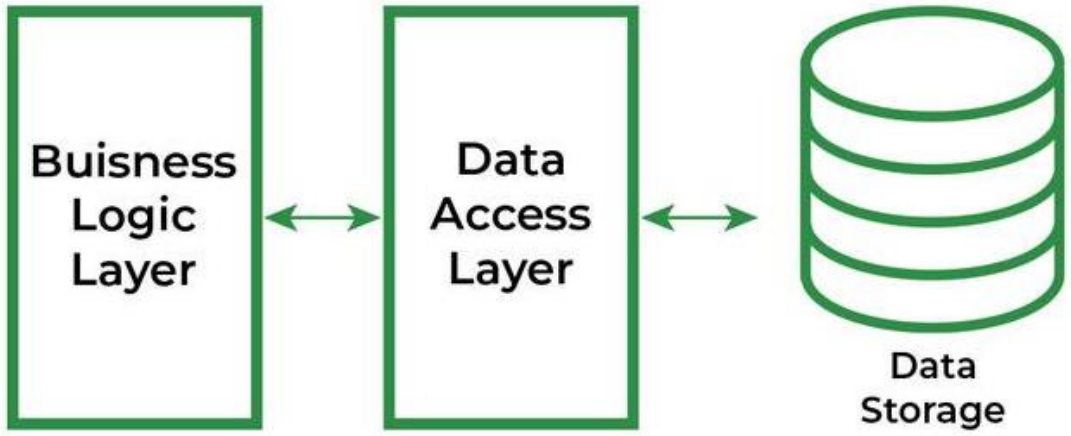
BAAS DATA ACCESS LAYER

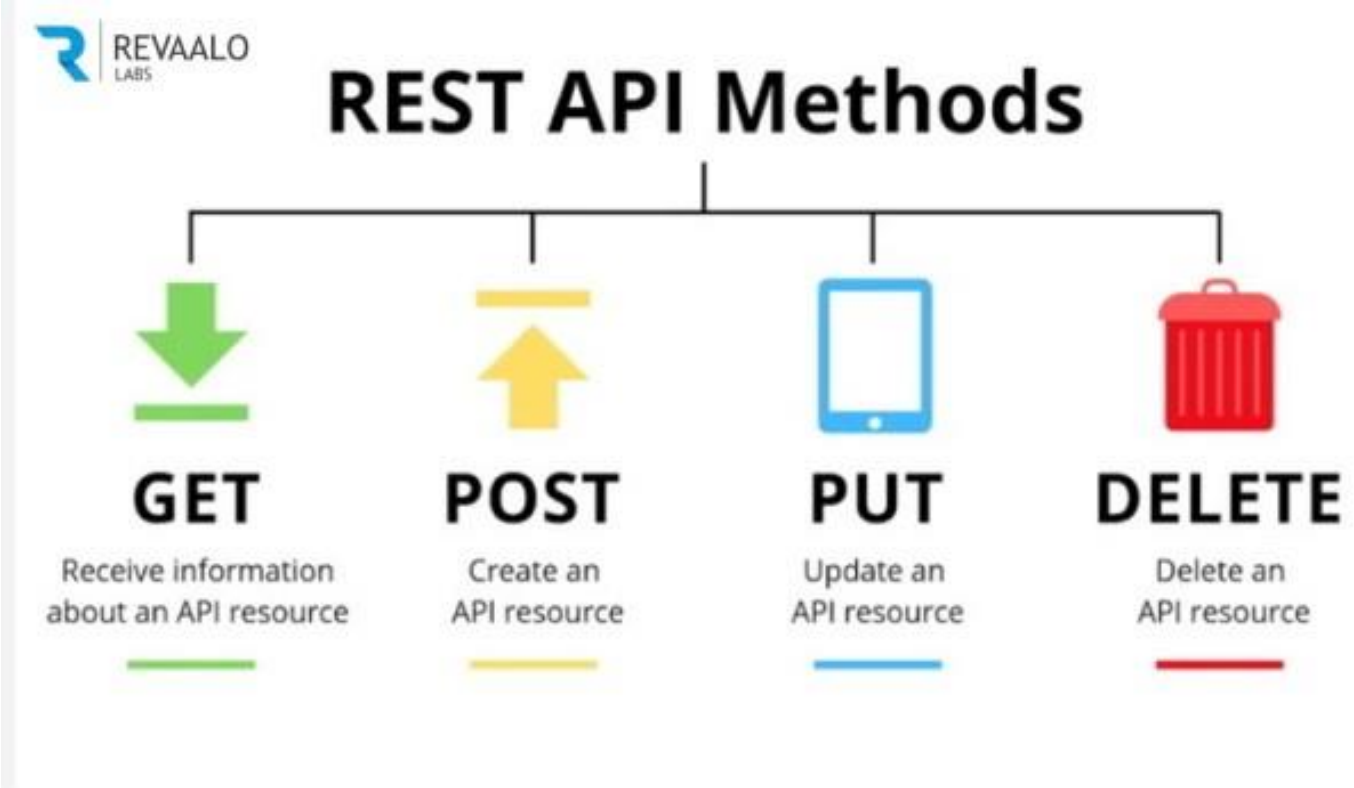
Meaning & Overview

- A data access layer (DAL) in computer software, is a layer of a computer program which provides simplified access to data stored in persistent storage of some kind, such as an entity-relational database.
- **Note: Persistent storage means a storage device that retains data after power to that device is shut off.**
- The DAL hides this complexity of the data store from the external world.
- Layman Understanding:
  DAL maintains the Database, Any software layer which want to go through/access the database first they need to go through the Data Access Layer.
- It is the place where do we perform all the **CRUD Operations**[Create, Read, Update, Delete]

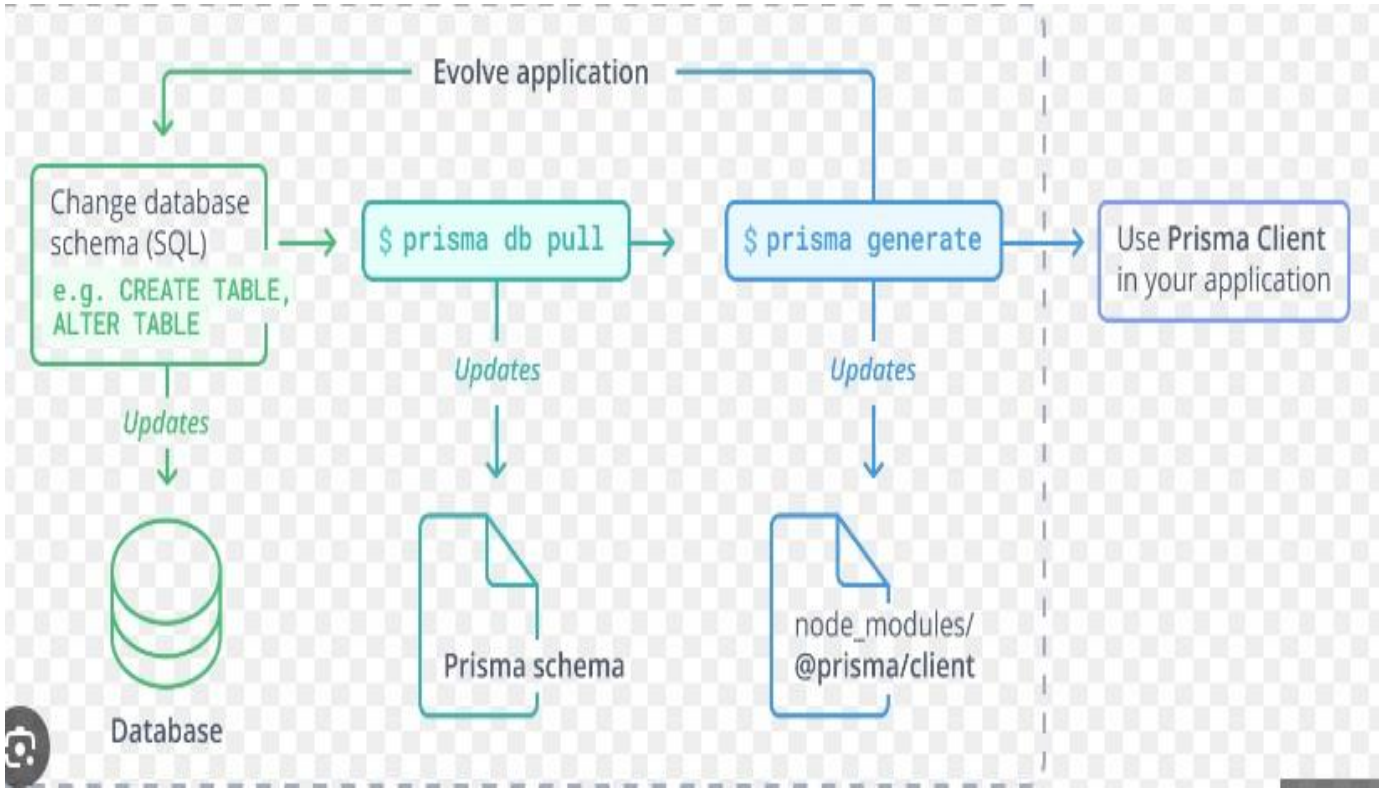


Characteristics of DAL

- It can support multiple database, so the application is able to make use of any database as per its requirement.
- Because segregating the data access code, enables better maintainability and easy migration of the database.
- It contains methods for accessing the underlying database data.
- As we read earlier, in this layer we do perform all the CRUD Operations by using all the REST API methods(GET,POST,PATCH/PUT,DELETE)



DAL Interaction with ORM



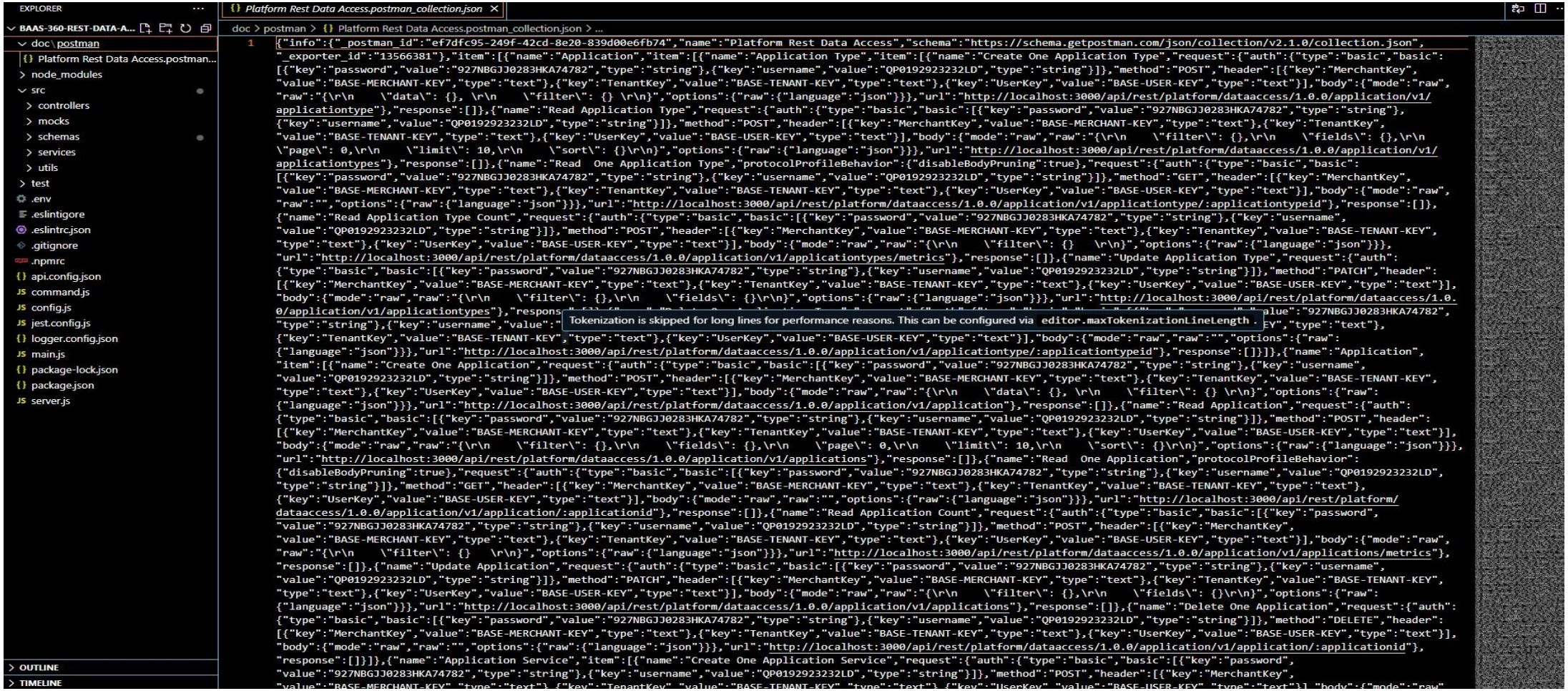
- ORM is known as Object-Relational Mapping Tools, it is a layer that connects object-oriented programming (OOP) to relational databases.
- **PRISMA:**
- Prisma is a **next-generation ORM** that consists of these tools:
- **Prisma Client**: Auto-generated and type-safe query builder for Node.js & TypeScript
- **Prisma Migrate**: Declarative data modeling & migration system
- **Prisma Studio**: GUI to view and edit data in your database
- Prisma Client can be used in *any* Node.js or TypeScript backend application (including serverless applications and microservices). This can be a **REST API**, a **GraphQL API** a gRPC API, or anything else that needs a database.

Code Base Followed in DAL

- doc
- node_modules
- src
  - controllers
  - mocks
  - schemas
  - services
  - utils
- Test
- env
- eslintigore
- eslintrc.json
- gitignore
- npmrc
- api.config.json
- command
- config
- Jest.config
- Logger.config
- main.js
- Server.js

What is doc

- Doc is a place where do we store the documentation
- Here in our docs, we do create a **postman** named folder and in that we do create a **postman collection Json**

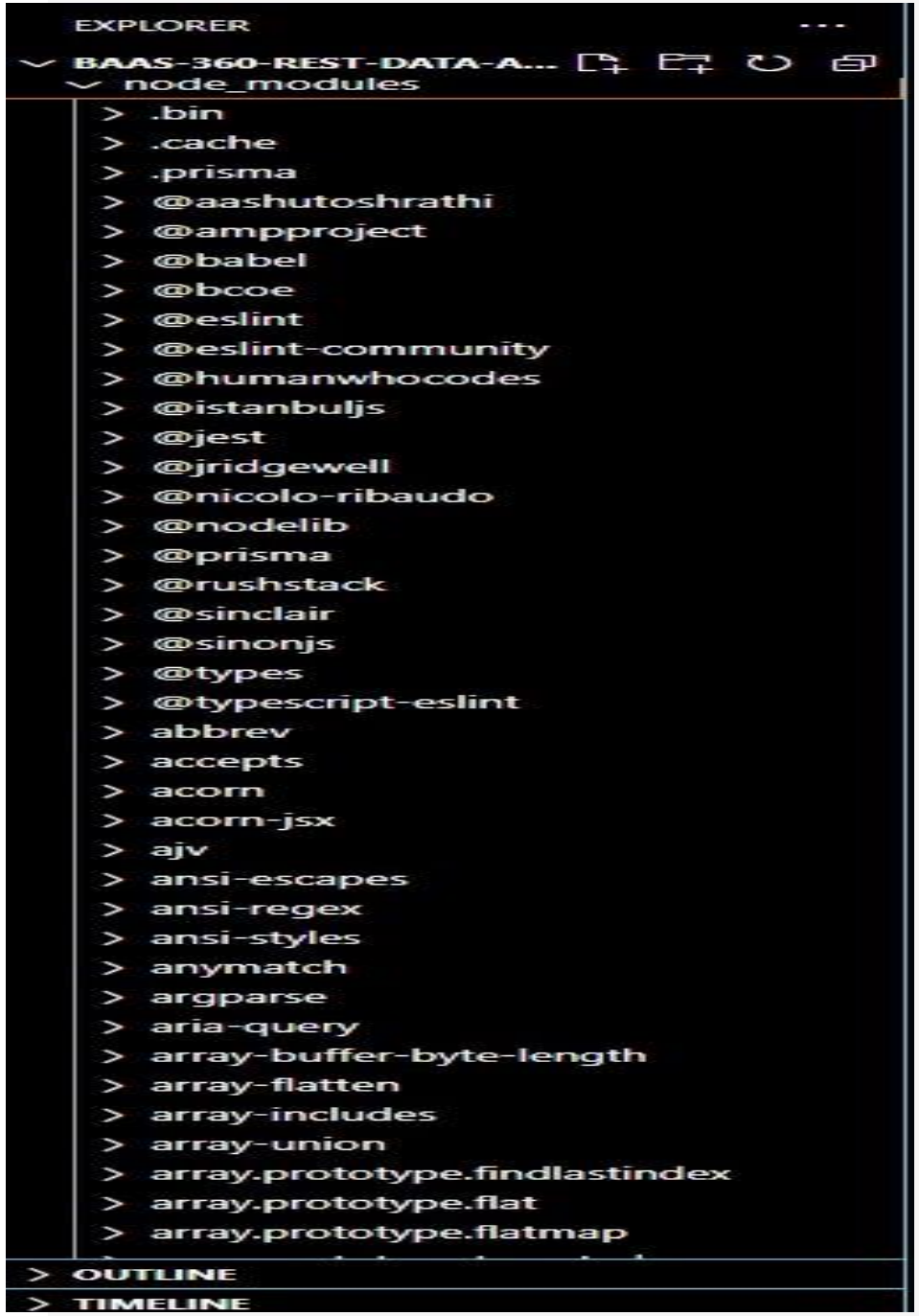


What is src

- Source folder is used to hold the primary source files for the project.
- The significance of the source folder vary depending on the context of the framework in use.
- The extension for the files is **".js"**
- It consist of controllers, mocks, schemas, services, utils.
- In schemas there would be a file called "**schema.prisma"** where there would be schemas of the required table and fields you want to provide.

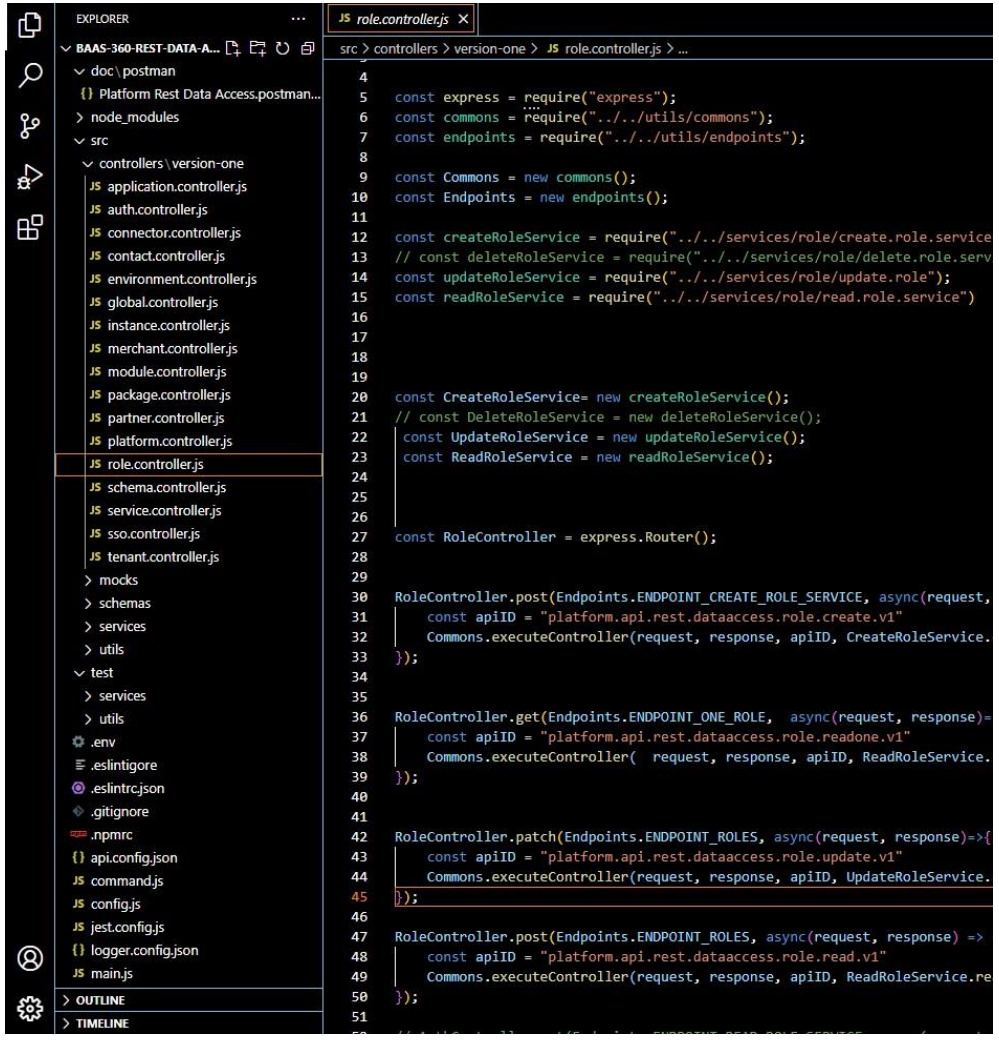
What are node_modules

- modules are nothing but a block of encapsulated code that communicate with an external application basis of their related functionality.
- modules can be single file or a collection of multiple files/folders



controllers

- Controllers is the folder which contains all the specified Api[actual Api] with the respective file in it.
- We will maintain one controller file for one module.
- Here it is an example of code base in controllers, we do write all the Api's in controller file of respective module.
- In this example methods like [post, get, delete, update] is done for the role module.



mocks

- Mock is nothing but a Json which consist of **Api id** as a **key** and data, status and message as a **value** for it.
- Mock is a **static data** of an api
- It is the structure of the actual output.

{} role.mock.json ×

src > mocks > {} role.mock.json > {} platform.api.rest.dataaccess.role.read.v1 > message

```
1  {
2      "platform.api.rest.dataaccess.role.create.v1": {
3          "data": {},
4          "status": 200,
5          "message": "[MOCK] Role has been created successfully"
6      },
7
8
9      "platform.api.rest.dataaccess.role.readone.v1":{
10         "data": {},
11         "status": 200,
12         "message": "[MOCK] Role has been read successfully"
13     },
14
15
16     "platform.api.rest.dataaccess.role.update.v1":{
17         "data": {},
18         "status": 200,
19         "message": "[MOCK] Role has been read successfully"
20     },
21
22
23     "platform.api.rest.dataaccess.role.read.v1":{
24         "data": {},
25         "status": 200,
26         "message": "[MOCK] Role has been readcompletely successfully"
27     }
28
29
30 }
```

schema.prisma 1

src > schemas > schema.prisma > ...

```
generator client {
  provider        = "prisma-client-js"
  previewFeatures = ["interactiveTransactions"]
  binaryTargets   = ["native", "debian-openssl-1.1.x", "linux-musl"]
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model Role {
  RoleName    String  @db.VarChar(250)
  Description String  @db.VarChar(250)
  CreatedBy   String  @unique @db.VarChar(250)
  CreatedDate String? @db.VarChar(250)
  RoleID      BigInt  @id(map: "Role_ID") @default(autoincrement())
}

model User {
  UserID    BigInt  @id(map: "PK_User") @default(autoincrement())
  UserName  String  @db.VarChar(250)
  Phone     String  @db.VarChar(250)
  Email     String  @unique @db.VarChar(250)
  Address   String? @db.VarChar(250)
  Education String? @db.VarChar(250)
}
```

Schemas

The main configuration file for your Prisma setup

Schema consist of the structure of the table.

Prisma schema is the place where you define the structure of your needed table.

Services

- Services is place where do we perform all the CRUD Operations by creating a six individual files for one folder
- We do write the conditions that need to be performed
- Here we connect with database

src > services > role > create.role.service.js > <unknown> > createRoleService > createOneRole > duplicateRecord > catch() callback

```
9   const Messages= new messages;
10
11  module.exports = class createRoleService{
12
13
14
15
16      async createOneRole(databaseConnector, apiID, config, input) {
17          try {
18              if (config.IsAvailable == false) { return Commons.generateServiceOutputForAvalibalityNotEnabled() }
19              if (config.IsMockEnabled == true) { return roleMock[apiID] }
20              else {
21                  let duplicateRecord = await databaseConnector.Role.findMany({
22                      where: input.filter || "null",
23                      select: { RoleID: true }
24                  }).catch((error) => {
25                      if (error instanceof Prisma.PrismaClientValidationError) {
26                          throw Commons.generateServiceOutput(null, 422, JSON.stringify(error.message))
27                      } else if (error instanceof Prisma.PrismaClientKnownRequestError) {
28                          throw Commons.generateServiceOutput(null, 422, JSON.stringify(error.message))
29                      } else {
30                          throw Commons.generateServiceOutput(null, 500, JSON.stringify(error.message))
31                      }
32                  });
33                  var filters = input.filter
34
35                  if (duplicateRecord.length === 0 || Object.keys(filters).length === 0) {
36
37                      let output = await databaseConnector.Role.create({
38                          data: input.data,
39                      }).catch((error) => {
40                          if (error instanceof Prisma.PrismaClientValidationError) {
41                              throw Commons.generateServiceOutput(null, 422, JSON.stringify(error.message))
42                          } else if (error instanceof Prisma.PrismaClientKnownRequestError) {
43                              throw Commons.generateServiceOutput(null, 422, JSON.stringify(error.message))
44                          } else {
45                              throw Commons.generateServiceOutput(null, 500, JSON.stringify(error.message))
46                          }
47                      });
48                      return Commons.generateServiceOutput(output, 200, Messages.MESSAGE_ROLE_CREATED_SUCCESSFULLY);
49                  }
50                  else {
51                      throw Commons.generateServiceOutput(null, 409, Messages.MESSAGE_DUPLICATE_RECORD)
52                  }
53              }
54          } catch (error) {
55              throw Commons.generateServiceOutput(null, 500, error.message);
56          }
57
```

```
/**
 * Common messages
 */
MESSAGE_SERVICE_RUNNING_SUCESSFULLY = "platform-Rest-Data-Access is now running on";
MESSAGE_AVAILABILITY_NOT_ENABLED = "Avaliability flag has not been enabled";
MESSAGE_MISSING_REQUEST_HEADER = "Missing 'TenantKey' or 'MerchantKey' or 'UserKey' in the request headers";
MESSAGE_INVALID_REQUEST_HEADERS = "'TenantKey' or 'MerchantKey' in the request headers is not valid";
MESSAGE_BASIC_AUTH_FAILED = "Basic authentication failed, incorrect username or password";
MESSAGE_UNABLE_TO_CONNECT_TO_DATABASE = "Unable to establish connection with the database";
MESSAGE_CONNECTED_TO_DEFAULT_DATABASE = "Connection has been established with the default database";
MESSAGE_INVALID_FILTER_OBJECT = "Request cannot be processed due to invalid filter object";
MESSAGE_DUPLICATE_RECORD = "Record with given filter already exist";

/* Messages For APIs (INJECTED USING CODE GENERATOR) */
MESSAGE_ROLE_CREATED_SUCCESSFULLY = "message realm has been created successfully"
MESSAGE_SSO_REALM_CREATED_SUCCESSFULLY = "Sso Realm has been created successfully";
MESSAGE_SSO_REALM_READ_SUCCESSFULLY = "Sso Realm has been read successfully";
MESSAGE_SSO_REALM_HAS_BEEN_FOUND = "Sso Realm has been readone successfully";
MESSAGE_SSO_REALM_READ_COUNT_SUCCESSFULLY = "Sso Realm count has been read successfully";
MESSAGE_SSO_REALM_UPDATED_SUCCESSFULLY = "Sso Realm has been updated successfully";
MESSAGE_SSO_REALM_DELETED_SUCCESSFULLY = "Sso Realm has been deleted successfully";
MESSAGE_SSO_REALM_NOT_FOUND="Sso Realm has not been found for the given filter"
MESSAGE_SSO_SERVER_CREATED_SUCCESSFULLY = "Sso Server has been created successfully";
MESSAGE_SSO_SERVER_READ_SUCCESSFULLY = "Sso Server has been read successfully";
MESSAGE_SSO_SERVER_HAS_BEEN_FOUND = "Sso Server has been readone successfully";
MESSAGE_SSO_SERVER_READ_COUNT_SUCCESSFULLY = "Sso Server count has been read successfully";
```

```
// Platform endpoint of the REST service
ENDPOINT_BASE_URL = "/api/rest/platform/dataaccess";

/* INJECT_ENDPOINT_FOR_DIFFERENT_MODULES */
ENDPOINT_MODULE_SSO = "/1.0.0/sso"
ENDPOINT_MODULE_TENANT = "/1.0.0/tenant"
ENDPOINT_MODULE_SERVICE = "/1.0.0/service"
ENDPOINT_MODULE_SCHEMA = "/1.0.0/schema"
ENDPOINT_MODULE_PLATFORM = "/1.0.0/platform"
ENDPOINT_MODULE_PARTNER = "/1.0.0/partner"
ENDPOINT_MODULE_PACKAGE = "/1.0.0/package"
ENDPOINT_MODULE_MODULE = "/1.0.0/module"
ENDPOINT_MODULE_MERCHANT = "/1.0.0/merchant"
ENDPOINT_MODULE_INSTANCE = "/1.0.0/instance"
ENDPOINT_MODULE_GLOBAL = "/1.0.0/global"
ENDPOINT_MODULE_ENVIRONMENT = "/1.0.0/environment"
ENDPOINT_MODULE_CONTACT = "/1.0.0/contact"
ENDPOINT_MODULE_CONNECTOR = "/1.0.0/connector"
ENDPOINT_MODULE_AUTH = "/1.0.0/auth"
ENDPOINT_MODULE_APPLICATION = "/1.0.0/application"
ENDPOINT_MODULE_ROLE = "/1.0.0/role"


//ENDPOINT_CREATE_ROLE_SERVICE = "/role"
//ENDPOINT_READ_ROLE_SERVICE = "/role/:roleid"

// Endpoint versions for the REST Service
```

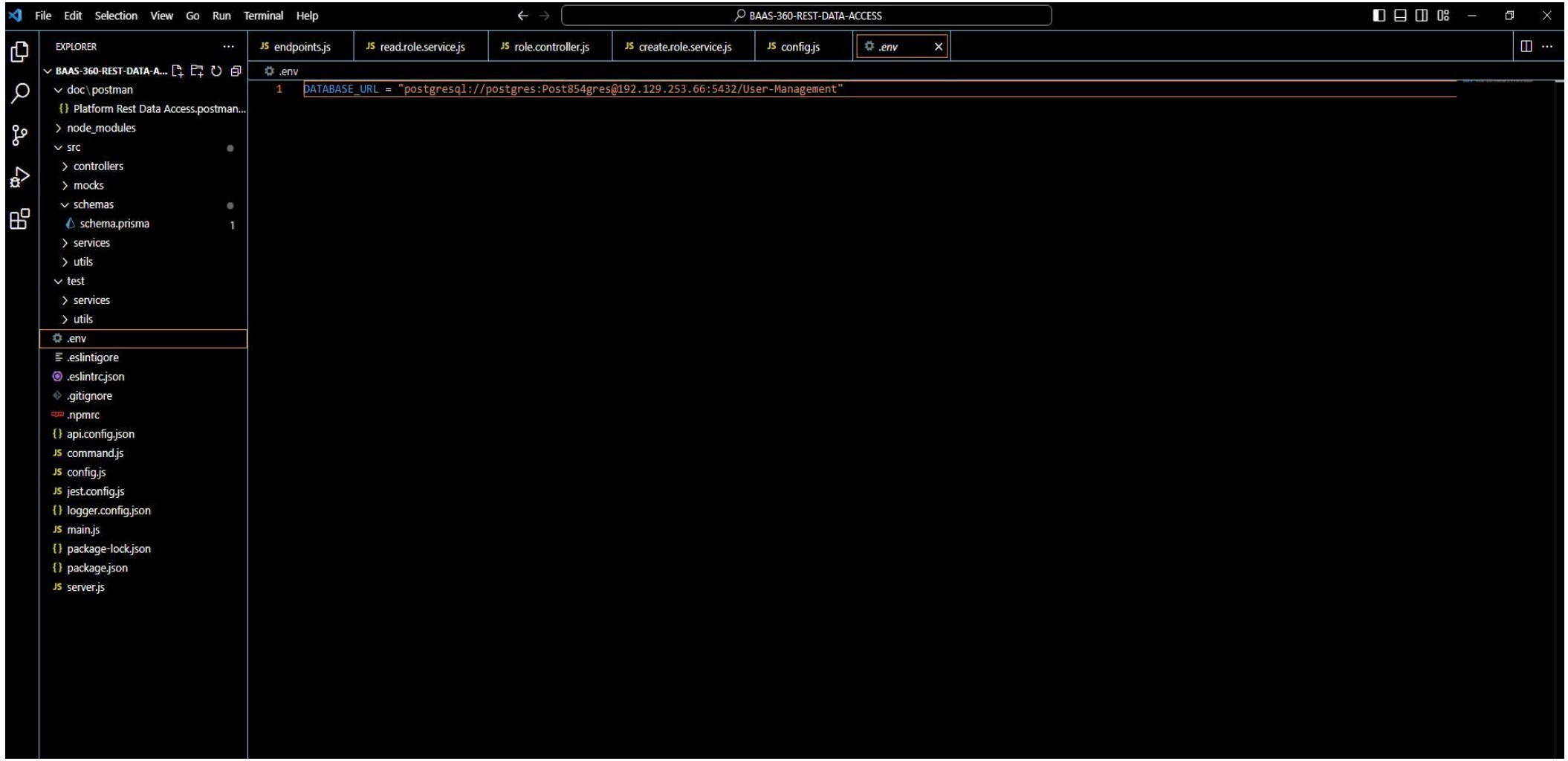
```
executeController(request, response, apiID, serviceFunction) {
// Initializing global variables
let executionStartTime = null; let executionEndTime = null; let serviceOutput = null;
try {
    executionStartT  import apiConfig  etCurrentDateTime();
    console.log(`${apiConfig[apiID].Name} API EXECUTION STARTS AT (${executionStartTime})`);
    const merchatKey = request.headers.merchantkey; const tenantKey = request.headers.tenantkey;
    this.validateRequestHeaders(merchatKey, tenantKey, userKey);

    const databaseConnection = this.getDatabaseConnection(merchatKey, tenantKey);
    const databaseConnector = this.generateDatabaseConnector(databaseConnection);

    if (apiID.includes("readone") || apiID.includes("deleteone")) {
        serviceOutput = await serviceFunction(databaseConnector, apiID, apiConfig[apiID], request.
    }
    else {
        serviceOutput = await serviceFunction(databaseConnector, apiID, apiConfig[apiID], request.
    }
    console.log(`${apiConfig[apiID].Name} SERVICE OUTPUT : (${serviceOutput})`);
    const controllerOutput = this.generateControllerOutput(response, serviceOutput.data, serviceOu
    response.json(this.parseControllerOutput(controllerOutput));
    executionEndTime = await this.getCurrentDateTime();
    console.log(`${apiConfig[apiID].Name} API EXECUTION ENDS AT (${executionEndTime})`);
    console.info({
        requestURL: request.originalUrl,
        requestMethod: request.method,
        requestHeaders: request.headers,
        requestBody: JSON.stringify(request.body),
        responseBody: JSON.stringify(this.parseControllerOutput(controllerOutput)),
        executionStartTime: executionStartTime,
        executionEndTime: executionEndTime
    });
} catch (error) {
    console.error("ERROR IN EXECUTE CONTROLLER FUNCTION : ", error)
    if (error.status == null) {
        let controllerOutput = this.generateControllerOutput(response, null, 500, error.message,
        response.json(controllerOutput);
    }
    let controllerOutput = this.generateControllerOutput(response, error.output ? error.data : nul
    response.json(this.parseControllerOutput(controllerOutput));
    executionEndTime = await this.getCurrentDateTime();
    console.error({
        requestURL: request.originalUrl,
        requestMethod: request.method,
        requestHeaders: request.headers,
```

utils

- It consist of commons, endpoints and messages
- Commons:-It consists of code for service response, executing controllers, handling unauthorized responses, validates request headers and many more
- Endpoints:-It contains all the endpoints required for the application
- Messages:-It contains common messages that are required when the application is successful or failed.

env

- It consist of database URL , the database you are going to use .



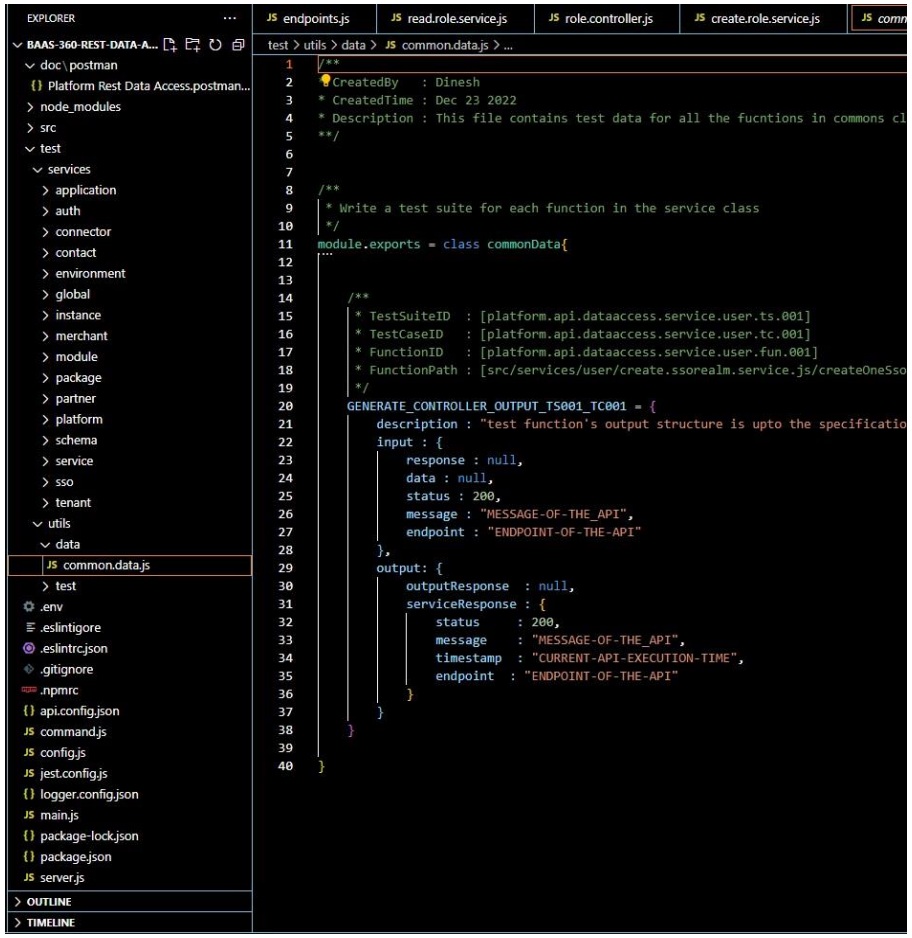
test

Test is for unitest

It consist of data and service

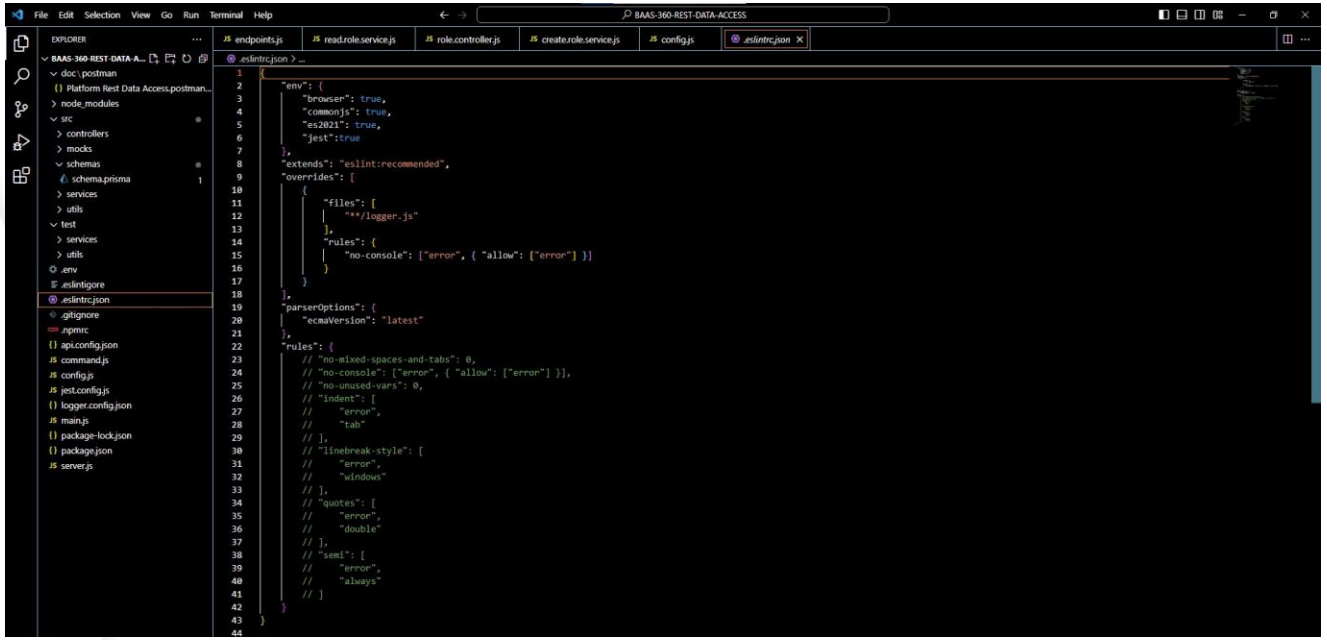
We write jest for unitest

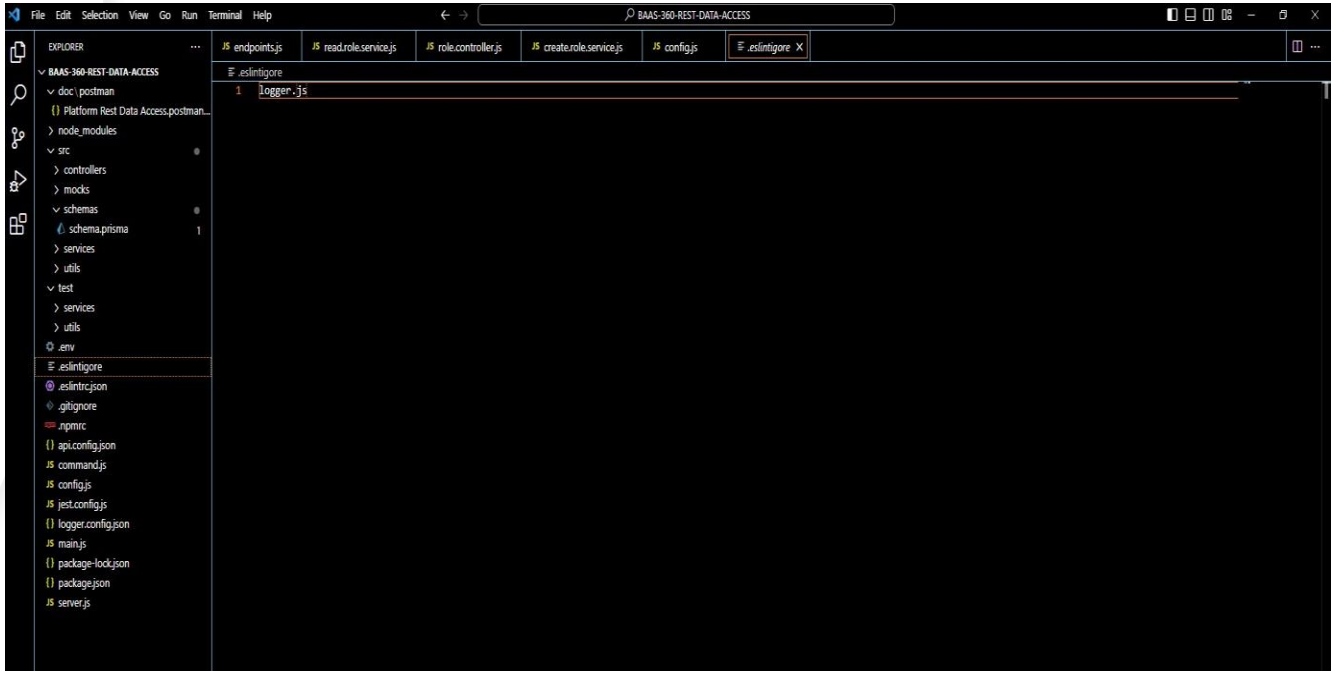
It is file we write all the code in for the one common class



eslintigore & eslintrc.json

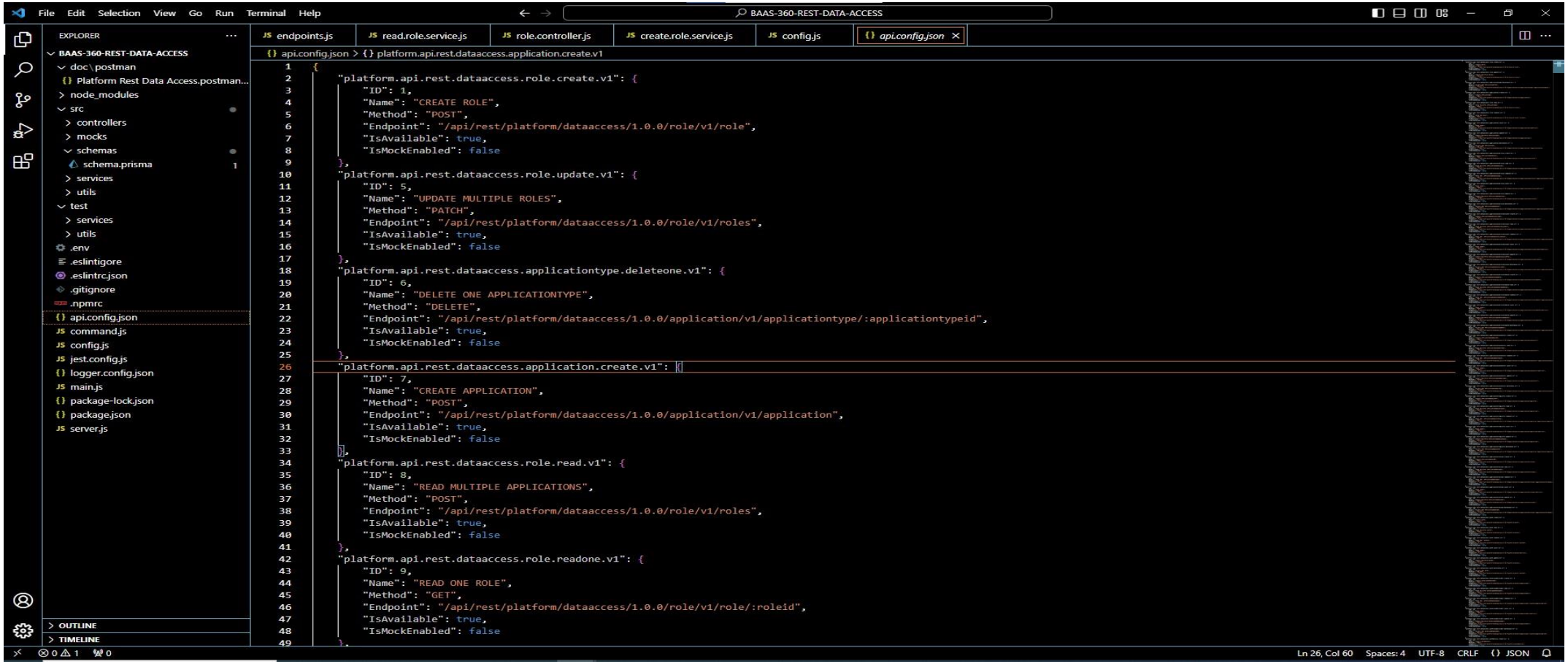
- Configuration changes based on environment
- Eslintrc is nothing but a static code analysis. It also consist of rules which are user-defined.
- eslintigore is a file which consist the files which need to ignored at the time of execution.





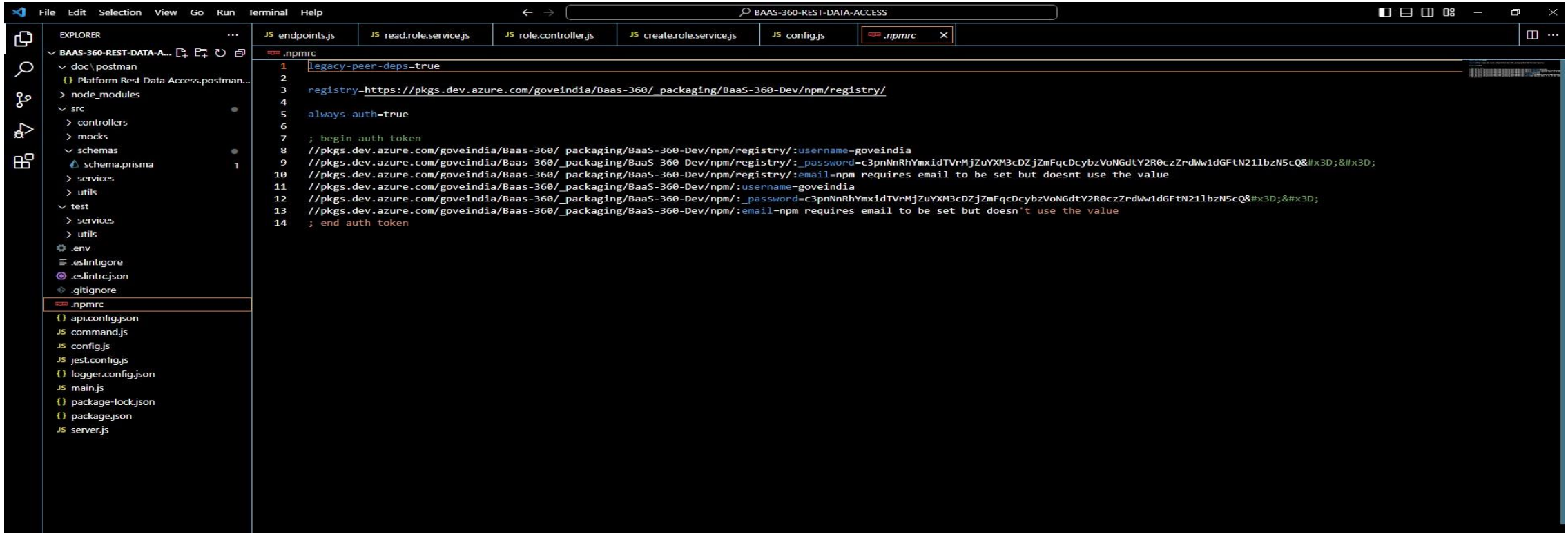
api.config.json

- This consist of json which contains all the details like api method and api id the keys which you want to enable and disable.



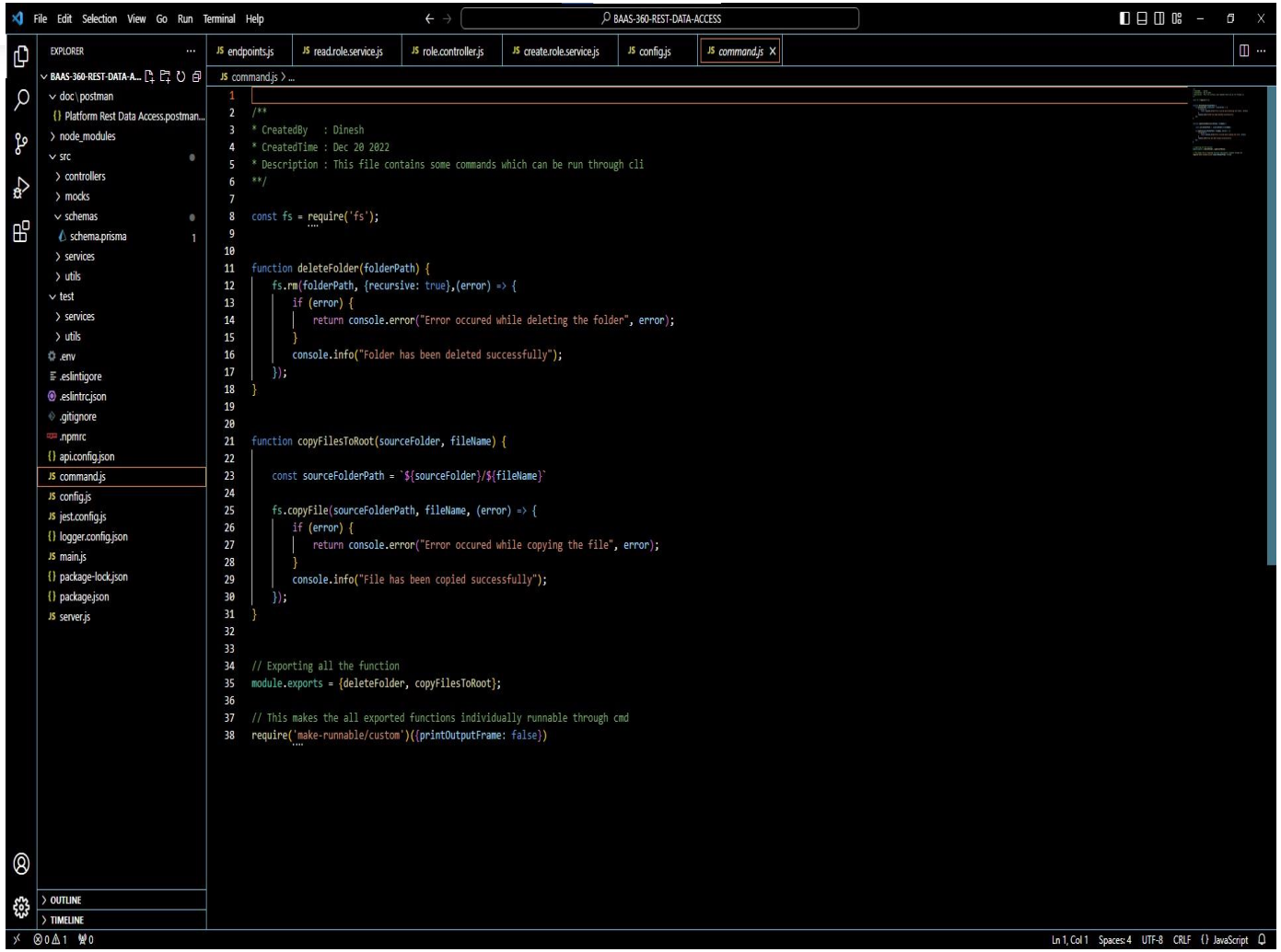
npmrc

- We use npmrc to configure a private registry in project.
- It manages the npm config file.
- It defines the settings on how NPM should behave when running commands.



command

- It contains some commands which run through cli
- It uses file system to give path where file need to be copied and deleted.
- **All file system operations have synchronous, callback, and promise-based forms, and are accessible using both Common JS syntax and ES6 Modules (ESM).**



Config

- It contains all the configuration that is required for the service.
- It contains basic and database configurations to access the services.
- It also contains basic authorization details.

```
module.exports = class config{
...

    // Basic Configuration
    PLATFORM_REST_DATA_ACCESS_NAME = "Platform-Rest-Data-Access";
    PLATFORM_REST_DATA_ACCESS_HOST = "localhost";
    PLATFORM_REST_DATA_ACCESS_PORT = "3001";

    PLATFORM_REST_DATA_ACCESS_USERNAME = "QP0192923232LD";
    PLATFORM_REST_DATA_ACCESS_PASSWORD = "927NBGJJ0283HKA74782";

    // Platform-Rest-Data-Access datatabase Connection Configurations
    PLATFORM_REST_DATA_ACCESS_DEFAULT_DATABASE_CONNECTION_ENABLED = true;

    PLATFORM_REST_DATA_ACCESS_DATABASE_CONNECTIONS = {
        isDefault    : true,
        databaseURL  : "postgresql://postgres:Post854gres@192.129.253.66:5432/User-Management",
        databaseName : "role"
    };
};
```

Jest.config

- It gives the details and allows to know how much the code is true and false.

```
const jestConfig = {

    "verbose": false,
    "reporters": [
        "default",
        ["./node_modules/jest-html-reporter", {
            "pageTitle": `Platform Data Access - Unit Test Report`,
            "dateFormat":"mm-dd-yyyy HH:MM:ss",
            "includeFailureMsg" : true,
            "outputPath": `test/.reports/Platform Rest Data Access - Unit Test Report.html`
        }]
    ]
};

module.exports = jestConfig;
```

Main.js

- This is the file where service gets started , it is considered as the entry point of the service.
- Connection with the database would be enabled here.
- It starts the service based on the security configuration.
- It consist of package-lock-Json and package-Json.

JS main.js > ...

```js
/**
 * CreatedBy    : Dinesh
 * CreatedTime  : Dec 23 2022
 * ModifiedBy   : Dinesh
 * ModifiedTime : Jan 23 2022
 * Description  : This file is the entry point for the service
 **/

/**
 * Importing all the required modules
 */
const config = require("./config");

const messages = require("./src/utils/messages");
const commons = require("./src/utils/commons");

const RestDataAccess = require("./server");

/**
 * Initializing objects from the imported classes
 */
const Config = new config();
const Commons = new commons();
const Messages = new messages();


/**
 * Connect with the Database if static database connection if dynamic database connection is enabled
 */
if (Config.PLATFORM_REST_DATA_ACCESS_DEFAULT_DATABASE_CONNECTION_ENABLED){
    Commons.generateDefaultDatabaseConnector();
}


/**
 * Starting the service based on the security configuration
 */
RestDataAccess.listen(Config.PLATFORM_REST_DATA_ACCESS_PORT);
console.log(Messages.MESSAGE_SERVICE_RUNNING_SUCESSFULLY + ` {${Config.PLATFORM_REST_DATA_ACCESS_HOST}` + ":" + `${Config.PLATFORM_REST_DATA_ACCESS_PORT}} `);
```

Package-lock. Json & package. Json

- Package-lock. Json is automatically generated for any operations where npm modifies either the node_modules tree, or package. Json. It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.
- It is the manifest file of any Node.js project and contains the metadata of the project. The package.json file is the essential part to understand, learn and work with the Node.js. It is the first step to learn about development in Node.js.


```
{} package.json > ...
1  {
2      "name": "platform-rest-data-access",
3      "version": "1.0.0",
4      "description": "This is a REST API Service which acts as data access layer for the database",
5      "main": "main.js",
       ▷ Debug
6      "scripts": {
7          "test": "jest",
8          "local": "nodemon main.js",
9          "start": "pm2 start main.js --name platformrestdataaccess --watch",
10         "schema-generate": "npx prisma generate",
11         "schema-pull": "npx prisma db pull",
12         "schema-push": "npx prisma db push",
13         "eslint": "npx eslint . --fix --max-warnings=120"
14     },
15     "prisma": {
16         "schema": "src/schemas/schema.prisma"
17     },
18     "pre-commit": [
19         "eslint",
20         "test"
21     ],
22     "keywords": [],
23     "author": {
24         "name": "Dinesh",
25         "email": "dinesh@baas360.com"
26     },
27     "license": "ISC",
28     "dependencies": {
29         "@prisma/client": "^5.4.2",
30         "axios": "^1.1.3",
31         "compression": "^1.0.8",
32         "cors": "^2.8.5",
33         "express": "^4.18.2",
34         "express-basic-auth": "^1.2.1",
35         "jest-html-reporter": "^3.7.0",
36         "jsonwebtoken": "^9.0.0",
37         "lodash": "^4.17.21",
38         "make-runnable": "^1.4.1",
39         "nodemon": "^2.0.20"
40     },
41     "devDependencies": {
42         "@types/jest": "^29.2.5",
43         "eslint": "^8.0.0",
44         "eslint-config-react-app": "^7.0.1",
45         "eslint-plugin-cypress": "^2.12.1",
46         "jest": "^29.3.1",
47         "pre-commit": "^1.2.2",
48         "prisma": "^5.4.2",
```

```
{} package-lock.json > {} packages > {} "" > {} devDependencies
1  {
2      "name": "platform-rest-data-access",
3      "version": "1.0.0",
4      "lockfileVersion": 2,
5      "requires": true,
6      "packages": {
7          "": {
8              "name": "platform-rest-data-access",
9              "version": "1.0.0",
10             "license": "ISC",
11             "dependencies": {
12                 "@prisma/client": "^5.4.2",
13                 "axios": "^1.1.3",
14                 "compression": "^1.0.8",
15                 "cors": "^2.8.5",
16                 "express": "^4.18.2",
17                 "express-basic-auth": "^1.2.1",
18                 "jest-html-reporter": "^3.7.0",
19                 "jsonwebtoken": "^9.0.0",
20                 "lodash": "^4.17.21",
21                 "make-runnable": "^1.4.1",
22                 "nodemon": "^2.0.20"
23             },
24             "devDependencies": {
25                 "@types/jest": "^29.2.5",
26                 "eslint": "^8.0.0",
27                 "eslint-config-react-app": "^7.0.1",
28                 "eslint-plugin-cypress": "^2.12.1",
29                 "jest": "^29.3.1",
30                 "pre-commit": "^1.2.2",
31                 "prisma": "^5.4.2",
32                 "supertest": "^6.3.3"
33             }
34         },
35         "node_modules/@aashutoshrathi/word-wrap": {
36             "version": "1.2.6",
37             "resolved": "https://pkgs.dev.azure.com/goveindia/Baas-360/_packaging/BaaS-360-Dev/npm/registry/@aashutoshrathi/word-wrap/-/word-wrap-1.2.6.tgz",
38             "integrity": "sha1-vZFUrsmYP3ezoDTsqgFcLkIB9s8=",
39             "dev": true,
40             "license": "MIT",
41             "engines": {
42                 "node": ">=0.10.0"
43             }
44         },
45         "node_modules/@ampproject/remapping": {
46             "version": "2.2.1",
47             "resolved": "https://pkgs.dev.azure.com/goveindia/Baas-360/_packaging/BaaS-360-Dev/npm/registry/@ampproject/remapping/-/remapping-2.2.1.tgz",
48             "integrity": "sha1-mejhGFES14cCzVfDNoTx0PJgtjA=",
49             "dev": true,
```

Server.JS

- In this we do add the middle wares to the express.
- In this we provide endpoints
- The Endpoints are nothing the URL which is required to access the data access layer.
- We add the service information endpoint without any authentication.
- We do inject all the endpoint modules to the respective controllers.

```
JS server.js > ...
/* Importing all the controller classes (INJECTED USING CODE GENERATOR) */
const SsoControllerVersionOne = require("./src/controllers/version-one/sso.controller");
const TenantControllerVersionOne = require("./src/controllers/version-one/tenant.controller");
const ServiceControllerVersionOne = require("./src/controllers/version-one/service.controller");
const SchemaControllerVersionOne = require("./src/controllers/version-one/schema.controller");
const PlatformControllerVersionOne = require("./src/controllers/version-one/platform.controller");
const PartnerControllerVersionOne = require("./src/controllers/version-one/partner.controller");
const PackageControllerVersionOne = require("./src/controllers/version-one/package.controller");
const ModuleControllerVersionOne = require("./src/controllers/version-one/module.controller");
const MerchantControllerVersionOne = require("./src/controllers/version-one/merchant.controller");
const InstanceControllerVersionOne = require("./src/controllers/version-one/instance.controller");
const GlobalControllerVersionOne = require("./src/controllers/version-one/global.controller");
const EnvironmentControllerVersionOne = require("./src/controllers/version-one/environment.controller");
const ContactControllerVersionOne = require("./src/controllers/version-one/contact.controller");
const ConnectorControllerVersionOne = require("./src/controllers/version-one/connector.controller");
const AuthControllerVersionOne = require("./src/controllers/version-one/auth.controller");
const ApplicationControllerVersionOne = require("./src/controllers/version-one/application.controller");

const rolecontrollerversionone=require("./src/controllers/version-one/role.controller");

const familycontrollerversionone = require("./src/controllers/version-one/family.controller");
const familymembercontrollerversionone = require("./src/controllers/version-one/familymember.controller");



/* Inject the endpoints to the respective controller modules */
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_SSO + Endpoints.ENDPOINT_VERSION_1, SsoControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_TENANT + Endpoints.ENDPOINT_VERSION_1, TenantControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_SERVICE + Endpoints.ENDPOINT_VERSION_1, ServiceControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_SCHEMA + Endpoints.ENDPOINT_VERSION_1, SchemaControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_PLATFORM + Endpoints.ENDPOINT_VERSION_1, PlatformControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_PARTNER + Endpoints.ENDPOINT_VERSION_1, PartnerControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_PACKAGE + Endpoints.ENDPOINT_VERSION_1, PackageControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_MODULE + Endpoints.ENDPOINT_VERSION_1, ModuleControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_MERCHANT + Endpoints.ENDPOINT_VERSION_1, MerchantControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_INSTANCE + Endpoints.ENDPOINT_VERSION_1, InstanceControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_GLOBAL + Endpoints.ENDPOINT_VERSION_1, GlobalControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_ENVIRONMENT + Endpoints.ENDPOINT_VERSION_1, EnvironmentControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_CONTACT + Endpoints.ENDPOINT_VERSION_1, ContactControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_CONNECTOR + Endpoints.ENDPOINT_VERSION_1, ConnectorControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_AUTH + Endpoints.ENDPOINT_VERSION_1, AuthControllerVersionOne);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_APPLICATION + Endpoints.ENDPOINT_VERSION_1, ApplicationControllerVersionOne);

RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_ROLE + Endpoints.ENDPOINT_VERSION_1, rolecontrollerversionone);

RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_FAMILY + Endpoints.ENDPOINT_VERSION_1,familycontrollerversionone);
RestDataAccess.use(Endpoints.ENDPOINT_BASE_URL + Endpoints.ENDPOINT_MODULE_FAMILYMEMBER + Endpoints.ENDPOINT_VERSION_1,familymembercontrollerversionone);

module.exports = RestDataAccess;
```

(property) endpoints.ENDPOINT_VERSION_1: string