DIGIT UI Manual

About This Manual

This manual covers the various features of DIGIT UI and every feature is defined with a flow and consists of screenshots for user assistance.

Intended Audience

This manual is intended to assist employees who will develop DIGIT UI

Document Conventions

If you see	It means
Bold Text	Screen elements like buttons, drop-down lists, input fields, and such UI elements are highlighted in Bold
Italicized text	*The word or set of words is especially emphasized.*
>	The Arrow '>' notation describes the flow of navigation in The Arrow '>' notation describes the flow of navigation in the application
Note:	The text following this icon provides additional information
Tip:	The text following this icon provides suggestive use
Mandatory:	The field following this icon is a Mandatory field
Attention:	The text following this icon indicates a call for attention

Document Version History

Version Number	Release Date	Release Description
Version 1.0		
Version 1.1		This release contains followings: FSM Module Common modules Citizen Login Payment module

Introduction to DIGIT UI

DIGIT UI contains

- UI Components
- API Services
- Config Service
- Localization service

Resolution of Key Challenges

A bundled UI

Currently, all our Apps get bundled into an employee and citizen app. This makes it difficult to send individual/selected apps to a particular state. Also, it makes it difficult for developers during the development process. They sometimes need to resolve conflicts with other apps code changes.

Resolution: microsite architecture will be adopted, similar to microservice architecture in the backend. Each application can be developed independently and stitched together for a single deployment or independently deployed. The specific strategy of deployment will be adopted based on studying the current distribution channel.

Update: We are working on microsite architecture. Although the structure is in place Independent libraries are currently used as components.

Application prototyping

Prototyping a new application either based on an existing UI flow or a new flow is not easy or straightforward. It takes some effort.

Resolution: starter kits will be created to enable adding new applications easily.

Update: We have created an E-Gov CLI service for prototyping new applications based on the current structure.

Learning Curve

The current framework has a steeper learning curve, which makes it enabling new resources is a time-consuming effort.

Resolution: A balance between framework features and resource allocation has to be calculated. However, we will provide cross-framework support.

Update

Technology dependence

Current react is developed on React and can integrate only react apps into it. This makes it difficult for state teams to get involved as many of them lack React resources and want to use the UI technology that they are comfortable with. A plain bootstrap + Jquery or similar else is what the state teams may be comfortable with.

Resolution: We will support all js frameworks. It can be React, Preact, Angular, jQuery, Vue, etc. All of them are JS frameworks and can be clubbed together. Simple tutorials can be provided to external developers to get them easily started with their choice of framework.

Update: We are making all the UI components and API utilities & services independent of each other so that each component can be reused with other frameworks as well.

of API calls

Currently, the UI gets created by merging multiple reusable components. Each of these components may need some master data or some API call to happen. When these components are combined to create a flow, the number of API calls increase. This puts additional load on the server, also it makes the application slower on 3G connections or low-end mobile devices.

Resolution: We will create a topological structure of components, enabling the maximum number of parallel API calls and minimizing the load time. To optimize and reduce the number of calls, we will work with the backend team. We can look at implementing GraphQL based API's to eliminate multiple API calls. The components only indicate the resource and the fields they require and the main component does the API calls and distributes data.

Update: API Services and Utilities achieve what was stated as a resolution to this. API integration is in progress and the UI team has taken into consideration that API calls have to be minimized

State-specific changes

Currently, we allow Javascript hooks at very specific places that allow states to add javascript on top of Digit and customize the UI to add or remove a field. This is currently a very basic form. The javascript hooks are not pretty straightforward while adding new elements

Resolution: We will add override support on default config for better integration of state-specific changes.

Update: Override Support has been created on default config, state-level hooks are in progress

The following configs are available:

- PGR: PGR: UI Implementation - Guidelines & FAQs
- FSM: FSM: UI Implementation - Guidelines & FAQs

Minimal/Non-standard configurations

Currently, UI configuration is not defined well and is available for very limited features. These configurations are used to control things like

- State Logo
- Which states to use for login

There should be a standard configuration approach, which can be used to enable/disable/configure features within the UI.

Resolution: Configuration support will be added for details as such, state logo, and other state-related parameters.

Update: State-level hooks are defined.

SEO

No SEO friendliness as there is no Server-Side rendering or google analytics

Resolution: Pre or Server rendered page support will be added to enable SEO. GA integration can be done easily.

Update: Will be implemented where required.

Dev Setup

The development environment can be set up by using git clone (To be updated with eGov Repo link on setup) & yarn.

After cloning the repository and changing drive into the repository, run *yarn install*, this will install the dependencies of the repository

Then run *yarn start* to run the application on the local machine.

```
1$ git clone git@github.com:egovernments/digit-ui.git
2$ cd digit-ui/web
3$ yarn install
4$ yarn start
```

Getting Started

This section will talk about the following aspects.

1. Work
2. File Structure
3. UI Components

Work

- [x] Config service to merge Default config with Delta config
- [x] JSON Config based UI renderer
- [x] i18n library - Localization & Translation mapping library
- [x] UI Component library as per new design system
- [x] CSS Component utilizing TailwindCSS Framework
- [x] API Services & Integration
- [x] Folder and file structure revamp as per new repo creation
- [x] Validations in PGR Module
- [x] Development environment setup
- [x] State-level hooks
- [ ] Citizen login UI is complete but for employee we are currently using temporary fix for Authorization of user by hardcoding auth-token
- [x] @egovernments/digit-ui-commons add sidebar and top navbar
- [x] Add citizen and employee routes
- [ ] Optimizations
- [ ] MDMS Service
- [x] Versioning
- [x] Documentation for Training

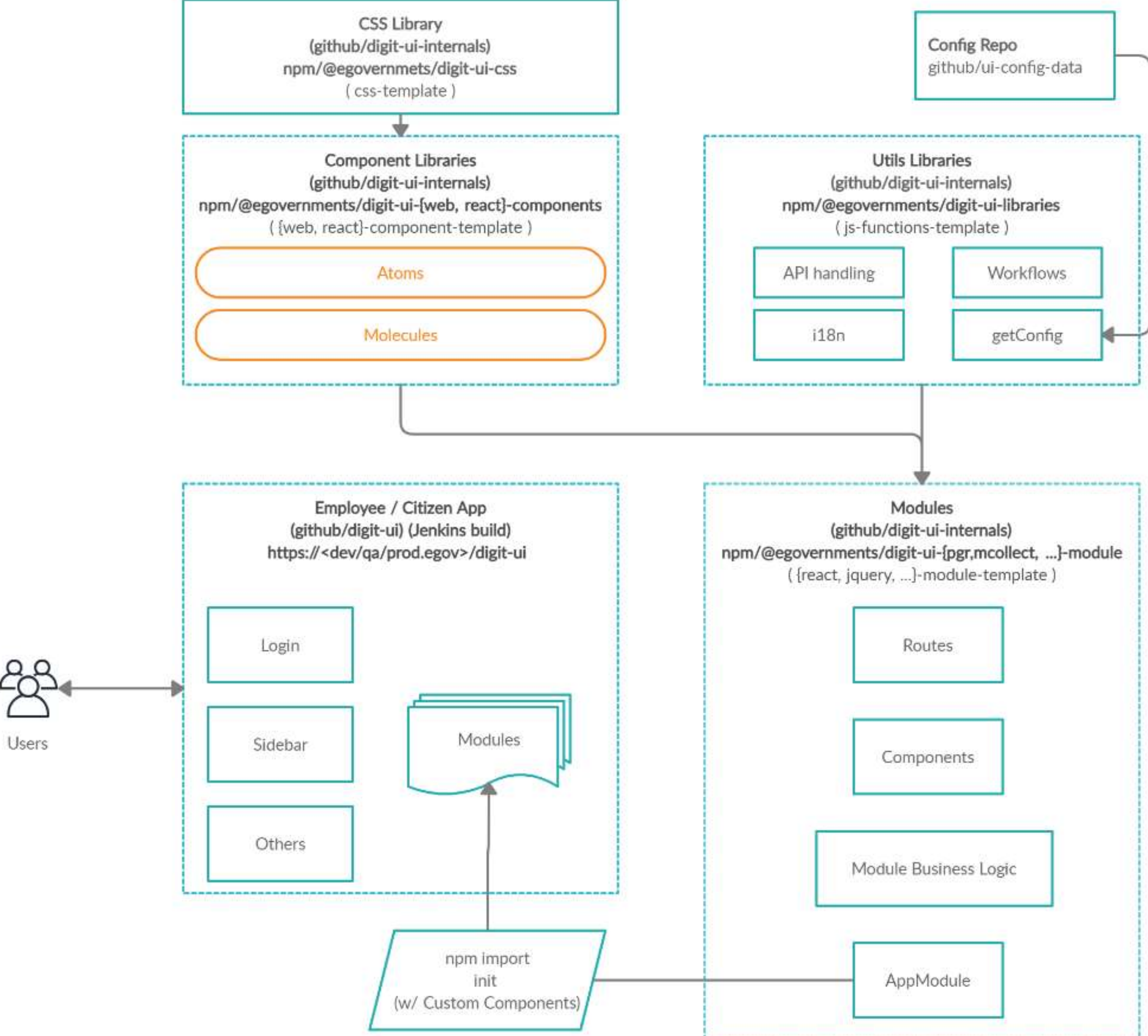
Details

Composition

```
1// Internal Components - only egov employee will have access
2- Github/digit-ui-internals
3  - @egovernments/digit-ui-css        // maybe renamed to something better
4  - @egovernments/digit-ui-components
5  - @egovernments/digit-ui-utils
6  - modules
7    - @egovernments/digit-ui-commons    // contains payment module
8    - @egovernments/digit-ui-core       // contains login, sidebar and other common modules
9    - @egovernments/digit-ui-pgr
10     - @egovernments/digit-ui-fsm
11
12// External - states will have edit access
13- Github/digit-ui                    // this will use all internal components to create employee and citizen renderings
14  - index.html
15
```

Deployment

The modules and packages in digit-ui-internals will be published over npm. The digit-ui will serve as the core app, which will use the modules via npm and will be deployed on state/eGov servers via Jenkins build.



CSS and Image assets Module

The CSS is maintained at Github/egov-ui-internals/digit-ui-css and published at npmjs/@egovernments/digit-ui-css

To use the CSS, we will add the following in the core app.

```
1<link href="https://unpkg.com/@egovernments/digit-ui-css@1.0.4/dist/index.min.css" rel="stylesheet">
```

This module also contains images that can be browsed at https://unpkg.com/browse/@egovernments/digit-ui.css/

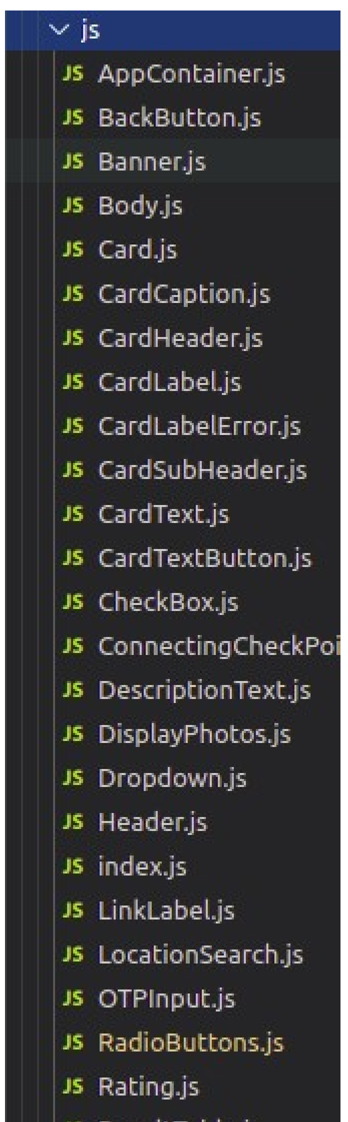
Components

Components will be maintained at github/egov-ui-internals/egov-ui-components and published at npmjs/@egovernments/egov-ui-components

The usage will be as follows

```
1import { Card, Dropdown } from "@egovernments/egov-ui-components"
2
3const Step = () => {
4  return (
5    <Card>
6      <Dropdown label="City" isMandatory={true} option={dropdownOptions} />
7    </Card>
8  )
9}
```

The directory consists of the following components



Libraries: Utils and Services

We have the business logic layer and utility library in a separate repo, to be maintained and shared across all modules. It will be maintained at GitHub/egov-ui-internals/egov-ui-utils and hosted over npm at @egovernments/egov-ui-utils
It will be used as follows

```
1// Module
2import React from 'react'
3
4const App = ({ stateCode, cityCode, moduleCode, role }) => {
5  const store = eGov.Services.useStore({ stateCode, cityCode, moduleCode, role })
6  return <p>Module Example </p>
7}
8export default App
```

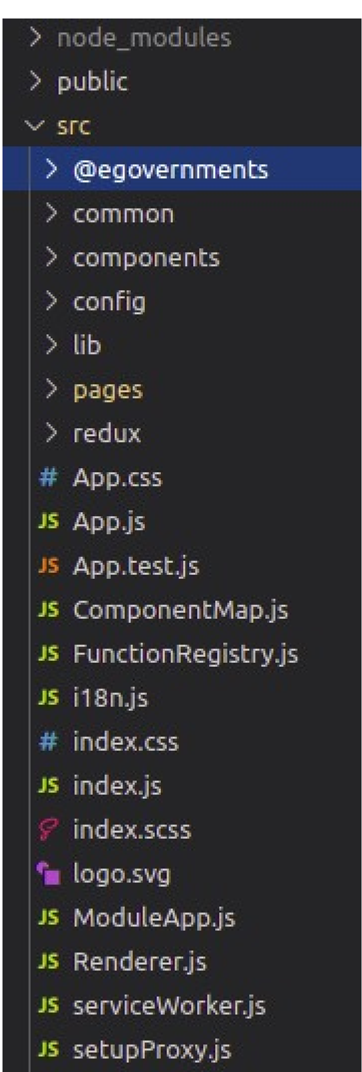
```
9
10// Core App
11import ReactDOM from 'react-dom'
12import initLibraries from '@egovernments/egov-ui-utils'
13
14import PGRApp from "@egovernments/egov-ui-pgr"
15
16initLibraries();
17ReactDOM.render(<PGRApp deltaConfig stateCode cityCode moduleCode role={"citizen"} />,
document.getElementById("root"))
```

This contains the following

- Translations
- Config handling
    - Configs will be maintained at GitHub/egov-ui-config
    - On init, module will fetch its configs from https://raw.githubusercontent.com/abhinav-egov/egov-ui-config/<state>/<module>/<userType>/<page>.json
    - It will fetch all pages configs.
    - The default config will be maintained at the main branch and states' on their respective branches. i.e., pb for Punjab
- Services
    - Localization
        - Localization is fetched only once per locale per module and stored.
    - Store
        - getStore: this gets state module config, MDMS data, and localization data and returns the init store for redux.
            - at init, the following locales are fetched
                - rainmaker-common
                - rainmaker-{module} (i.e., rainmaker-pgr)
                - rainmaker-{state} (i.e., rainmaker-pb)
                - rainmaker-{tenantId} (i.e., rainmaker-pb.amritsar)
- Location
- MDMS
- Search

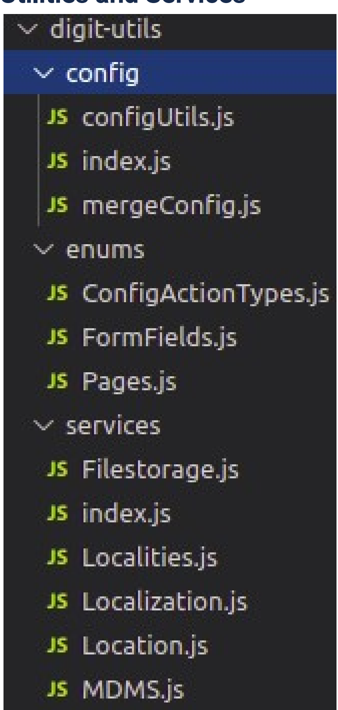
The following is in progress.

File Structure



The application structure follows the conventional react app file structure. The above Figure represents the basic structure of the application. Currently, the @egoverments Modules reside inside the src / “source” directory, although it will be a collection of separate node modules.

Utilities and Services



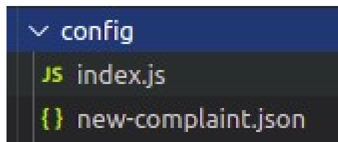
The above figure represents the file structure of different utilities and services we are using inside the application.

“config” directory encapsulates utilities and functions used for integration of dynamic config

“enums” directory consists of Enums used for integration of same

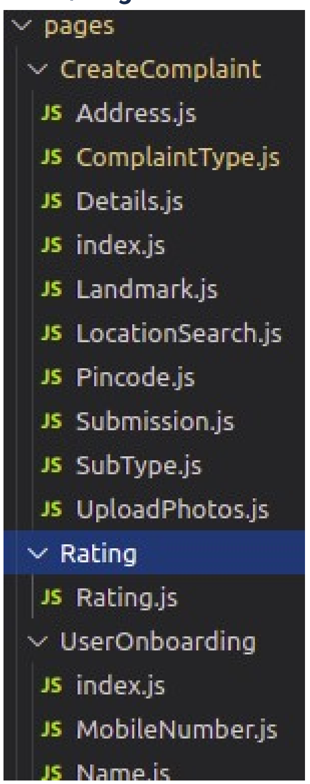
“services” directory consists of various API calls, Local Storage and Session Storage service made available to the application.

Configuration / Dynamic Config



The above figure represents the file structure of different config files used for the implementation of dynamic config

Views/ Pages



“pages” directory encapsulates all the Views inside the application, sub-directories of the same are grouped sub-views or views inside a specific flow.

Example: CreateComplaint subdirectory consists of all the views that reside with the “Create Complaint” workflow of the application.

Some common views may reside outside the scope of any specific subdirectory are usable by multiple workflows

Redux

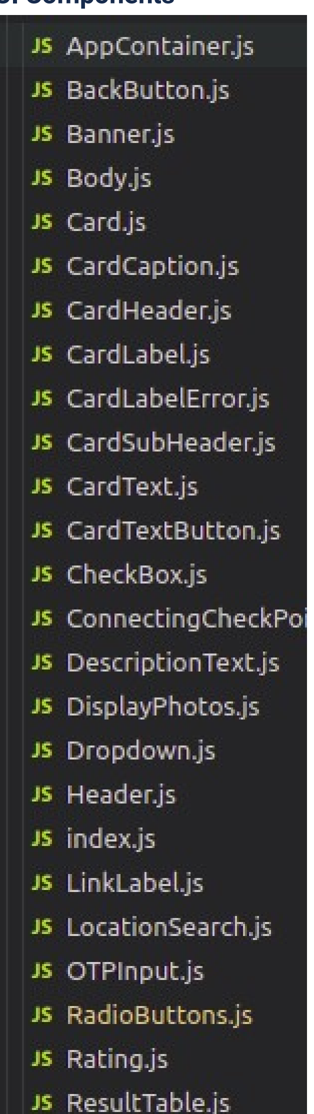
Since we are using the redux library for state management, the above figure represents the complete structure of the same.

“actions” directory contains the Action used inside the application

“reducers” directory contains the Reducers

Some additional directories are for using utility & service that is routed through redux for state management

UI Components



The figure shows UI components are used in the application.

The hierarchy of components is as follows:

Parent Component	Child Components
AppContainer	*Body*

Body	*Card, Header, LinkLabel, TopBar*
Card	*Banner, CardCaption, CardHeader, CardLabel, CardLabelError, CardSubHeader, CardText, CardTextButton, CheckBox, ConnectingCheckPoints, DescriptionText, DisplayPhotos, Dropdown, Header, LinkLabel, LocationSearch, OTPInput, RadioButtons, Rating, ResultTable, StatusTable, SubmitBar, TextArea, TextInput, UploadImages*