DIGIT UI: Frontend Architecture Design

Prerequisite reference study materials

Development and implementation team

- JavaScript ES6, ES7, ES8: Learn to Code on the Bleeding Edge (Full Course)
- Full React Course 2020 - Learn Fundamentals, Hooks, Context API, React Router, Custom Hooks

Development team

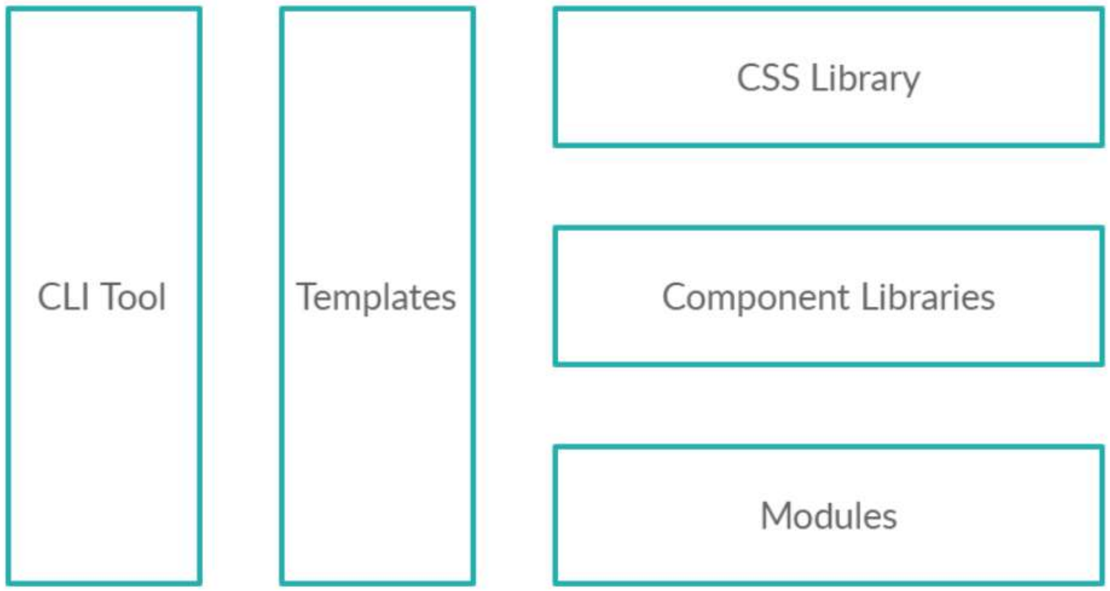
- Tailwind CSS Tutorial
- React Query Tutorial
- Multi-language Translate React JS APP with React Hook & i18next
- Axios Crash Course | HTTP Library
- Yarn Workspaces Tutorial
- React-hook-form Tutorial
- React Storybook Tutorial

Frontend Components

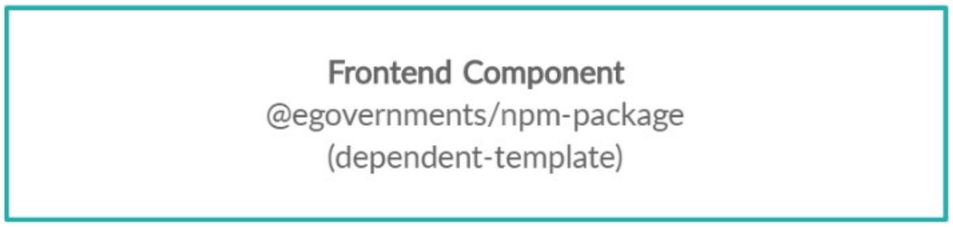
Broadly, the frontend components can be categorized as followings:

1. CLI Tool
2. Templates
3. CSS Library
4. Component Libraries
5. UI Modules

The CSS Library, Component Libraries, and UI modules are based on templates created by CLI Tool.



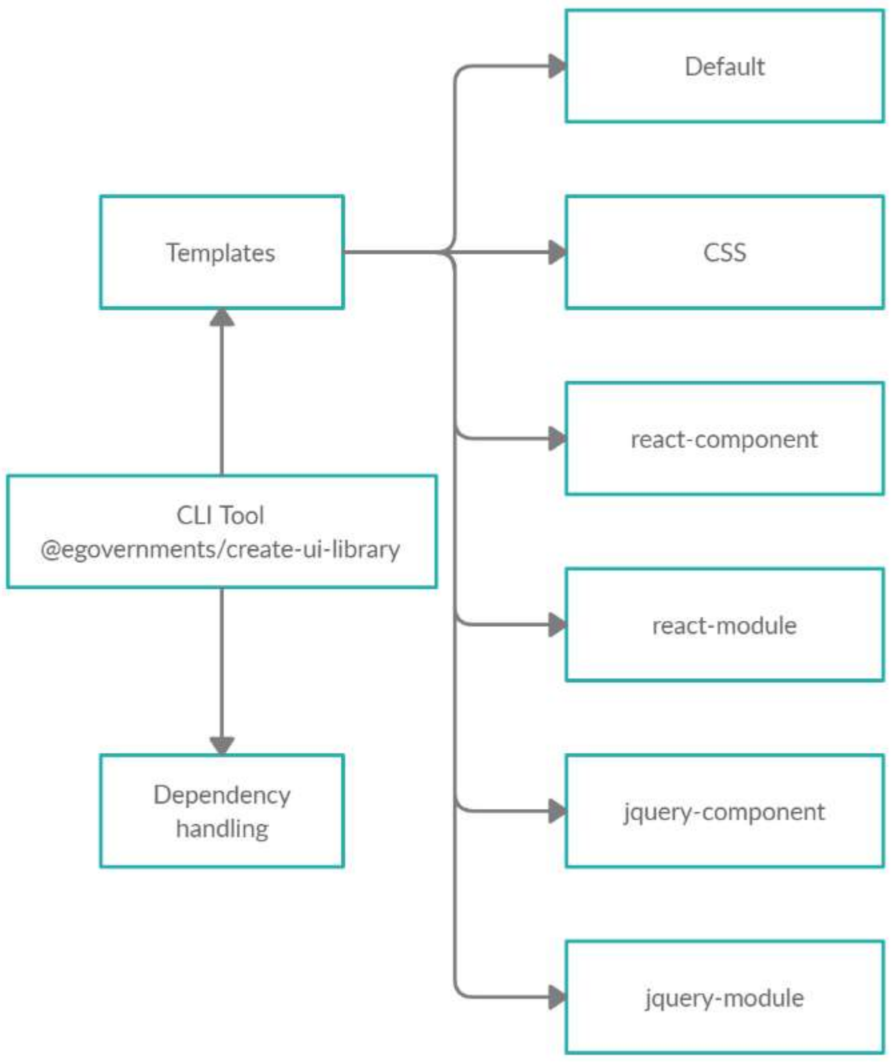
Nomenclature



The first line contains the Architecture Component name or info, the second line has npm-package and in the bracket, we have a template based on which the component will be created.

CLI Tool

This tool will be used to bootstrap libraries and micro frontend frameworks for the DIGIT platform. It contains a “template” folder, which will have several starter apps.



The following will be in the template:

- Default (WIP) - A default pure js template.
- React-component (Done) - Starter template to make the component library based on React.
- Module (React based) - Micro frontend framework, which will act as a base for different modules we will create.
- … more to come

Features

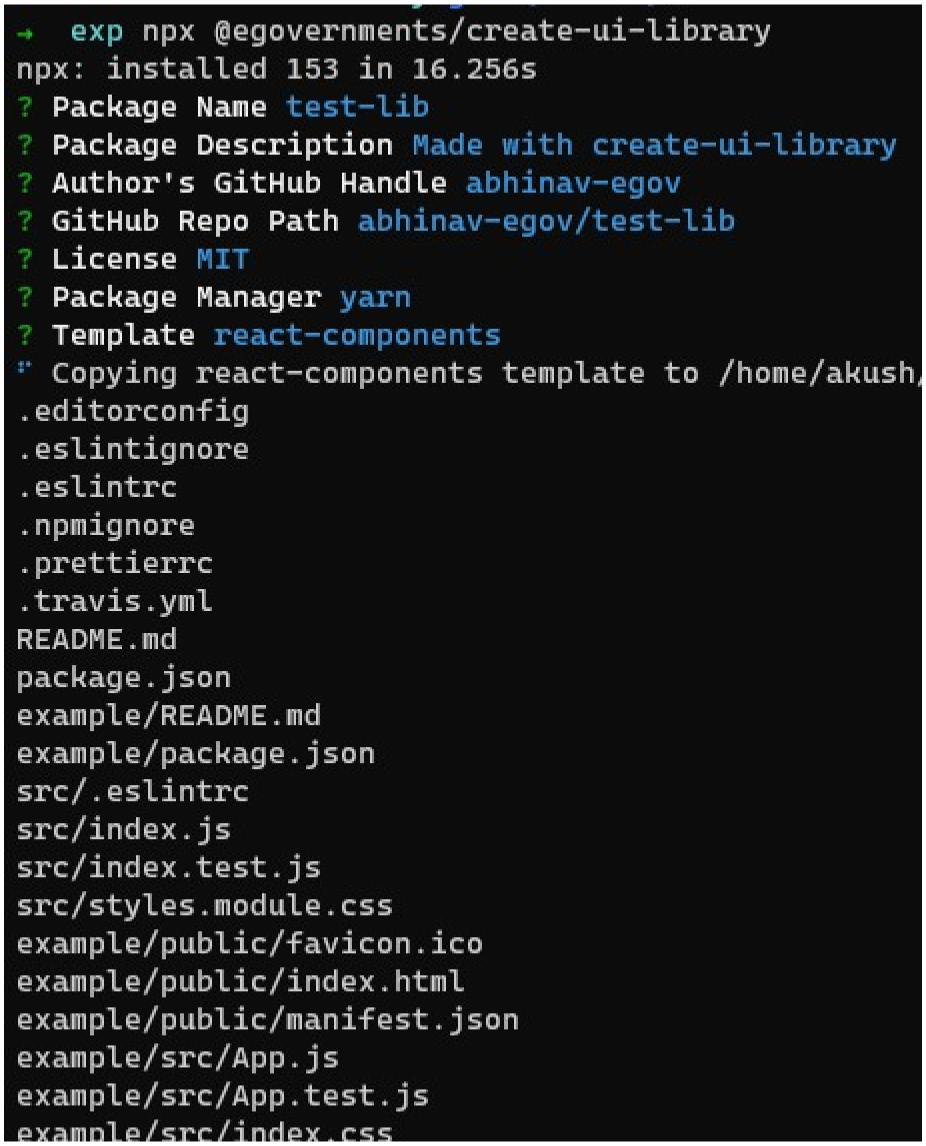
- Easy-to-use CLI
- Handles all modern JS features
- Bundles commonjs and es module formats
- create-react-app for example usage and local dev for React-based libraries
- Rollup for bundling

- Babel for transpiling
- Supports complicated peer-dependencies
- Supports CSS modules

Example

Answer some basic prompts about your module, and then the CLI will perform the following steps:

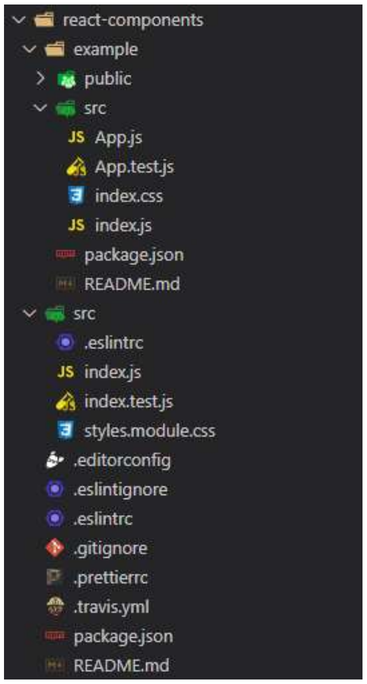
- copy over the template
- install dependencies via yarn or npm
- link packages together for local development
- initialize local git repo



```
→  exp npx @egovernments/create-ui-library
npx: installed 153 in 16.256s
? Package Name test-lib
? Package Description Made with create-ui-library
? Author's GitHub Handle abhinav-egov
? GitHub Repo Path abhinav-egov/test-lib
? License MIT
? Package Manager yarn
? Template react-components
⠋ Copying react-components template to /home/akush/
.editorconfig
.eslintignore
.eslintrc
.npmignore
.prettierrc
.travis.yml
README.md
package.json
example/README.md
example/package.json
src/.eslintrc
src/index.js
src/index.test.js
src/styles.module.css
example/public/favicon.ico
example/public/index.html
example/public/manifest.json
example/src/App.js
example/src/App.test.js
example/src/index.css
```

Templates

The templates have the following folder structure:



```
react-components
  example
    public
    src
      App.js
      App.test.js
      index.css
      index.js
    package.json
    README.md
  src
    .eslintrc
    index.js
    index.test.js
    styles.module.css
  .editorconfig
  .eslintignore
  .eslintrc
  .gitignore
  .prettierrc
  .travis.yml
  package.json
  README.md
```

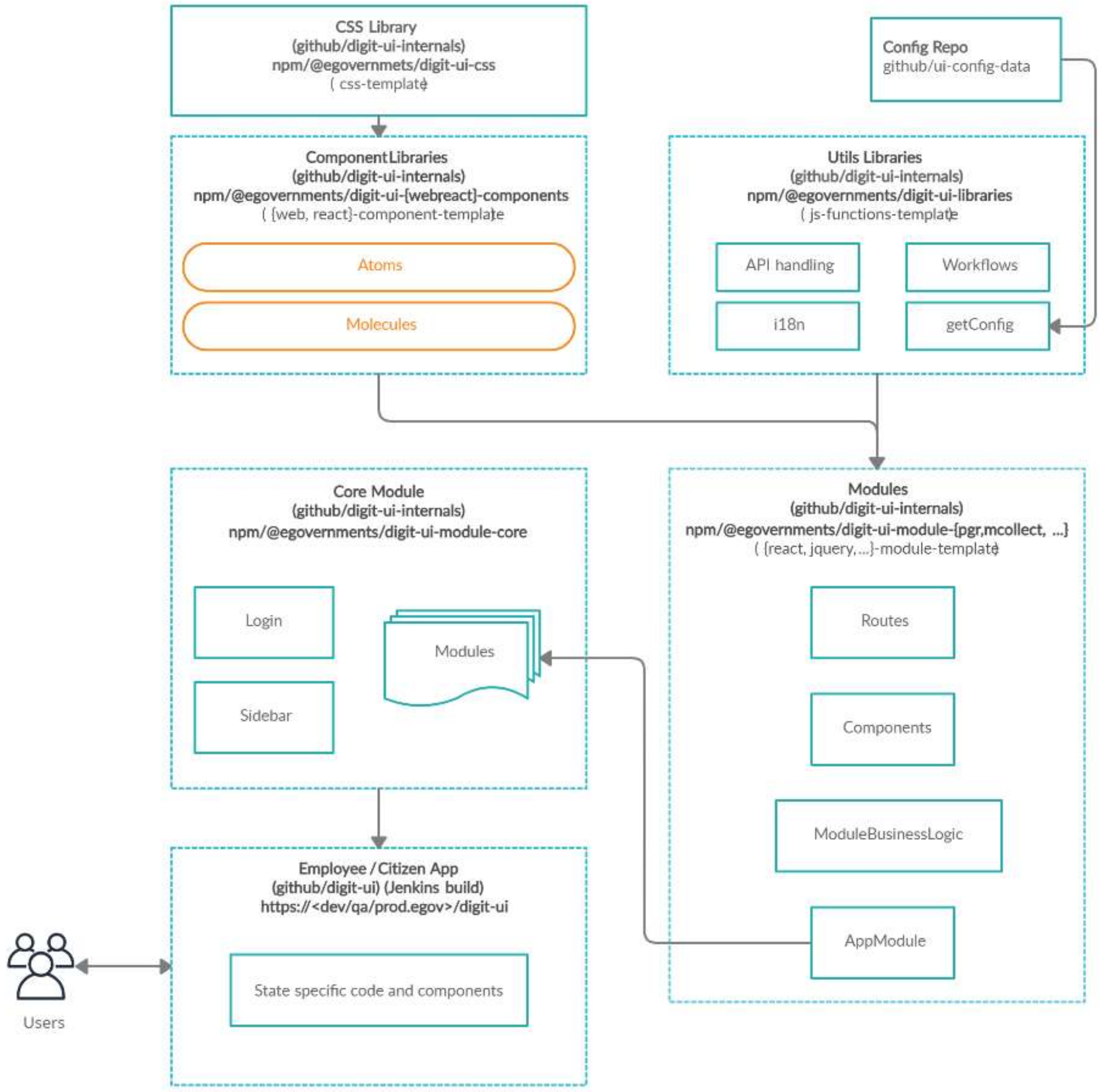
The components related to the template are inside the src folder of the template and an example is created to use and showcase the app created by the template.

Architecture

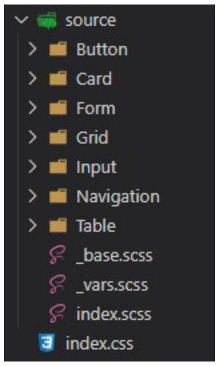
We have two repos:

- digit-ui-internals - https://github.com/egovernments/digit-ui-internals
  - Meant for eGov development team to build components and default modules.
  - Contains following modules:
    - CSS Library
    - UI Components (presently react-components)
    - Utils Library: Contains Services, Localization handling and React Hooks.
    - UI Modules
      - Core - containing login, routing and global state.
      - PGR
      - FSM
      - PT
      - Payment
- digit-ui - https://github.com/egovernments/digit-ui
  - Meant for state team to manage, make changes and deploy
  - Import digit-ui-internals modules.
  - Customizations
    - View
    - Services

- Build and deploy scripts
    - Dockerfile & nginx.conf
    - build-config.yaml


CSS Library
(github/digit-ui-internals)
npm/@egovernmets/digit-ui-css
(css-template)
Config Repo
github/ui-config-data
ComponentLibraries
(github/digit-ui-internals)
npm/@egovernments/digit-ui-{web,react}-components
({web, react}-component-template
Atoms
Molecules
Utils Libraries
(github/digit-ui-internals)
npm/@egovernments/digit-ui-libraries
(js-functions-template
API handling
Workflows
i18n
getConfig
Core Module
(github/digit-ui-internals)
npm/@egovernments/digit-ui-module-core
Login
Sidebar
Modules
Modules
(github/digit-ui-internals)
npm/@egovernments/digit-ui-module-{pgr,mcollect, ...}
({react, jquery,...}-module-template)
Routes
Components
ModuleBusinessLogic
AppModule
Employee / Citizen App
(github/digit-ui) (Jenkins build)
https://<dev/qa/prod.egov>/digit-ui
State specific code and components
Users

CSS Library



```
source
  Button
  Card
  Form
  Grid
  Input
  Navigation
  Table
  _base.scss
  _vars.scss
  index.scss
index.css
```

The CSS Library will have all the classes both in the module and compiled form.

Can be imported like

```
import "@egovernments/digit-ui.css/Button"
```

or like this for full CSS import

```
import "@egovernments/digit-ui.css"
```

Component Libraries

Component Library will have the set of all the required components defined in them.

```
# Atoms
<egov-button text action />

<egov-checkbox name value />

<egov-textarea placeholder />

# Molecules
<egov-list data>
  <egov-list-item />
</egov-list>

<egov-searchbox placeholder action />

<egov-list-filter>
  <egov-input />
  <egov-list data />
</egov-list-filter>
```

Utils Library

This will have the followings:

- Workflows
- API handling - API caching and handling strategies will be here, imported, and shared by all modules. Published as a function, can be used by anyone.

```
import { fetchAPI } from '@egovernments/digit-utils'

async function searchProperty() {
  const { data, error } = await fetchAPI("/api/searchProperty");
  if (error) return "";
  return data;
}
```

- Internationalization (i18n)

```
<egov-textarea placeholder="i18n.PGR_NEW_COMMENTS" />
```

- getConfig
  - This will be like MDMS config service
  - The default and state configs will be stored on GitHub.
  - getConfig will fetch the latest config on runtime and initiate the module.

default config:

```
{
  "name": "create_complaint",
  "action": "createComplaint",
  "init": "newComplaintInit",
  "saveUrl": "/rainmaker-pgr/v1/requests/_create",
  "redirectionRoute":"/complaint-submitted",
  "fields": [{
    "id": "city",
    "name": "city",
    "type": "dropdown",
    "labelText": "{{i18n.CORE_COMMON_CITY}}",
    "hintText": "{{i18n.CS_CREATECOMPLAINT_SELECT_PLACEHOLDER}}",
    "errMessage": "{{i18n.CS_ADDCOMPLAINT_COMPLAINT_TYPE_PLACEHOLDER}}",
    "required": true,
    "action": "createComplaint.updateDependentFields"
  }, {
    "id": "address",
    "name": "address",
    "type": "input-textarea",
    "labelText": "{{i18n.CS_ADDCOMPLAINT_LOCATION}}",
    "hintText": "{{i18n.CS_COMPLAINT_DETAILS_LOCATION}}",
    "errMessage": "",
    "action": ""
  }, {
    "id": "media",
    "name": "media",
    "type": "input-file",
    "errMessage": "{{i18n.CS_FILE_UPLOAD_FAILED}}",
    "action": ""
  }],
  "buttons": [{
    "id": "addComplaint-submit-complaint",
    "name": "submit",
    "labelText":
"{{i18n.CS_ADDCOMPLAINT_ADDITIONAL_DETAILS_SUBMIT_COMPLAINT}}",
    "action": "createComplaint.submit"
  }]
}
```

state config:

```
{
  "name": "create_complaint",
  "redirectionRoute":"__delete__",
  "saveUrl": "__delete__",
  "fields": [{
    "__action__": "insert_after",
    "__property__": "address",
    "id": "landmark",
    "name": "landmark",
    "type": "input-text",
    "labelText": "{{i18n.CS_ADDCOMPLAINT_LANDMARK}}",
    "hintText": "{{i18n.CS_ADDCOMPLAINT_LANDMARK_PLACEHOLDER}}",
    "errMessage": "{{i18n.PT_LANDMARK_ERROR_MESSAGE}}",
    "action": "",
  }, {
    "__action__": "update",
    "id": "address",
    "labelText": "{{i18n.CS_ADDCOMPLAINT_LOCATION}}",
    "hintText": "{{i18n.CS_COMPLAINT_DETAILS_LOCATION}}",
  }]
}
```

getConfig:

```
# @egovernments/digit-utils
```

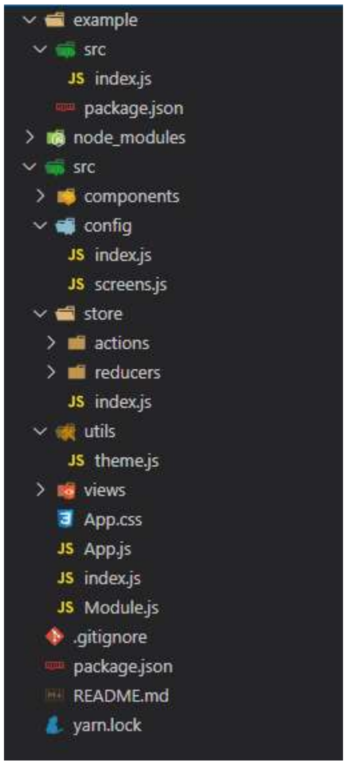
```
exports.getConfig = async (state, module = "", screen = "") => {
 let path = state;
 if (module) {
  path = `${path}.${module}`;
  if (screen) {
   path = `${path}.${screen}`
  }
 }
 const { data, error } = await fetchAPI(`/api/_get?config=${path}`)
 if (error) return {};
 return data;
}

# Module
import { getConfig } from '@egovernments/digit-utils'

const compaintConfig = getConfig("pb", "pgr", "create_complaint");
```

Modules

The module will be a black box for the states, they will only access throw `node_modules` or `CDN`.

Any state-specific components can be passed during the initialization of the module inside the state's employee or citizen app.

The modules structure will look like:



```
example
  src
    index.js
  package.json
node_modules
src
  components
  config
    index.js
    screens.js
  store
    actions
    reducers
    index.js
  utils
    theme.js
  views
  App.css
  App.js
  index.js
  Module.js
.gitignore
package.json
README.md
yarn.lock
```

The module code is in src and an example is created to show how it will be imported in the core app and can be modified at the state level.

The components passed to the module from the core app are:

- **components** (optional): Any custom component that is created outside of module scope.
- **onRouteChange**: This is a callback function that will be called by the module itself to alert the core app about the route changes so the sidebar handler can change the active item.
- **theme** (optional): This is a set of CSS variables which states can modify to change the theme of the module.

Eg, PGRModule:

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-rdux';
import { getConfig } from '@egovernments/digit-utils';

import configureStore from './store';
import applyTheme from "./utils/theme";

import PGRApp from './App';
```

```
const PGRModule = ({ config, components, onRouteChange, theme }) => {
  applyTheme(theme);
  return <Provider store={configureStore(config)}>
    <PGRApp components={components} onRouteChange={onRouteChange} />
  </Provider>
}

function initPGR = ({ state, element, components, onRouteChange, theme = {} }) => {
  getConfig(state, "pgr").then(config => {
    ReactDOM.render(
      <PGRModule config={config}
        components={PGRComponents}
        onRouteChange={onRouteChange} theme={theme} />,
      document.getElementById(element));
  })
}

export default initPGR
```

Modules will have the followings inbuilt

- Theme - this may change if we later decide to use any css-in-js library, like styled-components.
- Components
- Routes
- State management
- Business logic
- API integrations

Employee / Citizen App

The app will import the developed module.

```
import './index.css'

import { initPGR } from '@egovernments/pgr-module';
import punjabLogo from './assets/logo.png'

const theme = {
  "--primary-color": "#3f51b5",
  "--text-color": "#212121"
}
```

```
const PGRComponents = {
 "logo": punjabLogo
}

const initPunjabPGR = (onRouteChange) =>
 initPGR({
   state: "pb",
   element: "#appWrapper",
   components: PGRComponents,
   onRouteChange, theme
 });

export default initPunjabPGR;
```

Another method will be importing from CDN:

```
import './index.css'
import punjabLogo from './assets/logo.png'

const theme = {
 "--primary-color": "#3f51b5",
 "--text-color": "#212121"
}

const PGRComponents = {
 "logo": punjabLogo
}

const initPunjabPGR = (onRouteChange) => {
 import("https://unpkg.com/@egovernments/pgr-module").then(({ initPGR }) => {
   initPGR({
     state: "pb",
     element: "#appWrapper",
     components: PGRComponents,
     onRouteChange, theme
   })
 })
}
export default initPunjabPGR;
```

At the next phase, the Employee and Citizen app can be rewritten to be a single app with role and permissions based rendering.

Item	Covered	Comments
Standardization		

- UX	YES	@egovernments/digit-ui.css will provide the standard components css
- UI Controls	YES	button and input types will be defined and exposed to all
- Flow Design	YES	part of components library
- CSS	YES	@egovernments/digit-ui.css
- Project Structure	YES	Project structure is fixed by egov cli tool
- Caching	YES	API handling will be in egov utils library
Modular Application - Ability to deploy each module independently	YES	All modules are published independently
Tech Agnostic	YES	
Extensibility	YES	CSS is extended by published css files, config is published on Github like MDMS
Application Prototyping	YES	egov cli tool takes care of prototyping
Upgradibility	YES	module upgrades are independent of state deployments and is now runtime fetch