**MERN Training**

Onboarding 2 Days

UI basic 2-3 Days

# JavaScript + Typescript (basic) 3 Days

# Project 1 2 Days

[Basics Revision](https://docs.google.com/document/d/1mHqeRmrnoLqUOaRoGNdZQP0PkRUSM8krC7pdvax5VZw/edit?usp=sharing) 5 Days

# React.js

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 2-3 Days

# NEXT.JS 3-4 Days

# **Project 2 1-2 Days**

# **Node.js + Express.js +** [**Git Theory**](https://docs.google.com/document/d/1eU5XXOH_gd_mS8_wT-yrArgFwVlaEla6/edit?usp=sharing&ouid=100106978472609532910&rtpof=true&sd=true) 3 Days

# **(Mongoose + MongoDB) / (Sequelize / Postgres) +** [**Git Theory**](https://docs.google.com/document/d/1eU5XXOH_gd_mS8_wT-yrArgFwVlaEla6/edit?usp=sharing&ouid=100106978472609532910&rtpof=true&sd=true) 2 Days

# **Project 3 5-8 Days**

# **Redux + Redux Toolkit (optional) 1-2 Days**

**Instructions:**

* The training is divided into several sections. Each section has bullet points. Your goal is to clear concepts about each bullet.
* For [Basics Revision](https://docs.google.com/document/d/1mHqeRmrnoLqUOaRoGNdZQP0PkRUSM8krC7pdvax5VZw/edit?usp=sharing) You will be given a maximum of 5 days to complete basic training and have it evaluated. In-case, you are able to complete the document before the 5 day deadline; you need to move to the next part of training right away as the rest of the 2 days will be discarded.

**For example:** If you have completed the document in 3 days then you will not be given the leverage of 2 days. If you have any confusion regarding this point, it is your responsibility to reach out to L&D Team for clarification.

* [Git Theory](https://docs.google.com/document/d/1eU5XXOH_gd_mS8_wT-yrArgFwVlaEla6/edit?usp=sharing&ouid=100106978472609532910&rtpof=true&sd=true) is to be gone through as mentioned above. No extra time will be given for it specifically, you have to go through it alongside and it will be evaluated.
* You will work on a total of 3 projects in this training. Each project covers important concepts of your training.
* Make sure you visit mdn links for javascript concepts, which is the best platform to learn javascript. It helps you a lot in learning **React.js** and **Node.js** later on because they both use javascript.
* **Remember!** All the sections have video and documentation links. Make sure you read the documentation as well because video lectures are mostly scenario based so you may not be able to generalize the concepts.
* It is advised to prefer the documentation links provided with the topics over videos. Video lectures are there for reference but don’t spend too much time on them.
* At the end of this training you will be confident in MERN and have a huge knowledge about MERN Stack Development. Have Fun!

[**UI Basics**](https://docs.google.com/document/d/1lYXOGDda24X5TRdgGHP2q32wQiGDkopWDn2mNWkOs_0/edit#heading=h.tnjcu3m2k9pz)**(only follow UI section from this link)**

**JavaScript:**

Reference Book for Javascript: [Eloquent Javascript](https://eloquentjavascript.net/Eloquent_JavaScript.pdf)

* [Comparison Operators](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators)
* [Functions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions)
* [Functions with default parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions#default_parameters)
* [Callbacks](https://developer.mozilla.org/en-US/docs/Glossary/Callback_function)
* [Higher Order Functions](https://eloquentjavascript.net/05_higher_order.html)
* [Arrays](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)
* [Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)
* [null vs undefined](https://www.geeksforgeeks.org/undefined-vs-null-in-javascript/)
* [truthy](https://developer.mozilla.org/en-US/docs/Glossary/Truthy) / [falsy](https://developer.mozilla.org/en-US/docs/Glossary/Falsy) concept

[Error Handling](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch)

* [Hoisting in javascript](https://developer.mozilla.org/en-US/docs/Glossary/Hoisting)
* [Variable](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var) and [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function) Hoisting
* [Scope Chain and Lexical Scoping](https://www.freecodecamp.org/news/javascript-lexical-scope-tutorial/)
* [IIFE](https://developer.mozilla.org/en-US/docs/Glossary/IIFE)
* [“use strict” mode](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode)
* [Closures in javascript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures)
* [Bind() method](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind)
* [**this** keyword](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this)
* [Iterators and Generators](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Iterators_and_Generators)
* [Classes, Objects,](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes) and [Inheritance](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Inheritance)
* [Callback Hell](https://www.geeksforgeeks.org/what-is-callback-hell-in-node-js/)
* [Promises](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)
* [Async / Await](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Async_await)
* [Event Loop](https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop)

**ES6 / ES7 and latest versions:**

ECMAScript is nothing but a collection of some new features added to JavaScript that obviously are designed to make our life easier. Have a look at the following points and see how it solves the problem in a friendly way.

* [Object destructuring](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)
* [Object literal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#object_literals)
* [Spread operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax)
* [Rest operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters)
* [Arrow Functions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions)
* [Nullish coalescing operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Nullish_coalescing_operator)
* [Optional chaining operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining)
* [var vs let vs const](https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/)
* [Trailing commas](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Trailing_commas)

**Arrays and Object Functions:**

You might wonder why I need to cover all these functions separately but as you dive deeper in **MERN** you see that most of the time you are working with these functions, a better understanding of these helps you a lot in a longer run. So, you must clear concepts about these higher order functions at this point.

* Reference to learn and practice array functions: [w3schools array functions](https://www.w3schools.com/js/js_array_methods.asp). Practice all the array and object functions from the attached link.

**Project 1**

Use learned concepts of JavaScript to create a calculator application with HTML, CSS and JavaScript. Follow the latest ES6/ES7 syntax learned above. Kindly don’t waste your time on making the calculator visually appealing. Focus only on implementing the requirements.

**Modules:**

**1. Expressions:**

* The calculator should be able to parse full expressions using mathematical rules.
* The calculator should support following operations with the mentioned symbols:
* Addition: **+**
* Subtraction: **-**
* Multiplication: **\***
* Division: **/**
* Exponentiation: **^**
* Precedence/Parentheses: **(** expression... **)**
* Square root: **sqrt(** expression… **)**
* Sine: **sin(** expression… **)**
* Cosine: **cos(** expression… **)**
* Tangent: **tan(** expression… **)**
* Cross check your results using a scientific calculator with its corresponding symbols.
* White Spaces should be trimmed.

**2. Text/Expression Input**

* The calculator should have an editable text input.
* The user should be able to freely type in the desired expression through the keyboard.
* The expression should also update accordingly when the corresponding calculator button is clicked by the user.

**3. Output Display**

* The calculator should have a display where the entered expression's result is shown.
* Text input should not be overridden to display the result.
* The output should be fixed to 4 decimal points.

**4. Buttons**

* The calculator should have buttons for entering each element of the expression in the text input.
* Similarly, respective buttons should be added to fulfill the project requirements.

**5. Exception Handling**

* The application should handle all exceptions like dividing by zero or incomplete/wrong expression etc.
* A helpful error message should be shown on the output display.

**6. Variables and Constants**

* Calculator should support adding the following constants in expressions:
* Pi: **pi** (3.1415)
* Euler’s Number: **e** (2.7182)
* Calculator should support adding variables in the expressions.
* It should do so by creating a view with input(s) that allow the user to assign a name and a primitive value to a variable.
* The user should not be able to create a variable with the names of previously mentioned constants.
* When the user types these variables and constants in expressions, they should be evaluated to their stored values.

**7. History**

* The app should maintain the history of all evaluated expressions with their results (a history item).
* The view should allow the user to fill text input when a history item is clicked.
* The user should be able to delete an item from history.

[**Basics Revision**](https://docs.google.com/document/d/1mHqeRmrnoLqUOaRoGNdZQP0PkRUSM8krC7pdvax5VZw/edit#)

**React General Training Tasks**

Reference Notes on React.js: [Notes](https://books.goalkicker.com/ReactJSBook/ReactJSNotesForProfessionals.pdf)Reference Video Tutorial on React.js: [Video Lectures](https://www.youtube.com/watch?v=Ke90Tje7VS0)

React is a frontend javascript library for building fast and interactive web applications. It uses an approach to develop smaller components and combine them together to make fully functional applications.

* [Introducing JSX](https://react.dev/learn/writing-markup-with-jsx)
* [State vs Props](https://react.dev/learn/thinking-in-react#props-vs-state)
* [Class components vs Functional components](https://www.freecodecamp.org/news/functional-components-vs-class-components-in-react/)
* [Life Cycle Methods](https://blog.logrocket.com/react-lifecycle-methods-tutorial-examples/)
* componentWillMount()
* componentDidMount()
* shouldComponentUpdate()
* componentWillUpdate()
* componentDidUpdate()
* componentWillUnmount()
* [Passing props to Components](https://react.dev/learn/passing-props-to-a-component)
* [Lifting State Up Concept](https://react.dev/learn/sharing-state-between-components#lifting-state-up-by-example)
* [Conditional Rendering](https://react.dev/learn/conditional-rendering)
* [Controlled vs Uncontrolled Components](https://react.dev/learn/sharing-state-between-components#controlled-and-uncontrolled-components)
* [Formik](https://formik.org/docs/overview)

**React Router DOM**

So far, we have learned how to develop a single page application (SPA) but what if

we have multiple pages in our application like *Home* *page* and *About Page* etc. We

would want to navigate between these pages. So, we have a great mechanism to

handle all about navigation and routing called **React Router DOM.** Lets see what this

package provides:

Reference Video: [React Router DOM](https://www.youtube.com/watch?v=o__czqXJtqk)

* [Browser Router](https://reactrouter.com/en/6.20.0/start/tutorial)
* [Router](https://reactrouter.com/en/6.20.0/routers/picking-a-router)
* [Route](https://reactrouter.com/en/6.20.0/route/route)
* Navigate
* [Link](https://reactrouter.com/en/6.20.0/components/link)
* [NavLink](https://reactrouter.com/en/6.20.0/components/nav-link)

Reference Video to see what's new in React Router Version 6:

<https://www.youtube.com/watch?v=k2Zk5cbiZhg>

**React Hooks**

You have learned in previous sections about **Class** and **Functional** components**.**

You might find it difficult to switching between class and functional

components as you declared states in class components and passing it to functional

Components. This is really a headacf Don’t worry React Hooks come to the rescue!

React Hooks let you always use the functional components instead of switching

between the two. Lets see how the hooks helps you in functional components:

Reference Video: [React Hooks](https://www.youtube.com/watch?v=O6P86uwfdR0&list=PLZlA0Gpn_vH8EtggFGERCwMY5u5hOjf-h)

Reference Link: [React Hooks](https://react.dev/reference/react/hooks)

* [useState()](https://react.dev/learn/typescript#typing-usestate)
* [useEffect()](https://react.dev/reference/react/useEffect)
* [useRef()](https://react.dev/reference/react/useRef)
* [useMemo()](https://react.dev/reference/react/useMemo)
* [useCallback()](https://react.dev/reference/react/useCallback)
* [Custom Hooks](https://reactjs.org/docs/hooks-custom.html)

**NEXTJS**

It's a framework of ReactJS, it make the whole process easier, it helps your website loads really

fast and it's smart about splitting up your code so things work smoothly.

**1.** [**Pages Router**](https://nextjs.org/docs/pages)**:**

In Next.js, your application is structured around pages. Each file in the pages directory

becomes a route in your application. For example, pages/index.js corresponds to the

root route ("/"), and pages/about.js corresponds to "/about."

**2.** [**Routing**](https://nextjs.org/docs/pages/building-your-application/routing)**:**

Next.js provides automatic route handling based on the file structure in the pages

directory. No need to configure routes; it's intuitive and follows a

convention-over-configuration approach.

**3.** [**Server-Side Rendering (SSR)**](https://nextjs.org/docs/pages/building-your-application/rendering/server-side-rendering)**:**

Next.js allows for server-side rendering, which means that pages can be rendered on the

server before being sent to the client. This can improve performance and is particularly

useful for SEO.

**4.** [**Static Site Generation (SSG)**](https://nextjs.org/docs/pages/building-your-application/rendering/static-site-generation)**:**

Next.js supports static site generation, where pages can be pre-built at build time. This is

beneficial for pages that don't need to be dynamically generated on each request,

resulting in faster loading times.

**5.** [**API Routes**](https://nextjs.org/docs/pages/building-your-application/routing/api-routes)**:**

Next.js makes it easy to create API routes by placing files in the pages/api directory.

These routes can handle server-side logic and are accessible from the client side.

**6.** [**File-based Routing**](https://nextjs.org/docs/pages/building-your-application/routing/pages-and-layouts)**:**

The file system is used for routing, and the structure of the pages directory defines the

routes. This simplifies the organization of the application and reduces the need for

explicit route configurations.

**7.** [**Link Component**](https://nextjs.org/docs/pages/api-reference/components/link)**:**

The Link component in Next.js simplifies client-side navigation between pages. It

ensures that only the necessary parts of the page are updated when navigating, reducing

unnecessary full-page reloads.

**8.** [**Data Fetching**](https://nextjs.org/docs/pages/building-your-application/data-fetching)**:**

Next.js provides different methods for data fetching, such as getStaticProps for static

site generation and getServerSideProps for server-side rendering. These functions all

allow you to fetch data and pass it as props to your components.

**9.** [**Head Component**](https://nextjs.org/docs/pages/api-reference/components/head)**:**

The Head component in Next.js allows you to modify the document head,

including adding meta tags, title, and links to stylesheets.

**10.** [**Dynamic Routes**](https://nextjs.org/docs/pages/building-your-application/routing/dynamic-routes)**:**

You can create dynamic routes in Next.js by using brackets [] in the pages directory. For

example, pages/[id].js allows for dynamic routing based on the value of id.

**Project 2**

You are required to implement the following requirements:

*Note:* You have to use NEXT.js to create this project. It is advised to use React’s **Functional Components**. The operations will mostly be in memory if not stated otherwise. You are required to call mentioned endpoints to initialize the state.

**Modules:**

**1. Users**

* Create an authentication page where users can sign in and sign up.
* Users should have a name, email and password.

**2. Posts**

* A post should have a title.
* A post should have its content.
* The user can create a post.
* The user can edit his/her post.
* The user can delete his/her post.
* Create a posts page where all posts can be seen.

*Note:* You are required to initialize the posts state by calling the following api endpoint: <https://jsonplaceholder.typicode.com/posts>

**3. Comments**

* A post can have many comments.
* Only authorized users can comment on the post.
* The user can edit his/her own comment.
* The user can delete his/her own comment.

*Note:* You are required to initialize the comments for each fetched post by the calling following api endpoint accordingly: [https://jsonplaceholder.typicode.com/posts/{$postId}/comments](https://jsonplaceholder.typicode.com/posts/%7B$postId%7D/comments)

**Node.js and Express.js**

Node is a javascript runtime environment that runs on v8 engine and executes javascript code outside the web browser and Express is a framework for Node.js which is designed for building backend applications and APIs.

The above discussion simply means that [**node.js**](https://nodejs.org/en/docs/) allows us to run javascript at the backend (instead of browsers) and [**express.js**](https://expressjs.com/en/starter/installing.html)helps us in building APIs.

Reference Video Lectures: [Tutorial Link 1](https://www.youtube.com/watch?v=w-7RQ46RgxU&list=PL4cUxeGkcC9gcy9lrvMJ75z9maRw4byYp) (playlist), [Tutorial Link 2](https://www.youtube.com/watch?v=TlB_eWDSMt4&t=4187s)

Tip: It is recommended to watch video lectures of node first then move to the docs.

* [Introducing npm](https://docs.npmjs.com/)
* [Node.js Event Loop](https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/#what-is-the-event-loop)
* [RESTful APIs](https://developer.mozilla.org/en-US/docs/Glossary/REST)
* [HTTP Methods](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods)
* GET
* POST
* PUT
* DELETE
* [Routing and Endpoints](https://expressjs.com/en/guide/routing.html#routing)
* [Middleware](https://expressjs.com/en/guide/using-middleware.html#using-middleware)
* [Request Validation using express-validator](https://express-validator.github.io/docs/)
* [Custom Middlewares](https://expressjs.com/en/guide/writing-middleware.html#writing-middleware-for-use-in-express-apps)
* [Request Processing Pipeline](https://developer.okta.com/blog/2018/09/13/build-and-understand-express-middleware-through-examples#middleware-order-is-important)
* [Modules system in node](https://www.oreilly.com/library/view/learning-node/9781449326128/ch04.html#:~:text=Node%E2%80%99s%20module%20system%20is%20patterned%20after%20the%20CommonJS,CommonJS%20module%20system%20requirements%20implemented%20with%20Node%20are%3A)
* [EventEmitters](https://nodejs.org/api/events.html#class-eventemitter)
* [Built-in Modules](https://nodejs.org/dist/latest-v16.x/docs/api/)
* fs
* path
* http
* https
* globals
* events
* [Templating Engines](https://expressjs.com/en/guide/using-template-engines.html)
* [Job Queues using Bull](https://optimalbits.github.io/bull/)
* [Authentication using passport](https://www.digitalocean.com/community/tutorials/api-authentication-with-json-web-tokensjwt-and-passport)
* [Authorization](https://www.passportjs.org/concepts/delegated-authorization/)**,** [Blog](https://medium.com/@nil041297/mastering-role-based-authorization-in-your-mern-app-a-comprehensive-guide-with-jwt-and-passport-js-d7693143b33b)

**Testing**

* [Jest (JavaScript Testing Framework)](https://jestjs.io/docs/getting-started) [Video Reference (for jest)](https://www.youtube.com/watch?v=FgnxcUQ5vho)
* [SuperTes](https://www.npmjs.com/package/supertest)t (HTTP Assertions)
* [Testing an Express App With Supertest and Jest](https://sammeechward.com/testing-an-express-app-with-supertest-and-jest/)

**Mongoose and MongoDB**

In simple words, **MongoDB** is a database and **Mongoose** is **TypeODM** which helps us

with databases in **Node.js** by providing simple to use functions. We can perform all

to our database using Mongoose. So, we can say that its a bridge between our backend and

database.

* [MongoDB Installation](https://docs.mongodb.com/manual/installation/)
* [MongoDB Compass Installation (GUI for MongoDB)](https://docs.mongodb.com/compass/current/install/)
* [NoSQL vs SQL](https://www.ibm.com/cloud/blog/sql-vs-nosql) [(Alternate Link to SQL vs NoSQL)](https://www.mongodb.com/nosql-explained/nosql-vs-sql)
* [Connections](https://mongoosejs.com/docs/connections.html) - Connect mongodb using mongoose
* [Schema](https://mongoosejs.com/docs/guide.html)
* [Schema Types](https://mongoosejs.com/docs/api/schema.html#schema_Schema.Types)
* [​​Validation](https://mongoosejs.com/docs/validation.html)
* [Middlewares (hooks)](https://mongoosejs.com/docs/middleware.html)
* [Model](https://mongoosejs.com/docs/models.html#models)
* [Transactions](https://docs.mongodb.com/manual/core/transactions/)
* [Relationships in mongoose](https://vegibit.com/mongoose-relationships-tutorial/)
* Using References
* Using Embedded Documents
* Hybrid Approach
* [SQL to MongoDB mappings](https://docs.mongodb.com/manual/reference/sql-comparison/#sql-to-mongodb-mapping-chart)
* [Queries](https://mongoosejs.com/docs/queries.html)
* find()
* findById()
* findByIdAndDelete()
* findByIdAndRemove()
* findByIdAndUpdate()
* findOne()
* findOneAndDelete()
* findOneAndReplace()
* findOneAndUpdate()
* updateOne()
* updateMany()
* deleteOne()
* deleteMany()
* [Comparison Operators](https://docs.mongodb.com/manual/reference/operator/query-comparison/)
* eq (equal)
* ne (not Equal)
* gt (greater than)
* gte (greater than or equal)
* lt (less than )
* lte (less than or equal)
* in (in)
* nin (not in)
* [Aggregation Pipeline](https://docs.mongodb.com/manual/core/aggregation-pipeline/)
* Aggregation [Stages](https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/#std-label-aggregation-pipeline-operator-reference)
* Aggregation [Lookup vs Populate](https://medium.com/cameoeng/mongodb-lookups-and-populates-an-unexpected-journey-940e08e36a94)

**Sequelize**

Sequelize is a promise-based [Node.js](https://nodejs.org/en/about/) [ORM tool](https://en.wikipedia.org/wiki/Object-relational_mapping) for [Postgres](https://en.wikipedia.org/wiki/PostgreSQL), [MySQL](https://en.wikipedia.org/wiki/MySQL), [MariaDB](https://en.wikipedia.org/wiki/MariaDB), [SQLite](https://en.wikipedia.org/wiki/SQLite), [Microsoft SQL Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server), [Amazon Redshift](https://docs.aws.amazon.com/redshift/index.html) and [Snowflake’s Data Cloud](https://docs.snowflake.com/en/user-guide/intro-key-concepts.html). It features solid transaction support, relations, eager and lazy loading, read replication and more.

* [Getting started with Sequelize](https://sequelize.org/docs/v6/getting-started.html)
* [Model Basics](https://sequelize.org/docs/v6/core-concepts/model-basics/)
* [Model Instances](https://sequelize.org/docs/v6/core-concepts/model-instances/)
* [Model Querying - Basics](https://sequelize.org/docs/v6/core-concepts/model-querying-basics/)
* [Model Querying - Finders](https://sequelize.org/docs/v6/core-concepts/model-querying-finders/)
* [Getters, Setters & Virtuals](https://sequelize.org/docs/v6/core-concepts/getters-setters-virtuals/)
* [Validations & Constraint](https://sequelize.org/docs/v6/core-concepts/validations-and-constraints/)s
* [Associations](https://sequelize.org/docs/v6/core-concepts/assocs/)
* [Polymorphic Associations](https://sequelize.org/docs/v6/advanced-association-concepts/polymorphic-associations/)
* [Transactions](https://sequelize.org/docs/v6/other-topics/transactions/)
* [Hooks](https://sequelize.org/docs/v6/other-topics/hooks/)
* [Migration](https://sequelize.org/docs/v6/other-topics/migrations/)s
* [Scopes](https://sequelize.org/docs/v6/other-topics/scopes/)
* [Indexes](https://sequelize.org/docs/v6/other-topics/indexes/)
* [Query Interface](https://sequelize.org/docs/v6/other-topics/query-interface/)

**Project 3**

For project assignment, you are required to clear the React Theory Evaluation. Contact the **Training Coordinator** if you don’t know how to have your evaluation scheduled. Once the Theory Evaluation is clear, the Training team will assign you a Test Project on the LMS.

**State Management Tool - Redux**

Reference Video Link for Redux (Recommended): [Harvard University Class on Redux](https://www.youtube.com/watch?v=_zT8K6R_P7I&list=PLnzclEQVIaQ4FRq-0fh53dQXvN_CHtiXT&index=9)

As your project grows, you end up in a situation where too many props are being passed

up and down in the components, which is really hard to manage and not a

recommended approach as well. So, you need some state management tool to properly

manage your states. [ContextAPI](https://reactjs.org/docs/context.html)and [Redux](https://redux.js.org/introduction/getting-started)are widely used for this purpose. In this

document we will cover Redux. As recommended by the redux docs [Redux Toolkit](https://redux-toolkit.js.org/introduction/getting-started) is

the preferred approach for writing redux logic.

* [Redux Core Concepts](https://redux.js.org/introduction/core-concepts)
* [Redux Essentials](https://redux.js.org/tutorials/essentials/part-1-overview-concepts)
* [Redux Fundamentals](https://redux.js.org/tutorials/fundamentals/part-1-overview)
* [Redux Thunk](https://redux.js.org/usage/writing-logic-thunks)
* [Getting Started with Redux Toolkit](https://redux-toolkit.js.org/introduction/getting-started)
* [Redux Toolkit Quick Start](https://redux-toolkit.js.org/tutorials/quick-start)
* [Redux Toolkit Store Setup](https://redux-toolkit.js.org/api/configureStore)
* [Redux Toolkit Reducers and Actions](https://redux-toolkit.js.org/api/createReducer)
* [Getting Started with React Redux](https://react-redux.js.org/introduction/getting-started)
* [React Redux connect()](https://react-redux.js.org/api/connect)
* React Redux m[apStateToProps](https://react-redux.js.org/api/connect#mapstatetoprops-state-ownprops--object)
* React Redux [mapDispatchToProps](https://react-redux.js.org/api/connect#mapdispatchtoprops-object--dispatch-ownprops--object)
* React Redux [useDispatch()](https://react-redux.js.org/api/hooks#usedispatch)
* React Redux [useSelector()](https://react-redux.js.org/api/hooks#useselector)