**Setting up Project Dependencies**

We want to setup our Development Environment. We're going to have a Node Web server and going to write React components for the Front End. The Node Server will then render initial static HTML from the our Front End React components. We also want a Node API Server to talk to the data-base. We will create one Node project to perform all 3 tasks.

Check out article as jsComplete.com/reactfull

Create package.json file to store general info about the project and track its dependencies - npm init

There's two types of packages, the main dependencies package is used in production. Dev dependencies are only used in a local development environment.

Main dependencies, to create a Node Sever we will use Express js – npm install Express

Mongo DB – to connect from Node we need a driver. The official driver – npm install mongodb

Front End dependency will be React – npm install react react-dom. The React library will be used to describe the UI and React Dom to render both those UI's on the front and back end.

Now our Dev dependencies…

Npm install -D (the capital D marks the dependency as a Dev dependency)

Webpack and Webpack CLI

Since we're going to be writing modular code everywhere we need a tool to translate modular code into something Browsers understand. Webpack is the most popular tool for that job.

**Modular programming is** a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

Express is serving the public files directly with its static middleware

Api folder is for the back-end Server

**Course details**

Learn by doing! Learn full-stack JavaScript development by building a web application with MongoDB, Node.js, and React.js.  
  
Learn how to use Node as a web server and an API server, how to consume data and build user interfaces with React, and how to read and write data with a MongoDB database. React allows you to build full-featured, data-driven applications faster and more efficiently than what is possible with JavaScript alone.  
  
Samer Buna introduces modern JavaScript programming concepts, with an emphasis on functional methods and open-source tools.

**Learning objectives**

* Using modern JavaScript features
* Configuring webpack and Babel
* Working with native Node modules
* Creating an Express server
* Working with React components and JSX
* Loading and working with test mock data
* Fetching data from a remote API
* Working with data in MongoDB
* Isomorphic rendering on the server
* Front-end routing and back-end routing

## **The HTTP/HTTPS modules**

## **Using the HTTP module as both a Browser (Client) and Server Side**

## The HTTP modules are Core modules so you do not have to import them.

## Request or get method to get data. You can pipe | the output to paginate it.

import https from 'https';

https.get('https://www.lynda.com', res => {

console.log('Response status code:', res.statusCode);

});

**Using the HTTP Module as a Server**

import https from 'https';

//After we import it, we use createServer call to create the Server

const server = http.createServer();

// Use the listen method to run the Server on a certain port

server.listen(8080);

// Create Server gives us an event emitter object which we can subscribe to. The most important emitted event is the request event. The callback for this event receives two special objects (req, res). A request object and a response object. The user initiating the request will see anything we write to the response object. While we can use the request object to read things from the request. For example, what url was requested? Were there any query string parameters with the request and many other things. Let's write a simple hello http string back to the requester. The response object is a writable stream, which means we can use it to stream data to the user and that's extremely powerful. Let's write another line to the user after three seconds using a timer like this and after that let's go ahead and terminate the stream response with the .end call.

server.on('request', (req,res) => {

})