React Nanodegree Syllabus

Become a Professional React Developer

Before You Start

Congratulations on considering the React Nanodegree program! Before you get started, make sure to set aside adequate time on your calendar for focused work, and double-check that you meet the requirements: you should have prior programming experience that includes HTML, CSS, and JavaScript programming.

The React Nanodegree program is comprised of 3 courses and 3 projects. Each project you build will be an opportunity to demonstrate what you've learned in your lessons. Your completed projects become part of a career portfolio that will demonstrate your mastery of React to potential employers.

Prerequisites:

Students should have prior development experience building and deploying front-end applications with HTML, CSS, JavaScript, Git, GitHub, NPM, and experience using the command line interface (bash, terminal).

Students will need to be able to communicate fluently and professionally in written and spoken English.

Educational Objectives:

Length of Program*: 160 Hours
Textbooks required: None
Instructional Tools Available: Video lectures, Mentors, Forums

*The length is an estimation of total hours the average student may take to complete all required coursework, including lecture and project time. If you spend about 10 hours per week working through the program, you should finish in 16 weeks, so approximately 4 months. Actual hours may vary.

Course: React Fundamentals

Mastering React begins with learning your fundamentals, and this can pose a bit of a challenge, because while the modularity of the React ecosystem makes it really powerful for building applications, there is a great deal to learn. So we'll break everything down, and enable you to learn the foundational parts of the React ecosystem that are necessary to build production-ready apps.

As this is a project-based course, you're going to start building right away. This gives you an opportunity to get your hands dirty with React, and start mastering the skills you'll need. Plus, every project you build is reviewed by an expert Project Reviewer, and their detailed feedback will be instrumental in helping you to advance.

Lesson Title	Learning Outcomes
Why React	➔ Identify why React was built ➔ Use *composition* to build complex functions from simple ones ➔ Leverage *declarative code* to express logic without control flow ➔ Recognize that React is just JavaScript
Rendering UI with React	➔ Use *create-react-app* to create a new React application ➔ Create reusable, focused *Class components* with composition ➔ Leverage *JSX* to describe UI
State Management	➔ Manage state in applications ➔ Use *props* to pass data into a component ➔ Create *functional components* focused on UI rather than behavior ➔ Add *state* to components to represent mutable internal data ➔ Use the *this* keyword to access component data and properties ➔ Update state with *setState()* ➔ Use *PropTypes* to typecheck and debug components ➔ Use *controlled components* to manage input form elements
Render UI with External Data	➔ Conceptualize the *lifecycle* of a component ➔ Use React's *componentDidMount* lifecycle hook for HTTP requests
Manage App Location with React Router	➔ Use React Router to add different routes to applications ➔ Use state to dynamically render a different "page" ➔ Use React Router's *Route* component ➔ Use React Router's *Link* component

Project: MyReads: A Book Lending App

In this project, you will create a React application from scratch and utilize React components to manage the user interface. You'll create a virtual bookcase to store your books and track what you're reading. Using the provided Books API, you'll search for books and add them to a bookshelf as a React component. Finally, you'll use React's setState to build the functionality to move books from one shelf to another.

Course: React & Redux

Redux excels at state management, and in this course, you'll learn how Redux and React work together to make your application's state bulletproof.

As with the previous course, this is hand-on curriculum, and building projects is what it's all about. Here, you'll leverage React with Redux to build "Would You Rather", a popular party game.

Lesson Title	Learning Outcomes
Managing State	➔ Recognize how state predictability improves applications ➔ Create a store to manage an applications state ➔ Leverage store API: getState(), dispatch(), and subscribe() ➔ Create Actions and Action Creators that describe state changes ➔ Create Reducers that return state ➔ Use Reducer Composition to handle independent parts of state
UI + Redux	➔ Combine Redux with a user interface ➔ Build intuition for when to use Redux
Redux Middleware	➔ Identify the benefits of implementing middleware in applications ➔ Identify the role of middleware within the Redux cycle ➔ Apply middleware to a Redux application ➔ Build your own Redux middleware
Redux with React	➔ Combine Redux with the popular React library ➔ Identify when to use component state vs. Redux state
Asynchronous Redux	➔ Learn the pitfall of asynchronous requests in Redux ➔ Leverage Thunk middleware to support asynchronous requests ➔ Fetch data from a remote API
react-redux	➔ Install the react-redux bindings ➔ Leverage react-redux bindings to extend app functionality ➔ Use the Provider to pass a store to component trees ➔ Use connect() to access store context set by the Provider
Real World Redux	➔ Build a complex, real-world application with Tyler ➔ Add Redux to an application scaffolded with Create React App ➔ Normalize state shape to keep application logic simple with scale

Project: Would You Rather

Leverage the strengths of Redux to build a "Would You Rather" application in which users are given questions and must choose one of them. You'll build this dynamic application from scratch while combining the state management features of Redux and the component model of React. When complete, you'll be able to create your own sets of questions, choose between them, and keep track of question popularity.

Course: React Native

In this course, you'll learn how to to develop React applications that run on both iOS and Android devices. We'll explore everything from setting up a proper development environment, building and styling a cross-platform mobile application. You'll incorporate native APIs such as geolocation and local notifications, and even learn how to get your app ready for the Google Play Store and the App Store!

Lesson Title	Learning Outcomes
Up and Running with React Native	→ Identify the ideology behind React Native → Set up an ideal development environment → Inspect and debug applications
React vs React Native	→ Identify fundamental differences between web and native apps → Identify differences between Android and iOS platforms → Leverage common React Native components → Create forms in React Native applications → Utilize AsyncStorage to persist global application data → Incorporate Redux to manage shared application state
Styling & Layout	→ Style applications with CSS in JS → Identify differences and use-cases between styling with inline styles, object variables, and the *Stylesheet* API → Recognize the core philosophies and techniques of CSS *flexbox* → Identify key differences between flexbox on the web and React Native's implementation of flexbox → Identify best practices in how professionals handle styling
Navigation	→ Manage navigation through a React Native application → Utilize *StackNavigator* to render screens from a stack → Implement *TabNavigator* to switch between screens by using tabs → Utilize *DrawerNavigator* to switch between screens from a drawer menu
Native Features	→ Leverage native APIs to extend app functionality → Incorporate *Geolocation*, *Animations*, *Notifications*, and *ImagePicker* to take advantage of device features and data → Prepare applications for the Google Play Store and the App Store

Project: Mobile Flashcards

In this project, you'll use React Native to build a mobile flashcard app. Users will not only be able to create custom cards and decks, but they'll also be able to set up notifications to remind them to study. You'll leverage React Native components, AsyncStorage, proper styling, as well as device APIs to create a fully dynamic experience.