Frontend Design

Document Status	DRAFT
Document Owner	Vinay Lingampally
Document Approver	David Alters, Bryan Green, Marcin Susel
Category	Architecture

Table of Contents

1 Overview
    1.1 Component Breakdown
    1.2 Application Structure
2 Framework
    2.1 Introduction
    2.2 Vue Components
        2.2.1 Vue component Life-cycle
3 Tooling & Development
    3.1 Package Manager
    3.2 Task Runner
        3.2.1 Vue-CLI
        3.2.2 Webpack
    3.3 Linting & Enforcing
        3.3.1 ES Lint
        3.3.2 Editor Config
    3.4 Testing Methodologies
        3.4.1 Unit Testing
        3.4.2 Code Coverage
        3.4.3 Code Coverage Report
        3.4.4 End to End Testing
4 Project Template
5 State Management
    5.1 Introduction
    5.2 Dataflow
    5.3 Usage
6 Build
7 Performance Strategies
    7.1 Compilation
        7.1.1 Ahead-of-Time
        7.1.2 Just-in-Time
    7.2 Lazy Loading
8 Monitoring
9 Style Guide
10 Integrations
    10.1 Axios
11 Appendix

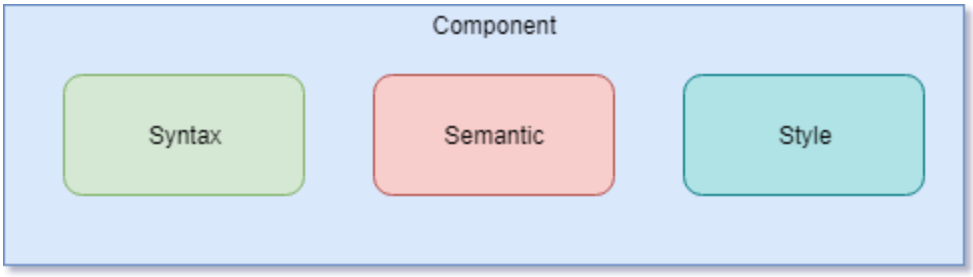
Overview

As a Kona Engineer, I want to define the Design & Approach for the front-end system of the LaaS Platform.

We will be following Component Based Design practice to build our app, Component Based Design is the practice where we split our functionality into smaller, more manageable parts. Which can be reused and customized on need, across the application.

We will define component based on Syntax, Semantic & Styling aspects.

Component Breakdown



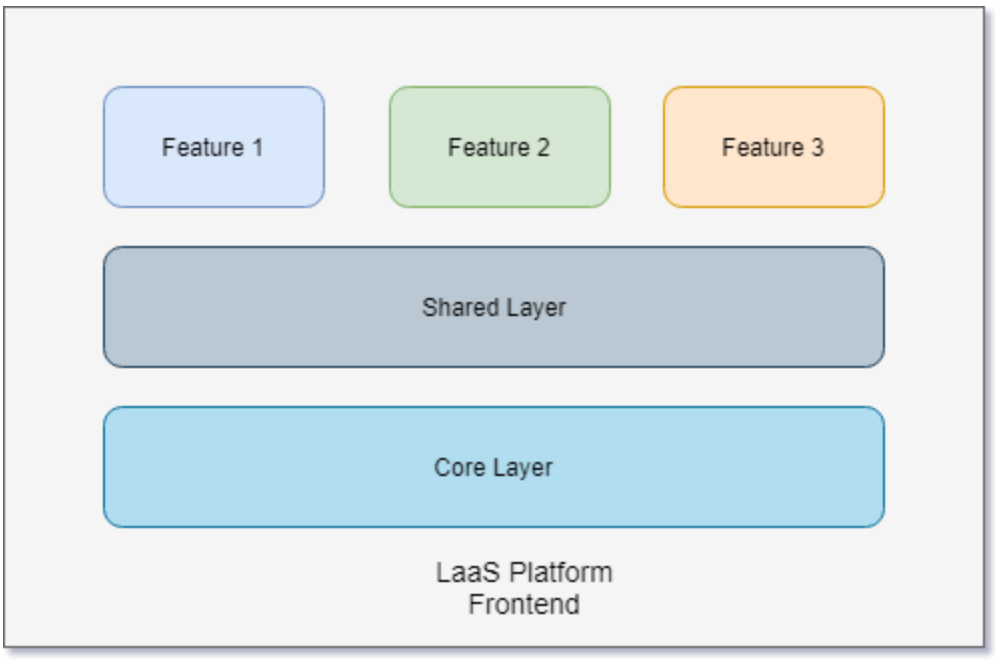
Syntax: It talks about the structure of component. A component will have container or parent element which will be the point of reference for the component. (It is usually the .html or the .vue files in the application)

```html
<div class="bp-dropdown">
    <button class="bp-btn" type="button" id="dropdownMenuButton">
        Dropdown button
    </button>
    <div class="bp-dropdown-menu" aria-labelledby="dropdownMenuButton">
        <a class="bp-dropdown-item" href="#">Action</a>
        <a class="bp-dropdown-item" href="#">Another action</a>
        <a class="bp-dropdown-item" href="#">Something else here</a>
    </div>
</div>
```

Semantic: Semantic talks about the logic of the component and behavior of various interactions with the component. (like what on-click, on-select).
Styling: A definition of attributes of the component which controls how the component is displayed.

```scss
.bp-dropdown{
    .bp-btn{
        font: 1em;
        &:hover{
            background-color: beige;
                }
    }
    .bp-dropdown-menu{
        .bp-dropdown-item{
            color: black;
        }
    }
}
```

Application Structure



Core: Layer where all the core functionality of the application like the Authentication, Session, Local Storage utility are defined.
Shared: Layer where all shared components or UI toolkit are defined.
Feature: Functionality related to each of the feature is implemented.

Framework

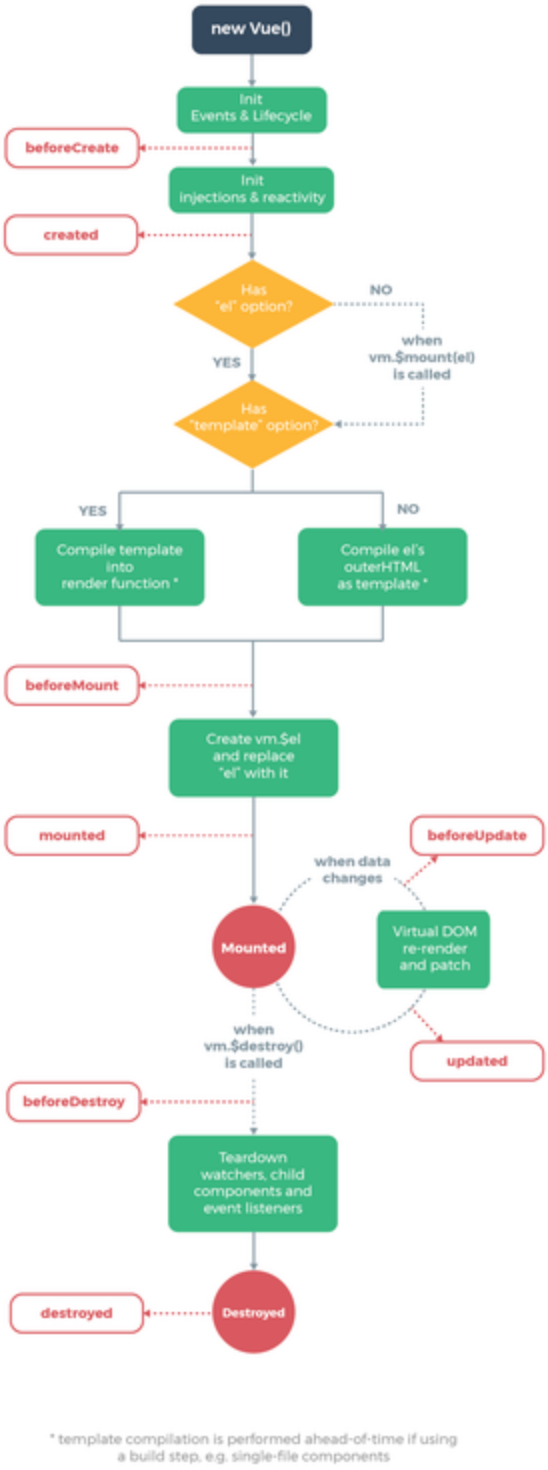
Introduction

Vue (pronounced /vju/, like view) is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries.

Vue Components

The component system is another important concept in Vue, because it's an abstraction that allows us to build large-scale applications composed of small, self-contained, and often reusable components.

Vue component Life-cycle



Source: https://vuejs.org/v2/guide/instance.html

Reusable Components

By Default Vue components by nature are meant to be re-used. This is easy when the component is only used within a single application. Going forward based on our requirements we will be identify and develop our common components that will be using in all feature modules so that we need shared folder which can be sharing to all the modules.

Example:

I have requirement to use the customized drop-down component across all the modules so for that we will create one reusable drop-down

component under share folder. After creation of drop-down we just import wherever is needed.

```
<div class="filter">
  <label for="component-dropdown">Component-based dropdown: </label>
  <dropdown id="component-dropdown" :options="fruitOptions"></dropdown>
</div>
```

Through NPM Package

But in the future, if you want use re-usable component in multiple sites/applications. So we will create npm package on demand if project requires but I will explain the packaging components to be shared via npm, it can be imported/required into a build process for use in full-fledged web applications

Vue Routing

Introduction

The Vue Router library is the way to go for Vue.js applications. Vue does not enforce the use of this library. You can use whatever generic routing library you want, or also create your own History API integration, but the benefit of using Vue Router is that it's official.

This means it's maintained by the same people who maintain Vue, so you get a more consistent integration in the framework, and the guarantee that it's always going to be compatible in the future, no matter what.

New Route Instance Creation

Although, we can install the vue-router by default while we were creating a new project, let me show how to install separately. So that we will integrate on our own.

```
npm install vue-router --save
```

I have installed the version 3.1.2 as per the tech stack.

Now, we can import the router inside the Vue.js application. So let us import inside the **src >> main.js** file.

```
// main.js

import Vue from 'vue'
import VueRouter from 'vue-router'

Vue.use(VueRouter)
```

Inside the **src** folder, we will create one file called **routes.js** and import all the components (this is example) inside **src >> routes.js** file. We will define the routes for each component.

In the future, when other feature modules are developed, we will define their routes inside this file.

```
const Promotion = () =>
  import(/* webpackChunkName: "promotion-welcome" */ "./components/promotion-welcome");

const PromotionDetails = () =>
  import(/* webpackChunkName: "promotion-details" */ "./components/promotion-details");

const PromotionAdd = () =>
  import(/* webpackChunkName: "product-add" */ "./components/promotion-add");

const routes = {
  path: "/main/promotion",
  component: Promotion,
  children: [
    {
      path: "/main/promotion-management/promotion-details",
      component: PromotionDetails
    },
    {
      path: "/main/promotion-management/promotion-add",
      name: "promotion-add",
      component: PromotionAdd
    }
  ]
};
export default routes;
```

Next step is to create a VueRouter instance and pass the routes object. So, we need to import **routes.js** file inside the **main.js** file.

```
// main.js

import Vue from 'vue';
import App from './App.vue';
import VueRouter from 'vue-router';

import routes from './routes';

Vue.config.productionTip = false;

Vue.use(VueRouter);

const router = new VueRouter({routes});

new Vue({
    router,
    render: h => h(App)
}).$mount('#app');
```

So, we have passed the router object while creating a vue instance.

Usage

Each feature module will have the route.js file to define components routes. There will be one app level route.js which include all feature module routes. Here we will be using lazy loading concept while implementing routing for each feature module level. This means there will generate chunk file each lazy load module or component.

Tooling & Development

Package Manager

Package managers allow you to grab dependencies from external hosts. For example we will use vue for our project and it will make sure if we want a latest copy or certain version to be update into our project.
We will use “npm” as our package manager

```
npm init
npm install
```

Configuration

```
{
  "name": "bp-laas",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "lint": "vue-cli-service lint"
  },
  "dependencies": {
    "core-js": "^2.6.5",
    "vue": "^2.6.10"
  },
  "devDependencies": {
    "@vue/cli-plugin-babel": "^3.11.0",
    "@vue/cli-plugin-eslint": "^3.11.0",
    "@vue/cli-service": "^3.11.0",
    "babel-eslint": "^10.0.1",
    "eslint": "^5.16.0",
    "eslint-plugin-vue": "^5.0.0",
    "vue-template-compiler": "^2.6.10"
  },
  "eslintConfig": {
    "root": true,
    "env": {
      "node": true
    },
```

Task Runner

Task runners will allow you to configure particular tasks depending on what you’re aiming to achieve. Managing third party code and their dependencies should not be a manual task performed by a human, it’s not productive.
For example, you can use a particular command from a task runner to start a local server, compile all assets and then serve those assets in your browser with ease. Webpack has become the default standard with Angular as it can run your tasks, tests, compile your code and serve locally - as well as much more

Vue-CLI

The CLI (@vue/cli) is a globally installed npm package and provides the vue command in your terminal. It provides the ability to quickly scaffold a new project via vue create, or instantly prototype new ideas via vue serve. You can also manage your projects using a graphical user interface via vue ui.

Webpack

Webpack is a static module bundler for modern JavaScript applications. When Webpack processes your application, it internally builds a dependency graph which maps every module your project needs and generates one or more bundles.

Linting & Enforcing

When working on a team, the goal should be that each file is written as if it were coded from a single developer's mind regarding error prevention, formatting, and style

ES Lint

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.

Rules:

We will be using standard lint rule for our application and we will be using Glob Pattern as Sometimes a more fine-controlled configuration is necessary, for example if the configuration for files within the same directory has to be different. Therefore you can provide configurations under the `overrides` key that will only apply to files that match specific glob patterns, using the same format you would pass on the command line (e.g., `app/**/*.test.js`).

```
module.exports = {
  root: true,
  env: {
    browser: true,
    node: true
  },
  extends: [
    // add more generic rulesets here, such as:
    // 'eslint:recommended',
    'plugin:vue/recommended',
  ],
  rules: {
    // override/add rules settings here, such as:
    // 'vue/no-unused-vars': 'error'
    'indent': ['error', 'tab'],
    'no-console': ['error', {'allow': ['debug', 'warn', 'error']}],
    'eqeqeq': 'error',
    'no-empty-function': 'error',
    'no-implicit-globals': 'error',
    'no-invalid-this': 'error',
    'no-undef': 'error',
    'no-unused-vars': 'warn',
    'no-use-before-define': 'warn',
    'no-sync': 'warn',
    'comma-dangle': 'off',
    'comma-spacing': 'warn',
    'comma-style': ['error', 'last'],
    'eol-last': 'warn',
    'no-extra-parens': 'warn',
    'no-extra-semi': 'warn',
    'no-irregular-whitespace': 'error',
    'block-spacing': 'warn',
    'brace-style': 'error',
    'jsx-quotes': ['error', 'prefer-single'],
    'no-inline-comments': 'warn',
    'keyword-spacing': 'warn',
    'key-spacing': 'warn',
    'no-mixed-spaces-and-tabs': 'warn',
    'no-multiple-empty-lines': 'error',
    'no-trailing-spaces': 'error',
    'no-whitespace-before-property': 'error',
    'object-curly-newline': 'warn',
    'object-curly-spacing': 'warn',
    'spaced-comment': 'warn',
```

```
    'space-in-parens': 'warn',
    'space-before-function-paren': 'warn',
    'space-before-blocks': 'warn',
    'no-var': 'error'
  },
  parserOptions: {parser: 'babel-eslint'}
}
```

Usage:

```
es lint filename.js
```

Editor Config

EditorConfig helps developers define and maintain consistent coding styles between different editors and IDEs. The EditorConfig project consists of a file format for defining coding styles and a collection of text editor plugins that enable editors to read the file format and adhere to defined styles. EditorConfig files are easily readable and they work nicely with version control systems.

Testing Methodologies

Unit Testing

We will use Jest framework for writing and running unit tests while developing the application. It can be an integral part of the project's development and continuous integration processes.

Code Coverage

- Jest can collect code coverage information from entire project, including untested files.
- Jest comes with functionality of generating report which help us in understanding test coverage's. this coverage include statement, functional, branch coverage's.
- Showing the coverage on files. Edit the file jest.config.js in the root of your project and drop this in there:

```
module.exports = {
  collectCoverage: true,
  collectCoverageFrom: ['src/**/*.{vue}'],
};
```

Code Coverage Report

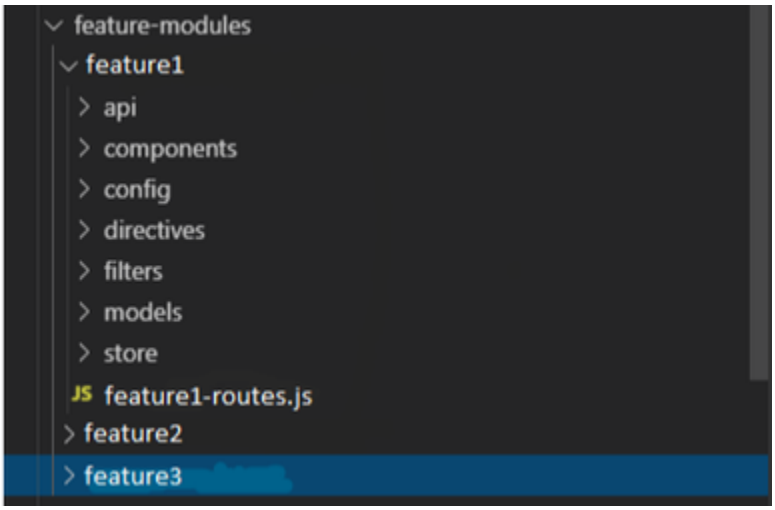
```
> vue-cli-service test:unit

 PASS  tests/unit/example.spec.js
  Counter.vue
    √ increments the counter value when button is clicked (27ms)
    √ reset the counter value when button is clicked (1ms)

-------------|----------|----------|----------|----------|-------------------|
File         |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
-------------|----------|----------|----------|----------|-------------------|
All files    |      100 |      100 |      100 |      100 |                   |
 Counter.vue |      100 |      100 |      100 |      100 |                   |
-------------|----------|----------|----------|----------|-------------------|
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        5.087s
Ran all test suites.
```

End to End Testing

We will use selenium framework along with C# to identify the x-path of element and NUnit will be used for End-to-end testing. It is a Software testing methodology to test an application flow from start to end. The purpose of End to end testing is to simulate the real user scenario and validate the system under test and its components for integration and data integrity.

Project Template

As per the architecture diagram folder-by-feature structure can be used into functional modules. Application folder structure is divided based on the features and that feature can be called as feature module. Root of the feature module can have components along with their test and a shared folder. Each feature module can also have its routes, state, and main component file.



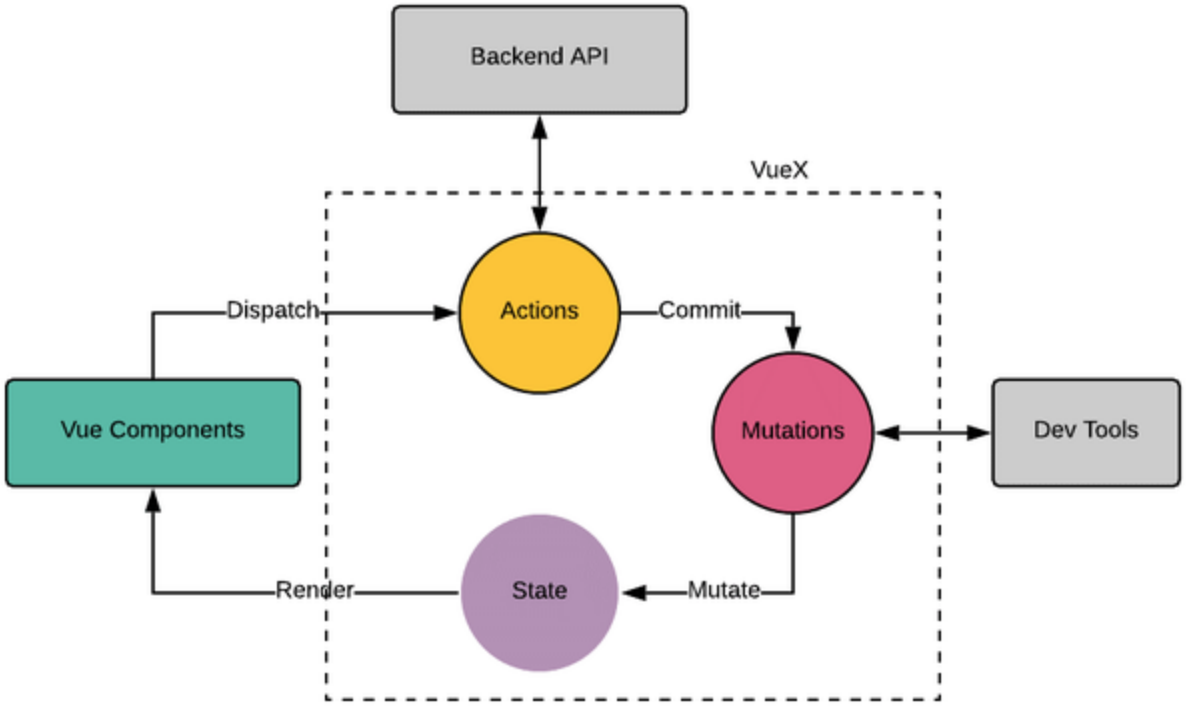
Config:
This folder will contain the configuration such as api related config, configuration constants etc.
Directives:
This folder will contain the directives. It will be used in the module level.
Filters:
This folder will contain the filters. It will be used in the module level.
API:
Services will contain the business logic or utilities or http calls that is independent of UI logic. Other terms used for Services are Utils, Helpers.
State:
Module level folder will contain the state that can be shared between different components.

State Management

Introduction

Vuex application is the store. A "store" is basically a container that holds your application state. Vuex stores are reactive. When Vue components retrieve state from it, they will reactively and efficiently update if the store's state changes. We cannot directly mutate the store's state. The only way to change a store's state is by explicitly committing mutations. This ensures every state change leaves a track-able record and enables tooling that helps us better understand our applications.

Dataflow



Usage

To maintain state between the module or components, We will use Vuex it provides state management technique so it can be used to session the data between the components or modules. each feature module has a store/state folder. This is where, we will keep all the state files related to that feature module. For eg. Feature1 module will have its state files in feature/store/state and suppose it will have states such as specific module list data, module details, etc.

Build

Build tools do two things:

1. Install things - install tools and package needed for the build
2. Do things - The goal of the “doing” tools is to automate all the menial and error prone tasks in web development. The things they do are sometimes called “tasks.”

Some of the tasks are:

1. Running my unit tests with a single command
2. Refreshing my browser when I save a file
3. Combining all my JavaScript files into one, and all my CSS files into one
4. Minifying my concatenated JavaScript and CSS files
5. Modifying the placement of <script> tags on an html page

Performance Strategies

Compilation

Ahead-of-Time

The Ahead-of-Time (AOT) compiler converts your HTML and Vue code into efficient JavaScript code during the build phase before the browser downloads and runs that code. Compiling your application during the build process provides a faster rendering in the browser.
vue-loader and vuetify supports Ahead-of-Time out of the box

Just-in-Time

A way of compilation during execution of a program – at runtime – rather than prior to execution. We can support it based on application need.

Lazy Loading

When building apps with a bundler, the JavaScript bundle can become quite large, and thus affect the page load time. It would be more efficient if we can split each route's components into a separate chunk, and only load them when the route is visited.

Monitoring

We will use NewRelic Browser for frontend monitoring, it supports following metrics:

1. Page Load Performance
2. Browser Performance
3. JavaScript Errors
4. Dashboard & Alerts
5. Security

More Info: https://newrelic.com/products/browser-monitoring/

Style Guide

1. VueJs
2. JavaScript
3. HTML/CSS

Integrations

Axios

Axios is a very popular JavaScript library you can use to perform HTTP requests, that works in both Browser and Node.js platforms. It supports all modern browsers, including support for IE8 and higher. It is promise-based, and this lets us write async/await code to perform XHR requests very easily.
Usage:
To handle http calls we will be using AXIOS for http communication with API. For that we will be writing wrapper class which can be injected into all the components wherever is needed.

Appendix

Any additional details