**Amex NeMo**

**Technical Governance Document V1.0**

****

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Prepared By** | **Reviewed By** | **Approved By** |
| **Name** | Anil Krishna &Prince Antony |  |  |
| **Role** |  |  |  |
| **Signature** |  |  |  |
| **Date** |  |  |  |

Table of Contents

[1.0 Introduction 4](#_Toc535220795)

[1.1 Purpose of this document 4](#_Toc535220796)

[1.2 Scope 4](#_Toc535220797)

[1.3 Intended Audience 4](#_Toc535220798)

[1.4 Acronyms 4](#_Toc535220799)

[2.0 Sprint Execution governance 5](#_Toc535220800)

[2.1.1 Scrum activities 5](#_Toc535220801)

[3.0 Development governance 7](#_Toc535220802)

[3.1 Amex Framework – One App 7](#_Toc535220803)

[3.2 Best practices and patterns to follow 8](#_Toc535220804)

[3.2.1 Components 8](#_Toc535220805)

[3.2.2 Component life cycle 9](#_Toc535220806)

[3.2.3 Services 9](#_Toc535220807)

[3.2.4 Routing 9](#_Toc535220808)

[3.2.5 State management - Redux 10](#_Toc535220809)

[4.0 Amex NeMo development guideline 11](#_Toc535220810)

[4.1 Development specification 11](#_Toc535220811)

[5.0 Consolidated API View 12](#_Toc535220812)

[5.1 Request – Response Mapping 12](#_Toc535220813)

[6.0 Coding Check list 13](#_Toc535220814)

[6.1 Code Review & Traceability Template 14](#_Toc535220815)

[7.0 Testing 14](#_Toc535220816)

[7.1 Unit Testing 14](#_Toc535220817)

[7.2 Important References for Unit Testing 14](#_Toc535220818)

[8.0 Code Coverage. 15](#_Toc535220819)

[9.0 Code Quality 15](#_Toc535220820)

[9.1 ESLint 15](#_Toc535220821)

[9.2 source-map-explorer 15](#_Toc535220822)

[9.3 React-developer-tool 15](#_Toc535220823)

[10.0 Branching and Merging Strategy 16](#_Toc535220824)

[11.0 Build & Deployment Strategy 16](#_Toc535220825)

[12.0 References 16](#_Toc535220826)

[13.0 Change History 17](#_Toc535220827)

# Introduction

## Purpose of this document

The purpose of this document is to provide development governance for Amex NeMo Project.

## Scope

The document covers the following

1. Project execution governance
2. Development governance
3. Review governance
4. Code promotion: Branching and Merging governance

## Intended Audience

* Development Team
* Architect
* Scrum Master
* Product Owner

## Acronyms

|  |  |
| --- | --- |
| **NFR** | Non Functional Requirement |
| **DOF** | Definition of Ready |
| **DOD** | Definition of Done |
| **PO** | Product Owner |
| **SM** | Scrum Master |
| **ST** | Scrum Team |
| **SLA** | Service Level Agreement |
| **CR** | Change Request |
| **CI** | Continues Integration |
| **CD** | Continues Development |
| **PR** | Pull Request |
| **HOC** | Higher Order Component |

# Sprint Execution governance

### Scrum activities

|  |  |
| --- | --- |
| **Sprint Planning meeting**  **Participants: PO, SM & ST** | * Select a user story based on the priority from the product backlog * Team brainstorms the story, size it based on the complexity. * Identify the tasks and effort (points or hours) – Should have an agreement between Amex NeMo Offshore team and other stockholders * API Dev team should be part of this meeting |
| **Sprint User Story Definition of Ready** | * User Story clearly defined in Rally or\and in Enterprise Confluence tool * User Story dependencies identified– Like inter story dependency along with the sequence * All NFR should be defined before we start the user story * All APIs design should be ready in the API interface document (refer below) * API mock and Swagger link should be ready when development start * All APIs should be ready by 2nd week of sprint, else integration testing will not able to complete * Clarifications/Query SLA should be defined in advance to avoid impact on the sprint * Need to define on how we will accommodate the mandatory CR, assumptions during the sprint which will impact the on-going sprint |
| **Execution of Sprint**  **Participants: ST & SM** | * Development – Will be done based on the user story tasks, confluence page, design and api references * Developers should follow coding standard and the best practices which is given in the below * Team should update their respective Rally user stories/tasks * Daily standup: Every day the scrum team meets to - not more than 15 minutes- state 3 points for each team members   + What did yesterday   + What will do today   + Any impediments (roadblocks) * In case, any team member is facing any kind of impediments, the scrum master follows up to get it resolved * Code review – Peer code review and reviews from leads will be done in the phase based on the PR to make sure all the functional requirements are met , coding standards are followed (Refer the coding standard check list) and the best practices are applied * Technical scrum : Daily or on demand technical/clarification with onsite lead, API team and product owner for the development impediment * Make sure unit testing code coverage by using Jest UI framework is 100% * Branching and merge strategy will flow as mentioned in this below document * Code merge and send to E1 for PO review and user story acceptance |
| **Sprint User Story Definition of Done** | * Completed User Story meets all of the acceptance criteria described in Rally and Confluence * 100% Unit Test coverage.  Visible to everyone via dashboard or Jenkins * No open showstopper defects * The code review using Bitbucket Pull Requests is completed. * Screenshots are attached to user story * Integrated functional testing should be done from the e1 environment for the quality approval * The code is executable based on the e1 environment data |
| **Sprint Review Meeting**  **Participants: PO, SM and ST** | * At the end of every sprint, scrum team meet again and demonstrates the implemented user stories to the product owner * The product owner should cross verify the stories as per its acceptance criteria. * SM to preside over this meeting * In the Rally tool, the Sprint is closed and the tasks are marked done |
| **Retrospective Meeting**  **Participants: PO, SM and ST** | * It should happens after the review meeting for each sprint * Team will meet and discuss on the below points * What went well during the Sprint (Best practices) * What did not go well in the Sprint * Lessons learned * Action Items |
| **Demo to Business/Customer** | * There should be a demo session for the shippable sprints to Business/Customer/Amex leaders. This will strengthen team’s confidence and motivation |
| **Sprint window** | * 2 weeks of 10 working days sprint * Sprint starts on alternate Tuesday * Team size is 10 cross functional developers and a lead from offshore * One lead from onshore * One Amex lead to approve technical approach and designs |

# Development governance

## Amex Framework – One App

All developers should follow Amex One App framework which is a platform for American Express React components. It will provide us an established pattern, development approach, approved tools and 3rd party libraries, deployment process and testing strategy. As per the One App framework, all ReactJs based project UI component should follow the below architecture

|  |
| --- |
|  |

Refer to [OneApp](https://one-dev.aexp.com/v2/guide/#One-App) Amex application for more details

## Best practices and patterns to follow

### Components

* The application should have a separation of concern with clear division of business logic and view logic.
* React components can divide as many numbers of components to improve application maintainability and reusability.
* Build encapsulated components that manage their own state, then compose them to make complex UIs. It helps to create reusable components.
* It is always advisable to use functional components, where the main purpose of the Component is to take the props and convert it into UI
* All data manipulations, api interaction through services and other business logics should be in the container components
* Use ‘axp-base’ amex built-in components for the presentational components
* Use React Context to pass data through the component tree without having to pass props down manually at every level
* Do not include component state management and manage through props only
* Include prop-types clearly and do not use module specific names.
* If one more components do the similar activity and following the same code pattern, then there could be a chance to create a HOC
* A component should striving for the single responsibility
* Data binding pattern should follow Lazy and Non-Lazy loading approach for performance

### Component life cycle

* Developers should use component life cycle at the right place
* Constructor should use to initialise all state and to assign default props
* [componentDidCatch()](https://reactjs.org/docs/react-component.html#componentdidcatch) must be used for component level error handling
* componentDidMount() should use to fetch the api data to the component
* componentWillUnmount() should use to clear all local state and redux states
* Make sure we will not use any deprecated life cycle methods like
  + componentWillMount()
  + componentWillUpdate(nextProps, nextState)
  + componentWillReceiveProps(nextProps)

### Services

* The API services are provided with separate folder structure and components may call or trigger the corresponding event to load the data.
* The api method fetch() is used to call Network Util nodejs methods from reactjs
* We need a generic API layer for making service call(Network Util)from the react application which will avoid redundant code in the service layer
* Network Util NodeJs layer should be a generic service layer to accept GET, POST, DELETE and PUT methods
* All logging must be done in the Network Util layer for the exceptions analysis

### Routing

We should follow Amex Routing Framework by using ‘holocron-module-route’. Refer Holocron Routing for more details

### State management - Redux

* Use Redux stores (axp-global-ducks) for all the state management and data is sent to React Components as props, so the data in the View layer is displayed using the React Components.
* Global state used for across all modules
* Module or component based state.

**

React Components : The functional , typed and presentations components(Components & Containers)

**Redux**: An application state management architecture (axp-global-ducks)

**Actions**: Actions provides a way to provide the type of action and payload to the Store.

**Dispatch**: All actions are dispatched to the redux store using the dispatch function.

**Reducers**: Specify how the application's state changes in response to actions sent to the store

**Store**: To hold application state

**Async Operations**: All the async operations like API, are handled by Redux-thunk Middleware

**Network Util** : It is the wrapper node.js script to connect APIs from React application

**Exception/Error** : All errors will push to local couch database/file for further verification

# Amex NeMo development guideline

## Development specification

|  |  |
| --- | --- |
| **Open Source** | |
| **Development environment editor** | Visual Studio Code |
| **Development library** | ReactJS 16.2 |
| **UI unit testing framework** | Jest |
| **Module bundler** | Webpack |
|  |  |
| **HTTP client api** | fetch |
| **CSS pre-processor** | SAAS |
| **Scripting standard/type** | ES6 |
| **Styles** | Css3 |
| **Responsive page** | Bootstrapper |
| **Utility library** | Lodash |
| **Software Configuration tool** | Bitbucket |
| **State management** | react-redux |

|  |  |
| --- | --- |
| **Amex In House** | |
| **Client State management** | axp-global-ducks, Holocron |
| **Reusable components** | axp-base |
| Module bundler | one-amex-bundler |
| Router | [holocron-module-route](https://stash.aexp.com/stash/projects/ONE/repos/holocron/browse/packages/holocron-module-route) |
|  |  |

# Consolidated API View

## Request – Response Mapping

The below details should be there as part of the API design to avoid rework and sooth UI development. API team can work with UI development team to make sure the tam is in the same understanding on the requirement. Below table can be move to an excel template for the maintainable purpose

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Screen Name** | **Screen Element** | **Service Name** | **Type of Service Operation** | **Request Field** | **Response Field** | **Remarks** |
| <An unique name / identifier for the screen> | <Identify an element on the screen which gets mapped to a particular service response field / an element that triggers a service invocation> | <Specify the name of the service with the complete api endpoint url> | GET/ POST/ PUT/ DELETE | <For POST operation, map the screen element with service element> | <For GET operation, map the screen element with service element> | <Optionally, add any relevant comments for extra headers, token or any other parameters and validations> |

# Coding Check list

|  |  |  |
| --- | --- | --- |
| 1 | All violations coming out of esLint have been fixed | Yes |
| 2 | All Components must have proptypes | Yes |
| 3 | All the list components are build with a key value | Yes |
| 4 | Minimum code in the render method and logic should be in helper methods. | Yes |
| 5 | Split the React Component till it handles single responsibility | Yes |
| 6 | No editing of props within the React Component | Yes |
| 7 | Name the boolean variables starts with "is", "has", "should" etc.. | Yes |
| 8 | Use the default values in the function. Use these default values , wherever applicable. Set default props, where for props of component, if applicable. | Yes |
| 9 | Use spread operators instead of Object.assign | Yes |
| 10 | Change all var and let to const, wherever possible. | Yes |
| 11 | No mutation of state in case of redux reducer | Yes |
| 12 | Remove commented code. No code should exist which is commented in the file. | Yes |
| 13 | For each file corresponding test file exists. | Yes |
| 14 | Each component should be a separate file. | Yes |
| 15 | Use Error Boundary component where applicable and show corresponding error message. | Yes |
| 16 | Use of constants instead of hard coded values. | Yes |
| 17 | Naming Conventions for Classes, filenames are followed | Yes |
| 18 | Proper folder structure is followed. | Yes |
| 19 | Don't use this.state, inside a setState function Call setState, with a function and previousState as a variable. this.setState( (prevState,props) => { return ( {value: prevSate.value + props.value } )} | Yes |
| 20 | Intialize the state in the component with expected default values. | Yes |
| 21 | Don't use ComponentWillMount, instead of useComponentDidMount.  "ComponentWillMount" will be deprecated. | Yes |
| 22 | Don't call, setState inside a "shouldComponentUpdate" it will cause infinite loop. | Yes |
| 23 | Don't call setState inside a render method, as setState triggers render and render is updating the state. | Yes |
| 24 | Don't call setState in componentWillUnmount, as the component is unmounting anyway. | Yes |
| 25 | The best place for setState, ComponentDidMount and in the event actions based on the trigger. | Yes |

## Code Review & Traceability Template



# Testing

## Unit Testing

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.

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 Quality

## ESLint

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/> |

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

## [source-map-explorer](https://www.npmjs.com/package/source-map-explorer)

[source-map-explorer](https://www.npmjs.com/package/source-map-explorer) is great. It shows you a tree map visualization to help you debug where all the code is coming from. Thus it’s easy to analyze and debug JavaScript (or Sass or LESS) code bloat through source maps.

## React-developer-tool

React Developer Tools is a Chrome DevTools extension for the open-source React JavaScript library. It allows you to inspect the React component hierarchies in the Chrome Developer Tools.

# Branching and Merging Strategy

We will follow the below branching strategy to minimize the code conflict and merge issues. Working branches will be created based on the modules (features) instead of the User Story. Once the module branch PRs are approved, and then lead will merge code to the Developer branch at the end of the sprint.

|  |
| --- |
|  |

# Build & Deployment Strategy

*Map it to Amex documentation*

# References

**https://reactjs.org/**

**https://github.com/reactjs**

[**https://github.com/facebook/react**](https://github.com/facebook/react)

[**https://redux-saga.js.org/**](https://redux-saga.js.org/)

[**https://redux.js.org/**](https://redux.js.org/)

# Change History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version Number | Changes Made | | | |
| V1.0 | <<First version>> | | | |
| V1.1 | <<If the change details are not explicitly documented in the table below, reference should be provided here>> | | | |
| Page No. | Changed by | Effective Date | Changes Effected |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| V1.2 | <<If the change details are not explicitly documented in the table below, reference should be provided here>> | | | |
| Page No. | Changed by | Effective Date | Changes Effected |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |