**BRISELLE WEB APPLICATION**

The front-end/ client of the web application is built using **React.JS**

The backend/server is built using **Node.JS** and uses APIs for interaction with the client.

**Express.JS** is the Node framework for building websites which also provides essential libraries providing various functionalities like cors for cross origin resource sharing so as to allow the front-end to access its resources.

The database used is **Postgres** which avails the advantages of sql as well as nosql as it has provisions to store and parse **JSON** data.

**DATABASE**

The database consists of 3 tables:

* The object\_info table – which stores the data related to an object which is created by the user
* The field\_info table – which stores the data related to fields of a particular object. An object can have n number of fields.
* The data\_values table – which stores the values entered into each of the fields by the end-customer.

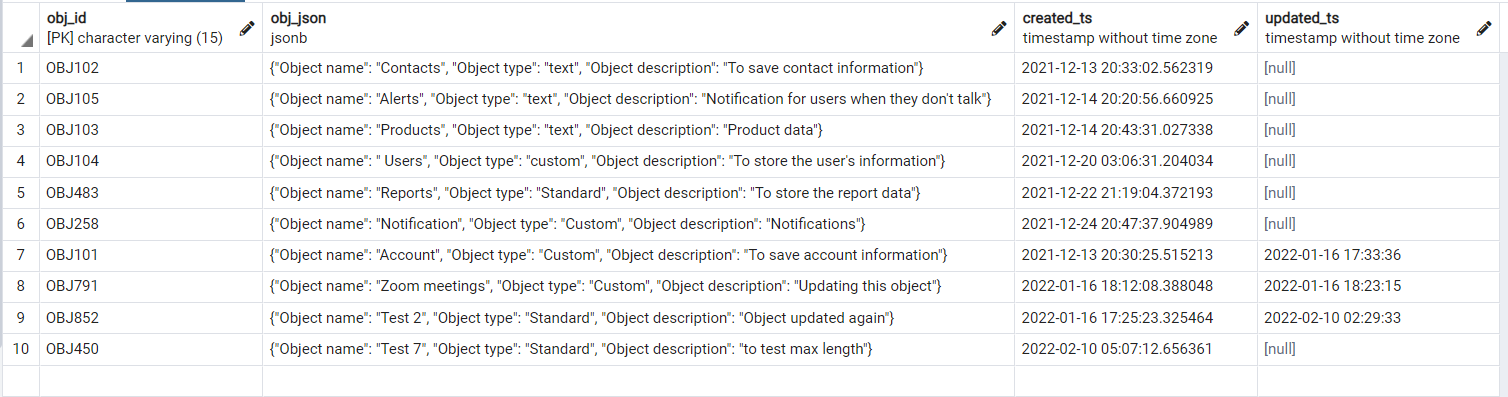
Database schema of the tables:

1. Object\_info

Consists of 4 columns:

* obj\_id – which serves as the primary key to uniquely identify objects
* obj\_json – to store the JSON of the object’s properties or attributes restricted to “Object name”, “Object type” and “Object description” [caution: avoid spaces in object key and separate words with an underscore, correction needs to be made in specific places i.e “Object name” is to be changed to object\_name]
* created\_ts – stores the timestamp at which the object was created
* updated\_ts – records the time at which the object was updated if in case changes were made to the object. [change the format of the timestamp to be displayed]

|  |  |  |  |
| --- | --- | --- | --- |
| obj\_id | obj\_json | created\_ts | updated\_ts |

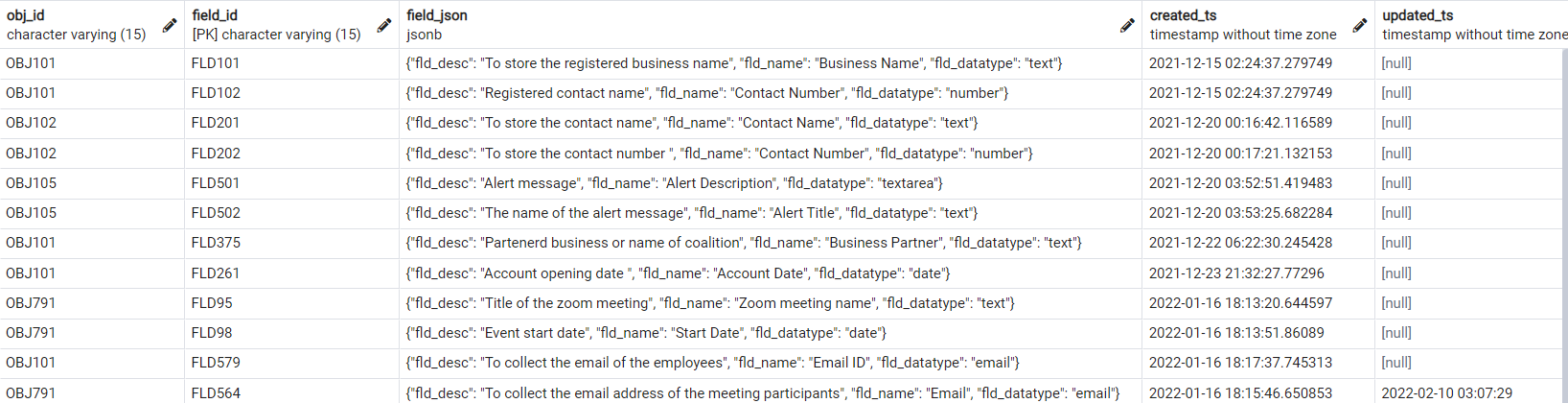


1. Field\_info

Consists of 5 columns

* obj\_id – is the foreign key of the object table
* field\_id – is the primary key which uniquely identifies the fields
* field\_json – stores the properties of the fields in JSON format
* created\_ts – stores the timestamp at which the object was created
* updated\_ts – records the time at which the object was updated if in case changes were made to the object. [change the format of the timestamp to be displayed]

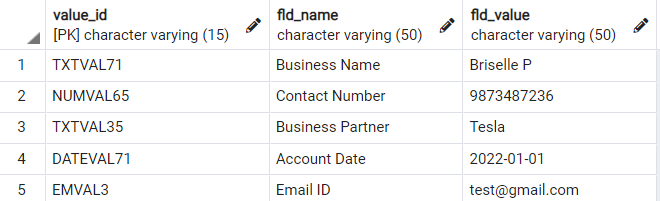
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| obj\_id | field\_id | field\_json | created\_ts | updated\_ts |



1. data\_values

Contains 3 columns

* value\_id – primary key to uniquely identify each value entered by the user
* fld\_name – name of the field for which the value is entered
* fld\_value – the value as submitted by the user



Creating the database: run the following commands to create the database and tables (backend -> database.sql)

(change the character limit as per standards i.e fld\_name VARCHAR(x) )

CREATE TABLE object\_info(

    obj\_id varchar(15) PRIMARY KEY,

    obj\_json jsonb,

    created\_ts timestamp default current\_timestamp,

    updated\_ts timestamp);

CREATE TABLE field\_info(

    obj\_id varchar(15),

    field\_id varchar(15) PRIMARY KEY,

    field\_json jsonb,

    created\_ts timestamp default current\_timestamp,

    updated\_ts timestamp,

    FOREIGN KEY(obj\_id)

      REFERENCES object\_info(obj\_id));

CREATE TABLE data\_values(value\_id VARCHAR(15) PRIMARY KEY,fld\_name VARCHAR(50),fld\_value VARCHAR(50));

**BACKEND**

The cors node package was installed as a middleware to allow cross origin resource sharing

The following routes were defined and implemented for the different types of functionalities.

POST methods:

1. /object\_data – to receive the object info from the frontend and save it in the database. The data is received in the body which needs to be parsed with the help of the ‘body-parser’ package
2. /field\_data - to receive the field info from the frontend and save it in the database.
3. /values – receives the values of the fields as entered by the user

GET methods:

1. /object\_data – sends the details of all the objects from the database to the front end
2. /object\_data/:id – sends the details of a particular object referenced by the obj\_id
3. /field\_data?id – send the details of all the fields corresponding to a particular object id which is submitted in the query string.
4. /values – sends the data of the values from the database to the front end.

PUT methods:

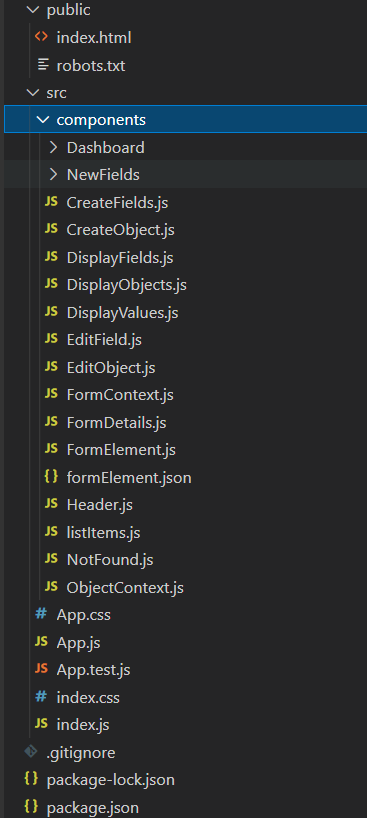
1. /object\_data/:id – updates the value of a particular object by taking in the id of the object and adds the updated timestamp in the database
2. /field\_data/:id - updates the value of a particular field by taking in the field\_id and adds the updated timestamp in the database

DELETE methods:

1. /object\_data/:id – deletes the object from the database whose object id is mentioned
2. /field\_data/:id – deletes the field from the database whose field id is mentioned
3. /values/:id – deletes the value from the database as per the given value id.

**FRONTEND**

Folder structure:



1. The **Dashboard** folder contains files related to the home page which follows the cube template with a graph and recent orders, the values of which are hardcoded at the moment.
2. The **NewFields** folder contains the components associated with the Input fields of various field types like Text field, Email field etc.

(The password should be encrypted before being saved in the database)

Each field receives the property info from the json like the max length property, label etc. (more properties can be further added)

The email field is validated for a valid email type according to the regex expression while all other fields are validated if value is entered.

The ‘handleChange’ function is provided from the formContext component to handle the change in value of the field and save the current value in the json which then sent to the backend on submit.

(The field DropDown displays a hardcoded dropdown menu, it needs to accept the options from the user, CheckBox and RadioButton needs to be worked on )

1. The **CreateObject.js** renders a form to get the data associated to create a new object. The object\_id is generated automatically (use uuid to generate a unique guid for each field ) and this field is frozen. On submitting the form, the post method containing the form details is sent to the backend. The ‘Create Field’ button enables the user to create a new field. On successfully creating an object, a toast is displayed
2. The **CreateField.js** renders a form to create a new field with the data relating to field properties, as submitted by the user. A new field is created corresponding to a particular object, therefore the object\_id of the object must first be selected from the dropdown or entered in the field. The field\_id is automatically generated (use uuid to generate a unique guid for each field ) and this field to is frozen.
3. **DisplayObjects.js** displays the list of objects received from the backend and renders it in the form of a table with all related information. The object\_id is an anchor element which on clicking will display all the fields corresponding to that object\_id. Each object has a set of associate actions to edit or delete the particular object. Clicking on the ‘Create Object’ allows the user to create a new object.
4. **DisplayField.js** displays all the fields corresponding to a particular object by passing the ‘object\_id’ as a parameter in the query string of the ‘GET’ method. Each field has a set of associated actions to edit or delete the particular field.
5. Similarly, The **DisplayValues.js** displays all the values entered by users, in the form of a table.
6. **EditField.js** and **EditObject.js** allows the user to edit the field or object respectively by means of the put request.
7. **FormContext.js** makes the data and functions accessible within the Provider to all the components in the component hierarchy. This is used specially to make the functionalities associated of the new fields common.
8. **FormDetails.js** displays the form for accepting field values. The fields are dynamically rendered as per the field properties in the json. It has the function for handling the values as entered by the user and saving it in form of json and sending it to the backend on submitting the form.
9. **FormElement.js** contains the switch statement and sends the properties to the particular component with respect to the field type mentioned in the json data.
10. **Navbar.js** lists the menu items and sets the router to the particular component on clicking the nav-item.
11. **NotFound.js** redirects the user to the ‘404 error’ page, if they try to go to a non-existing route.
12. **App.js** specifies the routes for each of the components like CreateObject, DisplayObject etc.
13. **Index.js** is the main file of the project which is rendered onto the screen.