1. This Node-API is serving the endpoints to register the vendor profile, get the vendor profile, update the existing or registered vendor profile.
2. This API can able to delete the vendor profile and has the mail functionality to send the registration confirmation, query confirmation and sending OTPs to registered mobile or mails.
3. Which uses OTP functionality to verify the vendor while showing vendor profile along with their products.
4. Which is built on top of express module to serve the web requests and corresponding responses.
5. **server.js** file is the entry point of this API.

**Dependencies and modules involved in this API:**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Module** | **Version** |
| 1. | node | 10.15.3 |
| 2. | npm | 6.4.1 |
| 3. | async | 3.2.0 |
| 4. | crypto | 1.0.1 |
| 5. | express | 4.17.1 |
| 6. | randomstring | 1.1.5 |
| 7. | fs | Built-in |
| 8. | Body-parser | Built-in |
| 9. | winston | 3.2.1 |
| 10. | App-root-path | Built-in |
| 11. | cors | 2.8.5 |
| 12. | mysql | 2.18.1 |
| 13. | mysql2 | 2.1.0 |
| 14. | sequelize | 5.21.7 |
| 15. | nodemailer | 6.4.6 |
| 16. | email-templates | 7.0.4 |
| 17. | pug | 2.0.4 |
| 18. | properties-reader | 2.0.0 |
| 19. | aws-sdk | 2.679.0 |

**.JS files Mainly Involved in this API:**

|  |  |
| --- | --- |
| **S.no** | **JS file** |
|  | server.js |
|  | vendorController.js |
|  | vendorModel.js |
|  | vendorRoutes.js |
|  | vendorFetchService.js |
|  | vendorRegisterService.js |
|  | vendorUpdateService.js |
|  | vendorDeleteService.js |
|  | vendorOTPService.js |
|  | securityUtil.js |
|  | vendorUtil.js |
|  | winston\_Logger.js |
|  | mailUtil.js |
|  | otpUtil.js |
|  | db.config.js |
|  | vendor.model.js |
|  | products.model.js |
|  | productFamilies.model.js |
|  | currency.model.js |
|  | reviews.model.js |
|  | countries.model.js |
|  | sequelize.js |
|  | phoneUtil |

**a**. **Data Base Configuration:**

1. **db.config.js:** This API uses **MySQL** for persisting the data. And to communicate with the database from the API it uses **sequelize**. The Sequelize is an ORM framework for Node JS, which is used to map the defined models with the database tables, and also generates the dynamic schema for the table creation based on the how the model’s have been defined. And the Sequelize can able to generates the dynamic queries for SELECT, UPDATE, INSERT and DELETE based on the methods we used like **findOrCreate**, **findOne**, **findAll** etc.

And the Sequelize can also manages the associations between the tables like one to one, one to many and many to many.

This file contains the basic DB configuration for sequelize with host, port, user , password and database.

1. **Sequelize.js:** This **js** file contains the logic for mapping the models with the DB tables and configuring the associations between the tables. Here models are vendor.model.js , products.model.js, currency.model.js etc.
2. **Schema:**
3. TableName**: vendor**

use catalog;

CREATE TABLE `vendor` ( `qr\_code` bigint(20) NOT NULL, `company\_name` varchar(256) DEFAULT NULL, `name` varchar(256) DEFAULT NULL, `email` varchar(256) DEFAULT NULL, `phone` bigint(20) DEFAULT NULL, `company\_address` varchar(256) DEFAULT NULL, `site` varchar(256) DEFAULT NULL, `is\_approved` tinyint(1) DEFAULT NULL, `comments` varchar(256) DEFAULT NULL, `token` varchar(256) DEFAULT NULL, `secret\_key` varchar(256) DEFAULT NULL, `timestamp` bigint(20) DEFAULT NULL, `zip\_code` char(6) DEFAULT NULL, `category` varchar(256) DEFAULT NULL, `language` varchar(256) DEFAULT NULL, `submitted\_date` varchar(256) DEFAULT NULL, `updated\_date` varchar(256) DEFAULT NULL, `country` varchar(256) DEFAULT NULL, `state` varchar(256) DEFAULT NULL, `city` varchar(256) DEFAULT NULL, PRIMARY KEY (`qr\_code`)) ENGINE=InnoDB DEFAULT CHARSET=utf8;

1. TableName**: currency**

use catalog;

CREATE TABLE `currency` ( `id` int(11) NOT NULL AUTO\_INCREMENT, `country\_id` int(11) DEFAULT NULL, `code` char(3) DEFAULT NULL, PRIMARY KEY (`id`), KEY `country\_id` (`country\_id`), CONSTRAINT `currency\_ibfk\_1` FOREIGN KEY (`country\_id`) REFERENCES `country` (`id`) ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8

1. Table Name: **country**

use catalog;

CREATE TABLE `country` ( `id` int(11) NOT NULL AUTO\_INCREMENT, `name` varchar(50) DEFAULT NULL, `code` varchar(4) DEFAULT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;

1. Table Name: **product\_family**

use catalog;

CREATE TABLE `product\_family` ( `id` int(11) NOT NULL AUTO\_INCREMENT, `name` varchar(256) DEFAULT NULL, `parent\_id` int(11) DEFAULT NULL, `full\_path` varchar(256) DEFAULT NULL, PRIMARY KEY (`id`), KEY `parent\_id` (`parent\_id`), CONSTRAINT `product\_family\_ibfk\_1` FOREIGN KEY (`parent\_id`) REFERENCES `product\_family` (`id`) ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB AUTO\_INCREMENT=8 DEFAULT CHARSET=utf8

1. Table Name: **product**

use catalog;

CREATE TABLE `product` ( `id` int(11) NOT NULL AUTO\_INCREMENT, `sku` char(8) NOT NULL, `name` varchar(256) DEFAULT NULL, `short\_description` varchar(256) DEFAULT NULL, `long\_description` varchar(256) DEFAULT NULL, `price` double DEFAULT NULL, `currency\_id` int(11) DEFAULT NULL, `vendor\_id` bigint(20) DEFAULT NULL, `quantity` varchar(256) DEFAULT NULL, `product\_family\_id` int(11) DEFAULT NULL, `small\_image\_path` varchar(256) DEFAULT NULL, `standard\_image\_path` varchar(256) DEFAULT NULL, `image\_caption` varchar(256) DEFAULT NULL, `is\_active` tinyint(1) DEFAULT NULL, PRIMARY KEY (`id`), KEY `currency\_id` (`currency\_id`), KEY `vendor\_id` (`vendor\_id`), KEY `product\_family\_id` (`product\_family\_id`), CONSTRAINT `product\_ibfk\_1` FOREIGN KEY (`currency\_id`) REFERENCES `currency` (`id`) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT `product\_ibfk\_2` FOREIGN KEY (`vendor\_id`) REFERENCES `vendor` (`qr\_code`) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT `product\_ibfk\_4` FOREIGN KEY (`product\_family\_id`) REFERENCES `product\_family` (`id`) ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB AUTO\_INCREMENT=30 DEFAULT CHARSET=utf8

1. Table Name: **reviews**

use catalog;

CREATE TABLE `reviews` ( `id` int(11) NOT NULL, `comment` varchar(255) DEFAULT NULL, `user\_id` int(11) DEFAULT NULL, `ip` varchar(255) DEFAULT NULL, `product\_id` int(11) DEFAULT NULL, `rating` float DEFAULT NULL, PRIMARY KEY (`id`), KEY `product\_id` (`product\_id`), CONSTRAINT `reviews\_ibfk\_1` FOREIGN KEY (`product\_id`) REFERENCES `product` (`id`) ON DELETE NO ACTION ON UPDATE CASCADE) ENGINE=InnoDB DEFAULT CHARSET=utf8

**Note :**  Here catalog is the database name.

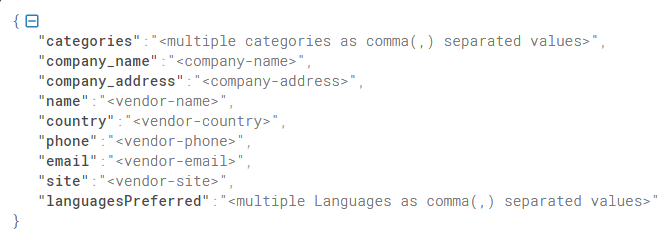
**b. Technical Functionality of the Application:**

1. **Server.js :** As mentioned earlier this file is the entry point of the application, and like normal Node applications which contains the express module instance, port configuration and routing instance. Any request (URI ) send to this API will be forwarded to **vendorRoutes** from this file.
2. **vendorRoutes.js :** This file is responsible for the routing in the complete API. Like normal applications the request will be mapped to the corresponding controller functions based on the URI in this file. In this API will be having eight URI’s routing for all the incoming requests namely: 1. ***/uni-commerce/vendor*** and 2***. /uni-commerce/vendors*** 3. /***send/query*** 4. ***/uni-commerce/vendors/decrypt*** 5. ***/uni-commerce/vendors/encrypt*** 6. ***/uni-commerce/vendor/delete*** 7. ***/uni-commerece/vendor/generate-otp*** 8***. /uni-commerce/vendor/verify***.9. ***/uni-commerce/vendor/products. 10.***  ***/uni-commerce/vendor/outlet.*** All these URI’s will be mapped / routed to corresponding controller functions based on request type (POST, GET, PUT, DELETE) as specified. Each controller function will be invoked by passing the corresponding request (req), response (res) objects.
3. **vendorController.js :** This file is responsible for processing the request and sending the corresponding response based on routing info.

**Available functions:**

1. **Create\_Vendor :** This function is invoked by the **vendorRoutes** when the request type is POST with the URI ‘***/uni-commerce/vendor’***with the request body shown below:

Sample request body of **POST** with the URI ‘***/uni-commerce/vendor***’:



This function is responsible to parse the request body and get the simple values, invoking the **vendorRegisterService** by passing the above simple values as arguments and sending response retuned by the service.

1. **get\_Vendor :** This function is invoked by the **vendorRoutes** when the request type is GET with the URI e.g. ‘***/uni-commerce/vendor?id=<phone>/<qrCode>/<email>***)***’*** *.*

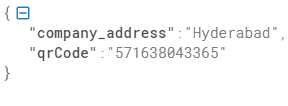
This function is responsible to read the query string parameter (**id**) value from the request and invoking the **fetchVendor** function of the **vendorFetchService** by passing the search param value as argument send the response got from the service.

1. **get\_Vendors :** This function is invoked by the **vendorRoutes** when the request type is GET with the URI e.g.’ ***/uni-commerce/vendors’*** *.*

This function is responsible to invoke the **fetchAllVendors** function of the **vendorFetchService** and getting the all existing vendor info , send the info got from the service as response.

1. **update\_Vendor :** This function is invoked by the **vendorRoutes** when the request type is PUT with the URI ‘***/uni-commerce/vendor’***with the request body shown below:

e.g.: updating company\_address by using qrCode.



This function is responsible to parse the request body and get the simple values, invoking the **vendorUpdateService** by passing the above simple values as arguments and sending response returned by the service.

1. **delete\_Vendor** : This function is invoked by the **vendorRoutes** when the request type is DELETE with URI **‘/uni-commerce/vendor/delete’** with the header data as **isDelete** and **id**.

Here **isDelete**=true&**id**=<qrCode/phone/email>. Here *isDelete* is just an additional flag kept for vendor deletion and *id* will be the vendor QR code or phone or email.

This function is responsible to collect the query params (*isDelete* and *id*) from the incoming request and verify them against undefined, empty and null and invokes the **vendorDeleteService** by passing these query params and sending the response returned by the service.

**Note :** Here Delete type would be soft deletion with the flag **isApproved.** If isApproved flag is *false,* then that vendor is considered as *deleted* with proper *comments* and vice-versa. If the URI is **/uni-commerce/vendor/hard-delete** and Header data as **isDelete**=true, **id**=<qrCode/phone/email> and user, password with appropriate values then the hard vendor profile deletion will takes place.

Here user, password will be verified against the data (encrypted format) present in users.json file.

1. **send\_Mail :** This function is invoked by the **vendorRoutes** when the request type is POST with URI **‘/send/query’** with the sample request body as shown below:

Sample request body:



This function is responsible to parse the JSON request body to simple values and invoking the **mailUtil.sendMail** function by passing these values and returning the response.

1. **send\_otp :** This function is invoked by the **vendorRoutes** when the request type is POST with URI **‘/uni-commerce/vendor/generate-otp’** with vendor mail in the request body as JSON object. This function is responsible to parse the vendor mail from the request body and invokes the required functions to send OTP to the given vendor mail and returns the appropriate response.
2. **Verify\_otp :** This function is invoked by the **vendorRoutes** when the request type is POST with URI **‘/uni-commerce/vendor/verify’** with sample request body as shown below:

Sample request body:



This function is responsible to parse the request body for vendor mail and OTP and invokes the required functions to verify the vendor OTP and returns the appropriate response.

1. **vendorRegisterService.js :** This file contains the business logic to save the vendor profile in the DB with qr\_code as primary key which will be generated logically. This function will be invoked by the v**endorController**.**create\_vendor** function by passing the vendor-data as json object.

And once the vendor is successfully saved into the DB, then this service invokes the **mailUtil.sendMail** with appropriate vendor info to send registration confirmation mail to the vendor.

1. **vendorfetchService.js :** This file contains two functions namely:
2. **fetchVendor:**

This function contains the business logic to get the specific vendor info from the DB based on vendor phone or email or qrCode . This function will be invoked by the v**endorController**.**get\_vendor** function by passing the search parameter as argument.

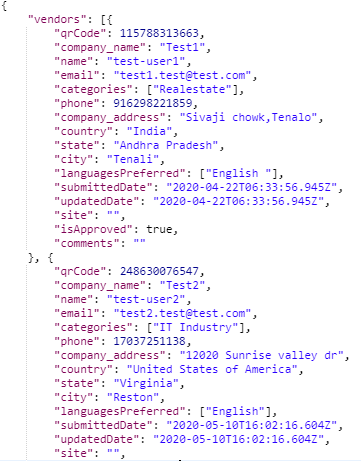
**sample response:**



1. **fetchAllVendors:**

This function contains the business logic to get the all vendors info from the DB. This function will be invoked by the v**endorController**.**get\_Vendors** function. And the result of this function would be array of vendor profiles.

**Sample response:**



**}]**

**}.**

1. **vendorUpdateService:**

This file contains the business logic to update the vendor profile in the DB. This function will be invoked by the v**endorController**.**update\_Vendor** function by passing the vendor-data in simple values as arguments.

Once registered, the vendor info can be updated by using either phone or qrCode or email. Based on the argument values from the **vendorController.update\_Vendor** the respective vendor will be fetched from the DB and updated based on provided data. It will through 404 error if is the vendor is not already registered.

The result of this function would be whether vendor updated or not.

**Sample response:**

Success: **Updates submitted successfully. One of our agents will reach you soon. Thank you for your business.**

Failed**: Unable to update vendor profile as no profile matching with the provided data. One of our agents will reach you soon. For quick response, please reach our support team at Contact Us. Thank you for your business.**

1. **vendorDeleteService** :

This file contains (**delete\_vendor**) the business logic to update the vendor profile from the DB. This function will be invoked by the v**endorController**.**delete\_Vendor** function by passing the vendor-data in simple values as arguments.

The vendor info can be deleted by using either phone or qrCode or email. Based on the argument values from the **vendorController.delete\_Vendor** the respective vendor will be marked as deleted. i.e updating the vendor **is\_approved** status as **FALSE** with appropriate **comments**.

**Note**: This file contains one more function i.e **hard\_delete\_vendor:** This file will delete the actual vendor profile from the DB permanently.

The result of these functions would be whether vendor deleted or not.

**Sample response:**

Success: **Vendor profile deleted successfully. Thank you for your business.**

Failed: **Unable to delete vendor profile as no data found. Thank you for your business**.

1. **vendorOTPService:**

This file contains the business logic to save the generated vendor otp, timestamp and verify the vendor otp functionality.

1. If the vendor is found, then this **update\_vendor** function will update the generated otp and corresponding timestamp in the DB for future usage.
2. If user enters the otp for the verification the **verify\_vendor** come into picture and takes the entered otp and compares the current timestamp with the updated timestamp in the DB. If OTP is valid and timestamp difference is 10Mins then the oulet will be sent as response.

**Sample response:**

*Success* ***: {***

***"products": [{***

***"sku": "B65R8F2F",***

***"name": "Ice tea (Georgia) Regular",***

***"desciption": "Ice tea (Georgia) Regular",***

***"price": 65,***

***"currency": "INR",***

***"is\_active": true,***

***"reviews": [{***

***"id": 1,***

***"rating": "4",***

***"comment": "Good Taste",***

***"user\_id": 12,***

***"ip": "12.064.874",***

***"product\_id": 1***

***},***

***{***

***"id": 2,***

***"rating": "5",***

***"comment": "Good Taste",***

***"user\_id": 12,***

***"ip": "12.064.874",***

***"product\_id": 2***

***}***

***],***

***"images": {***

***"small": "assets/products/ice-tea.jpg",***

***"standard": "assets/products/ice-tea.jpg",***

***"image\_caption": "Ice tea (Georgia) Regular"***

***},***

***"outlet": {***

***"qrcode": 983918150281,***

***"name": "Royal India Desserts"***

***},***

***"filters": {***

***"product\_family": "beverages",***

***"outlet\_location": "Guntur",***

***"outlet\_state": "AP",***

***"outlet\_pincode": "522003"***

***}***

***}***

***],***

***"vendor": {***

***"qrCode": 983918150281,***

***"company\_name": "Uniti LLC",***

***"name": "Sekhar",***

***"email": "munisekhar.muni123@gmail.com",***

***"categories": [***

***'Hotels'***

***],***

***"phone": 9052792215,***

***"company\_address": "Madhapur Kalahasti",***

***"country": "India",***

***"state": "Andra Pradesh",***

***"city": "Guntur",***

***"languagesPreferred": [***

***"Telugu",***

***"Hindi"***

***],***

***"submittedDate": "",***

***"updatedDate": "",***

***"site": "",***

***"isApproved": false,***

***"comments": "not paid bill"***

***}***

***}***

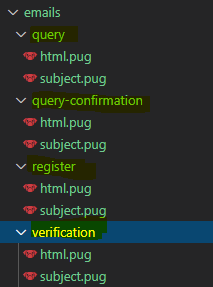
Failed: OTP entered is invalid and or expired. Please request for the new OTP.

1. **vendorModel:** This file contains the Vendor, Vendors constructors with arguments as like the data-format to be saved into the file as json.
2. **Vendor.model.js, products.model.js, currencty.model.js …etc. :** These files defines the model for each table in the DB.
3. **vendorUtil:** This file contains the util function for generating the qrCode and tokens while registering the vendor. The qrCode logic uses the Math function to generate the 12 digit integer value for every new vendor while registering.

And the token logic uses the **randomString** module to generate one unique alpha-numeric token for every new vendor while registering. And this token will be used to encrypt or decrypt that vendor info while required.

1. **mailUtil:** This file contains the logic to send the mail by using appropriate email template based on email type like registration-confirmation or vendor-query or query-confirmation or sending OTP. To achieve this mail functionality **nodemailer**, **pug** and **email-template** modules have been used here.

For each email type (specified above) respective email-template will be used. As shown below each email-template contains two pug files **html** and **subject.** Here query, query-confirmation, register and verification are template names. The html and subject pug files allow placeholders, html elements as well.

 x

1. **otpUtil:** This file contains the logic required to generate the OTP and to verify the generated OTP. This OTP functionality uses **Speakeasy** module of Nodejs to generate and verify the OTP. As of now the OTP life span is 11 mins, i.e. after 11 mins it expires.
2. **Winston\_Logger:** This file contains the logging logic for the entire API. Which is built on top of **Winston** module, which is the most commonly used module for Node applications, and as Winston module is not providing the support for the logging the log caller file name in the log file, we are getting the log caller name by using the module instance of each file. i.e. any JS file while importing the Winston logging module needs to pass their **module** instance to get the log instance.
3. **phoneUtil:** This file contains the logic required to send the generated OTPs to the registered mobile number. To send the OTPs to the mobile phones this takes the advantage of Amazon pinpoint service. The AWS configuration is stored in the config.json file with access key, secret key and region of the AWS service. And the application id of the pinpoint service, message Type etc. will be stored in the app.properties file.

Once the vendor phone is verified with the DB i.e existing or not, then that number will be the destination number for the AWS pinpoint service to send the OTP.

**NOTE :** Before using the AWS pinpoint service in the API, we need to create the application in AWS Pinpoint service manually, and **need to enable the SMS channel** in it, then only we can send messages through AWS pinpoint service.

To create the application in AWS Pinpoint service, go through the following reference link : <https://docs.aws.amazon.com/pinpoint/latest/developerguide/mobile-push-create-project.html>.

We just need to create a project and enable the SMS channel that is it. Once it is done, take the application ID from the created project use it in the Node API.

**NOTE: app.properties** file contains the DB properties like host, user, password and email configurations and AWS pinpoint configurations etc. in encrypted format.