***User Application***

**Project Structure :**

* **app\_server**

This directory contains all the code related to the server with all the functionality for interaction with databases and logic.

* **bin**

This directory contains a single file which contains code to start the http server, when 'npm start' command is executed on terminal it calls for this file for starting the application.

* **node\_modules**

This directory act as the repository for all the project's depedencies.

* **public**

This directory contains all the front end related stuff, like stylesheets, javascript code that has to be executed in browser or other http client and html or any other template webpages.

* **app.js**

This file is the main server configuration file. All the middlewares used in the project are defined here.

* **package.json**

All the dependencies, project version, project name, how to start the project are defined here.

**app\_server :**

* **controllers :**

This module contains are the controllers used in the project. These controllers contains all the logic and implementation to interact with databases.

* **models :**

This module contains models that define connections to databases and

provide CRUD operations.

* **routes :**

This module defines routers which provide controllers for each specified URL.

* **util :**

This module contains utility methods like validations check, constants.

* **views :**

This module contains all the views which can be static HTML pages or templated webpages.

**APIs Docs :**

Mongo APIs :

/mongo/login --> POST --> {username: '', password: ''}

/mongo/register --> POST --> {firstName: '', lastName: '', username: '', password: ''}

/mongo/userList --> POST --> {}

/mongo/updateUser --> POST --> {oldPassword: '', newPassword: ''}

/mongo/removeUser --> POST --> {username: ''}

Mysql APIs:

/mysql/login --> POST --> {username: '', password: ''}

/mysql/register --> POST --> {firstName: '', lastName: '', username: '', password: ''}

/mysql/userList --> POST --> {}

/mysql/updateUser --> POST --> {oldPassword: '', newPassword: ''}

/mysql/removeUser --> POST --> {username: ''}

Mysql Stored Procedure APIs :

/sp/mysql/loginWithSP --> GET --> login screen

/sp/mysql/createUserWithSP --> GET regisration screen

/sp/mysql/allUserWithFilter --> GET users list screen

Mongo Stored Procedure APIs :

/sp/mongo/loginWithSP --> GET --> login screen

/sp/mongo/createUserWithSP --> GET regisration screen

/sp/mongo/allUserWithFilter --> GET users list screen

**Mysql Tables Architecture :**

Users Table :

+-----------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+-----------+-------------+------+-----+---------+----------------+

| id | int(10) | NO | PRI | NULL | auto\_increment |

| firstName | varchar(25) | YES | | NULL | |

| lastName | varchar(25) | YES | | NULL | |

| username | varchar(25) | YES | | NULL | |

| password | varchar(25) | YES | | NULL | |

| country | int | YES | | NULL | |

| company | int | YES | | NULL | |

+-----------+-------------+------+-----+---------+----------------+

Users table is being used to store all users with there basic details and country and company Ids from country and company table respectively. These ids are saved instead of names and values due to reducing denormalization.

Country Table :

+-------------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------------+-------------+------+-----+---------+-------+

| country\_id | int | NO | PRI | NULL | |

| countryName | varchar(25) | YES | | NULL | |

+-------------+-------------+------+-----+---------+-------+

Country table contains id ( primary key ) and country name.This id field is being used as pointer from different users from users table.

Company Table :

+-------------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------------+-------------+------+-----+---------+-------+

| company\_id | int | NO | PRI | NULL | |

| companyName | varchar(25) | YES | | NULL | |

+-------------+-------------+------+-----+---------+-------+

Company table contains id ( primary key ) and company name. This id field is being used as pointer from different users from users table.

**MongoDB models :**

user : {firstName: String, lastName: String, username: String, password:String, company: String, country: String}

company : {company\_id: ObjectId, companyName: String}

country : {country\_id: ObjectId, countryName: String}

**Mysql To MongoDB conversion :**

1. In mongoDB, its usual to give up some known best practices from mysql background like normalization.
2. In user model, company and country are being stored as their real value instead of some id which acts as a pointer to another table which contains real value.
3. Normalization has been sacrificied considering two facts :
   1. By saving real values, there is no need to query the database two times as to first get users data then resolve their coutnry and company id's from company and country collections.
   2. The values of country and company are not very susceptible to change in near future, hence we don't have to update their value in every user document.

**\*\*NOTE : In this particular case, sacrificing normalization, leads to faster reads from database and update will be not their very often in near future. Bargain for faster reads with slower updates.**

**Mysql To MongoDB Migration :**

**Mongify** is the tool for the job. It is used to get data from the mysql database and import into MongoDB. Its open source too and can be changed according to the business needs.

1. Mapping the data strucuture
2. Export data from mysql and import it according to the defined structure.

**Mongoose** will be used as **ORM** for defining some structure for mongoDB. It will convert the data from Mysql according to its own structure and import it in the mongoDB.

If mysql contains normalized data then logic has to be developed for denormalizing the data before importing to the MongDB.

**Mongify will read your mysql database, build a translation file for you and all you have to do is map how you want your data transformed.  
  
It supports:**

* Updating internal IDs (to BSON ObjectID)
* Updating referencing IDs
* Typecasting values
* Embedding Tables into other documents
* Before filters (to change data manually before import)

**\*NOTE – Real procedure depends upon the current Mysql database structure, steps stated above may change according to the current schema and business use cases.**

**Security Implemention :**

Node.js provides Helmet API, that can be integrated in the ExpressJS Framework being currently used in our application. This api will be used as middleware and parse every request before reaching any logic or database layer.

**Some Services From API :**

* [dnsPrefetchControl](https://github.com/helmetjs/dns-prefetch-control) controls browser DNS prefetching
* [frameguard](https://github.com/helmetjs/frameguard) to prevent clickjacking
* [hidePoweredBy](https://github.com/helmetjs/hide-powered-by) to remove the X-Powered-By header
* [hpkp](https://github.com/helmetjs/hpkp) for HTTP Public Key Pinning
* [hsts](https://github.com/helmetjs/hsts) for HTTP Strict Transport Security
* [ieNoOpen](https://github.com/helmetjs/ienoopen) sets X-Download-Options for IE8+
* [noCache](https://github.com/helmetjs/nocache) to disable client-side caching
* [noSniff](https://github.com/helmetjs/dont-sniff-mimetype) to keep clients from sniffing the MIME type
* [xssFilter](https://github.com/helmetjs/x-xss-protection) adds some small XSS protections