later, and then I'm going to come and paste that into the onClick method. What I would expect for the onClick method to just obtain the dishId. I don't need the entire dish information, because that is already there in the dishes here, so I can easily grab the information about the dish from there. So what I'm going to track is only the dishId, in the state of the menu component of the dishId of the selected dish. So, I'll pass in the dish ID as the parameter to the onclick method here. With this change what I have done is I have passed in the onClick as a property to the menu component, and also this onClick when it is invoked, will pass the dishId parameter, which will come in as the parameter here for the onDishselect, and then I'll say, "This Setstate dishId". So the selectedDish property here is only tracking the dishId, not the entire dish information as we did in the menu component earlier. So we're only going to track the dishId not the entire dish. Now this also means that when I passed the dish information to the dish detail component, I need to go into this dishes array, and then select out the

specific dish information, and then pass that to the DishdetailComponent. So to do that, we'll just make use of the array operators in JavaScript to enable us to extract the details of the selectedDish here. So, for the Dishdetail that I am invoking here for the DishdetailComponent, I'm going to pass in the dish as "this.State.dishes." So from this dishes array, I'm going to select out one dish, that one particular dish is the one for which the dishId matches the selected dish here. So how do we do that? From your knowledge of JavaScript, you can simply say, "This state dishes", and inside here I'm going to supply the ADOL function here. So, again you see me using the essix ADOL function here. So I'll say dishId. So this dish is the parameter. So then I do this state dishes, and then for the dishes I apply the filter option here. So this state dishes filter, and when I apply the filter, this filter will operate on each dish in the dishes array. So I will get access to each dish. For each dish, what I'm going to do, is to check dishId is this the same as this state selected dish? Okay. So this ADOL function, what it does is, it helps me to select out all those dishes for which the dishId matches the selectedDish, which contains the dishId of the selectedDish. So, I will compare the dishId of each dish with that, and then extract the rock. So it does filter function, if you recall from your knowledge of JavaScript this filter function will give the sub array of the dishes for which the sub-array contains are rather, constrained part of the array, or just the elements from the array, for which this property, the dishId matches selected dish. Obviously, you will get only one dish because the dishId is unique for each dish, by that filter function will supply that as an array. So, I will have to select the first item from this array there. So, that's why I'll say zero. So index zero, the item in index zero although, you have to remember that there will be only one item in this array, but still we need to supply that one item. So, by doing this, what we are ending up doing? Is selecting the specific dish, which is the selectedDish and then passing the dish information to the DishdetailComponent. So, with this change I am passing in whatever I need to do to the DishdetailComponent in it's props as dish, and the menu component is also set up appropriately. Now, the DishdetailComponent you have already implemented that as part of your first assignment so it should be working correctly when the dish is passed in as the props to it and one more minor. Change here when you import. Recall that the MainComponent is already in the Components folder. So, I don't need that part. And also for the DishdetailComponent as such. And also, one more minor change we need to import the dishes from dot dot/shared dot dishes because this is in the shared folder which is in the source folder. So, you go up, one level into the shared dishes folder. And of course, one minor error that you would notice is that, this is main class in the MainComponent. So, I should be exporting the main class from the MainComponent not the app class. So with this update, now my main component is all ready to act as the container for my application. Now, this main

component is going to be used within that app Component. So, we'll go back to the app Component and then, I'm going to remove all these from the app Component. And so, we'll just import main from the MainComponent and then, I no longer need dishes in the app Component because that is not required. And also, I am not storing any state in the app Component, so I can simply remove the constructor completely and app Components itself receives no props. After these changes, now this Navbar has been moved into the MainComponent so I can remove that completely. And this is just the MainComponent rendered inside div here. With this change, my app component is now ready to simply make use of the MainComponent and then, render the MainComponent and then, return that to index start js file. Now, we need to go and complete the update of the MenuComponent. So, going to the MenuComponent, recall that the MainComponent is passing in this onClick as part of the props to the MenuComponent. So, noting that going into the MenuComponent. I realize that those props are coming into that MenuComponent itself. So, within the MenuComponent, when the card is clicked in onClick. So, this particular line in onClick, when that card is clicked, I will say, an added function and say this props onClick. And this onClick itself takes one parameter which is that dish id. So, the particular dish that has been rendered, that dishes id, I have passed this back to the onClick method that has been sent to me from the MainComponent. And when you go to the MainComponent, you realize that the onClick method is implemented to use the dish id as the parameter, which is then supplied to this onDishSelect, as the parameter here. And one final change in the MenuComponent. Now since, the MenuComponent no longer uses the DishDetail component, so I'm just going to go ahead and remove this DishDetail component completely from the MenuComponent. So with these updates, now my main MenuComponent is also updated. The MainComponent is also updated and app.js is also update. Now, one minor change, that you can make to your DishDetail component. Recall that in your DishdetailComponent, when you are rendering the date, you are simply rendering the date as an ISO string there. Now, I will give you some small snippet of code, which will turn that ISO string into an actual date that can be displayed in a format that is more easily readable. So, going to the DishdetailComponent, I'm not going to show you how I implemented my DishdetailComponent. I'm to hide on that part here. We'll just go into this comment.date here, and this comment.date here, instead of simply rendering the comment.date as a string here, I'm going to use a international date format that JavaScript supports and then, update this. So, we'll update this as new Intl, this is part of JavaScript itself. So, new Intl DateTimeFormat, and then, this date time format takes as the first parameter. The format of the date that you want to display. So, I'm going to use the US style of formatting the date. I'll say en-US, you can supply your own local formatting for the date, if you want here. So, this

is how the date, time, format function works here. So, the date, time, format function as you can see, new locales which is the string and then, further options here. So, the second part, is the options how you want to render the date itself. So, for the format, for the date itself. I will say year numeric, month, i will render the month as short. Meaning that the month will be either Jan, Feb, Mar, and so on. So, first three letters of that and then, that date would be rendered as two digit. Again, if you look up the documentation for the Intl DateTimeFormat. A link to which I have provided and the additional resources, you will see why this formatting works very well to print the date. So, this is the format that I'm going to apply to my date string that I have here. So, this will take that date in the ISO format and then, this particular thing, I'm going to apply to the comment.date here. And within the format, what I need to supply is convert this to a new date and then, say Date parse comment.date here, closing. So, one more to close this parenthesis, and so that's it. So, with this change, your date in your DishDetail will be printed out as, say for example, March 3, 1997 and so on, so in that format. So again, this whole string that I am applying here, I have supplied that in the instructions. So, just go ahead and take a look at that string that I've supplied and then, you'll be able to notice what I have done to this. So, this minor change or update to DishDetailComponent. So going into, let me apply the container class to the DishDetailComponent and then save the changes, and go and take a look at the browser. Going to the browser, let me click on a dish, and then you'll now notice that, once you applied the container glass in your DishDetailComponent, it is being correctly rendered, aligned properly with the rest of your view in the browser. Again, click on another dish and then, you can see the details of the dish. Also notice how the date is now being rendered because of the minor change that we did in the DishDetailComponent. So, that's the change that we make to the way the date is rendered in the comments. With this, we complete this exercise. In this exercise, we have seen how we can divide our application into different kinds of components, so we used presentational components and also container component. With this, we complete this exercise. This is a good time for you to do git-commit with the message presentational and container components.

## **Exercise (Instructions): Presentational and Container Components**

### **Objectives and Outcomes**

In this exercise we understand about how presentational components deal with the look and feel of the app and container components deal with the data and behavior. At the end of this exercise you will learn about:

- Organizing your React app into presentational and container components
- Enable your presentational components to be concerned with the look and feel of your app
- Enable container components to deal with the state, provide the data and handle user interactions.

## Add a Container Component

- Add a new component named *MainComponent.js* in the components folder and update its contents as follows:

```
import React, { Component } from 'react';
import { Navbar, NavbarBrand } from 'reactstrap';
import Menu from './MenuComponent';
import DishDetail from './DishdetailComponent';
import { DISHES } from '../shared/dishes';
```

```
class Main extends Component {

  constructor(props) {
    super(props);
    this.state = {
      dishes: DISHES,
      selectedDish: null
    };
  }

  onDishSelect(dishId) {
    this.setState({ selectedDish: dishId});
  }

  render() {
    return (
      <div>
        <Navbar dark color="primary">
          <div className="container">
```

```

        <NavbarBrand href="/">Ristorante Con Fusion</NavbarBrand>
      </div>
    </Navbar>
    <Menu dishes={this.state.dishes} onClick={(dishId) => this.onDishSelect(dishId)} />
    <DishDetail dish={this.state.dishes.filter((dish) => dish.id === this.state.selectedDish)[0]} />
  </div>
);
}
}

```

```
export default Main;
```

- Update the *App.js* by removing the state related information, and make use of Main Component to render the UI:

```

. . .
import Main from './components/MainComponent';

```

```
class App extends Component {
```

```

  render() {
    return (
      <div className="App">
        <Main />
      </div>
    );
  }
}

```

```

. . .

```

## Turn Menu Component into a Presentational Component

- Open MenuComponent.js and update its contents by removing the state and removing the DishdetailComponent reference, and make use of the onClick supplied by MainComponent through the props to deal with the clicking of a menu item:

• • •

```
<Card key={dish.id}
      onClick={() => this.props.onClick(dish.id)}>
```

• • •

- The DishdetailComponent is already structured as a presentational component and hence needs no further update, except wrapping the return value from render() within a <div> with the className as container.
- To print out the date for a comment in a format more suitable for human readability, you can update your renderComment function with the code snippet shown below:

1

```
{new Intl.DateTimeFormat('en-US', { year: 'numeric', month: 'short', day: '2-digit' }).format(new Date(Date.parse(comment.date)))}
```

- Save all the changes and do a Git commit with the message "Presentational and Container Components"

## Conclusions

In this exercise you learnt how to structure your app into presentational and container components.

## React Components: Lifecycle Methods Part 2

In the previous lesson, I had briefly introduced you to the Lifecycle Methods of React Components. We had already examined the lifecycle methods that are invoked when the component gets mounted. So, when we were looking at the react application example in the previous lesson, the component was just getting mounted at the top. Now, we have two components, the Menu component and the DishDetail component. You'll see that when you click on any of the menu items the DishDetail component gets re-rendered with the new details of the selected Dish. So, in this case, periodically, that DishDetail components view needs to be

updated whenever a new menu item is clicked. So that means that the DishDetail component will undergo an updating stage. So, let's look at the lifecycle methods that are invoked when a component gets updated. So again, reminding you that the lifecycle methods are invoked at the point of mounting of a component, the updating of a component, and unmounting of the component. When a component is being updated, there are several lifecycle methods associated with this process. So these lifecycle methods are called when a component is being re-rendered or being updated. Now, this could be caused either because the props that are supplied to the component changed, or the internal state of the component changed. So whatever be the reason, the component is re-rendered and at that point we have access to several lifecycle method hooks that are invoked. Now, the `getDerivedStateFromProps` and `render`, you have already seen associated with the mounting. The same are also available when the component is being updated. In addition, we have a method called `shouldComponentUpdate`. This method will return a Boolean variable. If the component will never get updated during your lifecycle, then you can return a false from `shouldComponentUpdate` to inform your react's rendering process that this component will never update so you don't even need to consider this while you're re-rendering the Doc. Now, normally we just ignore that point but if you wish to you can include that into certain components but be clear that the component will never update once it is mounted. The render will be called every time the component is re-rendered. So, every time you update the props of the stage for the component, the render has to be called, and so the component is re-rendered. Similarly, you have access to a method called `getSnapshotBeforeUpdate`. This may be needed in situations, for example, when you're scrolling, you have this scrollbar in a component and you're scrolling and you want to remember the position of the scroll at the point so that when the component re-renders then you will retain that scrolling position, then that would be useful for saving the information. Now, the other method that we will see is the `componentDidUpdate` method. This method, as you would expect, is invoked when the component is updated. So, let's go back to our application and then examine a couple of things. We'll look at the use of the `render` and the `componentDidUpdate`. Now, we will go into the DishDetail component and then configure that to check to see how these methods are being invoked. Also, this is a good stage for me to remind you that there are two methods that were used earlier called `componentWillReceiveProps()` and `componentWillUpdate()`. These two methods have been duplicated starting with React Version 16.3, and so you are discouraged from using these anymore in your React Application. Now going to my application, in the Menu component let me copy this `componentDidMount` from the Menu component and then I will paste it into the DishDetail component



here, so that we'll see that the `componentDidMount()` for `DishDetail` components will be invoked at that point. Now, in addition, since the `DishDetail` component doesn't have a constructor and actually you don't need it, so that's why I don't have the constructor method. Now, in addition to the `componentDidMount`, we will also invoke the `componentDidUpdate` method in the `DishDetail` component, and then in here, I will log this `DishDetail` component `componentDidUpdate` invoked, and then we will also include this into the render method of our `DishDetail` component. So `DishDetail` component render invoked. Let's save the changes and then go take a look at our application and see the changes in the JavaScript console. Going to our browser where our application is rendered, you can see that in the JavaScript console, you see the `Menu` component constructor invoked, `Menu` Component render invoked, and then the `DishDetail` component render invoked, and the `Menu` Component's `componentDidMount` invoked, and then the `DishDetail` Component's `componentDidMount` invoked, so they are called in their order. Now you are wondering why the `Menu` components `componentDidMount` was called after the `DishDetail` Component rendered was invoked. This was invoked because in the `Menu` component's to render better, we had included the `DishDetail` component into the `Menu` components then the render, so the `DishDetail` Component has to be created and then it's random method has to be invoked. And then at that point, the `Menu` component finishes mounting into the top. So that's where you'll see the sequence of methods being invoked. Now, let's click on one of the items in the menu. So when I click on the item in the menu, you would see that, that particular item is rendered at the bottom here, and then at that point, again, look at the sequence of the invocations of the methods. So you see that the `Menu` component render is invoked because you now changed what is in the `Menu` components are render invoked. The `DishDetail` component is render invoked, and notice that the `DishDetail` components `componentDidUpdate` method got invoked here. So, that's the reason because only are the `DishDetail` component was attached the `DishDetail` component was not rendering anything, it was rendering an empty dev earlier, but when we selected a dish that selected dish gets rendered in to the view there so that's why you see that the `Menu` component is invoked again and the `DishDetail` component. One more time, let me click on another item and then you would see that the sequence is still maintained, the `Menu` components render is invoked, the `DishDetail` Component's render is invoked, and the `DishDetail` Component's `componentDidUpdate` is invoked here. So that demonstrates to you how that lifecycle methods in both the mounting phase and in the updating phase are invoked for your components. Now, although we had made changes to these item, you can easily delete all those console logs from your files. You don't need to save those console logs, those were meant just to illustrate to you how the lifecycle methods are actually invoked.

---

## Functional Components 3m

So far, we have been always implementing React Components by using a class component. We have been sub-classing the React Component class and then defining our own components the way we implemented the Menu Component and also the detailed component and the main component. Now, for components that simply work only using their props that are sent by their parent, there is an even simpler way of implementing the components what are called as Functional Components. Then we implement a simple function that returns a bunch of react elements. Let's look at this in a little more detail next. Again, looking back at the way we have been implementing components so far, we have been extending the React Component class and then defining our components. And also, when we do that the class requires to implement this method called as the render which returns the view. So, inside the render we were including a written statement within which we were returning the group of react elements that define the view for this component. And we also notice that when we define a class component we can define the local state and this local state is defined in the constructor and then we can also operate the local state and so on. So furthermore, a class component gives you access to certain special methods called as lifecycle hooks. We will talk about lifecycle hooks a little bit later in this course. But if your component is a very simple component and doesn't require this much of elaborate implementation, there's a much simpler way of implementing React Components. This simple way of implementing React Components is called as Functional Components. In a Functional Component the easier way of doing this is to simply define a JavaScript function that returns either one react element or a collection of react elements too that defines the view that this component is supposed to render. So, a Functional Component is a lot more simpler way of implementing the component where you don't need a local state, where you don't need access to life lifecycle hooks. So, this Functional Components simply receives props as it's parameter. So, within the props all the attributes that we define, all the component will become available within our component, and we can make use of them when they render the views. But the only restriction is that Functional Components cannot have any local state and cannot have access to the lifecycle hooks that we will talk about later. So, in many cases you might have a simple React Component that just simply renders the view based on the props that it is sent to and a Functional Component, it is a much easier way of implementing a React Component. So, we will go back to our application and then restructure our application to simply use Functional Components. We will notice that our detailed component, the Menu Component can easily be turned

into pure Functional Components without losing any of the functionality of our application.

---

## Exercise (Video): Functional Components 14m

As we learned in the previous lecture, in this exercise, we're going to go into our React Application and then turn the menu component and dish detail component to pure functional components. As we realized, in the previous lecture, we have restructured our application such that the menu component and the dish detail component purely work on the props that they are passed into. They are not maintaining any local state, so they are very good candidates for being turned into functional components. So, if you have a React Application where a component is purely rendering of you based on props, then functional components is the way to go for designing your component. To further emphasize the point about how you could turn a component into a purely functional component, let me take a look at our terminal window or the command window. You immediately notice an interesting comment there which says Line 6 in the MenuComponent.js: Useless constructor. Now, what this is implying is that in the menu component, we have a constructor there, but we are not using it for anything because we are not storing any state or initializing anything there, so the constructor is unnecessary in that. Now as we saw when we designed the app component, we removed the constructor completely from that. So similarly, this is also a suggestion to tell us that we can easily go in and then modify our menu component into a pure functional component. Similarly, they will modify the dish detail component also into a pure functional component. It doesn't take a whole lot of effort to do this as you will see when we implement this in the exercise. Going to our code here, the first thing that we notice in the menu component is we don't need this constructor, so we can just get rid of the constructor completely. Furthermore, we can easily restructure this code into functional components. Not only that, we realized that all this part of the code which is rendering this card here, can be pulled out into its own functional component and implemented here. So, what I'm going to do is, since I am no longer going to implement this as a class component, I'm going to remove that first, and then furthermore, that is no longer necessary. That render is no longer necessary, so I am cleaning up all this code that I don't need anymore. So, I'm going to remove all this from my code, but I will still have the export menu. So after cleaning this up, the first thing that I'm going to do is to pull out this card view from here and then make it into its own functional component. So, we will define a functional component here as shown here. We'll say "function RenderMenuitem." And then inside here, we will receive prompts as the parameter here. Now this is one way of implementing it. Now, if you know what

you are going to be receiving to your props, so instead of just typing props, I can simply go in and instead type within braces. Because the process is JavaScript object, so I can even easily specify the various properties that are going to come in as part of the JavaScript object there. So I'll simply say "dish, onClick," and that's it. And this function is simply going to return a view here. So what is the view that it is going to return? It is going to return this card as the view here. So I'm going to just cut the card from here, and then put this into this return function. That is it. Now, my first functional component is ready. So this functional component is simply rendering that menu item based upon the dish information that it is sent and also based on click that it sent. It will ensure that the view is rendered in a react strap card format here. Now, once we have done this, then I can go in and make this part into its own functional component. Now here, I will show you another way of implementing a function component. So we'll define `const menu` equal to props. And in here. This is another way of implementing a functional component. So here I am using an added function from ES-6 to define this. So, this is the parameter that is going to receive the props parameter. I could easily rewrite this as function like this here. I can say `function menu`, within brackets, props, but I'm just showing you two different ways of implementing the same thing. Now in there, all I need to do is take us whole code that we have here and then move that into this here. So now, my `const menu` props. This function is going to return the menu and inside here. Now, this is where I am iterating over all the items. So, here I'm going to make use of the `render menu item` functional component that I have just implemented up here, and the `render menu item` takes two parameters. So, it takes the parameters `dish` which is the dish that has received here. So, this `dish` and then the second parameter is `onclick` which is nothing but props `onclick`. So, the `onclick` which was sent in by the main component that I am parsing in to the `render menu` component as the function here. So, I'm just parsing in the function from here into the `render menu item` component, that does change. I am now making use of the `Render menu item` functional components to construct the view for each dish and this in turn is inside the `div` which is going to construct a list of all the dishes, and on that I am now making use of `within` here to render the menu item. So, no further changes to this here, this remains as such. So, with this simple rearrangement, I have taken my menu component and turned it into a purely functional component. Now, within the menu component, one more change that we need to do is to change this to `const menu` `props.dishes.map` because it is not `dish.props`. So, this props is coming in as the parameter for this function, so this should be `props.dishes.map` rather than `dish.props`. Similarly, in the `render menu item`, we can remove the `dish.props` and it should be just `onclick` here, because this `onclick` is coming in as a parameter here. That's it, but this change my menu component is now turned into a function component. Now similarly, let's go to

the dish detail component and then modify that also into a functional component. So, going into the dish detail component, again same thing, I don't need this component anymore. I can remove that part completely from there and then this class component is also removed. We had the render dish function that you implemented earlier, now I'm going to turn that render-dish function into its own component here so, I will simply rename that as function render dish with the capital R. Remember that user defined components always start with capital letters. So, this basically turns this function that we had here in the class component into its own functional component, but when we do this, the render dish no longer gets the dish here. Instead, it'll get the dish in the form of props here. So, I'm going to enclose that within braces to turn that into that, since my props will only receive one property in the props object, so, I'm just extracting it out right there by simply defining that parameter as an object which contains dish here. Similarly, going to the render comments component, the render comments component also I'm going to turn that parameter into comments, and then I'll turn this also into a functional parameter with a capital R render comments and that's it. So, now we have two functions here, functional components render dish and render comments going down into my code here, I will simply remove this render function and then turn this into a function component called dish detail. Again, notice how I am changing this into a functional component. Now, once I change that into a functional component, in here I have to turn this into a call to the functional components. So I will say, render dish and will say, dish, props dish here, and similarly, the next line, we just turn that into render comments equal to (within braces), props dish comments, that is it. With these changes, both my dish detail component and the menu component are now updated to be functional components. Also, this should be just props.dish rather than dish.props start dish, because props is coming in as the parameter here. Let's save the changes and go and take a look at this in the browser. Going to the browser, you can now see that our application is rendering just as before. When I click on an item, then the details of the item are displayed down below, just like before. So, by turning our menu component and dish detail component into functional components, we have not changed any of the functionality of our application, our application behaves exactly like before. But, we see that we have a simpler way of implementing the functional components. With this, we complete this exercise. In this exercise, we have learnt how to define functional components and then modified our React application to make use of functional components. This is a good time for you to do a bit comment, with the message functional components.

## **Exercise (Instructions): Functional Components**

### **Objectives and Outcomes**

In this lesson we explore the design of functional components in React. We will reimplement both the MenuComponent and DishdetailComponent as pure functional components. At the end of this exercise you will be able to:

- Implement functional components in React
- Illustrate the reimplementation of presentational components as pure functional components

## Implementing Functional Components

- Open MenuComponent.js and update it as follows:

```
import React from 'react';
import { Card, CardImg, CardImgOverlay,
  CardTitle } from 'reactstrap';

function RenderMenuItem ({dish, onClick}) {
  return (
    <Card
      onClick={() => onClick(dish.id)}>
      <CardImg width="100%" src={dish.image} alt={dish.name} />
      <CardImgOverlay>
        <CardTitle>{dish.name}</CardTitle>
      </CardImgOverlay>
    </Card>
  );
}

const Menu = (props) => {

  const menu = props.dishes.map((dish) => {
    return (
      <div className="col-12 col-md-5 m-1" key={dish.id}>
        <RenderMenuItem dish={dish} onClick={props.onClick} />
      </div>
    );
  });
};
```

```

    return (
      <div className="container">
        <div className="row">
          {menu}
        </div>
      </div>
    );
  }

```

```
export default Menu;
```

- Then open DishdetailComponent.js and update it as follows:

```

import React from 'react';
import { Card, CardImg, CardText, CardBody,
  CardTitle } from 'reactstrap';

```

```
function RenderDish({dish}) {
```

```
  . . .
```

```
}
```

```
function RenderComments({comments}) {
```

```
  . . .
```

```
}
```

```
const DishDetail = (props) => {
```

```
  . . .
```

```
}
```

```
export default DishDetail;
```

- Save all the changes and do a Git commit with the message "Functional Components".

## Conclusions

In this exercise we have learnt to implement our components as pure functional components.

## React Component Types: Additional Resources

### PDFs of Presentations

1-Component-Types-Part1.pdf

PDF File

2-Component-Types-Part2.pdf

PDF File

### Other Resources

- [Presentational and Container Components](#)
- [Presentational and Container Components \(Redux Perspective\)](#)
- [React Component Patterns](#)
- [Functional Stateless Components in React](#)

## React Router: Objectives and Outcomes

In this lesson we cover the basics of React router. We examine how the router enables the navigation among views of various components that form part of a React application. At the end of this lesson you will be able to:

- Set up the router module to enable navigation among multiple component views



- Set up the routes to enable the navigation

---

## React Virtual DOM 6m

In the Browser world, you often hear people talking about the Browser DOM and DOM Manipulation and so on. In the React world, you often hear people talking about the Virtual DOM. What exactly is the Virtual DOM and how is it different from the Browser DOM, and why do we have a Virtual DOM? Similarly, if you learn about `view.js`, there also, people mention about the Virtual DOM. Let's learn a little bit about virtual DOM and why is it interesting in a React application. Before we talk about Virtual DOM, let me again reiterate that the Browser DOM is a browser object. And so when you render something in your web page, the Browser DOM is built up, and so any changes that you want to make to your web page will be effected when you make changes to the Browser DOM. Now, in the React world, the React application maintains a Virtual DOM. The Virtual DOM is a React object. The Virtual DOM in React terminology is a lightweight representation of a Browser DOM. Now, since the Virtual DOM is an in-memory object in your React application, the Virtual DOM can be easily manipulated by React whenever it's required. So, manipulations are extremely fast compared to manipulating the Browser DOM. When you change anything in the Browser DOM, you need to go and re-render the web page allover again. But the Virtual DOM, since it is maintained in memory by your React application, you can easily make changes to the Virtual DOM. So, when you have any changes to the state of your components or changes to the props that a component obtains, then that may result in the component being re-rendered. So, this re-rendering is initially done to the Virtual DOM, so any changes are reflected at the Virtual DOM first. And then, after that, we will see how the Browser DOM gets manipulated. So this Virtual DOM is created completely from scratch every time you call the `setState`. So, you'll see that whenever you call `setState`, you are changing the state with a new React application, which also means that it is possible that some of the props passed into the child components to change, and so the child components may have to be re-rendered. So, in the Virtual DOM hierarchy that you build up in your React application, not all components make it affected by changes to the state of your application. So, as an example, on the left side, I show you a Virtual DOM tree here. And in that Virtual DOM, you see I have marked three components there with red red color. These three components are the ones that have been affected by a change in the state for my React application. Now, what that means is that you will have to re-render your view in the browser, that means that you will need to manipulate the Browser DOM to make the Browser

DOM correspond to the changes that have been effective in the Virtual DOM. So, the React application's work is that they will do a difference between the previous version of the DOM and the current modified version of the DOM. So, to notice which parts of the Virtual DOM tree have changed, then you know which of the components need to be re-rendered. So in this case, for example, when these three red components are effected due to the change in the state, then it may result in re-rendering of all those nodes that are marked in purple to the right of the screen. So, only that part of the DOM tree may have to be re-rendered to ensure that the changes are reflected into the Browser DOM. So how is this actually done? Now, with React because React maintains the Virtual DOM which is a React object, and as we have understood, manipulating the Virtual DOM is very very fast and effective. So every time there are changes to the Virtual DOM and you are at the point of re-rendering the view in the Browser DOM, then to update the DOM, React runs in diffing algorithm. The diffing algorithm will detect all the nodes that have changed. And so, once it detects all the nodes that are changed as we have seen, the red color nodes that were marked in the tree there, this may result in updates to the entire sub-tree if the diffing detects that two elements are of different types. So, the diffing algorithm identifies the minimum number of components or minimum part of the tree that needs to be updated in order to make the modified version in sync with that Browser DOM. Then you are rendering multiple items, for example, in a list. You can use the key attribute in the list items in order to indicate which child elements are stable. Indeed, that is the reason when we rendered list, we always supply the key attribute to each list item, and each key attribute was a unique identifier for that particular item in the list. So, when the diffing algorithm works, if it notices that some parts of the list cannot change, they don't need to be re-rendered. And so, it'll re-render only those list items that have actually been modified. Now, with the React 16, there is a new version of the React diffing algorithm called React Fiber. This is a new reconciliation algorithm that has been launched with React 16 and it is a lot more faster in performing the diffing and then identifying what needs to be changed in the Browser DOM to update the views. With this quick understanding of what the React Virtual DOM is, let's move on to learn more about how to make use of the React router in this lesson.

---

## Exercise (Video): Header and Footer 17m

In this exercise, we're going to add in a little more of details to our React application that we have been working on. We will add in Font Awesome based icons to our React application, and also add in a header and footer for our application so that it gets a nice, clean structure for the views within our application. To get started on this exercise, let's first install Font Awesome into

our React application. So to do that, add the prompt type `yarn add font-awesome 4.7.0` or do `NPM install font-awesome 4.7.0`, minus, minus save. And once that is installed then we'll also add in yet another CSS classes based NPM module called Bootstrap-Social which allows us to create social media buttons in our application. So to do that add the front type `yarn add bootstrap-social@5.1.1` or do `NPM install` of the same. So once that is completed, then we'll go ahead and update our application to make use of both Font-Awesome and Bootstrap. So going to our application, let's open up `index.js` and the import Font-Awesome and Bootstrap Social so that we can make use of it within our application. So import `font-awesome/css/font-awesome.min.css`. And also import `bootstrap-social/bootstrap-social.css`. Now, our application can make use of both Font-Awesome and Bootstrap-Social. Next, going into the components for React, we will add in a header component and a footer component. The reason for adding a header and footer component is so that all our pages in our application they get the same header and footer. So going into the Components folder, let's add in the file named `HeaderComponent.js`. And in the `HeaderComponent.js`, let's set up the header for our application. So we'll say, `import React { Component } from 'react'`. So in the header, what I'm going to do is I'm going to move the Navbar from Main Component into the header and also create a footer here. So in the header, let's create a class named Header which extends the component and then in here as you realize when you create a class component, you need to implement the render method here. Now the reason for creating this as a class component as opposed to a functional component will become more clearer when we go to the next exercise because we need to maintain some UI state in the Header Component. So in here we'll return, in here we can make use of what is called as React fragment which enables us to group together a bunch of React elements and then return it. Now when you use a React fragment here this is the short form syntax for using the React fragment. The long form would be to say, `React.Fragment` like this here. Now I'm going to use the short form for this and this enables us to group together a bunch of React elements and then return it. And also ensures that their alternative they're going to use a `div` to enclose all the React elements. Now if you use a `div` that'll add in one more node into our DOM by using the React fragment, you don't add in an extra node into the DOM, you just add in the React elements directly into the DOM. So that's the reason for using the React fragment here. So inside here I'm going to go into the Main Component and then I'm going to cut out the Navbar and the NavbarBrand from here and then move it into the Header Component. And also from the Main Component, I'm going to remove the Navbar from the Main Component and then move it to the Header Component here. Because that is where we're going to be making use of the Navbar element. So the header will contain the Navbar for our application. At the same time I'm going to remove this color primary from the

Navbar because I'm going to define my own CSS classes such that the Navbar dark color would be determined by myself here. And also, I will import one more reactstrap component called the Jumbotron. Now using the Jumbotron, I'm going to create a Jumbotron into my header here, the Jumbotron allows me to specify some information that can be displayed at the top in my header. Now it'll become more clearer to you once you see what the Jumbotron looks like. If you have taken the bootstrap course, you know what the Jumbotron does. So that's what we're doing inside here. So in the Jumbotron, let me add in a div here for the container. And then inside the div, I will add in another div with the class name row and then row-header. Now the row-header would be a CSS class that I'm going to introduce into my application a little bit later. So inside here row-header and then in here I will introduce a column class here, column div here and inside this will define some content that goes into the Jumbotron. So we'll say, h1 and then P. Now if you want to, you can simply copy and paste this sentence from the exercise instructions. There has just some content for my header, here. A simple sentence to describe the Ristorante. So, with this, we have added in a header component into our application, and then, don't forget to export the header from here. And once we have done that, we will import the header component into the main component and then, use that in the main component. Now that we have designed the header component, let's go into main component and then, import the header component from the header component file, here. And then, we will make use of the header component in our main component, here. So, right there, I would introduce header component into our application. Let's save the change and then, go and take a quick look at our application in the browser. Going to the browser, the browser points out that it doesn't understand this. So, from the fragments documentation in the React site, it says the short syntax may not be understood by all the popular browsers. So, let me go ahead and change that to React Fragment. So, going to header component, let me rewrite this as React Fragment instead, and then, save the changes, and then let's go and take a look at our application in the browser. Going to the browser, you can now see how the jumbotron is added into our application. You see the header has disappeared. It's because we haven't applied the proper color to the header. I removed the color primary from the header, so that's why it has disappeared, but we're going to fix that a little bit later by adding in some CSS classes. And, you'll see the header with the jumbotron display there and then, the content right there as before. So, continuing to work on our application, let's add in a footer to our application. To add in a footer to our application, go into the Components Folder again and then add in the FooterComponent.js file here, and in footer component, let's import React from React. And then, they'll define the footer component as a function component. So, they'll return from the footer, the details of the function component, and then, export footer from this. Now, the content of the footer

itself, you have a long set of code and it is more easier for us to just simply copy that from the instructions and then, paste it in here. So, let's go ahead and copy that from the instructions and then, paste it into this location. So, we are back in the footer component and I have already cut and pasted all the codes from the instructions in here. Because it's such a long piece of code, I feel that it is more easier just to copy and paste the code from there. Now, this code should be familiar to those of you who have done the previous Bootstrap course because that is how we define the footer in the previous Bootstrap course. After this, let's go ahead and import the footer into our main component, and then, apply it to our application. So, we'll import footer from footer component, and then, let me go ahead and add the footer to our application after the dish detail here. So, we will add in footer into our main component, and then, let's save the changes. Taking a look at our application in the browser, you now see, along with the header, you'll see a footer for our application, but it is not clearly delineated. Let's apply a few CSS classes so that our header and footer come to life with vibrant colors. Going back to our application, in the App.js file, we already have the App.css imported there. So, going into App.css, I'm going to add in a few more CSS classes. Now, even for this, I would suggest you copy all the CSS code from the instructions and then paste it into App.css file. So, here, I have pasted the CSS code in here. You can see that I have defined a row header, a row content, and the footer, and the jumbotron here with some of my own colors, and address, and the navbar dark. And then, let's save the changes, and then, this should apply the CSS classes to the element that I have included in my React application. If you have taken the previous Bootstrap course, you are familiar with all the CSS classes because we have used exactly these in the previous course. After saving all the changes, let's go and take one final look at our modified application. Going to the application in the browser, we can already see how this application is now defined here. So, you can see that the navbar has a new dark purple color, the jumbotron has a light purple color defined here and clearly delineated, and then, down below here, we have the footer for which we have also applied the appropriate background color with the CSS classes that we just defined. And also, this is the use of bootstrap-social. So, it allows us to create social media buttons in our application which you have already learned in the previous Bootstrap course. So with this, our application is already getting a nice structure in place there, with this may complete this exercise. In this exercise, we have done some bookkeeping and adding in a header and footer to our React application to give it a nice clean definition. We added in our own CSS classes and then, applied them to some of the React elements, and also, we use Font Awesome and then bootstrap-social classes to design bootstrap-social buttons in our footer. We will continue to use the Font Awesome icons in the header in the next exercise. Also, we have seen the use of React Fragment in this exercise.

This is a good time for you to do a git-commit with the message, header, and footer.

## Exercise (Instructions): Header and Footer

### Objectives and Outcomes

In this exercise you will add in a header and a footer to our React application using two React components. This will illustrate the use of multiple components put together form the application's view. You will also add in the Font Awesome icons and Bootstrap-social for use within your application. At the end of this exercise you will be able to:

- Use multiple components and their views to put together the view of the application.
- Make use of Font Awesome icons and Bootstrap-social within your React application

### Using Font Awesome Icons and Bootstrap-Social

- First use yarn or npm to fetch Font Awesome and Bootstrap-social to the project by typing the following at the prompt:

1

2

```
yarn add font-awesome@4.7.0
```

```
yarn add bootstrap-social@5.1.1
```

- Then, open index.js file and update it as follows to enable your application to use Font Awesome and Bootstrap Social:

. . .

```
import 'font-awesome/css/font-awesome.css';  
import 'bootstrap-social/bootstrap-social.css';
```

. . .

### Adding a Header and a Footer

- Create a new file named HeaderComponent.js and add the following to it:

```
import React, { Component } from 'react';
```

```

import { Navbar, NavbarBrand, Jumbotron } from 'reactstrap';

class Header extends Component {
  render() {
    return(
      <React.Fragment>
        <Navbar dark>
          <div className="container">
            <NavbarBrand href="/">Ristorante Con Fusion</NavbarBrand>
          </div>
        </Navbar>
        <Jumbotron>
          <div className="container">
            <div className="row row-header">
              <div className="col-12 col-sm-6">
                <h1>Ristorante con Fusion</h1>
                <p>We take inspiration from the World's best cuisines, and
                create a unique fusion experience. Our lipsmacking creations will tickle your cu
                linary senses!</p>
              </div>
            </div>
          </div>
        </Jumbotron>
      </React.Fragment>
    );
  }
}

export default Header;

```

- Then, add another file named FooterComponent.js and add the following to it:

```

import React from 'react';

function Footer(props) {

```

```

return(
<div className="footer">
  <div className="container">
    <div className="row justify-content-center">
      <div className="col-4 offset-1 col-sm-2">
        <h5>Links</h5>
        <ul className="list-unstyled">
          <li><a href="#">Home</a></li>
          <li><a href="#">About</a></li>
          <li><a href="#">Menu</a></li>
          <li><a href="contactus.html">Contact</a></li>
        </ul>
      </div>
      <div className="col-7 col-sm-5">
        <h5>Our Address</h5>
        <address>
          121, Clear Water Bay Road<br />
          Clear Water Bay, Kowloon<br />
          HONG KONG<br />
          <i className="fa fa-phone fa-lg"></i>: +852 1234 5678<br />
          <i className="fa fa-fax fa-lg"></i>: +852 8765 4321<br />
          <i className="fa fa-envelope fa-
lg"></i>: <a href="mailto:confusion@food.net">
            confusion@food.net</a>
          </address>
        </div>
        <div className="col-12 col-sm-4 align-self-center">
          <div className="text-center">
            <a className="btn btn-social-icon btn-
google" href="http://google.com/+><i className="fa fa-google-plus"></i></a>
            <a className="btn btn-social-icon btn-
facebook" href="http://www.facebook.com/profile.php?id="><i className="fa fa-
facebook"></i></a>
            <a className="btn btn-social-icon btn-
linkedin" href="http://www.linkedin.com/in/"><i className="fa fa-
linkedin"></i></a>
            <a className="btn btn-social-icon btn-
twitter" href="http://twitter.com/"><i className="fa fa-twitter"></i></a>

```



```

        <a className="btn btn-social-icon btn-
google" href="http://youtube.com/"><i className="fa fa-youtube"></i></a>
        <a className="btn btn-social-
icon" href="mailto:"><i className="fa fa-envelope-o"></i></a>
    </div>
</div>
</div>
<div className="row justify-content-center">

```

## Integrating Header and Footer into the React Application

- Now we open MainComponent.js and update it to integrate the header and footer into our application:

. . .

```

import Header from './HeaderComponent';
import Footer from './FooterComponent';

```

. . .

```

    <Header />
    <Menu dishes={this.state.dishes} onClick={(dishId) => this.onDishSelect(d
ishId)} />
    <DishDetail dish={this.state.dishes.filter((dish) => dish.id === this.sta
te.selectedDish)[0]} />
    <Footer />

```

. . .

- Then update App.css to add some new CSS classes for use in our application:

```

.row-header{
    margin:0px auto;
    padding:0px auto;
}

```

```

.row-content {
    margin:0px auto;
}

```

```
padding: 50px 0px 50px 0px;
border-bottom: 1px ridge;
min-height:400px;
}
```

```
.footer{
  background-color: #D1C4E9;
  margin:0px auto;
  padding: 20px 0px 20px 0px;
}
```

```
.jumbotron {
  padding:70px 30px 70px 30px;
  margin:0px auto;
  background: #9575CD ;
  color:floralwhite;
}
```

```
address{
  font-size:80%;
  margin:0px;
  color:#0f0f0f;
}
```

```
.navbar-dark {
  background-color: #512DA8;
}
```

- Save all the changes and do a Git commit with the message "Header and Footer"

## Conclusions

In this exercise we updated the React application to use Font Awesome and Bootstrap Social, and also integrated two new components, Header and Footer, into our application.

## **React Router** 8m

---

**Exercise (Video): React Router** 36m

---

**Single Page Applications** 9m

---

**Exercise (Video): Single Page Applications Part 1** 23m

---

**React Router: Parameters** 6m

---

**Exercise (Video): Single Page Applications Part 2** 23m

---

**Assignment 2: React Router and Single Page Applications** 4m

---

**16 readings**

---

**React Component Types: Objectives and Outcomes** 10m

---

**Exercise (Instructions): Presentational and Container Components** 10m

---

**Exercise (Instructions): Functional Components** 10m

---

**React Component Types: Additional Resources** 10m

---

**React Router: Objectives and Outcomes** 10m

---

**Exercise (Instructions): Header and Footer** 10m

---

**Exercise (Instructions): React Router** 10m

---

**React Router: Additional Resources** 10m

---

**Single Page Applications: Objectives and Outcomes** 10m

---

**Exercise (Instructions): Single Page Applications Part 1** 10m

---

**Exercise (Instructions): Single Page Applications Part 2** 10m

---

**Single Page Applications: Additional Resources** 10m

---

Assignment 2: React Router and Single Page Applications: Additional Resources  
10m

---

UI Design and Prototyping: Objectives and Outcomes 10m

---

UI Design and Prototyping Report Template 10m

---

UI Design and Prototyping: Additional Resources 10m

## WEEK

3

7 hours to complete

## React Forms, Flow Architecture and Introduction to Redux

---

In this module you will be introduced to uncontrolled and controlled forms and briefly examine form validation in React applications. You will get an overview of the Flux architecture and introduced to Redux as a way of realizing the flux architecture.

Controlled Forms 5m

---

Exercise (Video): Controlled Forms 35m

---

Exercise (Video): Controlled Form Validation 23m

Uncontrolled Components 2m

---

Exercise (Video): Uncontrolled Forms 17m

---

The Model-View-Controller Framework 7m

---

The Flux Architecture 11m

---

Introduction to Redux 20m

---

Exercise (Video): Introduction to Redux 23m

---

React Redux Forms 4m

---

Exercise (Video): React Redux Form 13m

---

Exercise (Video): React Redux Form Validation 15m

---

Assignment 3: React Forms and Redux 5m

---

**15 readings**

---

Controlled Forms: Objectives and Outcomes 10m

---

Exercise (Instructions): Controlled Forms 10m

---

Exercise (Instructions): Controlled Form Validation 10m

---

Controlled Forms: Additional Resources 10m

---

Uncontrolled Forms: Objectives and Outcomes 10m

---

---

Exercise (Instructions): Uncontrolled Forms 10m

---

Uncontrolled Forms: Additional Resources 10m

---

Introduction to Redux: Objectives and Outcomes 10m

---

Exercise (Instructions): Introduction to Redux 10m

---

Introduction to Redux: Additional Resources 10m

---

React Redux Form: Objectives and Outcomes 10m

---

Exercise (Instructions): React Redux Form 10m

---

**Exercise (Instructions): React Redux Form Validation 10m**

---

**React Redux Form: Additional Resources 10m**

---

**Assignment 3: React Forms and Redux: Additional Resources 10m**

**WEEK**

**4**

12 hours to complete

**More Redux and Client-Server Communication**

---

In this module you will explore Redux further including Redux action, combining reducers, and Redux thunk, client-server communication using Fetch and the REST API. You will get a brief introduction to animation in React. You will also learn about testing, building and deploying React applications.

**Redux Actions 8m**

---

**Exercise (Video): Combining Reducers 9m**

---

**Exercise (Video): Redux Actions** 24m

---

**Redux Thunk** 10m

---

**Exercise (Video): Redux Thunk** 50m

---

**Exercise (Video): React-Redux-Form Revisited** 12m

---

Networking Essentials 17m

---

Brief Representational State Transfer (REST) 16m

---

Exercise (Video): Setting up a Server using json-server 6m

---

Promises 10m

---

Fetch 20m

---

Exercise (Video): Fetch from Server 33m

---

Exercise (Video): Fetch Handling Errors 15m

---

Exercise (Video): Fetch Post Comment 18m

---

React Animations 13m

---

Exercise (Video): React Animations 9m

---

Exercise (Video): React Animation Components 9m

---

Assignment 4: Redux, Client-Server Communication and Fetch 4m

---

Introduction to Webpack 7m

---



Exercise (Video): Building and Deploying the React Application 11m

---

**26 readings**

---

Redux Actions: Objectives and Outcomes 10m

---

Exercise (Instructions): Combining Reducers 10m

---

Exercise (Instructions): Redux Actions 10m

---

Redux Actions: Additional Resources 10m

---

Redux Thunk: Objectives and Outcomes 10m

---

Exercise (Instructions): Redux Thunk 10m

---

Exercise (Instructions): React-Redux-Form Revisited 10m

---

Redux Thunk: Additional Resources 10m

---

Client-Server Communication: Objectives and Outcomes 10m

---

Exercise (Instructions): Setting up a Server using json-server 10m

---

Client-Server Communication: Additional Resources 10m

---

Fetch: Objectives and Outcomes 10m

---

Exercise (Instructions): Fetch from Server 10m

---

Exercise (Instructions): Fetch Handling Errors 10m

---

Exercise (Instructions): Fetch Post Comment 10m

---

Fetch: Additional Resources 10m

---

React Animations: Objectives and Outcomes 10m

---

Exercise (Instructions): React Animations 10m

---

Exercise (Instructions): React Animation Components 10m

---

React Animations: Additional Resources 10m

---

Assignment 4: Redux, Client-Server Communication and Fetch: Additional Resources 10m

---

Building and Deployment: Objectives and Outcomes 10m

---

Exercise (Instructions): Building and Deploying the React Application 10m

---

Building and Deployment: Additional Resources 10m

---

Project Implementation: Objectives and Outcomes 10m

---

Final Report Template 10m

---

## Exercise (Instructions): Setting up Git

### Objectives and Outcomes

In this exercise you will learn to install Git on your computer. Git is required for using all the remaining Node.js and Node based tools that we encounter in the rest of the course. At the end of this exercise, you would be able to:

- Install Git on your computer

- Ensure that Git can be used from the command-line or command-prompt on your computer
- Set up some of the basic global configuration for Git

## Downloading and Installing Git

- To install Git on your computer, go to <https://git-scm.com/downloads> to download the Git installer for your specific computing platform.
- Then, follow the installation steps as you install Git using the installer.
- You can find more details about installing Git at <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>. This document lists several ways of installing Git on various platforms.
- Installing some of the GUI tools like GitHub Desktop will also install Git on your computer.
- On a Mac, setting up XCode command-line tools also will set up Git on your computer.
- You can choose any of the methods that is most convenient for you.

## Some Global Configuration for Git

- Open a cmd window or terminal on your computer.
- Check to make sure that Git is installed and available on the command line, by typing the following at the command prompt:

```
git -version
```

- To configure your user name to be used by Git, type the following at the prompt:

```
git config --global user.name "Your Name"
```

- To configure your email to be used by Git, type the following at the prompt:

```
git config --global user.email <your email address>
```

- You can check your default Git global configuration, you can type the following at the prompt:

```
git config --list
```

## Conclusions

At the end of this exercise you should have Git available on the command-line of your computer.

Exercise (Video): Basic Git Commands

## Exercise (Instructions): Basic Git Commands

## Objectives and Outcomes

In this exercise you will get familiar with some basic Git commands. At the end of this exercise you will be able to:

- Set up a folder as a Git repository
- Perform basic Git operations on your Git repository

## Basic Git Commands

- At a convenient location on your computer, create a folder named **git-test**.
- Open this git-test folder in your favorite editor.
- Add a file named *index.html* to this folder, and add the following HTML code to this file:

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <h1>This is a Header</h1>
  </body>
</html>
```

## Initializing the folder as a Git repository

- Go to the git-test folder in your cmd window/terminal and type the following at the prompt to initialize the folder as a Git repository:

```
git init
```

## Checking your Git repository status

- Type the following at the prompt to check your Git repository's status:

```
git status
```

## Adding files to the staging area

- To add files to the staging area of your Git repository, type:

```
git add .
```

## Committing to the Git repository

- To commit the current staging area to your Git repository, type:

```
git commit -m "first commit"
```

## Checking the log of Git commits

- To check the log of the commits to your Git repository, type

```
git log --oneline
```

- Now, modify the *index.html* file as follows:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head></head>
```

```
<body>
```

```
<h1>This is a Header</h1>
```

```
<p>This is a paragraph</p>
```

```
</body>
```

```
</html>
```

- Add a sub-folder named **templates** to your **git-test** folder, and then add a file named *test.html* to the templates folder. Then set the contents of this file to be the same as the *index.html* file above.
- Then check the status and add all the files to the staging area.
- Then do the second commit to your repository
- Now, modify the *index.html* file as follows:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head></head>
```

```
<body>
```

```
<h1>This is a Header</h1>
```

```
<p>This is a paragraph</p>
```

```
<p>This is a second paragraph</p>
```

```
</body>
```

```
</html>
```

- Now add the modified *index.html* file to the staging area and then do a third commit.

## Checking out a file from an earlier commit

- To check out the index.html from the second commit, find the number of the second commit using the git log, and then type the following at the prompt:

```
git checkout <second commit's number> index.html
```

## Resetting the Git repository

- To discard the effect of the previous operation and restore index.html to its state at the end of the third commit, type:

```
git reset HEAD index.html
```

- Then type the following at the prompt:

```
git checkout -- index.html
```

- You can also use *git reset* to reset the staging area to the last commit without disturbing the working directory.

## Conclusions

At the end of this exercise you should have learnt some basic Git commands. Experiment with these commands until you fully understand how to use Git.

## Online Git Repositories

### Exercise (Instructions): Online Git Repositories

#### Objectives and Outcomes

In this exercise you will learn about how to set up and use an online Git repository and synchronize your local Git repository with your online repository. At the end of this exercise, you will be able to:

- Set up the online repository as a remote repository for your local Git repository
- Push your commits to the online repository
- Clone an online Git repository to your computer

#### Setting up an Online Git repository

- Sign up for an account either at Bitbucket (<https://bitbucket.org>) or GitHub (<https://github.com>).
- Then set up an online Git repository named **git-test**. Note the URL of your online Git repository. Note that private repositories on GitHub requires a paid account, and is not available for free accounts.

#### Set the local Git repository to set its remote origin

- At the prompt, type the following to set up your local repository to link to your online Git repository:

```
git remote add origin <repository URL>
```

## Pushing your commits to the online repository

- At the prompt, type the following to push the commits to the online repository:

```
git push -u origin master
```

## Cloning an online repository

- To clone an online repository to your computer, type the following at the prompt:

```
git clone <repository URL>
```

## Conclusions

In this exercise you have learnt to set up an online Git repository, synchronize your local repository with the remote repository, and clone an online repository.

## Node.js and NPM

JavaScript which was designed as a scripting language for the browser, has seen deployment far beyond the browser.

Node.js has played a significant role in this shift of JavaScript from the browser to the desktop.

Let's now learn a little bit about what node.js is and what role does NPM, the Node Package Manager, play in the context of node.js.

Node.js as I mentioned earlier, allows us to bring the power of JavaScript to the desktop.

Node.js is based on the JavaScript runtime engine that has been built for the Chrome browser.

The Chrome V8 JavaScript engine has been ported from the browser to run on the desktop and support the execution of JavaScript programs on the desktop.

Node.js is built around an event driven non blocking I/O model which makes it very efficient to run JavaScript programs on the desktop and synchronous Javascript on the desktop.

Now, this is where node finds its true pouch.

Right now, we will examine Node.js In the context of its use as a JavaScript runtime.

We'll look at the server-side application of Node.js in detail in the last course of this specialization.

This is the typical architecture of Node.js.

In this, the Chrome V8 engine is at the bottom layer together with libuv forms, the layer that interacts with the underlying computer system to support the execution of JavaScript programs.

On top of it, we have Node Bindings which are also implemented in C++.

At the top layer, you have the Node.js and standard library which are all implemented in JavaScript, and this is what enables us to write JavaScript programs and run them on the desktop.

Naturally, the ability to run JavaScript programs on the desktop energize the web development community to explore using JavaScript to develop a significant number of web development tools.

Tools such as Bower, Grunt, Gulp, Yeoman, and many others.

We will explore some of these in the later part of this course and in subsequent courses.

The last course in the specialization, as I mentioned, looks at the use of Node.js on the server side.

How we can develop Web server, Business logic, all implemented in JavaScript on the server site.

Together with Node, you often hear people talking about the Node Package Manager or NPM.

When you install Node on your computer, NPM automatically gets installed.

The Node Package Manager is the manager for the node ecosystem that manages all the node modules and packages that have been made publicly available by many different users.

A typical node package consist of JavaScript files together with a file called package.json which is the manifest file for this node module.

We will look at how we can use the package.json file in more detail in the subsequent exercises.

## **Exercise (Video): Setting up Node.js and NPM**

## **Exercise (Instructions): Setting up Node.js and NPM**



**Note: Make sure you have installed Git on your machine before you install Node.js. Please complete the previous Git installation exercise before proceeding with this exercise.**

## Objectives and Outcomes

In this exercise, you will learn to set up the Node.js environment, a popular Javascript based server framework, and node package manager (NPM) on your machine.

To learn more about NodeJS, you can visit <https://nodejs.org>.

For this course, you just need to install Node.js on your machine and make use of it for running some front-end tools.

You will learn more about the server-side support using Node.js in a subsequent course.

At the end of this exercise, you will be able to:

- Complete the set up of Node.js and NPM on your machine
- Verify that the installation was successful and your machine is ready for using Node.js and NPM.

## Installing Node

- To install Node on your machine, go to <https://nodejs.org> and click on the Download button. Depending on your computer's platform (Windows, MacOS or Linux), the appropriate installation package is downloaded.
- As an example, on a Mac, you will see the following web page. Click on the Download button. Follow along the instructions to install Node on your machine. (Note: Now Node gives you the option of installing a mature and dependable LTS version and a more newer stable version. You should to install the LTS version. I will use this version in the course.)

**Note: On Windows machines, you may need to configure your PATH environmental variable in case you forgot to turn on the add to PATH during the installation steps.**

## Verifying the Node Installation

- Open a terminal window on your machine. If you are using a Windows machine, open a cmd window or PowerShell window with **admin** privileges.
- To ensure that your NodeJS setup is working correctly, type the following at the command prompt to check for the version of **Node** and **NPM**

Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#). Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, [npm](#), is the largest ecosystem of open source libraries in the world.

[Important March 2018 security upgrades now available](#)

## Download for macOS (x64)

**8.11.1 LTS**

Recommended For Most Users

**9.10.1 Current**

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#) [Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [LTS schedule](#).

Sign up for [Node.js Everywhere](#), the official Node.js Weekly Newsletter.

 **LINUX FOUNDATION** [COLLABORATIVE PROJECTS](#)

[Report Node.js issue](#) | [Report website issue](#) | [Get Help](#)

© 2018 Node.js Foundation. All Rights Reserved. Portions of this site originally © 2018 Joyent.

Node.js is a trademark of Joyent, Inc. and is used with its permission. Please review the [Trademark Guidelines](#) of the Node.js Foundation.

Linux Foundation is a registered trademark of The Linux Foundation.

Linux is a registered trademark of Linus Torvalds.

[Node.js Project Licensing Information](#).

`npm -v`

## Conclusions

At the end of this exercise, your machine is now ready with the Node installed for further development. We will examine web development tools next.

## Exercise (Instructions): Basics of Node.js and NPM

### Objectives and Outcomes

In this exercise you will learn the basics of Node and NPM. At the end of this exercise, you will be able to:

- Set up package.json file in the project folder for configuring your Node and NPM for this project
- Install a NPM module and make use of it within your project

### Initializing package.json

- At the command prompt in your **git-test** folder, type

```
npm init
```

- Follow along the prompts and answer the questions as follows: accept the default values for most of the entries, except set the entry point to index.html
- This should create a *package.json* file in your **git-test** folder.

### Installing an NPM Module

- Install an NPM module, **lite-server**, that allows you to run a Node.js based development web server and serve up your project files. To do this, type the following at the prompt:

```
npm install lite-server --save-dev
```

- You can check out more documentation on **lite-server** [here](#).
- Next, open package.json in your editor and modify it as shown below. Note the addition of two lines, line 7 and line 9.

```
"start": "npm run lite",
```

```
"test": "echo \"Error: no test specified\" && exit 1",
```

```
"lite": "lite-server"
```

```
},
```

```
"repository": {
```

```
"type": "git",
```

```
"url": "git+https://jogesh k muppala@bitbucket.org/jogesh k muppala/git-test.git"
```

```
},
```

```
"author": "",
```

```
"license": "ISC",
```

```
"homepage": "https://bitbucket.org/jogesh_k_muppala/git-test#readme",
```

```
"devDependencies": {
```

```
  "lite-server": "^2.2.2"
```

```
}
```

```
}
```

- Next, start the development server by typing the following at the prompt:

```
npm start
```

- This should open your *index.html* page in your default browser.
- If you now open the *index.html* page in an editor and make changes and save, the browser should immediately refresh to reflect the changes.

## Setting up .gitignore

- Next, create a file in your project directory named *.gitignore* (Note: the name starts with a period) Then, add the following to the *.gitignore* file

```
node_modules
```

- Then do a git commit and push the changes to the online repository. You will note that the *node\_modules* folder will not be added to the commit, and will not be uploaded to the repository.

## Conclusions

In this exercise you learnt to set up *package.json*, install a npm package and start a development server.

## Setting up your Development Environment: Git and Node: Additional Resources

### PDFs of Presentations

Git.pdf

Git-Exercises.pdf

NodeJS.pdf

Exercises-Node-NPM.pdf

### Additional Resources (Git)

- Git site <http://git-scm.com>.

- [Installing Git](#) chapter from Pro Git
- [Git reference manual](#)
- Quick reference guides: [GitHub Cheat Sheet](#) (PDF) | [Visual Git Cheat Sheet](#) (SVG | PNG)
- [Atlassian comprehensive Git tutorial](#)

## Additional Resources (Node.js and NPM)

- [Nodejs.org](#)
- [Npmjs.com](#)
- [Node API Documentation](#)
- [NPM Documentation](#)
- [lite-server](#)

## Introduction to React: Objectives and Outcomes

In this lesson you will be given a quick overview of JavaScript frameworks and libraries and then introduced to React in particular. We will learn some basics of React and how to configure a React application using the `create-react-app` the command line tool. At the end of this lesson, you will be able to:

- Get a basic overview of JavaScript frameworks and libraries
- Understand the architecture of an React application
- Scaffold out a starter React application using *create-react-app*, the command line tool

## Front-end JavaScript Frameworks and Libraries Overview

### Introduction to React

We ended the previous lecture with a question, is React a library or is it a framework and what exactly is React?

Let's examine these questions in a bit more detail in this lecture.

Let's start out with the very first question.

What exactly is React and how is it different from the other frameworks or libraries that we have mentioned in the previous lecture?

When you visit React website, you see it clearly specified right on the front page that React is a JavaScript library for building user interfaces.

Now again, depending on who you ask some people tend to call React a library and others tend to call it a framework.

Now, let's not bother ourselves too much in splitting our hairs over whether it is a framework, or a library but let's concentrate more on what it actually helps us accomplish.

It is more important for us to understand that rather than worrying about whether it is a framework or a library.

The React approach to implementing them applications is what we are after in this course.

So, React also states that it uses a declarative approach.

Now that leaves you in a confused state because we saw that frameworks generally tend to use the declarative approach.

But in React, the declarative approach used by React as specified on its website says that, it makes it easy to create interactive UIs with simple views for each state within your application.

And also React takes care of automatically updating the UI and then rendering any changes to their specific components as required on your page.

You just heard me mentioning the term Component.

React Indeed is a component based approach.

In a Component based approach, we encapsulate behaviors into small units called Components.

We will examine Components in more detail in the next lesson.

There it will become more clear to you how, and why a component based approach is useful for implementing our Web applications in React.

Furthermore, React makes no assumptions about the entire technology stack that you're going to use for implementing your Web applications.

React plays well with any technology stack that you can use behind the seats.

React itself concentrates only on the user interface side of the story, and that leaves it up to the application designer to decide how they want to implement the architecture and how they want your application to interact with the back-end server.

So, as we go through this course, we will examine one approach that we use for implementing the entire technology stack which includes the Flux architecture approach, and in specifically the use of Redux for implementing a state based storage for our Web application and also the use of Fetch for interacting with our back-end server.

Again, I've mentioned a few terms like Flux, Redux, and Fetch.

We will examine these towards the second half of this course.

It's obviously useful to examine the history behind the React approach.

So, to understand where React originated, and how it came about to the state it is today.

React was first designed by Jordan Walke who was part of the Facebook team.

It was first deployed for Facebook's news feed around 2011.

Subsequently in 2013, React was open sourced at this JS conference.

React took off as an approach for implementing Web applications from then onwards.

React is designed for speed, speed of implementing the application, simplicity, and scalability.

The three essence of React, and why it has become so popular in the real value.

As we examine React more in this course, you'll become more and more familiar with why this approach is very suited for implementing Web applications.

As you enter the React world you will be bombarded with a lot of vocabulary that is used in the React world.

You will hear people talking about One-way data flow especially in the context of the Flux Architecture.

You will often hear people mentioning about JSX, we will examine JSX in the very next lecture and understand what role it plays in developing a React application.

We'll hear about Components which we will examine in the next lesson and also in the second module of this course in more detail and we'll here about the state and how a React Component interacts with the state of your application and the way you store the state of your application, or do you store the state in a specific component.

We'll also hear about Props, a way of passing data between the various components.

Also, we'll hear about Virtual Dom, and how is it different from Real DOM.

Why React manipulate the Virtual DOM, and how the Virtual DOM eventually gets incorporated, or rendered onto the Real DOM.

And Element, the React Element, which is the smallest unit of building up a React application.

A component being a collection of React elements.

Then we hear about the Flux and Redux architectures in a bit more detail.

Again, as we go along this course we will examine these concepts and these terminology in more detail.

Again, don't get overwhelmed with the vocabulary that you hear in the React world.

If you learn step by step, you'll begin to pretty soon get a very good handle on all this vocabulary and you can easily go ahead and impress people by throwing these words at them, and trying to impress them or how much you know about React.

So, this is the jargon that you will end up learning also at the end of this course. Enough of the jargon.

Let's go ahead and get our hands dirty by starting to build a full fledged React application, which will form part of all the exercises as you go through the rest of this course.

We will start with our first exercise where we'll install the `create-react-app`, which we will use to scaffold out our very first React application in the first exercise, and then we will start building upon this application throughout the remaining exercises of this course.

## **Exercise (Video): Getting Started with React**

Now that we have examined React briefly in the previous two lectures, I'm sure you're curious about getting started with React, getting your hands dirty with starting out on a React application.

So, in this exercise, we will look at how we will get started with React. I'm sure by now your computer is already configured with Node and you have access to NPM, the package manager that comes with Node.

## **Exercise (Instructions): Getting Started with React**

### **Objectives and Outcomes**

In this first React exercise, you will first install `create-react-app`, the command line tool for scaffolding React applications. You will then use the tool to scaffold out a basic React application. We will thereafter develop this application into a full-fledged React application in the process of doing the exercises in this course. At the end of this exercise you will be able to:

- Install `create-react-app`
- Scaffold out a basic React application

### **Installing Yarn**



- Yarn is another package manager like NPM, but is better suited and faster to work with for React applications. So let us install yarn and use it for building our React applications.
- To install Yarn, you can find the instructions for your specific platform at <https://yarnpkg.com/en/docs/install>.
- If you choose not to install Yarn, you can continue to use npm in place of yarn without any problem.

## Installing *create-react-app*

From the React documentation we learn that the *create-react-app* CLI makes it easy to create an application that already works, right out of the box. It already follows the best practices suggested by the React community!

- To install *create-react-app* globally, type the following at the prompt:

```
yarn global add create-react-app@1.5.2
```

Use *sudo* on a Mac and Linux. Alternately you can use npm, by typing "npm install -g create-react-app@1.5.2".

- This will make the command line tool for creating React applications. To learn more about the various commands that this CLI provides, type at the prompt:

```
create-react-app --help
```

## Generating and Serving a React Project using *create-react-app*

- At a convenient location on your computer, create a folder named *React* and move into that folder.
- Then type the following at the prompt to create a new React application named *confusion*:

```
create-react-app confusion
```

- This should create a new folder named *confusion* within your *React* folder and create the React application in that folder.
- Move to the *confusion* folder and type the following at the prompt:

```
yarn start
```

- This will compile the project and then open a tab in your default browser at the address <http://<Your Computer's Name>:3000>.
- You can initialize your project to be a Git repository by typing the following commands at the prompt:

```
git commit -m "Initial Setup"
```

```
git init
```

```
git add .
```

- Thereafter you can set up an online Git repository and synchronize your project to the online repository. Make sure that the online Git repository is a *private* repository.

## Conclusions

In this exercise you installed the create-react-app CLI tool and created a basic React project and served up the compiled project to your browser.

## React App Overview

### Introduction to JSX

### Exercise (Video): Configuring your React Application

### Exercise (Instructions): Configuring your React Application

#### Objectives and Outcomes

In this exercise we will set up our project to use Reactstrap (a package supporting easy to use React based Bootstrap 4 components). We will then introduce our first reactstrap component into our application. At the end of this exercise you will be able to:

- Configure your React project to use reactstrap.
- Start using reactstrap components in your application.

#### Configure your React Project to use Reactstrap

- To configure your project to use reactstrap, type the following at the prompt to install reactstrap, and Bootstrap 4:

```
yarn add bootstrap@4.0.0
```

```
yarn add reactstrap@5.0.0
```

```
yarn add react-popper@0.9.2
```

**Note:** You can also install the same using npm using the "npm install <package> --save" option if you are using npm instead of yarn.

#### Configure to use Bootstrap 4

- Next, open index.js file in the src folder and add the following line into the imports:

```

. . .
import 'bootstrap/dist/css/bootstrap.min.css';
. . .

```

## Adding a Navigation Bar:

- Open App.js in the src folder and update it as follows:

```

|
class App extends Component {
  render() {
    return (
      <div className="App">
        <Navbar dark color="primary">
          <div className="container">
            <NavbarBrand href="/">Ristorante Con Fusion</NavbarBrand>
          </div>
        </Navbar>
      </div>
    );
  }
}
|
. . .
import { Navbar, NavbarBrand } from 'reactstrap';
|
. . .

```

- Do a Git commit with the message "Configuring React"

## Conclusions

In this exercise we learnt to configure our React application to use Reactstrap.

## Introduction to React: Additional Resources

## PDFs of Presentations

# 1-JavaScript-Frameworks.pdf

PDF File

2-Intro-React.pdf

PDF File

3-React-App-Overview.pdf

PDF File

4-Intro-JSX.pdf

PDF File

## React Resources

- [Reactjs.org](https://reactjs.org)
- [create-react-app](https://create-react-app.dev)
- [reactstrap](https://reactstrap.net)
- [reactstrap Navbar](#)
- [Introducing JSX](#)
- [Convert JSX using Online Babel Compiler](#)

## Definitions

- Framework
- Hollywood Principle
- Inversion of Control
- Imperative vs Declarative Programming
- Imperative vs Declarative

## Blog Articles

- [5 Best JavaScript Frameworks in 2017](#)
- [Top JavaScript Frameworks & Topics to Learn in 2017](#)
- [Declarative vs. Imperative Programming for the Web](#)

- [Is React library or a framework?](#)
- [Is React a library or a framework and why?](#)
- [An Introduction to the React Framework](#)
- [React is a framework](#)
- [Why isn't React called framework? What does it lack to be a framework?](#)

## React Components: Objectives and Outcomes

In this lesson you will learn about React components and how we construct an React component and design its views. At the end of this lesson you will be able to:

- Create a React component
- Construct the React component code and the view for your component using JSX and JavaScript

## React Components

### Exercise (Video): React Components Part 1

### Exercise (Instructions): React Components Part 1

#### Exercise Resources

images

ZIP File

#### Objectives and Outcomes

In this exercise you will add the first component to your React application and update its view using JSX. At the end of this exercise you will be able to:

- Add components to your React application
- Use JSX to define the views of your component.

#### Adding a Menu Component

- First, download the *images.zip* file provided above and then unzip the file. Create a folder named *assets* in the *public* folder. Move the resulting *images* folder containing some PNG files to the React project's *public/assets* folder. These image files will be useful for our exercises.

- Next, add a new folder named *components* in the *src* folder, and create a new file named *MenuComponent.js* in this folder.
- Add the following code to *MenuComponent.js*:

```

    this.state = {
      dishes: [
        {
          id: 0,
          name: 'Uthappizza',
          image: 'assets/images/uthappizza.png',
          category: 'mains',
          label: 'Hot',
          price: '4.99',
          description: 'A unique combination of Indian Uthappam (pancake) and Italian pizza, topped with Cerignola olives, ripe vine cherry tomatoes, Vidalia onion, Guntur chillies and Buffalo Paneer.'
        },
        {
          id: 1,
          name: 'Zucchipakoda',
          image: 'assets/images/zucchipakoda.png',
          category: 'appetizer',
          label: '',
          price: '1.99',
          description: 'Deep fried Zucchini coated with mildly spiced Chickpea flour batter accompanied with a sweet-tangy tamarind sauce'
        },
        {
          id: 2,
          name: 'Vadonut',
          image: 'assets/images/vadonut.png',
          category: 'appetizer',
          label: 'New',
          price: '1.99',

```

```

        description: 'A quintessential ConFusion experience, is it a vad
a or is it a donut?'
      },
    {
      id: 3,
      name: 'ElaiCheese Cake',
      image: 'assets/images/elaicheesecake.png',
      category: 'dessert',
      label: '',
      price: '2.99',
      description: 'A delectable, semi-
sweet New York Style Cheese Cake, with Graham cracker crust and spiced with India
n cardamoms'
    }
  ],
}

class Menu extends Component {
  constructor(props) {
    super(props);
  }

  import React, { Component } from 'react';
  import { Media } from 'reactstrap';
}

```

- Next, open *App.js* file and update it as follows:

```

...

import Menu from './components/MenuComponent';

...

<Menu />

...

```

- Open *App.css* file and delete all its contents.
- Save all changes and do a Git commit with the message "Components Part 1".

## Conclusions

In this exercise we added a new component to our React application, added data to its class, and then updated the app to show the information in the web page.

## React Components: State and Props

### Exercise (Video): React Components Part 2

### Exercise (Instructions): React Components Part 2

#### Objectives and Outcomes

In this exercise we will continue modifying the menu component from the previous exercise. Instead of a list, we will use a Card component from reactstrap to display the menu in a different way. Also we will use the Card component to display the details of a selected dish.

At the end of this exercise you will be able to:

- Make use of the Card component to display a list of items.
- Use the Card component to display detailed information.

#### Exercise Resources

dishes

JS File

#### Updating the Menu Component

- Open *MenuComponent.js* and update its contents as follows. Note that we have removed the dishes variable from the state of the component, and updated it to use the Card:

```
constructor(props) {  
  super(props);  
  
  this.state = {  
    selectedDish: null  
  }  
}  
  
onDishSelect(dish) {
```



```

        this.setState({ selectedDish: dish});
    }
}

renderDish(dish) {
    if (dish !== null)
        return(
            <Card>
                <CardImg top src={dish.image} alt={dish.name} />
                <CardBody>
                    <CardTitle>{dish.name}</CardTitle>
                    <CardText>{dish.description}</CardText>
                </CardBody>
            </Card>
        );
    else
        return(
            <div></div>
        );
}

render() {
    const menu = this.props.dishes.map((dish) => {
        return (
            <div className="col-12 col-md-5 m-1">
class Menu extends Component {
}

import { Card, CardImg, CardImgOverlay, CardText, CardBody,
    CardTitle } from 'reactstrap';
...

```

- Add a folder named *shared* under the *src* folder.
- In the *shared* folder, create a new file named *dishes.js* and add the following content to it (Note: Alternately you can download the *dishes.js*

file given above in the Exercise Resources and move it to the shared folder. Make sure the file is named *dishes.js*):

```
export const DISHES =  
[  
  {  
    id: 0,  
    name: 'Uthappizza',  
    image: 'assets/images/uthappizza.png',  
    category: 'mains',  
    label: 'Hot',  
    price: '4.99',  
    description: 'A unique combination of Indian Uthappam (pancake) and Italian pizza, topped with Cerignola olives, ripe vine cherry tomatoes, Vidalia onion, Guntur chillies and Buffalo Paneer.',  
    comments: [  
      {  
        id: 0,  
        rating: 5,  
        comment: "Imagine all the eatables, living in conFusion!",  
        author: "John Lemon",  
        date: "2012-10-16T17:57:28.556094Z"  
      },  
      {  
        id: 1,  
        rating: 4,  
        comment: "Sends anyone to heaven, I wish I could get my mother-in-law to eat it!",  
        author: "Paul McVites",  
        date: "2014-09-05T17:57:28.556094Z"  
      },  
      {  
        id: 2,  
        rating: 3,  
        comment: "Eat it, just eat it!",  
        author: "Michael Jaikishan",
```

```

      date: "2015-02-13T17:57:28.556094Z"
    },
    {
      id: 3,
      rating: 4,
      comment: "Ultimate, Reaching for the stars!",
      author: "Ringo Starry",
      date: "2013-12-02T17:57:28.556094Z"
    },
    {

```

- Open *App.js* and update it as follows:

```

. . .

```

```

import { DISHES } from './shared/dishes';

```

```

. . .

```

```

class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      dishes: DISHES
    };
  }
}

```

```

. . .

```

```

<Menu dishes={this.state.dishes} />

```

```

. . .

```

- Save the changes and do a Git commit with the message "Components Part 2".

## Conclusions

In this exercise we used a list of Cards to display the information in the menu component. Also, we used a card to display the details of a selected dish.

## React Components: Lifecycle Methods Part 1

## React Components: Additional Resources

### PDFs of Presentations

5-Component-Part1.pdf

PDF File

6-Component-Part2.pdf

PDF File

7-Lifecycle-Methods.pdf

PDF File

### React Resources

- [React Components](#)
- [React Component State](#)
- [React Component Props](#)
- [reactstrap Media Object](#)
- [React.Component and Lifecycle Methods](#)
- [reactstrap Card](#)
- [Bootstrap unstyled list](#)
- [React Dev Tools](#)
- [React Dev Tools Chrome Extension](#)

## Assignment 1 Requirements (Video): React Components

## Peer-graded Assignment: React Components

Deadline Oct 23, 11:59 PM PDT

It looks like this is your first peer-graded assignment. [Learn more](#)

Ready for the assignment?

You will find instructions below to submit.

In this assignment you will add a new component to the React application to show the details of a selected dish. You will use the Card component and the Bootstrap unstyled list component to prepare the view for this new component.

## Step-By-Step Assignment Instructions

[less](#)

### Objectives and Outcomes

In this assignment, you will continue to work with the React application that you have been developing in the exercises. You will add a new component named *DishdetailComponent* that will display the details of a selected dish. You will then design the view for the component using the card component. At the end of this assignment, you should have completed the following tasks:

- Created a new DishdetailComponent and added it to your React application.
- Updated the view of the DishdetailComponent to display the details of the selected dish using an reactstrap card component.
- Updated the view of the DishdetailComponent to display the list of comments about the dish using the Bootstrap unstyled list component.

### Assignment Requirements

This assignment requires you to complete the following tasks. Detailed instructions for each task are given below. The picture of the completed web page included below indicates the location within the web page that will be updated by the three tasks.

#### Task 1

In this task you will be adding a new *DishdetailComponent* to your React application and include the component into the menu component's view so that the details of a specific dish are displayed there:

- Replace the card showing the selected dish in MenuComponent's view with the DishdetailComponent, and make sure to pass the selected dish information as props to the DishdetailComponent.
- Create a new *DishDetail* class in a file named *DishdetailComponent.js* in the *components* folder
- Export the *DishDetail* class from this file so that it can be imported in *MenuComponent.js* and used to construct the view of the selected dish.

- Return a `<div>` from the `render()` function. This `<div>` should use the Bootstrap `row` class to position the content within the `<div>`. This div will display both the details of the dish in a Card and the list of comments side-by-side for medium to extra large screens, but will stack them for xs and sm screens.
- The card should be enclosed inside a `<div>` appropriate Bootstrap column classes so that it occupies the entire 12 columns for the xs and sm screen sizes, and 5 columns for md screens and above. Also apply a class of `m-1` to this div.
- The comments should be enclosed in a `<div>` to which you apply appropriate column classes so that it occupies the entire 12 columns for the xs and sm screen sizes, and 5 columns for md screens and above. Also apply a class of `m-1` to this div.
- If the dish is null then you should return an empty `<div>`

## Task 2

In this task you will be adding a card component to the *DishdetailComponent* view to display the details of the dish given above:

- Implement a function named `renderDish()` that takes the dish as a parameter and returns the JSX code for laying out the details of the dish in a reactstrap Card. You have already seen this as part of the *MenuComponent* class in the exercise earlier.
- Display the name of the dish as the Card title, and the description as the Card text.

## Task 3

In this task you will use the comments that are included in the dish object above to display a list of the comments for the dish. Please use your JavaScript knowledge to recall how you would access an inner property in a JavaScript object that itself points to an array of JavaScript objects (comments). This task involves the following steps:

- Implement a function named `renderComments()` that takes the comments array as a parameter and lays out each comment as shown in the image below. You should use the Bootstrap `list-unstyled` class on the list.
- Each comment should be displayed on two lines, the first one showing the comment, and the second line showing the comment author's name and the date.
- The comments should contain a `<h4>` header with the word "Comments".
- Remember to enclose the header and comments inside a `<div>` before returning the JSX code. Otherwise React will not do the layout correctly.

- If the comments are null, then you should return an empty <div>.

## Ristorante Con Fusion



Uthappizza



Zucchipakoda



Vadonut



Elai Cheese Cake



Vadonut

**Task 1 & 2**

A quintessential ConFusion experience, is it a vada or is it a donut?

### Comments

Imagine all the eatables, living in conFusion!

-- John Lemon , Oct 17, 2012

Sends anyone to heaven, I wish I could get my mother-in-law to eat it!

-- Paul McVites , Sep 06, 2014

Eat it, just eat it!

-- Michael Jaikishan , Feb 14, 2015

Ultimate, Reaching for the stars!

-- Ringo Starry , Dec 03, 2013

It's your birthday, we're gonna party!

-- 25 Cent , Dec 03, 2011

**Task 3**

## Restaurante Con Fusión

Chimichurri



Zucchini pakoda



Vadonut



ElaCheese Cake



Vadonut

A quintessential ConFusion experience, is it a vado or is it a donut?

### Comments

Imagine all the variables, living in confusio

— John Lammie , Oct 17, 2013

Sends anyone to heaven, I wish I could get my mother-in-law to eat it!

— Paul McViee , Sep 06, 2014

Eat it, just eat it!



## Review criteria

less

Upon completion of the assignment, your submission will be reviewed based on the following criteria:

### Task 1:

- A new ***DishdetailComponent*** has been added to your React application.
- Included the DishDetail into your MenuComponent's view to show the selected dish.
- Passing the selected dish as props to the DishDetail Component.
- Used the appropriate Bootstrap classes to the card so that it occupies the entire row for xs and sm screen sizes, and 5 columns for md screens and above.
- Used the appropriate Bootstrap classes to the div containing the list of comments so that it occupies the entire row for xs and sm screen sizes, and 5 columns for md screens and above.

### Task 2:

- Used the Card component to display the details of the dish.

### Task 3:

- Included a list of comments about the dish into the dishdetail view.

## Assignment 1: React Components: Additional Resources

### Assignment 1 Screenshots



Uthappizza



Zucchipakoda



Vadonut



Elai Cheese Cake



Vadonut

**Task 1 & 2**

A quintessential ConFusion experience, is it a vada or is it a donut?

### Comments

Imagine all the eatables, living in conFusion!

-- John Lemon , Oct 17, 2012

Sends anyone to heaven, I wish I could get my mother-in-law to eat it!

-- Paul McVites , Sep 08, 2014

Eat it, just eat it!

-- Michael Jaikishan , Feb 14, 2015

Ultimate, Reaching for the stars!

-- Ringo Starry , Dec 03, 2013

It's your birthday, we're gonna party!

-- 25 Cent , Dec 03, 2011

**Task 3**

## Restaurante Con Fusión

Chimichurri



Zucchini pakoda



Vadonut



ElaCheese Cake



Vadonut

A quintessential ConFusion experience, is it a vado or is it a donut?

### Comments

Imagine all the variables, living in confusio

— John Lammie, Oct 17, 2013

Sends anyone to heaven, I wish I could get my mother-in-law to eat it!

— Paul McViee, Sep 06, 2014

Eat it, just eat it!

## React Resources

- [React Components](#)
- [React Component State](#)
- [React Component Props](#)
- [reactstrap Media Object](#)
- [reactstrap Card](#)
- [Bootstrap unstyled list](#)
- [Bootstrap Grid](#)

## Module 5: Ideation: Objectives and Outcomes

The first step in your journey towards the implementation of the Capstone project begins with an idea. In this module you will develop the idea for your project, the set of expected features, survey the market to look at similar ideas to enable you to differentiate your project from others, while at the same time drawing inspiration from them. You are required to submit a formal ideation report following the structure given in the template. This will enable your peers to provide you feedback and suggestions for your project.

Before you get started on a project, the first step is to develop the idea for the project. In this module you will explore how you develop your idea and come up with possible set of features for your project. At the end of this step you should be able to:

- Clearly express the central idea of your project, and identify the problem being addressed
- Delineate a set of features that you expect your website and app should support
- Identify other projects that might have similar features and would act as exemplars for your project

## Ideation Report Template

### Project Title

#### 1. Introduction

- A brief introduction to your website idea. State the goals of the project.
- The values / benefits (tangible and intangible) this application can bring to a company/organization/end-user.

#### 2. Expected List of Features

- A brief list of features that you expect your website to support.
- Brief justifications for including these features.

### 3. Market Survey

- Do a survey of the Web to find about five web sites that might have similar ideas as yours.
- Briefly compare the features of these applications with your application idea.

### 4. References

- Give references to any material / websites / books etc. relevant to your application idea
- Give the links to the websites relevant to your idea, that you listed in the section above.

## Honors Peer-graded Assignment: Ideation

Deadline Oct 23, 11:59 PM PDT

Ready for the assignment?

You will find instructions below to submit.

In this assignment you will be submitting a written report describing the general idea of your project, the expected list of features and a survey of existing projects, websites and/or apps that are similar to your ideas and/or have some features similar to your proposed project. The structure of the written report should adhere to the report template given in this module, and emphasize the points specified in the template. The written submission needs to be no more than three standard Letter/A4 sized pages.

### Review criteria

less

Your submission will be reviewed based on the following criteria by peers in order to provide you with constructive feedback on your project idea:

1. Does the Ideation report clearly state the idea of the project and the primary aim and purpose of the proposed website ?
2. Does the Ideation report list the expected features that will be supported by the website?

3. Did the user provide a survey of related ideas/projects/websites that have some similarities to the proposed idea?
4. Does the Ideation report provide references to suitable sources in support of the project idea?

## **Ideation: Additional Resources**

### **General Resources**

- [Ideation \(creative process\)](#)

### **Volunteer your Services**

- [VolunteerMatch.org](#)
- [Free Code Camp](#)

## **Module 6: React Component Types: Objectives and Outcomes**

In this lesson you will learn about various types of React components: Presentational, Container and Functional components. At the end of this lesson you will be able to:

- Identify the salient features and uses for the various types of components
- Create presentational, container and functional components in your React application

## **Presentational and Container Components**

Notes

### **Exercise (Video): Presentational and Container Components**

### **Exercise (Instructions): Presentational and Container Components**

Objectives and Outcomes

In this exercise we understand about how presentational components deal with the look and feel of the app and container components deal with the data and behavior. At the end of this exercise you will learn about:

- Organizing your React app into presentational and container components
- Enable your presentational components to be concerned with the look and feel of your app
- Enable container components to deal with the state, provide the data and handle user interactions.

## Add a Container Component

- Add a new component named *MainComponent.js* in the components folder and update its contents as follows:

```
import React, { Component } from 'react';
import { Navbar, NavbarBrand } from 'reactstrap';
import Menu from './MenuComponent';
import DishDetail from './DishdetailComponent';
import { DISHES } from '../shared/dishes';

class Main extends Component {

  constructor(props) {
    super(props);
    this.state = {
      dishes: DISHES,
      selectedDish: null
    };
  }

  onDishSelect(dishId) {
    this.setState({ selectedDish: dishId});
  }

  render() {
```

```

    return (
      <div>
        <Navbar dark color="primary">
          <div className="container">
            <NavbarBrand href="/">Ristorante Con Fusion</NavbarBrand>
          </div>
        </Navbar>
        <Menu dishes={this.state.dishes} onClick={(dishId) => this.onDishSelect(dishId)} />
        <DishDetail dish={this.state.dishes.filter((dish) => dish.id === this.state.selectedDish)[0]} />
      </div>
    );
  }
}

export default Main;

```

- Update the *App.js* by removing the state related information, and make use of Main Component to render the UI:

```

...
import Main from './components/MainComponent';

class App extends Component {
  render() {
    return (
      <div className="App">
        <Main />
      </div>
    );
  }
}

```



```
}
```

```
|
```

```
. . .
```

## Turn Menu Component into a Presentational Component

- Open MenuComponent.js and update its contents by removing the state and removing the DishdetailComponent reference, and make use of the onClick supplied by MainComponent through the props to deal with the clicking of a menu item:

```
. . .
```

```
|
```

```
<Card key={dish.id}
```

```
onClick={() => this.props.onClick(dish.id)}>
```

```
. . .
```

- The DishdetailComponent is already structured as a presentational component and hence needs no further update, except wrapping the return value from render() within a <div> with the className as container.
- To print out the date for a comment in a format more suitable for human readability, you can update your renderComment function with the code snippet shown below:

```
{new Intl.DateTimeFormat('en-US', { year: 'numeric', month: 'short', day: '2-digit'}).format(new Date(Date.parse(comment.date)))}
```

- Save all the changes and do a Git commit with the message "Presentational and Container Components"

## Conclusions

In this exercise you learnt how to structure your app into presentational and container components.

## React Components: Lifecycle Methods Part 2

## Functional Components

## Exercise (Video): Functional Components

# Exercise (Instructions): Functional Components

## Objectives and Outcomes

In this lesson we explore the design of functional components in React. We will reimplement both the MenuComponent and DishdetailComponent as pure functional components. At the end of this exercise you will be able to:

- Implement functional components in React
- Illustrate the reimplementation of presentational components as pure functional components

## Implementing Functional Components

- Open MenuComponent.js and update it as follows:

```
import React from 'react';
import { Card, CardImg, CardImgOverlay,
  CardTitle } from 'reactstrap';

function RenderMenuItem ({dish, onClick}) {
  return (
    <Card
      onClick={() => onClick(dish.id)}>
      <CardImg width="100%" src={dish.image} alt={dish.name} />
      <CardImgOverlay>
        <CardTitle>{dish.name}</CardTitle>
      </CardImgOverlay>
    </Card>
  );
}

const Menu = (props) => {
  const menu = props.dishes.map((dish) => {
    return (
      <div className="col-12 col-md-5 m-1" key={dish.id}>
        <RenderMenuItem dish={dish} onClick={props.onClick} />
      </div>
    );
  });
  return <div>{menu}</div>;
}
```

```

        </div>
      );
    });
  }
  return (
    <div className="container">
      <div className="row">
        {menu}
      </div>
    </div>
  );
}

export default Menu;

```

- Then open DishdetailComponent.js and update it as follows:

```

import React from 'react';
import { Card, CardImg, CardText, CardBody,
  CardTitle } from 'reactstrap';


function RenderDish({dish}) {
  . . .
}

function RenderComments({comments}) {
  . . .
}

```

```
const DishDetail = (props) => {  
    
  . . .  
    
}  
  
export default DishDetail;
```

- Save all the changes and do a Git commit with the message "Functional Components".

## Conclusions

In this exercise we have learnt to implement our components as pure functional components.

## React Component Types: Additional Resources

### PDFs of Presentations

1-Component-Types-Part1.pdf

PDF File

2-Component-Types-Part2.pdf

PDF File

### Other Resources

- [Presentational and Container Components](#)
- [Presentational and Container Components \(Redux Perspective\)](#)
- [React Component Patterns](#)
- [Functional Stateless Components in React](#)

## Module 8: React Router: Objectives and Outcomes

In this lesson we cover the basics of React router. We examine how the router enables the navigation among views of various components that form part of a React application. At the end of this lesson you will be able to:

- Set up the router module to enable navigation among multiple component views
- Set up the routes to enable the navigation

# React Virtual DOM

## Exercise (Video): Header and Footer

## Exercise (Instructions): Header and Footer

### Objectives and Outcomes

In this exercise you will add in a header and a footer to our React application using two React components. This will illustrate the use of multiple components put together form the application's view. You will also add in the Font Awesome icons and Bootstrap-social for use within your application. At the end of this exercise you will be able to:

- Use multiple components and their views to put together the view of the application.
- Make use of Font Awesome icons and Bootstrap-social within your React application

### Using Font Awesome Icons and Bootstrap-Social

- First use yarn or npm to fetch Font Awesome and Bootstrap-social to the project by typing the following at the prompt:

```
yarn add font-awesome@4.7.0
```

```
yarn add bootstrap-social@5.1.1
```

- Then, open index.js file and update it as follows to enable your application to use Font Awesome and Bootstrap Social:

```
. . .
```

```
|
```

```
import 'font-awesome/css/font-awesome.css';
```

```
import 'bootstrap-social/bootstrap-social.css';
```

```
|
```

```
. . .
```

### Adding a Header and a Footer

- Create a new file named HeaderComponent.js and add the following to it:

```

import React, { Component } from 'react';
import { Navbar, NavbarBrand, Jumbotron } from 'reactstrap';

class Header extends Component {
  render() {
    return(
      <React.Fragment>
        <Navbar dark>
          <div className="container">
            <NavbarBrand href="/">Ristorante Con Fusion</NavbarBrand>
          </div>
        </Navbar>
        <Jumbotron>
          <div className="container">
            <div className="row row-header">
              <div className="col-12 col-sm-6">
                <h1>Ristorante con Fusion</h1>
                <p>We take inspiration from the World's best cuisines, and create a unique fusion experience. Our lipsmacking creations will tickle your culinary senses!</p>
              </div>
            </div>
          </div>
        </Jumbotron>
      </React.Fragment>
    );
  }
}

export default Header;

```

- Then, add another file named FooterComponent.js and add the following to it:

```

import React from 'react';

function Footer(props) {
  return(
    <div className="footer">
      <div className="container">
        <div className="row justify-content-center">
          <div className="col-4 offset-1 col-sm-2">
            <h5>Links</h5>
            <ul className="list-unstyled">
              <li><a href="#">Home</a></li>
              <li><a href="#">About</a></li>
              <li><a href="#">Menu</a></li>
              <li><a href="contactus.html">Contact</a></li>
            </ul>
          </div>
          <div className="col-7 col-sm-5">
            <h5>Our Address</h5>
            <address>
              121, Clear Water Bay Road<br />
              Clear Water Bay, Kowloon<br />
              HONG KONG<br />
              <i className="fa fa-phone fa-lg"></i>: +852 1234 5678<br />
              <i className="fa fa-fax fa-lg"></i>: +852 8765 4321<br />
              <i className="fa fa-envelope fa-lg"></i>: <a href="mailto:confusion@food.net">
                confusion@food.net</a>
            </address>
          </div>
          <div className="col-12 col-sm-4 align-self-center">
            <div className="text-center">
              <a className="btn btn-social-icon btn-
google" href="http://google.com/+><i className="fa fa-google-plus"></i></a>

```

```

      <a className="btn btn-social-icon btn-
facebook" href="http://www.facebook.com/profile.php?id="><i className="fa fa-
facebook"></i></a>

      <a className="btn btn-social-icon btn-
linkedin" href="http://www.linkedin.com/in/"><i className="fa fa-linkedin"></i></a>

      <a className="btn btn-social-icon btn-
twitter" href="http://twitter.com/"><i className="fa fa-twitter"></i></a>

      <a className="btn btn-social-icon btn-
google" href="http://youtube.com/"><i className="fa fa-youtube"></i></a>

      <a className="btn btn-social-
icon" href="mailto:"><i className="fa fa-envelope-o"></i></a>

    </div>
  </div>
</div>

<div className="row justify-content-center">

```

## Integrating Header and Footer into the React Application

- Now we open MainComponent.js and update it to integrate the header and footer into our application:

```

. . .

import Header from './HeaderComponent';
import Footer from './FooterComponent';

. . .

<Header />

<Menu dishes={this.state.dishes} onClick={(dishId) => this.onDishSelect(dishI
d)} />

<DishDetail dish={this.state.dishes.filter((dish) => dish.id === this.state.s
electedDish)[0]} />

<Footer />

. . .

```



- Then update App.css to add some new CSS classes for use in our application:

```
.row-header{
  margin:0px auto;
  padding:0px auto;
}

.row-content {
  margin:0px auto;
  padding: 50px 0px 50px 0px;
  border-bottom: 1px ridge;
  min-height:400px;
}

.footer{
  background-color: #D1C4E9;
  margin:0px auto;
  padding: 20px 0px 20px 0px;
}

.jumbotron {
  padding:70px 30px 70px 30px;
  margin:0px auto;
  background: #9575CD ;
  color:floralwhite;
}

address{
  font-size:80%;
  margin:0px;
  color:#0f0f0f;
}

.navbar-dark {
  background-color: #512DA8;
```

```
}
```

- Save all the changes and do a Git commit with the message "Header and Footer"

## Conclusions

In this exercise we updated the React application to use Font Awesome and Bootstrap Social, and also integrated two new components, Header and Footer, into our application.

## React Router

### Exercise (Video): React Router

### Exercise (Instructions): React Router

#### Objectives and Outcomes

In this exercise we learn to use the React Router to configure and set up navigation among various pages in a React application. At the end of this exercise you will be able to:

- Install and configure your application to use React Router
- Configure the routes for React router to enable you to navigate to various pages within your React application

#### Installing and Configuring React Router

- First install React Router into your project by typing the following at the prompt:

```
yarn add react-router-dom@4.2.2
```

- Then, open *App.js* and update it as follows:

```
. . .
```

```
|
```

```
import { BrowserRouter } from 'react-router-dom';
```

```
|
```

```
. . .
```

```
|
```

```
<BrowserRouter>
```

```

    <div className="App">
      <Main />
    </div>
  </BrowserRouter>
  . . .

```

## Add a Home Component

- Create a new file named `HomeComponent.js` in the components folder and add the following to it:

```

import React from 'react';

function Home(props) {
  return(
    <div className="container">
      <h4>Home</h4>
    </div>
  );
}

export default Home;

```

## Configuring the Router

- Open `MainComponent.js` file and update it as follows:

```

. . .

import Home from './HomeComponent';

. . .

```

```

import { Switch, Route, Redirect } from 'react-router-dom';

...

render() {

const HomePage = () => {
  return(
    <Home
  />
  );
}

...

<Switch>
  <Route path='/home' component={HomePage} />
  <Route exact path='/menu' component={() => <Menu dishes={this.state.dishes} />} />
  <Redirect to="/home" />
</Switch>

...

```

- Open *HeaderComponent.js* and update its contents with the following:

```

import { Nav, Navbar, NavbarBrand, NavbarToggler, Collapse, NavItem, Jumbotron } from
'reactstrap';

import { NavLink } from 'react-router-dom';

class Header extends Component {

```

```

    constructor(props) {
      super(props);

      this.toggleNav = this.toggleNav.bind(this);

      this.state = {
        isNavOpen: false
      };
    }

    toggleNav() {
      this.setState({
        isNavOpen: !this.state.isNavOpen
      });
    }

    render() {
      return(
        <div>
          <Navbar dark expand="md">
            <div className="container">
              <NavbarToggler onClick={this.toggleNav} />
              <NavbarBrand className="mr-
auto" href="/"><img src='assets/images/logo.png' height="30" width="41" alt='Ristoran
te Con Fusion' /></NavbarBrand>
              <Collapse isOpen={this.state.isNavOpen} navbar>
                <Nav navbar>
                  <NavItem>
                    <NavLink className="nav-
link" to='/home'><span className="fa fa-home fa-lg"></span> Home</NavLink>
                  </NavItem>
                  <NavItem>
                    <NavLink className="nav-
link" to='/aboutus'><span className="fa fa-info fa-lg"></span> About Us</NavLink>
                  </NavItem>
                  <NavItem>

```

```

        <NavLink className="nav-
link" to="/menu"><span className="fa fa-list fa-lg"></span> Menu</NavLink>
    </NavItem>
    <NavItem>
        <NavLink className="nav-
link" to="/contactus"><span className="fa fa-address-card fa-
lg"></span> Contact Us</NavLink>
    </NavItem>

```

- Then, open FooterComponent.js and update it as follows:

```

. . .
import { Link } from 'react-router-dom';
. . .
    <li><Link to="/home">Home</Link></li>
    <li><Link to="/aboutus">About Us</Link></li>
    <li><Link to="/menu">Menu</Link></li>
    <li><Link to="/contactus">Contact Us</Link></li>
    . . .

```

- Open MenuComponent.js and remove the onClick() from the Card in the RenderMenuItem() function.
- Save all the changes and do a Git commit with the message "React Router".

## Conclusions

In this exercise you learn about installing, configuring and using the React Router for navigation within your React app.

## React Router: Additional Resources

### PDFs of Presentations

3-VirtualDOM.pdf

PDF File

4-React-Router.pdf

PDF File

## React Resources

- [react-router](#)
- [react-router-dom](#)
- [React Router Documentation](#)
- [React Router Dom Documentation](#)

## Other Resources

- [React Router DOM: set-up, essential components, & parameterized routes](#)
- [Basic intro to React Router v4](#)
- [A Simple React Router v4 Tutorial](#)

## Module 09: Single Page Applications: Objectives and Outcomes

In this lesson you will explore single page applications (SPA) and React support for SPA. You will learn to use the routes and React router that enables the development of SPAs. At the end of this lesson, you will be able to:

- Design SPA using React
- Use the *React router* to construct SPA

### Single Page Applications (video)

### Exercise (Video): Single Page Applications Part 1

### Exercise (Instructions): Single Page Applications Part 1

## Objectives and Outcomes

In this exercise you will continue to develop the React application as a single page application, integrating the various components. At the end of this exercise you will be able to:

- Leverage the React router to enable the development of single page applications
- Provide a way of navigating among various pages using the React router support.

## Exercise Resources

dishes

JS File

promotions

JS File

leaders

JS File

comments

JS File

## Integrating the Contact Component

- Add a new file named *ContactComponent.js* file and update its contents as follows:

```
import React from 'react';  
  
function Contact(props) {  
  return(  
    <div className="container">  
      <div className="row row-content">
```



```

        <div className="col-12">
          <h3>Location Information</h3>
        </div>
        <div className="col-12 col-sm-4 offset-sm-1">
          <h5>Our Address</h5>
          <address>
            121, Clear Water Bay Road<br />
            Clear Water Bay, Kowloon<br />
            HONG KONG<br />
            <i className="fa fa-phone"></i>: +852 1234 5678<br />
            <i className="fa fa-fax"></i>: +852 8765 4321<br />
            <i className="fa fa-envelope"></i>: <a href="mailto:confusion@food.net">confusion@food.net</a>
          </address>
        </div>
        <div className="col-12 col-sm-6 offset-sm-1">
          <h5>Map of our Location</h5>
        </div>
        <div className="col-12 col-sm-11 offset-sm-1">
          <div className="btn-group" role="group">
            <a role="button" className="btn btn-primary" href="tel:+85212345678"><i className="fa fa-phone"></i> Call</a>
            <a role="button" className="btn btn-info"><i className="fa fa-skype"></i> Skype</a>
            <a role="button" className="btn btn-success" href="mailto:confusion@food.net"><i className="fa fa-envelope-o"></i> Email</a>
          </div>
        </div>
      </div>
    </div>
  );
}

export default Contact;

```

- Update the *MainComponent.js* file to integrate the ContactComponent by adding in the following:

```

. . .
|
import Contact from './ContactComponent';
|
. . .
|
<Route exact path='/contactus' component={Contact} /> />
|

```

## Updating the Home Component

- First update *dishes.js* file in the shared folder to update the dishes as follows. (NOTE: Alternately you can download *dishes.js*, *comments.js*, *promotions.js* and *leaders.js* given above in Exercise resources and move them to the shared folder):

```

export const DISHES =
[
  {
    id: 0,
    name: 'Uthappizza',
    image: '/assets/images/uthappizza.png',
    category: 'mains',
    label: 'Hot',
    price: '4.99',
    featured: true,
    description: 'A unique combination of Indian Uthappam (pancake) and Italian pi
zza, topped with Cerignola olives, ripe vine cherry tomatoes, Vidalia onion, Guntur c
hillies and Buffalo Paneer.'
  },
  {
    id: 1,
    name: 'Zucchipakoda',

```

```

    image: '/assets/images/zucchipakoda.png',
    category: 'appetizer',
    label: '',
    price: '1.99',
    featured: false,
    description: 'Deep fried Zucchini coated with mildly spiced Chickpea flour batter accompanied with a sweet-tangy tamarind sauce'
  },
  {
    id: 2,
    name: 'Vadonut',
    image: '/assets/images/vadonut.png',
    category: 'appetizer',
    label: 'New',
    price: '1.99',
    featured: false,
    description: 'A quintessential ConFusion experience, is it a vada or is it a donut?'
  },
  {
    id: 3,
    name: 'ElaiCheese Cake',
    image: '/assets/images/elaicheesecake.png',
    category: 'dessert',
    label: '',
    price: '2.99',
    featured: false,

```

- Now add a new file named *comments.js* to the shared folder and update it as follows. We are now moving the comments about the dishes into its own file:

```

export const COMMENTS =
[
  {

```

```
    id: 0,
    dishId: 0,
    rating: 5,
    comment: "Imagine all the eatables, living in conFusion!",
    author: "John Lemon",
    date: "2012-10-16T17:57:28.556094Z"
  },
  {
    id: 1,
    dishId: 0,
    rating: 4,
    comment: "Sends anyone to heaven, I wish I could get my mother-in-law to eat it!",
    author: "Paul McVites",
    date: "2014-09-05T17:57:28.556094Z"
  },
  {
    id: 2,
    dishId: 0,
    rating: 3,
    comment: "Eat it, just eat it!",
    author: "Michael Jaikishan",
    date: "2015-02-13T17:57:28.556094Z"
  },
  {
    id: 3,
    dishId: 0,
    rating: 4,
    comment: "Ultimate, Reaching for the stars!",
    author: "Ringo Starry",
    date: "2013-12-02T17:57:28.556094Z"
  },
  {
    id: 4,
```

```
dishId: 0,  
rating: 2,  
comment: "It's your birthday, we're gonna party!",  
author: "25 Cent",
```

- Next add a new file named *promotions.js* file to the shared folder and update its contents as follows:

```
export const PROMOTIONS = [  
  {  
    id: 0,  
    name: 'Weekend Grand Buffet',  
    image: '/assets/images/buffet.png',  
    label: 'New',  
    price: '19.99',  
    featured: true,  
    description: 'Featuring mouthwatering combinations with a choice of five different salads, six enticing appetizers, six main entrees and five choicest desserts. Free flowing bubbly and soft drinks. All for just $19.99 per person'  
  }  
];
```

- Next add a new file named *leaders.js* file to the shared folder and update its contents as follows:

```
export const LEADERS = [  
  {  
    id: 0,  
    name: 'Peter Pan',  
    image: '/assets/images/alberto.png',  
    designation: 'Chief Epicurious Officer',  
    abbr: 'CEO',  
    featured: false,  
    description: "Our CEO, Peter, credits his hardworking East Asian immigrant parents who undertook the arduous journey to the shores of America with the intention of giving their children the best future. His mother's wizardy in the kitchen whipping up the tastiest dishes with whatever is available inexpensively at the supermarket, wa
```

s his first inspiration to create the fusion cuisines for which The Frying Pan became well known. He brings his zeal for fusion cuisines to this restaurant, pioneering cross-cultural culinary connections."

```
    },  
    {  
      id: 1,  
      name: 'Dhanasekaran Witherspoon',  
      image: '/assets/images/alberto.png',  
      designation: 'Chief Food Officer',  
      abbr: 'CFO',  
      featured: false,  
      description: 'Our CFO, Danny, as he is affectionately referred to by his colleagues, comes from a long established family tradition in farming and produce. His experiences growing up on a farm in the Australian outback gave him great appreciation for varieties of food sources. As he puts it in his own words, Everything that runs, wins, and everything that stays, pays!'  
    },  
    {  
      id: 2,  
      name: 'Agumbe Tang',  
      image: '/assets/images/alberto.png',  
      designation: 'Chief Taste Officer',  
      abbr: 'CTO',  
      featured: false,  
      description: 'Blessed with the most discerning gustatory sense, Agumbe, our CFO, personally ensures that every dish that we serve meets his exacting tastes. Our chefs dread the tongue lashing that ensues if their dish does not meet his exacting standards. He lives by his motto, You click only if you survive my lick.'  
    },  
    {  
      id: 3,  
      name: 'Alberto Somayya',  
      image: '/assets/images/alberto.png',  
      designation: 'Executive Chef',  
      abbr: 'EC',  
      featured: true,
```

```

        description: 'Award winning three-
star Michelin chef with wide International experience having worked closely with whos
-who in the culinary world, he specializes in creating mouthwatering Indo-
Italian fusion experiences. He says, Put together the cuisines from the two craziest
cultures, and you get a winning hit! Amma Mia!'
    }
  ];

```

- Now update the HomeComponent.ts file to fetch and display the featured dish, promotion and leader as follows:

```

import React from 'react';
import { Card, CardImg, CardText, CardBody,
  CardTitle, CardSubtitle } from 'reactstrap';

function RenderCard({item}) {
  return(
    <Card>
      <CardImg src={item.image} alt={item.name} />
      <CardBody>
        <CardTitle>{item.name}</CardTitle>
        {item.designation ? <CardSubtitle>{item.designation}</CardSubtitle> : nul
1 }
        <CardText>{item.description}</CardText>
      </CardBody>
    </Card>
  );
}

function Home(props) {
  return(
    <div className="container">
      <div className="row align-items-start">

```

```

        <div className="col-12 col-md m-1">
          <RenderCard item={props.dish} />
        </div>
        <div className="col-12 col-md m-1">
          <RenderCard item={props.promotion} />
        </div>
        <div className="col-12 col-md m-1">
          <RenderCard item={props.leader} />
        </div>
      </div>
    </div>
  );
}

export default Home;

```

- Next, update MainComponent.js as follows:

```

...
import { COMMENTS } from '../shared/comments';
import { PROMOTIONS } from '../shared/promotions';
import { LEADERS } from '../shared/leaders';
...
class Main extends Component {
  constructor(props) {
    super(props);

    this.state = {
      dishes: DISHES,

```



```

    comments: COMMENTS,
    promotions: PROMOTIONS,
    leaders: LEADERS
  };
}
...
const HomePage = () => {
  return(
    <Home
      dish={this.state.dishes.filter((dish) => dish.featured)[0]}
      promotion={this.state.promotions.filter((promo) => promo.featured)[0]}
      leader={this.state.leaders.filter((leader) => leader.featured)[0]}
    />
  );
}
...

```

- Save all the changes and do a Git commit with the message "Single Page Applications Part 1".

## Conclusions

In this exercise you developed the Angular application further by integrating the components into a single page application.

## Module 7: React Router: Parameters

### Exercise (Video): Single Page Applications Part 2

### Exercise (Instructions): Single Page Applications Part 2

## Objectives and Outcomes

In this exercise you will integrate the DishdetailComponent into your single page application. You will use a route parameter in the URL to pass in the details of the selected dish to the DishdetailComponent. At the end of this exercise you will be able to:

- Configure the routes in your React router configuration to enable the use of route parameters within the URL to pass information to a component.

## Updating to Use Parameters on Routes

- Open *MenuComponent.js* and add the following changes to it to enable the information about the selected dish to be passed to the DishdetailComponent:

```
. . .  
  
import { Card, CardImg, CardImgOverlay,  
  CardTitle, Breadcrumb, BreadcrumbItem } from 'reactstrap';  
import { Link } from 'react-router-dom';  
  
function RenderMenuItem ({dish, onClick}) {  
  return (  
    <Card>  
      <Link to={` /menu/${dish.id}`} >  
        <CardImg width="100%" src={dish.image} alt={dish.name} />  
        <CardImgOverlay>  
          <CardTitle>{dish.name}</CardTitle>  
        </CardImgOverlay>  
      </Link>  
    </Card>  
  );  
}
```

```

    return (
      <div className="container">
        <div className="row">
          <Breadcrumb>
            <BreadcrumbItem><Link to="/home">Home</Link></BreadcrumbItem>
            <BreadcrumbItem active>Menu</BreadcrumbItem>
          </Breadcrumb>
          <div className="col-12">
            <h3>Menu</h3>
            <hr />
          </div>
        </div>
        <div className="row">
          {menu}
        </div>
      </div>
    );
  }
  ...

```

- Open *MainComponent.js* and update it as follows:

```

...
|
|
const DishWithId = ({match}) => {
  return(
    <DishDetail dish={this.state.dishes.filter((dish) => dish.id === parseInt(match.params.dishId,10))[0]}
    comments={this.state.comments.filter((comment) => comment.dishId === parseInt(match.params.dishId,10))} />
  );
};

```

```

...
|
|
|   <Route path='/menu/:dishId' component={DishWithId} />
|
|
|   ...

```

## Updating DishDetail Component

- Open *DishdetailComponent.js* and update it as follows:

```

...
|
|
|   import { Card, CardImg, CardText, CardBody,
|           CardTitle, Breadcrumb, BreadcrumbItem } from 'reactstrap';
|   import { Link } from 'react-router-dom';
|
|
|   ...
|
|   return (
|       <div className="container">
|           <div className="row">
|               <Breadcrumb>
|
|               <BreadcrumbItem><Link to="/menu">Menu</Link></BreadcrumbItem>
|               <BreadcrumbItem active>{props.dish.name}</BreadcrumbItem>
|               </Breadcrumb>
|               <div className="col-12">
|                   <h3>{props.dish.name}</h3>
|                   <hr />
|               </div>
|           </div>
|       </div>
|   );

```

```

        <div className="row">
          <div className="col-12 col-md-5 m-1">
            <RenderDish dish={props.dish} />
          </div>
          <div className="col-12 col-md-5 m-1">
            <RenderComments comments={props.comments} />
          </div>
        </div>
      </div>
    );
  }
  ...

```

## Adding Breadcrumbs to ContactComponent

- Open ContactComponent.js and add Breadcrumbs to it as follows:

```

...
|
import { Breadcrumb, BreadcrumbItem } from 'reactstrap';
import { Link } from 'react-router-dom';
|
...
|
        <div className="row">
          <Breadcrumb>
            <BreadcrumbItem><Link to="/home">Home</Link></BreadcrumbItem>
            <BreadcrumbItem active>Contact Us</BreadcrumbItem>
          </Breadcrumb>
          <div className="col-12">
            <h3>Contact Us</h3>
            <hr />
          </div>
        </div>

```



- Save all the changes and do a Git commit with the message "Single Page Applications Part 2".

## Conclusions

In this exercise you have seen the use of parameters within the URL to pass information to another component.

## Single Page Applications: Additional Resources

### PDFs of Presentations

5-Single-Page-Apps.pdf

PDF File

6-React-Router-Parameters.pdf

PDF File

### React Resources

- [react-router](#)
- [react-router-dom](#)
- [React Router Documentation](#)
- [React Router Dom Documentation](#)

### Other Resources

- [Arrow Functions](#)
- [Single Page Applications \(Wikipedia\)](#)
- [Deep linking](#)
- [Single Page Apps in depth](#)
- [SPA and the Single Page Myth](#)

## Assignment 2: React Router and Single Page Applications

### Peer-graded Assignment: React Router and Single Page Applications

Deadline Oct 30, 11:59 PM PDT

Ready for the assignment?

You will find instructions below to submit.

1. [Instructions](#)
2. [My submission](#)
3. [Discussions](#)

In this assignment you will continue working with the React application by adding a new component named `AboutComponent` to serve up the details of the corporate leaders, and you will then integrate the `AboutComponent` into the single page application.

### Step-By-Step Assignment Instructions

less

#### Assignment Resources

AboutComponent

JS File

#### Objectives and Outcomes

At the end of this assignment, you should have completed the following tasks:

- Integrated the `AboutComponent` given above into the single page application.
- Added a new functional component named `<RenderLeader>` through the `RenderLeader()` function to `AboutComponent.js` that renders the details of a given leader using the reactstrap `<Media>` component.
- Construct and render the list of leaders in the About Us page using the `<RenderLeader>` component implemented above.

## Assignment Requirements

This assignment requires you to complete the following tasks. Detailed instructions for each task are given below. The picture of the completed web page included below indicates the location within the web page that will be updated by the three tasks.

### Task 1

In this task you will be integrating the `AboutComponent` into the single page application:

- First, download the *AboutComponent.js* given above and move it to the components folder, and
- Update the *MainComponent.js* file to integrate the `AboutComponent` into the single page application. You should be able to navigate to the `AboutComponent` by clicking the links in the navigation bar and the footer.

### Task 2

In this task you will implement a new functional component named `<RenderLeader>` through the `RenderLeader()` function:

- The `RenderLeader()` function receives the details of a leader as its parameter,
- Implement `RenderLeader()` function to use the Leader information received as a parameter and render the leader information as shown in the screenshot below using the reactstrap `<Media>` component. Recall how we used the `<Media>` component in the first module to render the details of a dish in the `MenuComponent`.
- This will make available the `<RenderLeader>` component for use within your `AboutComponent`.

### Task 3

In this task you will use the `<RenderLeader>` component created in the previous task to render a list of leaders within your `AboutComponent`:

- Update the `leaders` variable within the `AboutComponent()` function to make use of the `<RenderLeader>` component to render the list of leaders.





## Review criteria

less

Upon completion of the assignment, your submission will be reviewed based on the following criteria:

### Task 1:

- The AboutComponent.js file has been downloaded and integrated into our React application.
- The React application has been appropriately updated to enable navigation to the About Us page of our application.

### Task 2:

- Implemented a new <RenderLeader> functional component in your application.
- Used the reactstrap <Media> component to render the details of a leader.

### Task 3:

- Updated the leaders variable within the AboutComponent() function to make use of the <RenderLeader> component to render the list of leaders.

## Assignment 2: React Router and Single Page Applications: Additional Resources

### Assignment Resources

AboutComponent

JS File

Screenshots



## React Resources

- [react-router](#)
- [react-router-dom](#)
- [React Router Documentation](#)
- [React Router Dom Documentation](#)
- [reactstrap <Media> component](#)

## UI Design and Prototyping: Objectives and Outcomes

Now that you are more clear about your project idea, it's time to conceive how your project is going to look like for the end-users. This is the time to design the user interface and the flow of your application. User interface design and prototyping helps you to conceptualize the look and feel of your application. This can be achieved in two ways: using wireframe diagrams, or using prototyping tools. We provide links to wireframing and prototyping tools in the additional resources. The focus in this lesson is to be able to visually represent various UI elements to enable designing your application. The aim is to deliver a reasonable representation of the end-user experience with your application. At the end of this lesson, you should be able to:

- Construct a wireframe diagram to visually represent the structure of your user interface
- Construct a prototype to enable understanding the flow of your application

## UI Design and Prototyping Report Template

### Project Title

#### 1. Introduction

- Give a brief introduction to your project and the list of features. Summarize in a few sentences what you proposed in the ideation report.

#### 2. User Interface Design and Prototype

- Give some sample user interface layouts for your application. You can use either wireframe diagrams or prototyping tools to construct the mock representations of your UI design
- Briefly explain the rationale behind designing your UI and how it is geared towards supporting the list of features for your application.

### 3. Navigation Structure

- Give a brief overview of the navigation structure for your application.
- Briefly indicate a typical flow of your application in terms of user experience. You can use any way of representing the flow. You can also construct a prototype using one of the prototyping tools to illustrate this.

### 4. References

- Provide any references relevant to the report.

## Honors Peer-graded Assignment: UI Design and Prototyping

Deadline Oct 30, 11:59 PM PDT

Ready for the assignment?

You will find instructions below to submit.

1. [Instructions](#)
2. [My submission](#)
3. [Discussions](#)

In this assignment you will be submitting a written report that includes some user interface designs and the flow of your application and the navigation structure. The report should follow the report template structure given earlier in this lesson.

## Review criteria

less

Your submission will be reviewed based on the following criteria by peers in order to provide you with constructive feedback on your project idea:

1. Does the report provide a brief overview of the project and the list of features ?
2. Does the report contain some user interface designs that showcase the realization of the features listed above?
3. Does the report contain information about the application flow and navigation structure?
4. Does the report provide references to suitable sources in support of the project?

## UI Design and Prototyping: Additional Resources

### Wireframing, Mockups and UI Design

- [Wireframe.cc](https://www.wireframe.cc)
- [Moqups.com](https://moqups.com)
- [Axure](https://axure.com)
- [proto.io](https://proto.io)
- [framerjs.com](https://framerjs.com)
- [The 20 best wireframe tools](#)
- [Web Design Inspirations](#)
- [Adobe Experience Design](#)
- [Free Bootstrap Wireframing Set for PowerPoint](#)

### UI Templates

- [Bootstrap Expo](#)
- [Ionic Showcase](#)

### Information Architecture

- [A visual vocabulary for describing information architecture and interaction design](#)
- [The Elements of User Experience](#)
- [The Elements of User Experience: User-Centered Design for the Web and Beyond \(2nd Edition\) \(Voices That Matter\)](#)

## Controlled Forms: Objectives and Outcomes

In this lesson you will learn about controlled components and how they can be used to design controlled forms in your React application. At the end of this exercise you will be able to:

- Design a controlled form in your React application.

## Controlled Forms

## Exercise (Video): Controlled Forms

## Exercise (Instructions): Controlled Forms

### Objectives and Outcomes

In this exercise you will learn about using controlled components to design controlled forms in your React application. At the end of this exercise you will be able to:

- Design a controlled form in your React application.

### Importing the Necessary Components

- You will start out by importing the necessary components from `reactstrap` into *ContactComponent.js* as follows:

```
...  
  
import React, { Component } from 'react';  
import { Breadcrumb, BreadcrumbItem,  
        Button, Form, FormGroup, Label, Input, Col } from 'reactstrap';  
  
...
```

- You will then change the *ContactComponent* to a class-based component as follows:

```
...  
  
class Contact extends Component {  
  
  render() {  
  
    ...  
  
  }  
  
}
```

## Creating the Controlled Form

- Update the *ContactComponent.js* file as follows to set up the Controlled Form:

```
...
|
|   constructor(props) {
|     super(props);
|
|     this.state = {
|       firstname: '',
|       lastname: '',
|       telnum: '',
|       email: '',
|       agree: false,
|       contactType: 'Tel.',
|       message: ''
|     };
|
|     this.handleChange = this.handleChange.bind(this);
|     this.handleSubmit = this.handleSubmit.bind(this);
|
|   }
|
|   handleChange(event) {
|     const target = event.target;
|     const value = target.type === 'checkbox' ? target.checked : target.value;
|     const name = target.name;
|
|     this.setState({
|       [name]: value
|     });
|   }
| }
```



```

    }
  }
  handleSubmit(event) {
    console.log('Current State is: ' + JSON.stringify(this.state));
    alert('Current State is: ' + JSON.stringify(this.state));
    event.preventDefault();
  }
}
...

```

- Then add the controlled form to it as follows:

```

...
<div className="row row-content">
  <div className="col-12">
    <h3>Send us your Feedback</h3>
  </div>
  <div className="col-12 col-md-9">
    <Form onSubmit={this.handleSubmit}>
      <FormGroup row>
        <Label htmlFor="firstname" md={2}>First Name</Label>
        <Col md={10}>
          <Input type="text" id="firstname" name="first
name"
            placeholder="First Name"
            value={this.state.firstname}
            onChange={this.handleInputChange} />
        </Col>
      </FormGroup>
    </Form>
  </div>
</div>

```

```

<Label htmlFor="lastname" md={2}>Last Name</Label>
>
<Col md={10}>
  <Input type="text" id="lastname" name="lastna
me"
  placeholder="Last Name"
  value={this.state.lastname}
  onChange={this.handleInputChange} />
</Col>
</FormGroup>
<FormGroup row>
  <Label htmlFor="telnum" md={2}>Contact Tel.</Label>
  <Col md={10}>
    <Input type="tel" id="telnum" name="telnum"
    placeholder="Tel. number"
    value={this.state.telnum}
    onChange={this.handleInputChange} />
  </Col>
</FormGroup>
<FormGroup row>
  <Label htmlFor="email" md={2}>Email</Label>
  <Col md={10}>
    <Input type="email" id="email" name="email"
    placeholder="Email"

```

- Save all changes and do a Git commit with the message "Controlled Forms".

## Conclusions

In this exercise you have learnt to create a controlled form within your React application.

## Exercise (Video): Controlled Form Validation

## Exercise (Instructions): Controlled Form Validation

### Objectives and Outcomes

In this exercise you will be introduced to simple form validation for controlled forms in React. At the end of this exercise you will be able to:

- Configure and perform simple form validation for your controlled forms

## Simple Form Validation

- Open *ContactComponent.js* and update it as follows to introduce the support to track form errors and perform validation:

```
...  
import { Breadcrumb, BreadcrumbItem, Button, Form, FormGroup, Label, Input, Col, Row, FormFeedback } from 'reactstrap';  
...  
  
class Contact extends Component {  
  
  constructor(props) {  
    super(props);  
  
    this.state = {  
      firstname: '',  
      lastname: '',  
      telnum: '',  
      email: '',  
      agree: false,  
      contactType: 'Tel.',  
      message: '',  
      touched: {  
        firstname: false,  
        lastname: false,  
        telnum: false,  
        email: false  
      }  
    }  
  }  
}
```

```

    this.handleSubmit = this.handleSubmit.bind(this);
    this.handleChange = this.handleChange.bind(this);
    this.handleBlur = this.handleBlur.bind(this);
  }
  ...
  handleBlur = (field) => (evt) => {
    this.setState({
      touched: { ...this.state.touched, [field]: true }
    });
  }
  validate(firstname, lastname, telnum, email) {
    const errors = {
      firstname: '',

```

- Now that we have introduced some functions that can be used for form validation, let us update the form itself to make use of these as follows:

```

  ...
  <FormGroup row>
    <Label htmlFor="firstname" md={2}>First Name</Label>
    <Col md={10}>
      <Input type="text" id="firstname" name="firstname"
        placeholder="First Name"
        value={this.state.firstname}
        valid={errors.firstname === ''}
        invalid={errors.firstname !== ''}
        onBlur={this.handleBlur('firstname')}
        onChange={this.handleChange} />
      <FormFeedback>{errors.firstname}</FormFeedback>
    </Col>
  </FormGroup>

```

```

    </Col>
  </FormGroup>
  <FormGroup row>
    <Label htmlFor="lastname" md={2}>Last Name</Label>
  >
    <Col md={10}>
      <Input type="text" id="lastname" name="lastna
me"
      placeholder="Last Name"
      value={this.state.lastname}
      valid={errors.lastname === ''}
      invalid={errors.lastname !== ''}
      onBlur={this.handleBlur('lastname')}
      onChange={this.handleChange} />
      <FormFeedback>{errors.lastname}</FormFeedback>
    >
  </Col>
</FormGroup>
<FormGroup row>
  <Label htmlFor="telnum" md={2}>Contact Tel.</Labe
l>
  <Col md={10}>
    <Input type="tel" id="telnum" name="telnum"
    placeholder="Tel. Number"
    value={this.state.telnum}
    valid={errors.telnum === ''}
    invalid={errors.telnum !== ''}
    onBlur={this.handleBlur('telnum')}
    onChange={this.handleChange} />
    <FormFeedback>{errors.telnum}</FormFeedback>
  </Col>

```

- You can now test your form by typing in invalid input and check how the form validation works.

- Save all the changes and do a Git commit with the message "Controlled Form Validation"

## Conclusions

In this exercise you have learnt about doing simple form validation for controlled forms in React.

## Controlled Forms: Additional Resources

### PDFs of Presentations

1-Controlled-Components-Forms.pdf

PDF File

### React Resources

- [Controlled Components](#)
- [reactstrap Form](#)

### Other Resources

- [Controlled / Uncontrolled React Components](#)
- [Controlled and Uncontrolled Input Values in React](#)
- [Controlled and uncontrolled form inputs in React don't have to be complicated](#)
- [How to Work with Forms, Inputs and Events in React](#)
- [Some Thoughts On Forms in React](#)
- [Instant form field validation with React's controlled inputs](#)

## Module 8: Uncontrolled Forms: Objectives and Outcomes

Forms in React can be approached in two ways, either through uncontrolled components, or controlled components. In this lesson we will deal with uncontrolled forms. At the end of this lesson you will be able to:

- Create uncontrolled forms through uncontrolled components in React
- Handle the form submission in the React application

# Uncontrolled Components

## Exercise (Video): Uncontrolled Forms

## Exercise (Instructions): Uncontrolled Forms

In this exercise we will create an uncontrolled form within our React application using the uncontrolled component approach. At the end of this exercise you will be able to:

- Create uncontrolled forms through uncontrolled components in React
- Handle the form submission in the React application

### Adding a Modal to Host the Form

- Update *HeaderComponent.js* as follows to add a new Modal to the application to host the form:

```
...
|
import { Navbar, NavbarBrand, Nav, NavbarToggler, Collapse, NavItem, Jumbotron,
  Button, Modal, ModalHeader, ModalBody,
  Form, FormGroup, Input, Label } from 'reactstrap';
|
...
|
  this.state = {
    isNavOpen: false,
    isModalOpen: false
  };
|
...
|
  this.toggleModal = this.toggleModal.bind(this);
|
...
|
```

```

    toggleModal() {
      this.setState({
        isModalOpen: !this.state.isModalOpen
      });
    }
  }
  ...
  <Modal isOpen={this.state.isModalOpen} toggle={this.toggleModal}>
    <ModalHeader toggle={this.toggleModal}>Login</ModalHeader>
    <ModalBody>
      ...
    </ModalBody>
  </Modal>
  ...

```

- Then, add a button to the Navbar to enable toggling the modal:

```

  ...
  <Nav className="ml-auto" navbar>
    <NavItem>
      <Button outline onClick={this.toggleModal}><span
        className="fa fa-sign-in fa-lg"></span> Login</Button>
    </NavItem>
  </Nav>
  ...

```

## Adding the Uncontrolled Form

- Add the form to the modal body as shown below:

```

  ...

```



```

        <Form onSubmit={this.handleLogin}>
          <FormGroup>
            <Label htmlFor="username">Username</Label>
            <Input type="text" id="username" name="username"
              innerRef={(input) => this.username = input} /
            >
          </FormGroup>
          <FormGroup>
            <Label htmlFor="password">Password</Label>
            <Input type="password" id="password" name="password"
              innerRef={(input) => this.password = input}
            />
          </FormGroup>
          <FormGroup check>
            <Label check>
              <Input type="checkbox" name="remember"
                innerRef={(input) => this.remember = input}
              />
              Remember me
            </Label>
          </FormGroup>
          <Button type="submit" value="submit" color="primary">
            Login</Button>
        </Form>
      </div>
    </div>
  )
}

```

- Then, add the following function to the class to handle the form submission:

```

...

this.handleLogin = this.handleLogin.bind(this);
...

```

```

|
|
| handleLogin(event) {
|     this.toggleModal();
|     alert("Username: " + this.username.value + " Password: " + this.password.
value
|         + " Remember: " + this.remember.checked);
|     event.preventDefault();
|
|
| }
|
|
| . . .

```

- Save all the changes and do a Git commit with the message "Uncontrolled Form"

## Conclusions

In this exercise we learnt to use an uncontrolled component approach to add a form to our React application.

## Uncontrolled Forms: Additional Resources

PDFs of Presentations

### 2-Uncontrolled-Components-Forms.pdf

PDF File

## React Resources

- [Uncontrolled Components](#)
- [reactstrap Form](#)
- [reactstrap Modal Component](#)

## Other Resources

- [Controlled / Uncontrolled React Components](#)
- [Controlled and Uncontrolled Input Values in React](#)
- [Controlled and uncontrolled form inputs in React don't have to be complicated](#)

- [How to Work with Forms, Inputs and Events in React](#)
- [Some Thoughts On Forms in React](#)

## Introduction to Redux: Objectives and Outcomes

In this lesson you will learn about the Flux architecture as a way of structuring your React application. You will also be introduced to Redux, a realization of the flux architecture. At the end of this lesson you will be able to:

- Install and Configure Redux in your application
- Enable your React app to make use of Redux

## Module 9: The Model-View-Controller Framework

## Module 10: The Flux Architecture

## Module xx: Introduction to Redux

## Chapter 1: Exercise (Video): Introduction to Redux

## Chapter 2: Exercise (Instructions): Introduction to Redux

### Objectives and Outcomes

In this exercise you will learn to use Redux. You will install and configure Redux and use it within your React application. At the end of this exercise you will be able to:

- Install and configure Redux within your application
- Configure your React application to make use of Redux

### Installing and Configuring Redux

- As a first step you will install Redux and React-Redux into your application as follows:

```
yarn add redux@3.7.2
```

```
yarn add react-redux@5.0.7
```

- Next, create a folder named *redux* in the *src* folder and then add a file named *reducer.js* with the code below:

```
import { DISHES } from '../shared/dishes';
```

```

import { COMMENTS } from '../shared/comments';
import { PROMOTIONS } from '../shared/promotions';
import { LEADERS } from '../shared/leaders';

export const initialState = {
  dishes: DISHES,
  comments: COMMENTS,
  promotions: PROMOTIONS,
  leaders: LEADERS
};

export const Reducer = (state = initialState, action) => {
  return state;
};

```

- Then, add a file named *configureStore.js* in the *redux* folder and add the following code to it:

```

import { createStore } from 'redux';
import { Reducer, initialState } from './reducer'

export const ConfigureStore = () => {
  const store = createStore(
    Reducer, // reducer
    initialState, // our initialState
  );

  return store;
}

```

- Next, open *App.js* and update it as follows:

```

...

```

```


```

```


```

```

import { Provider } from 'react-redux';
import { ConfigureStore } from '../redux/configureStore';

const store = ConfigureStore();

<Provider store={store}>
  <BrowserRouter>
    <div className="App">
      <Main />
    </div>
  </BrowserRouter>
</Provider>

```

- Finally, update MainComponent.js to connect it to Redux store and use it:

```

import { Switch, Route, Redirect, withRouter } from 'react-router-dom'
import { connect } from 'react-redux';

const mapStateToProps = state => {
  return {
    dishes: state.dishes,
    comments: state.comments,
    promotions: state.promotions,
    leaders: state.leaders
  }
}

```

```

}
|
|
|
|
const HomePage = () => {
  return(
    <Home
      dish={this.props.dishes.filter((dish) => dish.featured)[0]}
      promotion={this.props.promotions.filter((promo) => promo.featured)[0]}
      leader={this.props.leaders.filter((leader) => leader.featured)[0]}
    />
  );
}

const DishWithId = ({match}) => {
  return(
    <DishDetail dish={this.props.dishes.filter((dish) => dish.id === parseInt(m
atch.params.dishId,10))[0]}
      comments={this.props.comments.filter((comment) => comment.dishId === pars
eInt(match.params.dishId,10))} />
  );
};

return (
  <div>
    <Header />
    <div>
      <Switch>
        <Route path='/home' component={HomePage} />

```

- Save all the changes and do a Git commit with the message "Intro. to Redux".

## Conclusions

In this exercise you learnt to install and configure Redux and use it in your React application.

## Introduction to Redux: Additional Resources

### PDFs of Presentations

3-MVC.pdf

PDF File

4-Flux-Arch.pdf

PDF File

5-Intro-Redux.pdf

PDF File

### React Resources

- [Redux](#)
- [Redux on Github](#)
- [React and Redux](#)
- [Redux Basics Documentation](#)
- [The Flux Architecture](#)

### Other Resources

- [Redux Tutorials](#)
- [Flux Architecture In Depth Overview](#)

## Module xx: React Redux Form: Objectives and Outcomes

In this lesson we will learn to create forms using react-redux-form that enables us to store form information in the Redux store, and brings additional benefits. At the end of this lesson you will be able to:

- Configure and use react-redux-form to create Controlled forms
- Store the form state in the Redux store

## Chapter 1. React Redux Forms

## Chapter 2. Exercise (Video): React Redux Form

## Chapter 3. Exercise (Instructions): React Redux Form

### Objectives and Outcomes

In this exercise you will install `react-redux-form` and then convert the controlled form that we created earlier into a form supported by `react-redux-form`. At the end of this exercise you will be able to:

- Install and configure `react-redux-form`
- Implement a controlled form using `react-redux-form`

### Installing and Using `react-redux-form`

- We first install the *react-redux-form* into our project as follows:

```
yarn add react-redux-form@1.16.8
```

- Then open *ContactComponent.js* and update the Feedback Form to use `react-redux-form`:

```
. . .  
  
import { Breadcrumb, BreadcrumbItem,  
        Button, Row, Col, Label } from 'reactstrap';  
import { Control, LocalForm, Errors } from 'react-redux-form';  
  
. . .  
  
handleSubmit(values) {  
    console.log('Current State is: ' + JSON.stringify(values));  
    alert('Current State is: ' + JSON.stringify(values));  
    // event.preventDefault();  
}
```



```

<LocalForm onSubmit={(values) => this.handleSubmit(values)}>
  <Row className="form-group">
    <Label htmlFor="firstname" md={2}>First Name</Label>
    <Col md={10}>
      <Control.text model=".firstname" id="firstname" name="firstname"
        placeholder="First Name"
        className="form-control"
      />
    </Col>
  </Row>
  <Row className="form-group">
    <Label htmlFor="lastname" md={2}>Last Name</Label>
    <Col md={10}>
      <Control.text model=".lastname" id="lastname" name="lastname"
        placeholder="Last Name"
        className="form-control"
      />
    </Col>
  </Row>
  <Row className="form-group">
    <Label htmlFor="telnum" md={2}>Contact Tel.</Label>
    <Col md={10}>
      <Control.text model=".telnum" id="telnum" name="telnum"

```

- Save all the changes and do a Git commit with the message "React Redux Form".

## Conclusions

In this exercise we installed and used the react-redux-form to design a controlled form.

# Module xx: Exercise (Video): React Redux Form Validation

## Chapter 2: Exercise (Instructions): React Redux Form Validation

### Objectives and Outcomes

In this exercise we will explore simple form validation for react-redux-form. At the end of this exercise you will be able to:

- Configure and implement simple form validation for controlled forms designed using react-redux-form.

### Implementing Simple Form Validation

- Open *ContactComponent.js* and update it as follows to implement form validation:

```
. . .  
  
  
  
const required = (val) => val && val.length;  
const maxLength = (len) => (val) => !(val) || (val.length <= len);  
const minLength = (len) => (val) => val && (val.length >= len);  
const isNumber = (val) => !isNaN(Number(val));  
const validEmail = (val) => /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$/i.test(val);  
  
. . .  
  
  
      <Row className="form-group">  
        <Label htmlFor="firstname" md={2}>First Name</Label>  
        <Col md={10}>  
          <Control.text model=".firstname" id="firstname" name="firstname"  
            placeholder="First Name"  
            className="form-control"/>
```

```

        validators={{
          required, minLength: minLength(3), maxLength:
          maxLength(15)
        }}
      />
    <Errors
      className="text-danger"
      model=".firstname"
      show="touched"
      messages={{
        required: 'Required',
        minLength: 'Must be greater than 2 characters',
        maxLength: 'Must be 15 characters or less'
      }}
    />
  </Col>
</Row>
<Row className="form-group">
  <Label htmlFor="lastname" md={2}>Last Name</Label>
  <Col md={10}>
    <Control.text model=".lastname" id="lastname" name="lastname"
      placeholder="Last Name"
      className="form-control"
      validators={{

```

- Save all the changes and do a Git commit with the message "React Redux Form Validation"

## Conclusions

In this exercise you learnt about implementing simple form validation using react-redux-form.

## Chapter 2: React Redux Form: Additional Resources

PDFs of Presentations

6-React-Redux-Form.pdf

PDF File

## React Resources

- [React Redux Form Documentation](#)

## Other Resources

- [The boring React Redux forms](#)
- [How to populate react-redux-form with dynamic default values](#)
- [Should you store your form state in Redux?](#)

## Assignment 3: React Forms and Redux

### Peer-graded Assignment: React Forms and Redux

Deadline Nov 6, 11:59 PM PST

Ready for the assignment?

You will find instructions below to submit.

In this assignment you will be updating the React application with a react-redux-form based local form and do the form validation in code.

### Step-By-Step Assignment Instructions

less

### Assignment Overview

In this assignment you will update the *DishdetailComponent.js* to include a form and do the form validation in code. At the end of this assignment, you should have completed the following tasks to update the page:

- Added a new class component named `CommentForm` to *DishdetailComponent.js*.
- Provide a form to enable users to submit their comments
- Validate the information entered by the users in the form

## Assignment Requirements

### Task 1

In this task you will add a new class component named `CommentForm`. You need to complete the following:

- Add a new class component named `CommentForm` that will add a button to the view as shown in the image below.
- When the button is clicked, it toggles a modal that will display the comment form.
- The `CommentForm` component is used by the `RenderComments` function to display the button for toggling the modal.

### Task 2

In this task you will construct the form for users to submit their comments as shown in the image below. You need to complete the following:

- Set up the form as a local form using the `react-redux-form` with the three fields: `author`, `rating` and `comment`.
- The `rating` field in the comment form is implemented using a `select`, the `author` is implemented using a text field, while the `comment` uses a `textarea` with six rows.

### Task 3

In this task, you will enable form validation as shown in the images below. You need to complete the following:

- The `author` field should at least be three characters long.
- The `author` field should be less than or equal to 15 characters.
- The user should be alerted by showing the invalid message displayed at the bottom of the field.



## Ristorante con Fusion

We take inspiration from the World's best cuisines, and create a unique fusion experience. Our lipsmacking creations will tickle your culinary senses!

[Home](#) / [Menu](#) / Uthappizza

### Menu



#### Uthappizza

A unique combination of Indian Uthappam (pancake) and Italian pizza, topped with Cerignola olives, ripe vine cherry tomatoes, Vidalia onion, Guntur chillies and Buffalo Paneer.

#### Comments

Imagine all the eatables, living in conFusion!

-- John Lemon , Oct 17, 2012

Sends anyone to heaven, I wish I could get my mother-in-law to eat it!

-- Paul McVites , Sep 06, 2014

Eat it, just eat it!

-- Michael Jaikishan , Feb 14, 2015

Ultimate, Reaching for the stars!

-- Ringo Starry , Dec 03, 2013

It's your birthday, we're gonna party!

-- 25 Cent , Dec 03, 2011

**Task 1**

#### Links

[Home](#)  
[About](#)  
[Menu](#)  
[Contact](#)

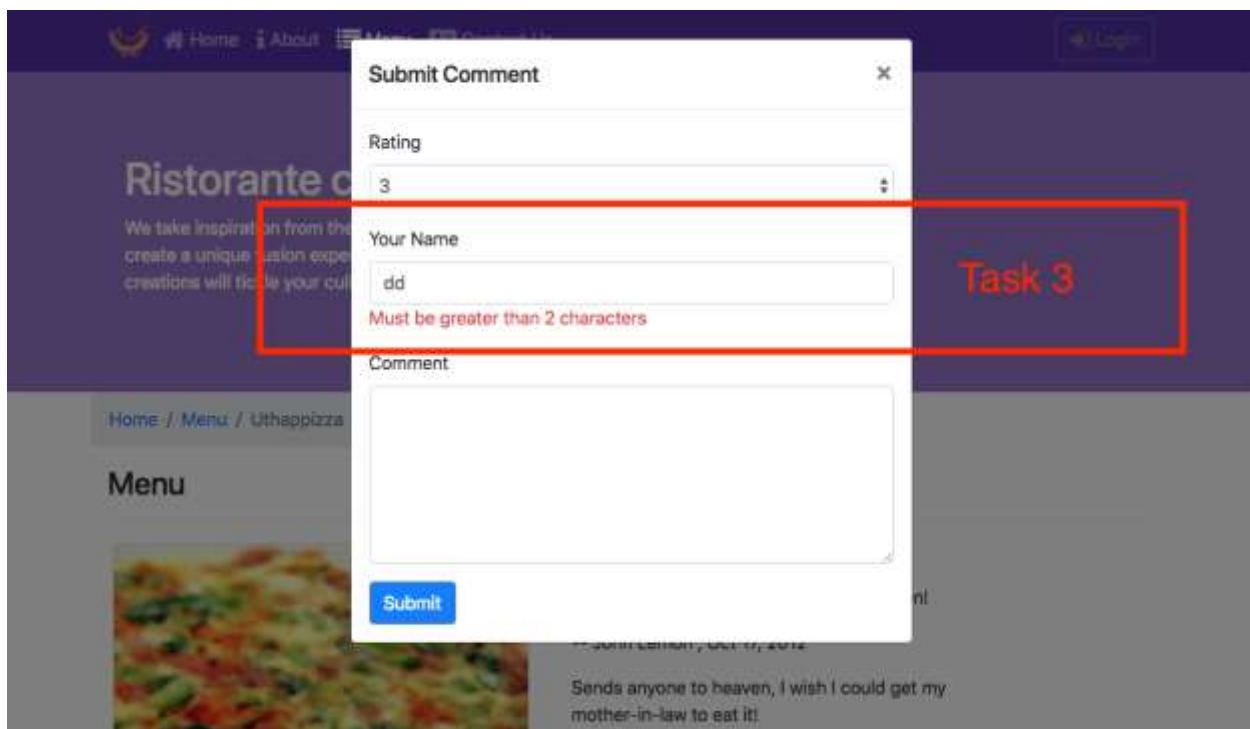
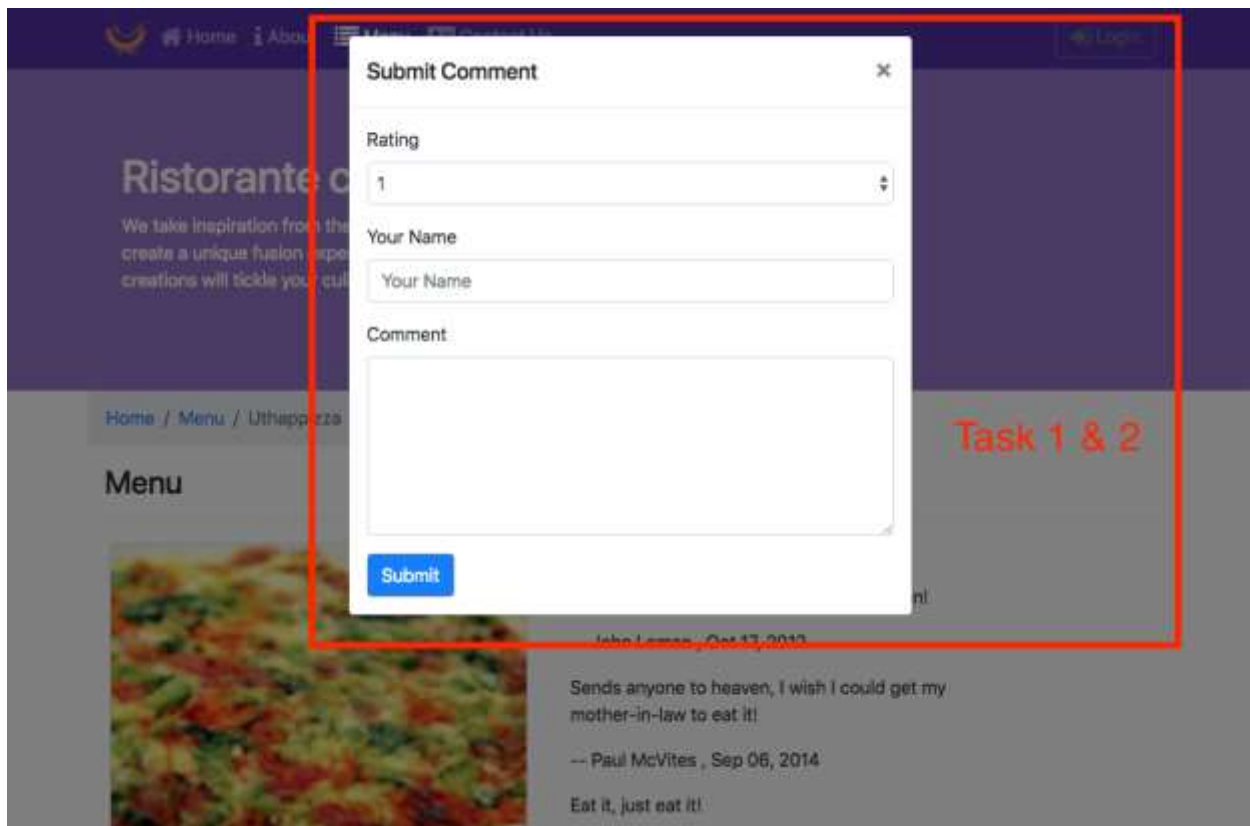
#### Our Address

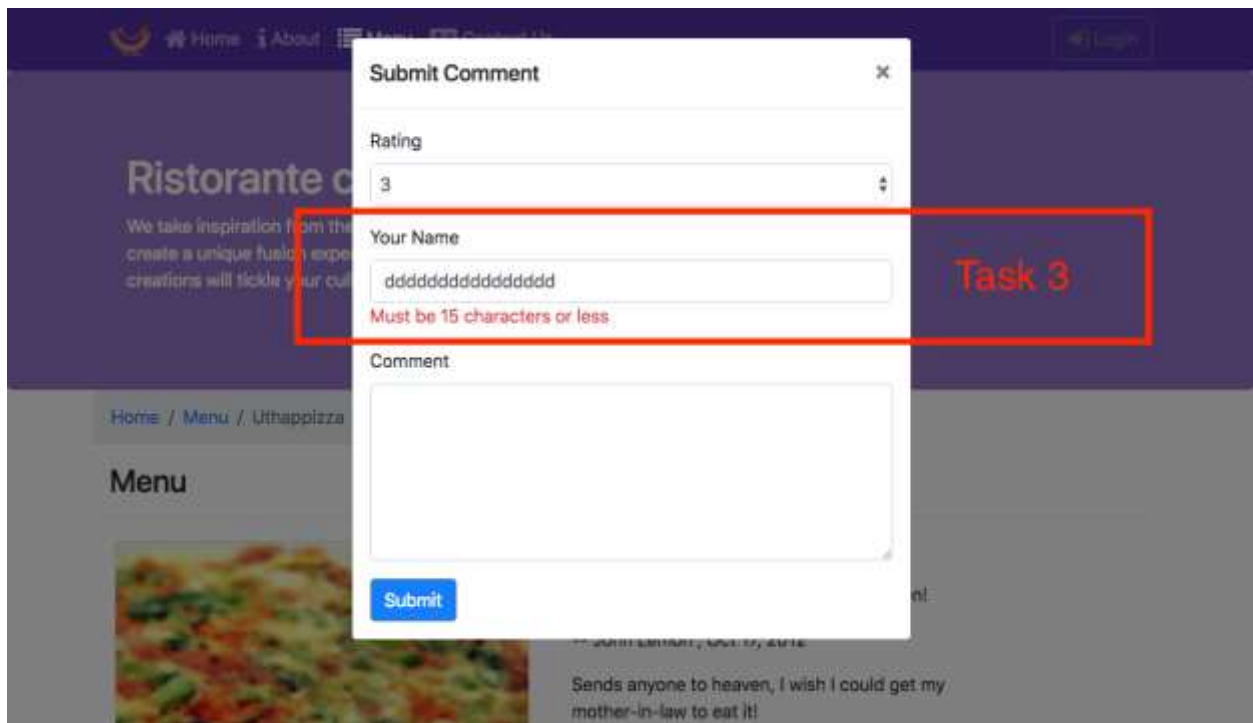
121, Clear Water Bay Road  
Clear Water Bay, Kowloon  
HONG KONG  
☎ +852 1234 5678  
📠 +852 8765 4321  
✉ [confusion@food.net](mailto:confusion@food.net)



© Copyright 2018 Ristorante Con Fusion







## Review criteria

- Upon completion of the assignment, your submission will be reviewed based on the following criteria:

### Task 1:

- A `CommentForm` component is implemented that adds a button to the view, which when clicked will toggle a modal containing the form.

### Task 2:

- The form is set up with the three fields correctly.
- A select is included in the form to enable users to submit the rating, a text field for author and a text area for the comment.

### Task 3:

- The author field is being properly validated. If incorrect, the user is alerted.

## Assignment 3: React Forms and Redux: Additional Resources

### Assignment Screenshots





## Ristorante con Fusion

We take inspiration from the World's best cuisines, and create a unique fusion experience. Our lipsmacking creations will tickle your culinary senses!

[Home](#) / [Menu](#) / Uthappizza

### Menu



#### Uthappizza

A unique combination of Indian Uthappam (pancake) and Italian pizza, topped with Cerignola olives, ripe vine cherry tomatoes, Vidalia onion, Guntur chillies and Buffalo Paneer.

#### Comments

Imagine all the eatables, living in conFusion!

-- John Lemon , Oct 17, 2012

Sends anyone to heaven, I wish I could get my mother-in-law to eat it!

-- Paul McVites , Sep 06, 2014

Eat it, just eat it!

-- Michael Jaikishan , Feb 14, 2015

Ultimate, Reaching for the stars!

-- Ringo Starry , Dec 03, 2013

It's your birthday, we're gonna party!

-- 25 Cent , Dec 03, 2011

**Task 1**

#### Links

[Home](#)  
[About](#)  
[Menu](#)  
[Contact](#)

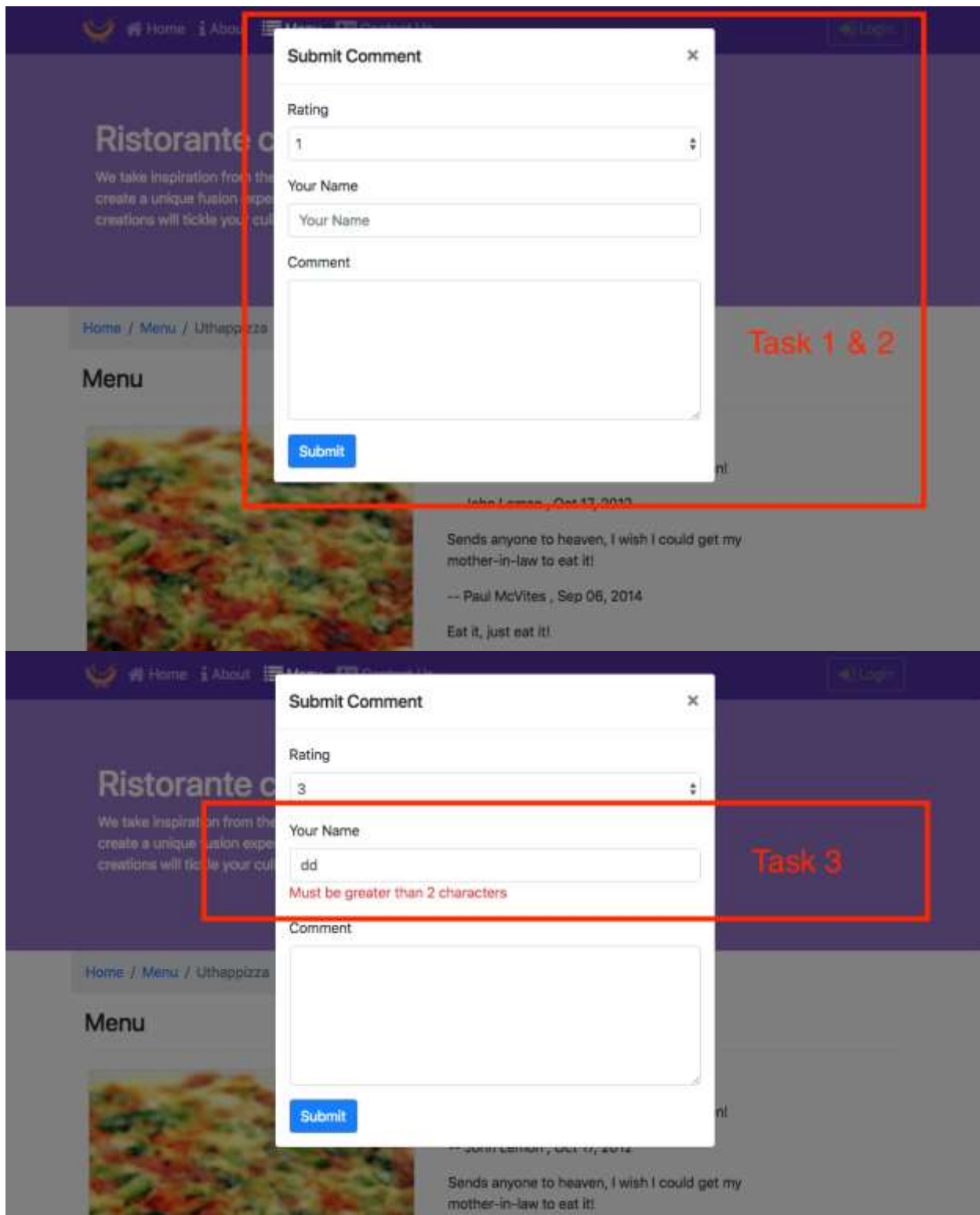
#### Our Address

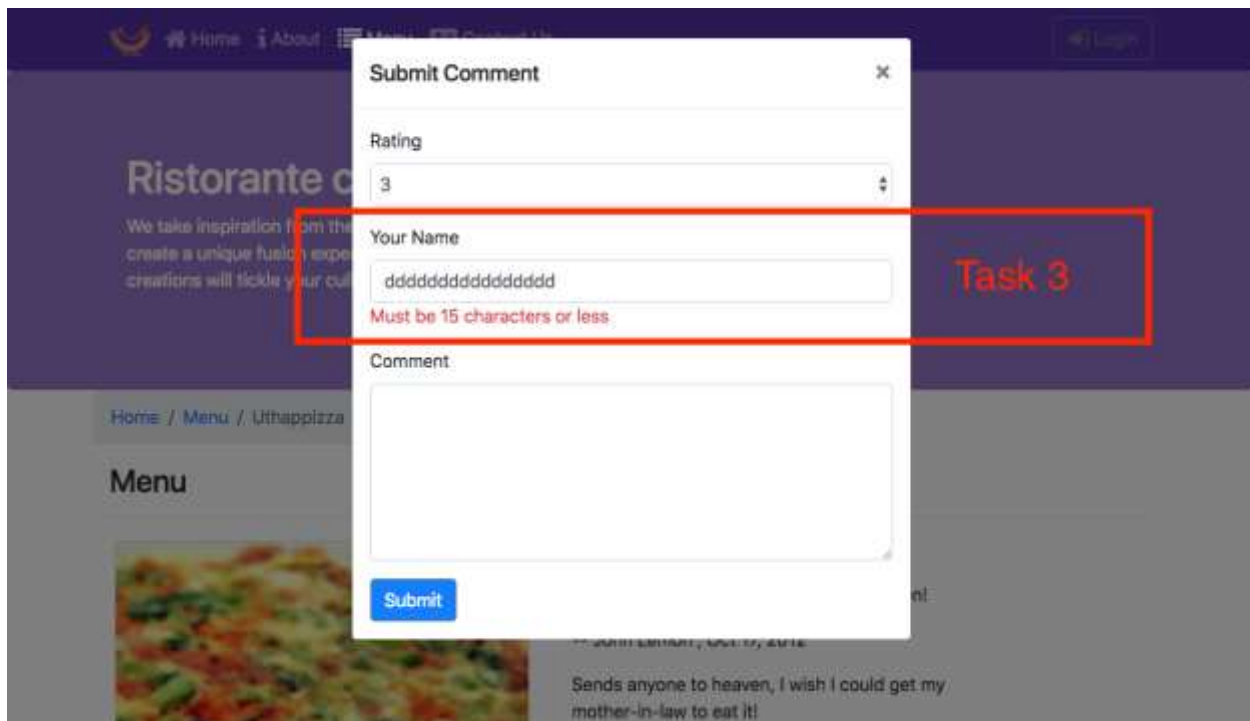
121, Clear Water Bay Road  
Clear Water Bay, Kowloon  
HONG KONG  
☎ +852 1234 5678  
📠 +852 8765 4321  
✉ [confusion@food.net](mailto:confusion@food.net)



© Copyright 2018 Ristorante Con Fusion







## React Resources

- [reactstrap Modal Component](#)
- [React Redux Form Documentation](#)

## Redux Actions: Objectives and Outcomes

In this lesson we will look at how to create redux actions and how action creators can be designed to return the action objects. Thereafter we look at how to dispatch actions. We also look at how to split the reducer into simpler functions and combine them. At the end of this exercise, you will be able to:

- Define Redux actions
- Create action creator functions that return action objects
- Split the reducer function into multiple simpler functions and combine the reducer functions

## Module xx: Redux Actions

### Module xx: Exercise (Video): Combining Reducers

### Exercise (Instructions): Combining Reducers

#### Objectives and Outcomes

In this exercise you will learn about how we can have separate reducers managing parts of the state, and how they can be combined together to manage the whole state. At the end of this exercise you will be able to:

- Implement reducers that are responsible for only part of the state
- Combine the reducers to manage the entire state

## Splitting the Reducer

- In the redux folder, create a new file named dishes.js and add the following to it:

```
import { DISHES } from '../shared/dishes';

export const Dishes = (state = DISHES, action) => {
  switch (action.type) {
    default:
      return state;
  }
};
```

- Then, create a file named comments.js and add the following to it:

```
import { COMMENTS } from '../shared/comments';

export const Comments = (state = COMMENTS, action) => {
  switch (action.type) {
    default:
      return state;
  }
};
```

- Similarly, create a new file named promotions.js and add the following to it:

```
import { PROMOTIONS } from '../shared/promotions';

export const Promotions = (state = PROMOTIONS, action) => {
  switch (action.type) {
```

```

    default:
      return state;
  }
};

```

- And finally, create a new file named `leaders.js` and add the following to it:

```

import { LEADERS } from '../shared/leaders';

export const Leaders = (state = LEADERS, action) => {
  switch (action.type) {
    default:
      return state;
  }
};

```

- Now that we have split the management of state into different reducers that manage partial state, we need to combine them together. Open `configureStore.js` and update it as follows:

```

import { createStore, combineReducers } from 'redux';
import { Dishes } from './dishes';
import { Comments } from './comments';
import { Promotions } from './promotions';
import { Leaders } from './leaders';

export const ConfigureStore = () => {
  const store = createStore(
    combineReducers({
      dishes: Dishes,
      comments: Comments,
      promotions: Promotions,
      leaders: Leaders
    })
  );
};

```

```

return store;
}
```

- Now we can safely delete the `reducer.js` file from the project.
- Save all the changes and do a Git commit with the message "Combining Reducers".

## Conclusions

In this exercise we have learnt to split the reducer into multiple reducers that manage partial state, and then combine them together.

## Module xx: Exercise (Video): Redux Actions

### Exercise (Instructions): Redux Actions

#### Objectives and Outcomes

In this exercise we will learn to define various Redux actions and implement the action creators to dispatch the actions to the Redux store. At the end of this exercise you will be able to:

- Define Redux actions and implement the action creators
- Dispatch actions from the action creators to update the system state in the Redux store

#### Creating Actions

- In the `redux` folder create a new file named `ActionTypes.js` and add the following to it:

```
export const ADD_COMMENT = 'ADD_COMMENT';
```

- Then, create a file named `ActionCreators.js` and add the following to it:

```
import * as ActionTypes from './ActionTypes';
```

```
export const addComment = (dishId, rating, author, comment) => ({
  type: ActionTypes.ADD_COMMENT,
  payload: {
    dishId: dishId,
    rating: rating,
    author: author,
```

```

    comment: comment
  }
});

```

- Next, update *comments.js* to initiate action when the action is dispatched by the *ActionCreator* as follows:

```

import { COMMENTS } from '../shared/comments';
import * as ActionTypes from './ActionTypes';

export const Comments = (state = COMMENTS, action) => {
  switch (action.type) {
    case ActionTypes.ADD_COMMENT:
      var comment = action.payload;
      comment.id = state.length;
      comment.date = new Date().toISOString();
      console.log("Comment: ", comment);
      return state.concat(comment);
    default:
      return state;
  }
};

```

- Now update *MainComponent.js* to make the action available for use within the *DishdetailComponent* as follows:

```

. . .

import { addComment } from '../redux/ActionCreators';

. . .

const mapDispatchToProps = dispatch => ({

```

```

    addComment: (dishId, rating, author, comment) => dispatch(addComment(dishId, rating, author, comment))
  };
  ...
  <DishDetail dish={this.props.dishes.filter((dish) => dish.id === parseInt(match.params.dishId,10))[0]}
    comments={this.props.comments.filter((comment) => comment.dishId === parseInt(match.params.dishId,10))}
    addComment={this.props.addComment}
  />
  ...
export default withRouter(connect(mapStateToProps, mapDispatchToProps)(Main));

```

- Finally, update *DishdetailComponent.js* as follows to initiate the action upon the user submitting the comment form:

```

  ...
  function RenderComments({comments, addComment, dishId}) {
    ...
    <CommentForm dishId={dishId} addComment={addComment} />
    ...
  }

```



```

    this.props.addComment(this.props.dishId, values.rating, values.author, values
    .comment);
}

...

<RenderComments comments={props.comments}
    addComment={props.addComment}
    dishId={props.dish.id}
/>
...

```

- Save all the changes and do a Git commit with the message "Redux Actions"

## Conclusions

In this exercise we have learnt to create and dispatch actions to update the system state in the Redux store.

## Redux Actions: Additional Resources

PDFs of Presentations

Redux Resources

- [Redux Actions](#)
- [Redux Reducers](#)
- [Redux Usage with React](#)

## Module xx: Redux Thunk: Objectives and Outcomes

Redux Thunk [middleware](#) allows you to write action creators that return a function instead of an action. In this lesson you will see the use of redux thunk to return a function. At the end of this lesson you will be able to:

- Use Redux Thunk middleware to return a function instead of an action

- Use a logger middleware to print a log of actions initiated on the Redux store.

## Redux Thunk

### Exercise (Video): Redux Thunk

Notes

### Exercise (Instructions): Redux Thunk

#### Objectives and Outcomes

Redux Thunk [middleware](#) allows you to write action creators that return a function instead of an action. In this exercise you will see the use of redux thunk to return a function. At the end of this exercise you will be able to:

- Use Redux Thunk middleware to return a function instead of an action
- Use a logger to print a log of actions initiated on the Redux store.

#### Installing Redux Thunk and Logger

- Install Redux Thunk and Logger as shown below:

```
yarn add redux-thunk@2.2.0
```

```
yarn add redux-logger@3.0.6
```

- Then open *configureStore.js* and update it to use the Thunk and Logger as follows:

```
import {createStore, combineReducers, applyMiddleware } from 'redux';  
  
...  
  
import thunk from 'redux-thunk';  
import logger from 'redux-logger';  
  
...  
  
combineReducers({
```

```

    dishes: Dishes,
    comments: Comments,
    promotions: Promotions,
    leaders: Leaders
  }},
  applyMiddleware(thunk, logger)
  ...

```

- Next, open *ActionTypes.js* and add new action types as follows:

```

...
|
|
export const DISHES_LOADING = 'DISHES_LOADING';
export const DISHES_FAILED = 'DISHES_FAILED';
export const ADD_DISHES = 'ADD_DISHES';

```

- Then open *ActionCreators.js* and add new actions:

```

...
|
|
import { DISHES } from '../shared/dishes';
|
|
...
|
|
|
export const fetchDishes = () => (dispatch) => {
|
|
  dispatch(dishesLoading(true));
|
|
  setTimeout(() => {
    dispatch(addDishes(DISHES));
  }, 2000);
}
|
|
export const dishesLoading = () => ({

```

```

    type: ActionTypes.DISHES_LOADING
  });
}

export const dishesFailed = (errmess) => ({
  type: ActionTypes.DISHES_FAILED,
  payload: errmess
});

export const addDishes = (dishes) => ({
  type: ActionTypes.ADD_DISHES,
  payload: dishes
});

```

- Next, open dishes.js and add the code to respond to actions as follows:

```

import * as ActionTypes from './ActionTypes';

export const Dishes = (state = { isLoading: true,
  errMess: null,
  dishes: [], action) => {
  switch (action.type) {
    case ActionTypes.ADD_DISHES:
      return {...state, isLoading: false, errMess: null, dishes: action.payload};
    case ActionTypes.DISHES_LOADING:
      return {...state, isLoading: true, errMess: null, dishes: []};
    case ActionTypes.DISHES_FAILED:
      return {...state, isLoading: false, errMess: action.payload};
    default:
      return state;
  }
}

```

```
};
```

- Add a new component named *LoadingComponent.js* to display a loading message as follows:

```
import React from 'react';
```

```
export const Loading = () => {
```

```
  return(
```

```
    <div className="col-12">
```

```
      <span className="fa fa-spinner fa-pulse fa-3x fa-fw text-primary"></span>
```

```
      <p>Loading . . .</p>
```

```
    </div>
```

```
  );
```

```
};
```

- Now we will update the remaining components to use the actions. First, open *MainComponent.js* and update it as follows:

```
. . .
```

```
import { addComment, fetchDishes } from '../redux/ActionCreators';
```

```
. . .
```

```
  fetchDishes: () => { dispatch(fetchDishes()) }
```

```
. . .
```

```
  componentDidMount() {
```

```
    this.props.fetchDishes();
```

```
  }
```

```
. . .
```

```
const HomePage = () => {
```

```

    return(
      <Home
        dish={this.props.dishes.dishes.filter((dish) => dish.featured)[0]}
        dishesLoading={this.props.dishes.isLoading}
        dishesErrMsg={this.props.dishes.errMess}
        promotion={this.props.promotions.filter((promo) => promo.featured)[0]}
        leader={this.props.leaders.filter((leader) => leader.featured)[0]}
      />
    );
  }
}

const DishWithId = ({match}) => {
  return(
    <DishDetail dish={this.props.dishes.dishes.filter((dish) => dish.id === parseInt(match.params.dishId,10))[0]}
      isLoading={this.props.dishes.isLoading}
      errMsg={this.props.dishes.errMess}
      comments={this.props.comments.filter((comment) => comment.dishId === parseInt(match.params.dishId,10))}
      addComment={this.props.addComment}
    />
  );
};

```

- Open *DishdetailComponent.js* and update it as follows:

```

...
import { Loading } from './LoadingComponent';
...

```

```

    if (props.isLoading) {
      return(
        <div className="container">
          <div className="row">
            <Loading />
          </div>
        </div>
      );
    }
    else if (props.errMess) {
      return(
        <div className="container">
          <div className="row">
            <h4>{props.errMess}</h4>
          </div>
        </div>
      );
    }
    else if (props.dish !== null)
    ...

```

- Open *HomeComponent.js* and update it as follows:

```

...
|
import { Loading } from './LoadingComponent';
|
...
|
|
function RenderCard({item, isLoading, errMess}) {
  if (isLoading) {

```

```

    return(
      <Loading />
    );
  }
  else if (errMess) {
    return(
      <h4>{errMess}</h4>
    );
  }
  else
    return(
      <Card>
        <CardImg src={item.image} alt={item.name} />
        <CardBody>
          <CardTitle>{item.name}</CardTitle>
          {item.designation ? <CardSubtitle>{item.designation}</CardSubtitle> :
            null }
          <CardText>{item.description}</CardText>
        </CardBody>
      </Card>
    );
  }
}
...
<RenderCard item={props.dish} isLoading={props.dishesLoading} err
Mess={props.dishesErrMess} />
...

```

- Finally, update *MenuComponent.js* as follows:

```

...

```



```

import { Loading } from './LoadingComponent';

...

const menu = props.dishes.dishes.map((dish) => {

...

if (props.dishes.isLoading) {
  return(
    <div className="container">
      <div className="row">
        <Loading />
      </div>
    </div>
  );
}

else if (props.dishes.errMess) {
  return(
    <div className="container">
      <div className="row">
        <div className="col-12">
          <h4>{props.dishes.errMess}</h4>
        </div>
      </div>
    </div>
  );
}

else
...

```

- Save all the changes and do a Git commit with the message "Redux Thunk".

## Conclusions

In this exercise we saw the use of Redux Thunk and the Logger.

## Exercise (Video): React-Redux-Form Revisited

## Exercise (Instructions): React-Redux-Form Revisited

### Objectives and Outcomes

In this exercise we will explore the interaction between react-redux-form and the Redux store. We will see how to map the form into the store so that the state of the form will be persisted in the store. At the end of this exercise you will be able to:

- Use react-redux-form to interact with Redux store and store the state of the form in the store.

### Updating the Feedback Form

- Add a new file named *forms.js* in the *redux* folder and add the following to it:

```
export const InitialFeedback = {
  firstname: '',
  lastname: '',
  telnum: '',
  email: '',
  agree: false,
  contactType: 'Tel.',
  message: ''
};
```

- Then, open *configureStore.js* and update it to add the form to the reducers:

```
. . .
|
|
import { createForms } from 'react-redux-form';
|
|
. . .
|
```

```
import { InitialFeedback } from './forms';
```

```
|
```

```
. . .
```

```
|
```

```
    combineReducers({
```

```
      dishes: Dishes,
```

```
      comments: Comments,
```

```
      promotions: Promotions,
```

```
      leaders: Leaders,
```

```
      ...createForm({
```

```
        feedback: InitialFeedback
```

```
      })
```

```
    })),
```

```
|
```

```
. . .
```

- Next, open *MainComponent.js* and update it as follows:

```
. . .
```

```
|
```

```
import { actions } from 'react-redux-form';
```

```
|
```

```
. . .
```

```
|
```

```
    resetFeedbackForm: () => { dispatch(actions.reset('feedback'))}
```

```
|
```

```
. . .
```

```
|
```

```
    <Route exact path='/contactus' component={() => <Contact resetFeedbackF  
orm={this.props.resetFeedbackForm} />} />
```

```
|
```

```
. . .
```

- Open *CommentComponent.js* and update it as follows:

```
. . .
```

```
|
```

```

import { Control, Form, Errors, actions } from 'react-redux-form';

...

handleSubmit(values) {
  console.log('Current State is: ' + JSON.stringify(values));
  alert('Current State is: ' + JSON.stringify(values));
  this.props.resetFeedbackForm();
  // event.preventDefault();
}

...

<Form model="feedback" onSubmit={(values) => this.handleSubmit(values)}>
  ...
</Form>

...

```

- Save all the changes and do a Git commit with the message "React Redux Forms Revisited".

## Conclusions

In this exercise we have seen how to use react-redux-form together with Redux to persist form state.

## Redux Thunk: Additional Resources

### PDFs of Presentations

2-Redux-Thunk.pdf

PDF File

## Redux Resources

- [Redux Middleware](#)
- [Redux Thunk](#)
- [Redux Logger](#)
- [React-redux-form](#)

## Module: Client-Server Communication: Objectives and Outcomes

In this lesson you will learn about communication between your React application and a server. You will establish a simple server using the `json-server` node module. At the end of this lesson, you will be able to:

- Set up a simple server that makes data available for clients
- Access the data from the server using a browser.
- Use the `json-server` as a simple static web server.

## Networking Essentials

### Brief Representational State Transfer (REST)

### Exercise (Video): Setting up a Server using `json-server`

### Exercise (Instructions): Setting up a Server using `json-server`

#### Exercise Resources

db

JSON File

images

ZIP File

#### Objectives and Outcomes

The Node module, `json-server`, provides a very simple way to set up a web server that supports a full-fledged REST API server. It can also serve up static web

content from a folder. This lesson will leverage these two features to provide the back-end for your React application. In this exercise, you will configure and start a server using *json-server* to enable serving your application data to your Angular application. At the end of this exercise, you will be able to:

- Configure and start a simple server using the *json-server* module
- Configure your server to serve up static web content stored in a folder named *public*.

## Installing json-server

- *json-server* is a node module, and hence can be installed globally by typing the following at the command prompt:

```
npm install json-server -g
```

If you are using OSX or Linux, use **sudo** at the front of the command. This will install *json-server* that can be started from the command line from any folder on your computer.

## Configuring the Server

- At any convenient location on your computer, create a new folder named **json-server**, and move to this folder.
- Download the *db.json* file provided above to this folder.
- Move to this folder in your terminal window, and type the following at the command prompt to start the server:

```
json-server --watch db.json -p 3001 -d 2000
```

- This should start up a server at port number 3001 on your machine. The data from this server can be accessed by typing the following addresses into your **browser address bar**:

<http://localhost:3001/dishes>

<http://localhost:3001/promotions>

<http://localhost:3001/leaders>

<http://localhost:3001/feedback>

- Type these addresses into the browser address and see the JSON data being served up by the server. This data is obtained from the *db.json* file
- The *json-server* also provides a static web server. Any resources that you put in a folder named **public** in the **json-server** folder above, will be served by the server at the following address:

<http://localhost:3001/>

- Shut down the server by typing **ctrl-C** in the terminal window.

## Serving up the Images

- Create a public folder in your json-server folder.
- Download the images.zip file that we provide above, unzip it and move the images folder containing the images to the public folder.
- Restart the json-server as we did before. Now your server will serve up the images for our React app. You can view these images by typing the following into your browser address bar:

<http://localhost:3001/images/<image name>.png>

## Conclusions

In this exercise, you learnt how to configure and start a simple server using the json-server node module. You also learnt how the server can serve up static web content.

## Client-Server Communication: Additional Resources

### PDFs of Presentations

03-Networking-Essentials.pdf

04-REST.pdf

## Exercise Resources

db

JSON File

images

ZIP File

## Other Resources

- [json-server](#)
- [Creating Demo APIs with json-server](#)
- [JSON](#)

## Fetch: Objectives and Outcomes

In this lesson you will learn about Fetch as a means of communication between your React application and a REST API server. At the end of this lesson you will be able to:

- Install Fetch in your React application
- Use Fetch to communicate from your React application with a REST API server

## Promises

## Fetch

### Exercise (Video): Fetch from Server

### Exercise (Instructions): Fetch from Server

#### Objectives and Outcomes

In this exercise you will incorporate Fetch into your React app and then use it to communicate with the REST API server. At the end of this exercise you will be able to:

- Incorporate Fetch into your React app
- Use Fetch to communicate with the REST API server

## Fetch

- As a first step, let us install Fetch into our project as follows:

```
yarn add cross-fetch@2.1.0
```

- Now that we have installed Fetch, let us configure your application to connect to the server. First, create a file named *baseUrl.js* in the *shared* folder and add the following to it:

```
export const baseUrl = 'http://localhost:3001/';
```

- Make sure that the json-server is running and serving up the data as illustrated in the previous exercise
- Next, open *ActionTypes.js* and add the following:

```
...
```

```
export const ADD_COMMENTS = 'ADD_COMMENTS';
```

```
export const COMMENTS_FAILED = 'COMMENTS_FAILED';
```

```
export const PROMOS_LOADING = 'PROMOS_LOADING';
```

```
export const ADD_PROMOS = 'ADD_PROMOS';
```



```
export const PROMOS_FAILED = 'PROMOS_FAILED';
```

- Then, open *ActionCreators.js* and update it as follows:

```
. . .
```

```
|
```

```
import { baseUrl } from '../shared/baseUrl';
```

```
|
```

```
. . .
```

```
|
```

```
    return fetch(baseUrl + 'dishes')
```

```
      .then(response => response.json())
```

```
      .then(dishes => dispatch(addDishes(dishes)));
```

```
|
```

```
. . .
```

```
|
```

```
|
```

```
export const fetchComments = () => (dispatch) => {
```

```
    return fetch(baseUrl + 'comments')
```

```
      .then(response => response.json())
```

```
      .then(comments => dispatch(addComments(comments)));
```

```
};
```

```
|
```

```
export const commentsFailed = (errmess) => ({
```

```
    type: ActionTypes.COMMENTS_FAILED,
```

```
    payload: errmess
```

```
});
```

```
|
```

```
export const addComments = (comments) => ({
```

```
    type: ActionTypes.ADD_COMMENTS,
```

```
    payload: comments
```

```
});
```

```
|
```

```
export const fetchPromos = () => (dispatch) => {
```

```
|
```

```

    dispatch(promosLoading());
  }

  return fetch(baseUrl + 'promotions')
    .then(response => response.json())
    .then(promos => dispatch(addPromos(promos)));
}

export const promosLoading = () => ({

```

- Next, open *comments.js* and update it as follows:

```

import * as ActionTypes from './ActionTypes';

export const Comments = (state = { errMsg: null, comments: [], action } => {
  switch (action.type) {
    case ActionTypes.ADD_COMMENTS:
      return {...state, errMsg: null, comments: action.payload};

    case ActionTypes.COMMENTS_FAILED:
      return {...state, errMsg: action.payload};

    case ActionTypes.ADD_COMMENT:
      var comment = action.payload;
      comment.id = state.comments.length;
      comment.date = new Date().toISOString();
      return { ...state, comments: state.comments.concat(comment)};

    default:
      return state;
  }
});

```

- Similarly, open *promotions.js* and update it as follows:

```

import * as ActionTypes from './ActionTypes';

```

```

export const Promotions = (state = { isLoading: true,
                                errMsg: null,
                                promotions: [] }, action) => {
  switch (action.type) {
    case ActionTypes.ADD_PROMOS:
      return { ...state, isLoading: false, errMsg: null, promotions: action.payload };
    case ActionTypes.PROMOS_LOADING:
      return { ...state, isLoading: true, errMsg: null, promotions: [] };
    case ActionTypes.PROMOS_FAILED:
      return { ...state, isLoading: false, errMsg: action.payload };
    default:
      return state;
  }
};

```

- Now that the Redux actions are all updated, it's time to update the components.
- Open *MainComponent.js* and update it as follows:

```

...

import { addComment, fetchDishes, fetchComments, fetchPromos }
  from '../redux/ActionCreators';

...

const mapDispatchToProps = dispatch => ({
  addComment: (dishId, rating, author, comment) =>
    dispatch(addComment(dishId, rating, author, comment)),

```

```

    fetchDishes: () => { dispatch(fetchDishes())},
    resetFeedbackForm: () => { dispatch(actions.reset('feedback'))},
    fetchComments: () => dispatch(fetchComments()),
    fetchPromos: () => dispatch(fetchPromos())
  });
  |
  |
  | . . .
  |
  |
  | componentDidMount() {
  |   this.props.fetchDishes();
  |   this.props.fetchComments();
  |   this.props.fetchPromos();
  | }
  |
  | . . .
  |
  | <Home
  |   dish={this.props.dishes.dishes.filter((dish) => dish.featured)[0]}
  |   dishesLoading={this.props.dishes.isLoading}
  |   dishErrMess={this.props.dishes.errMess}
  |   promotion={this.props.promotions.promotions.filter((promo) =>
  |     promo.featured)[0]}
  |   promoLoading={this.props.promotions.isLoading}
  |   promoErrMess={this.props.promotions.errMess}
  |   leader={this.props.leaders.filter((leader) => leader.featured)[0]}
  | />
  |
  | . . .
  |
  | <DishDetail dish={this.props.dishes.dishes.filter((dish) =>
  |   dish.id === parseInt(match.params.dishId,10))[0]}
  |   isLoading={this.props.dishes.isLoading}
  |   errMess={this.props.dishes.errMess}

```

- Then, open *MenuComponent.js* and update it as follows:

```

. . .
|
import { baseUrl } from '../shared/baseUrl';
|
. . .
|
<CardImg width="100%" src={baseUrl + dish.image} alt={dish.name}
/>
|
. . .

```

- Then, open *HomeComponent.js* and update it as follows:

```

. . .
|
import { baseUrl } from '../shared/baseUrl';
|
. . .
|
<CardImg src={baseUrl + item.image} alt={item.name} />
|
. . .
|
<RenderCard item={props.promotion} isLoading={props.promoLoading}
errMess={props.promoErrMess} />
|
. . .

```

- Then, open *DishdetailComponent.js* and update it as follows:

```

. . .
|
import { baseUrl } from '../shared/baseUrl';
|
. . .

```

```
|
```

```
<CardImg top src={baseUrl + dish.image} alt={dish.name} />
```

```
|
```

```
. . .
```

- Save all the changes and do a Git commit with the message "Fetch from Server".

## Conclusions

In this exercise you have learnt to install Fetch and use it communicate with the server.

## Exercise (Video): Fetch Handling Errors

## Exercise (Instructions): Fetch Handling Errors

### Objectives and Outcomes

In this exercise you will learn how to handle errors encountered while communicating with the server. At the end of this exercise you will be able to:

- Configure your app to appropriately handle errors encountered while communicating with the server

### Handling Errors

- Open *ActionCreators.js* and update it as follows:

```
. . .
```

```
export const fetchDishes = () => (dispatch) => {
```

```
|
```

```
  dispatch(dishesLoading(true));
```

```
|
```

```
  return fetch(baseUrl + 'dishes')
```

```
    .then(response => {
```

```
      if (response.ok) {
```

```
        return response;
```

```
      } else {
```

```
        var error = new Error('Error ' + response.status + ': ' + response.statusText);
```

```

        error.response = response;
        throw error;
    }
},
    error => {
        var errmess = new Error(error.message);
        throw errmess;
    })
    .then(response => response.json())
    .then(dishes => dispatch(addDishes(dishes)))
    .catch(error => dispatch(dishesFailed(error.message)));
}
|
...
|
export const fetchComments = () => (dispatch) => {
    return fetch(baseUrl + 'comments')
        .then(response => {
            if (response.ok) {
                return response;
            } else {
                var error = new Error('Error ' + response.status + ': ' + response.statusText);
                error.response = response;
                throw error;
            }
        }),
    error => {
        var errmess = new Error(error.message);

```

- Save all the changes and do a Git commit with the message "Fetch Handling Errors".

## Conclusions

In this exercise you learnt how to configure your app to handle errors in communicating with the server.

## Exercise (Video): Fetch Post Comment

## Exercise (Instructions): Fetch Post Comment

### Objectives and Outcomes

In this exercise you will learn how to configure Fetch to be able to post data to the server. At the end of this exercise, you will be able to:

- Configure Fetch to post data to the server
- Receive and process the response to the POST operation on the server

### Posting a Comment

- Open *ActionCreators.js* and update it as follows:

```
. . .  
  
export const addComment = (comment) => ({  
  type: ActionTypes.ADD_COMMENT,  
  payload: comment  
});  
  
export const postComment = (dishId, rating, author, comment) => (dispatch) => {  
  
  const newComment = {  
    dishId: dishId,  
    rating: rating,  
    author: author,  
    comment: comment  
  };  
  
  newComment.date = new Date().toISOString();  
  
  return fetch(baseUrl + 'comments', {  
    method: "POST",  
    body: JSON.stringify(newComment),  
    headers: {
```



```

    "Content-Type": "application/json"
  },
  credentials: "same-origin"
})
.then(response => {
  if (response.ok) {
    return response;
  } else {
    var error = new Error('Error ' + response.status + ': ' + response.statusText);
    error.response = response;
    throw error;
  }
},
error => {
  throw error;
})
.then(response => response.json())
.then(response => dispatch(addComment(response)))
.catch(error => { console.log('post comments', error.message); alert('Your comment could not be posted\nError: ' + error.message); });

```

- Open *comment.js* and remove the following two lines from it:

```

...

comment.id = state.comments.length;
comment.date = new Date().toISOString();
...

```

- Open *MainComponent.js* and update it as follows:

```

...

import { postComment, fetchDishes, fetchComments, fetchPromos } from '../redux/ActionCreators';

```

```

    ...
  ]
  postComment: (dishId, rating, author, comment) => dispatch(postComment(dishId, rating, author, comment))
]
...
]
  postComment={this.props.postComment}
]
...

```

- Finally, open *DishdetailComponent.js* and update it as follows:

```

...
]
function RenderComments({comments, postComment, dishId}) {
  ...
  ...
  <CommentForm dishId={dishId} postComment={postComment} />
  ...
  ...
  this.props.postComment(this.props.dishId, values.rating, values.author, values.comment);
  ...
  ...
  postComment={props.postComment}
  ...
  ...

```

- Save all the changes and do a Git commit with the message "Fetch Post Comment".

## Conclusions

In this exercise you learnt to use Fetch to post data to the server.

## Fetch: Additional Resources

### PDFs of Presentations

5-Promises.pdf

6-Fetch.pdf

### Fetch Resources

- [Cross-Fetch](#)

### Promise Resources

- [JavaScript Promise](#)
- [JS Promise \(Part 1 - Basics\)](#)
- [JavaScript Promises for Dummies](#)
- [JavaScript Promises: an Introduction](#)

### Other Resources

- [Introduction to fetch\(\)](#)
- [Using Fetch](#)
- [Fetch vs. Axios.js for making http requests](#)

## Module xx: React Animations: Objectives and Outcomes

In this lesson we will learn about adding various subtle animations to our React app for a better user experience. At the end of this lesson you will be able to:

- Add subtle animations using the react-transition-group
- Add additional component animations using react-animation-components

### React Animations

#### Exercise (Video): React Animations

#### Exercise (Instructions): React Animations

## Objectives and Outcomes

In this exercise you will learn to implement animations in your React app using react-transition-group. At the end of this exercise you will be able to:

- Configure your app to use react-transition-group for animations
- Implement simple animation using the react-transition-group

## Installing React-Transition-Group

- Install react-transition-group in your React project as follows:

```
yarn add react-transition-group@2.3.0
```

- Configure CSS classes for use in animation. Open *App.css* and add the following classes:

```
. . .  
  
.  
  
.page-enter {  
  opacity: 0.01;  
  transform: translateX(-100%);  
}  
  
.  
  
.page-enter-active {  
  opacity: 1;  
  transform: translateX(0%);  
  transition: all 300ms ease-in;  
}  
  
.  
  
.page-exit {  
  opacity: 1;  
  transform: translateX(0%);  
}  
  
.  
  
.page-exit-active {  
  opacity: 0.01;  
  transform: translateX(100%);  
  transition: all 300ms ease-out;  
}
```

- Then, open MainComponent.js and add in the following to configure the animation:

```

. . .
|
import { TransitionGroup, CSSTransition } from 'react-transition-group';
|
. . .
|
<TransitionGroup>
  <CSSTransition key={this.props.location.key} classNames="page" timeout={300}>
    <Switch location={this.props.location}>
      <Route path="/home" component={HomePage} />
      <Route exact path="/aboutus" component={() => <About leaders={this.props.leaders} />} /> /> />
      <Route exact path="/menu" component={() => <Menu dishes={this.props.dishes} />} />
      <Route path="/menu/:dishId" component={DishWithId} />
      <Route exact path="/contactus" component={() => <Contact resetFeedbackForm={this.props.resetFeedbackForm} />} />
      <Redirect to="/home" />
    </Switch>
  </CSSTransition>
</TransitionGroup>
. . .

```

- Save all the changes and do a Git commit with the message "React Animations".

## Conclusions

In this exercise we implemented simple animation using react-transition-group.

## Exercise (Video): React Animation Components

## Exercise (Instructions): React Animation Components

## Objectives and Outcomes

In this exercise you will learn to use react-animation-components to add more subtle animations to your React app. At the end of this exercise you will be able to:

- Use react-animation-components to add more subtle animations to your React app.

## Installing React-Animation-Components

- Install react-animation-components into your React app as follows:

```
yarn add react-animation-components@3.0.0
```

```
yarn add prop-types@15.6.0
```

## Adding Animations

- Open *HomeComponents.js* and update as follows:

```
...  
  
import { FadeTransform } from 'react-animation-components';  
  
...  
  
    <FadeTransform  
      in  
      transformProps={{  
        exitTransform: 'scale(0.5) translateY(-50%)'  
      }}>  
      <Card>  
        <CardImg src={baseUrl + item.image} alt={item.name} />  
        <CardBody>  
          <CardTitle>{item.name}</CardTitle>  
          {item.designation ? <CardSubtitle>{item.designation}</CardSubtitl  
e> : null }  
          <CardText>{item.description}</CardText>  
        </CardBody>  
      </Card>
```

```
</FadeTransform>
```

```
. . .
```

- Open DishdetailComponents.js and update it as follows:

```
. . .
```

```
import { FadeTransform, Fade, Stagger } from 'react-animation-components';
```

```
. . .
```

```
<FadeTransform
```

```
in
```

```
transformProps={{
```

```
  exitTransform: 'scale(0.5) translateY(-50%)'
```

```
}}>
```

```
<Card>
```

```
<CardImg top src={baseUrl + dish.image} alt={dish.name} />
```

```
<CardBody>
```

```
<CardTitle>{dish.name}</CardTitle>
```

```
<CardText>{dish.description}</CardText>
```

```
</CardBody>
```

```
</Card>
```

```
</FadeTransform>
```

```
. . .
```

```
<Stagger in>
```

```
{comments.map((comment) => {
```

```
  return (
```

```
<Fade in>
```

```
<li key={comment.id}>
```

```
<p>{comment.comment}</p>
```

```

        <p>-
- {comment.author} , {new Intl.DateTimeFormat('en-
US', { year: 'numeric', month: 'short', day: '2-
digit'}).format(new Date(Date.parse(comment.date)))}</p>
    </li>
</Fade>
);
}}}
</Stagger>
...

```

- Save all the changes and do a Git commit with the message "React Animation Components".

## Conclusions

In this exercise you saw yet another way of adding subtle animations using react-animation-components.

## React Animations: Additional Resources

### PDFs of Presentations

7-Animations.pdf

PDF File

### React Animations

- [Animation Add-Ons](#)
- [react-transition-group](#)
- [React Transition Group Documents](#)
- [react-animation-components](#)

### Other Resources

- [How to build animated microinteractions in React](#)
- [UI Animations with React—The Right Way](#)
- [React Animations in Depth](#)
- [What's the most developer-friendly React animation library?](#)
- [Amazing React animation with react-pose](#)



# Assignment 4: Redux, Client-Server Communication and Fetch

## Peer-graded Assignment: Redux, Client-Server Communication and Fetch

Ready for the assignment?

You will find instructions below to submit.

In this assignment, you will update the web application to get data from the server to render the information corresponding to the leadership team of the company. In addition, you will handle the submission of the feedback form by posting the feedback to the server.

### Step-By-Step Assignment Instructions

[less](#)

#### Assignment Overview

At the end of this assignment, you should have completed the following:

- Introduced new action types and action creators to support the fetching of the leaders information from the server and update the Redux store.
- Updated the Home and the About component to render the information about the leaders using the downloaded data from the server
- Add simple animations to the About component where the leaders information is displayed.
- Enabled the users to submit feedback through the feedback form by creating a new feedback service that accepts the form data and uses Restangular to record their feedback on the server.

#### Assignment Requirements

##### Task 1

In this task, you will update the Redux actions and the Home and About components to use the data from the server for rendering the leader information:

- Add new action types in *ActionTypes.js* to support the fetching of the leaders information from the server
- Add new action creators in *ActionCreators.js* to enable the fetching of the leaders information from the server and update the Redux store

- Update the code in *leaders.js* to respond to the dispatched Redux actions and update the Redux store and appropriately handle the loading and errors.
- Update the code in *MainComponent.js* to fetch and use the leaders information.
- Update *HomeComponent.js* to render the leader information.
- Update *AboutComponent.js* to render the leaders information. You should handle the loading and error condition appropriately.

## Task 2

In this task, you will enable the saving of the feedback data submitted using the feedback form in the Contact component. You will save the feedback form data submitted by the user to the server:

- Implement a new action creator named `postFeedback()` that takes a Feedback object as a parameter and submits the feedback to the server using Fetch. Recall that the feedback data is accessible at <http://localhost:3001/feedback> on the json-server.
- Update *MainComponent.js* to make the new dispatch method `postFeedback()` available to `ContactComponent`.
- Update the *ContactComponent.js* to submit the form data using the `postFeedback()` method by passing the feedback form data.

## Task 3

In this task you will use simple animation using `react-animation-components` to enable a staggered rendering of the list of leaders in `AboutComponent`:

- Use the expand animation that we have already used earlier to judiciously apply animation to the various stages of the form submission.

## Screenshots



## Ristorante con Fusion

We take inspiration from the World's best cuisines, and create a unique fusion experience. Our lipsmacking creations will tickle your culinary senses!

### Task 1



#### Uthappizza

A unique combination of Indian Uthappam (pancake) and Italian pizza, topped with Cerignola olives, ripe vine cherry tomatoes, Vidalia onion, Guntur chillies and Buffalo Paneer.



#### Weekend Grand Buffet

Featuring mouthwatering combinations with a choice of five different salads, six enticing appetizers, six main entrees and five choicest desserts. Free flowing bubbly and soft drinks. All for just \$19.99 per person



#### Alberto Somayya

**Executive Chef**  
Award winning three-star Michelin chef with wide International experience having worked closely with whos-who in the culinary world, he specializes in creating mouthwatering Indo-Italian fusion experiences. He says, Put together the cuisines from the two craziest cultures, and you get a winning hit! Amma Mia!

#### Links

[Home](#)  
[About](#)  
[Menu](#)  
[Contact](#)

#### Our Address

121, Clear Water Bay Road  
Clear Water Bay, Kowloon  
HONG KONG  
☎ +852 1234 5678  
✉ +852 8765 4321  
✉ [confusion@food.net](mailto:confusion@food.net)



© Copyright 2018 Ristorante Con Fusion

## Location Information

### Our Address

121, Clear Water Bay Road  
Clear Water Bay, Kowloon  
HONG KONG  
☎: +852 1234 5678  
📠: +852 8765 4321  
✉: confusion@food.net

jogeshs-imac.ust.hk:3000 says

Thank you for your feedback!

```
{
  "firstname": "Jogesh",
  "lastname": "Muppala",
  "telnum": "12345678",
  "email": "abc@def.gh",
  "agree": true,
  "contactType": "Email",
  "message": "testing",
  "id": 5
}
```

OK

Task 2

📞 Call    💬 Skype    ✉ Email

## Send us your Feedback

First Name

First Name

Last Name

Last Name

Tel. Number

Tel. Number

Email

Email

☐ May we contact you?

Tel.

Your Feedback

Send Feedback

Cancel

Links

Our Address

[Home](#) / [About Us](#)

## About Us

### Our History

Started in 2010, Ristorante con Fusion quickly established itself as a culinary icon par excellence in Hong Kong. With its unique brand of world fusion cuisine that can be found nowhere else, it enjoys patronage from the A-list clientele in Hong Kong. Featuring four of the best three-star Michelin chefs in the world, you never know what will arrive on your plate the next time you visit us.

The restaurant traces its humble beginnings to The Flying Pan, a successful chain started by our CEO, Mr. Peter Pan, that featured for the first time the world's best cuisines in a pan.

### Facts At a Glance

<b>Started</b>	3 Feb. 2013
<b>Major Stake Holder</b>	HK Fine Foods Inc.
<b>Last Year's Turnover</b>	\$1,250,375
<b>Employees</b>	40

You better cut the pizza in four pieces because I'm not hungry enough to eat six.  
 — Yogi Berra, *The Wit and Wisdom of Yogi Berra*, *It Pays*, *Diversion Books*, 2014

### Corporate Leadership



**Peter Pan**  
 Chief Epicurious Officer

Our CEO, Peter, credits his hardworking East Asian immigrant parents who undertook the arduous journey for the pleasure of providing excellent food and service to their children. He is a devoted family man and a successful businessman.

## Task 3

### Links

[Home](#)  
[About](#)  
[Menu](#)  
[Contact](#)

### Our Address

121, Clear Water Bay Road  
 Clear Water Bay, Kowloon  
 HONG KONG  
 ☎ +852 1234 5678  
 ✉ [info@confusion.com](mailto:info@confusion.com)  
 🌐 [confusion.com](http://confusion.com)

© Copyright 2018 Ristorante Con Fusion



## Review criteria

less

Your assignment will be assessed based on the following criteria:

### Task 1

- Appropriate action types and action creators have been added.
- The Home component is correctly using the leader data, and handling any errors that might arise.
- The About component is correctly using the leader data, and handling any errors that might arise.

### Task 2

- A new `postFeedback()` action creator is correctly implemented to post the feedback data to the server.
- The Contact component has been correctly updated to use `postFeedback()` to post the form data to the server.

### Task 3

- Appropriate animation has been added to stagger the rendering of the leaders in the AboutComponent.

## Assignment 4: Redux, Client-Server Communication and Fetch: Additional Resources

Assignment Screenshots

[Home](#)[About](#)[Menu](#)[Contact Us](#)[Login](#)

## Ristorante con Fusion

We take inspiration from the World's best cuisines, and create a unique fusion experience. Our lipsmacking creations will tickle your culinary senses!

### Task 1



#### Uthappizza

A unique combination of Indian Uthappam (pancake) and Italian pizza, topped with Cerignola olives, ripe vine cherry tomatoes, Vidalia onion, Guntur chillies and Buffalo Paneer.



#### Weekend Grand Buffet

Featuring mouthwatering combinations with a choice of five different salads, six enticing appetizers, six main entrees and five choicest desserts. Free flowing bubbly and soft drinks. All for just \$19.99 per person



#### Alberto Somayya

Executive Chef

Award winning three-star Michelin chef with wide International experience having worked closely with whos-who in the culinary world, he specializes in creating mouthwatering Indo-Italian fusion experiences. He says, Put together the cuisines from the two craziest cultures, and you get a winning hit! Amma Mia!

#### Links

[Home](#)[About](#)[Menu](#)[Contact](#)

#### Our Address

121, Clear Water Bay Road

Clear Water Bay, Kowloon

HONG KONG

☎ +852 1234 5678

☎ +852 8765 4321

✉ [confusion@food.net](mailto:confusion@food.net)



© Copyright 2018 Ristorante Con Fusion

## Location Information

### Our Address

121, Clear Water Bay Road  
Clear Water Bay, Kowloon  
HONG KONG  
☎: +852 1234 5678  
📠: +852 8765 4321  
✉: confusion@food.net

jogeshs-imac.ust.hk:3000 says

Thank you for your feedback!

```
{
  "firstname": "Jogesh",
  "lastname": "Muppala",
  "telnum": "12345678",
  "email": "abc@def.gh",
  "agree": true,
  "contactType": "Email",
  "message": "testing",
  "id": 5
}
```

OK

Task 2

📞 Call    💬 Skype    ✉ Email

## Send us your Feedback

First Name

First Name

Last Name

Last Name

Tel. Number

Tel. Number

Email

Email

☐ May we contact you?

Tel.

Your Feedback

Send Feedback

Cancel

Links

Our Address



## Ristorante con Fusion

We take inspiration from the World's best cuisines, and create a unique fusion experience. Our lip-smacking creations will tickle your culinary senses!

Home / About Us

### About Us

#### Our History

Started in 2010, Ristorante con Fusion quickly established itself as a culinary icon par excellence in Hong Kong. With its unique brand of world fusion cuisine that can be found nowhere else, it enjoys patronage from the A-list clientele in Hong Kong. Featuring four of the best three-star Michelin chefs in the world, you never know what will arrive on your plate the next time you visit us.

The restaurant traces its humble beginnings to The Flying Pan, a successful chain started by our CEO, Mr. Peter Pan, that featured for the first time the world's best cuisines in a pan.

#### Facts At a Glance

Started	3 Feb. 2013
Major Stake Holder	HK Fine Foods Inc.
Last Year's Turnover	\$1,250,375
Employees	40

You better cut the pizza in four pieces because I'm not hungry enough to eat six.  
— Yogi Berra, *The Wit and Wisdom of Yogi Berra, R. Pepi, Diversion Books, 2014*

#### Corporate Leadership



##### Peter Pan

Chief Epicurious Officer

Our CEO, Peter, credits his hardworking East Asian immigrant parents who undertook the arduous journey for the pleasure of American cuisine as his inspiration and who inspired the inspiration of Ristorante con Fusion.

## Task 3

#### Links

[Home](#)  
[About](#)  
[Menu](#)  
[Contact](#)

#### Our Address

121, Chest Water Bay Road  
Chest Water Bay, Kowloon  
+852 0000  
+852 1234 5678  
+852 8700 4321  
[confusion@hkd.com](mailto:confusion@hkd.com)



## React Resources

- [Redux Actions](#)
- [Redux Reducers](#)
- [Redux Usage with React](#)
- [Cross-Fetch](#)
  
- [Animation Add-Ons](#)
- [react-transition-group](#)
- [React Transition Group Documents](#)
- [react-animation-components](#)

## Module xx: Building and Deployment: Objectives and Outcomes

In this lesson you will learn about Webpack and how react-scripts uses webpack to package your React application to create a distribution folder.

At the end of this exercise you will be able to:

- Understand the Webpack way of packaging applications into bundles
- Use react-scripts to build a distribution folder with your React application bundled using Webpack

## Introduction to Webpack

### Exercise (Video): Building and Deploying the React Application

### Exercise (Instructions): Building and Deploying the React Application

#### Objectives and Outcomes

In this exercise you will learn to use the react-scripts to build a distribution folder with the set of application files that can be copied to a server to deploy your React application.

At the end of this exercise you will be able to:

- Build your React application using the react-scripts to create a distribution folder
- Deploy your application to a server by copying the built files to your server

## Building the Distribution Folder

- To build the distribution folder containing your React application, type the following at the prompt:

```
npm run build
```

- This should build a distribution folder named *build* containing all your application files.

## Deploying your React Application

- To deploy your React application you need a server. Fortunately we already have the json-server available on our computer.
- Copy the contents of the build folder to the public folder of your json-server
- Now your React application can be accessed at the link <http://localhost:3001/>.
- If you are setting up a server on the cloud or anywhere, all that you need to do is copy the contents of the build folder to the server side to deploy your React application. The exact procedure depends on the cloud service provider that you choose to use. Please consult their documentation to see the procedure to set up the server.

## Conclusions

In this exercise you learnt to use the react-scripts to build and deploy your React application.

## Building and Deployment: Additional Resources

### PDFs of Presentations

8-Webpack.pdf

PDF File

### React Resources

- [Webpack: an Introduction](#)
- [create-react-app Build](#)

### Other Resources

- [Webpack](#)
- [Webpack on Github](#)

## Project Implementation: Objectives and Outcomes

This is the final stretch before you complete your project.

By this time, you should already have a good scaffolding for your website.

As you flesh out your project and race towards completion, it is time to pause and take stock of the current situation.

Perhaps a critical look back at the past, with the view of learning from our experience and consolidating this learning into an effective, organized and repeatable process is in order.

Upon completion of the project it is important not only to demonstrate the working project, but also summarize the process of reaching the final goal.

At the end of this lesson, you will be able to:

- Document the process of starting from an idea and reaching the conclusion of the project, not just the implementation, but also recognizing the process of reaching the end.
- Learn lessons from the process in understanding what worked and what did not, and being able to make intelligent choices in the future based on the experience
- Understand the design and development process through the practice.

## Final Report Template

Project Title

### 1. Introduction

- Briefly describe the salient features of your project.

### 2. Design and Implementation

Give a detailed system description and design and implementation details.

In particular, this section should contain:

- Details of how you converted from design to the actual realization of your project in terms of implementing the code.

- Any choices that you made, and any modifications that you made to the design, in response to difficulties that you might have encountered while implementing the project.
- A brief discussion of various modules and libraries that you used in implementing your project. In particular highlight the reasons for your choices briefly.
- Include a few screen shots of your website in the report

### 3. Conclusions

- Briefly state what results you obtained from your project.
- Discuss any features and shortcomings of the project.
- Discuss any choices that you might have made differently, in hindsight after completing the project.

### 4. References

- Give references to any material / websites / books etc. relevant to your project.

## **Honors Peer-graded Assignment: Project Implementation and Final Report**

Deadline Nov 13, 11:59 PM PST

Ready for the assignment?

You will find instructions below to submit.

It's finally time to submit your Capstone project for evaluation.

You need to write a final report giving some details about your implementation as per the specification in the final report template.

In addition, you will also submit your source code after uploading it to any suitable site like Github, Bitbucket or others that can be publicly accessed by providing the URL of the site.

Make sure that you have sufficiently documented the implementation of your project code by adding appropriate comments within the code so that reviewers can easily understand the code.

If your server is running on any publicly hosted server, please submit the URL of the site

## **Review criteria**

Your final project submission will be evaluated by your peers considering the following items:

- Have the details of the implementation been briefly described in the final report?
- Have any justifications for the choices made been provided in the final report?
- Has the source code been made available for review?
- Has the URL to the server where the project is running been provided?

1-18-23 11:30pm

..the end.

## Free Courses

- **Glitch: React Starter Kit** - A free, 5-part video course with interactive code examples that will help you learn React.
- **Codecademy: React 101** - Codecademy's introductory course for React.
- **Egghead.io: Start Learning React** - This series will explore the basic fundamentals of React to get you started.
- **React Crash Course 2018** - A beginner-friendly crash course through the most important React topics.
- **Frontend Armory: React Fundamentals** - Learn React without the buzzwords.
- **Egghead.io: The Beginner's Guide to ReactJS** - Free course for React newbies and those looking to get a better understanding of React fundamentals.
- **Free React Bootcamp** - Recordings from three days of a free online React bootcamp.
- **Scrimba: Learn React for free** - 48 hands-on video tutorials building react apps.
- **University of Helsinki: Full Stack Open MOOC** - Learn to build web applications with React. Available in English, Spanish, Chinese and Finnish.