Sri Lanka Institute Information Technology

Distributed System

Assignment 2

Department of Software Engineering, Sri Lanka Institute of Information Technology, Sri Lanka.

Table of Contents

I. ABSTRACT

Within this effort, we have tried to develop some kind of collaborative shopping platform for medicines by fulfilling a number of requirements from the business analysis of certain companies by following the principles of service-oriented architecture. It has facilitated the normal users(buyers) and the administrative party for engaging their works which are given by different web services from the RESTful web service platform via the same web API respectively. Here the web services use common interface standards and an architectural pattern so they can be rapidly incorporated into new applications. To implement that task, it has been used PHP with Laravel framework for the backend in order to implement the asynchronous web client, and connecting to the MYSQL databases and also for the frontend Bootstrap, react.js, CSS, HTML, JavaScript technologies were used. Also, we implemented a common(one) web API using WSO2 Enterprise Integrator and you can view the integration process and the architecture of the program clearly from steps and the diagrams within this respective report for the project. Within this effort we have successfully created that application after the clarifications at several stages on the requirements, services and the integration.

Index Terms - Internet of Things (IoT), Enterprise Integration (EI), Representational State Transfer Application Programming Interface (REST API), Application Programming Interface (API)

- WSO2 - In contrast to WSO2 ESB, WSO2 BPS is designed for stateful and long-running business processes. WSO2 Message Broker (WSO2 MB) enables applications to exchange communications asynchronously or publish messages for timely access by many subscribers. WSO2 MB supports JMS and AMQP standards for persistence messaging and can be used in scenarios where guaranteed message delivery is vital. [01]

- REST API - REST API stands for Representational State Transfer Application Programming Interface. It means when a RESTful API is called, the server will transfer a representation of the requested resource's state to the client system. [02]

II. INTRODUCTION

Within this effort, we mainly focused on taking the real use of one of the modern and well-organized approaches of Distributed Systems. That is none other than RESTful web services. Within this implementation several kinds of technologies, principles, and environments were used for implementing the APIs to fulfil the requirements.

Within this suggested medical system, we are suggesting to fulfil different kinds of requirements for this specific application which was recognized by analysing the business requirements up to the mark. From this solution for the medical system, there are two different kinds of user groups who are able to work with the services of the system. Those are “Customers” and the “Administrator” Parties.

We have suggested maintaining particular administrators for maintaining the drugs of the system as the responsible person. He or she will initially login to the system and maintain the drugs of this medicinal application. We had planned a proper validated login for him\her as the very first interface. He\She will be able to edit his\her account details and login credentials under certain restrictions and validations of the system as he\she is the only administrator of the application. But he\she will not be able to delete his\her account completely because it violates the system performance.

After a successful login he\she will be able to add the drugs for the system with several details regarding that particular drug like the name of the drug, code of the drug, price of the drug, cart ID of the drug, description or the precautions and so on. After the initial addition he\she will be able to view those drugs and he\she can edit, update or delete the added items under the initial system restrictions to keep the accuracy. The uploaded details will be listed out at his\her interface so as to make decisions on them.

We have implemented our service for the second user group or the customers by facilitating them with several comforts. From the initial page they will be able to login by entering the initial details of their account. If they have no account, they can create one and login to the system as well. Also, they will be able to delete the account as their needs then they are able to make their initial deal with the drugs. They will be able to search the drug that they need to buy by typing the name, code or some detail on the drug, otherwise they can type their disability and see the suggestions as well.

Then, they will be able to add them to their shopping cart. The cart at the top right corner will show you how many items did they add for making this deal. Also, there is a facility to delete or update the added items before purchasing the cart. Once they have fixed those items in the cart and if they are ok with that they can simply click purchase and make a move to the delivery service. They can select the delivery option at that time. According to the selection that they wish to make they can have a delivery service from that.

The payment can be done by using two methods, the first one is planned to be done by using credentials of the credit card. We have connected it to the payment gateway (Model is using for this project up to now) and it may take necessary details of the user such as credit card number, CVC number which is special three-digit number at the back of the credit card for further clarifications, card holders name along with the total amount of the items that he\she has added to the card for this transaction.

The other payment method will be done by using the mobile number of the customer and it will respectively be added to the mobile bill of that particular user. It will need to receive the details of mobile phone number, four-digit pin number along with the amount for mobile company service respectively at that process.

After having a successful service, the system will generate system email or SMS to conform the purchasing for that particular user.

In here we have added those used services that we have implemented successfully at the needy stages as we have mentioned above so as to make a complete service package for the customers as well as the administrators within the system.

The separate services that have been implemented and the architecture, interconnectivity of the services and the integration will be clearly discussed and shown within the content of this project followed report respectively.

III. BACKGROUND

For implementing the above-mentioned requirements, we have used a project with three separate services which have been empowered to do certain tasks as “Payment service” “Delivery Service”, “Customer Service” Within this report we have clearly discussed the tasks and the connections with the system. For implementing those services PHP with Laravel framework have been used with latest updates. As the IDE used for the implementation is Visual Studio Code.

For storing the data of the services such as details of Medicines, Login credentials of administrator and the customers, we had to use MYSQL as the database solution along with SQL languages to connect with that. The frontend was developed by using React.js with its necessary libraries and also the normal web implementing languages such as HTML, CSS, JavaScript. The database connection is specially done by the configuration file called “api.php” at the PHP backend.

IV. STEPS OF CREATION

There are two files:

- Frontend and backend build by ReactJS (VS code)
- Database handled by:
  - ✓ Laravel
  - ✓ WSO2

1. create database and implement API's

For the manipulation of data using Laravel programming. We have three tables to be made under mcart schema that will acts as the database for mcart services.

- User information – (*Implemented using Laravel*)

```
    public function up()
    { Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string("first_name")->nullable();
            $table->string("last_name")->nullable();
            $table->string("full_name")->nullable();
            $table->string('email')->unique();
            $table->string('password');
            $table->string("aType")->nullable();
            $table->rememberToken();
            $table->timestamps();
        });
    public function down()
    {
        Schema::dropIfExists('users');
    }
```

- Product information – (*Implemented using Laravel*)

```
    public function up()
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->string("pro_name")->nullable();
            $table->string("pro_code")->nullable();
            $table->string("cat_id")->nullable();
            $table->string('pro_info')->nullable();
            $table->string('pro_price')->nullable();
```

```
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('products');
    }

```

- Payment information – (*Implemented using WSO2*)

The full middleware component that we have configured and worked on so far is attached in the ZIP file along with the front-end and back-end. However, the work is still in progress at the time of submitting this document

2. Calling Route to make execution between database and site.

The database API's uses **http://127.0.0.1:8000** as the route path, while the site uses **http://localhost:3000/** ., so we use the route to connect the API with the site.

```
//for user and admin login
Route::post("user-signup", [UserController::class, 'userSignUp']);
Route::post("user-login", [UserController::class, 'userLogin']);
Route::get("user/{email}", [UserController::class, 'userDetail']);
Route::delete("user/{id}", [UserController::class, 'deleteuser']);
//for cart
Route::post("additems", [ProductController::class, 'store']);
Route::get("products", [ProductController::class, 'index']);
Route::get("products/{id}/show", [ProductController::class, 'show']);
Route::put("products/{id}/edit", [ProductController::class, 'update']);
Route::delete("products/{id}", [ProductController::class, 'destroy']);
//search nav
Route::get("frontend", [ProductController::class, 'frontend']);
Route::get("backend", [ProductController::class, 'backend']);
//cart
Route::get("cartview/{id}", [ProductController::class, 'show']);

//payment api will be made using wso2
```

3. Create the frontend using ReactJS.

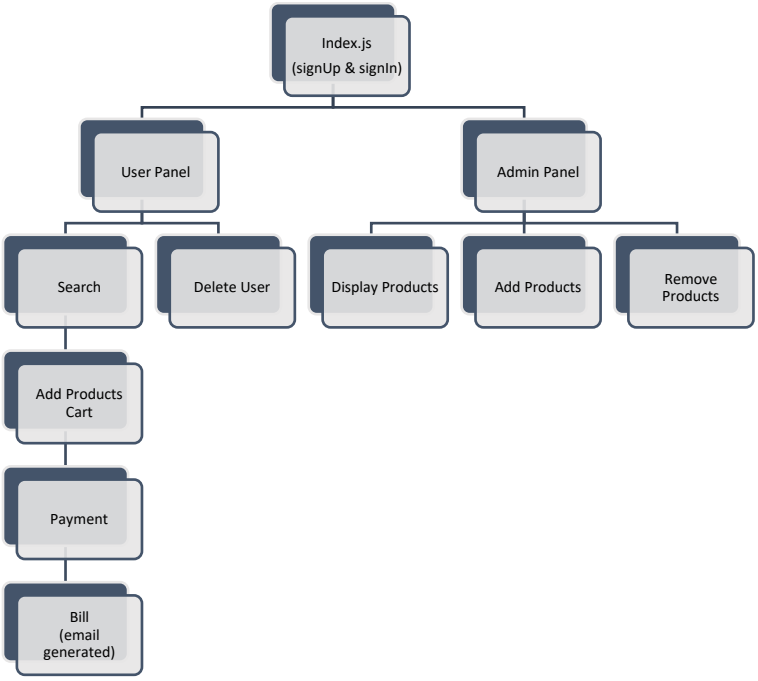


Fig: DOM view of the front end

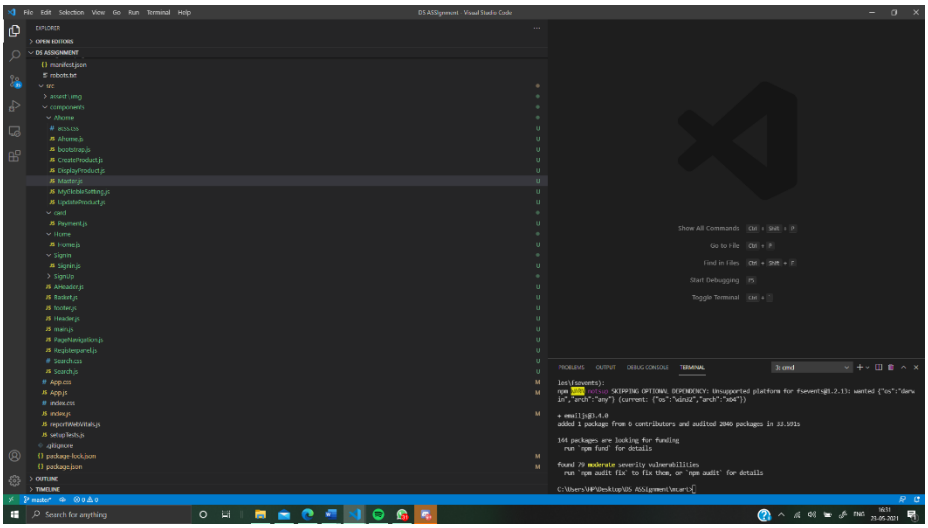


Fig: Application DOM view

4. connecting the frontend and the API by axios method.

GET method called(to get data from the user and insert to the database)

```
axios.get(MyGlobleSetting.url + '/api/products')
```

Post method called(to retrieve the data from the database)

```
axios.post("http://localhost:8000/api/user-login"
```

Put method called(to update the data from the database by given user)

```
axios.put(MyGlobleSetting.url + `/api/products${this.props.location.pathname}
```

5. manipulation of the data and finding the total items value.

```
const value = this.props.location.state.valuesss;
console.log(value);

const Total = value.reduce((totalvalue, value) => totalvalue + parseInt(value[2], 10), 0);
console.log(Total);
```

value stores the data from the value passed from the parent components to the child components know as *valuesss*.
Total stores the total value of the given element of the array in integer format.

V. DISCUSSION ON THE SERVICES

SERVICE INTERFACES

<< Customer Service >>

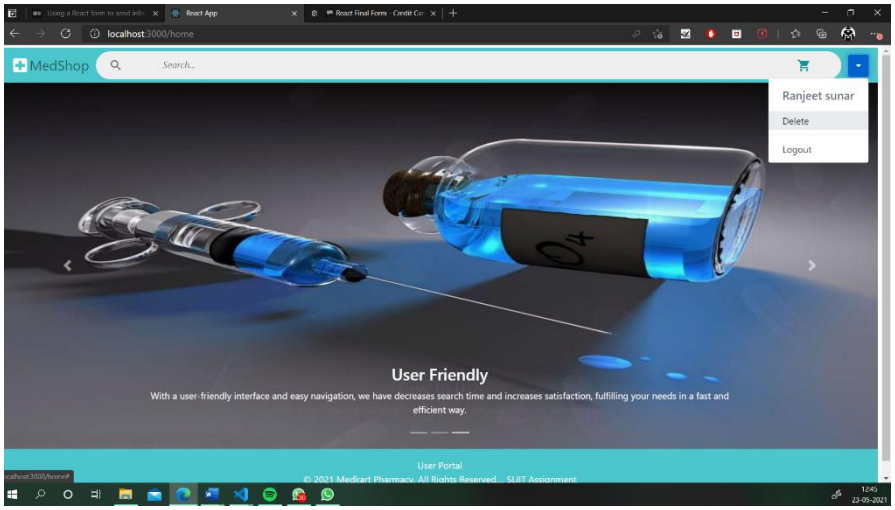
It’s the service where the user/ customer is having the functionalities to delete and view after they have been registered

```
class UserController extends Controller

{   private $status_code    =       200;
   // -------------------[ deleteing user ] -----------------
   public function deleteuser($id)
   {
    $user = User::find($id);
    $user->delete();
    return response()->json('User Deleted Successfully.');
   }
}
```

<<-trigger this end point->>

```
Route::delete("user/{id}", [UserController::class, 'deleteuser']);
```

<<-Working Diagram of delete function->>



<< Delivery Service >>

It’s the service where the user/ customer have the functionalities to know the cart products they bought by giving the user/ customer a receipt via email

```
submit = (e) => {
    e.preventDefault();
    //console.log(creditCardData);
    //method to be added
    alert(`payment successful`);
    const templateId = 'template_xp5p19d';
    const serviceID = 'service_3709jtd';
    emailjs.send(serviceID,templateId, 'serviceID', 'tandinwangchen272gmail.co
m')
    .then((response) => {
       console.log('SUCCESS!', response.status, response.text);
    }, (err) => {
       console.log('FAILED...', err);
    });

 this.props.history.push('/home');
}
```

This function uses the *EmailJS* submit() function that includes the parameters that are set in email template — sender email, user email and the message .

<<Working Diagram>>

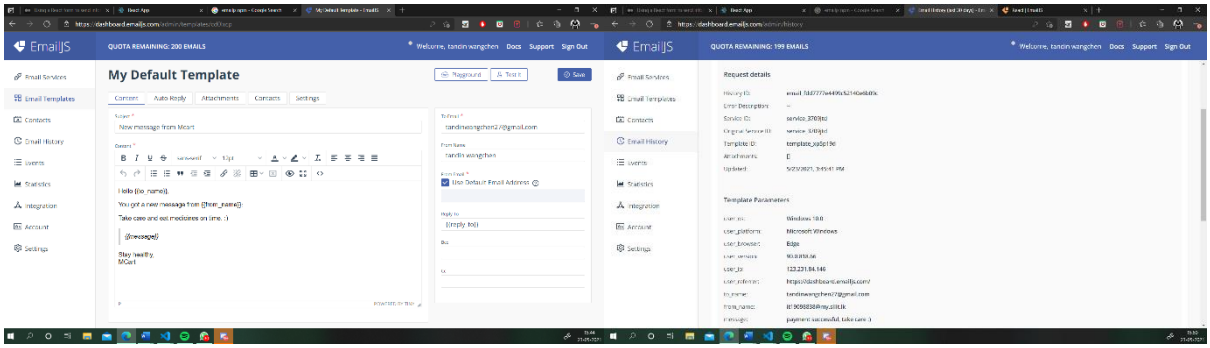


Fig: 3rd party Template build

Fig: showing successful transaction

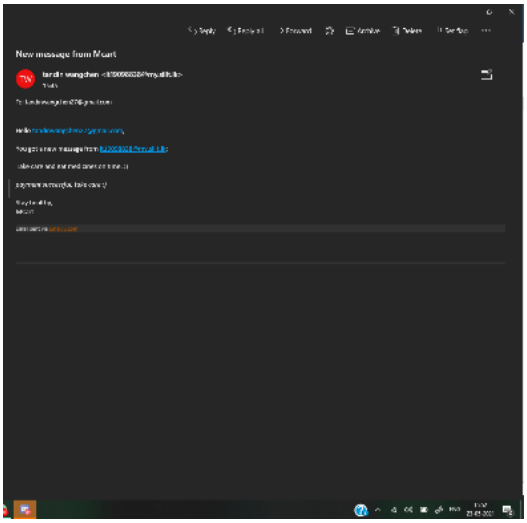


Fig: showing real world email receipt

<< Product service >>

It's the service where the admin has the functionalities to review/ update/ and delete the products he has added to the system

```
public function index(){
 public function show($id)
   {
      $product = Product::find($id);
      return response()->json($product);
   }
   public function update(Request $request, $id)
   {
      $product = Product::find($id);
      $product->pro_name = $request->get('pro_name');
      $product->pro_code = $request->get('pro_code');
      $product->cat_id = $request->get('cat_id');
      $product->pro_info = $request->get('pro_info');
      $product->pro_price = $request->get('pro_price');
      $product->save();

      return response()->json([
         'ty' => $request->get('pro_name'),
         'message' => 'update succesfully',
         'product' => $product,
      ]);
```

```
  }

  public function destroy($id)
  {
    $product = Product::find($id);
    $product->delete();
    return response()->json('Product Deleted Successfully.');
  }
```

<<trigger this end point>>

```
Route::post("additems", [ProductController::class, 'store']);
Route::get("products/{id}/show", [ProductController::class, 'show']);
Route::put("products/{id}/edit", [ProductController::class, 'update']);
Route::delete("products/{id}", [ProductController::class, 'destroy']);
```

<<Working Diagram>>

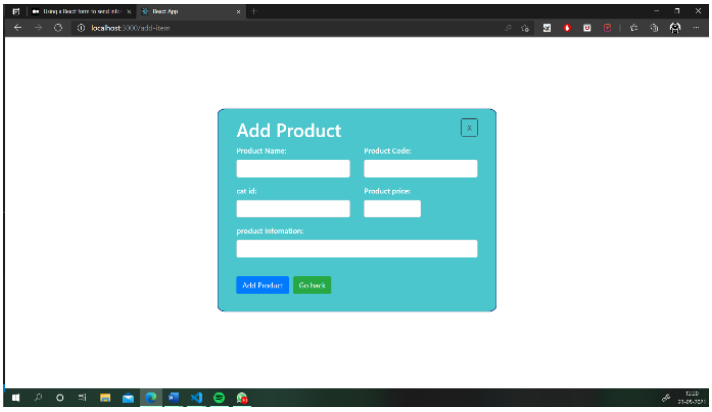


Fig: Post method to add products

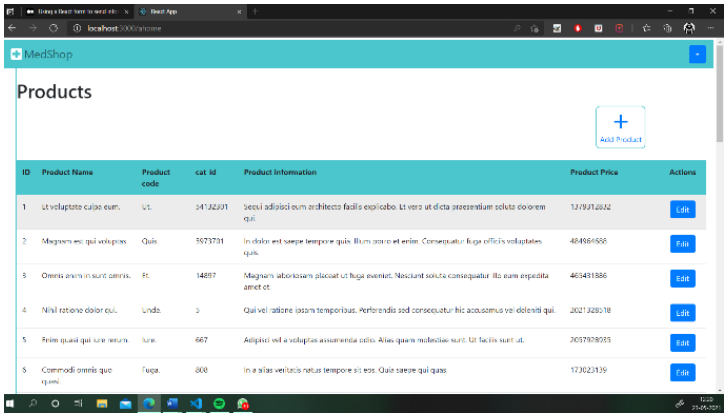


Fig: Get method to show the products

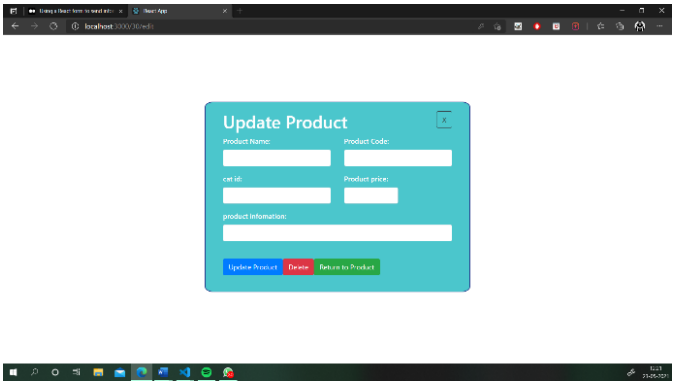


Fig: Put method to update products/Delete method to delete product

<< Payment service >>

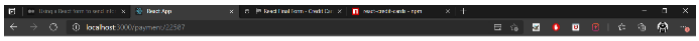
It's the service where the users card information is verified and directs to the purchase of the products

```
import React from 'react';
import Cards from 'react-credit-cards';
import { Link  } from 'react-router-dom';
import 'react-credit-cards/es/styles-compiled.css';
import '../Ahome/acss.css'
export default class Payment extends React.Component {

 state = {
  cvc: '',
  expiry: '',
  focus: '',
  name: '',
  number: '',
 };

 handleInputFocus = (e) => {
  this.setState({ focus: e.target.name });
 }

 handleInputChange = (e) => {
  const { name, value } = e.target;

  this.setState({ [name]: value });
 }
```

```
render() {
  const total = this.props.match.params.V;
  return (
    <div className="prodview">
      <Link to="/home" className="btn btn-outline-dark" style={{float:'right'}}>X</Link>
      <Cards
        cvc={this.state.cvc}
        expiry={this.state.expiry}
        focused={this.state.focus}
        name={this.state.name}
        number={this.state.number}
      />
      <form className="cardv">
        <input
        type="tel"
        name="number"
        placeholder="Card Number"
        onChange={this.handleInputChange}
        onFocus={this.handleInputFocus}
        />

        <input
        type="tel"
        name="name"
        placeholder="Name"
        onChange={this.handleInputChange}
        onFocus={this.handleInputFocus}
        />
        <input
        type="tel"
        name="expiry"
        placeholder="expiry"
        onChange={this.handleInputChange}
        onFocus={this.handleInputFocus}
        />
        <input
        type="tel"
        name="cvc"
        placeholder="cvc"
        onChange={this.handleInputChange}
        onFocus={this.handleInputFocus}
        />
      <br /><br />
```

```
      Total: {total}<br />
      <input type="submit" value="pay" className="btn btn-success" />
      <input type="reset" value="reset" className="btn btn-danger"/>
      </form>
    </div>
  );
 }
}
```

<<Working Diagram>>



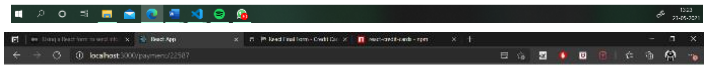


Fig: Preview of the card before entry

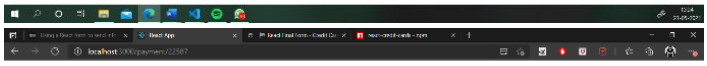


Fig: Preview of the card during the entry

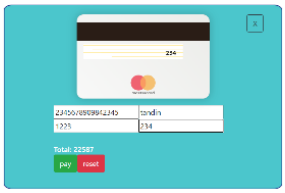


Fig: Preview of the card during ccv entry

<<necessary authentication and security mechanisms>>

During creation of login of the user

```
public function userSignUp(Request $request) {
        $validator = Validator::make($request->all(), [
          "name" => "required",
          "email" => "required|email",
          "password" => "required",
        ]);

        if($validator->fails()) {
          return response()->json(["status" => "failed", "message" => "validation_error", "errors" => $validator->errors()]);
        }

        $name = $request->name;
        $name = explode(" ", $name);
        $first_name = $name[0];
        $last_name = "";
        $aType = "User";


        if(isset($name[1])) {
          $last_name = $name[1];
        }

        $userDataArray = array(
          "first_name" => $first_name,
          "last_name" => $last_name,
          "full_name" => $request->name,
          "email" => $request->email,
          "aType" => $aType,
          "password" => md5($request->password),

        );

        $user_status = User::where("email", $request->email)->first();
```

```
•
•       if(!is_null($user_status)) {
•         return response()->json(["status" => "failed", "success" => false, "message" => "Whoops! email already registered"]);
•       }
•
•       $user       =       User::create($userDataArray);
•
•       if(!is_null($user)) {
•         return response()->json(["status" => $this->status_code, "success" => true, "message" => "Registration completed successfully", "data" => $user]);
•       }
•
•       else {
•         return response()->json(["status" => "failed", "success" => false, "message" => "failed to register"]);
•       }
•     }
```

During the Login

```
public function userLogin(Request $request) {

        $validator          =       Validator::make($request->all(),
            [
                "email"             =>          "required|email",
                "password"          =>          "required"
            ]
        );

        if($validator->fails()) {
            return response()->json(["status" => "failed", "validation_error" => $validator->errors()]);
        }


        // check if entered email exists in db
        $email_status       =       User::where("email", $request->email)->first();
```

```
        // if email exists then we will check password for the same email

        if(!is_null($email_status)) {
            $password_status    =   User::where("email", $request->email)-
>where("password", md5($request->password))->first();

            // if password is correct
            if(!is_null($password_status)) {
                $user           =       $this->userDetail($request->email);

                return response()->json(["status" => $this-
>status_code, "success" => true, "message" => "You have logged in successfully
", "data" => $user]);
            }

            else {
                return response()-
>json(["status" => "failed", "success" => false, "message" => "Unable to login
. Incorrect password."]);
            }
        }

        else {
            return response()-
>json(["status" => "failed", "success" => false, "message" => "Unable to login
. Email doesn't exist."]);
        }
    }
```

During the Route protect

```
  onChangehandler = (e, key) => {
    const { signupData } = this.state;
    signupData[e.target.name] = e.target.value;
    this.setState({ signupData });
  };
  onSubmitHandler = (e) => {
    e.preventDefault();
    this.setState({ isLoading: true });
```

```
  axios
    .post("http://localhost:8000/api/user-signup", this.state.signupData)
    .then((response) => {
      this.setState({ isLoading: false });
      if (response.data.status === 200) {
        this.setState({
          msg: response.data.message,
          signupData: {
            name: "",
            email: "",
            password: "",
          },
        });
        setTimeout(() => {
          this.setState({ msg: "" });
        }, 2000);
      }

      if (response.data.status === "failed") {
        this.setState({ msg: response.data.message });
        setTimeout(() => {
          this.setState({ msg: "" });
        }, 2000);
      }
    });
};
```

VI. DISCUSSION ON THE INTEGRATION

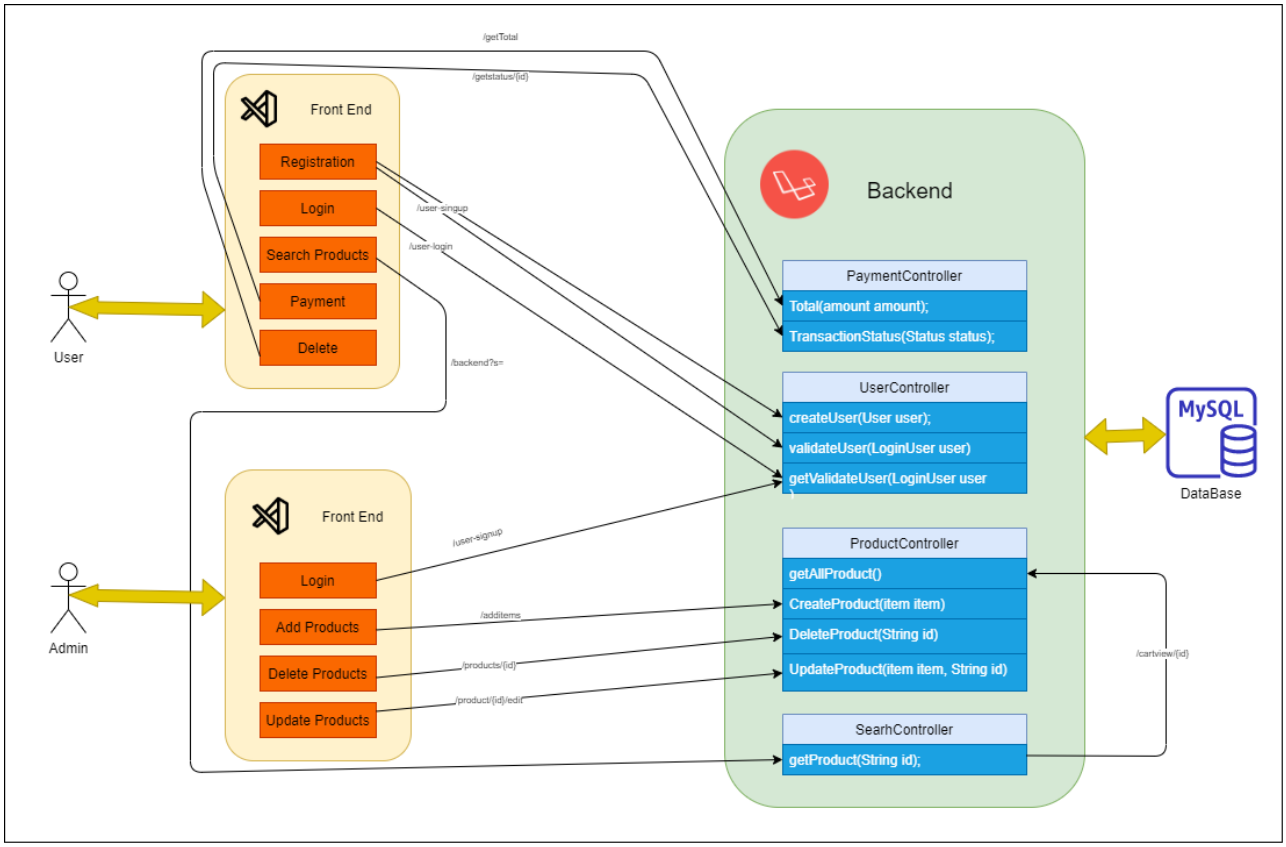


Fig. Hi-Level Architectural Diagram of the Interconnectivity of the System

This diagram shows the relationships between the front-end API and how the endpoints are calling the backend method. Each controller class shows in light blue colour and the backend methods show in dark blue colour. Front-end UI components are shown by square shapes. These are the modules that clients will access.

VII. EXTRA

Search System

When the user searches some medicine, the user can either use the product name or the description or the use of the medicine that gives some product there in the system. If description product does not exist then the user will get a message to search other products.

```
class Search extends  React.Component {
  constructor( props ) {
    super( props );
    this.state = {
      //search and pagination
      query: '',
      results: {},
      loading: false,
      message: '',
      totalResults: 0,
      totalPages: 0,
      currentPageNo: 0,
      //cartitmes
      count : 0,
      proID : '',
      cartItem : []

    };
    //implementing axios for serach sumbit entries
    this.cancel = '';

  }

  //calculated no of pages in db for search
  getPageCount = (total, denominator) => {
    const divisible = 0 === total % denominator;
    const valueToBeAdded = divisible ? 0 : 1;
    return Math.floor( total/denominator) + valueToBeAdded;
  };


  //handles event listner
  handleOnInputChange = (event) => {
```

```
    const query = event.target.value;
    if ( ! query ) {
        this.setState({ query, results: {}, message: '', totalPages:0, totalResults:0 } );
    } else {
         this.setState({query, loading: true, message: ''  }, () => {
              this.fetchSearchResults(1, query);
            });
        }
};

fetchSearchResults = (updatedPageNo = '', query) => {
    const pageNumber = updatedPageNo ? `&page=${updatedPageNo}` : '';
    const searchUrl = `http://127.0.0.1:8000/api/backend?s=${query}&${pageNumber}`;


    if (this.cancel){
        this.cancel.cancel();
    }

    this.cancel = axios.CancelToken.source();

    axios.get(searchUrl,{
        cancelToken: this.cancel.token,
    })
        .then (res => {

            const total = res.data.total;
            const totalPagesCount = this.getPageCount(total, 5);
            const resultNotFoundMsg = ! res.data.hits.length ?
                                  'There are no more search results. Please try a new search' :
                                  '';
            this.setState({
                results: res.data.hits,
                message: resultNotFoundMsg,
                totalResults: total,
                totalPages: totalPagesCount,
                currentPageNo: updatedPageNo,
                loading: false
            });

        }).catch ((error)  => {
            if(axios.isCancel(error) || error ){
                this.setState({
```

```
                loading: false,
                message: 'failed to fetch the data. pls check network'
              });
          }

      });
};

handlePageClick = (type) => {

   const updatePageNo = 'prev' === type ? this.state.currentPageNo - 1 :
                                this.state.currentPageNo + 1;

   if(! this.state.loading){
      this.setState({loading: true, message: ''}, () => {
         this.fetchSearchResults(updatePageNo, this.state.query);
      });
   }
};

//adding to cart

 addNewItem = (input) => {
    console.log("==========");
    console.log(input);
    if(input !== ""){
     console.log("pushing values");
     //due to thiss...
     //this.setState({cartItem : [...this.state.cartItem, input]});

    console.log(this.state.cartItem);
    }
 };


renderSearchResults = () => {
   const {results} = this.state;
   // rtiis iscalled multiple times
   this.addNewItem(this.state.proID);

   if (Object.keys(results).length && results.length) {

      return (
```

```
          <div className="results-container">

                    <div className="uhh1">
                      Product Name
                    </div>

                    <div className="uhh2">
                      Price
                    </div>
              {results.map((result) => {
                return (
                  <ul key={result.id} className="result-items">


                    <li className="username">
                      <div className="u1">
                        {result.pro_name}
                      </div>

                      <div className="u2">
                        LKR {result.pro_price}
                      </div>

                      <div className="u3 mr-5">
                        <ButtonGroup>
                          <Addbutton
                            className="addbtn"
                            aria-label="add"
                            onClick={() => this.setState({
                              count: this.state.count + 1,
                              proID: result.id,
                              cartItem : [...this.state.cartItem, [result.id, result.pro_name, res
ult.pro_price ]]
                            })}
                          >
                            Add
                          </Addbutton>
                        </ButtonGroup>
                      </div>
                    </li>
                  </ul>
                );
              })}
```

```
        </div>
      );
    }
};

render() {
    const {query, loading, message, currentPageNo, totalPages} = this.state;

    const showPrevLink = 1 < currentPageNo;
    const showNextLink = totalPages > currentPageNo;



    return (
      <div className="container">
        {/*Search Input*/}
        <div className="ocontainer">
        <label className="search-label" htmlFor="search-input">
          <input
            type="text"
            name="query"
            value={query}
            id="search-input"
            placeholder="Search..."
            onChange={this.handleOnInputChange}
          />

        </label>
        </div>

        <div className="scontainer" >
          {/*Error Message*/}
            { message && <p className="message">{message}</p> }


          <PageNavigation
          loading={loading}
          showPrevLink={showPrevLink}
          showNextLink={showNextLink}
          handlePrevClick={() => this.handlePageClick('prev')}
          handleNextClick={() => this.handlePageClick('next')}
          />
```

```
                {/*Loader*/}
                    <img  src={Loader} className={`search-
loading ${loading ? 'show' : 'hide' }`}  alt="loader"/>

                {/* Search result */}
                    {this.renderSearchResults()}
                </div>

                <div className="bage-c">
                <Badge badgeContent={this.state.count} color="primary">
                    <Link to={{
                        pathname: "/cart-items",
                        state: {valuesss : this.state.cartItem}

                        }} className="cartss">
                    <ShoppingCartIcon />
                    </Link>

                </Badge>
                </div>
            </div>

            )
    }
}
export default Search;
```

Front end and API calling for search feature

```
Route::get("backend", [ProductController::class, 'backend']);
```

Routing of the API for search feature

```
public function backend(Request $request){
        //filter for search
        $query = Product::query();

        if($s = $request -> input('s')){
            //where we can use our own condition
            $query -> whereRaw("pro_name LIKE '%" . $s . "%'")
                -> orWhereRaw("pro_info LIKE '%" . $s . "%'");
        }
        //displaying no. of items per page
        $perPage = 5;
        $page = $request -> input ('page', 1);
```

```
    $total = $query -> count();

    $result = $query -> offset(($page -1) * $perPage) -> Limit($perPage) -> get();


    return [
        'hits' => $result,
        'total' => $total,
        'page' => $page,
        'last_page' => ceil($total / $perPage)
    ];
}
```

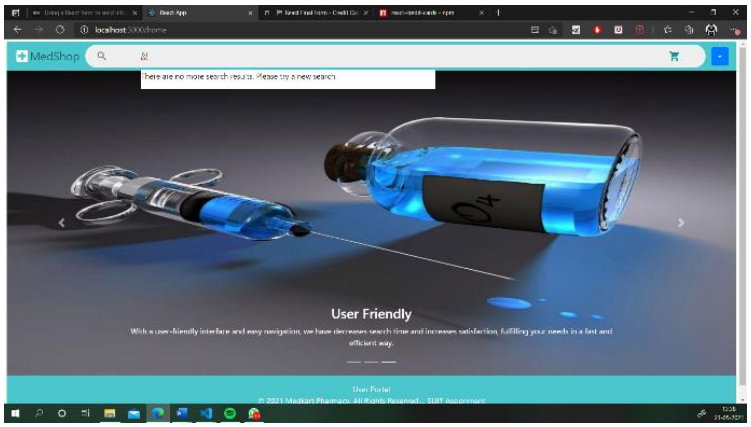


Fig: Preview of the search navigation when no data found

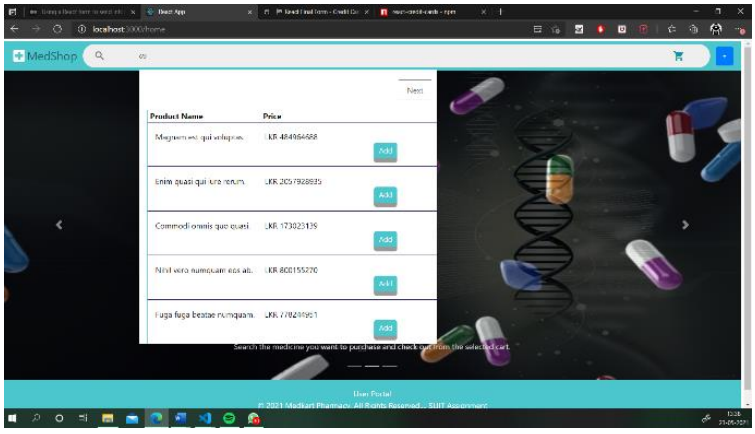


Fig: Preview of the search navigation when data if there in database

VIII. An overview of System

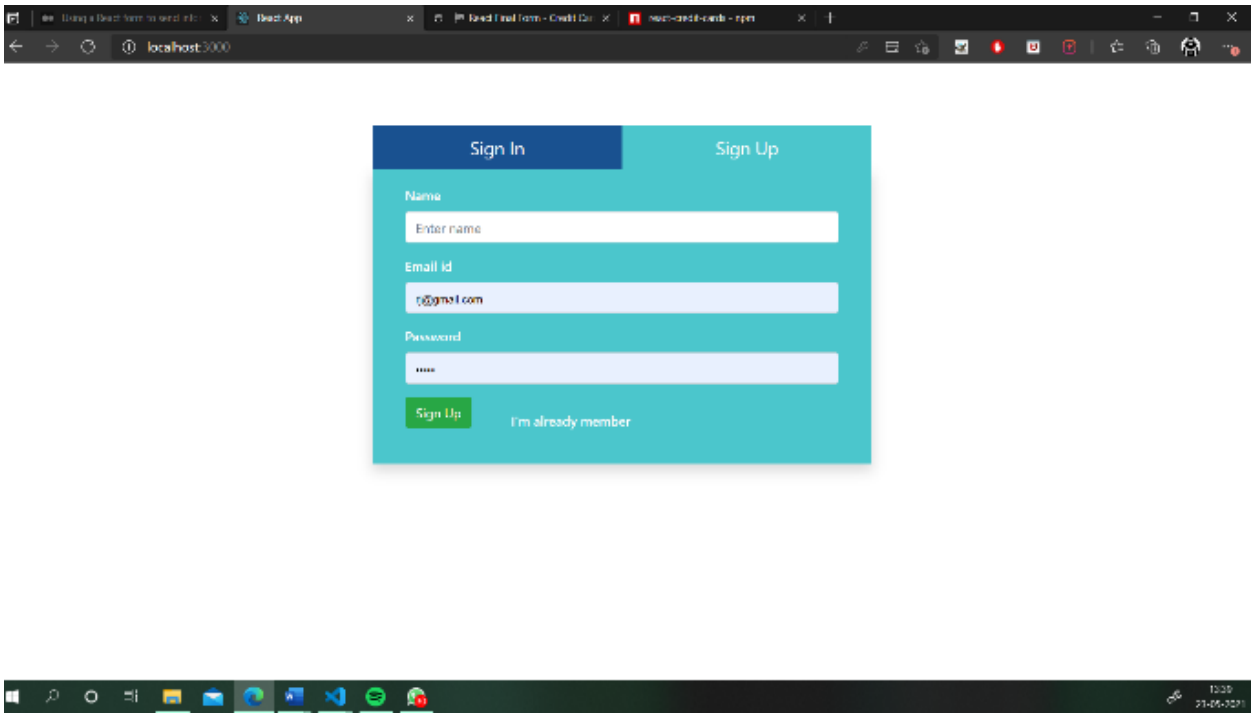


Fig: Preview of the SignUp page

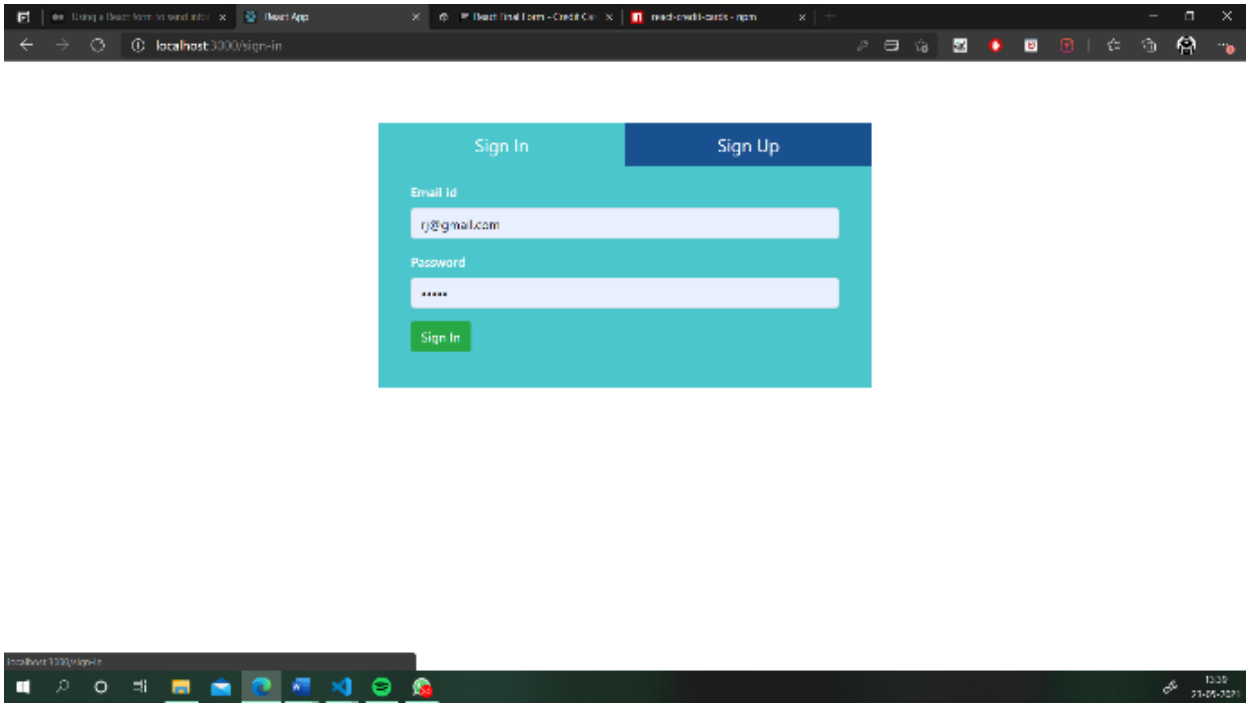


Fig: Preview of the SignIn page

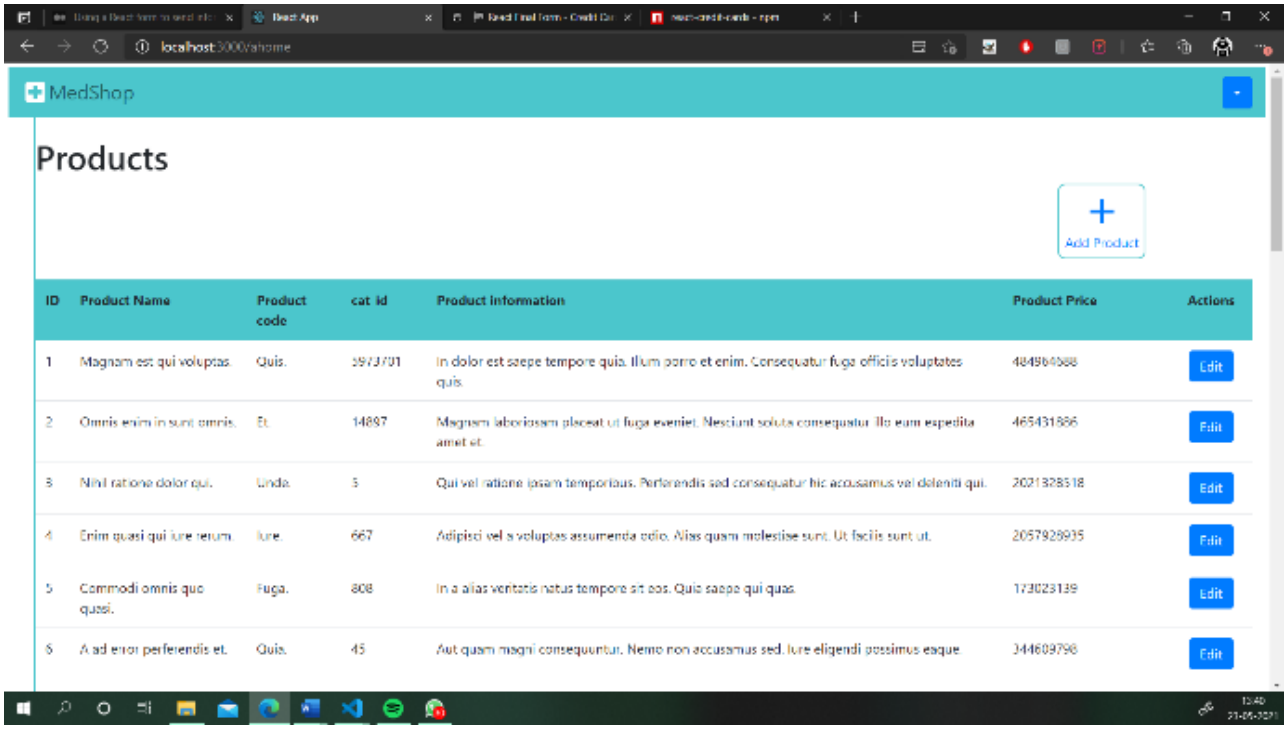
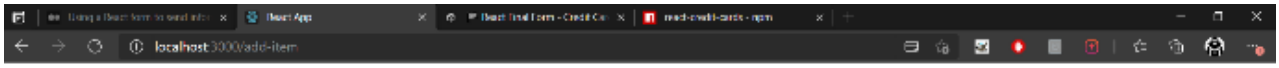


Fig: Preview when admin login > amin home page
(he can edit and add in the page)



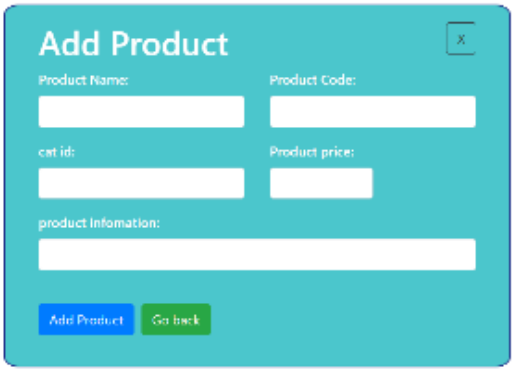


Fig: Preview of add product where the admin can add the product to the db

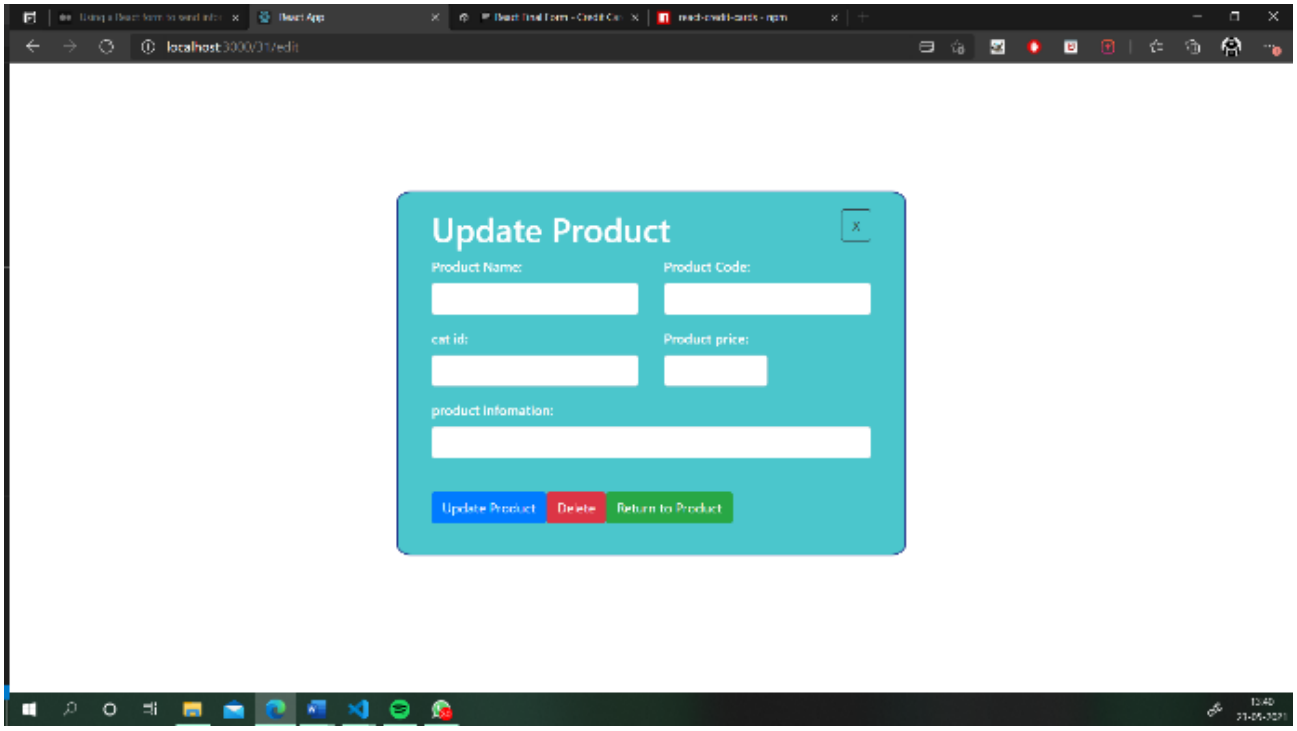


Fig: Preview of update product where the admin can Update/delete the product from the db

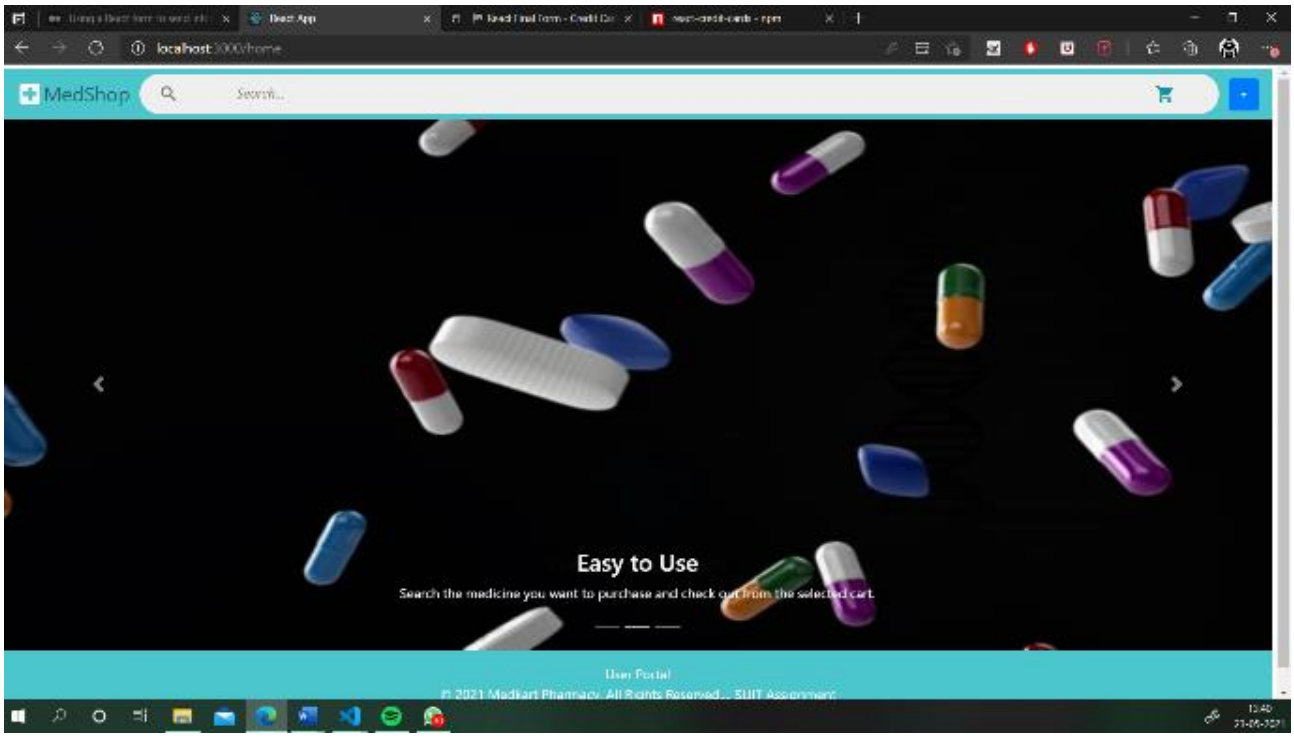


Fig: Preview when user login > user home page
(he can delete his account and add search for products)

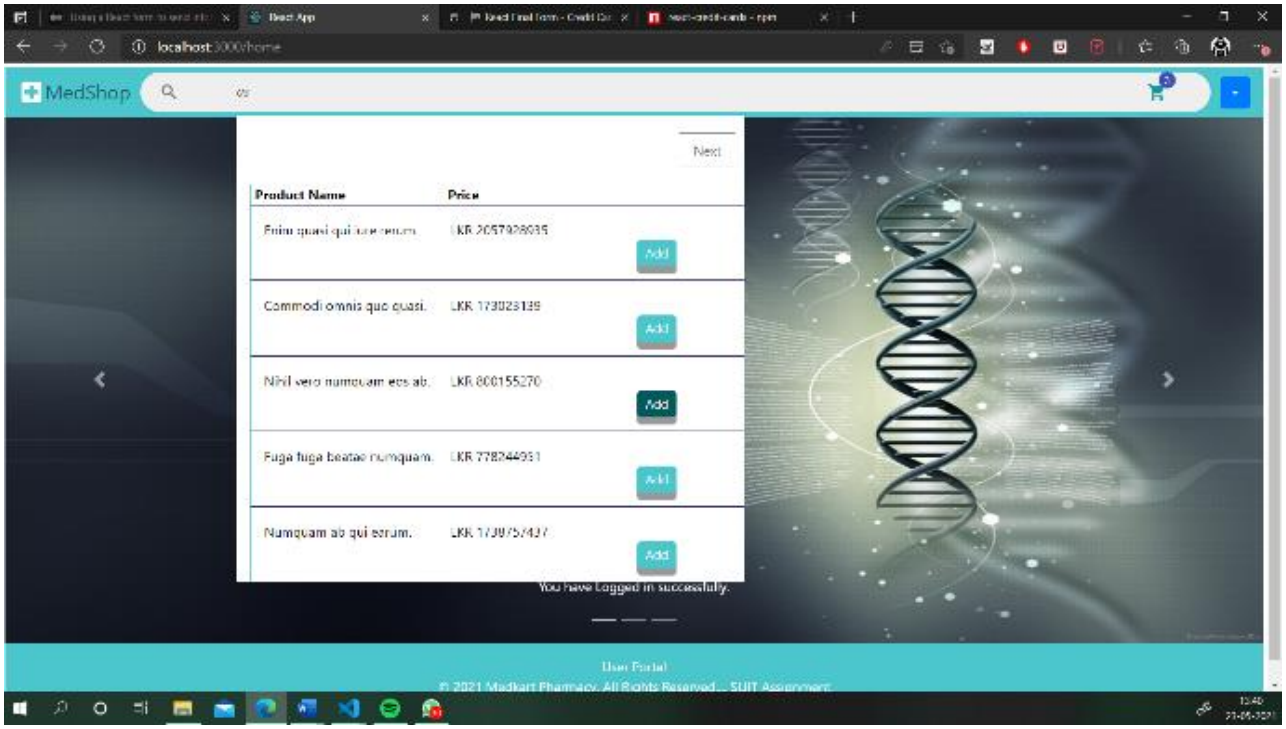


Fig: Preview of search bar, where the search items are listed
In the qty of 5 per page

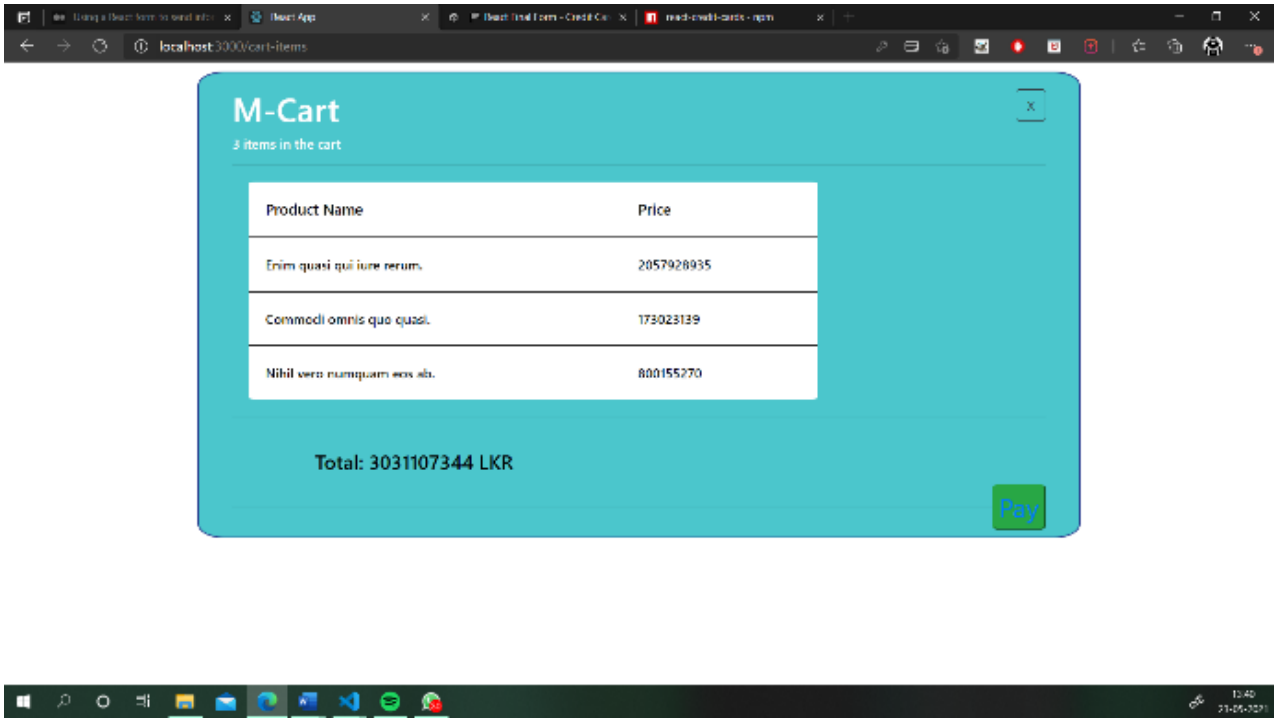


Fig: Cart view

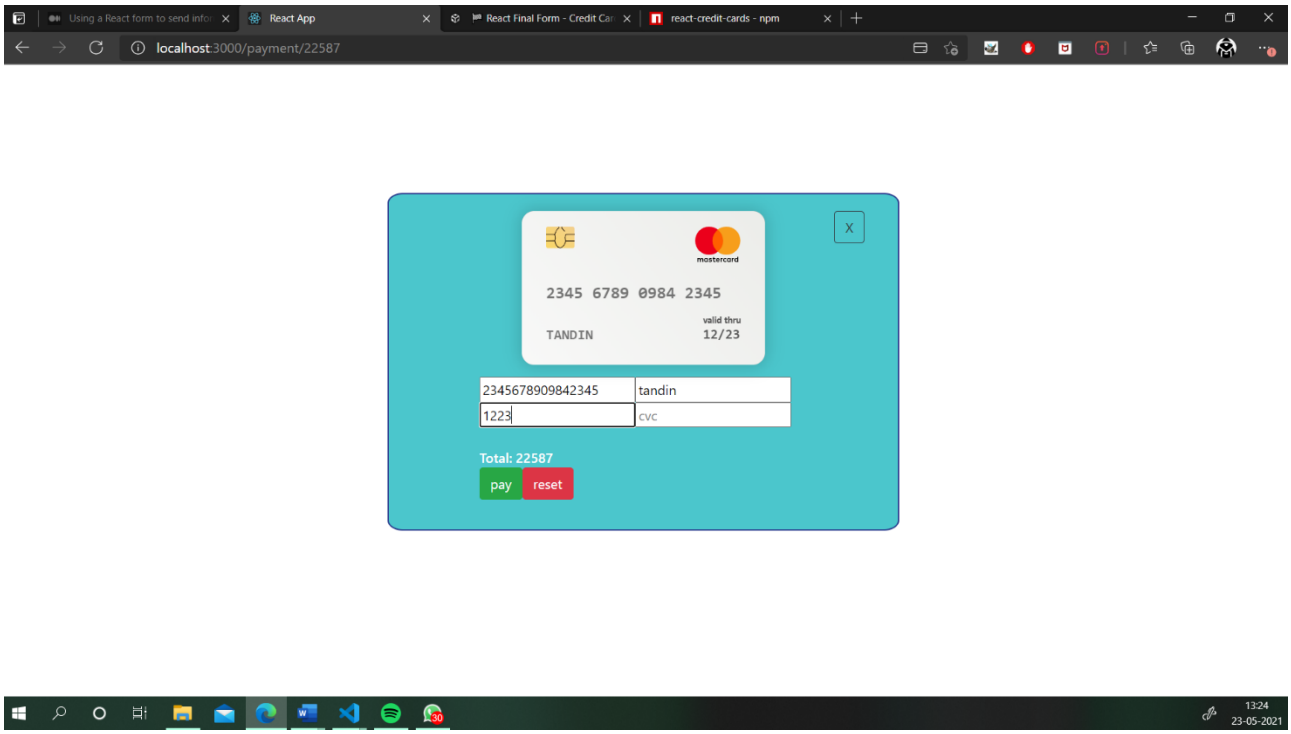
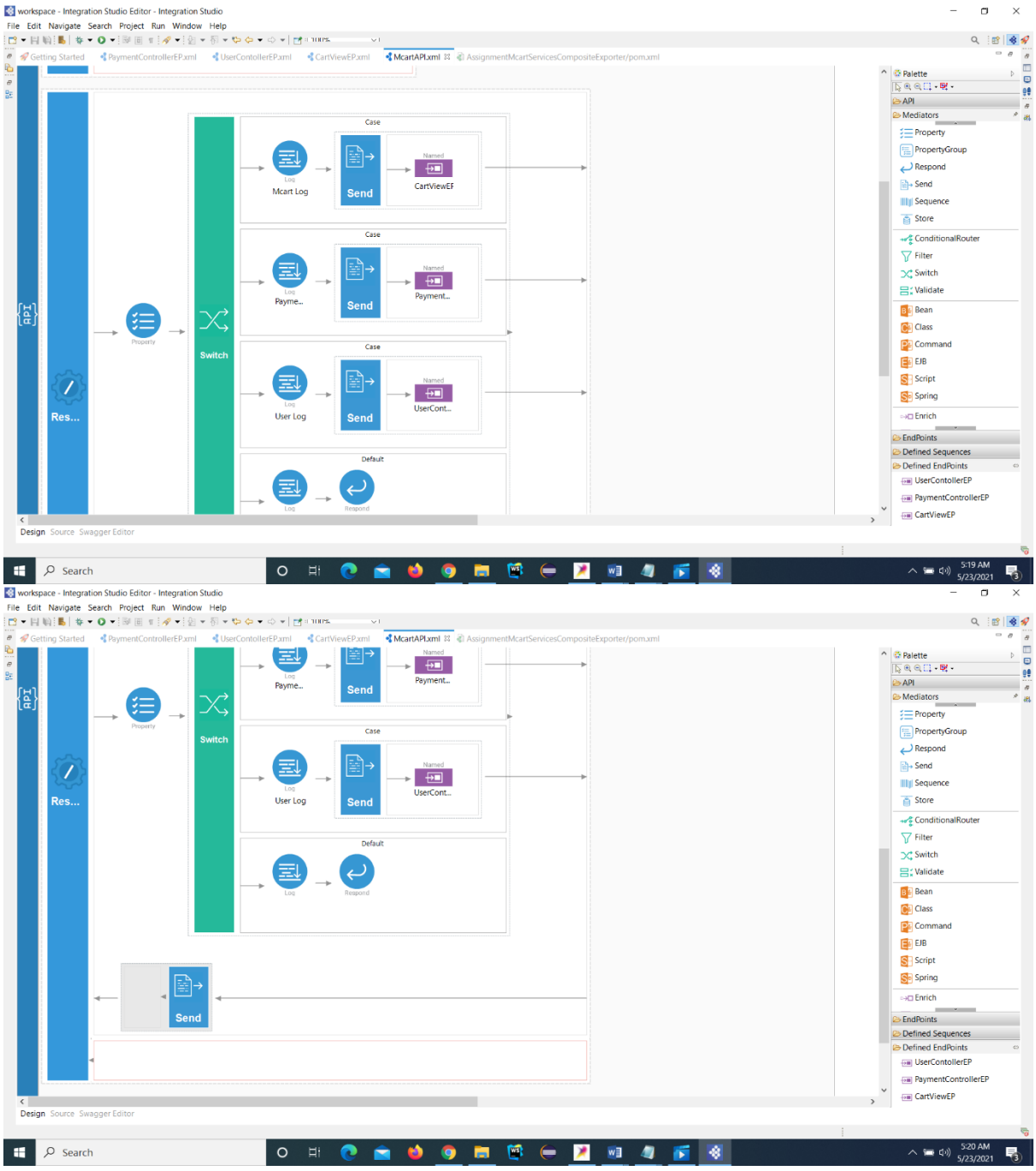


Fig: Payment view where the payment transaction is done.

IX. WSO2 Enterprise Integrator



Appendix

Backend implementation of the user validation

```
onSignInHandler = () => {
    this.setState({ isLoading: true });
    axios
      .post("http://localhost:8000/api/user-login", {
        email: this.state.email,
        password: this.state.password,
      })
      .then((response) => {
        this.setState({ isLoading: false });
        if (response.data.status === 200) {
            localStorage.setItem("isLoggedIn", true);
            localStorage.setItem("userData", JSON.stringify(response.data.data
));
          this.setState({
            msg: response.data.message,
            redirect: true,
            atyp: response.data.data.aType,
          });
        }
        if (
          response.data.status === "failed" &&
          response.data.success === undefined
        ) {
          this.setState({
            errMsgEmail: response.data.validation_error.email,
            errMsgPwd: response.data.validation_error.password,
          });
          setTimeout(() => {
            this.setState({ errMsgEmail: "", errMsgPwd: "" });
          }, 2000);
        } else if (
          response.data.status === "failed" &&
          response.data.success === false
        ) {
          this.setState({
            errMsg: response.data.message,
          });
          setTimeout(() => {
            this.setState({ errMsg: "" });
          }, 2000);
        }
```

```
      })
      .catch((error) => {
        console.log(error);
      });
  };
```

validation to check which account they hold (admin or user) during logIn

```
if (this.state.redirect) {
      if(this.state.atyp === "User"){
      return <Redirect to="/home" />;}else{
        return <Redirect to="/ahome" />;
      }
    }
    const login = localStorage.getItem("isLoggedIn");
    if (login) {
      if(this.state.atyp === "User"){
        return <Redirect to="/home" />;}else{
          return <Redirect to="/ahome" />;
        }
    }
```

Crud implementation for Products

Add in backend

```
public function store(Request $request)
    {
        $product = new Product([
          'pro_name'=> $request->get('pro_name'),
          'pro_code'=> $request->get('pro_code'),
         'cat_id'=> $request->get('cat_id'),
         'pro_info'=> $request->get('pro_info'),
         'pro_price'=> $request->get('pro_price')
        ]);
        $product->save();

            return response()->json('Product Added Successfully.');

    }
```

Add in front end

```
constructor(props){
    super(props);
    this.state = {
      productName: '',
      productCode: '',
      catid: '',
      productinfo: '',
      productprice: ''
    };


    this.handleChange1 = this.handleChange1.bind(this);
    this.handleChange2 = this.handleChange2.bind(this);
    this.handleChange3 = this.handleChange3.bind(this);
    this.handleChange4 = this.handleChange4.bind(this);
    this.handleChange5 = this.handleChange5.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);


  }
  handleChange1(e){
    this.setState({
      productName: e.target.value
    })
  }
  handleChange2(e){
    this.setState({
      productCode: e.target.value
    })
  }
  handleChange3(e){
    this.setState({
      catid: e.target.value
    })
  }
  handleChange4(e){
    this.setState({
      productinfo: e.target.value
    })
  }
  handleChange5(e){
    this.setState({
      productprice: e.target.value
```

```
    })
  }
  handleSubmit(e){

    e.preventDefault();
    const products = {
      pro_name:  this.state.productName,
      pro_code:  this.state.productCode,
      cat_id:  this.state.catid,
      pro_info:  this.state.productinfo,
      pro_price:  this.state.productprice
    }

    console.log(products);
    let uri = MyGlobleSetting.url + '/api/additems';
    console.log(uri);
    alert(`added successful`);
    axios.post(uri, products).then((response) => {
      this.props.history.push('/ahome');
    });
  }
```

Update in backend

```
public function update(Request $request, $id)
    {
        $product = Product::find($id);
        $product->pro_name = $request->get('pro_name');
        $product->pro_code = $request->get('pro_code');
        $product->cat_id = $request->get('cat_id');
        $product->pro_info = $request->get('pro_info');
        $product->pro_price = $request->get('pro_price');
        $product->save();

        return response()->json([
            'ty' => $request->get('pro_name'),
            'message' => 'update succesfully',
            'product' => $product,
        ]);
    }
```

Update in frontend

```
constructor(props) {
      super(props);
      this.state = {
       productName: '',
       productCode: '',
        catid: '',
       productinfo: '',
       productprice: ''
      };

    this.handleChange1 = this.handleChange1.bind(this);
    this.handleChange2 = this.handleChange2.bind(this);
    this.handleChange3 = this.handleChange3.bind(this);
    this.handleChange4 = this.handleChange4.bind(this);
    this.handleChange5 = this.handleChange5.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
    this.handleDSubmit = this.handleDSubmit.bind(this);


  }


  componentDidMount(){
    axios.put(MyGlobleSetting.url + `/api/products${this.props.location.pathna
me}`)
    .then(response => {
      this.setState({
        pro_name:  response.data.productName,
        pro_code:  response.data.productCode,
        cat_id:  response.data.catid,
        pro_info:  response.data.productinfo,
        pro_price:  response.data.productprice


      });
    })
    .catch(function (error) {
      console.log(error);
    })
  }
  handleChange1(e){
    this.setState({
      productName: e.target.value
```

```
    })
  }
  handleChange2(e){
    this.setState({
      productCode: e.target.value
    })
  }
  handleChange3(e){
    this.setState({
      catid: e.target.value
    })
  }
  handleChange4(e){
    this.setState({
      productinfo: e.target.value
    })
  }
  handleChange5(e){
    this.setState({
      productprice: e.target.value
    })
  }
  handleSubmit(event) {
    event.preventDefault();
    const products = {
      pro_name:  this.state.productName,
      pro_code:  this.state.productCode,
      cat_id:  this.state.catid,
      pro_info:  this.state.productinfo,
      pro_price:  this.state.productprice
    }
    let uri = MyGlobleSetting.url + `/api/products${this.props.location.pathna
me}`;
    alert(`updated successful`);
    axios.put(uri, products).then((response) => {
          this.props.history.push('/ahome');
    });
  }
```

Delete in backend

```
 public function destroy($id)
    {
      $product = Product::find($id);
      $product->delete();


      return response()->json('Product Deleted Successfully.');
    }
```

Delete in frontend

```
handleDSubmit(event){
    event.preventDefault();
   let uri = MyGlobleSetting.url + `/api/products/${this.props.match.params.id
}`;
   axios.delete(uri);
   alert(`deleted successful`);
   this.props.history.push('/ahome');
 };
```

Implementing search products along with pagination

Backend

```
public function backend(Request $request){
        //filter for search
        $query = Product::query();

        if($s = $request -> input('s')){
            //where we can use our own condition
            $query -> whereRaw("pro_name LIKE '%" . $s . "%'")
                    -> orWhereRaw("pro_info LIKE '%" . $s . "%'");
        }
        //displaying no. of items per page
        $perPage = 5;
        $page = $request -> input ('page', 1);
        $total = $query -> count();

        $result = $query -> offset(($page -1) * $perPage) -> Limit($perPage) -
> get();


        return [
```

```
            'hits' => $result,
            'total' => $total,
            'page' => $page,
            'last_page' => ceil($total / $perPage)
        ];
    }
```

Frontend

```
//calculated no of pages in db for search
    getPageCount = (total, denominator) => {
        const divisible = 0 === total % denominator;
        const valueToBeAdded = divisible ? 0 : 1;
        return Math.floor( total/denominator) + valueToBeAdded;
    };


    //handles event listner
    handleOnInputChange = (event) => {
        const query = event.target.value;
        if ( ! query ) {
            this.setState({ query, results: {}, message: '', totalPages:0, tot
alResults:0 } );
        } else {
             this.setState({query, loading: true, message: ''  }, () => {
                    this.fetchSearchResults(1, query);
                });
            }
    };

    fetchSearchResults = (updatedPageNo = '', query) => {
        const pageNumber = updatedPageNo ? `&page=${updatedPageNo}` : '';
        const searchUrl = `http://127.0.0.1:8000/api/backend?s=${query}&${page
Number}`;


        if (this.cancel){
            this.cancel.cancel();
        }

        this.cancel = axios.CancelToken.source();

        axios.get(searchUrl,{
            cancelToken: this.cancel.token,
```

```
        })
            .then (res => {

                const total = res.data.total;
                const totalPagesCount = this.getPageCount(total, 5);
                const resultNotFoundMsg = ! res.data.hits.length ?
                                            'There are no more search results.
Please try a new search' :
                                            '';
                this.setState({
                    results: res.data.hits,
                    message: resultNotFoundMsg,
                    totalResults: total,
                    totalPages: totalPagesCount,
                    currentPageNo: updatedPageNo,
                    loading: false
                });

            }).catch ((error)  => {
                if(axios.isCancel(error) || error ){
                    this.setState({
                        loading: false,
                        message: 'failed to fetch the data. pls check network'
                    });
                }

            });
    };

    handlePageClick = (type) => {

        const updatePageNo = 'prev' === type ? this.state.currentPageNo - 1 :
                                                this.state.currentPageNo + 1;

        if(! this.state.loading){
            this.setState({loading: true, message: ''}, () => {
                this.fetchSearchResults(updatePageNo, this.state.query);
            });
        }
    };
```

Handling Routing for link navigation

```
const login = localStorage.getItem("isLoggedIn");
{login ?  (
          <Router>
            <Route exact path="/" component={Signup}></Route>
            <Route path="/sign-in" component={Signin}></Route>
            <Route path="/home" component={Home}></Route>
            <Route path="/ahome" component={Ahome}></Route>

      <Route path="/avhome" component={Master} />
      <Route path="/add-item" component={CreateProduct} />
        <Route path="/display-item" component={DisplayProduct} />
        <Route path="/:id/edit" component={UpdateProduct} />

        <Route path="/cart-items" component={Basket} />

        <Route path="/payment/:V" component={Payment} />

          </Router>
        ) : (

          <Router>
            {navLink}
            <Route exact path="/" component={Signup}></Route>
            <Route path="/sign-in" component={Signin}></Route>
            <Route path="/home" component={Home}></Route>
            <Route path="/ahome" component={Ahome}></Route>

            <Route path="/avhome" component={Master} />
      <Route path="/add-item" component={CreateProduct} />
        <Route path="/display-item" component={DisplayProduct} />
        <Route path="/:id/edit" component={UpdateProduct} />

        <Route path="/cart-items"  component={Basket} />
        <Route path="/payment/:V" component={Payment} />
          </Router>
        )}
```

ACKNOWLEDGEMENT

Foremost, we would like to express our sincere gratitude to our lecturer in charge Mr. Darshana Kasturiratna for the continuous instructions for our assignment two, for his patience, motivation, enthusiasm, and immense knowledge. Besides our advisor, we would like to thank the rest of our instructors' panel for their encouragement. Their guidance helped us in all the time of this research and documentation of our research. Also, I thank my friends who have supported us even from words at our progress to success.

REFERENCES

[01] WSO2 https://docs.wso2.com/display/EI660/Routing+Requests+Based+on+Message+Content

[02] https://www.astera.com