cityfinance - an initiative by janaagraha

Technical Design Document

Issue 1

TABLE OF CONTENTS

0 PREFACE

0.1 PURPOSE OF THIS DOCUMENT

#1 *This document is a generic Technical Design Document document for Cityfinance project. It provides guidance and template material which is intended to assist the technical team to define a project specific Technical Design Document document.*

0.2 USE OF THIS DOCUMENT

#1 *This Preface is addressed to the users of this generic document and is not meant to be retained in any project specific Technical Design Document documents based on it.*

#2 *The remaining sections (numbered 1, 2, 3,…) constitute a template that should be used to construct the project-specific Technical Design Document document.*

a. "Summary" Properties

Title	Type of document (i.e. Technical Design Document)
Author	Vivek Sharma
Keywords	Cityfinance
Version	Issue number (currently Issue 1)
Date	09-10-2019

0.3 OVERVIEW

TO BE DEFINED

0.4 TECHNICAL STACK

Technical stack of the application has been defined based on the best interest of the project i.e. scalable, maintainable, reusable code base, easy to develop etc. with cutting edge technologies. Below are the technologies being used to develop the application :

Particular	***Technology***
Frontend	*Angular 6, Bootstrap, SCSS*
Backend	*NodeJS, ExpressJS*
Database	*MongoDB*
Operating System(Current deployment)	*Linux(Ubuntu)*
Infrastructure used	*AWS(EC2, S3 bucket)*
IDE used	*VS Code*

SYSTEM OVERVIEW

Cityfinance.in is an initiative of Janaagraha Centre for Citizenship and Democracy (JCCD). The primary objective of this portal is to create a national database of city finances and design framework for analysing the financial and operational performance of the cities.

21.1 SYSTEM CHARACTERISTICS

The description of the system should be given in terms of the Architecture of the solution that is being implemented with high level data flows described to set the context of the system, i.e. to look at its external interfaces. This section should also set out to characterise the system describing aspects of its operation that indicate if the system has, inter alia:

- *the nature of the interface to the users of the system*
- *to be highly resilient or fault tolerant*
- *to provide security features to protect data*
- *to be scaleable and easily maintainable in the future*
- *to have any special back-up facilities to protect important data.*

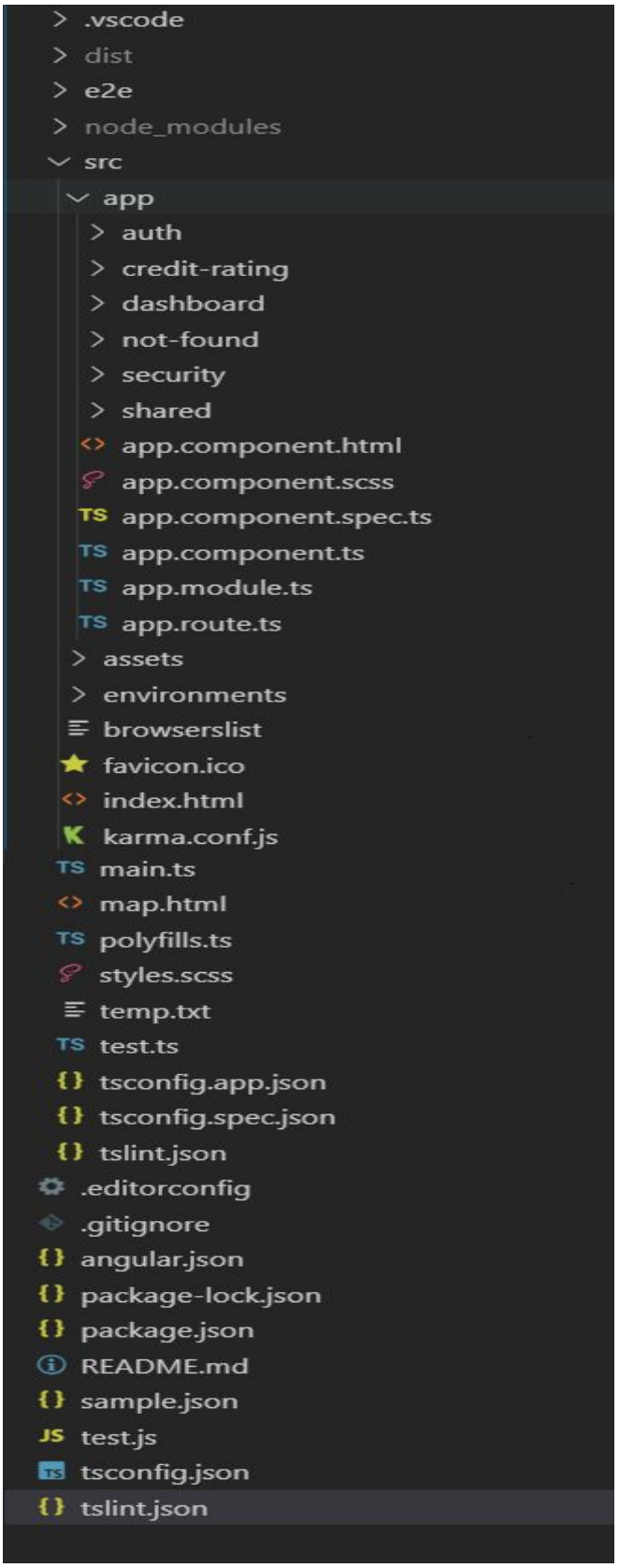
21.2 PROGRAMMING STANDARDS

#1 *This section should define the project programming standards. Whatever languages or standards are chosen, the aim should be to create a convenient and easily usable method for writing good-quality software. If an application development tool is used there may be other conventions that need to be defined.*

#2 *When programming in any new language, a standard for its use should be written to provide guidance for programmers. This standard may be referenced or included here.*

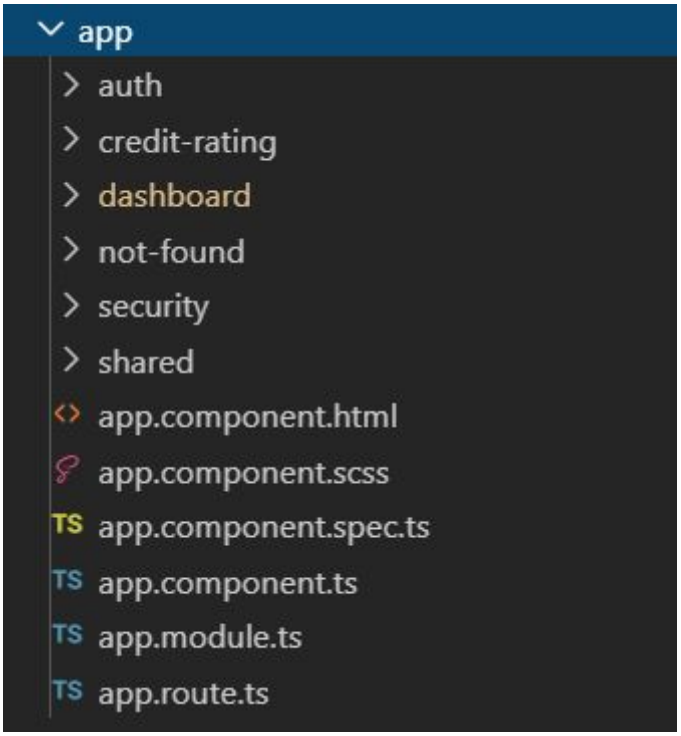
#3 *Where there are external interfaces, the programming standards for the interfaces required should be referenced.*

2. UI APPLICATION STRUCTURE:

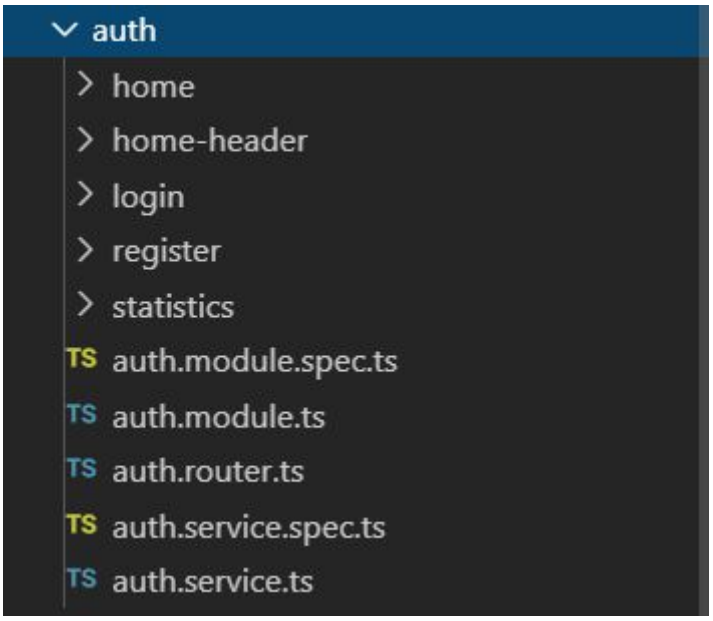


In order to create the application reusable and scalable, application has been structured into multiple modules. Each module is responsible to perform specific task and brief description has been provided below:

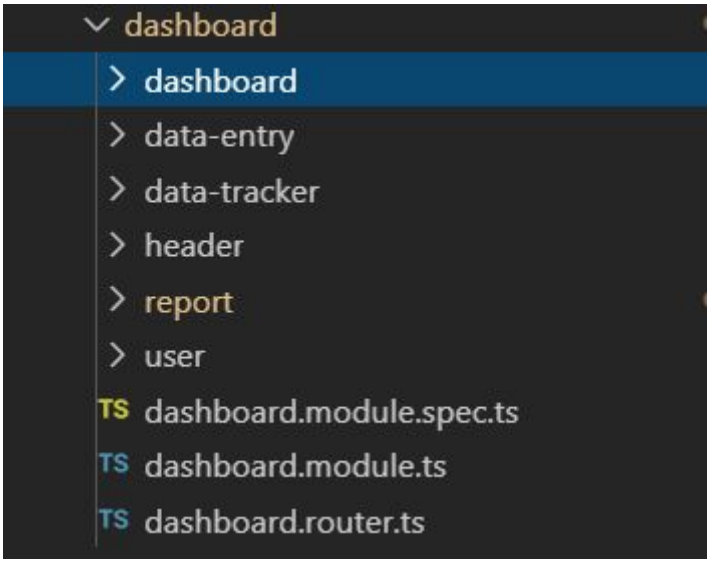
1. ***App Module** : to handle overall application flow on higher level. Application execution starts from here and load feature module as requested.*



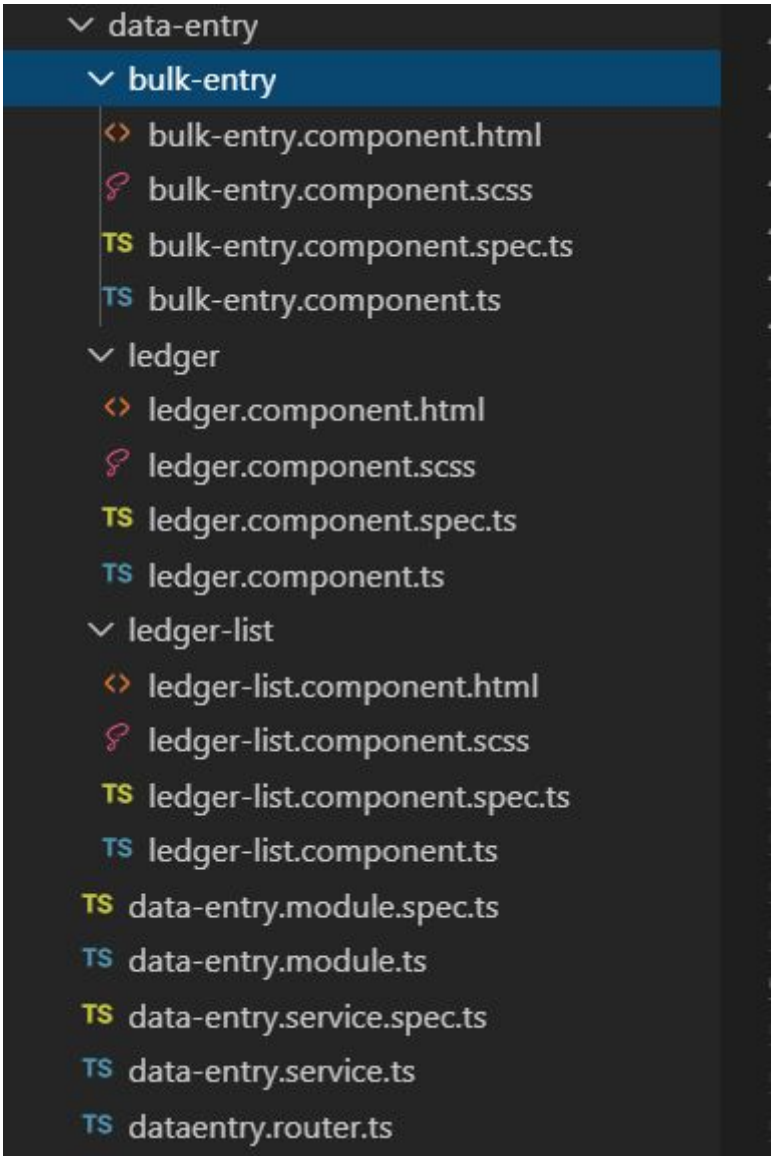
2. ***Auth Module***: *This module is responsible to handle authentication and authorization of the application. all the components like registration, login are available in this module.*



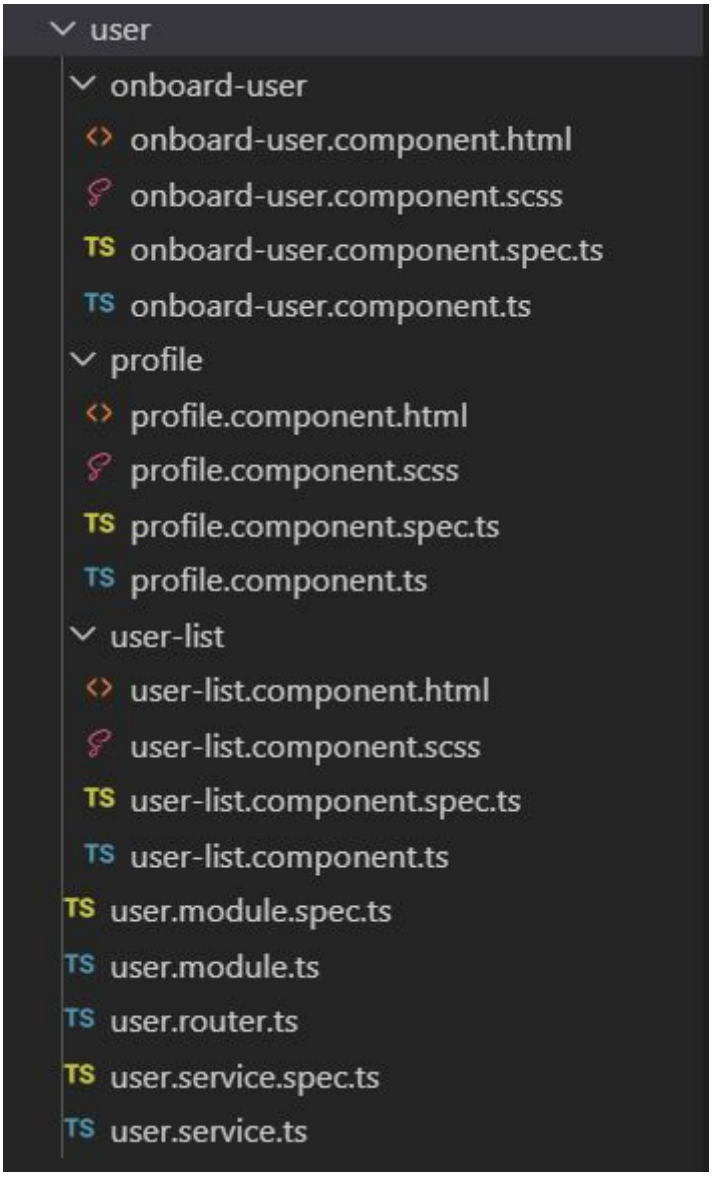
3. ***Dashboard Module***: *This module takes care of the protected pages. Different modules like Report, Data Tracker, User Management, Data Entry etc are available here.*



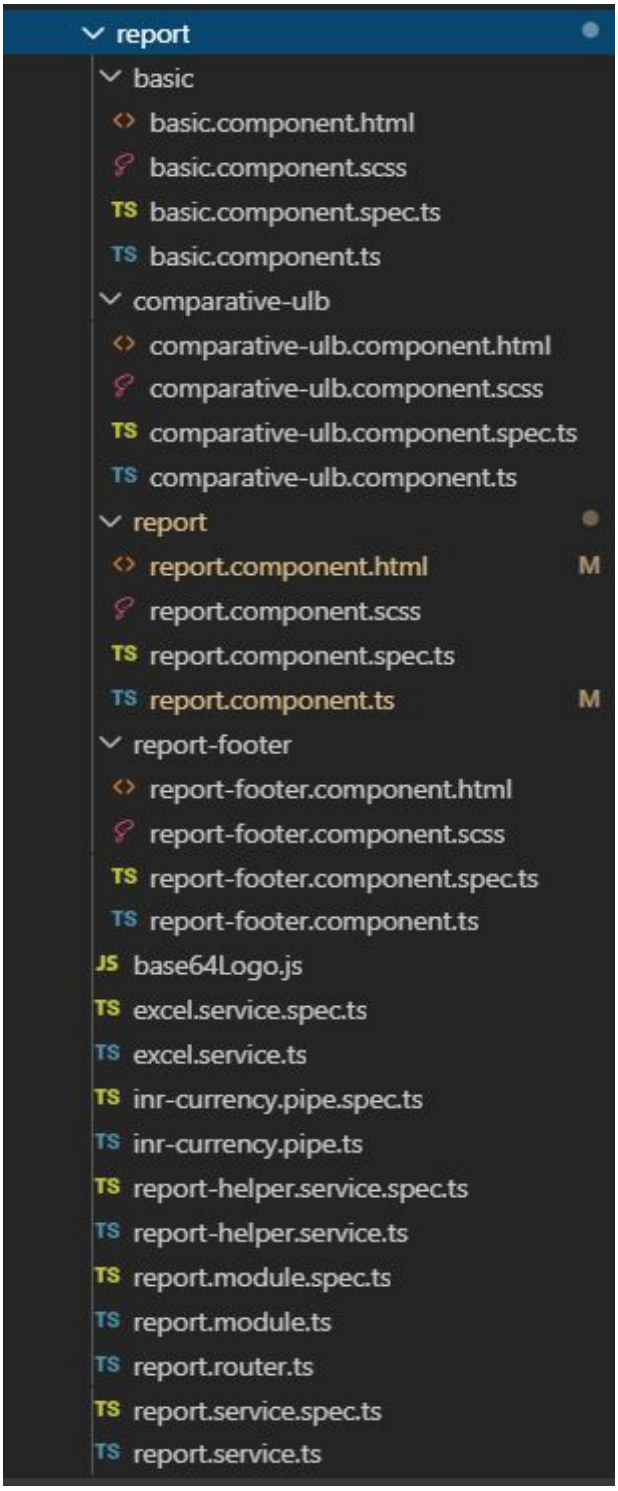
3.1. ***Data Entry Module***: *This module takes care of uploading data from the input sheet and upload them into database which will be used to generate report later. Data Entry Module consists of 3 major components which are Bulk Entry, Ledger, Ledger List component.*



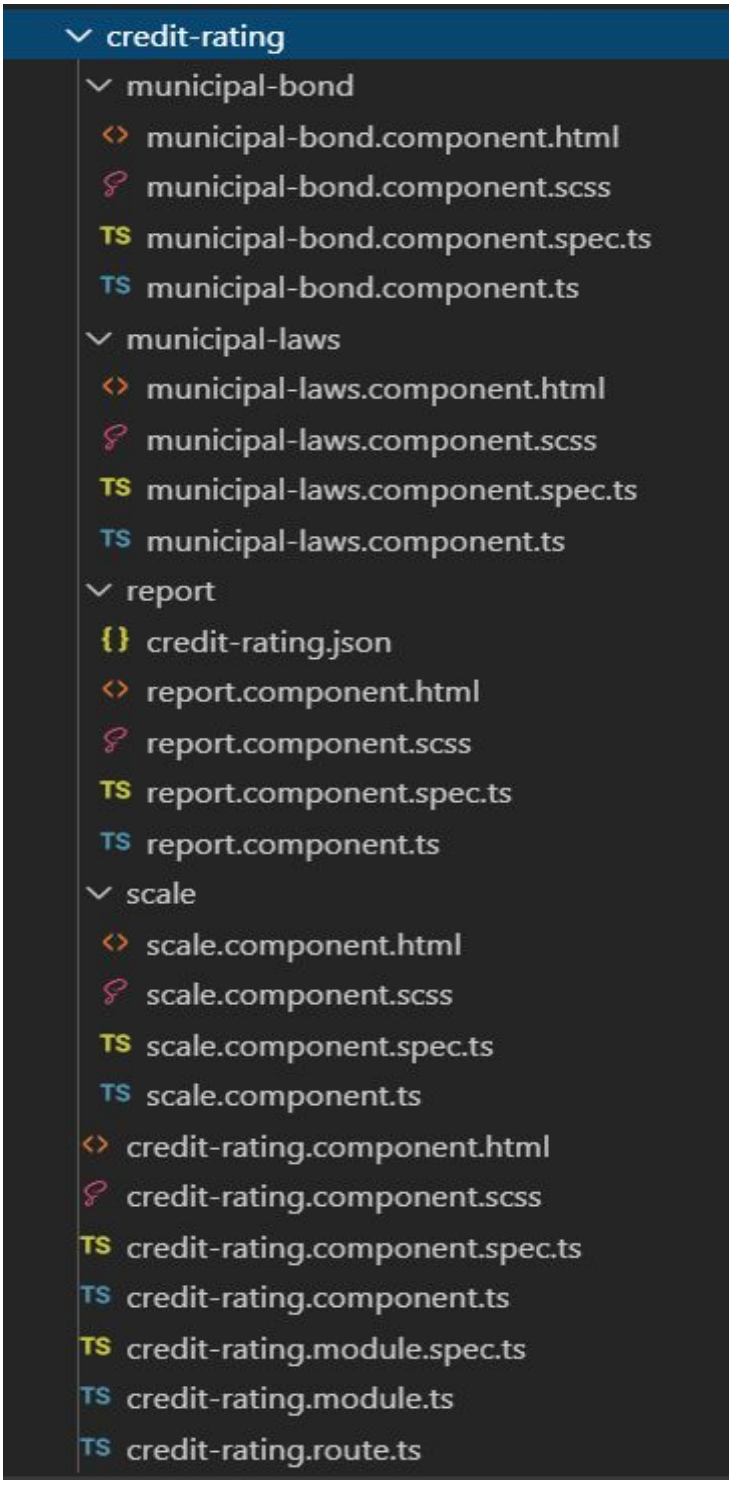
3.2. ***User Module****: This module is responsible to enable administration with User Management tools. Admin can onboard a user, define specific role and view different users with their current details available in the application.*



3.3. ***Report Module****: This module is responsible to generate report based on user's requirement/ selection/ filter applied by the user on report page. This is the heart of application and contains logic to generate report for the application. All the formula available in the FRS Sheet shared with the documentation.*



4. ***Credit Rating Module****: This module takes care of the components related to credit rating. This module handles pages like Municipal Laws, Municipal Bond, Credit Rating Report and Credit Rating Scale etc.*

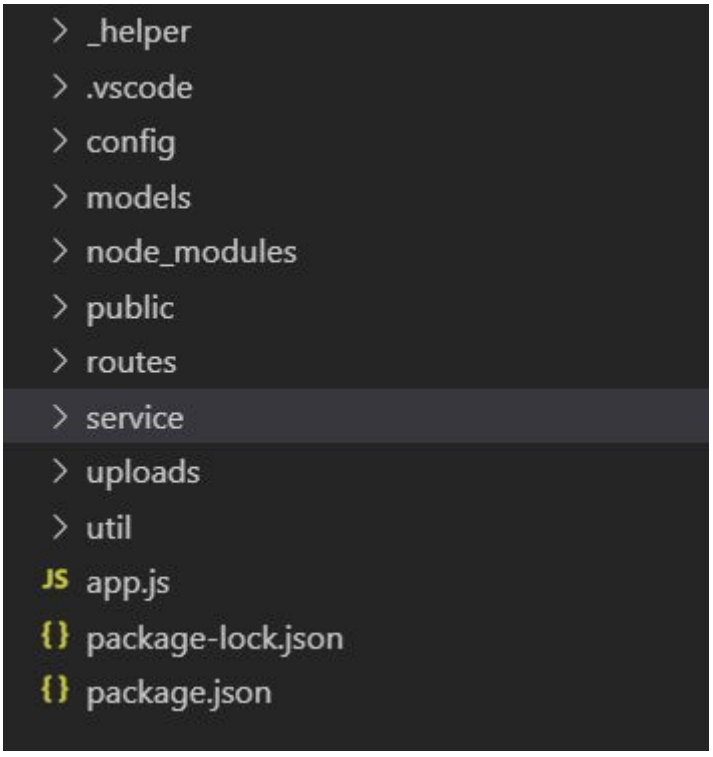


***Application Environment**:*

Application environment are required to manage and provide data to the application based on region. Currently application has 2 environments:

- ***Development Environment**: which contains data required while development of the application.*
- ***Production Environment**: which contains data when we deploy the application in production mode.*

3. BACKEND APPLICATION STRUCTURE:



Backend of the application has been built as a standard Node-Express application with the structure depicted above. Each directory has been created to define separation of concerns and their usabilities are as follows:

- ***app.js**: This is the entry point of the application. All the middleware have been configured here so before passing any request to subsequent controller or router classes, it arrives here, get mapped with provided mapping and traverse to required method.*

- ***Authentication and Authorization**: has been handled by Passport-JWT strategy. Required configuration details are available in config folder.*

- ***_helper:** This directory consist of helper classes which can be used throughout the application.*

- ***config:** As the name suggest, this directory contains all the configuration files required in the project.*

- ***models:** This directory contains all the model classes. These class work as a data access layer of the application and provides required data from the*

database to service layer.

- ***public**: This directory contains all the static resource required in the application.*

- ***routes**: All the controllers / routes / endpoints are available at this directory.*

- ***service**: This directory contains all the files with business logic on how the data needs to be curated / processed before storing to database or after retrieving from the database and before serving to the client system.*

- ***uploads**: This directory works as a resource folder and all the input files uploaded by the users gets stored at this location.*

- ***util**: Purpose of this directory is to have all reusable code or utilities to group in a single place and use them throughout the application.*

- *Application dependencies are defined below:*

```
"dependencies": {
    "express": "4.16.4",
    "mongoose": "5.3.8",
    "bcryptjs": "2.4.3",
    "cors": "2.8.4",
    "jsonwebtoken": "8.3.0",
    "body-parser": "1.18.3",
    "passport": "0.4.0",
    "passport-jwt": "4.0.0",
    "multer": "1.4.1",
    "xls-to-json-lc": "0.3.4",
    "xlsx-to-json-lc": "0.5.0"
  },
```

4. REST APIS:

// Register User ⇒

URI = /users/signup
Method = POST
Payload = {
name:<Full name of the user>,
username:<Email id of the user>,
password:<Password of the user>,
mobile:<Mobile number>,
}

// Signin User ⇒

URI = /users/signin
Method = POST
Payload = {
username:<Email id of the user>,
password:<Password of the user>,
}

// Onboard User ⇒

URI = /users/onboard
Method = POST
Payload = {
name:<Full name of the user>,
username:<Email id of the user>,
password:<Password of the user>,
mobile:<Mobile number>,
}

// Fetch all Users ⇒

URI = /users/getAll
Method = POST
Payload = {
name:<Full name of the user>,
username:<Email id of the user>,
password:<Password of the user>,
mobile:<Mobile number>,
}

// Get User Profile ⇒

URI = /users/Profile
Method = GET
Payload Header = JWT Token

// Ledger bulk entry ⇒

URI = /ledger/bulkEntry
Method = POST
Payload Header = {
year : <data year>
files : <list of input sheet files>
}

// Ledger entry ⇒

URI = /ledger/entry
Method = POST
Payload = {
year: <data year>,
state_code:<state code>,
state: <state name>,
ulb: <ulb name>,
ulb_code:<ulb code>,
wards: <number of wards>,
area: <number of area>,
population: <population of the ulb>,
}

// Ledger getIE ⇒

URI = /ledger/getIE
Method = POST
Payload = {
ulbList : <list of ulb codes>
}

// Ledger getBS ⇒

URI = /ledger/getBS
Method = POST
Payload = {
ulbList : <list of ulb codes>
}

// Lookup states ⇒

URI = /lookup/states
Method = GET

// Lookup ulbs by state ⇒

URI = /lookup//states/:stateCode/ulbs
Method = GET
Path Param = stateCode

// Lookup all states with ulbs⇒

URI = /lookup/ulbs
Method = GET

// download logs - add⇒

URI = /logs/addLog
Method = POST
payload = {
particular: req.body.particular,
mobile: req.body.mobile,
email: req.body.email,
isUserExist: req.body.isUserExist
}

// download logs - addLogByToken ⇒

URI = /logs/addLogByToken
Method = POST
payload Header = JWT Token

// download logs - getAll⇒

URI = /logs/getAll
Method = POST
payload Header = JWT Token

5. DATABASE MODELING:

- **download_log_model :**

```
const DownloadLogSchema = mongoose.Schema({
    particular: {
        type: String,
        require: true
    },
    mobile: {
        type: String
    },
    email: {
        type: String,
    },
    isUserExist: {
        type: String,
        default: false,
    },
    createdAt: {
        type: String,
        default: Date.now
    },
    updatedAt: {
        type: String,
        default: Date.now
    },
    isDeleted: {
        type: String,
        default: false,
    }

});
```

- ledger_log_model:

```
const LedgerLogSchema = mongoose.Schema({

    state_code: {
        type: String,
        required: true
    },
    state: {
        type: String,
        required: true
    },
    ulb: {
        type: String,
        required: true
    },
    ulb_code: {
        type: String,
        required: true
    },
    ulb_code_year: {
        type: String,
        required: true,
        unique: true
    },
    year: {
        type: String,
        enum: LOOKUP.BUDGET.YEAR,
        required: true
    },
    wards: {
        type: Number,
    },
    population: {
        type: Number,
        required: true
    },
    area: {
```

```
        type: Number
    },
    audit_status: {
        type: String,
        enum: LOOKUP.AUDIT.STATUS,
        required: true
    },
    audit_firm: {
        type: String
    },
    partner_name: {
        type: String
    },
    icai_membership_number: {
        type: String
    },
    created_at: {
        type: String
    },
    created_by: {
        type: String
    },
    verified_at: {
        type: String
    },
    verified_by: {
        type: String
    },
    reverified_at: {
        type: String
    },
    reverified_by: {
        type: String
    }

});
```

- ledger_model:

```
const LedgerSchema = mongoose.Schema({
    ulb_code: {
        type: String,
        required: true,
    },
    head_of_account: {
        type: String,
        required: true
    },
    code: {
        type: Number,
        required: true
    },
    groupCode: {
        type: Number,
    },
    line_item: {
        type: String,
        required: true
    },
    budget: [{
        year: {
            type: String,
            enum: LOOKUP.BUDGET.YEAR,
            required: true
        },
        amount: {
            type: Number,
            required: true
        },
    }]

});
```

- user_model:

```
const UserSchema = mongoose.Schema({
    name: {
        type: String,
        required : true
    },
    mobile: { type: String },
    username: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    },
    role: {
        type: String,
        enum: CONSTANTS.USER.ROLES,
        required: true
    },
    isActive: {
        type: String,
        default: true,
    },
    createdAt: {
        type: String,
        default: Date.now
    },
    updatedAt: {
        type: String,
        default: Date.now
    },
    isDeleted: {
        type: String,
        default: false,
    }
});
```

6. DOCUMENT CONTROL

Title: Technical Design Document

Issue: Issue 1

Date: 11 Oct 2019

Author: Vivek Sharmar

7. DOCUMENT SIGN-OFF

Nature of Signoff	Person	Date	Role
Authors	Vivek Sharma	11-Oct-2019	Consultant
Reviewers	Sumit Arora		

8. DOCUMENT CHANGE RECORD

Date	Version	Author	Change Details
11 Oct 2019	Issue 1 Draft 1	Vivek Sharma	First complete draft
23 Oct 2019	Issue 1 Draft 2	Vivek Sharma	API doc, component listing