Glossary: Getting Started with Git and GitHub

Welcome! This alphabetized glossary contains many of the terms in this course. This comprehensive glossary also includes additional industry-recognized terms not used in course videos. These terms are essential for you to recognize when working in the industry, participating in user groups, and in other professional certificate programs.

Estimated reading time: 5 minutes

Term	Definition
Branch	A separate line of development that allows to work on features or fixes independently.
Clone	A local copy of the remote Git repository on the computer.
Cloning	A process of creating a copy of the project's code and its complete version history from the remote repository on the local machine.
Commit	A snapshot of the project's current state at a specific point in time, along with a description of the changes made.
Continuous delivery (CD)	The automated movement of software through the software development lifecycle.
Continuous integration (CI)	A software development process in which developers integrate new code into the code base at least once a day.
Developer	A computer programmer who is responsible for writing code.
Distributed version control system (DVCS)	A system that keeps track of changes to code, regardless of where it is stored. It allows multiple users to work on the same codebase or repository, mirroring the codebase on their computers if needed, while the distributed version control software helps manage synchronization amongst the various codebase mirrors.
Fork	A copy of a repository. You can fork a repository to use it as the base for a new project or to work on a project independently.
Forking	Forking creates a copy of a repository on which one can work without affecting the original repository.

Term	Definition
Git	Free and open-source software distributed under the GNU General Public License. It is a distributed version control system that allows users to have a copy of their own project on their computer anywhere in the world.
GitHub	A web-hosted service for the Git repository.
GitHub branches	A branch stores all files in GitHub. Branches are used to isolate changes to code. When the changes are complete, they can be merged back into the main branch.
GitLab	A complete DevOps platform delivered as a single application. It provides access to Git repositories, controlled by source code management.
Integrator	A role that is responsible for managing changes made by developers.
Master branch	A branch that stores the deployable version of your code. The master branch is created by default and is definitive.
Merge	A process to combine changes from one branch to another, typically merging a feature branch into the main branch.
Origin	A term that refers to the repository where a copy is cloned from.
Pull request	A process used to request that someone reviews and approves your changes before they become final.
Remote repositories	Repositories that are stored elsewhere. They can exist on the internet, on your network, or on your local computer.
Repository	A data structure for storing documents, including application source code. It contains the project folders that are set up for version control.
Repository administrator	A role that is responsible for configuring and maintaining access to the repository.
SSH protocol	A method for secure remote login from one computer to another.
Staging area	An area where commits can be formatted and reviewed before completing the commit.
Upstream	A term used by developers to refer to the original source where the local copy was cloned from.
Version control	A system that allows you to keep track of changes to your documents. This process allows you to recover older versions of the documents if any mistakes are made.

Term	Definition
Working directory	A directory in your file system that contains files and subdirectories on your computer that are associated with a Git repository.

Glossary: Developing Front-End Apps with React

Welcome! This alphabetized glossary contains many of the terms in this course. This comprehensive glossary also includes additional industry-recognized terms not used in course videos. These terms are essential for you to recognize when working in the industry, participating in user groups, and in other professional certificate programs.

Estimated reading time: 12 minutes

Term	Definition
Action creators	Functions that create actions.
Actions	JSON objects that contain information about changes that need to be made to the state. For example, a GET action obtains information, a POST action sends information, and a DELETE action removes information.
Angular	Angular is a platform for building mobile and desktop web applications.
App.js	Contains App, which is the root React component that you will add into your HTML page.
Arrow functions	Allow you to declare functions the same way you declare variables.
Asynchronous or async code	Runs in parallel, and an operation can occur while another one is still being processed.
Babel	A special in-memory tool that compiles JSX and interprets it as JavaScript.
Callback	A normal method called once a condition is met.
Central Store	holds the entire application list in the form of the 'state tree'.

Term	Definition
Chai	An object that can be used as the assertion library.
Class	A template or blueprint for creating objects.
Class component	Use JavaScript ES6 classes to create class-based components in React.
Component	Core building blocks of React.
Component Lifecycle	A series of four phases make up the lifecycle of a component. These phases are: Initialization, Mounting, Updating, and Unmounting.
Component testing	When you render component trees in a simple test environment and assert their output.
Const	Allows you to declare constants whose values cannot be changed.
Controlled input	Use React to fully control the element by setting and updating the input value directly.
Create React App	Simplifies the process of creating React applications.
Cross-origin requests	Requests that come from multiple sources.
CSS	Cascading style sheets.
Custom Hook	A new composition of one or multiple Hooks.
Document Object Model (DOM)	It defines the logical structure of documents and the way a document is accessed and manipulated.
ECMAScript	A standard used for client-side scripting.
End-to-end testing	A multi-step test that combines multiple units and integrates all tests into one big test. It requires the running of a complete application in a realistic browser environment.
Enzyme	A library that can be added to your testing tools in order to render your React components.
ES	ES is short for ECMAScript, which is a standards organization that creates a wide range of global information and communications technology standards.
ES6	JavaScript adheres to ECMA's specification EcmaScript6.

Term	Definition
event	Enable the component to manage document object model (DOM) actions as a result of user interaction on the system.
Front-end frameworks	Are used to create a dynamic client that can connect to the server.
Functional component	Created by writing a JavaScript function.
High-order component	Are used to share logic with other components.
Hooks	Enable functional components to attach local state with it, and this state will be preserved by React when the component re-renders.
HTML	Hypertext markup language.
index.js	Where you will add App to the HTML.
Initialization phase	The first component lifecycle phase. During this phase, the component is constructed with the given Props and default state.
Jest	A JavaScript test runner and assertion library used to test React components. It can be combined with Enzyme or the React Testing Library.
JSX	JavaScript XML which resembles HTML.
let	Allows you to restrict the scope of variables within the block where they were declared.
Local scope	Is the limited scope.
Local state	One of the two types of React states. Local state lives in a single component and is not used in other components. An example of local state is hiding and showing information. (See **Shared state**.)
Middleware	Library that connects applications and services.
Mocha	Object that can be used as the test runner.
Mounting Phase	The second component lifecycle phase. During this phase, the JSX is rendered.
NPM	Package manager for the Node JavaScript platform.
onChange	Attribute that controls changes in the form.

Term	Definition
Payload property	An optional property that contains some data that is required to perform a task.
Promise	Represents the eventual completion of an asynchronous operation and its return value.
Props	Short for 'properties.' Read-only objects that store attribute values of tags. Used to pass data from parent to child components.
Pure component	Do not depend on or modify the state of variables. Outside their scope.
Push	When a user or software automation moves data from one location to another.
React elements	Used to render the component to DOM.
React Hook form	A form state management and validation library for React web-based applications and React Native mobile applications.
React package	Holds the React source for components and their states and properties.
React Testing Library	A set of helpers that let you test React components without depending on their implementation details. This library is a replacement for Enzyme.
ReactDOM package	This is the glue between React and the DOM.
Reducer	Pure functions that receive the current state and an Action object and return a new state with the actions performed.
Redux	A state management library that is often used with React to handle the state of your application.
Redux Saga	Middleware that uses an ES6 feature called Generators to enable async operations.
Redux Thunk	Middleware that allows to pass functions within action creators to create async Redux.
Ref function	Used to get the form values from DOM.
setState	Updates state of the form input elements.
Shared state	One of the two types of React states. Shared state is shared by multiple components and is complicated. An example of shared state is the list of all orders in an order application. (See **Local state**.)

Term	Definition
Sinon	An object that can be used to test JavaScript logic with objects such as spies, stubs, and mocks.
Snapshot testing	Testing that helps check and verify the component rendering result.
src folder	The main folder in which you will make changes.
State	A plain JavaScript object used by React to represent information about the component's current situation. States help track changes in a component.
Store	Contains the Redux application's current state, other functions, and objects.
subclass	The class that is inheriting one other class.
Subscription	Is triggered in the components whenever the state is updated in the store.
superclass	The class being inherited by the subclass.
Synchronous or sync code	Runs in sequence from top to bottom, and each operation must wait for the previous one to complete before executing.
Testing	A line-by-line review of how code executes. Testing replicates end user actions to ensure features work as intended.
this	Keyword that refers to the current object.
Type property	A string that identifies the action.
Uncontrolled input	Allows the browser to handle most of the form elements and collect data through React's change events.
Unmounting phase	The final component lifecycle phase. During this phase, the component is removed from the page.
Updating phase	The third component lifecycle phase. During this phase, the state of a component is updated, and the application is repainted.
useContext	Manages context changes and provides the component with access to a context.
useEffect	Manages side effects such as document changes, HTTP, and so on.
useReducer	Manages Redux state changes.

Term	Definition
useState	A hook that allows you to use state in your function. It adds state to a function component.

Cheat Sheet: Developing Front-End Apps with React

Estimated reading time: 19 minutes

Package/Method	Description	Code Example
Arrow function	Arrow functions allow you to write shorter function syntax.	1. 1 2. 2 3. 3 4. 4 1. hello = () => 2. { 3. return "Hello World!"; 4. } Copied!
class	Class is a template or blueprint for creating an object.	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9

Package/Method	Description	Code Example
		10.10 1. function car(name,year) 2. { 3. this.name = name 4. this.year = year 5. return this; 6. } 7. let car = car("Ford", 2014) 8. console.log(car) 9. console.log(car.name) 10. console.log(car.year) Copied!
Hooks	Functions called hooks enable "hooking into" features of the React state and lifecycle from function components.	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 1. importReact,{useState}from 'react'; 2. function CntApp() { 3. const[count,setCount]=useState(0); 4. return(Youclicked{count}many times 5. <buttononClick={()=>setCount(count+1)}>Clickme</button> 6.);} 7. export default CntApp; Copied!

Package/Method	Description	Code Example

Inheritance	A class created with a class inheritance inherits all the methods from another class.	1. 1
		2. 2
		3. 3
		4. 4
		5. 5
		6. 6
		7. 7
		8. 8
		9. 9
		10.10
		11.11
		12.12
		13.13
		14.14
		15.15
		1. class Square extends Rectangle
		2. {
		3. constructor(height,width)
		4. {
		5. if(height === width)
		6. {
		7. super(height,width)
		8. }
		9. else
		10.{
		11.super(width,width)
		12.}
		13.}

Package/Method	Description	Code Example
		14.} 15.let mySquare = new Square(5,5) Copied!
let and const	let allows you to restrict the scope of variables within the block where they are declared. const allows you to declare constants whose values cannot be changed.	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10.10 11.11 1. { 2. let a = 10 3. console.log(a) 4. a = 15 5. console.log(a) 6. } 7. console.log(a) 8. const num = 5 9. console.log(num) 10.num = 8 11.console.log(num) Copied!
Mounting	When a component instance is created	1. 1 2. 2

Package/Method	Description	Code Example

	and added to the DOM, these methods are invoked in the following order: constructor(), getDerivedStateFromProps(), render(), componentDidMount().	3. 3
		4. 4
		5. 5
		6. 6
		7. 7
		8. 8
		9. 9
		10.10
		11.11
		12.12
		13.13
		14.14
		15.15
		16.16
		1. class Header extends React.Component {
		2. constructor(props) {
		3. super(props);
		4. console.log("Inside the constructor")
		5. }
		6. componentDidMount =()=> {
		7. console.log("Inside component did mount")
		8. }
		9. render() {
		10.console.log("Inside render method")
		11.return (

Package/Method	Description	Code Example
		12. The component is rendered 13.) 14. } 15. } 16. export default App Copied!

onClick	When an event fires, event handlers decide what should happen next. This could involve pressing a button or altering a text entry.	1. 1
		2. 2
		3. 3
		4. 4
		5. 5
		6. 6
		7. 7
		8. 8
		9. 9
		10. 10
		11. 11
		1. function changeColor() {
		2. const shoot = () => {
		3. alert("Color Changed!");
		4. }
		5. return (
		6. <button onClick={change}>Change the Color! </button>
		7.);
		8. }
		9. const root =

Package/Method	Description	Code Example
		10. ReactDOM.createRoot(document.getElementById('root')); 11. root.render(<changeColor />); Copied!

Promises	The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.	1. let promiseArgument = (resolve, reject) =>
		2. setTimeout (() => {
		3. let currTime = new Date().getTime();
		4. if(currTime % 2 === 0){
		5. resolve("Success")
		6. }else{
		7. reject("Failed!!!")}
		8. },2000)
		9. let myPromise = new Promise(promiseArgument);
		Copied!

Props	Props is short for properties, and it is used to pass data	1. 1 2. 2 3. 3

Package/Method	Description	Code Example

	between React components.	4. 4
		5. 5
		6. 6
		7. 7
		1. class TestComponentextendsReact.component{
		2. render() {
		3. return
		4. Hi {this.props.name}}}
		5. //passing the props as examples to the test component
		6. TestComponentname='John'
		7. TestComponentname='Jill'
		Copied!

React class Component	React class component contains- Props: set from outside the class, State: internal to the class	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10.10 11.11 12.12 13.13 1. import React from "react";

Package/Method	Description	Code Example

		2. class App extends React.Component {
		3. constructor(props) {
		4. super(props);
		5. this.state={change: true };
		6. }
		7. render() {
		8. return(
		9. <button Click={()=>{this.setState({change:
		10. !this.state.change});}}>Click Here!</button>
		11. {this.state.change?(Hello!!):(Welcome to the React Course)}
		12.);}}
		13. export default App;
		Copied!

React components	Components are reusable segments of code that come under the class and functional component types.	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 1. import React from 'react'; 2. import {Text} from 'react-native'; 3. const Helloworld= ()=> 4. {

Package/Method	Description	Code Example
		5. return 6. (Hello, World!); 7. } 8. export default Helloworld; Copied!
React Forms	React makes use of forms to enable user interaction with the website.	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 14. 14 15. 15 16. 16 17. 17 18. 18 1. import React, {Component} from "react"; 2. export default functionApp() { 3. const [email, setEmail] = React.useState("");

Package/Method	Description	Code Example

		4. const [password, setPassword] = React.useState("");
		5. const handleSubmit = (event) => {
		6. console.log(`Email: ${email}Password: ${password}`);
		7. event.preventDefault();
		8. }
		9. return (< formonSubmit = {
		10.handleSubmit
		11.} >
		12.< h1 > Registration < /h1>
		13.<label> Email:< input name="email" type="email" value={email}
		14.onChange={e => setEmail(e.target.value)} required/ > < /label>
		15.<label>Password:<input name="password" type="password" value={password}
		16.onChange={e => setPassword(e.target.value)}required /></label>
		17.<button>Submit</button>
		18.</form>);}
		Copied!

React state	The state object is where you keep the	1. 1 2. 2 3. 3

Package/Method	Description	Code Example

	component's property values.	4. 4
		5. 5
		6. 6
		7. 7
		8. 8
		9. 9
		10.10
		1. class TestComponentextendsReact.component{
		2. constructor() {
		3. this.state= {
		4. id: 1,
		5. name: “John”
		6. age: 28
		7. };}
		8. render() {
		9. return ({this.state.name}{this.state.age}
		10.)}}
		Copied!

Redux	Redux is a state management library that is often used with React to handle the state of your application. An application state is like a global object that holds	1. 1 1. $ npm install redux react-redux --save Copied!

Package/Method	Description	Code Example
	information that you use for various purposes later in the app.	
Unmounting	When a component is removed or unmounted from the DOM, the componentWillUnmount() method enables us to run React code.	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10.10 11.11 12.12 13.13 14.14 15.15 16.16 17.17 18.18 19.19 20.20 21.21 22.22 23.23 1. `import React from 'react';`

Package/Method	Description	Code Example

		2. class ComponentOne extends React.Component {
		3. componentWillUnmount() {
		4. console.log('The component is going to be unmounted');
		5. }
		6. render() {
		7. return Inner component;
		8. }
		9. }
		10. class App extends React.Component {
		11. state = {
		innerComponent:<AppInner/>};
		12. componentDidMount() {
		13. setTimerout(()=>{
		14. this.setState({
		innerComponent:unmounted});
		15. },5000)
		16. }
		17. render() {
		18. console.log("Inside render")
		19. return (
		20. {this.state.innerComponent});
		21. }
		22. }
		23. export default App;
		Copied!

Package/Method	Description	Code Example
Updating	When a component is updated, five methods are called in the same order: getDerivedStateFromProps(), shouldComponentUpdate(), render(), getSnapshotBeforeUpdate(), componentDidUpdate()	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 14. 14 15. 15 16. 16 17. 17 18. 18 19. 19 20. 20 21. 21 22. 22 23. 23 24. 24 1. class App extends React.Component{ 2. state = {counter: "0"}; 3. incrementCounter = () =>

Package/Method	Description	Code Example

		4. this.setState({counter:parseInt(this.state.counter)+1})
		5. shouldComponentUpdate(){
		6. console.log("Inside shouldComponent update")
		7. return true;
		8. }
		9. getSnapshotBeforeUpdate(prevProps,prevState){
		10. console.log("Inside getSnapshotBeforeUpdate")
		11. console.log("Prev counter is" +prevState.counter);
		12. console.log("New counter is" +this.state.counter);
		13. return prevState
		14. }
		15. componentDidUpdate(){
		16. console.log("Inside componentDidUpdate")
		17. }
		18. render(){
		19. console.log("Inside render")
		20. return(<button onClick={this.incrementCounter}>Click
		21. Me!</button>{this.state.counter}
		22.)

Package/Method	Description	Code Example
		23.}} 24.export default App;

Glossary: Developing Back-End Apps with Node.js and Express

Welcome! This alphabetized glossary contains many of the terms in this course. This comprehensive glossary also includes additional industry-recognized terms not used in course videos. These terms are essential for you to recognize when working in the industry, participating in user groups, and in other professional certificate programs.

Estimated reading time: 12 minutes

Term	Definition
Access Token	A small piece of code that contains information about the user, their permissions, groups, and expirations that get passed from a server to the client.
Anonymous Function	A function that is not named. An anonymous function is often passed into another function as a parameter.
API Endpoint	The touchpoint where the API connects to the application it is communicating with.
Application Server	Transforms data into dynamic content and runs the business logic, which is the data storage and transfer rules.
Application-Level Middleware	Acts as a gatekeeper and is bound to the application. No request to the application server can go past it.
Async	Short for "asynchronous". A process that runs independently of other processes.
Authentication	The process of confirming a user's identity using credentials by validating who they claim to be. Authentication assures an application's

Term	Definition
	security by guaranteeing that only those with valid credentials can access the system.
Authorization	In token-based authentication, it is the process that gets executed when a web application wants to access a protected resource. A user authenticates against an authorization server.
Axios Package	The axios package wraps promises around HTTP requests. It returns a promise object.
Built-In Middleware	Can be bound to either the entire application or to specific routers. Useful for activities such as rendering HTML pages from the server, parsing JSON input from the front end, and parsing cookies.
Callback Function	A function passed into another function as a parameter, which is then invoked inside the outer function to complete an action. Instead of blocking on asynchronous I/O operations, callback functions are used to handle results when the operations complete.
Callback Hell	Many nested callback functions.
Controller	The layer in an MVC application regulates the flow of the data. It is responsible for processing the data supplied to it by the user and sends that data to the model for manipulation or storage.
Database Server	A server dedicated to providing database services.
Dependencies	Code, usually in the form of libraries and packages, that are called from other modules and reused in a program.
Error-Handling Middleware	Can be bound to either the entire application or to specific routers. Error-handling middleware always takes four arguments: error, request, response, and the next function that it needs to be chained to. Even if you don't use the next parameter, you still define it in the method signature.
Event-Driven	Where the flow of a program is determined by particular events such as user input.
Express.js	A highly configurable web framework for building applications on Node.js.
Framework	Generates code that cannot be altered to perform common tasks. Examples include Django, Ruby on Rails, and Express.js.
HTTP Headers	Additional information about and contained in an HTTP response or request.

Term	Definition
HTTP Request	A method called by a client and sent to a server requesting access to a resource on the server.
HTTP Response	A method called by a server and sent to a client in response to an HTTP request.
HTTP Server	A type of software-based server that understands URLs and hypertext transfer protocol.
ID Token	An artifact that proves that a user has been authenticated and contains information about authorized API routes.
Inversion of Control	When the flow of control, such as the execution of instructions, is controlled by a third party.
JSON Payload	Data that is transferred in JSON format between the client and server in the header of an HTTP method.
JWT	A JSON Web token. An internet standard for creating encrypted payload data in JSON format.
Load	Refers to the number of concurrent users, the number of transactions, and the amount of data transferred back and forth between the clients and servers.
Middleware	Includes functions that have access to the request and response objects and the next() function.
Model	The layer in an MVC application responsible for managing the data of the application. It interacts with the database and handles the data logic.
Module	Files containing related, encapsulated JavaScript code that serve a specific purpose.
Multi-Threaded	Where multiple tasks are executed simultaneously.
MVC	Short for "Model-View-Controller". It is an architectural pattern that divides an application into three components: model, view, and controller.
Node Framework	A framework that works in conjunction with Node.js. A framework is a skeleton on which an application is built for a specific environment. The framework is the fundamental structure that supports the application.
Node.js	A JavaScript runtime environment that runs on Google Chrome's V8 engine.

Term	Definition
Non-Blocking	Failure of a given thread does not cause failure in another, and the execution of a task is not blocked until execution of another task is completed.
Npm	Stands for node package manager. It is the default package manager for the Node.js runtime environment.
Package	A directory with one or more modules bundled together.
Package.json	Contains metadata information about the project, including dependencies and scripts.
Passwordless Authentication	A type of authentication that uses public/private key pairs to encrypt and decrypt data passed between client and server without the need for a password.
Payload	The data transmitted between client and server.
Private Key	In cryptography, it is a key that is known only to a specific client used to decrypt messages. Used in conjunction with a public key.
Promise	An object in JavaScript that acts as a placeholder for an asynchronous task that is yet to be completed. It is the object that is returned from an asynchronous method. When you define a promise object in your script, instead of returning a value immediately, it returns a promise. The promise has three states, which are pending, fulfilled, and rejected.
Public Key	In cryptography, it is a key that can be used by anyone to encrypt messages for a specific client. Used in conjunction with a private key.
Pyramid of Doom	Another term for "callback hell".
REST	"Representational state transfer" is a set of guidelines for creating stateless client/server interfaces using HTTP methods.
REST API	An API used for communicating between clients and servers that conforms to REST architecture principles.
Route	The code that associates an HTTP request method and a URL.
Router-Level Middleware	Bound to a router and not bound to an application. You can use specific middleware for a specific route instead of having all requests go through the same middleware. Then you bind the application routes to each router.
Runtime Environment	Behaves similarly to a mini operating system that provides the resources necessary for an application to run. It is the infrastructure that supports

Term	Definition
	the execution of a codebase. It is the hardware and software environment in which an application gets executed. Node.js is an example of a backend runtime environment.
Scalability	The application's ability to dynamically handle the load as is or shrinks without it affecting the application's performance.
Server.js	A file that contains the code that handles server creation.
Session-based Authentication	A type of authentication where a user provides login credentials that are verified against stored credentials in a database on a server. A session ID is provided to the client and stored in the browser as a cookie.
Single-Threaded	Where only one command is processed at a given point of time.
Statelessness	Implies that each HTTP request happens in isolation in relation to other requests. The state of the client is not stored on the server; the state is passed to the server by the client for each request.
Template Rendering	The ability of the server to fill in dynamic content in an HTML template.
Token	Contains three parts, the header, the payload, and the signature. The header contains information about the type of token and the algorithm used to create it. The payload contains user attributes, called claims, such as permissions, groups, and expirations. The signature verifies the token's integrity, meaning that the token hasn't changed during transit.
Token-based Authentication	A type of authentication that uses access tokens, often JWTs, which get passed between server and client with the data that is passed between the two.
TypeScript	A language that builds on top of JavaScript and provides type safety which assists with the development of large-scale applications by reducing the complexity of component development in JavaScript.
View	The layer in an MVC application responsible for rendering the presentation of the data that is passed to it by the model.
Web Server	Ensures client requests are responded to, often using HTTP.
Web Service	A type of web API that communicates using HTTP requests. It is the web service in the programming interface that sends and receives requests using HTTP among web servers and the client.
xml2js	Node.js package to parse a string of XML elements into a JavaScript object.

Cheat Sheet: Developing Back-End Apps with Node.js and Express

Estimated reading time: 5 minutes

Package/Method	Description	Code Example
Async-await	We can await promises as long as they are being called inside asynchronous functions.	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 1. `const axios = require('axios').default;` 2. `let url = "some remote url"` 3. `async function asyncCall() {` 4. `console.log('calling');` 5. `const result = await axios.get(url);` 6. `console.log(result.data);` 7. `}` 8. `asyncCall();` Copied!
callback	Callbacks are methods that are passed as parameters. They are	1. 1 2. 2 3. 3 4. 4 5. 5

Package/Method	Description	Code Example

	invoked within the method to which they are passed as a parameter, conditionally or unconditionally.
	We use callbacks with a promise to process the response or errors.
		6. 6
		7. 7
		8. 8
		1. //function(res) and function(err) are the anonymous callback functions
		2. axios.get(url)
		3. .then(function(res) {
		4. console.log(res);
		5. })
		6. .catch(function(err) {
		7. console.log(err)
		8. })
		Copied!

Default/Date	new Date() method returns the current date as an object. You can invoke methods on the date object to format it or change the timezone.	1. 1 2. 2 3. 3 4. 4 1. module.exports.getDate = function getDate() { 2. var aestTime = new Date().toLocaleString("en-US", {timeZone: "Australia/Brisbane"}); 3. return aestTime; 4. } Copied!

Package/Method	Description	Code Example
express.get();	This method is meant to serve the retrieve requests to the server. The get() method is to be implemented with two parameters; the first parameter defines the end-point and the second parameter is a function taking the request-handler and response-handler.	1. 1 2. 2 3. 3 1. // handles GET queries to endpoint /user/about/id. 2. 3. app.get(“user/about/:id”, (req,res)=>{ res.send(“Response about user “ +req.params.id) }) Copied!
express.listen()	The listen method is invoked on the express object with the port number on which the	1. 1 2. 2 3. 3 1. app.listen(3333, () => { 2. console.log("Listening at http://localhost:3333) 3. })

Package/Method	Description	Code Example
	server listens. The function is executed when the server starts listening.	Copied!
express.post();	This method is meant to serve the create requests to the server. The post() method is to be implemented with two parameters: the first parameter defines the endpoint and the second parameter is a function taking the request-handler and response-handler.	1. 1 2. 2 3. 3 4. 4 5. 5 1. // handles POST queries to the same endpoint. 2. app.post("user/about/:id", (req,res)=>{ 3. res.send("Response about user " 4. +req.params.id) 5. }) Copied!

Package/Method	Description	Code Example
express.Router()	Router-level middleware is not bound to the application. Instead, it is bound to an instance of express.Router(). You can use specific middleware for a specific route instead of having all requests going through the same middleware. Here, the route is /user, and you want the request to go through the user router. Define the router, define the middleware function that the router will	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10.10 11.11 12.12 13.13 14.14 15.15 16.16 17.17 1. const express = require("express"); 2. const app = new express(); 3. var userRouter = expess.Router() 4. var itemRouter = express.Router() 5. userRouter.use(function (req, res, next){ 6. console.log("User quert time:", Date()); 7. next(); 8. })

Package/Method	Description	Code Example

	use and what happens next, and then you bind the application route to the router.	9. userRouter.get("/:id", function (req, res,
		10. next) {
		11. res.send("User "+req.params.id+ "
		last
		12. successful login "+Date())
		13. })
		14. app.listen(3333, () => {
		15. console.log("Listening at
		16. http://localhost:3333)
		17. })
		Copied!

express.static()	This is an example of static middleware that is used to render static HTML pages and images from the server side. The static files can be rendered from the cad220_staticfiles directory at the application	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 1. const express = require("express"); 2. const app = new express(); 3. app.use(express.static("cad220_static files 4. ")) 5. app.listen(3333, () => { 6. console.log("Listening at 7. http://localhost:3333)

Package/Method	Description	Code Example
	level. Notice that the URL has only the server address and the port number followed by the filename.	8. }) Copied!

express.use()	This method takes middleware as a parameter. Middleware acts as a gatekeeper in the same order that it is used before the request reaches the get() and post() handlers. The order in which the middleware is chained depends on the order in which the .use() method is used to	const express = require("express");
		1. const app = new express();
		2. function myLogger(req, res, next){
		3. req.timeReceived = Date();
		4. next();
		5. }
		6. app.get("/", (req, res)=>{
		7. res.send("Request received at
		8. "+req.timeReceived+" is a success!")
		9. })
		Copied!

Package/Method	Description	Code Example
	bind them. The middleware myLogger() function takes three parameters, which are request, response, and next. You can define a method that takes these three parameters and then bind it with express.use() or router.use(). Here, you are creating middleware named myLogger and making the application use it. The output rendered	

Package/Method	Description	Code Example
	includes the time the request is received.	
express-react-views.createEngine() and express.engine()	This example uses express-react-views, which renders React components from the server. You set the view engine property, which is responsible for creating HTML from your views. Views are JSX code. The views are in a directory named myviews. The view engine will look for a JSX file named index in the myviews	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 14. 14 15. 15 16. 16 17. 17 1. `const express = require("express");` 2. `const app = new express();` 3. `const expressReactViews =` 4. `require("express-react-views);` 5. `const jsxEngine =` 6. `expressReactViews.createEngine();` 7. `app.set("view engine", "jsx");`

Package/Method	Description	Code Example

	directory and pass the property name to it. The output rendered will have the name of the user.	8. app.set("views", "myviews");
		9. app.engine("jsx", jsxEngine);
		10. app.get("/:name", (req, res)=>{
		11. res.render("index", { name:
		12. req.params.name });
		13. });
		14. app.listen(3333, () => {
		15. console.log("Listening at
		16. http://localhost:3333)
		17. })
		Copied!

http/createServer	http package is used for establishing remote connections to a server or to create a server which listens to client. CreateServer - Takes a requestListener, a function which takes request and response parameters, where request is the handle	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 1. const http = require('http'); 2. const requestListener = function(req, res) { 3. res.writeHead(200); 4. res.end('Hello, World!'); 5. } 6. const port = 8080; 7. const server = http.createServer(requestListener);

Package/Method	Description	Code Example
	to the request from the client and response is the handle to be sent to the client.	8. console.log('server listening on port: '+ port); Copied!
Import()	The import statement is used to import modules that some other module has exported. A file that includes reusable code is known as a module.	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 1. // addTwoNos.mjs 2. function addTwo(num) { 3. return num + 4; 4. } 5. 6. export { addTwo }; 7. // app.js 8. import { addTwoNos } from './addTwoNos.mjs'; 9.

Package/Method	Description	Code Example
		10. // Prints: 8 11. console.log(addTwo(4)); Copied!
new express()	Creates an express object which acts as a server application.	1. 1 2. 2 1. const express = require("express"); 2. const app = new express(); Copied!
Promise	An object that is returned by some methods, representing eventual completion or failure. The code continues to run without getting blocked until the promise is fulfilled or an exception is thrown.	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 1. axios.get(url) 2. .then(3. //do something 4.) 5. .catch(6. //do something 7.) Copied!
Promise use case	Promises are used when the processing time of the function we	1. 1 2. 2 3. 3 4. 4 5. 5

Package/Method	Description	Code Example

	invoke takes time like remote URL access, I/O operations file reading, etc.	6. 6
		7. 7
		8. 8
		9. 9
		10. 10
		11. 11
		12. 12
		13. 13
		14. 14
		15. 15
		16. 16
		1. var prompt = require('prompt-sync')();
		2. var fs = require('fs');
		3. const methCall = new Promise((resolve,reject)=>{
		4. var filename = prompt('What is the name of the file ?');
		5. try {
		6. const data = fs.readFileSync(filename, {encoding:'utf8', flag:'r'});
		7. resolve(data);
		8. } catch(err) {
		9. reject(err)
		10. }
		11. });
		12. console.log(methCall);

Package/Method	Description	Code Example
		13. methCall.then(14. (data) => console.log(data), 15. (err) => console.log("Error reading file") 16.); Copied!
Require()	The built-in NodeJS method require() is used to incorporate external modules that are included in different files. The require() statement essentially reads and executes a JavaScript file before returning the export object.	1. 1 2. 2 1. module.exports = 'Hello Programmers'; 2. message.js;var msg = require('./messages.js');console.log(message);