# Overview

Omega - ReactJS is the set of Core Engineering practices for enabling Cognizant developers to deliver quality code to the customers. These are guidelines to help build secure, performant & maintainable code during the SDLC phases. It consists of 10 core engineering practices to ensure that the code developed by our developers passes various quality gates set internally or by our customers.

* Coding Standards
* Code Reference Slice
* Unit Testing
* Code Coverage
* Code Analysis
* Code Profiling
* Check In Policies
* Automated Build
* Continuous Integration
* Code Promotion

The engineering practices covers all critical aspects related to application development & helps in establishing governance process as well as to ensure better code quality.

# ReactJS

React is a powerful library developed by Facebook. It is used for handling the view layer for web and mobile apps. ReactJS allows us to create reusable UI components.

**React** is a JavaScript library for building user interfaces. It encourages the creation of reusable UI components, which present data that change over time. Lots of people use React as the V in MVC. React abstracts away the DOM from you, offering a simpler programming model and better performance. React can also render on the server, and it can power native apps using React Native. React implements one-way reactive data flow, which reduces the boilerplate and is easier to reason about than traditional data binding.

|  |  |
| --- | --- |
| **Suggested React IDEs** | Microsoft Visual Studio Code  WebStorm  Spring Eclipse IDE  Sublime  Notepad++ (at least) |
| **Prerequisites** | [Node.js](https://nodejs.org/en/download/)  [NPM](https://www.npmjs.com/) (Node Package Manager) |
| **Scaffolding & Build Tool** | [webpack](https://webpack.js.org)  Other Build Tools ([Gulp](http://gulpjs.com/), [Grunt](https://gruntjs.com/)) |

Latest version of React: v16.3.2

|  |  |
| --- | --- |
| **React Official Website** | <https://reactjs.org/> |
| **Docs** | [React Docs](https://reactjs.org/docs/hello-world.html) |
| **Tutorials** | <https://reactjs.org/tutorial/tutorial.html> |
| **React Seed Project** | <https://github.com/facebook/create-react-app> |

**React Components** let you split the UI into independent, reusable pieces, and think about each piece in isolation. React.Component is provided by [React](https://reactjs.org/docs/react-api.html).

**Overview:**

React.Component is an abstract base class, so it rarely makes sense to refer to React.Component directly. Instead, you will typically subclass it, and define at least a [render()](https://reactjs.org/docs/react-component.html) method.

Normally you would define a React component as a plain [JavaScript class](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Classes):

class Greeting extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

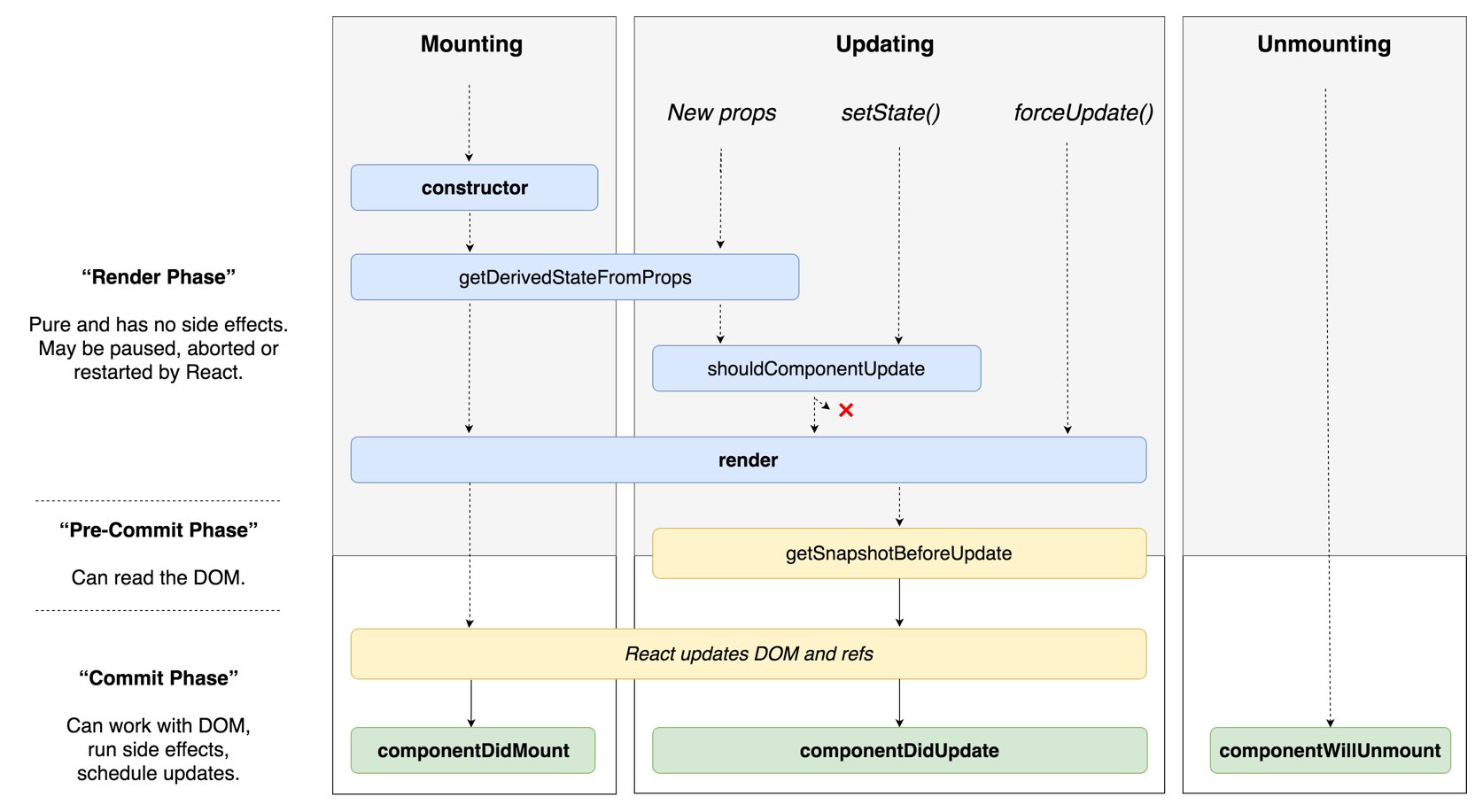
}

If you don’t use ES6 yet, you may use the create-react-class module instead. Take a look at [Using React without ES6](https://reactjs.org/docs/react-without-es6.html) to learn more.

Note that **we don’t recommend creating your own base component classes**. Code reuse is primarily achieved through composition rather than inheritance in React.

**The Component Lifecycle:**

Each component has several “lifecycle methods” that you can override to run code at particular times in the process. Methods prefixed with will are called right before something happens, and methods prefixed with did are called right after something happens.



## State Management

* The problem with traditional MVC Architecture

Before diving deeper into the concepts of Redux let’s first focus on the problems it solves. The Model-View-Controller (MVC) pattern is familiar to most front-end web developers nowadays. This pattern describes a separation between the data (model), the presentation (view) and the application logic (controller). This ensures that your application is built in a structured way and that you achieve a separation of concerns.   
   
 However, the disadvantage is that you’re loosing control of your data flow. In general, the data flow is bi-directional. The user input in one component can affect other components and vice versa. Controlling the flow of data and making sure that all user interface components update accordingly is an error-prone task.

* How State Management solves the problem

By using Redux we’re solving this problem by introducing a central data store in our application. The store contains the state of the application and is the source of truth for components. By using the store concept, you do not need to synchronize state between components manually. Instead you can fully rely on the Redux store at any time.

## Virtual DOM

* What is the Virtual DOM?

The virtual DOM (VDOM) is a programming concept where an ideal, or “virtual”, representation of a UI is kept in memory and synced with the “real” DOM by a library such as ReactDOM. This process is called [reconciliation](https://reactjs.org/docs/reconciliation.html). More on Reconciliation: <https://reactjs.org/docs/reconciliation.html>

This approach enables the declarative API of React: You tell React what state you want the UI to be in, and it makes sure the DOM matches that state. This abstracts out the attribute manipulation, event handling, and manual DOM updating that you would otherwise have to use to build your app.

Since “virtual DOM” is more of a pattern than a specific technology, people sometimes say it to mean different things. In React world, the term “virtual DOM” is usually associated with [React elements](https://reactjs.org/docs/rendering-elements.html) since they are the objects representing the user interface. React, however, also uses internal objects called “fibers” to hold additional information about the component tree. They may also be considered a part of “virtual DOM” implementation in React.

* Is the Shadow DOM the same as the Virtual DOM?

No, they are different. The Shadow DOM is a browser technology designed primarily for scoping variables and CSS in web components. The virtual DOM is a concept implemented by libraries in JavaScript on top of browser APIs.

* What is “React Fiber”?

Fiber is the new reconciliation engine in React 16. Its main goal is to enable incremental rendering of the virtual DOM.

More on React Fiber: <https://github.com/acdlite/react-fiber-architecture>

Below are the few practices recommended as part of a react app based on requirement:

**Authentication**: JWT is probably the best way to handle authentication for SPAs built using react (or any other framework for that matter). Have to figure out how we will tie authentication to route transitions, how will you handle authentication failures or session expiry.

**Server Side Rendering**: I think this is a mandatory requirement for any new web app. It is fairly straightforward to implement it.

**Animations**: ReactMotion is very popular but GreenSock is really powerful.

**Error Recovery & Reporting**: How will you report unexpected errors and recover from them? This depends a lot on the other architectural choices you would’ve made. **Sentry** is a popular choice for error reporting.

# Coding Standards

A good piece of code should be easy to understand and tell us what is going on — not more and not less. This can be achieved by writing sensible code by following the simple Universal Standards.

Each developer should feel accountable for his/her piece of code and deliver a good chunk of code on each day of the project that ultimately leads to a Quality Product at the end.

Coding Standards benefit both the developers and the application in the following ways.

* Suggests Syntax, Naming Conventions, Definition & Declaration of the Variables and Methods, Application Structure of the project, etc.
* Guides all the developers to follow the same set of standards across the application that keeps the code more maintainable and consistent throughout Development, Testing and Maintenance phases.
* Increases **Readability**, **Maintainability** and **Quality** of the code.



# Code Reference Slice

<https://reactjs.org/versions> site provides the latest updates about the React related to the versions, fixes, tutorials, etc.. Each topic of React is explained conceptually with an example in this site. We can start with the [Create React app](https://github.com/facebook/create-react-app) and develop it by eventually implementing the concepts as we learn. Beginners can start with this web site and each concept is walkthrough in the simplest manner.

# Unit Testing

We need to write at least basic unit tests to make sure regressions don't spring up because some new developer doesn't understand what's going on. The core units of the application should be verified with accompanying unit tests. In JavaScript apps, the smallest units of code we can test are usually individual functions.

Each developer has to verify and make sure his/her chunk of code works perfectly as intended. So, when all the small chunks of code are joined together, they have a good chance of working as a whole.

**Jest** is preferred to do the Unit Testing. [Facebook's Jest](https://facebook.github.io/jest/docs/en/tutorial-react.html) hand walks the developer by explaining its implementation.

## Important References for Unit Testing

|  |  |
| --- | --- |
| **Jest Unit Testing** | <https://facebook.github.io/jest/docs/en/tutorial-react.html> |
| **Jest Unit Testing**  **Best Practices** | <https://facebook.github.io/jest/docs/en/tutorial-react.html> |

# Code Coverage

One of the important things in the phase of Unit Testing is Code Coverage. It is a measure used to describe the degree to which the source code of a program is executed when a particular test suite runs. A program with high code coverage, measured as a percentage, has had more of its source code executed during testing which suggests it has a lower chance of containing undetected software bugs compared to a program with low code coverage.

Jest’s Built-in code coverage reports: Easily create code coverage reports using [--coverage](https://facebook.github.io/jest/docs/en/cli.html). No additional setup or libraries needed! Jest can collect code coverage information from entire projects, including untested files.

# Code Analysis

Static Code Analysis is a method of computer program debugging that is done by examining the code without executing the program. It is a collection of algorithms and techniques used to analyze source code in order to automatically find potential errors or poor coding practices. The process provides an understanding of the code structure, and can help to ensure that the code adheres to industry standards.

It commonly refers to the running of Static Code Analysis tools that attempt to highlight possible vulnerabilities like syntax errors within static source code.

It is performed as part of a Code Review that happens without the manual intervention. It is advised to carry out continuously thorough out the development of the project

## ESLint

ESLint is an open source JavaScript linting utility originally created by Nicholas C. Zakas in June 2013. Code [linting](https://en.wikipedia.org/wiki/Lint_(software)) is a type of static analysis that is frequently used to find problematic patterns or code that doesn’t adhere to certain style guidelines. There are code linters for most programming languages, and compilers sometimes incorporate linting into the compilation process.

JavaScript, being a dynamic and loosely-typed language, is especially prone to developer error. Without the benefit of a compilation process, JavaScript code is typically executed in order to find syntax or other errors. Linting tools like ESLint allow developers to discover problems with their JavaScript code without executing it.

|  |  |
| --- | --- |
| **ESLint** | [https://eslint.org](https://eslint.org/docs/rules/) |
| **ESLint Rules Explained** | <https://eslint.org/docs/rules/> |
| ESLint can be configured as a plugin in Microsoft Visual Code, WebStorm, Sublime, and Spring Eclipse IDEs to get the errors during the development. | |

Please do configure all the rules that are recommended in the ESLint website for the better quality of the static code.

# File Structure

Is there a recommended way to structure React projects?

React doesn’t have opinions on how you put files into folders. That said there are a few common approaches popular in the ecosystem you may want to consider.

* Grouping by features or routes

One common way to structure projects is locating CSS, JS, and tests together inside folders grouped by feature or route.



The definition of a “feature” is not universal, and it is up to you to choose the granularity. If you can’t come up with a list of top-level folders, you can ask the users of your product what major parts it consists of, and use their mental model as a blueprint.

* Grouping by file type

Another popular way to structure projects is to group similar files together, for example:



Some people also prefer to go further, and separate components into different folders depending on their role in the application. For example, [Atomic Design](http://bradfrost.com/blog/post/atomic-web-design/) is a design methodology built on this principle. Remember that it’s often more productive to treat such methodologies as helpful examples rather than strict rules to follow.

## SonarQube – Continuous Code Quality

SonarQube collects and analyzes source code, measuring quality and providing reports for our projects. It combines static and dynamic analysis tools and enables quality to be measured continuously over time.

Everything that affects our code base, from minor styling details to critical design errors, is inspected and evaluated by SonarQube, thereby enabling developers to access and track code analysis data ranging from styling errors, potential bugs, and code defects to design inefficiencies, code duplication, lack of test coverage, and excess complexity.

The Sonar platform analyzes source code from different aspects and hence it drills down to our code layer by layer, moving from the module level down to the class level. At each level, SonarQube produces metric values and statistics, revealing problematic areas in the source that require inspection or improvement.



|  |  |
| --- | --- |
| **SonarQube** | <https://www.sonarqube.org/> |
| **SonarQube Configuration** | [Configuration](file:///D:\Kondal_data\C&DS\AdvJS\Seed\React\Sonar%20Plugin%20for%20the%20Code%20Analysis%20Dashboard%20for%20Angular.docx) |

# Code Profiling

In general, Browser Developer Tools are used to profile JavaScript and Typescript.

React Developer Tools is the most used Google Chrome Developer Tool extension for debugging and profiling React applications. [React Dev Tools](https://github.com/facebook/react-devtools) helps React developers visualize the application through component trees, and visual debugging tools. Developers get immediate insight into their application structure, change detection and performance characteristics.

|  |  |
| --- | --- |
| **New React**  **Developer Tools** | <https://reactjs.org/blog/2015/09/02/new-react-developer-tools.html> |
| **React Dev Tools** | <https://github.com/facebook/react-devtools> |

# Build Process

Builds are created when a certain point in development has been reached or the code has been deemed ready for implementation, either for testing or for production.

The following multiple tasks are to be combined into a single command and configured as part of the Build process for Production.

* SASS Preprocessor (Compile SASS/SCSS/LESS to CSS)
* Autoprefix CSS with PostCSS (Automatically add vendor prefixes)
* Linting (Using ESLint)
* Babel Transpiler (Transpile the code)
* Uglifying JavaScript files (Minification)
* Chunking or Code splitting (lazy loading modules)
* Unit Testing
* Bundling

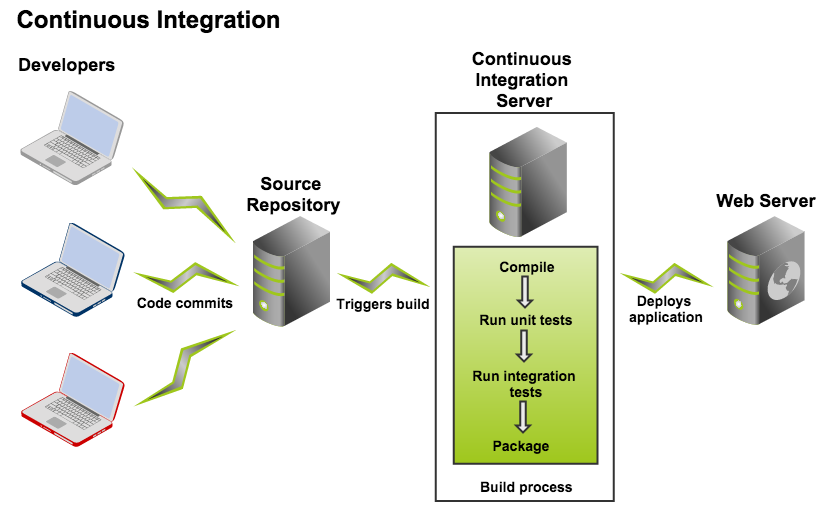
Following are the build and automation tools:

* Webpack
* Grunt
* Gulp
* Browserify
* Yeoman

# Continuous Integration

Continuous Integration(CI) is a software development methodology in which a build, unit tests and integration tests are performed, or triggered, whenever code is committed to the repository, to ensure that new changes integrate well into the existing code base. Integration builds provide early 'fail fast' feedback on the quality of new changes.

Release management describes the steps that are typically performed to release a software application, including building and functional testing, tagging releases, assigning versions, and deploying and activating the new version in production.



Below are the tools for Continuous Integration:

**Bamboo CI server**: Bamboo is a continuous integration (CI) server that can be used to automate the release management for a software application, creating a continuous delivery pipeline.

**Jenkins**: Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.

The architecture and design guidelines are attached below:

