Sri Lanka Institute of Information Technology

Collaborative Shopping Platform Using Restful Services And WSO2 Enterprise Integrator

Project Report

Distributed System Assignment 02

Submitted by:

IT19133850 – (Yasas R.M)

Submitted to:

……

Dr. Dharshana Kasthurirathna

Group members

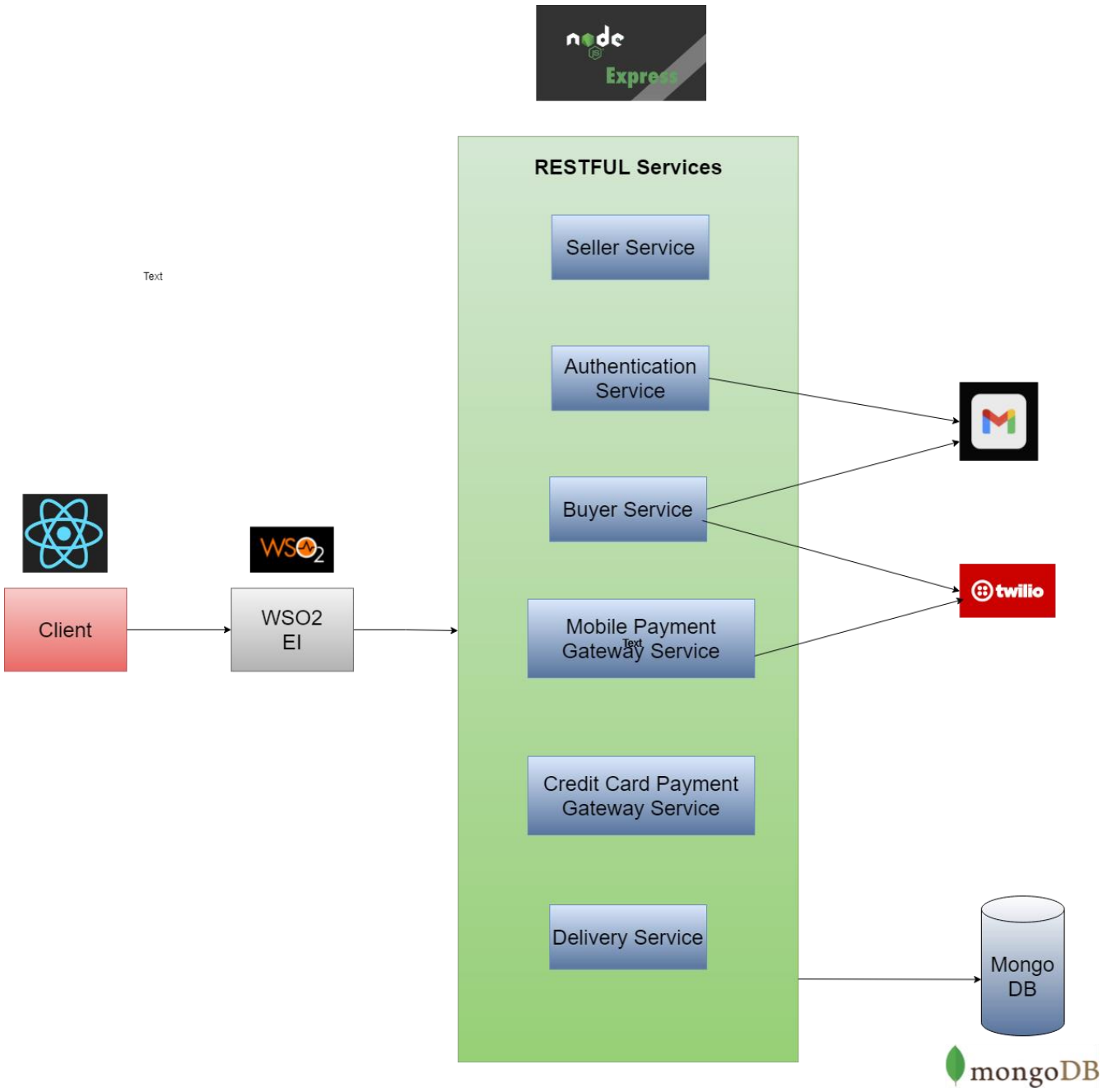
Student ID	Name
IT19133850	Yasas R.M.
IT19152288	Bandara M.H.M.N.D.T.
IT19215884	Gunawardana K.H.R

Group Contribution

Student ID	Implementations
IT19133850	1. Product service to search (Full text), filter and perform all crud operations. 2. Shopping cart management. 3. Dummy credit card payment gateway service. 4. Dummy mobile payment gateway service using Twilio. 5. Place orders with multiple products. 6. Track orders. 7. Using WSO2 EI (Enterprise Integration – ESB) to integrate services at the backend and expose a common web API. 8. Route the payment to either the banking payment gateway or the mobile operator, based on some parameter of the payment request message. 9. Frontend of Delivery service.
IT19152288	1. Seller service where sellers can add/update/delete items.
IT19215884	1. JWT authentication service to perform role based authentication. 2. Delivery service to deliver orders.

Contents

1. High Level Architectural Diagram of the System



2. Service Interfaces exposed by each service

2.1.Authentication service (IT19215884)

auth_rou.js | sendEmail.js | auth.js

routes > auth_rou.js > ...

```
1   const express = require("express");
2   const router = express.Router();
3
4   const {
5     register,
6     login,
7     forgotPassword,
8     resetPassword,
9     ValidateToken,
10  } = require("../controller/auth");
11
12  // register user
13  router.post("/register", register);
14
15  // authenticate
16  router.post("/login", login);
17
18  // recover password
19  router.post("/forgotPassword", forgotPassword);
20
21  // accepting reset token when resetting the password
22  router.put("/passwordreset/:resetToken", resetPassword);
23
24  // validate token
25  router.post("/validateToken", ValidateToken);
26
27  module.exports = router;
28
```

2.2.Seller service (IT19152288)

ProductRoutes.js

seller_service > routes > ProductRoutes.js > ...

```
1   const router = require("express").Router();
2   const ProductController = require("../controllers/ProductController");
3
4   // save product
5   router.post("/", ProductController.saveProduct);
6
7   // update product
8   router.put("/:pid", ProductController.saveProduct);
9
10  // get all products
11  router.get("/", ProductController.listProducts);
12
13  // get product based on id
14  router.get("/:pid", ProductController.findProduct);
15
16  // delete product
17  router.delete("/:pid", ProductController.deleteProduct);
18
19  module.exports = router;
20
```

2.3.Buyer service (IT19133850)

Cart interface (IT19133850)

JS CartRoutes.js ×

buyer_service > routes > JS CartRoutes.js > ...

```
const CartController = require("../controllers/CartController");
const userAuth = require("../middlewares/UserAuth");

const router = require("express").Router();

// add product to the user cart
router.post("/add", userAuth, CartController.addToCart);

// get the user cart
router.get("/", CartController.getCart);

// add, update and delete product , product _quantity from cart
router.post("/", userAuth, CartController.storeToCart);

module.exports = router;
```

Order interface (IT19133850)

JS OrderRoutes.js ×

buyer_service > routes > JS OrderRoutes.js > ...

```
const router = require("express").Router();
const OrderController = require("../controllers/OrderController");
const userAuth = require("../middlewares/UserAuth");

// new order
router.post("/", userAuth, OrderController.newOrder);

//get the order by id
router.get("/:order_id", OrderController.getOrderDetails);

// save order details including delivery
router.post("/:order_id", userAuth, OrderController.saveOrderDetails);

module.exports = router;
```

Payment interface (IT19133850)

PaymentRoutes.js

buyer_service > routes > PaymentRoutes.js > ...

```
1   const router = require("express").Router();
2   const PaymentController = require("../controllers/PaymentController");
3   const userAuth = require("../middlewares/UserAuth");
4
5   // receive order comple notification from gateways
6   router.post("/notify", PaymentController.paymentOrderNotification);
7
8   // complete cod order
9   router.post("/pay/cod", userAuth, PaymentController.codPayment);
10
11  module.exports = router;
```

Product interface (IT19133850)

ProductRoutes.js

buyer_service > routes > ProductRoutes.js > ...

```
1   const router = require("express").Router();
2   const ProductController = require("../controllers/ProductController");
3
4   // get all products
5   router.get("/", ProductController.listProducts);
6
7   // save product
8   router.post("/", ProductController.storeProduct);
9
10  // get product based on id
11  router.get("/:pid", ProductController.findProduct);
12
13  // search product
14  router.get("/search/:text", ProductController.searchProducts);
15
16  // update product
17  router.patch("/:pid", ProductController.updateProduct);
18
19  // delete product
20  router.delete("/:pid", ProductController.deleteProduct);
21
22  module.exports = router;
```

2.4.Credit card payment gateway service (IT19133850)

JS GatewayRoutes.js ×

card_payment_gateway_service > routes > JS GatewayRoutes.js > ...

```
1   const GatewayController = require("../controllers/GatewayController");
2
3   const router = require("express").Router();
4
5   router.post("/", GatewayController.makePayment);
6
7   module.exports = router;
```

2.5.Mobile payment gateway service (IT19133850)

JS GatewayRoutes.js ×

mobile_payment_gateway_service > routes > JS GatewayRoutes.js > ...

```
1   const GatewayController = require("../controllers/GatewayController");
2
3   const router = require("express").Router();
4
5   router.post("/", GatewayController.makePayment);
6
7   router.post("/request-pin/:number", GatewayController.requestPin);
8
9   module.exports = router;
10
```

2.6.Delivery service

Delivery authentication interface (IT19133850)

AuthController.js | User.js | DeliveryController.js | server.js | AuthRoutes.js

routes > api > AuthRoutes.js > ...

```
const router = require("express").Router();
const { UserAuth } = require("../../middlewares/UserAuth");
const AuthController = require("../../controllers/AuthController");

// register user
router.post("/register", AuthController.Register);

// login user
router.post("/login", AuthController.Authenticate);

router.get("/profile", UserAuth, AuthController.Profile);

module.exports = router;

```

Delivery interface (IT19215884)

AuthController.js | User.js | DeliveryController.js | *delivery.js* | server.js

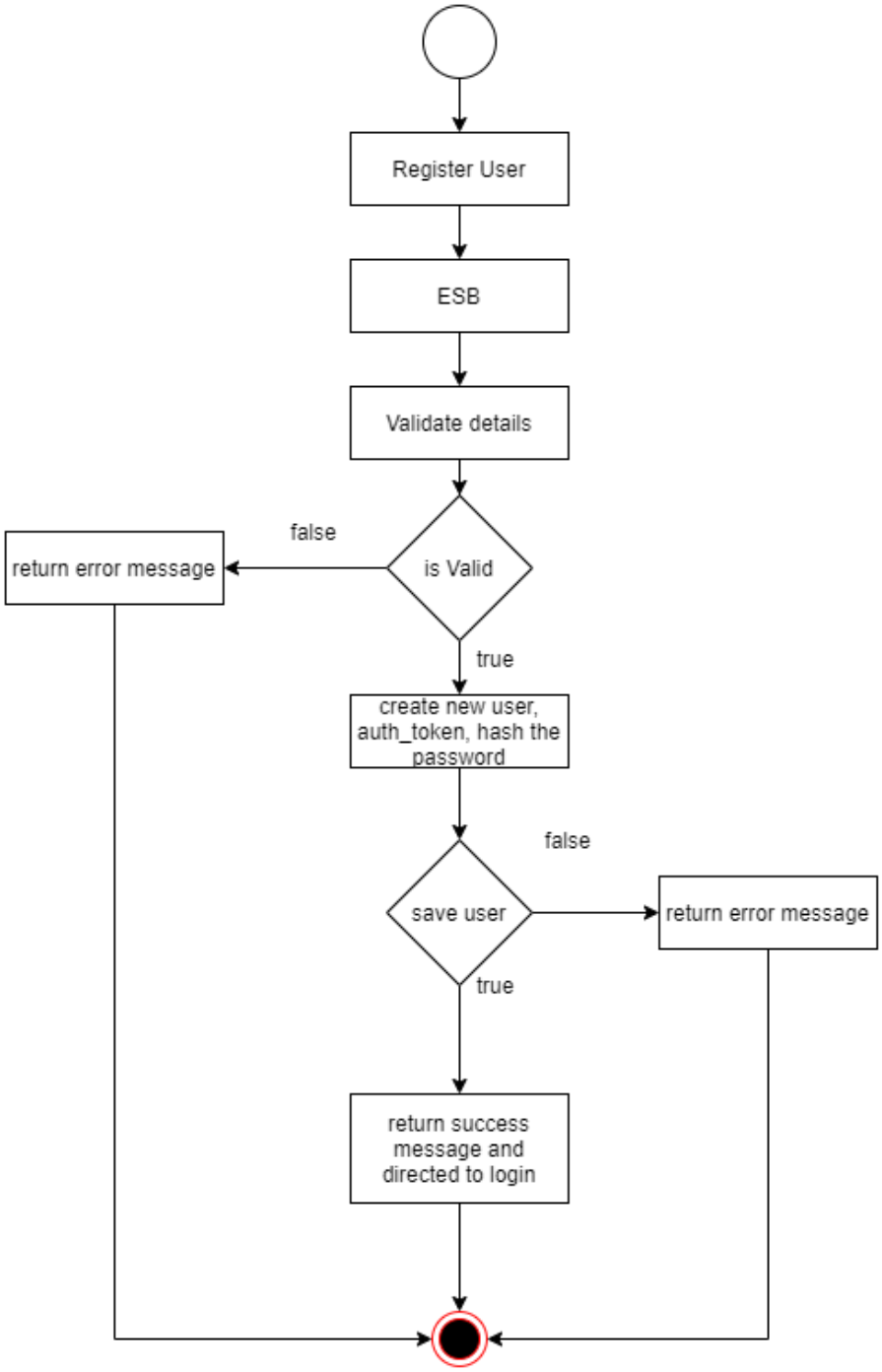
routes > api > delivery.js > ...

```
const express = require("express");
const router = express.Router();
const DeliveryController = require("../../controllers/DeliveryController");

//Create delivery POST
router.post("/", DeliveryController.newDelivery);

// UPDATE delivery  api/delete/:id
router.patch("/:id", DeliveryController.updateDelivery);

// DELETE delivery  api/delete/:id
router.delete("/:id", DeliveryController.deleteDelivery);

// GET all deliveries
router.get("/", DeliveryController.listAllDelivery);

// GET one delivery
router.get("/:id", DeliveryController.findDelivery);

module.exports = router;

```

3. Workflows explained with code Snippets

3.1.Process of registering a new user

Workflow diagram



A POST request with the details (first name, last name, username, email, password, and user role) will be sent to the authentication service through ESB (WSO2 E1). The email should be unique, and it will be checked in exports.register in auth.js. If the request is valid, a new user is created by adding new JSON document to the collection in mongoDB. Simultaneously a token for authentication also created (auth_token) . It will be used for the authentication of the user when login to the system .

The password is converted to a hash code for better privacy using bcrypt and stored. When querying, password is not passing unless specially asked for it. Once the registration is successful the user will be directed to the menu by considering the user role, as a seller to the seller dashboard and a buyer to the online store.

Creation of the token

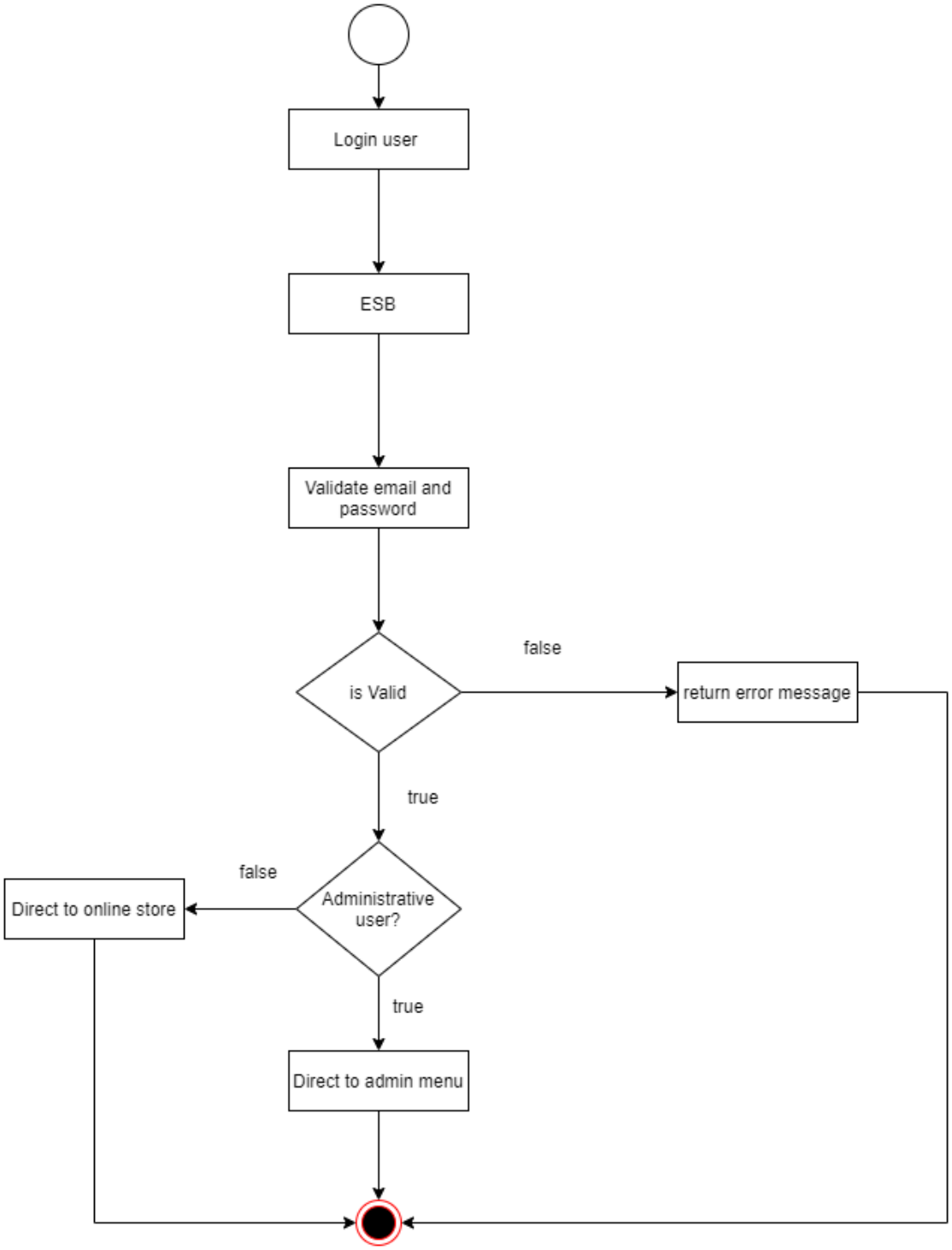
```
exports.protect = async (req,res,next)=>{
    let token;

    if(req.headers.authorization && req.headers.authorization.startsWith("Bearer"))
    {
       token = req.headers.authorization.split("")[1]
    }
    if(!token)
    {
      return res.status(401).json ({success:false, error:"Not authorized to access this route"});
    }
    try
    {
        const decoded = jwt.verify(token,process.env.JWT_SECRET);
        const user = await user.findById(decoded.id); // find the user by token
        if(!user)
        {
            return next(new ErrorResponse ("No user found with this id",404)); // if the user not found return the error
        }
        req.user= user;
        next();
    }
    catch (error)
    {
        return next (new ErrorResponse (" Not authorized to access this route",401));
    }
};
```

Converting password into hash cod

```
UserSchema.pre("save", async function(next){  //
    if (!this.isModified("password")){
        next();
    }

const salt = await bcrypt.genSalt(10);    // await bcz it returns a promise
this.password = await bcrypt.hash (this.password,salt); // this refers to password in controller/auth.js
// Change the password that was sent, save  the new password in password field

next();

});
```

3.2.Process of login

Workflow diagram



A POST request with email and password will be sent to the authentication service through ESB (WSO2 E1). Once request is received the system check the availability of the user by searching for the email and match the passwords. If the passwords match a sign in token will be generated and the user is directed to the respective menus based on the user role. Otherwise, an error message will be generated and if the user is not found , the user will be directed to the registration to create new user. If in case, user forgot the password, then the user will be directed to the forgot password to reset the password.

```
UserSchema.methods.matchPasswords = async function (password)
{
    return await bcrypt.compare (password, this.password);    // comparing passwords
};
```

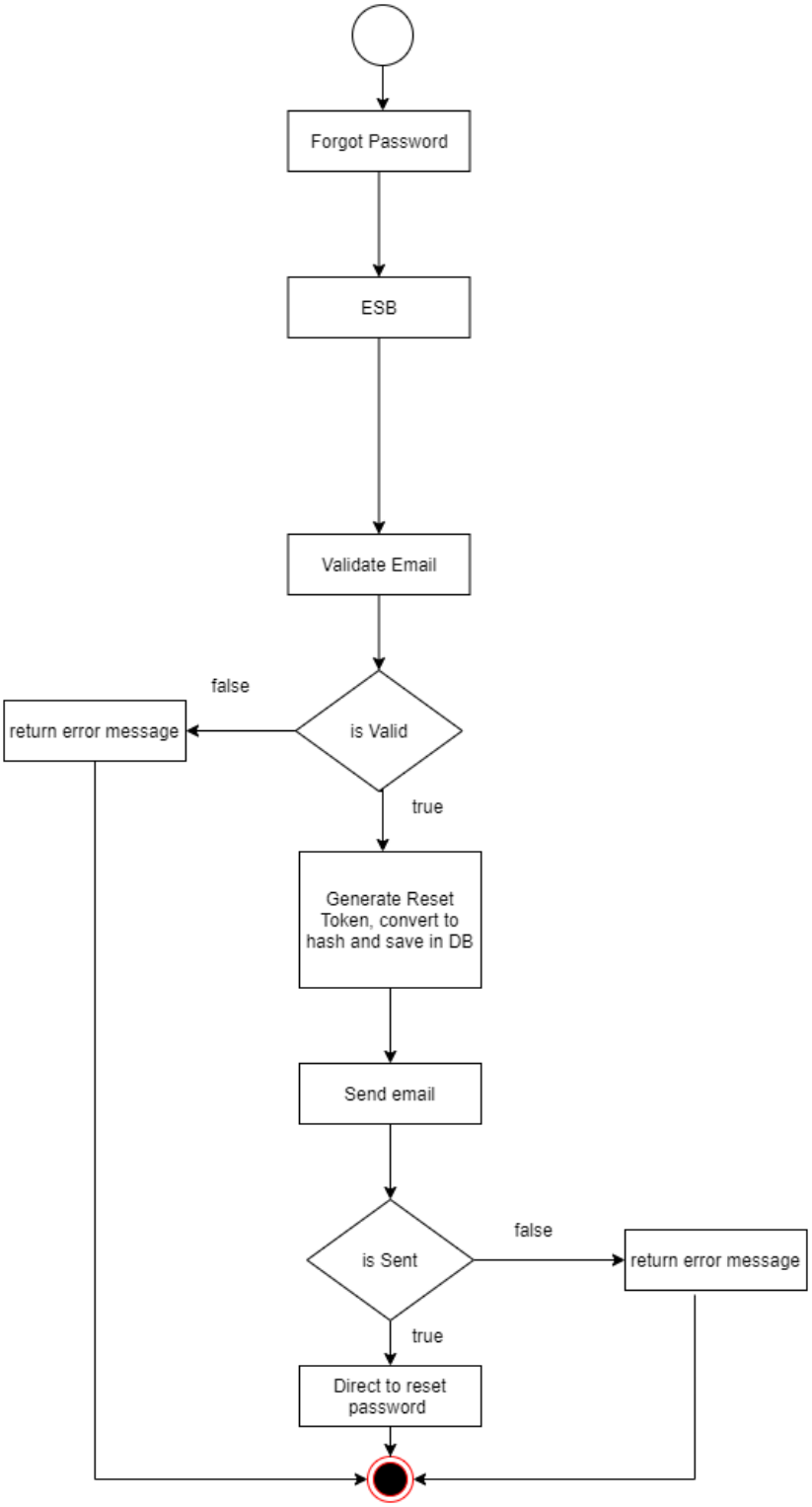
Comparing Password

Generating Sign in Token

```
UserSchema.methods.getSignedJwtToken =  function ()
{
   return jwt.sign({id:this._id}, process.env.JWT_SECRET,{expiresIn: process.env.JWT_EXPIRE}); //
};
```

3.3.Process of forgot password

Workflow diagram



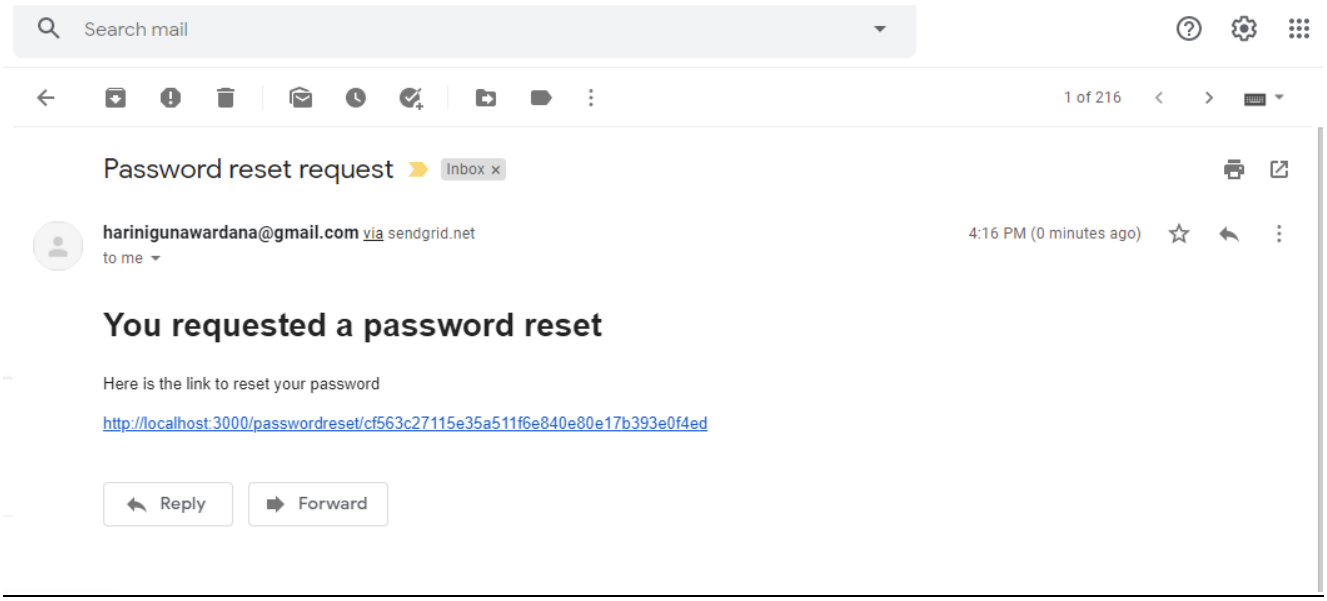
If the user forgot password, a POST request with the email will be sent to the authentication service through the ESB (WSO2 E1). export.forgotPassword in auth.js checks whether that email is existing in the database. If the email exists a reset token is created which is expired in 10mins, and the hashed version of the token will be stored in the database. An email with the http message embedded with the reset url and the reset token will be sent to the provided email address. Nodemailer and sendGrid is used for this purpose. Once the email sending procedure is successfully done, success message will be generated. Otherwise, it will generate error messages.

Generation Reset Password Token

```
UserSchema.methods.getResetPasswordToken = function()
{
    const resetToken = crypto.randomBytes(20).toString("hex");
    this.resetPasswordToken = crypto.createHash("sha256").update(resetToken).digest("hex");
    this.resetPasswordExpire = Date.now() + 10* (60*1000);// 10 mins  //Reset Password token  expires on 10 mins
    return resetToken;
}
```

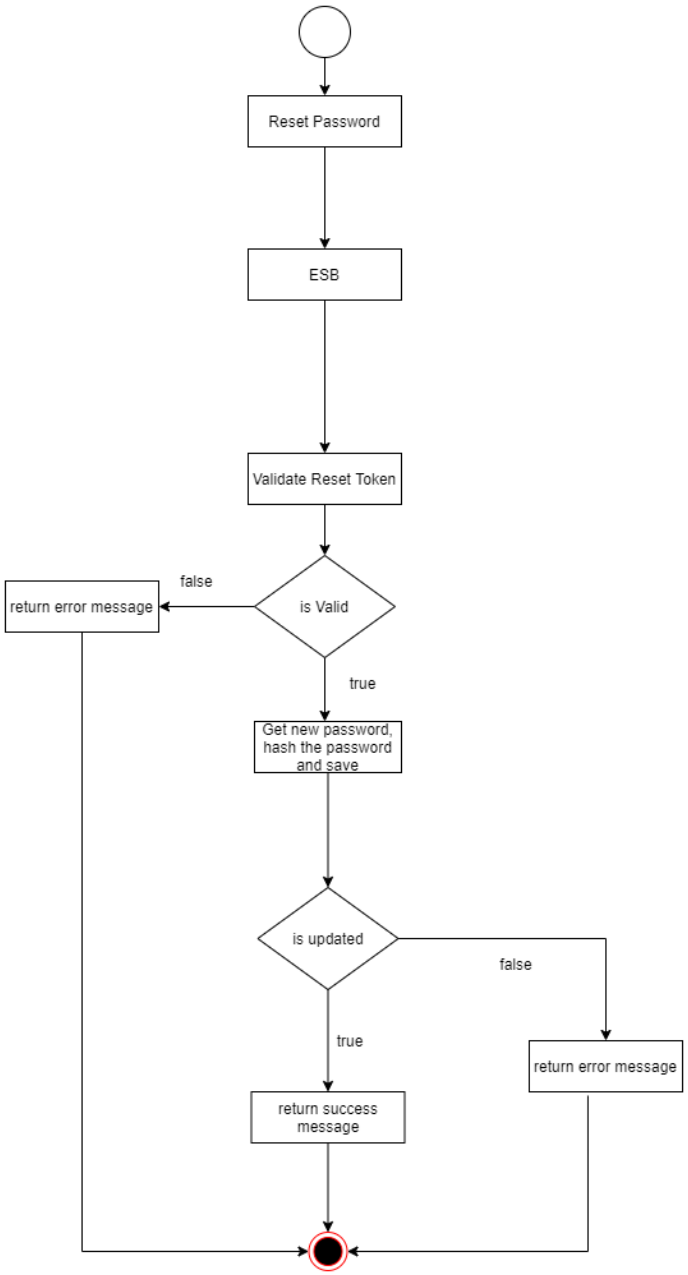
Creation of Email Service with Nodemailer and SendGrid

```
const nodemailer = require("nodemailer");

const sendEmail = (options) => {
  const transporter = nodemailer.createTransport({
    service: process.env.EMAIL_SERVICE,
    auth: {
      user: process.env.EMAIL_USERNAME,
      pass: process.env.EMAIL_PASSWORD,
    },
  });

  const mailOptions = {
    from: process.env.EMAIL_FROM,
    to: options.to,
    subject: options.subject,
    html: options.text,
  };

  transporter.sendMail(mailOptions, function (err, info) {
    if (err) {
      console.log(err);
    } else {
      console.log(info);
    }
  });
};

module.exports = sendEmail;
```


Search mail
1 of 216
Password reset request
Inbox ×
harinigunawardana@gmail.com via sendgrid.net
to me
4:16 PM (0 minutes ago)
You requested a password reset
Here is the link to reset your password
http://localhost:3000/passwordreset/cf563c27115e35a511f6e840e80e17b393e0f4ed
Reply
Forward

3.4.Process of resetting user password

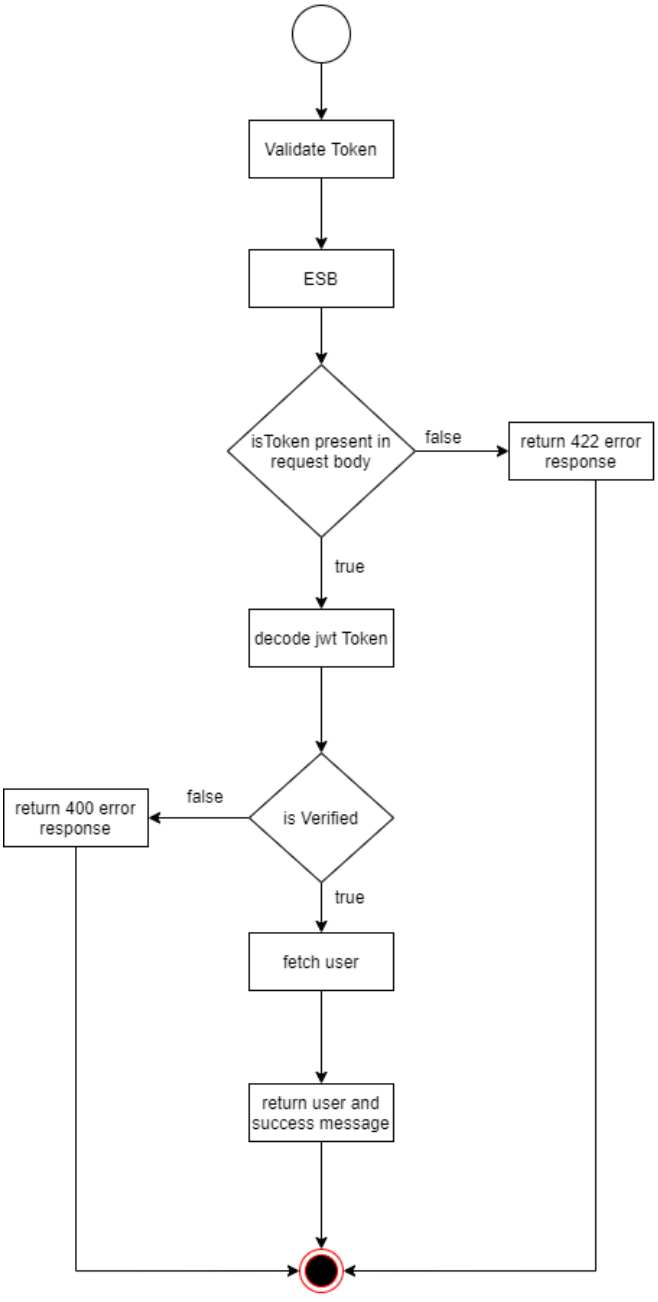
Workflow diagram



A PUT request with a reset token will be sent to the authentication service through the ESB (WSO2 E1) when user click on the link in the password reset request email. export.resetPassword compares the token in the URL parameters with the hashed token. If the validation is success, a user interface to reset the password will be appeared. User can reset the password through that UI and after the new password created it will be hashed and the database the JSON document will be updated with the new password. After a successful password reset, the user will be again directed to the login. If some error occurred during the process, an error message will be created.

3.5.Process of validating the token

Workflow diagram



Service interface

```
// validate token
router.post("/validateToken", AuthController.ValidateToken);

module.exports = router;
```

Service function

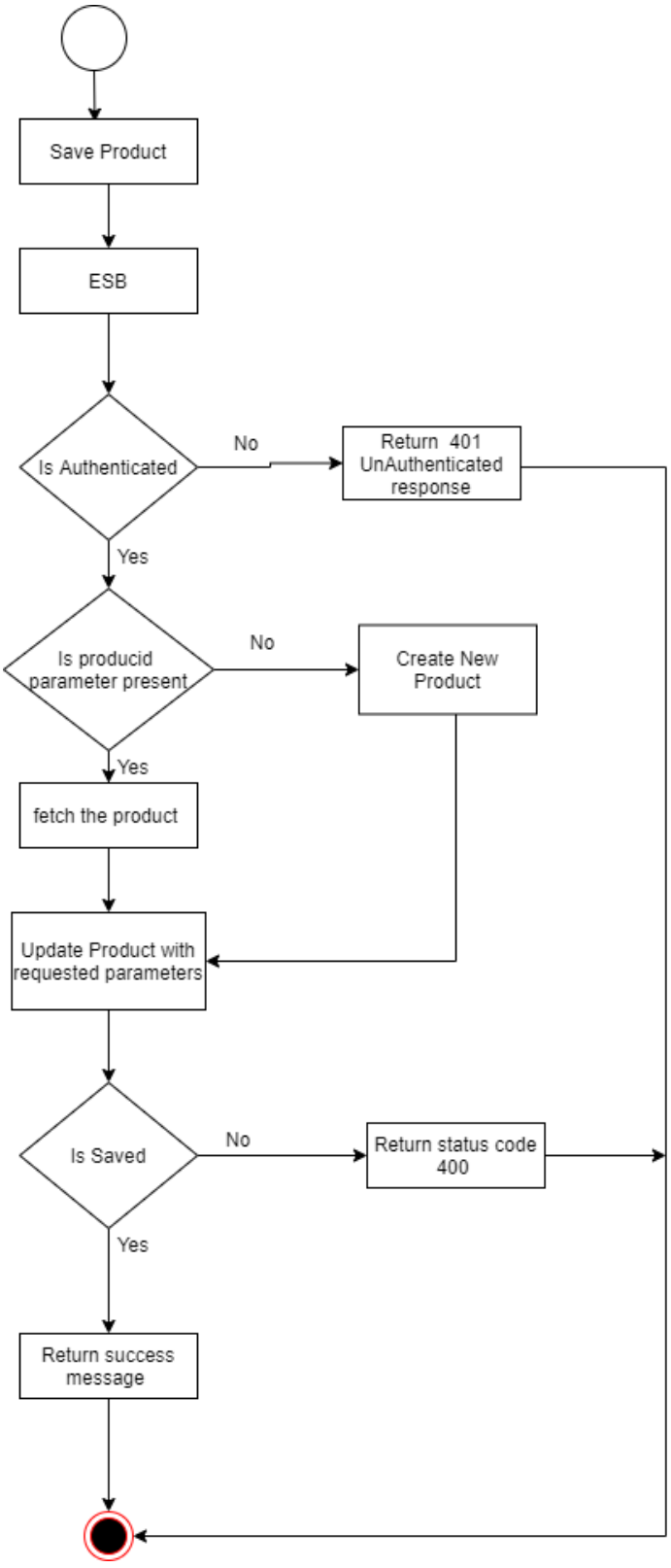
```
109
110   exports.ValidateToken = async (req, res) => {
111     try {
112       // validate token in request
113       if (!req.body.token)
114         return res.status(422).json({ message: "JWT token is required" });
115       // validate JWT token
116       var decodedToken = jwt.verify(req.body.token, KEY);
117       // fetch user token token user id
118       var user = await User.findById(decodedToken.user_id);
119       return res.status(200).json(user);
120     } catch (error) {
121       return res.status(400).json({ message: "JWT token is not valid" });
122     }
123   };
```

The initial POST request will be sent to the authentication service through ESB. (WSO2 EI). The token will be sent as a property in the request and it will be validated. If the token is not present then an error response with status code 422 will be returned.

To validate this token jwt library will be used. If the token is validated then the user will be retrieved from the token id and returned back. Else, an error response with status code 400 will be returned.

3.6.Process of creating a new product

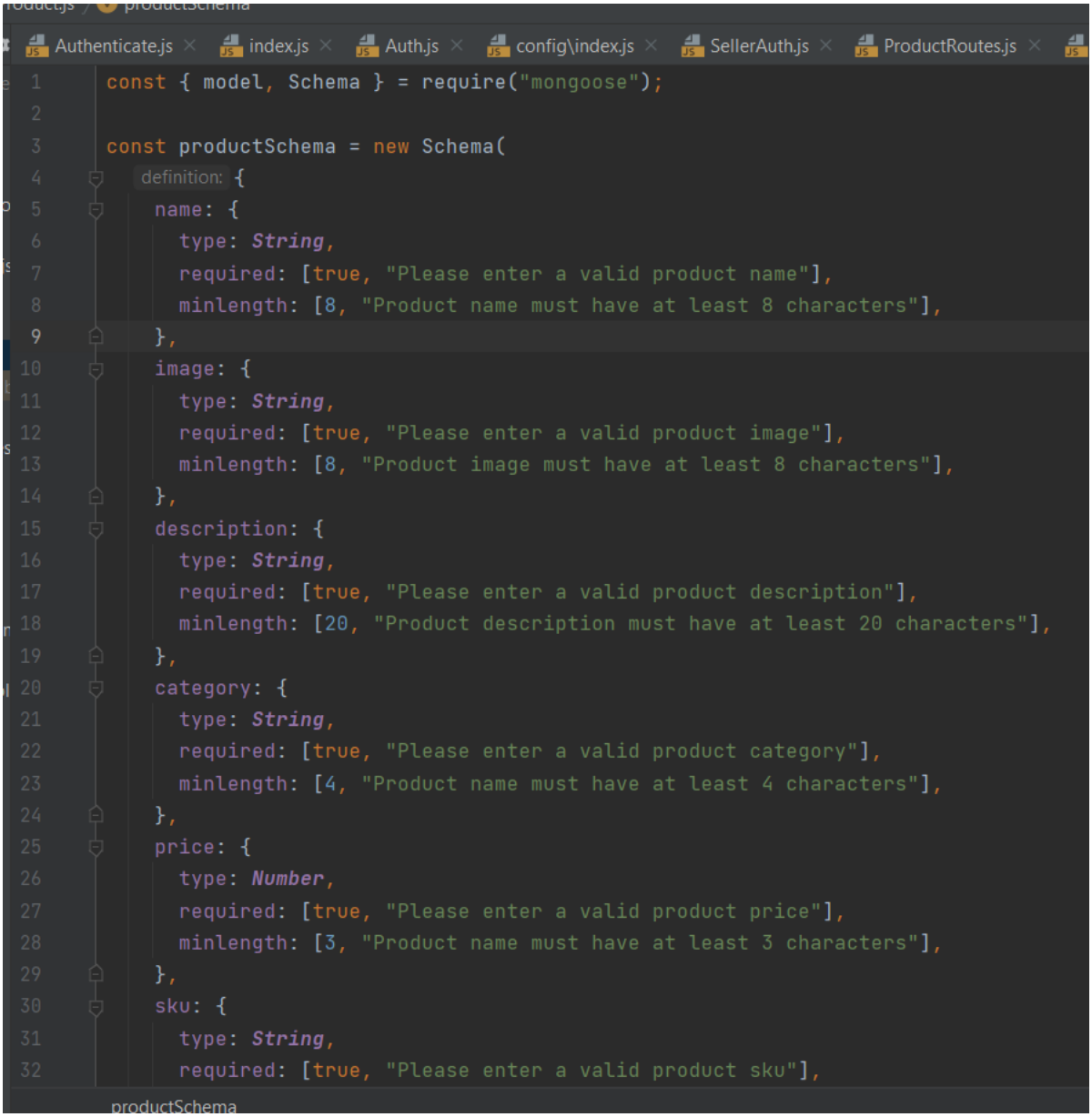
Workflow diagram

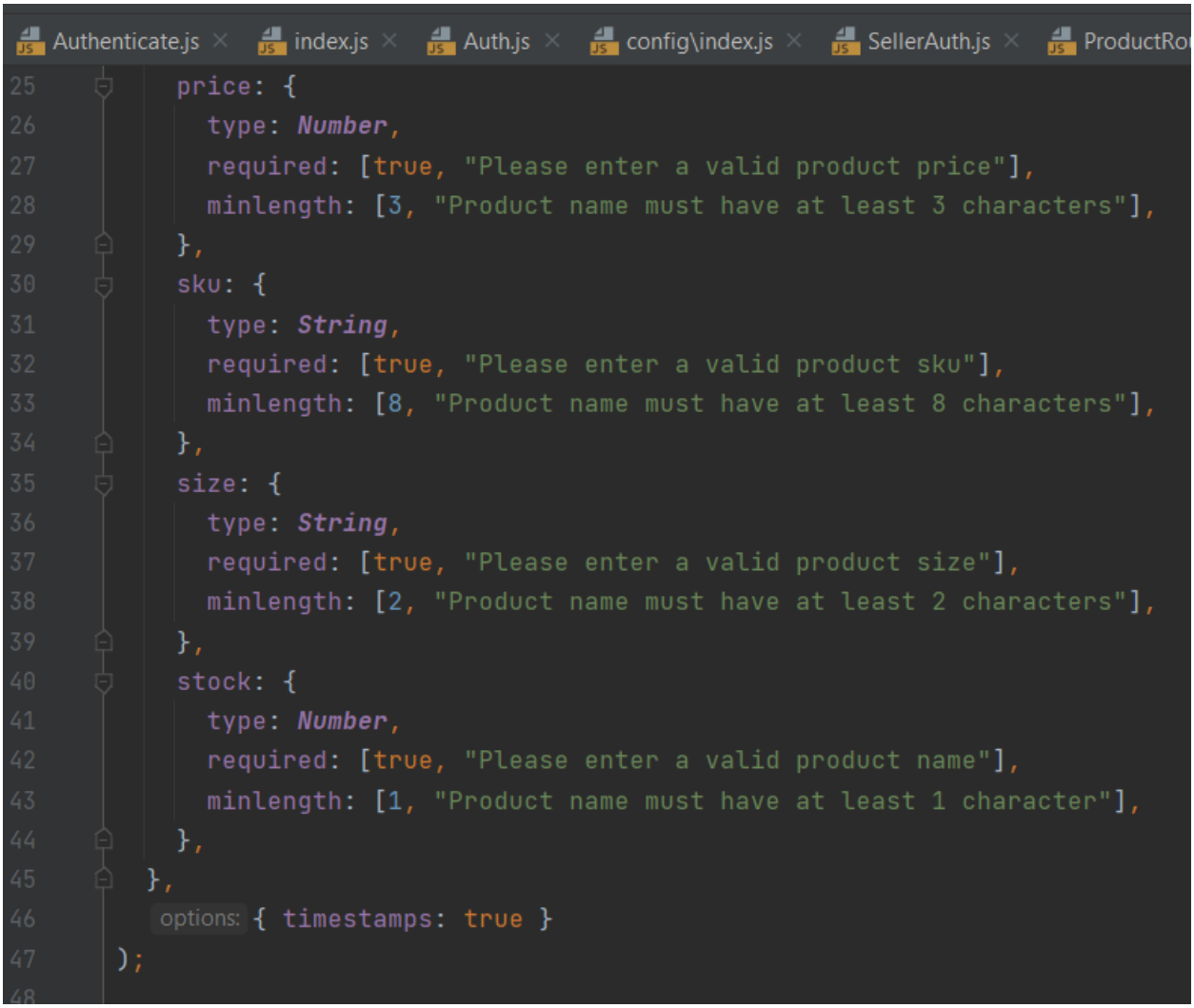


POST request will be sent to the seller service through the ESB. An authenticated user with role seller can save and update the products.

The request body must have name, image, description, price, sku, size, stock, and category. If there is a parameter with product id then product will be fetched. If there is no product id then new product will be initialized and data will be appended to product model we created and data will be saved. If there is product then the fetched product will be updated with the requested data.

If an exception occurred then it will return an error response with 400 status code. If data is saved then a response with 200 status code will be send back with the product data. If any error occurred during saving product (invalid data types or character length), then a error response will be sent with status code 422. Error messages are mentioned in the product model.




Authenticate.js
index.js
Auth.js
config\index.js
SellerAuth.js
ProductRo
price: {
type: Number,
required: [true, "Please enter a valid product price"],
minlength: [3, "Product name must have at least 3 characters"],
},
sku: {
type: String,
required: [true, "Please enter a valid product sku"],
minlength: [8, "Product name must have at least 8 characters"],
},
size: {
type: String,
required: [true, "Please enter a valid product size"],
minlength: [2, "Product name must have at least 2 characters"],
},
stock: {
type: Number,
required: [true, "Please enter a valid product name"],
minlength: [1, "Product name must have at least 1 character"],
},
},
options: { timestamps: true }
);

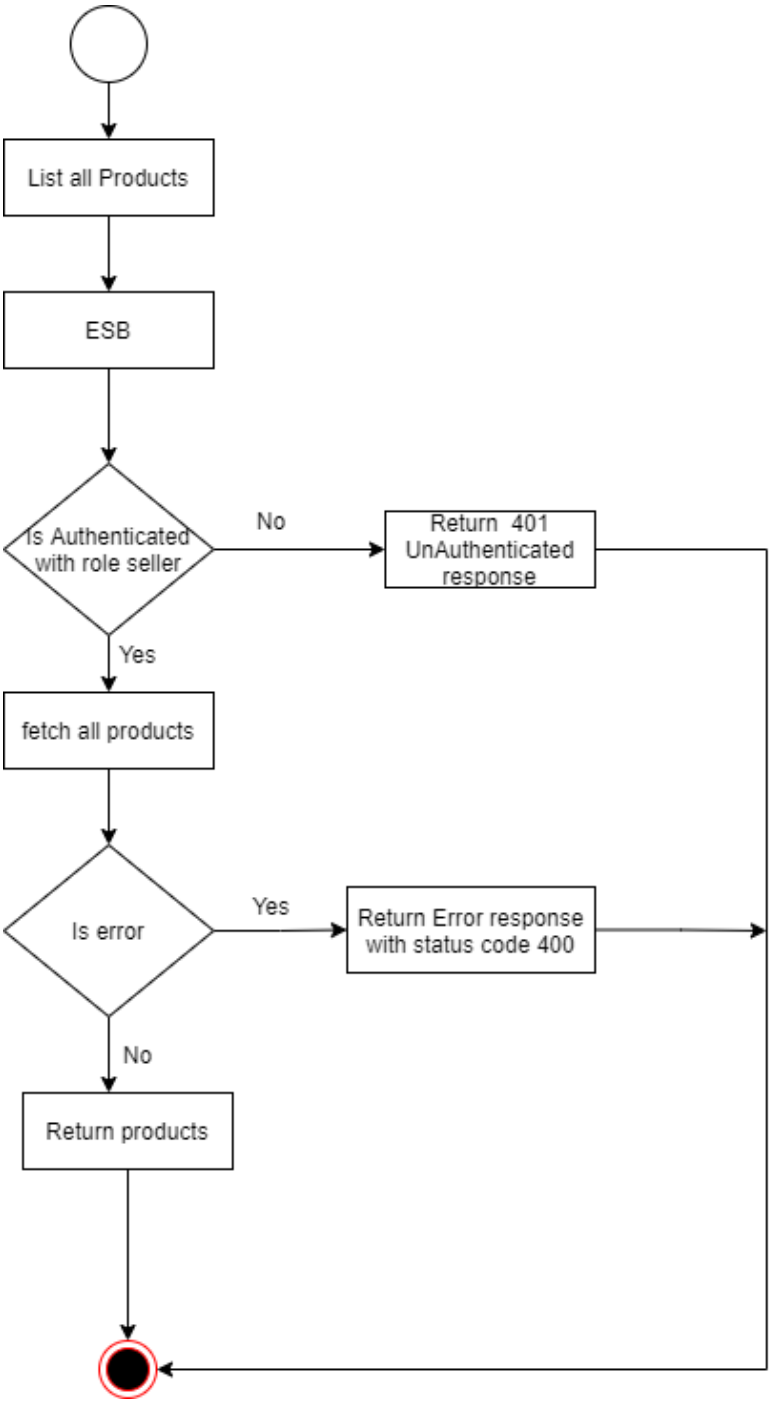
3.7.Process of updating a product

To update product PUT request will be send to the seller service through the ESB. An authenticated user with role seller can update the products. The request must have a parameter productid. and then save product function will be invoke in order to update product. Creating the product and updating the product will be done using the same function to facilitate reusability.

Since the productid is present in request parameter it will be used to fetch the product and request new body data will be appended to this and then updated or else error message will be send back.

3.8.Process of retrieving all products

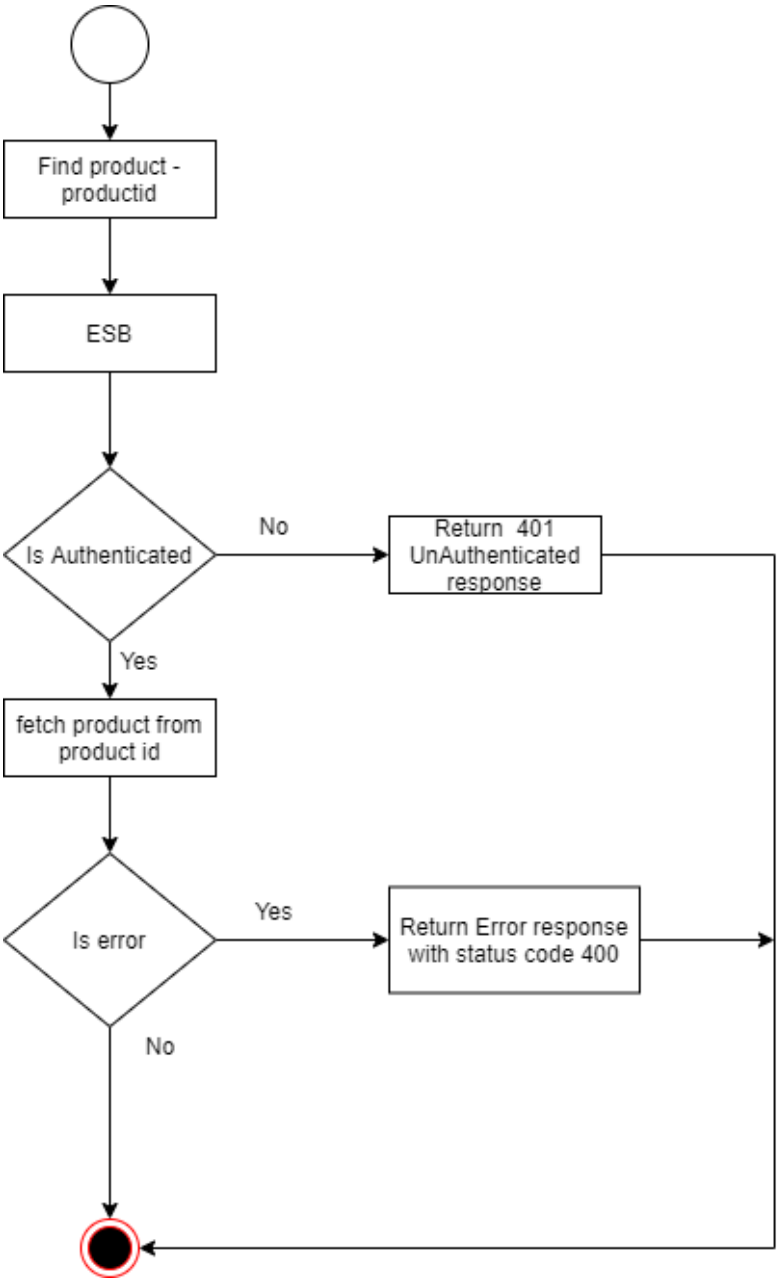
Workflow diagram



GET request will be sent to list products toward the seller service through the ESB. An authenticated user with role seller can list all the products. The products will be fetched from the product model. If the product is available successful message with 200 status code will be sent back. If there is error it will return error message with 400 status code.

3.9.Process of retrieving a product

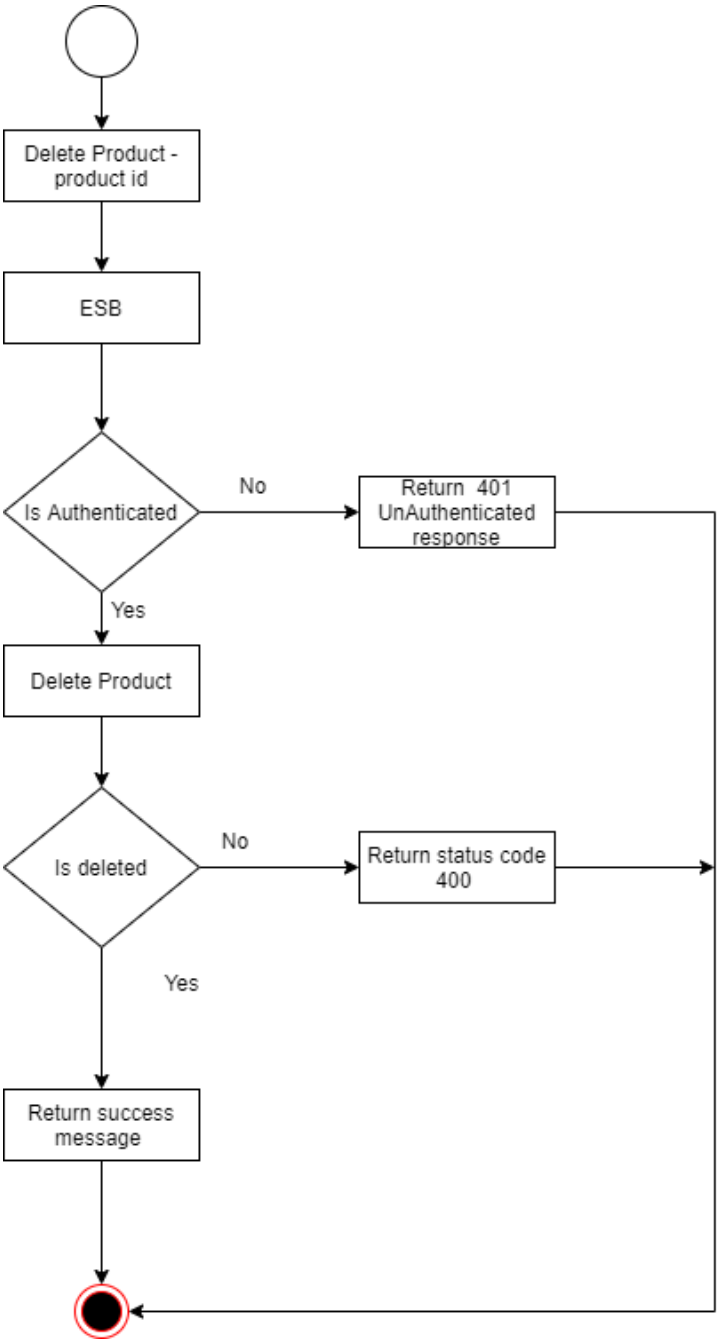
Workflow diagram



GET request will be sent with product id as a parameter toward the seller service through the ESB. An authenticated user with role seller can view a product. The products will be fetched from the product model using the parameter send in the request. If the product is fetched return successful message with 200 status code. If there is error it will return error message with 400 status code.

3.10. Process of deleting a product

Workflow diagram



DELETE request will be sent with product id as a request parameter toward the seller service through the ESB. An authenticated user with role seller can delete a product. The products will be deleted from database. If product was deleted successfully message with 200 status code will be send back. Else error response will be sent.

3.11. Process of searching the products

Service interface

```
12
13    // search product
14    router.get("/search/:text", ProductController.searchProducts);
15
```

Service function

```
6   // search products , full text search
7   exports.searchProducts = async (req, res) => {
8     try {
9       let products = await Product.find({
0         $text: { $search: req.params.text },
1       });
2       if (products.length > 0) return res.status(200).json(products);
3       else return res.status(400).json({ message: "Products not found" });
4     } catch (error) {
5       console.error(error);
6       return res.status(400).json(error);
7     }
8   };
9
```

The initial GET request will be sent to the buyer service through ESB. (WSO2 EI) with a parameter (search text). Products will be search from this parameter and will be returned back to the client through ESB. To perform the text search the product model has been modified as below.

```
productSchema.index({ name: "text", description: "text" });

module.exports = model("product", productSchema);
```

When searching products, it will match the text with product names and descriptions. If no products were found an error response with status code 400 will be sent back.

3.12. Shopping cart management

Service functions / interfaces to manage the shopping cart.

JS CartRoutes.js

routes > JS CartRoutes.js > ...

```
1   const CartController = require("../controllers/CartController");
2   const userAuth = require("../middlewares/UserAuth");
3
4   const router = require("express").Router();
5
6   // add product to the user cart
7   router.post("/add", userAuth, CartController.addToCart);
8
9   // get the user cart
10  router.get("/", CartController.getCart);
11
12  // add, update and delete product , product _quantity from cart
13  router.post("/", userAuth, CartController.storeToCart);
14
15  module.exports = router;
```

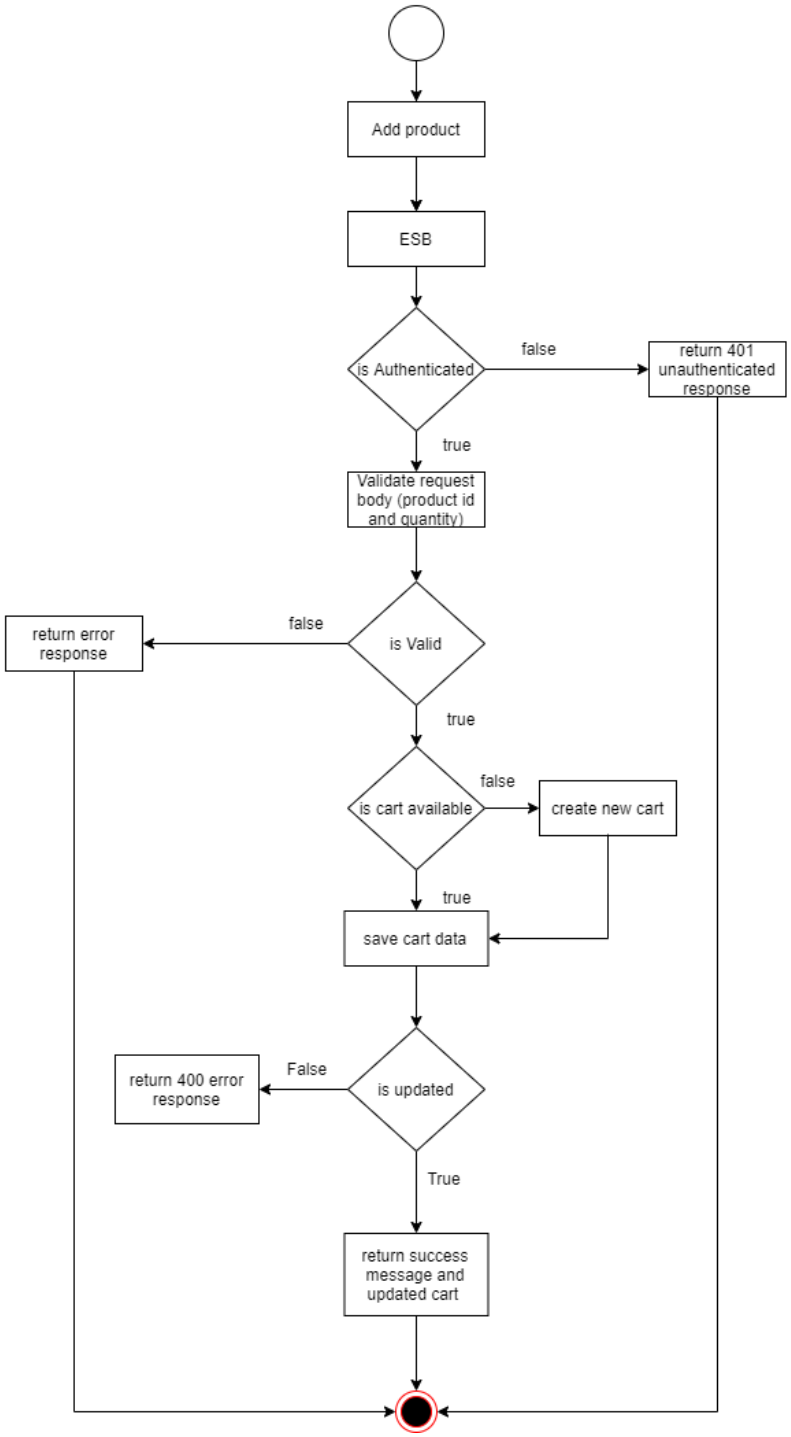
There are three routes in cart management in the buyer service.

1. A POST request with product and its quantity to add a product to shopping cart.
2. A GET request to retrieve user's cart details with products.
3. A POST request to save / update product quantities in the shopping cart.

All these routes are mapped using the WSO2 Enterprise Integrator.

3.12.1. Add a product to shopping cart

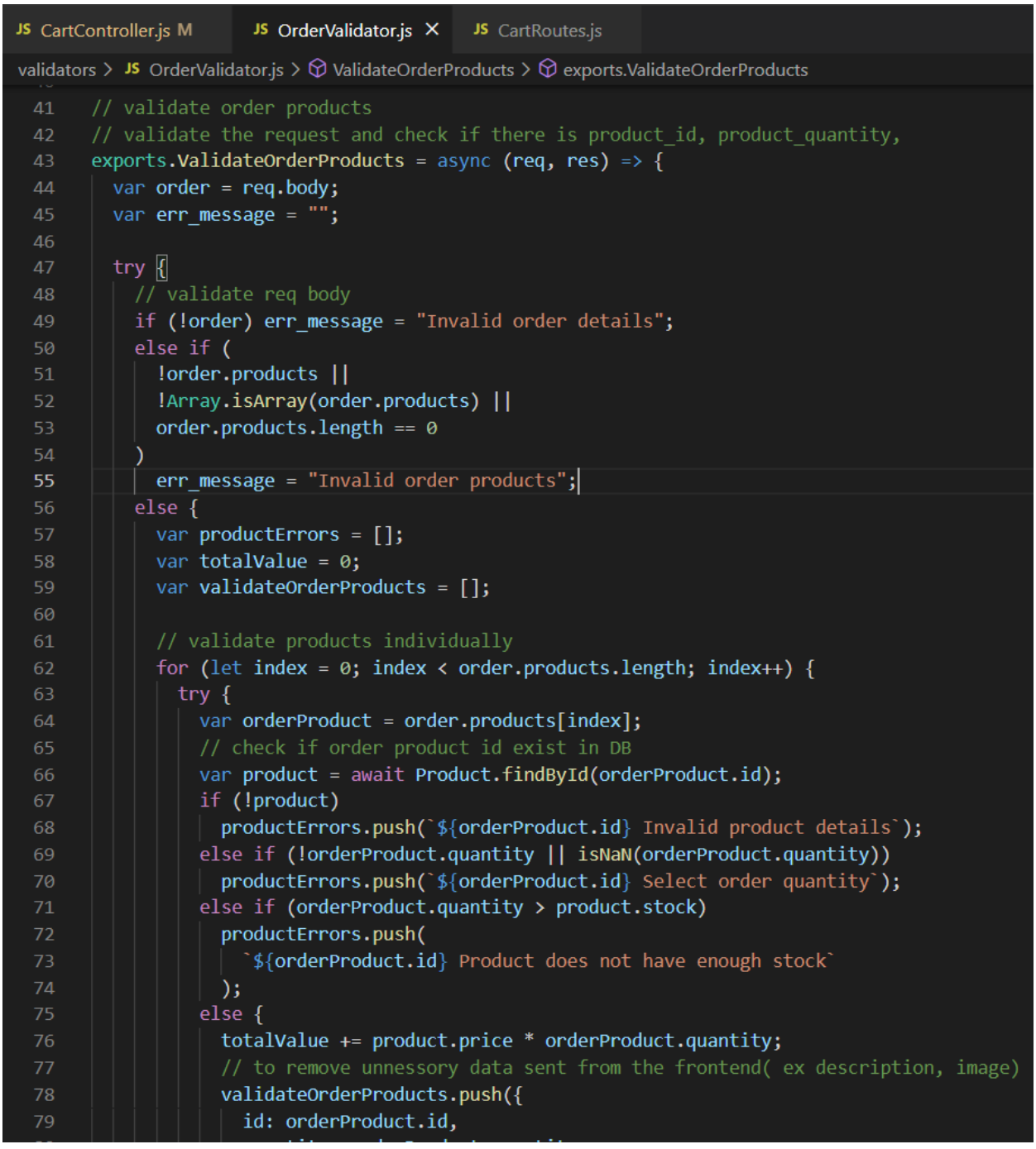
Workflow diagram



Service function to add a product to cart

```
exports.addToCart = async (req, res) => {
  try {
    // validate the products in the cart
    //use the validator used to validate order products
    var validatedOrder = await OrderValidator.ValidateOrderProducts(req, res);

    var cart = await Cart.findOne({ user_id: req.user._id });

    if (!cart) {
      // ifthe user does not habve a cart
      cart = new Cart({
        user_id: req.user._id,
        products: validatedOrder.products,
        payment_value: validatedOrder.payment_value,
      });
    } else {
      // if the user already have a cart
      cart.products = [...cart.products, ...validatedOrder.products];
      cart.payment_value =
        validatedOrder.payment_value + validatedOrder.payment_value;
    }
    var result = await cart.save();

    // if cart save fail
    if (result && result.error) return res.status(400).json(result);

    return res
      .status(200)
      .json({ message: "All changes was saved", cart: result._doc });
  } catch (error) {
    console.error(error);
    return res.status(400).json({ message: "Unexpected error" });
  }
};
```

Validator function to validate request body such as valid products, products have stock (validateOrderProducts)


JS CartController.js M
JS OrderValidator.js
JS CartRoutes.js
validators > JS OrderValidator.js > ValidateOrderProducts > exports.ValidateOrderProducts
41 // validate order products
42 // validate the request and check if there is product_id, product_quantity,
43 exports.ValidateOrderProducts = async (req, res) => {
44 var order = req.body;
45 var err_message = "";
46
47 try {
48 // validate req body
49 if (!order) err_message = "Invalid order details";
50 else if (
51 !order.products ||
52 !Array.isArray(order.products) ||
53 order.products.length == 0
54)
55 err_message = "Invalid order products";
56 else {
57 var productErrors = [];
58 var totalValue = 0;
59 var validateOrderProducts = [];
60
61 // validate products individually
62 for (let index = 0; index < order.products.length; index++) {
63 try {
64 var orderProduct = order.products[index];
65 // check if order product id exist in DB
66 var product = await Product.findById(orderProduct.id);
67 if (!product)
68 productErrors.push(`${orderProduct.id} Invalid product details`);
69 else if (!orderProduct.quantity || isNaN(orderProduct.quantity))
70 productErrors.push(`${orderProduct.id} Select order quantity`);
71 else if (orderProduct.quantity > product.stock)
72 productErrors.push(
73 `${orderProduct.id} Product does not have enough stock`
74);
75 else {
76 totalValue += product.price * orderProduct.quantity;
77 // to remove unnessory data sent from the frontend(ex description, image)
78 validateOrderProducts.push({
79 id: orderProduct.id,

```
            else {
              totalValue += product.price * orderProduct.quantity;
              // to remove unnessory data sent from the frontend( ex description, image)
              validateOrderProducts.push({
                id: orderProduct.id,
                quantity: orderProduct.quantity,
              });
            }
          } catch (error) {
            productErrors.push(`${orderProduct.id} Invalid product details`);
          }
        }

        if (productErrors.length > 0)
          return res
            .status(422)
            .json({ message: "There were errors in products", productErrors });
        else {
          // set order payment and validated order products
          order.payment_value = totalValue;
          order.products = validateOrderProducts;
        }
      }
      if (err_message.length > 0)
        return res.status(422).json({ message: err_message });
      else return order;
    } catch (error) {
      return res.status(422).json({ message: "There were errors in products" });
    }
  };
```

POST request will be sent to the buyer service through ESB. (WSO2 EI).

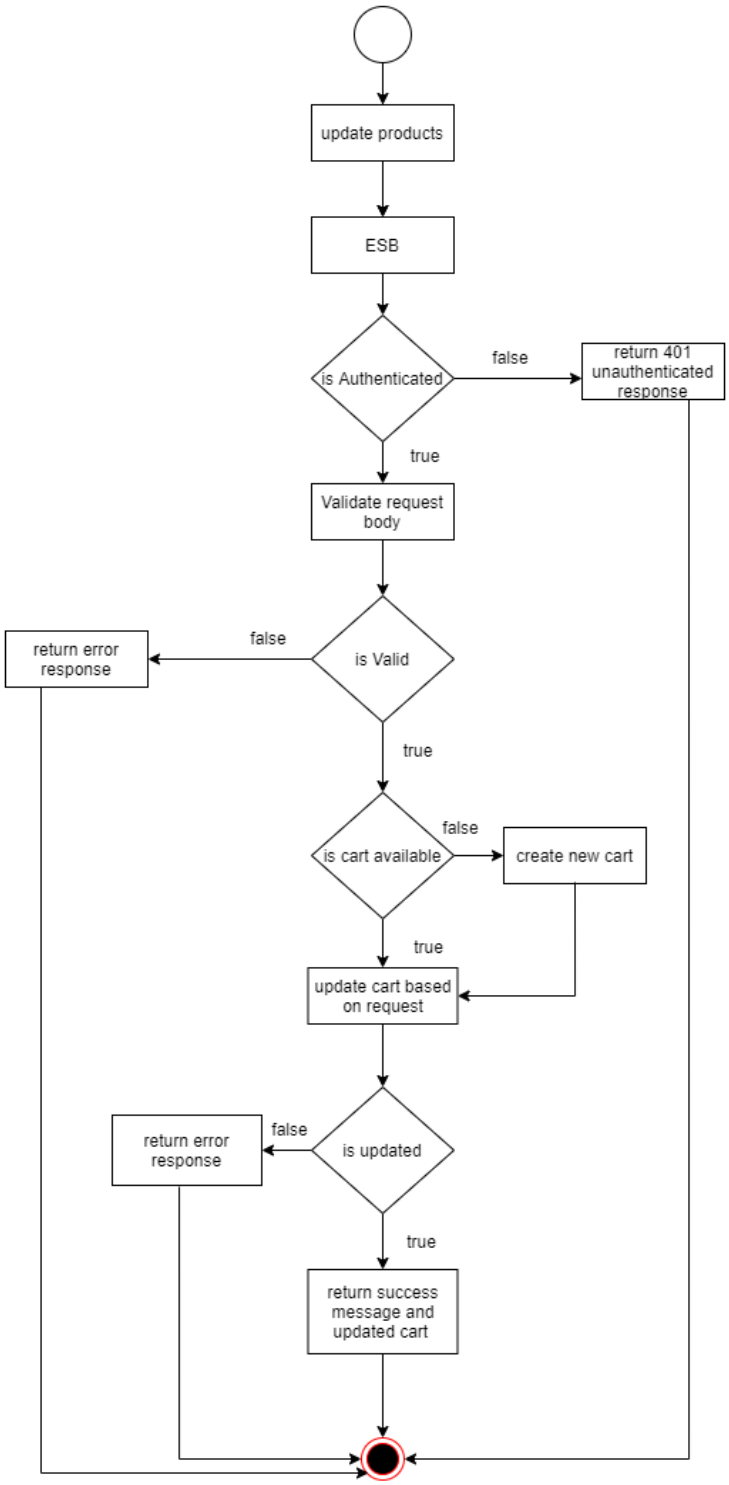
To add a product to cart, user must be authenticated for this “userAuth” middleware is used to filter the request and allow only authenticated users. Request body must have product id and its quantity. Then the request is validated to check if it contains product data that exist in database also the product quantity requested is less than or equal with its stock else relevant error messages are sent back to the client through ESB. If the request body is validated, it will return the total cart value(value of products * quantity) with the products.

Then user cart will be retrieved from the database using the user’s id. If the cart does not exist, create a new cart by providing user’s ID, products and the total cart value. Else, update the existing cart by adding the new product to cart products and the total cart value.

The cart will be saved/updated and if it fails then an error message will be sent else, the updated user cart object will be sent to the client through ESB.

3.12.2. Update products quantity in shopping cart

Workflow diagram



Service function

```
// add, update or delete a product in cart
exports.storeToCart = async (req, res) => {
  try {
    var validatedCart = {};
    // validate the products in the cart if it has products only
    if ( req.body.products && Array.isArray(req.body.products) && req.body.products.length > 0 )
      validatedCart = await OrderValidator.ValidateOrderProducts(req, res);
    else {
      // no cart products means that user have deleted all the products from the cart
      // we can allow it because the cart can be empty
      validatedCart.products = [];
      validatedCart.payment_value = 0;
    }
    // get cart details
    var cart = await Cart.findOne({ user_id: req.user._id });
    if (!cart) {
      // ifthe user does not habve a cart
      cart = new Cart({
        user_id: req.user._id,
        products: validatedCart.products,
        payment_value: validatedCart.payment_value,
      });
    } else {
      // if the user already have a cart
      cart.products = validatedCart.products;
      cart.payment_value = validatedCart.payment_value;
    }
    // save cart
    var result = await cart.save();
    if (result && result.error) return res.status(400).json(result);
    return res
      .status(200)
      .json({ message: "All changes was saved", cart: result._doc });
  } catch (error) {
    console.error(error);
    return res.status(400).json({ message: "Invalid cart details" });
  }
};

```

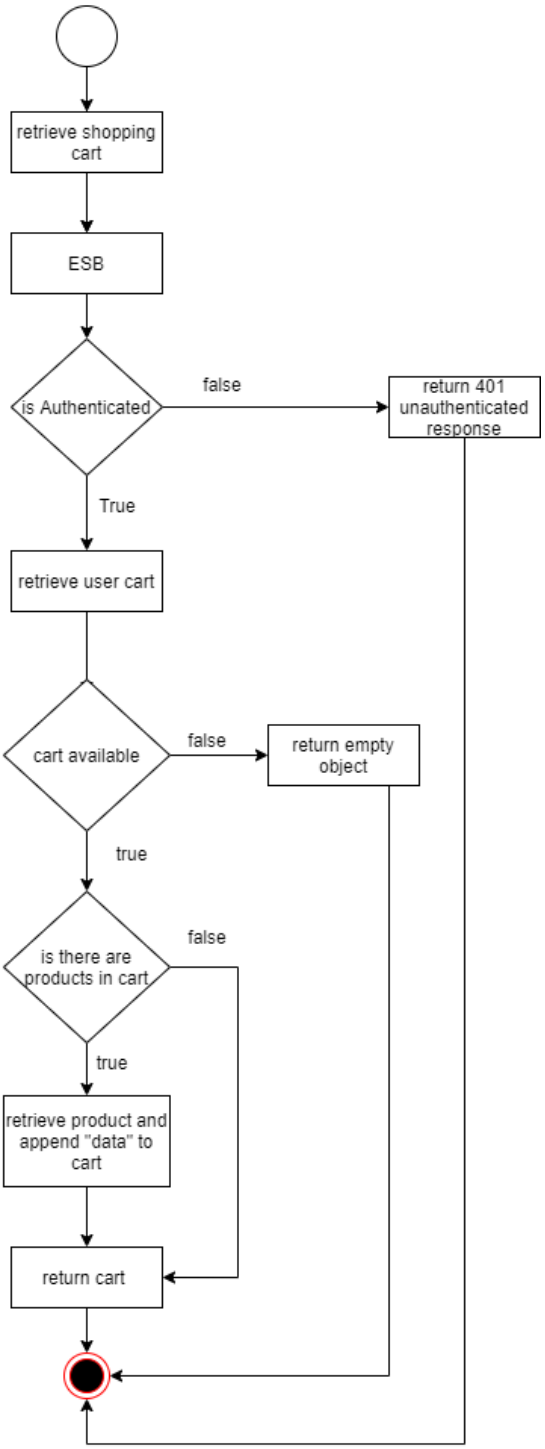
A POST request will be sent to the buyer service through ESB. (WSO2 EI).

To update product quantity user must be authenticated. Therefore the “userAuth” middleware is used to filter the authenticated users / requests . Then the request body is validated to check if it contains products. If no products were found that means user has deleted all products from cart, but if products are found then these product data must be validated for this the previous function “validateOrderProducts” will be used. If all the data are validated then retrieve the user cart using the user’s ID and then if there is a cart, then replace previous products with new products along with their quantity and update the cart payment value. Else, create a new cart by providing user’s id, products data and the total payment/cart value.

The cart will be saved/updated and if it fails then an error message will be sent else, the updated user cart object will be sent to the client through ESB.

3.12.3. Retrieve user's shopping cart

Workflow diagram



Service function

```
39
40  // get cart od the logged in user
41  exports.getCart = async (req, res) => {
42    try {
43      var cart = await Cart.findOne({ user_id: req.user._id });
44      // if the cart exist and if the cart has products, get the products rom the cart
45      if (cart && cart.products.length > 0) {
46        for (var index = 0; index < cart.products.length; index++) {
47          try {
48            // add all the product data
49            cart.products[index].data = await Product.findById(
50              cart.products[index].id
51            );
52          } catch (error) {
53            console.log(error);
54          }
55        }
56      }
57      return res.status(200).json(cart ? cart : {});
58    } catch (error) {
59      console.error(error);
60      return res.status(400).json({ message: "User cart not found" });
61    }
62  };
63
```

A GET request will be sent to the buyer service through ESB. (WSO2 EI).

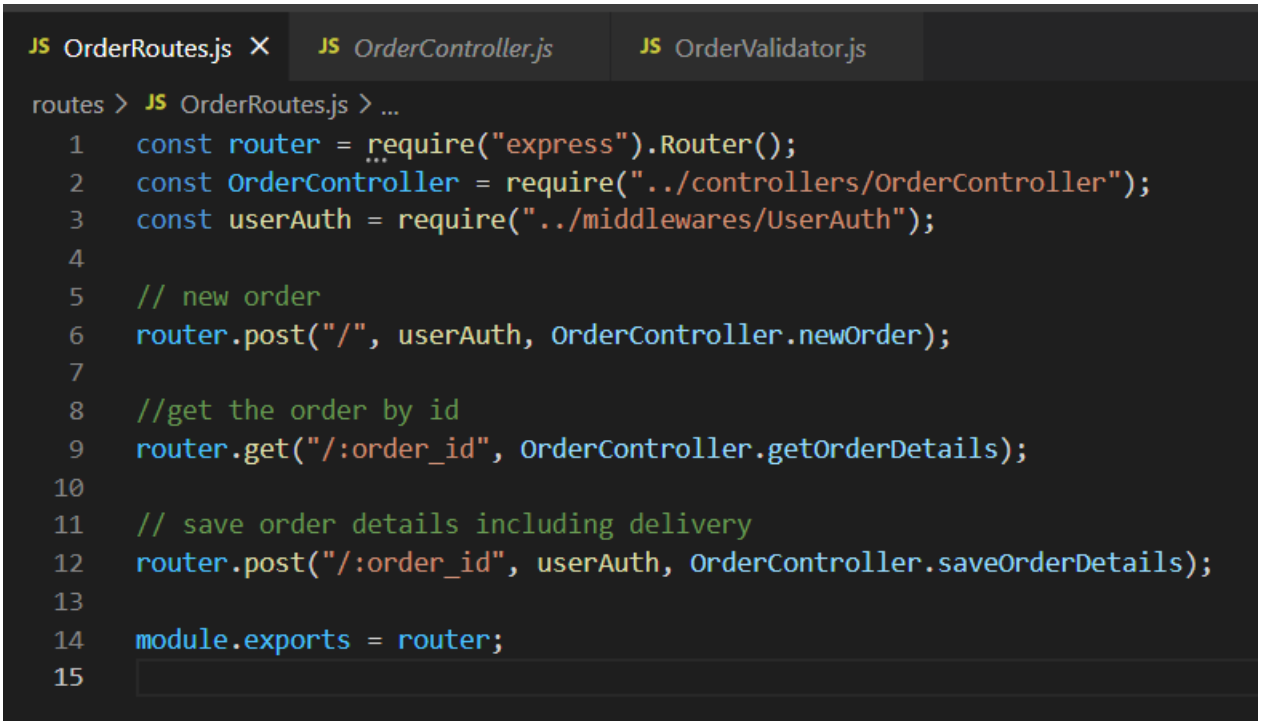
To retrieve user's shopping cart, user must be authenticated. Therefore "userAuth" middleware is used to filter the authenticated users/ requests .

User's cart will be retrieved from the database by using user id. If the cart contains products, loop through each product and retrieve it from the database using product id and append the product data to the cart product with a new property "data". This is required to display product image, category, price and other data in the frontend other than only displaying the product ID.

Return the new cart with all product data and if there is no cart found then return an empty object via ESB to the client.

3.13. Order management

Service functions / interfaces to manage orders.



There are three routes to manage orders.

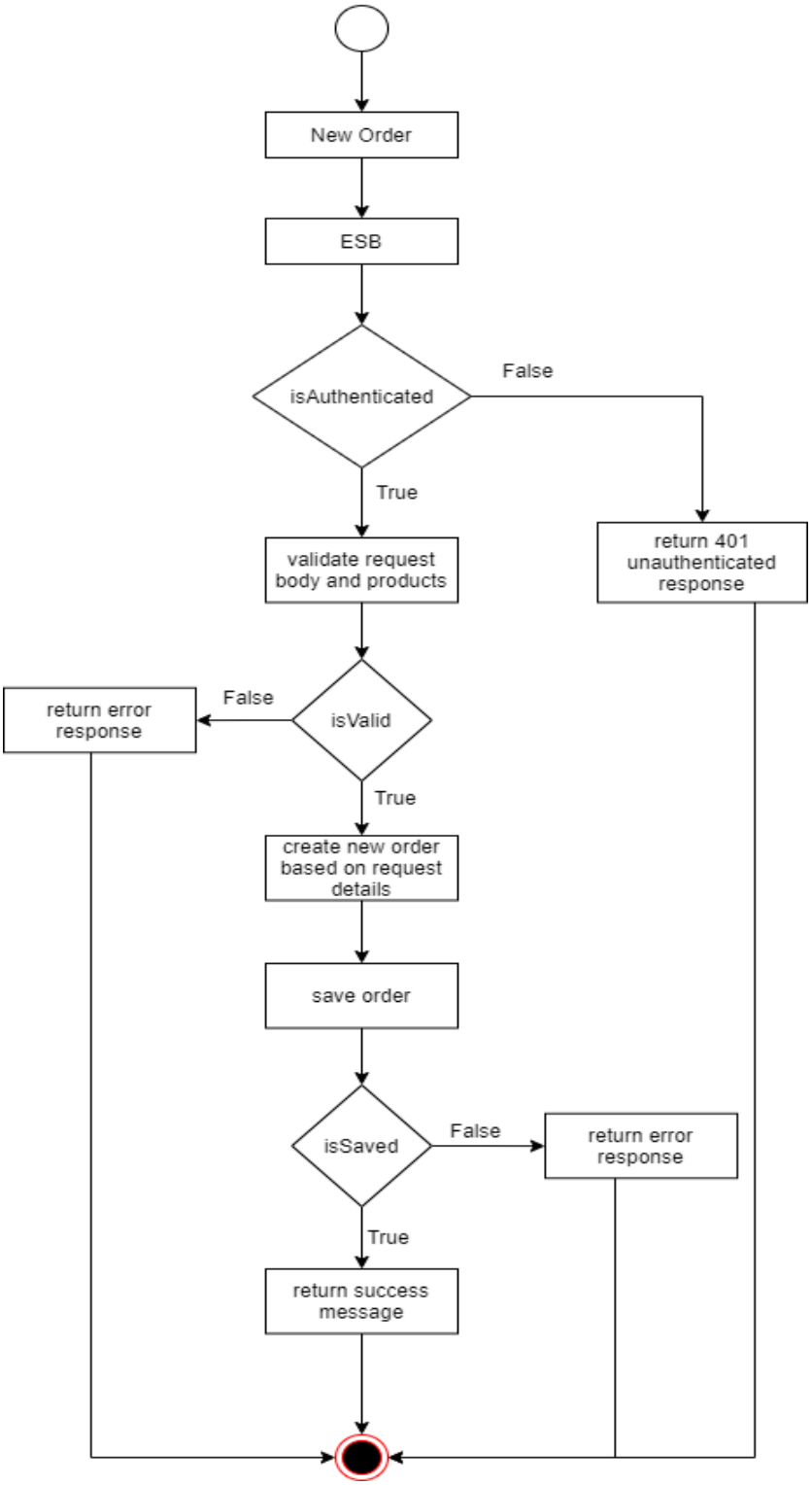
1. A POST request to create a new order with products and its quantities.
2. A GET request to retrieve an order by the order ID sent as a parameter in the request.
3. A POST request to save / update the order buyer details and delivery information.

All these routes are mapped using the WSO2 Enterprise Integrator.

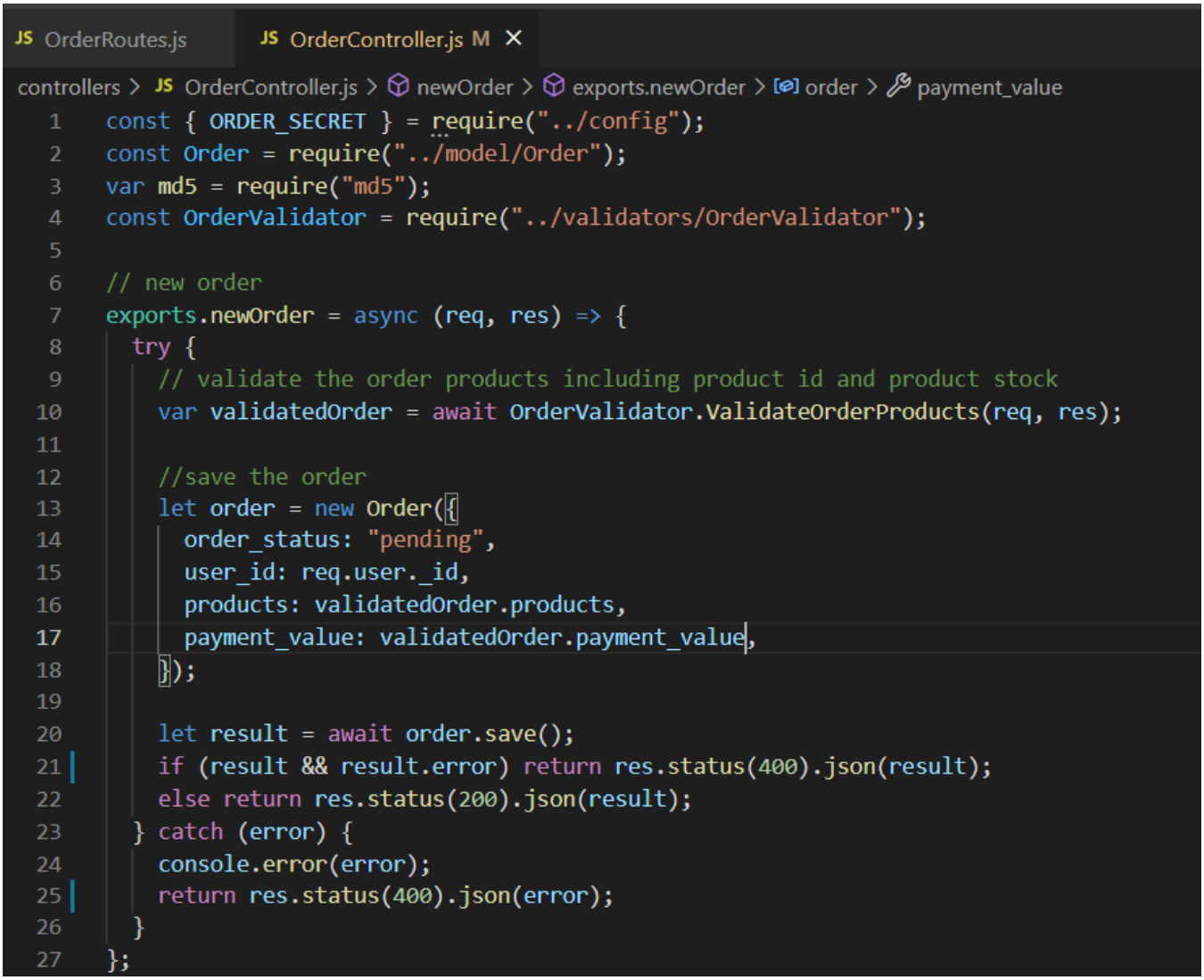
To place an order there are few steps. Initially products with their quantity will be sent to the buyer service and then it will save it and return an object with the payment value and a unique order id. User will navigate to save buyer information and delivery information through the order id. After saving the above details user will navigate for the payments. Payments can be made through cash on delivery, credit / debit card (dummy service) or though mobile service provider (dummy service). After a successful payment user will get and email (through Gmail) and a SMS (though Twillio).

3.13.1. Process of creating a new order

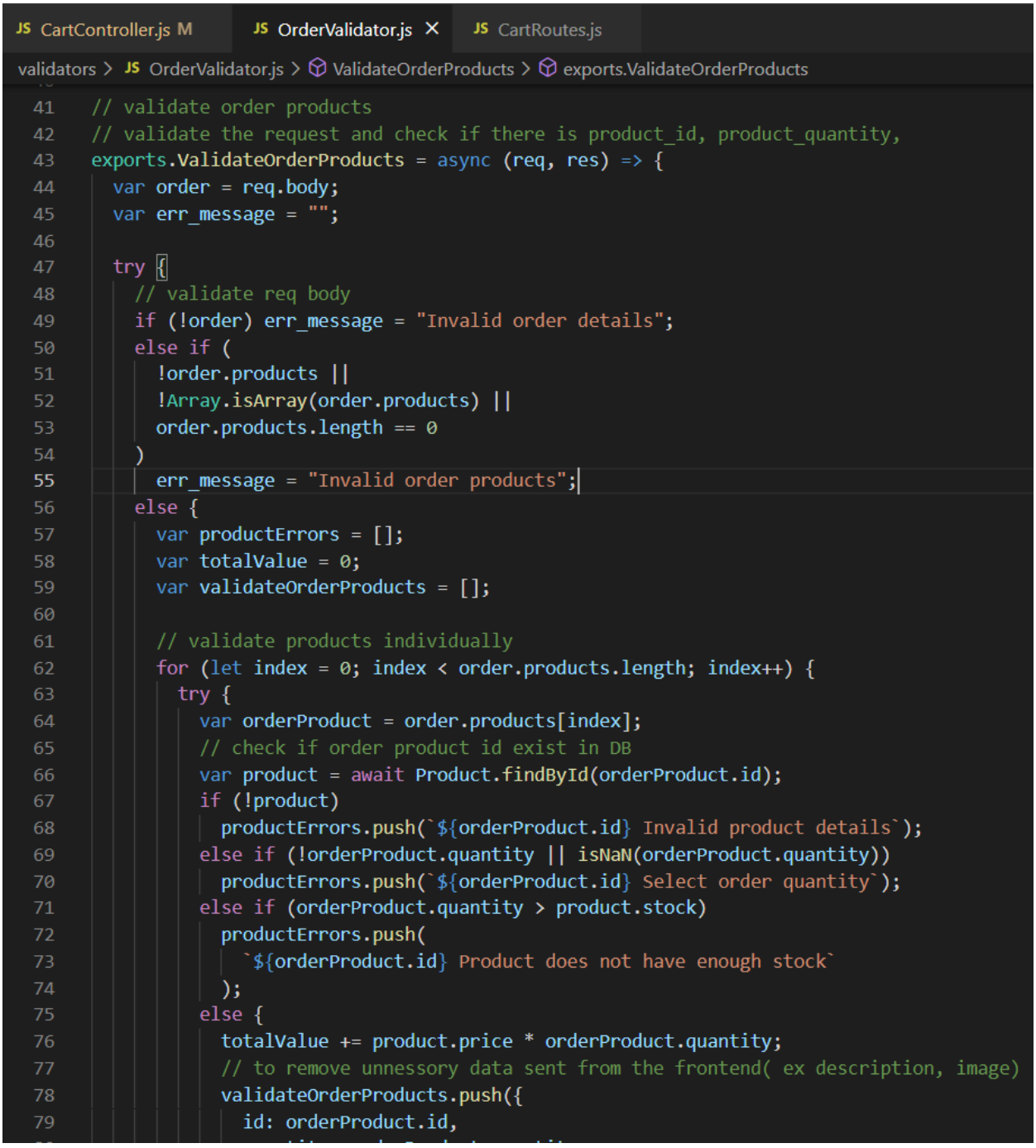
Workflow diagram



Service function



Validator function to validate request body such as valid products, products have stock (validateOrderProducts)



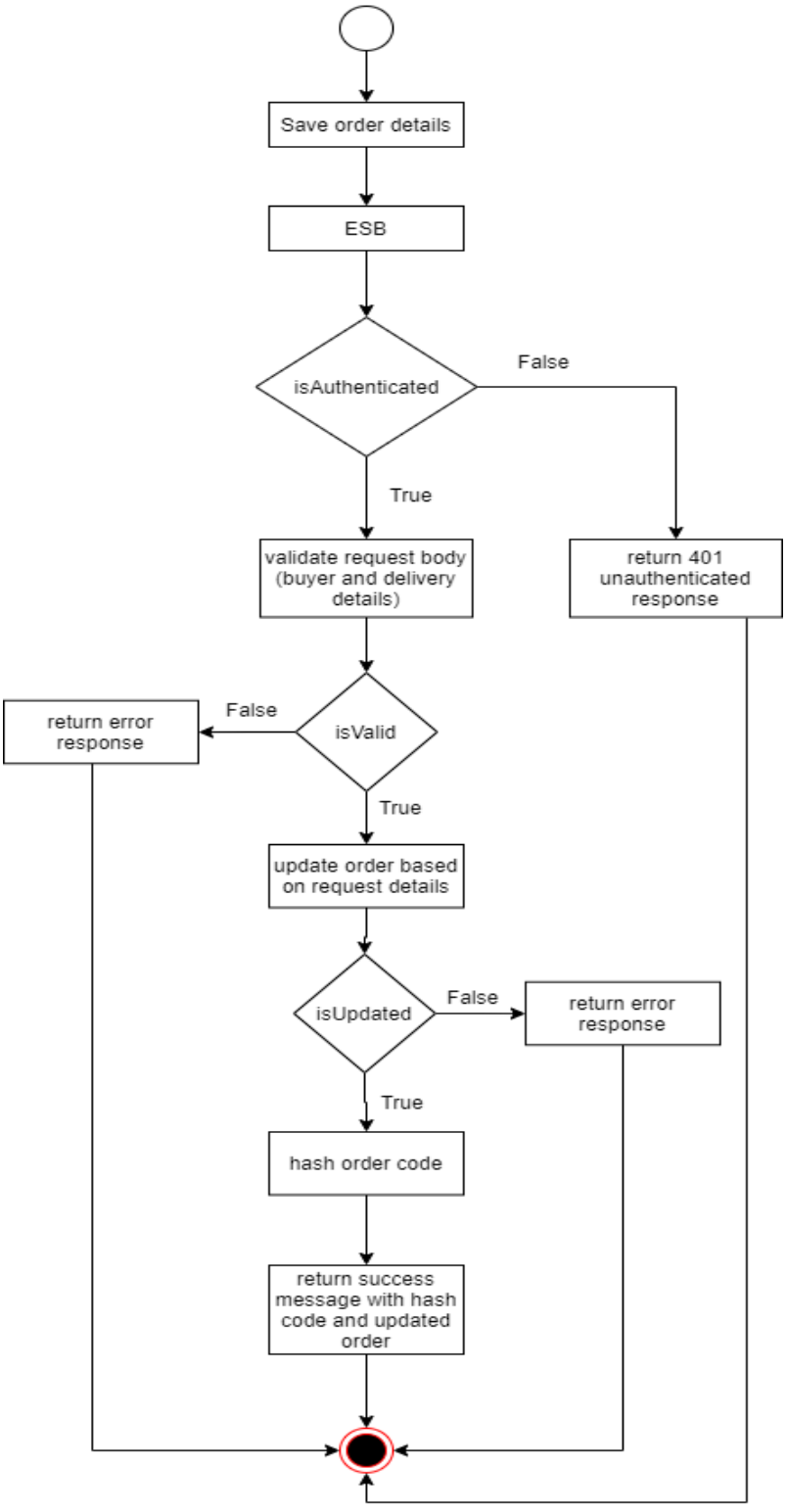
```
            else {
              totalValue += product.price * orderProduct.quantity;
              // to remove unnessory data sent from the frontend( ex description, image)
              validateOrderProducts.push({
                id: orderProduct.id,
                quantity: orderProduct.quantity,
              });
            }
          } catch (error) {
            productErrors.push(`${orderProduct.id} Invalid product details`);
          }
        }

        if (productErrors.length > 0)
          return res
            .status(422)
            .json({ message: "There were errors in products", productErrors });
        else {
          // set order payment and validated order products
          order.payment_value = totalValue;
          order.products = validateOrderProducts;
        }
      }
      if (err_message.length > 0)
        return res.status(422).json({ message: err_message });
      else return order;
    } catch (error) {
      return res.status(422).json({ message: "There were errors in products" });
    }
  };
```

A POST request will be sent to the buyer service through ESB. (WSO2 EI). To create a new order, user must be authenticated. Therefore the “userAuth” middleware is used to filter the authenticated users/ requests. Request body must have an array of objects where each object must have the product id and its quantity. This request is validated using the above “validateOrderProducts” function. This function will iterate the array of objects and check if each object has a product id and then it checks if there is an existing product in the database with the product id in the object. Additionally, it validates that the product has enough stock to supply. If any of the validations fail, respective error messages will be sent back to the client through ESB. Else, an object with the validated products array and total payment value (product price * quantity) will be returned.

A new order will be created by setting the user’s id, products and payment value fields. The order status will be set to “pending”. This order object is saved and if it fails to save an error message is sent back to the client through ESB. Else, the saved object will be returned back to the client through ESB which will be required to save the delivery details and payment.

3.13.2. Process of Saving / Updating buyer details and delivery in order

Workflow diagram



Service function

```
39
40  // after the order is created user can save the buyer info and the delivery info
41  exports.saveOrderDetails = async (req, res) => {
42    try {
43      // validate the requset
44      var validatedOrder = await OrderValidator.ValidateOrderDetails(req, res);
45
46      // get the order
47      var order = await Order.findById(req.params.order_id);
48
49      order.buyer_name = validatedOrder.buyer_name;
50      order.buyer_email = validatedOrder.buyer_email;
51      order.buyer_phone = validatedOrder.buyer_phone;
52      order.delivery_type = validatedOrder.delivery_type;
53      order.delivery_address = validatedOrder.delivery_address;
54
55      // save details
56      var result = await order.save();
57      if (result && result.error) return res.status(400).json(result.error);
58      else {
59        // hash parameters of the order to confirm payment details from gateways
60        var hash_order_code = md5(
61          `${ORDER_SECRET}${order._id}${order.payment_value}`
62        );
63        result = { ...result._doc, hash_order_code };
64        return res.status(200).json(result);
65      }
66    } catch (error) {
67      console.error(error);
68      return res.status(400).json({ message: "Order not found" });
69    }
70  };
71
```

Function “ValidateOrderDetails” will be used to validate buyer details and delivery information.

validators > JS OrderValidator.js > ValidateOrderDetails > exports.ValidateOrderDetails

```
// validate the request and check if there is user_id,
//buyer_email, buyer_name, buyer_phone, delivery_type and address
exports.ValidateOrderDetails = async (req, res) => {
  var mailformat = /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/;
  var order = req.body;
  var err_message = "";

  if (!order) err_message = "Invalid order";
  else if (order.user_id !== req.user._id)
    err_message = "This order does not belongs to you";
  else if (order.order_status !== "pending")
    err_message =
      "User cannot modify order details at this stage please contact our customer service";
  else if (!order.buyer_name || order.buyer_name.length == 0)
    err_message = "Buyer name is required";
  else if (!order.buyer_email || order.buyer_email.length == 0)
    err_message = "Buyer email is required";
  else if (!order.buyer_email.match(mailformat))
    err_message = "Email format is invalid";
  else if (!order.buyer_phone) err_message = "Buyer contact number is required";
  else if (isNaN(order.buyer_phone))
    err_message = "Mobile number contains invalid characters";
  else if (order.buyer_phone.toString().length != 9)
    err_message =
      "Mobile number must only have 9 characters without the initial 0";
  else if (!order.delivery_type) err_message = "Select delivery type";
  else if (
    !(order.delivery_type == "pickup" || order.delivery_type == "delivery")
  )
    err_message = "Selected delivery type is not valid";
  else if (order.delivery_type == "delivery" && !order.delivery_address)
    err_message = "Enter delivery address";
  else if (order.delivery_type == "pickup") order.delivery_address = "";

  if (err_message.length > 0)
    return res.status(422).json({ message: err_message });
  else return order;
};
```

The initial POST request will be sent to the main application server through ESB. (WSO2 EI). To save / update the order, user must be authenticated. Therefore the “userAuth” middleware is used to filter the authenticated users/ requests. Request body must have the order id, buyer information such as buyer name, contact number and email; also, it must contain the delivery type mentioned. If the order needs to be delivered then delivery address field must not be empty. Delivery type can only be one of pick up or delivery. Mobile number must be a valid nine digit (Integer) number and the user email must be a valid email. All the above are validated using the

function “ValidateOrderDetails” and if errors are found then respective error messages will be sent back to the client through ESB. Else, the validate request body will be returned.

Order object will be retrieved using the order id. If any error is occurred, an error message will be sent back to the client with status code 422 through the ESB.

The order object will be updated using the validated order information and saved to the database.

If it fails to update an error message will be sent back to the client.

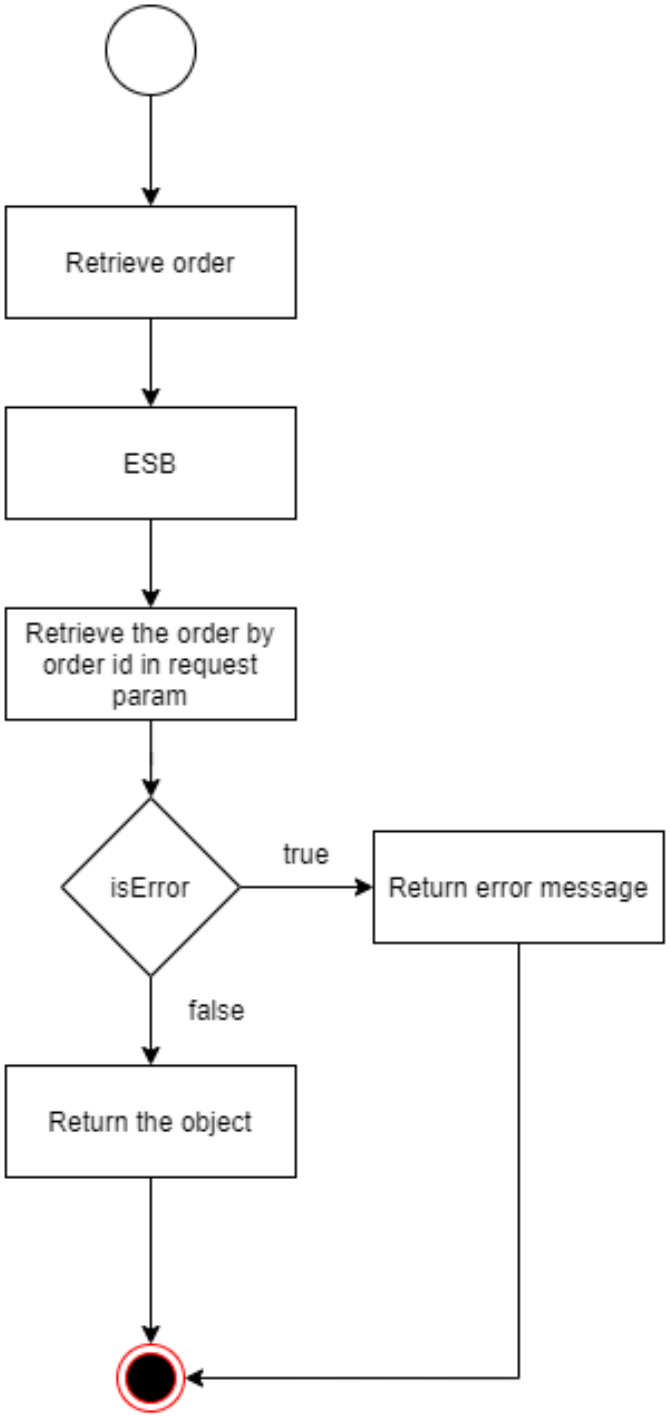
After successfully saving the order, user can make the payment. To make the payment user can make it through credit debit card, mobile service provider or through cash on delivery. If the user select credit card payment or mobile payment, user will navigate to the respective payment gateway to make the payment. Payment gateways will accept the order id, transfer amount along with the credentials it require.

To provide extra security when validating the order id and transfer by the payment gateways, an md5 encrypted string with the order secret (environment variable), order id and transfer amount will be sent. This hashed payment code is generated and the updated order object is sent to the client through ESB.

```
else {
  // hash parameters of the order to confirm payment details from gateways
  var hash_order_code = md5(
    `${ORDER_SECRET}${order._id}${order.payment_value}`
  );
  result = { ...result._doc, hash_order_code };
  return res.status(200).json(result);
}
```

3.13.3. Process of retrieving order details

Workflow diagram



Service function

```
28
29  // retrive the order tails based on id
30  exports.getOrderDetails = async (req, res) => {
31    try {
32      var order = await Order.findById(req.params.order_id);
33      return res.status(200).json(order);
34    } catch (error) {
35      console.error(error);
36      return res.status(400).json({ message: "Order not found" });
37    }
38  };
```

A GET request will be sent to the buyer service through ESB. The order id is sent as a parameter and the order is retrieved using the order id. The retrieved order will be sent back to the client through the ESB. If an error is occurred then a response is sent back to the client with 422 status code.

3.14. Processing payments

To process payments request must have a parameter with the payment type. Based on this parameter user will be route to the required service. There are three payment types, cash on delivery, credit card payment and mobile payment.

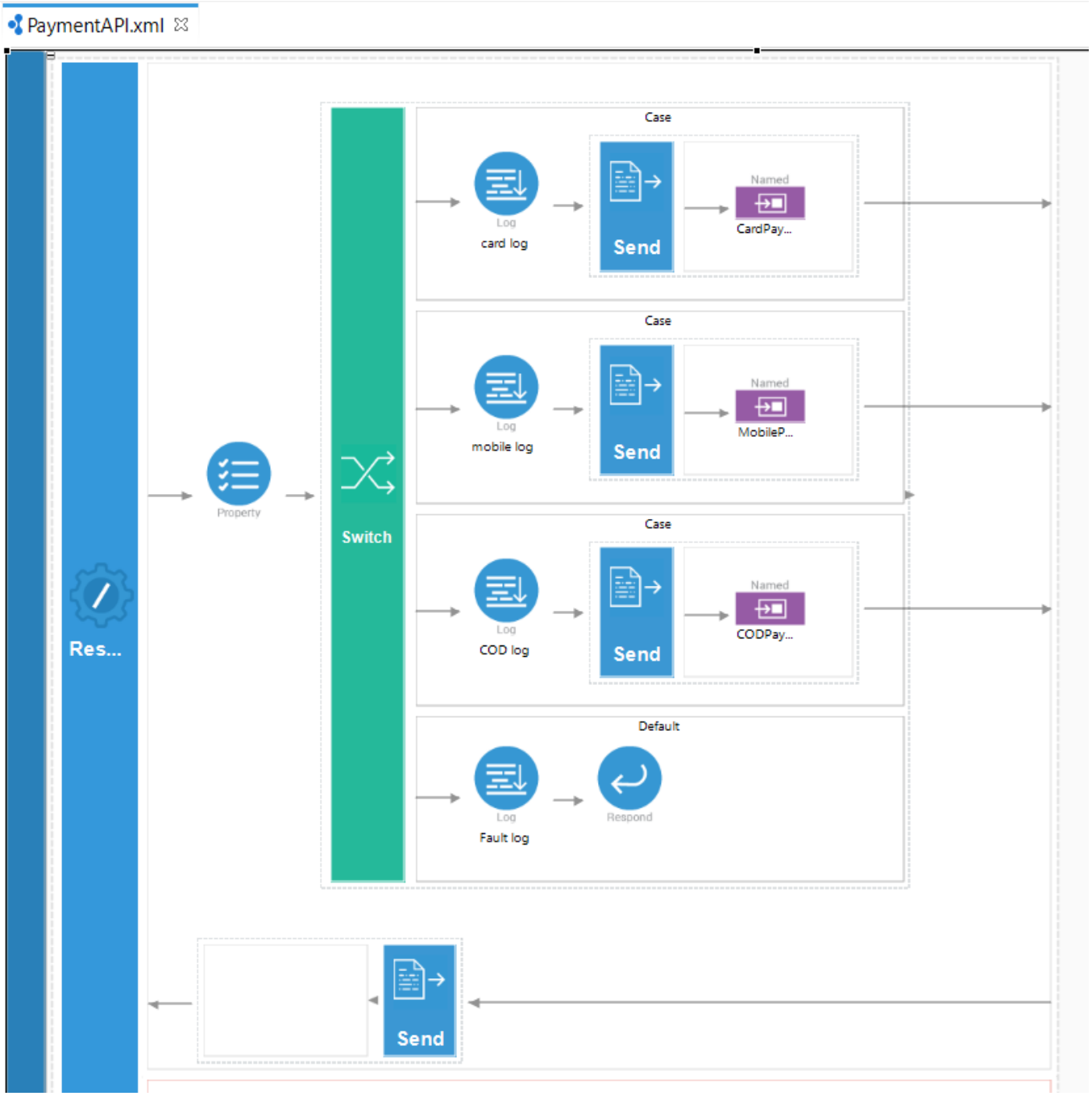
If the payment type parameter is “COD”, the ESB will route the request to the buyer service to process the cash on delivery payment.

If the payment type parameter is “bank”, the ESB will route the request to the credit card payment gateway service to process the payment.

If the payment type parameter is “mobile”, the ESB will route the request to the mobile payment gateway service to process the payment.

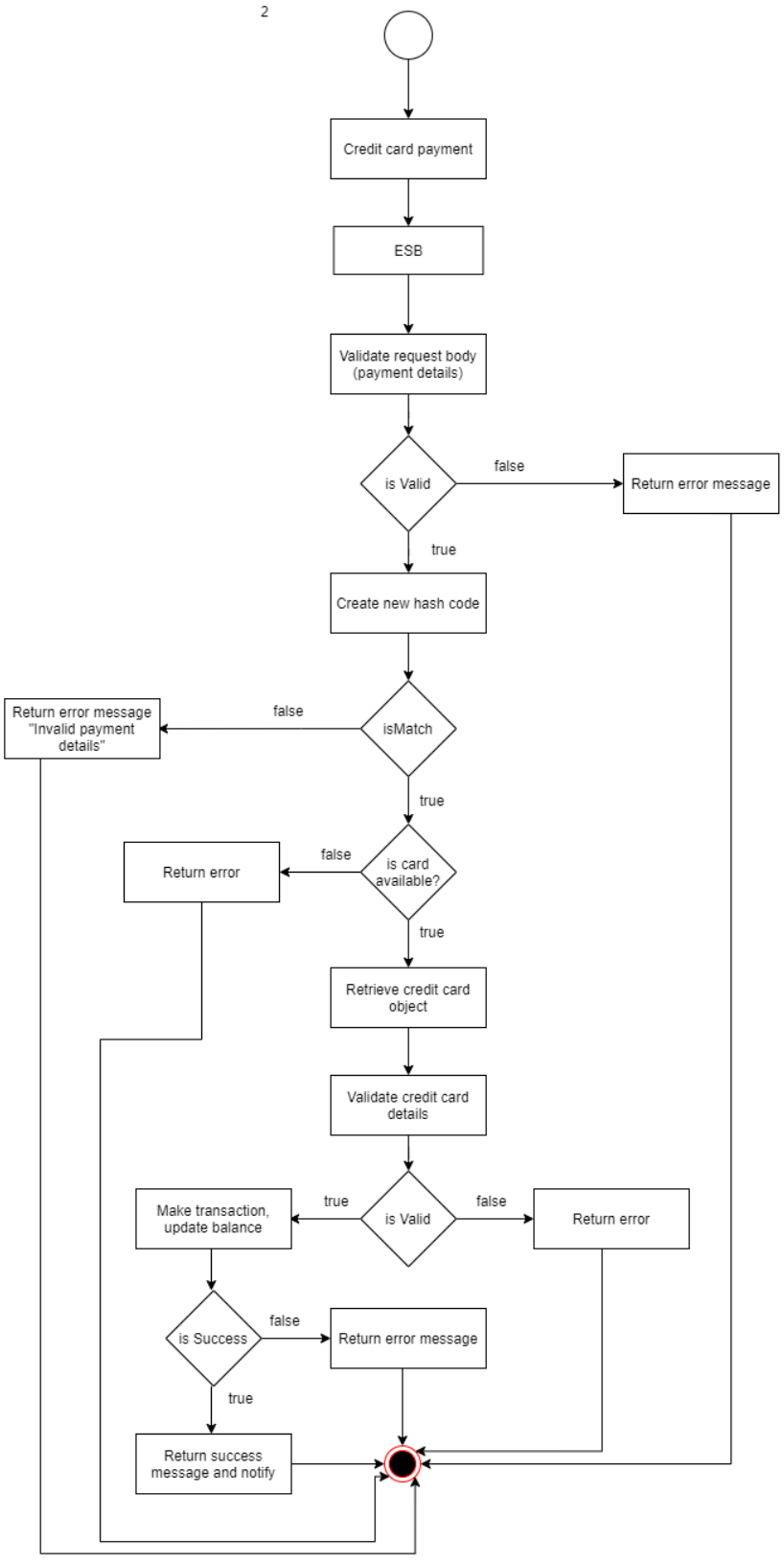
PaymentAPI.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<api context="/payment" name="PaymentAPI" xmlns="http://ws.apache.org/ns/synapse">
    <resource methods="POST" uri-template="/">
        <inSequence>
            <property expression="json-eval($.payment_type)" name="Payment_Type" scope="default" type="STRING"/>
            <switch source="get-property('Payment_Type')">
                <case regex="card">
                    <log description="card log">
                        <property expression="fn:concat('Routing to ', get-property('Payment_Type'))" name="message"/>
                    </log>
                    <send>
                        <endpoint key="CardPayment"/>
                    </send>
                </case>
                <case regex="mobile">
                    <log description="mobile log">
                        <property expression="fn:concat('Routing to ', get-property('Payment_Type'))" name="message"/>
                    </log>
                    <send>
                        <endpoint key="MobilePayment"/>
                    </send>
                </case>
                <case regex="COD">
                    <log description="COD log">
                        <property expression="fn:concat('Routing to ', get-property('Payment_Type'))" name="message"/>
                    </log>
                    <send>
                        <endpoint key="CODPayment"/>
                    </send>
                </case>
                <default>
                    <log description="Fault log">
                        <property expression="fn:concat('Invalid payment type - ', get-property('Payment_Type'))" name="message"/>
                    </log>
                    <respond/>
                </default>
            </switch>
        </inSequence>
        <outSequence>
            <send/>
        </outSequence>
        <faultSequence/>
    </resource>
    <resource methods="POST" uri-template="/gateway/mobile/request-pin/{number}">
```


PaymentAPI.xml
Res...
Property
Switch
Case
Log
card log
Send
Named
CardPay...
Case
Log
mobile log
Send
Named
MobileP...
Case
Log
COD log
Send
Named
CODPay...
Default
Log
Fault log
Respond
Send

3.14.1. Process of payment through credit card

Workflow diagram

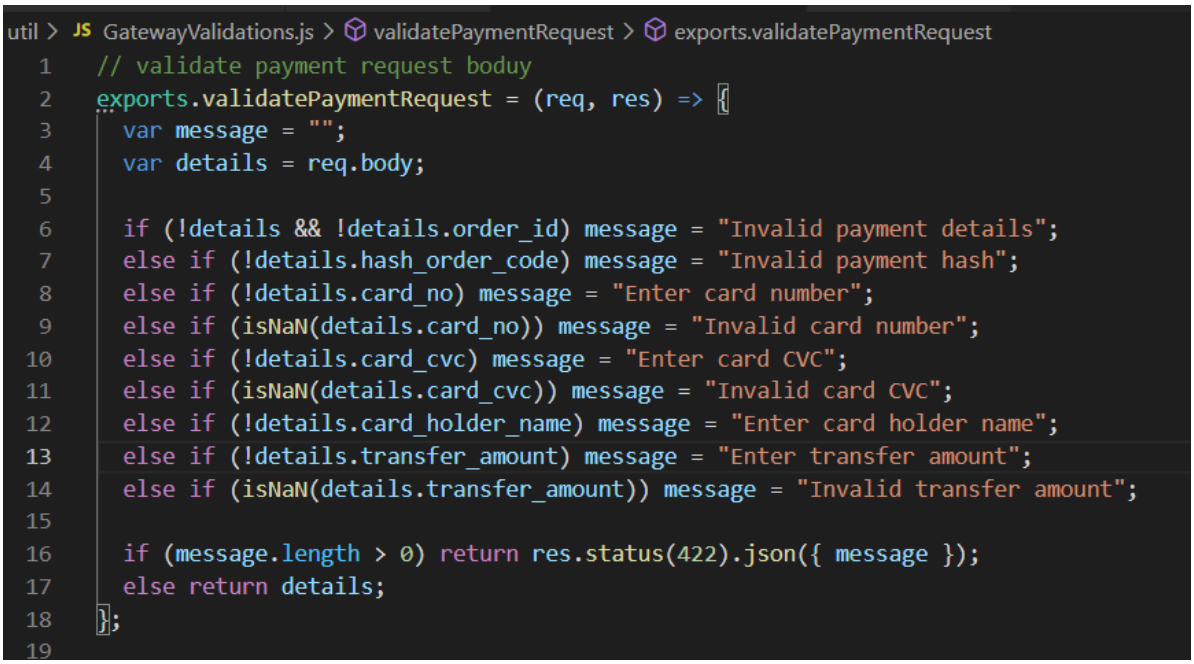


Service interface

Service function

```
13
14  exports.makePayment = async (req, res) => {
15    try {
16      // validate request body
17      var validatedDetails = validatePaymentRequest(req, res);
18      // hash paras and validate if the payment details are valid
19      var hash_order_code_valid = md5(
20        `${ORDER_SECRET}${req.body.order_id}${req.body.transfer_amount}`
21      );
22      //check if payment details and hash matches
23      if (hash_order_code_valid == req.body.hash_order_code) {
24        // get card details
25        var card = await Card.findOne({ card_no: req.body.card_no });
26
27        // match credit card details
28        var transferInfo = matchCardDetails(card, validatedDetails, res);
29        // complete transaction
30        card.balance -= transferInfo.transfer_amount;
31        var result = await card.save();
32
33        // save failed
34        if (result && result.error) return res.status(400).json(result.error);
35        //payment complted therefour notify main server payment complted
36        notifyServer(transferInfo.order_id, transferInfo.transfer_amount);
37        return res.status(200).json({
38          payment: "Payment was successfull",
39          status: 1,
40        });
41      }
42      // if hashing fails
43      return res
44        .status(400)
45        .json({ message: "Invalid payment details, refresh and try again" });
46    } catch (error) {
47      console.log(error);
48      return res.status(400).json({ message: "Invalid card number" });
49    }
50  };
```

A POST request will be sent to the credit card payment gateway service through ESB if the request has a parameter “payment_type” with its value “card”. The request body will be validated by the “validatePaymentRequest” method to check if it has order id, order hash code, transfer amount, credit card number, CVC number and a card holder name.



If the validation fail an error response with the respective message will be sent back to the client via ESB. Else, the order hash will be validated. To validate the hash sent from the request, a new hash will be created by using the order secret string constant along with the order id and transfer amount.

If the newly created hash string does not match with the hash code sent by the request that means the payment details does not match with the order details therefore an error response will be sent back to the client.

If hash matches credit card object will be retrieved from the database based on the credit card number. If there is no credit card object then an error response will be sent back. Else then the credit card details will be validated.

To validate credit card details “matchCardDetails” function will be used. Credit card number, CVC number and the card holder name must be matched with the request data and also the remaining balance of the account must be greater than the transfer amount.

```
// validate credit card deatils with provided details
exports.matchCardDetails = (card, req_details, res) => {
  var err_message = "";
  if (!card)
    err_message =
      "There is no credit card associated with the card no provided";
  else if (card.card_cvc != req_details.card_cvc)
    err_message = "Card CVC number did not match";
  else if (card.card_holder_name != req_details.card_holder_name)
    err_message = "Card holder name did not match";
  else if (card.balance < req_details.transfer_amount)
    err_message = "Card balance is not suffient";

  if (err_message.length > 0)
    return res.status(422).json({ message: err_message });
  else return req_details;
};

```

If the validation fail an error response with the respective message will be sent back to the client via ESB. Else the transaction will be made updating the credit card balance. If the transaction failed then an error response will be sent back. Else a success message will be sent back via ESB and the buyer service will be notified that the order was completed.

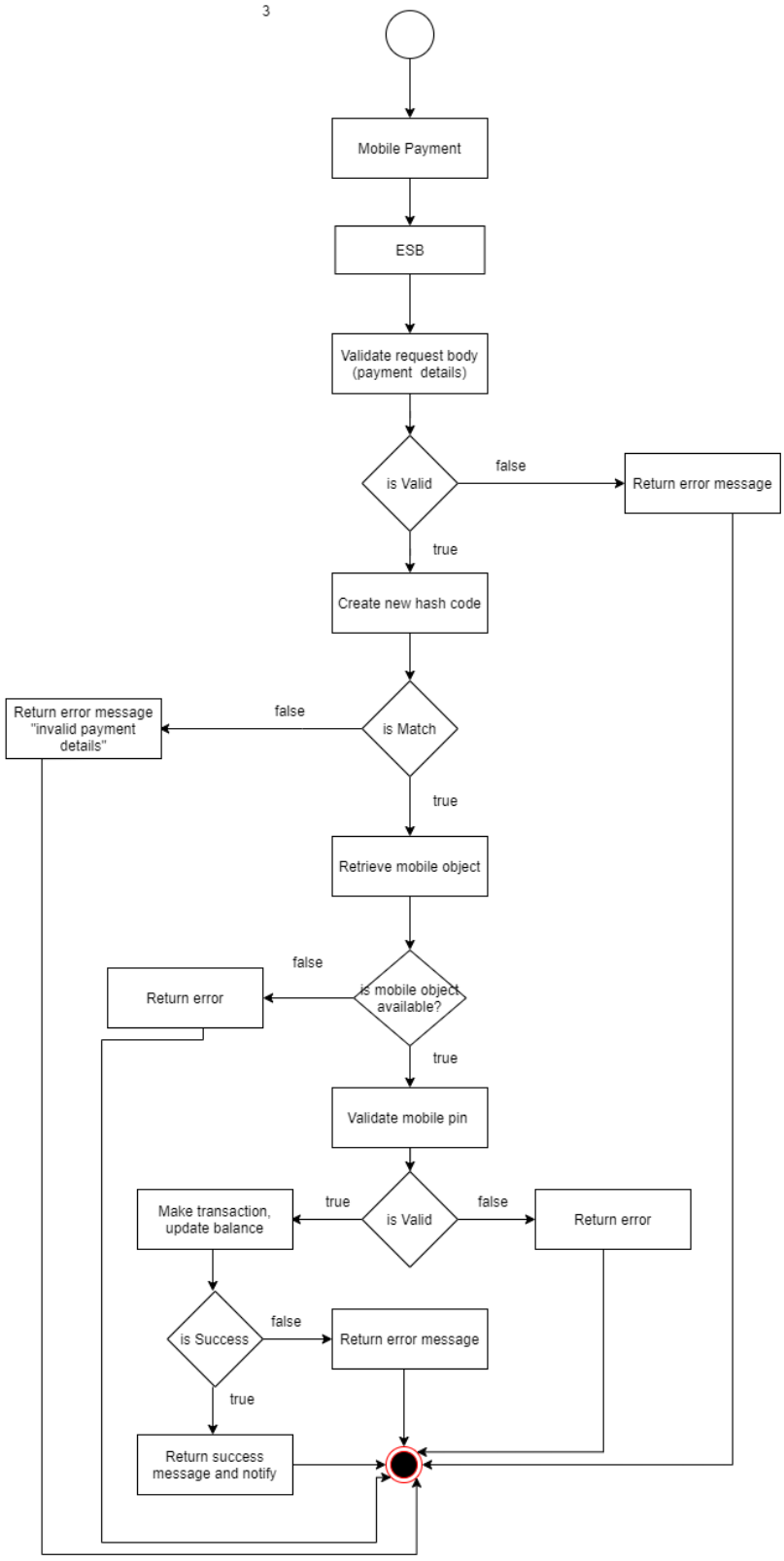
```
//payment complted therefour notify main server payment complted
notifyServer(transferInfo.order_id, transferInfo.transfer_amount);
return res.status(200).json({
  payment: "Payment was successfull",
  status: 1,
});
```

Notify server method will send a POST request to the buyer service with data such as order id, transfer amount and payment hash code. Payment hash code is generated using payment secret (string constant in the environment configuration), order id and the transfer amount. This hash code is useful to validate that the payment was completed through the payment gateway.

```
const notifyServer = (orderID, transfer_amount) => {
  // hash the payment data to validate at the buyer server(main)
  var hash_pay_code = md5(`${PAYMENT_SECRET}${orderID}${transfer_amount}`);
  axios
    .post(PAYMENT_NOTIFY_URL, {
      order_id: orderID,
      payment_type: "card",
      hash_pay_code,
    })
    .then((res) => console.log(res))
    .catch((err) => console.log(err));
};
```

3.14.2. Process of payment through mobile payment gateway

Workflow diagram



Service interface

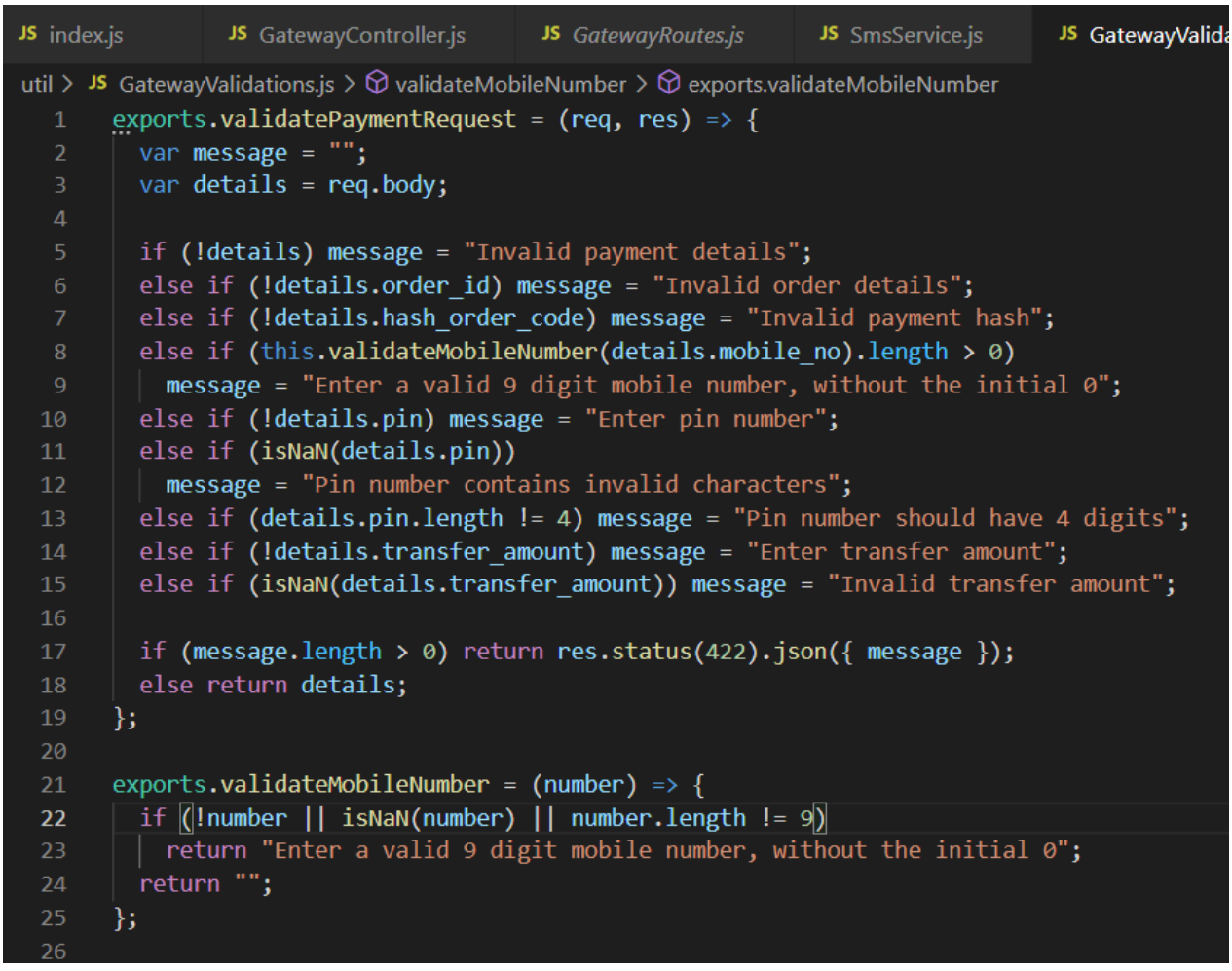
```
5    router.post("/", GatewayController.makePayment);
```

Service function

```
controllers > GatewayController.js > makePayment > exports.makePayment
15  exports.makePayment = async (req, res) => {
16    try {
17      var validatedPayment = validatePaymentRequest(req, res);
18
19      // hash paras and validate if the payment details are valid
20      var hash_order_code_valid = md5(
21        `${ORDER_SECRET}${req.body.order_id}${req.body.transfer_amount}`
22      );
23      //check if payment details and hash matches
24      if (hash_order_code_valid == req.body.hash_order_code) {
25        // get the mobile object from DB to validate
26        var mobile = await Mobile.findOne({
27          mobile_no: validatedPayment.mobile_no,
28        });
29
30        // match mobile data with request data
31        var err_message = "";
32        if (!mobile) err_message = "invalid mobile number";
33        else if (mobile.pin != req.body.pin)
34          err_message = "pin number did not match";
35        // iff data is not matched
36        if (err_message.length > 0)
37          return res.status(400).json({ message: err_message });
38
39        // if data is matched
40        // complete transaction
41        mobile.balance += Number.parseFloat(req.body.transfer_amount);
42        var result = await mobile.save();
43
44        // save failed
45        if (result && result.error) return res.status(400).json(result.error);
46
47        //payment complted therefour notify main server payment complted
48        notifyServer(req.body.order_id, req.body.transfer_amount);
49        return res.status(200).json({
50          payment: "Payment was successfull",
51          status: 1,
52        });
53      }
```

```
    // if hashing fails
    return res
      .status(400)
      .json({ message: "Invalid payment details, refresh and try again" });
  } catch (error) {
    console.log(error);
    return res.status(400).json({
      errors: { message: "This mobile number does not accept payments" },
    });
  }
};

```

A POST request will be sent to the mobile payment gateway service through ESB if the request has a parameter “payment_type” with its value “mobile”. The request body will be validated by the “validatePaymentRequest” method to check if it has order id, order hash code, transfer amount, mobile number and mobile pin number.



If the validation fail an error response with the respective message will be sent back to the client via ESB. Else, the order hash will be validated. To validate the hash sent from the request, a new hash will be created by using the order secret string constant along with the order id and transfer amount. If the newly created hash string does not match with the hash code sent by the request that means the payment details does not match with the order details therefore an error response will be sent back to the client. If hash matches then the mobile payment details will be validated.

To validate the mobile payment, mobile object will be retrieved from the database based on the mobile number. If the mobile object does not exist then an error response will be sent. Else, mobile pin will be validated. If the validation fails then an error response will be sent back. Else, the mobile balance will be updated (transfer amount will be added) and saved. If the payment fail then an error response will be sent back. Else, a success message will be sent back via ESB and the buyer service will be notified that the order was completed.

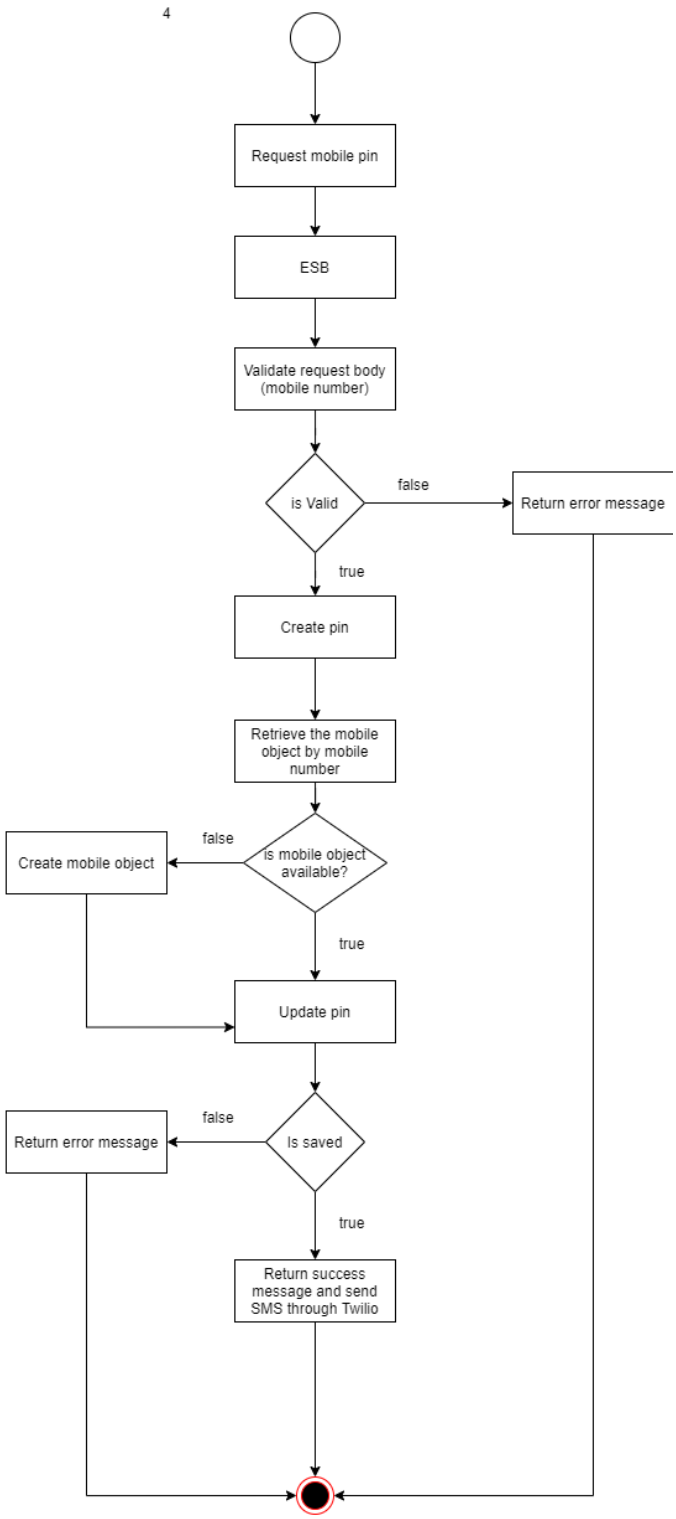
```
//payment complted therefour notify main server payment complted
notifyServer(transferInfo.order_id, transferInfo.transfer_amount);
return res.status(200).json({
  payment: "Payment was successfull",
  status: 1,
});
```

Notify server method will send a POST request to the buyer service with data such as order id, transfer amount and payment hash code. Payment hash code is generated using payment secret (string constant in the environment configuration), order id and the transfer amount. This hash code is useful to validate that the payment was completed through the payment gateway.

```
const notifyServer = (orderID, transfer_amount) => {
  // hash the payment data to validate at the buyer server(main)
  var hash_pay_code = md5(`${PAYMENT_SECRET}${orderID}${transfer_amount}`);
  axios
    .post(PAYMENT_NOTIFY_URL, {
      order_id: orderID,
      payment_type: "card",
      hash_pay_code,
    })
    .then((res) => console.log(res))
    .catch((err) => console.log(err));
};
```

3.14.3. Process of requesting mobile pin number

Workflow diagram



Service interface

```
6   router.post("/request-pin/:number", GatewayController.requestPin);
7
8   module.exports = router;
```

Service function

```
66  exports.requestPin = async (req, res) => {
67    try {
68      var phone_number = req.params.number;
69      console.log(phone_number);
70
71      // validate mobile number
72      var error = validateMobileNumber(phone_number);
73      if (error.length > 0) return res.status(400).json({ mesaage: error });
74
75      // send pin number
76      // generate random 4 digit pin
77      var pin = Math.floor(1000 + Math.random() * 9000);
78      console.log(pin);
79
80      var mobile = await Mobile.findOne({ mobile_no: phone_number });
81      if (!mobile) {
82        mobile = new Mobile({
83          mobile_no: phone_number,
84          pin,
85        });
86      } else mobile.pin = pin;
87
88      var result = await mobile.save();
89
90      // save failed
91      if (result && result.error)
92        return res
93          .status(400)
94          .json({ message: "Unexpected error please try again" });
95
96      // send the pin via sms
97      var smsOptions = {
98        to: phone_number,
99        body: `Your XMart Payment Pin is - ${pin}`,
100     };
101
102     await sendSms(smsOptions);
```

```
101
102       await sendSms(smsOptions);
103
104       return res.status(200).json({
105         message: "Pin was sent successfull",
106       });
107     } catch (error) {
108       console.log(error);
109       return res
110         .status(400)
111         .json({ errors: { message: "Invalid mobile number" } });
112     }
113   };
114
```

A POST request will be sent to the mobile payment service through ESB. The mobile number is sent as a parameter. Mobile number will be validated from the below function and if any errors were found an error response will be sent back.

```
20
21  exports.validateMobileNumber = (number) => {
22    if (!number || isNaN(number) || number.length != 9)
23      return "Enter a valid 9 digit mobile number, without the initial 0";
24    return "";
25  };
26
```

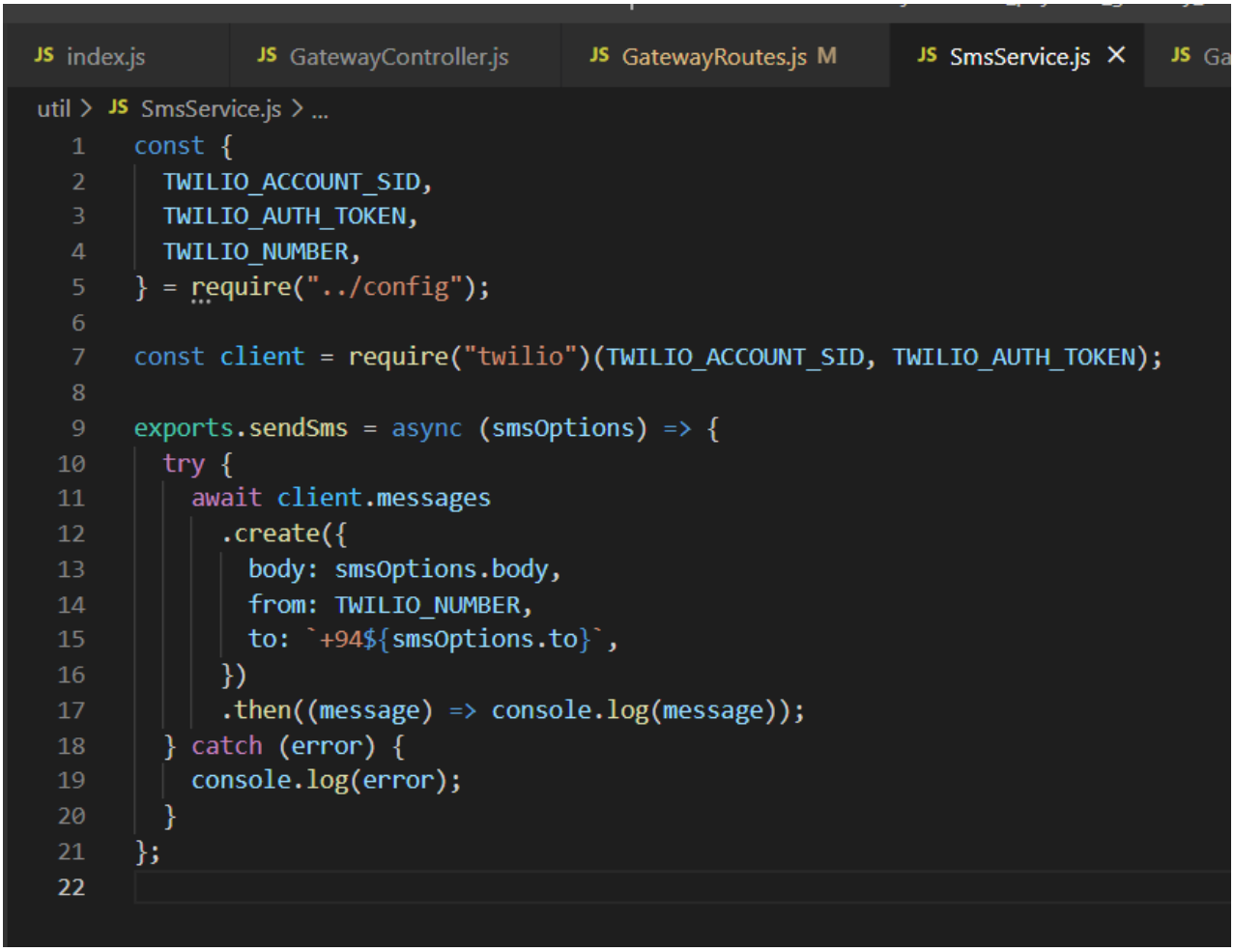
After the validation was completed successfully then a four digit pin number will be created.

The mobile object will be retrieved based on the mobile number from the database and the pin number will be updated and saved. If there is no object in the database then a new mobile object will be created and the pin number will be saved.

If an error is caught while saving an error response will be sent back to the client through the ESB. Else, a success message will be sent back and a SMS will be sent to the number with the pin number. SMS will be sent using Twilio.

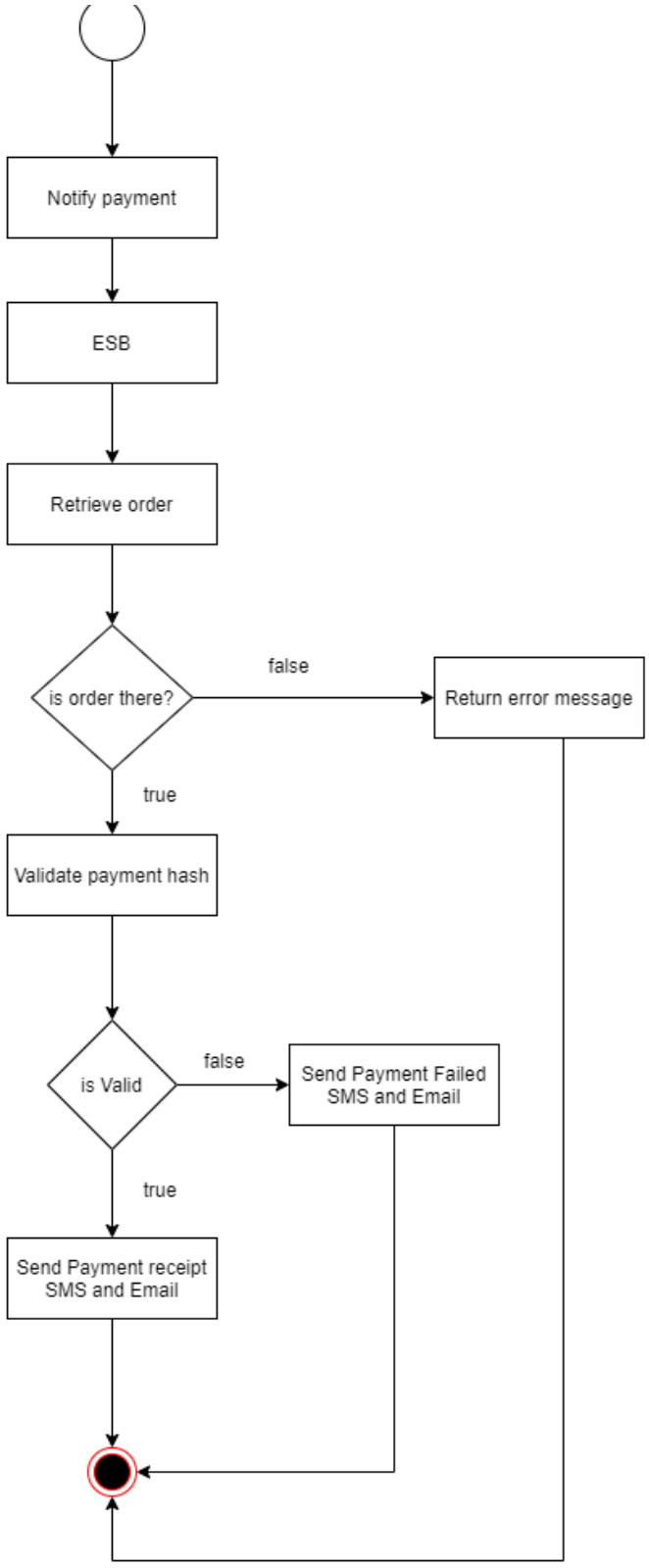
```
96    // send the pin via sms
97    var smsOptions = {
98      to: phone_number,
99      body: `Your XMart Payment Pin is - ${pin}`,
100   };
101
102   await sendSms(smsOptions);
103
104   return res.status(200).json({
105     message: "Pin was sent successfull",
106   });
```

SMS service



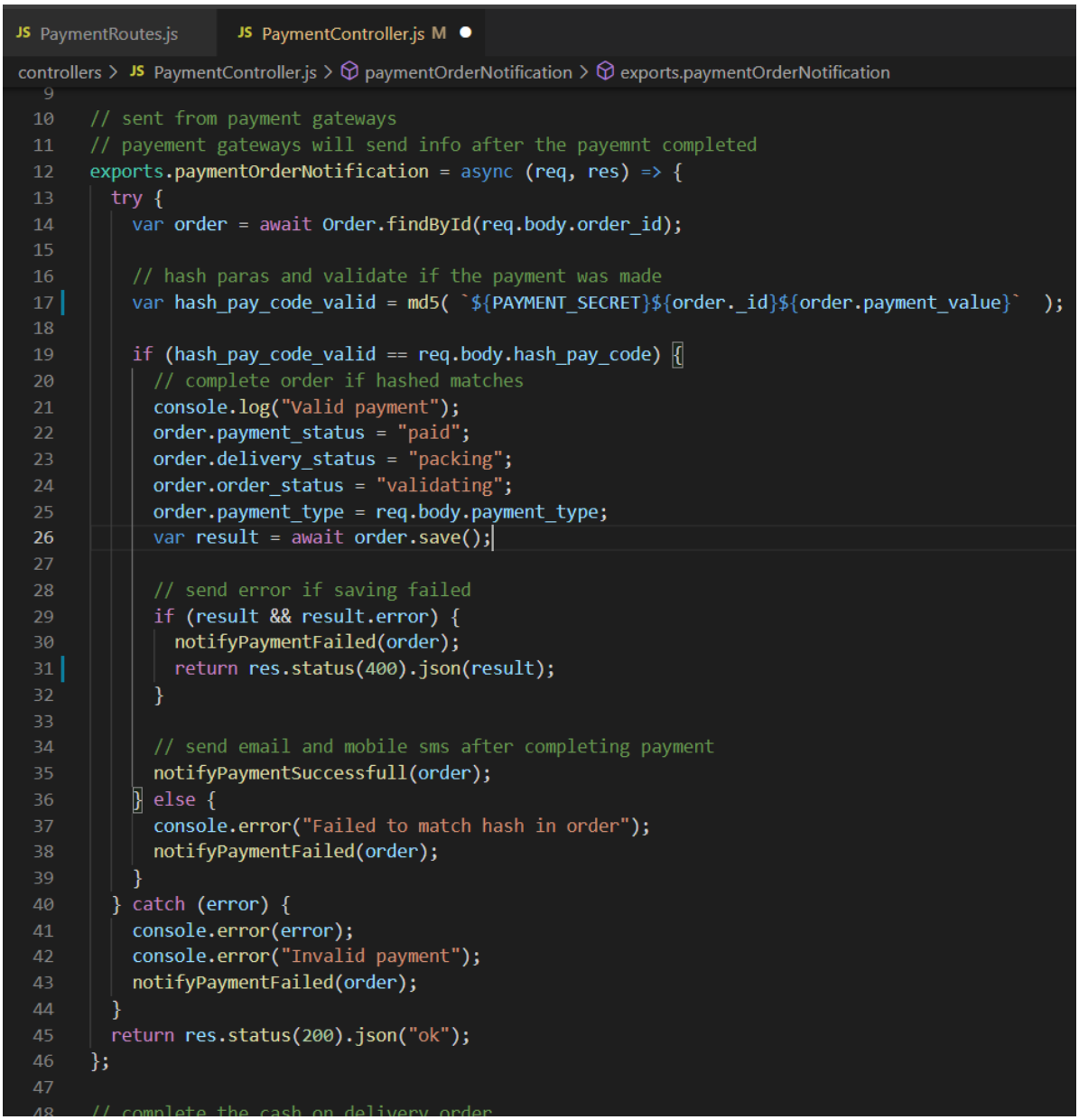
3.14.4. Process of listening to payment notifications

Workflow diagram



Service interface

Service function



After completing the payment by payment gateways (credit card / mobile) it will notify the buyer service. A POST request will be sent through the ESB to the buyer service with order id, transfer amount and the payment hash. Order will be retrieved from the order id and if there is an order then the payment hash needs to be validated.

To validate the payment hash, a new md5 hash will be created from the payment secret (Environment variable), order id and order payment value. Then if the payment hash from the request and the newly created hash from the order object matches, the payment is validated.

If the payment was not validated, “notifyPaymentFailed” method will be called and an unsuccessful email and SMS will be sent based on the order buyer information. Else, “notifyPaymentSuccessful” method will be called and a successful email and SMS will be sent based on the order buyer information.

SMS service through TWILIO

util > JS SmsService.js > ...

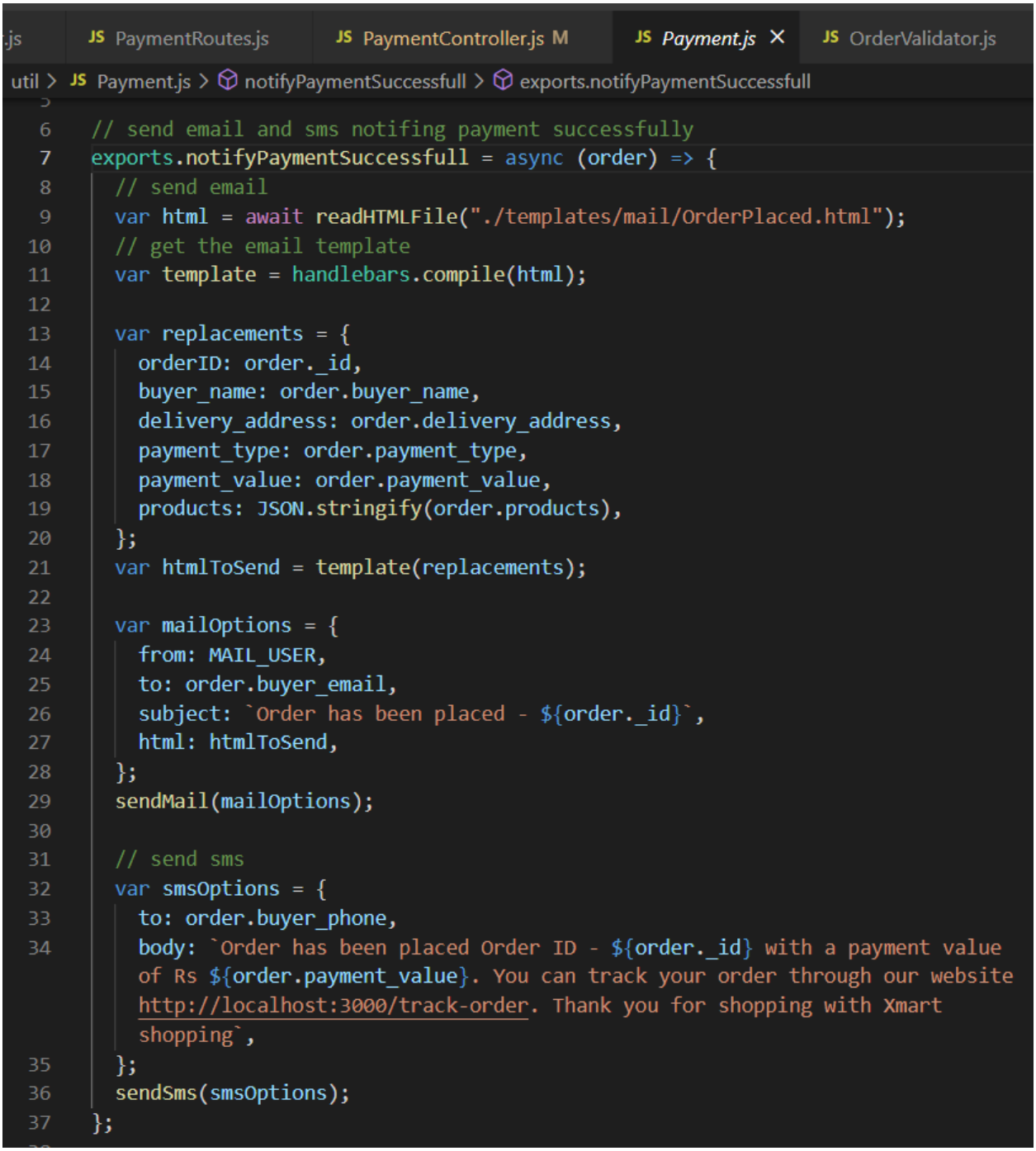
```
const {
    TWILIO_ACCOUNT_SID,
    TWILIO_AUTH_TOKEN,
    TWILIO_NUMBER,
} = require("../config");

const client = require("twilio")(TWILIO_ACCOUNT_SID, TWILIO_AUTH_TOKEN);

// send sms using twilio
exports.sendSms = (smsOptions) => {
    try {
        client.messages
            .create({
                body: smsOptions.body,
                from: TWILIO_NUMBER,
                to: `+94${smsOptions.to}`,
            })
            .then((message) => console.log(message));
    } catch (error) {
        console.log(error);
    }
};
```

Mail service through Gmail

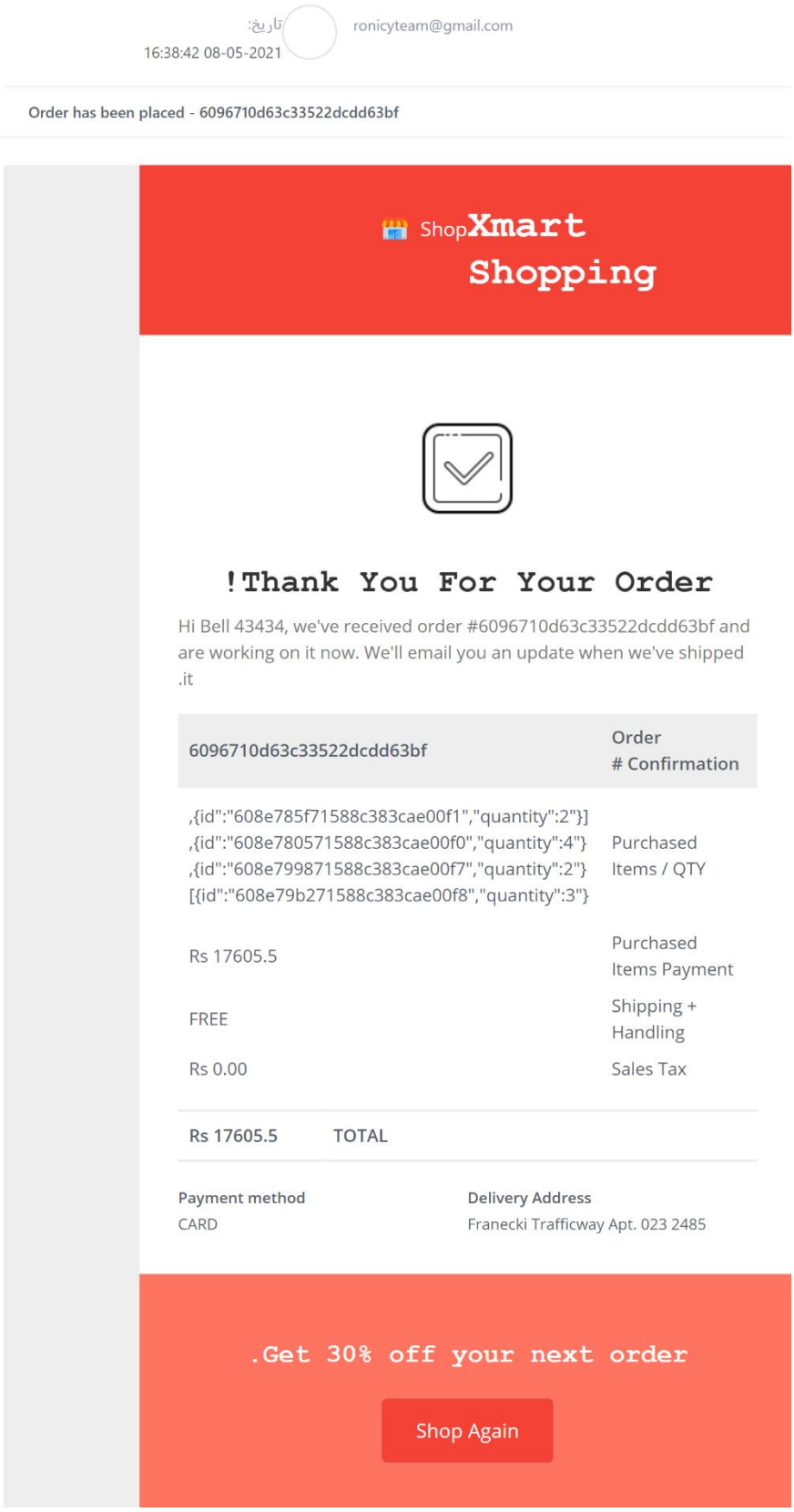
util > JS MailService.js > ...

```
var nodemailer = require("nodemailer");
var { MAIL_USER, MAIL_PASSWORD } = require("../config");
var fs = require("fs");

// send a mail using gmail
exports.sendMail = (mailOptions) => {
    var transporter = nodemailer.createTransport({
        service: "gmail",
        auth: {
            user: MAIL_USER,
            pass: MAIL_PASSWORD,
        },
    });

    transporter.sendMail(mailOptions, function(error, info) {
        if (error) {
            console.log(error);
        } else {
            console.log("Email sent: " + info.response);
        }
    });
};

// read a .html file and get its data
exports.readHTMLFile = async(path) => {
    try {
        var data = fs.readFileSync(path, { encoding: "utf-8" });
        return data;
    } catch (error) {
        console.error(error);
        return null;
    }
};
```

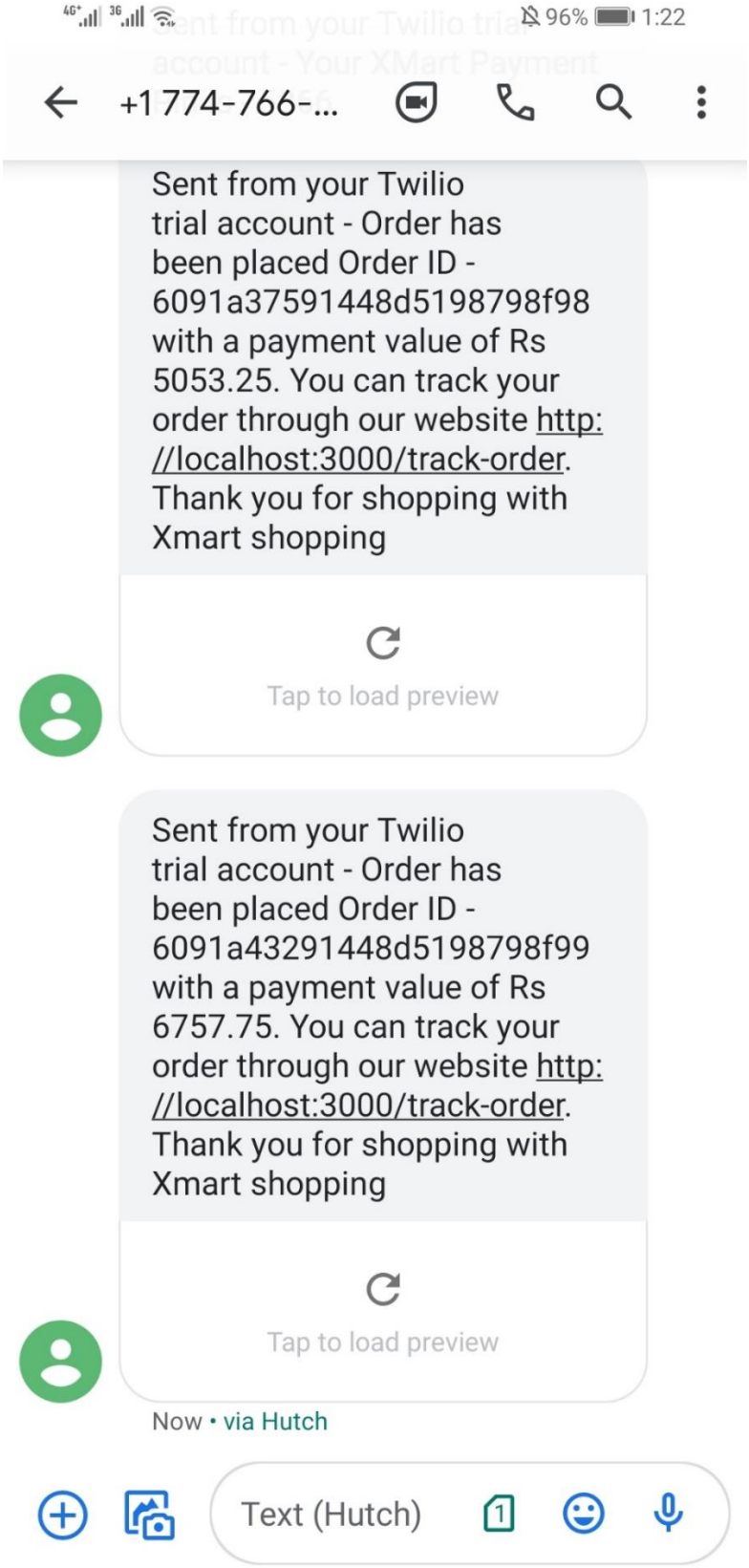
Notify payment completed through email and SMS



Email template with data sent through Gmail



SMS sent through Twilio

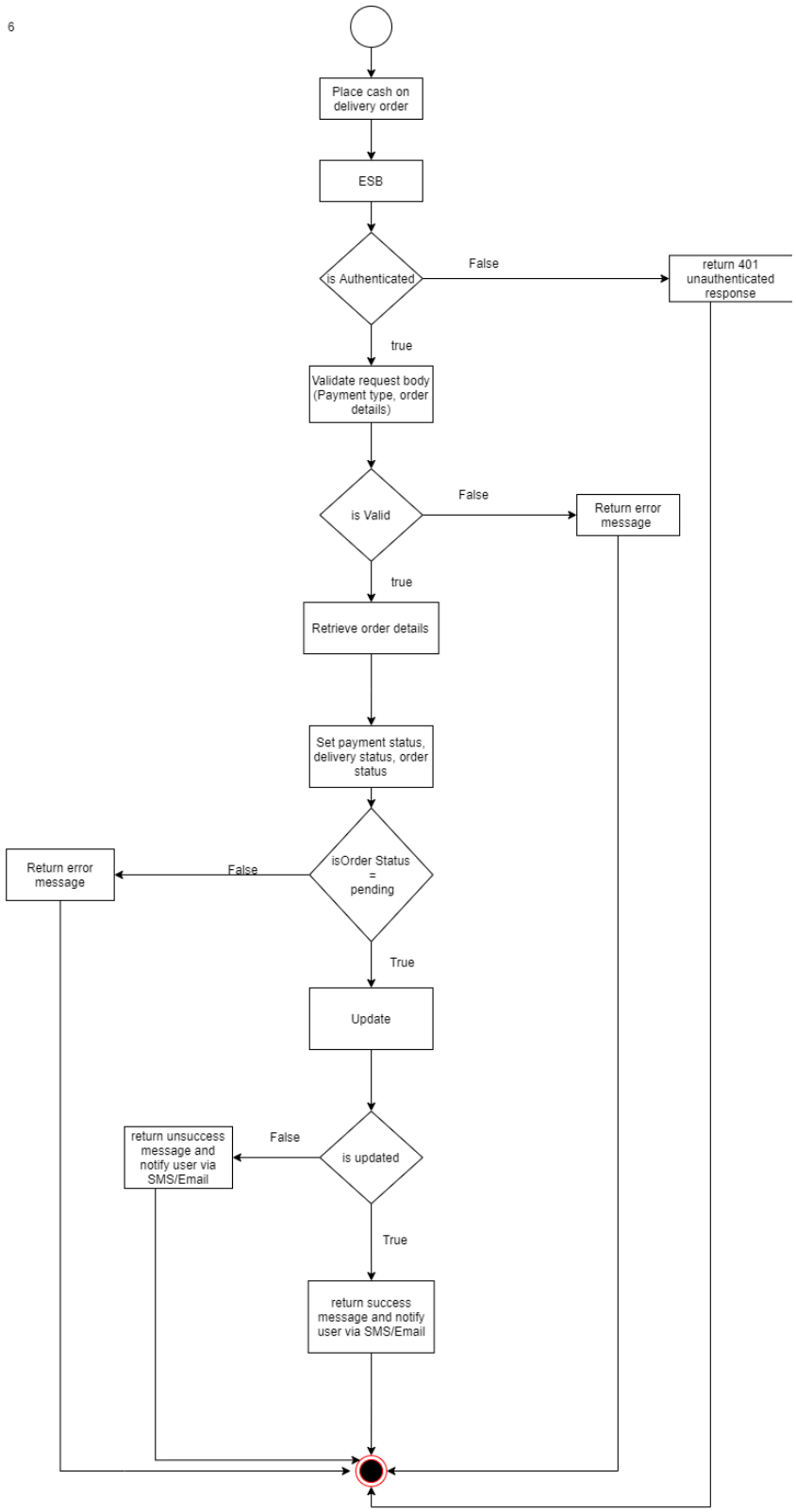


Notify payment failed through SMS and email

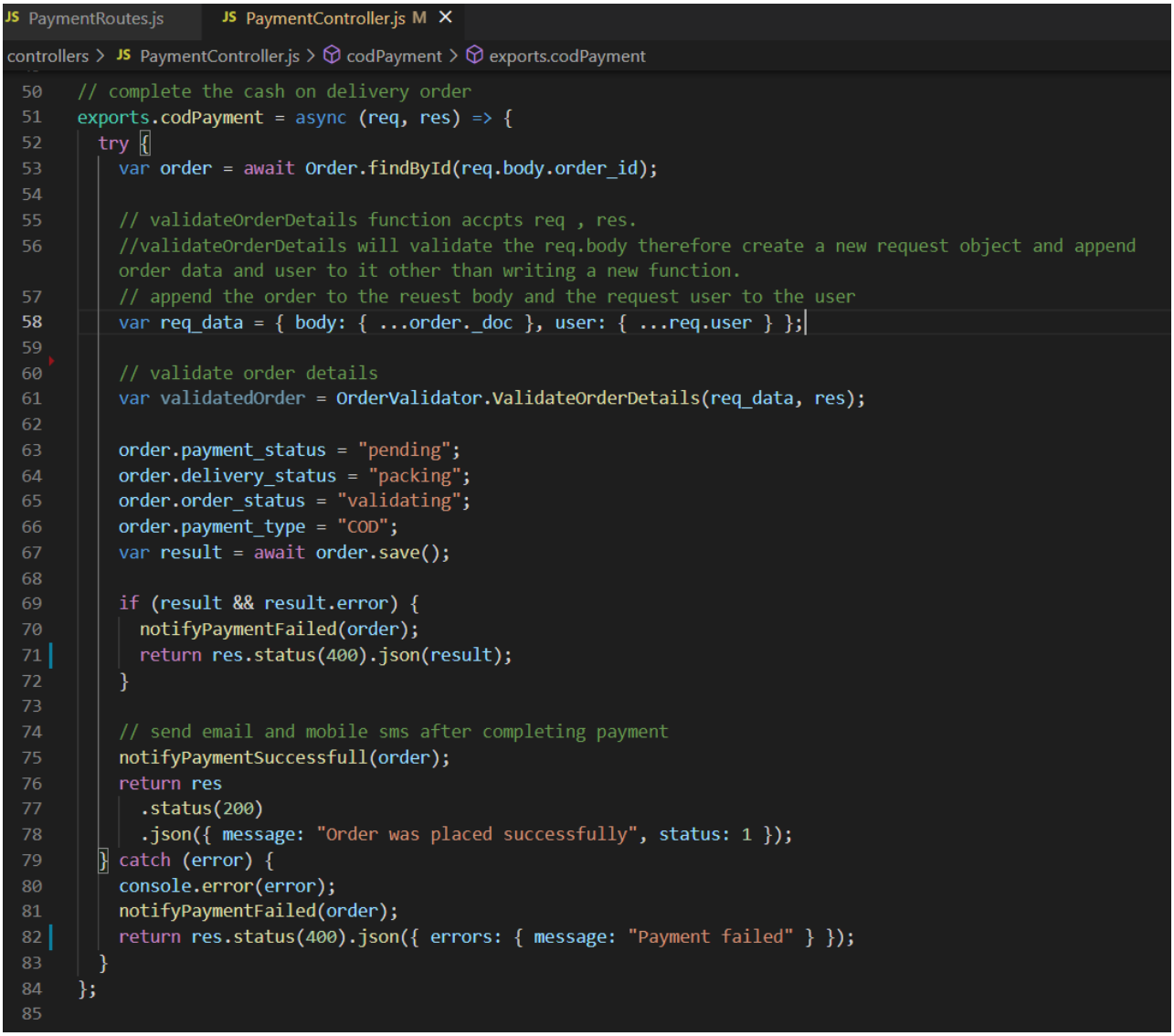
```
// send email and sms notifing payment successfully
exports.notifyPaymentFailed = async (order) => {
  var message = `Order has not been placed - ${order._id}, There were errors
  while placing your order please contact our customer service for assistance`;

  var mailOptions = {
    from: MAIL_USER,
    to: order.buyer_email,
    subject: `Order has not been placed - ${order._id}`,
    text: message,
  };
  sendMail(mailOptions);

  // send sms
  var smsOptions = {
    to: order.buyer_phone,
    body: message,
  };
  sendSms(smsOptions);
};
```

3.14.5. Process of placing cash on delivery order.

Workflow diagram



Service function



A POST request will be sent to the buyer service through ESB if the request has a parameter “payment_type” with its value “COD”. User must be authenticated to place a Cash on delivery order for this “userAuth” middleware is used to filter the request and allow only authenticated users.

The request is required to send the order ID. Then the order object will be retrieved. Order details including the buyer and delivery information must be validated before placing the order. As in above functions “ValidateOrderDetails” function is used to validate the request. If any errors are found then an error response with 422 status code will be sent.

Then update the order details setting payment status to “pending”, delivery status to “packing”, order status to “validating” and payment type to “COD”. Setting the order status to validating will avoid users from updating its details. Order details can only be updated if the order status is “pending”. This order object is saved and if it fails to save an error message is sent back to the client through ESB and error SMS and an error email will be sent to the user. Else, the user will be notified as the payment completed and a success SMS will be sent to the order’s buyer phone number and an email will be sent to the order’s buyer email.

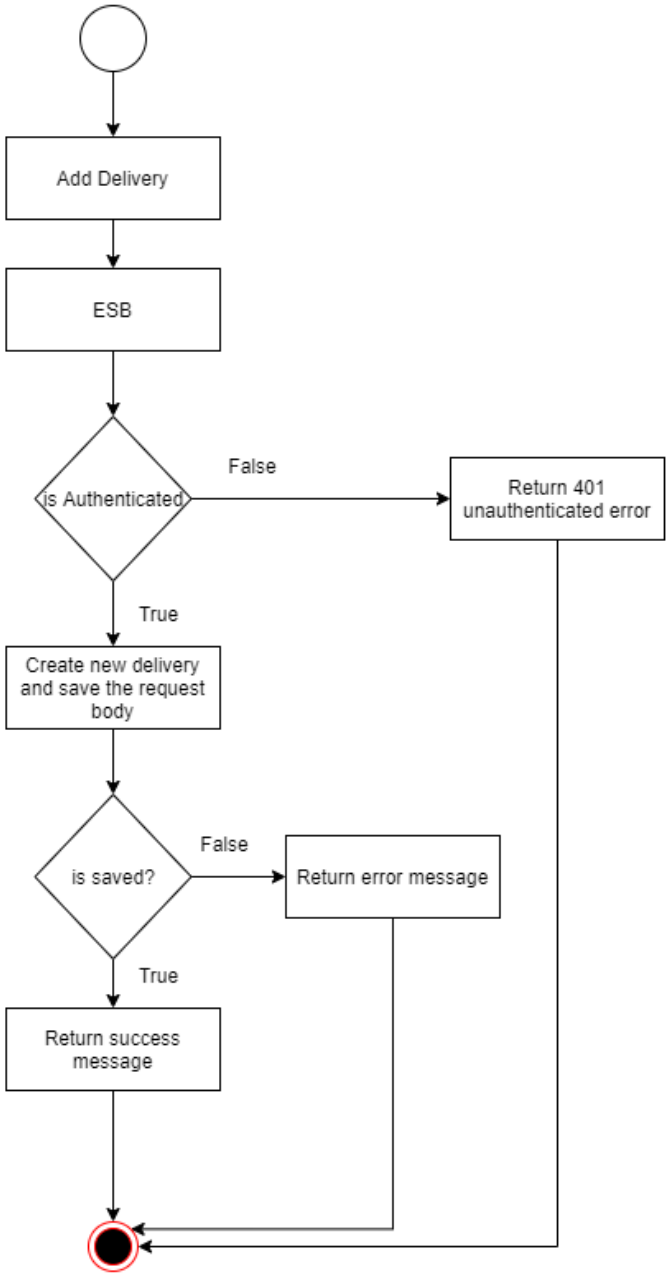
(please refer “Process of listening to payment notifications” for the screenshots of email and SMS and the code)

3.15. Delivery service

A third party rest service was developed to deliver products which were ordered through the Xmart shopping platform. After an order was placed, seller will manually validate the order details and payment details. If necessary seller will contact the user (buyer). After the order was validated, the seller will add delivery details to the Express delivery web application. Through this application, drivers will pick up the product from store and deliver to the location provided by the buyer. Sellers needs to register with this third party application and delivery is free of charge for orders placed through Xmart Shopping.

3.15.1. Process of creating a new delivery

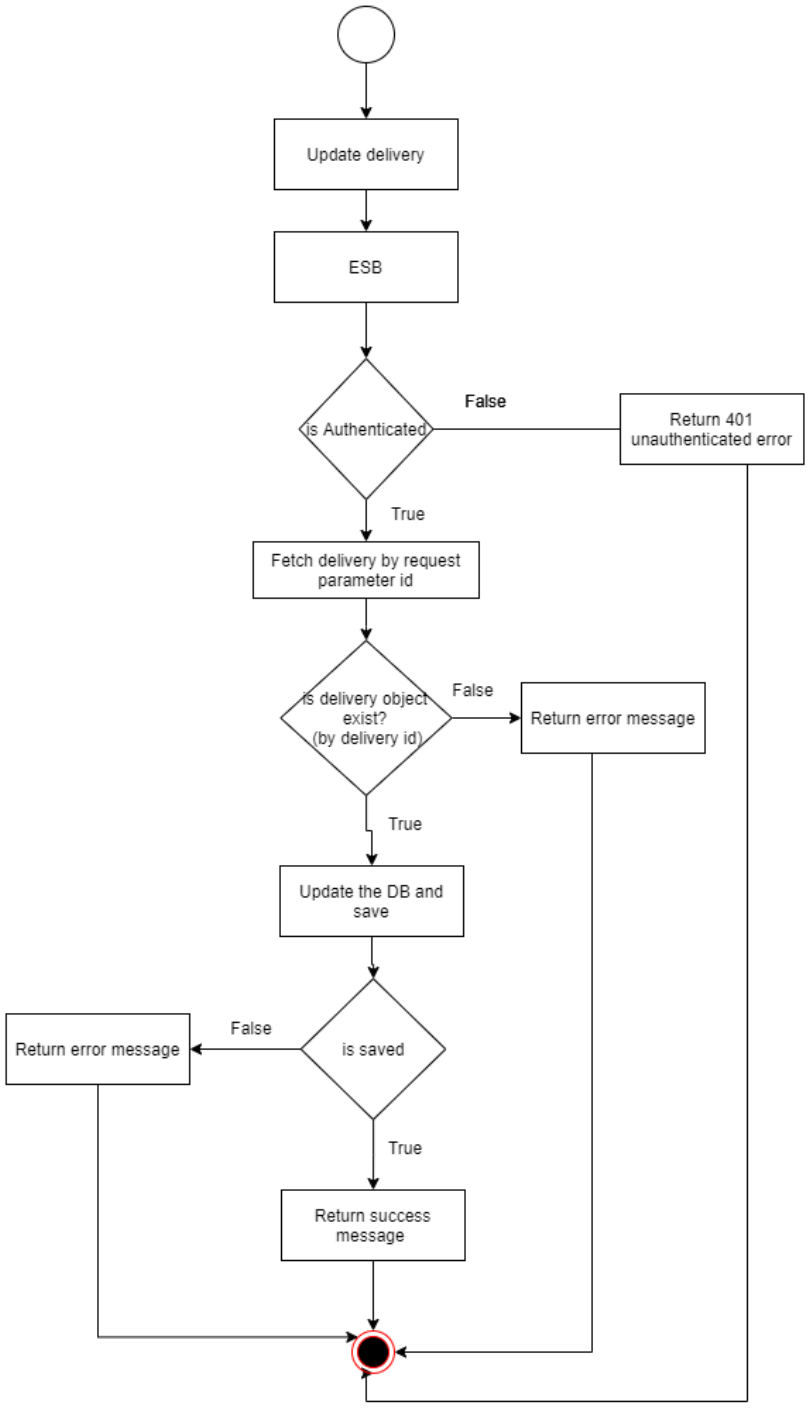
Workflow diagram



A POST request with the details (Buyer name, Buyer email, buyer telephone, address, payment type, payment value) will be sent through ESB (WSO2 E1) to the delivery service. The user should be an authenticated user, so that user id will be gone through an authentication checkup. If there is an error, an error message will be returned. If the buyer is an authenticated user, the provided data in the request body will be validated. If there is no error occurred in validation, a new delivery will be created in and saved in the DB. If an error occurred, the user will be informed by an error message. After the new delivery saved in the DB successfully, a success message will be returned. Otherwise, an error message will be returned.

3.15.2. Process of updating a delivery

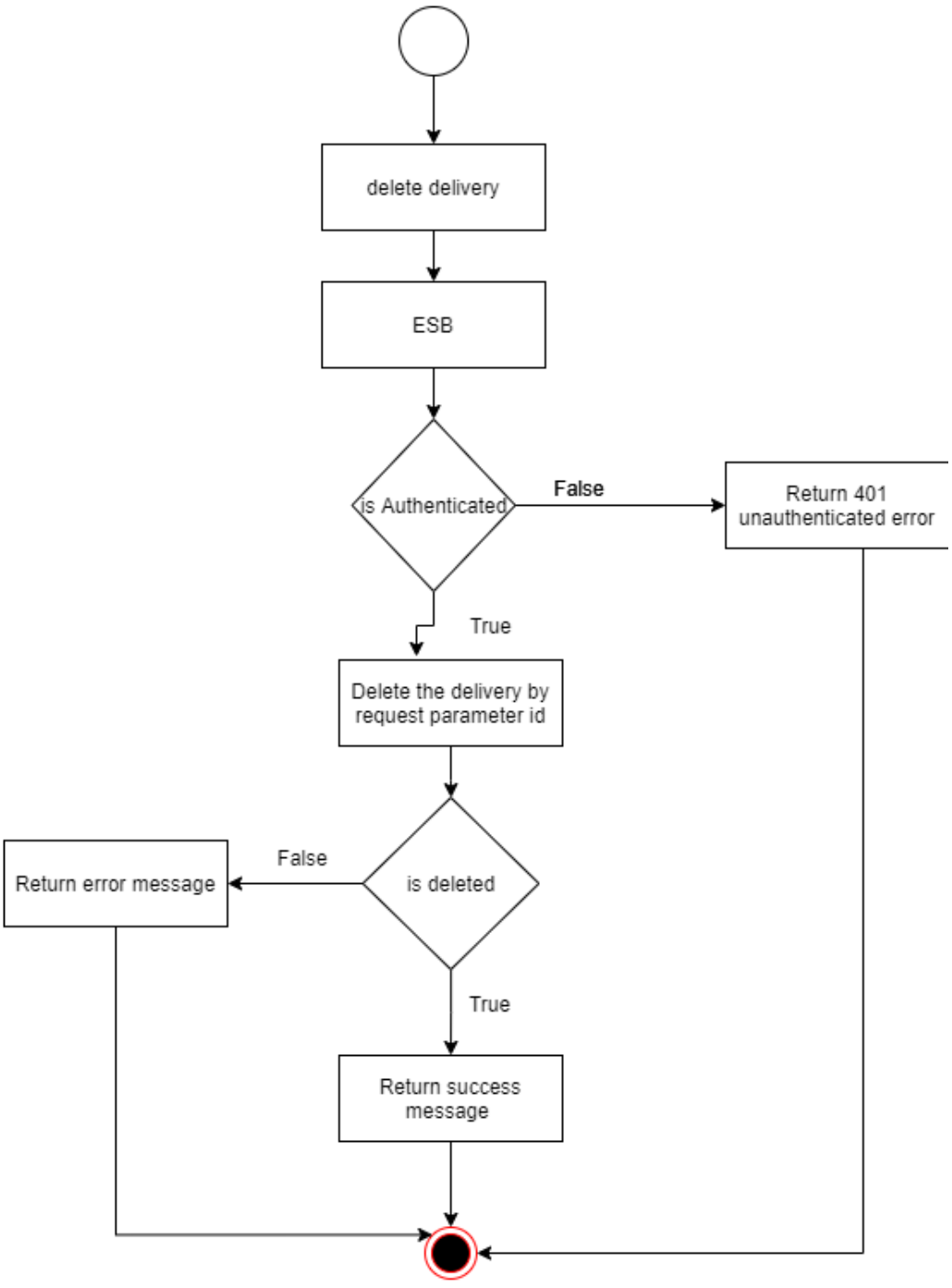
Workflow diagram



A PATCH request with the delivery id and updated details will be sent through ESB (WSO2 E1) to the delivery service. After the authentication, the request will be validated. Then the availability of the delivery object is checked and if available delivery will be updated and saved in the DB. If there is error in the process error message will be returned.

3.15.3. Process of deleting a delivery

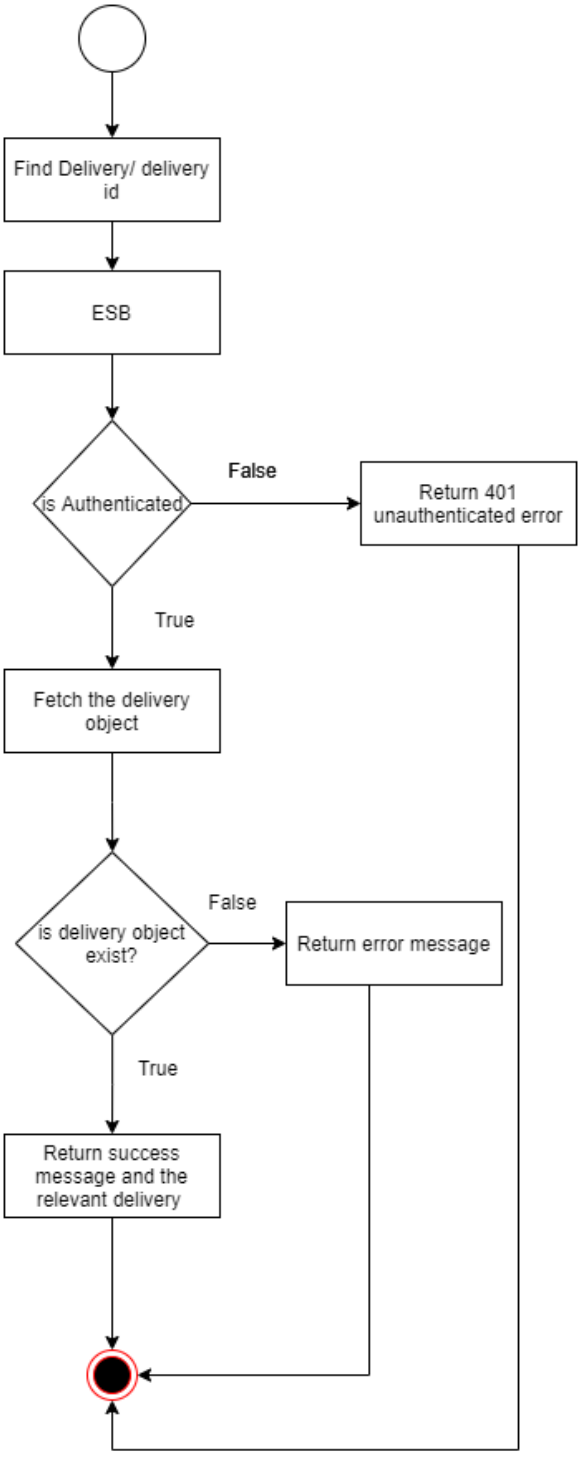
Workflow diagram



A DELETE request with the delivery id will be sent through ESB (WSO2 E1) to the delivery service. After the authentication, the request will be validated. Then the availability of the delivery object is checked and if available the delivery will be deleted. If there is error in the process error message will be returned.

3.15.4. Process of finding a delivery

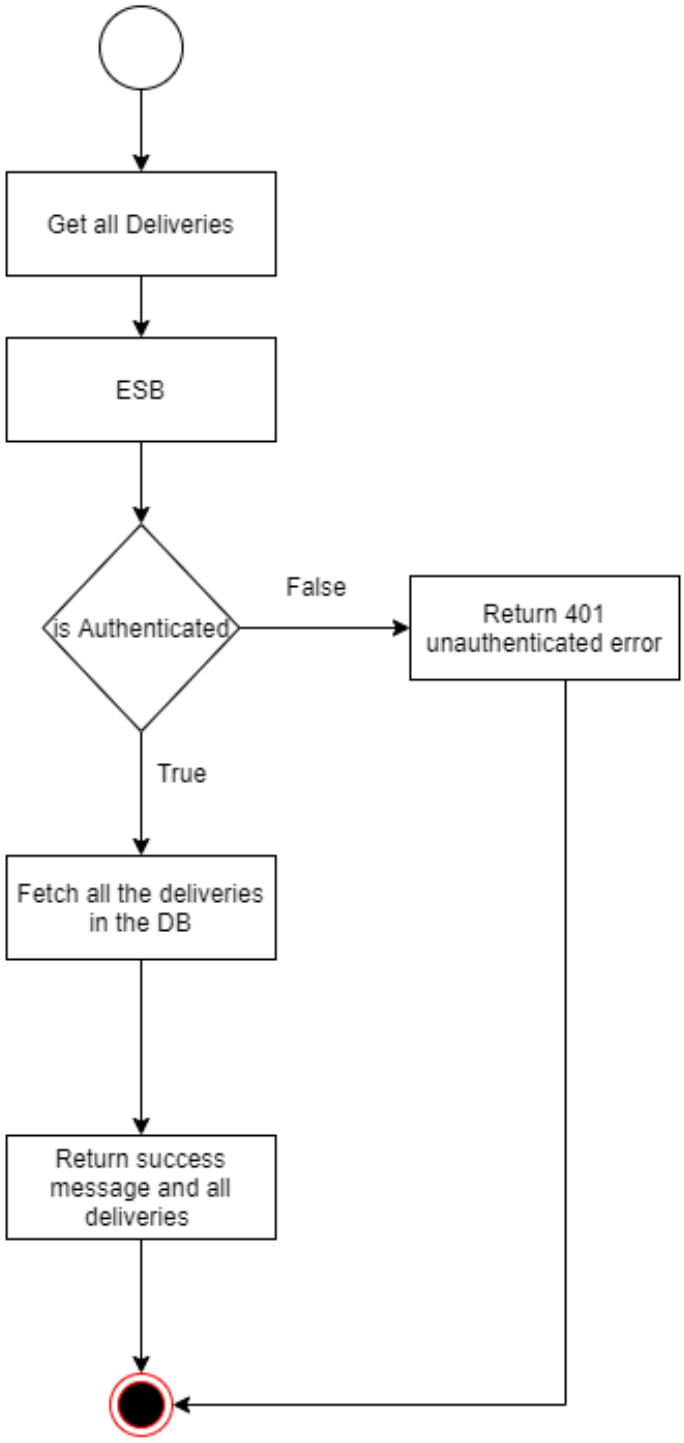
Workflow diagram



A GET request with the specified id will be sent through ESB (WSO2 E1) to the delivery service. After authentication, the delivery id will be validated and if there is no error, the delivery corresponding to the id will be fetched. In case if an error occurred, error message will be returned.

3.15.5. Process of listing all deliveries

Workflow diagram



A GET request will be sent through ESB (WSO2 E1) to the delivery service. After authentication, the validation check will be taking placed and if it is successfully validated, all the deliveries in the DB will fetched. In case if an error occurred, error message will be returned.

4. Authentication and Security Mechanism adopted with system

4.1. User Authentication

Json web token based authentication mechanism is used to verify user identity by returning a unique token. Guest have to verify the credential once and in return they will get a unique token which is allowed to access for 10 minutes. The token will have user's basic details and role details.

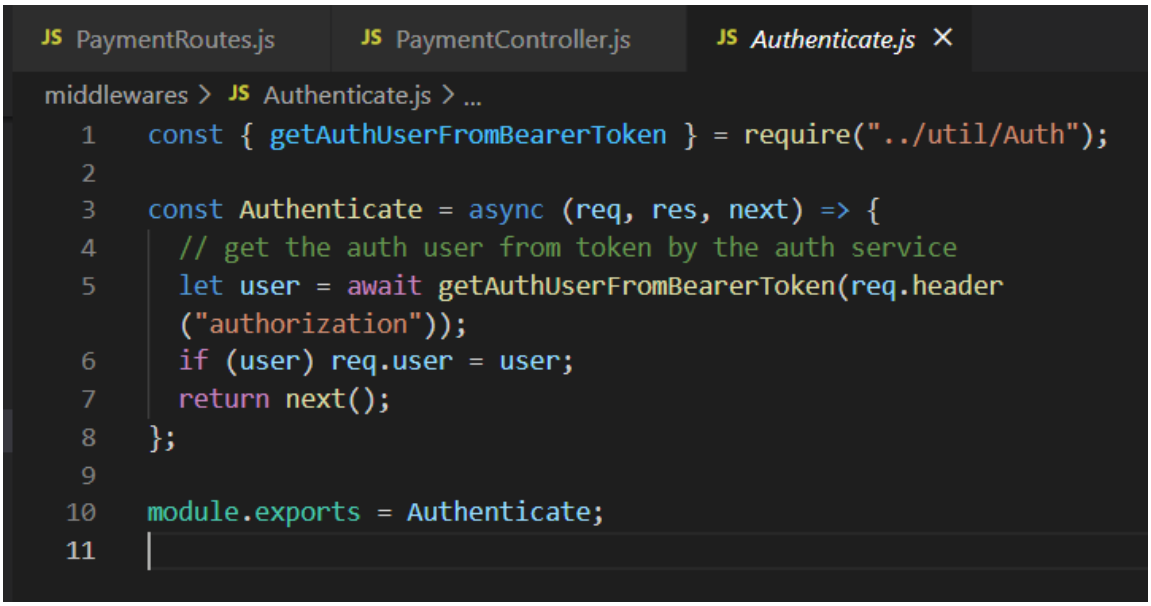
When registering a user and resetting a password the password entered is encrypted (using `bcrypt`) and stored in the database. Even when querying we can get only the highest version of the password unless specially quarried for the real password.

User can request to reset the password then a hashed reset token will be generated and an email will be sent to the user's email address along with the reset URL. Once user visit the reset URL, server checks the validity of the reset token and allows to change the password.

4.2. Service Authentication

Since the authentication is developed as a service, the buyer service and the seller service requires to be authenticated externally. For this the buyer service and the seller service will be using a middleware to send a request to the authentication service to validate the bearer token sent in request headers through the WSO2 EI (ESB).

"Authenticate" middleware in the buyer/seller service is used to validate the response that is returned after requesting the authentication service with authorization token in request headers. If the response sent from the authentication service has a user, it will be added to the request object and the next function will be invoke.



Function “getAuthUserFromBearerToken” is used in the buyer/seller service to request the authentication service with the authorization token(JWT) and return back the response. The request will be sent through the ESB.

PaymentRoutes.js | PaymentController.js | Authenticate.js | *Auth.js*

util > Auth.js > getAuthUserFromBearerToken > options

```
1   const util = require("util");
2   var request = require("request");
3   const { AUTH_GATEWAY } = require("../config");
4
5   // contact auth service through esb to validate the jwt token and get the user
6   const getAuthUserFromBearerToken = async (bearer_token) => {
7     var token = String(bearer_token).slice(7);
8     const options = {
9       url: AUTH_GATEWAY,
10      method: "POST",
11      headers: {
12        "Content-Type": "application/json",
13      },
14      maxRedirects: 20,
15      body: JSON.stringify({
16        token: `${token}`,
17      }),
18    };
19
20    const requestPromise = util.promisify(request);
21    try {
22      const response = await requestPromise(options);
23      // console.log(response.body);
24      let jsonBody = response.body ? JSON.parse(response.body) : {};
25      if (response.statusCode == 200 && jsonBody._id) return jsonBody;
26      else return null;
27    } catch (error) {
28      console.error(error);
29    }
30  };
31
32  module.exports = { getAuthUserFromBearerToken };
33
```

Function “validate token” is used in the Authentication service to validate the token sent in the request body and if the token is verified, user will be retrieved and returned.

```
109
110  exports.ValidateToken = async (req, res) => {
111    try {
112      // validate token in request
113      if (!req.body.token)
114        return res.status(422).json({ message: "JWT token is required" });
115      // validate JWT token
116      var decodedToken = jwt.verify(req.body.token, KEY);
117      // fetch user token token user id
118      var user = await User.findById(decodedToken.user_id);
119      return res.status(200).json(user);
120    } catch (error) {
121      return res.status(400).json({ message: "JWT token is not valid" });
122    }
123  };
124
```

If the token is valid then the user will be added to the request object in the Buyer and Seller service.

Since the role “seller” must be validated in the seller service another middleware is used to validate the role in the request user.

JS index.js | JS SellerAuth.js | JS Authenticate.js | JS ProductRoutes.js

middlewares > JS SellerAuth.js > ...

```
1  const SellerAuth = async (req, res, next) => {
2    if (req.user && req.user.role == "seller") return next();
3    else return res.send(401, "Unauthorized");
4  };
5
6  module.exports = SellerAuth;
7
```

These middlewares will be used as bellow.

```
8
9   const app = express();
10
11  app.use(bodyParser.json());
12  app.use(cors());
13
14  app.all("*", Authenticate);
15  app.use("/api/admin/product", SellerAuth, require("./routes/ProductRoutes"));
16
```

4.3.Payment Security

In order to provide extra security when making a payment through credit card gateway or mobile payment gateway, order details along with the transfer amount will be hashed.

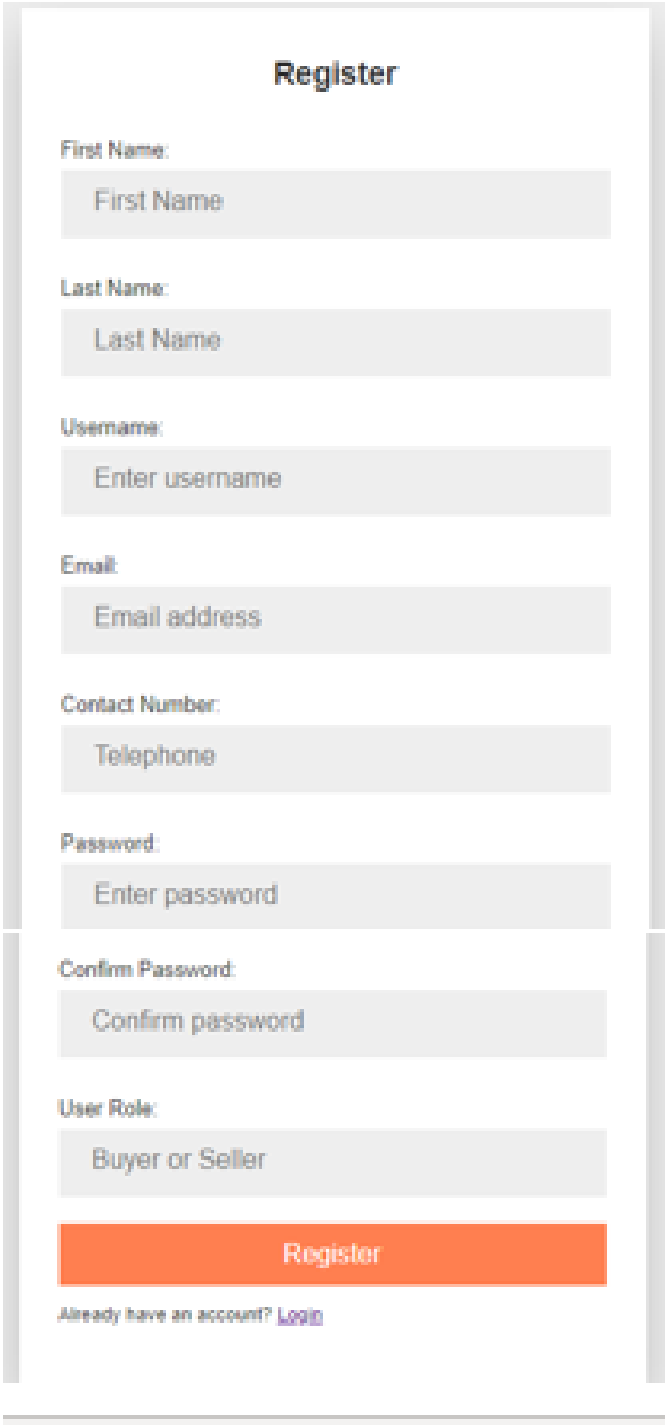
When the user is paying through credit card / mobile then a request will be sent to the gateway with transfer amount and order id. Since these data could be changed, another parameter is sent which is a hashed value generated using the order id, transfer amount and order secret key. Order secret key will be saved in environment configurations in both the services. The payment gateway will hash the data received (transfer amount and order id) with the order secret and generate a hashed value. If this value and the hashed parameter sent along with the data matches then the payment is secure and can be made.

(In real scenario payment gateways will use the Client User ID and Secret received when registering to the gateway)

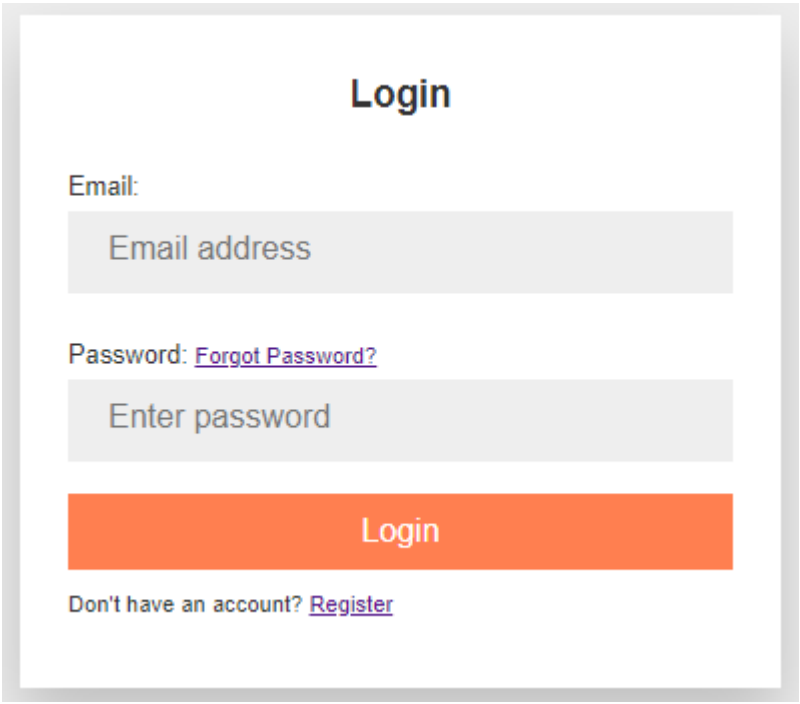
Also after completing the payment, gateway will send a request to the buyer service with the data such as order id, transfer amount and payment hash code. Payment hash code is generated using the order id, transfer amount and payment secret key. Payment secret key will be saved in environment configurations in both the services. Then the buyer service will validate and compare a payment hash codes and if valid the order payment will be updated as paid and notifications will be sent through mail and SMS.

5. Screenshots of the User Interfaces

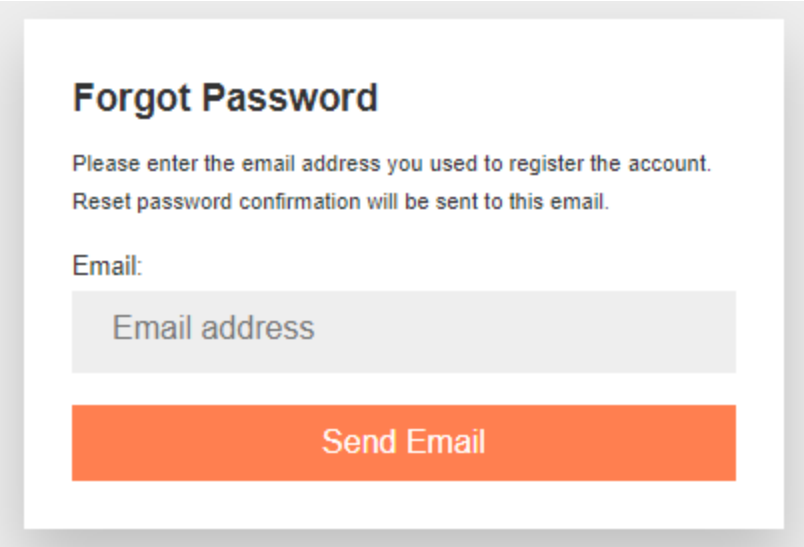
Register

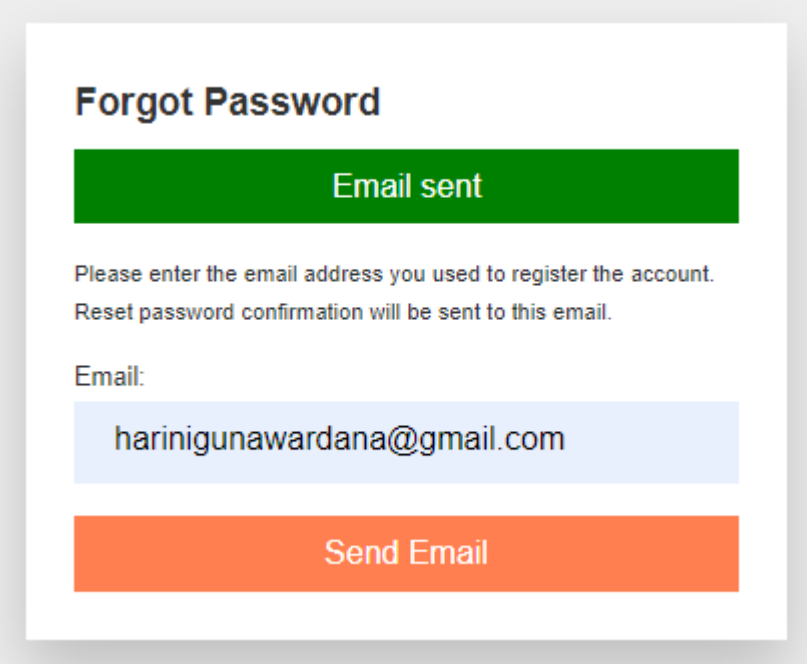


Login

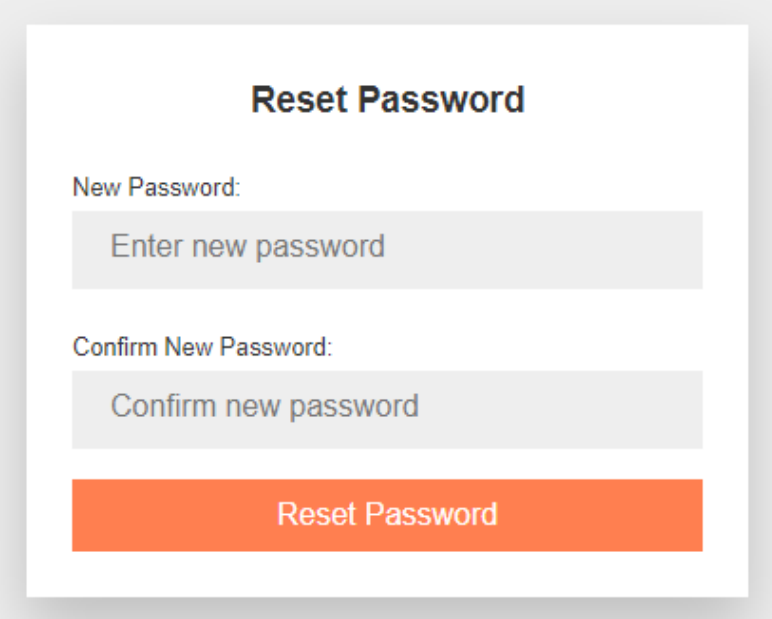

Login
Email:
Email address
Password: Forgot Password?
Enter password
Login
Don't have an account? Register

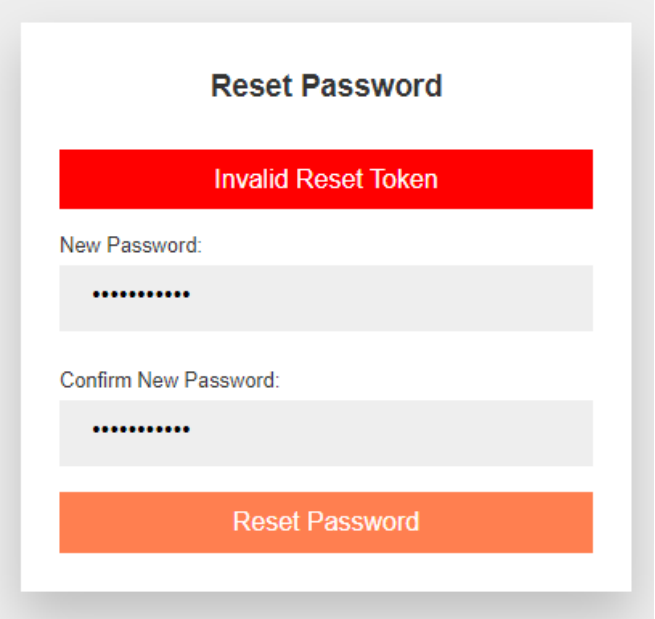
Forgot Password

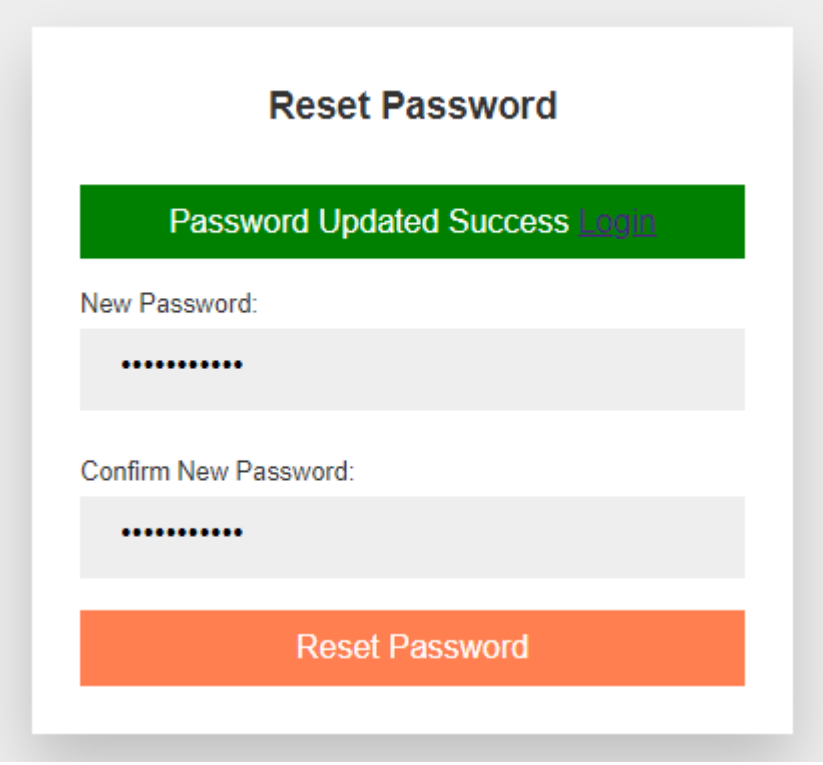


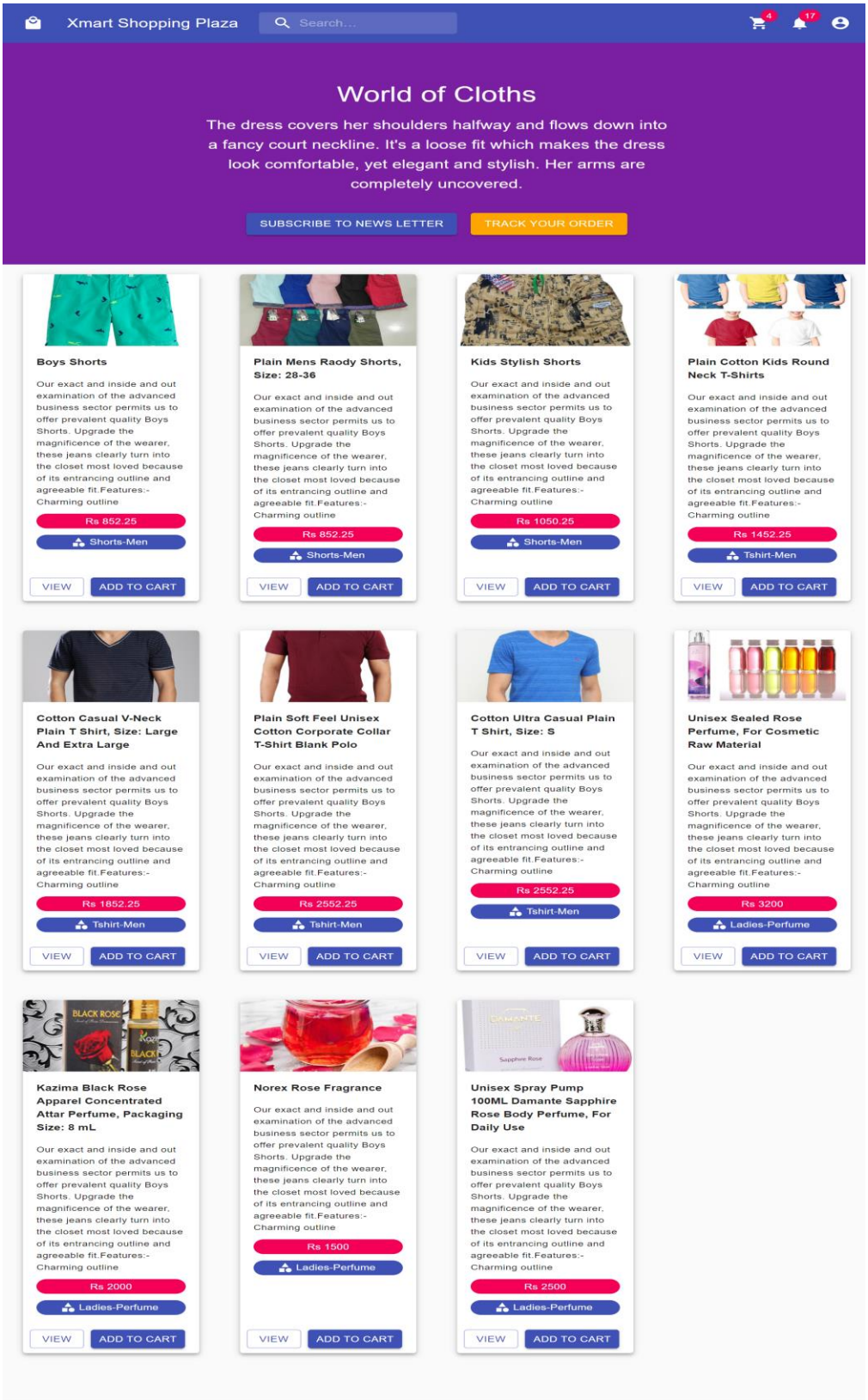


Reset Password

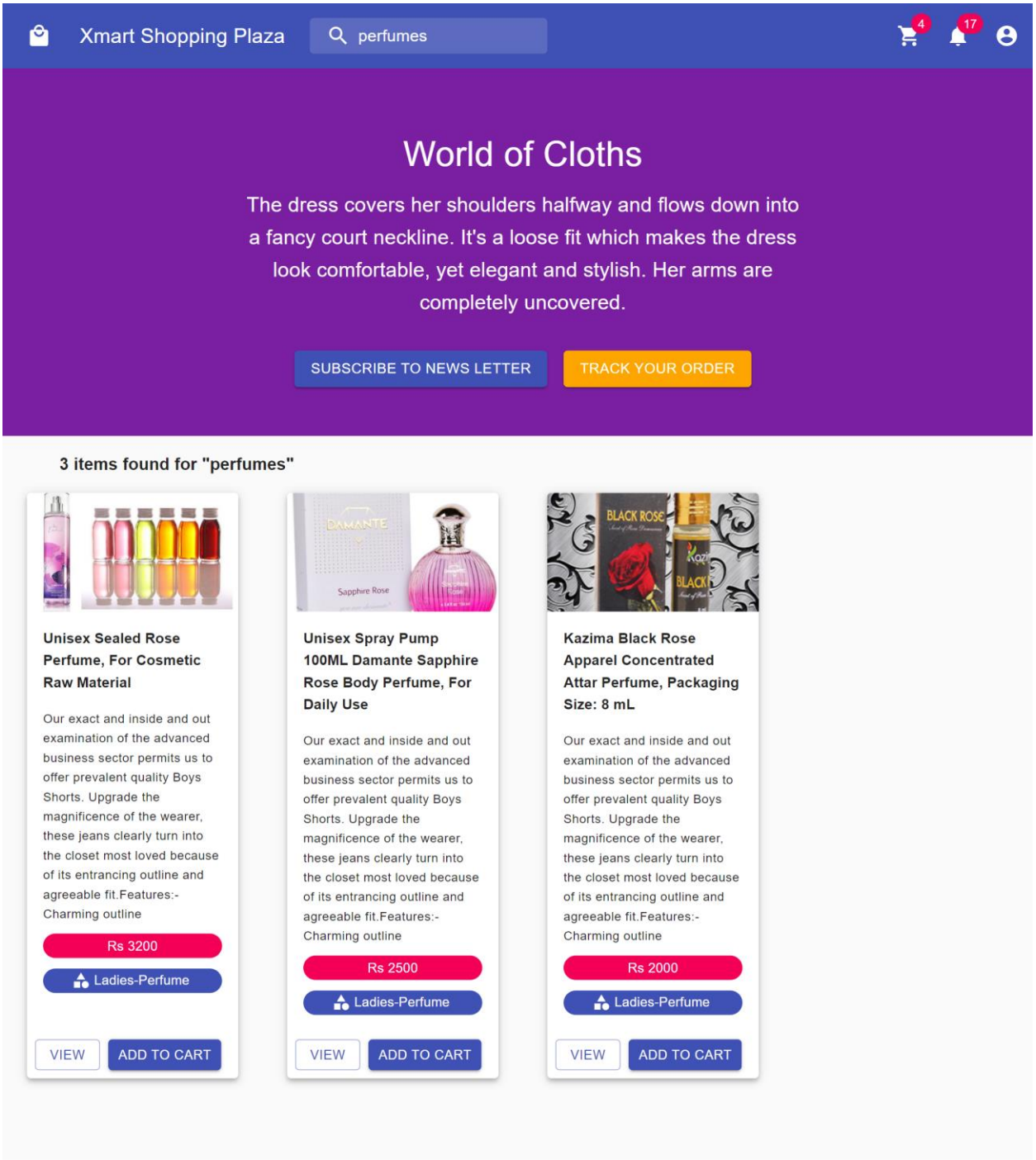




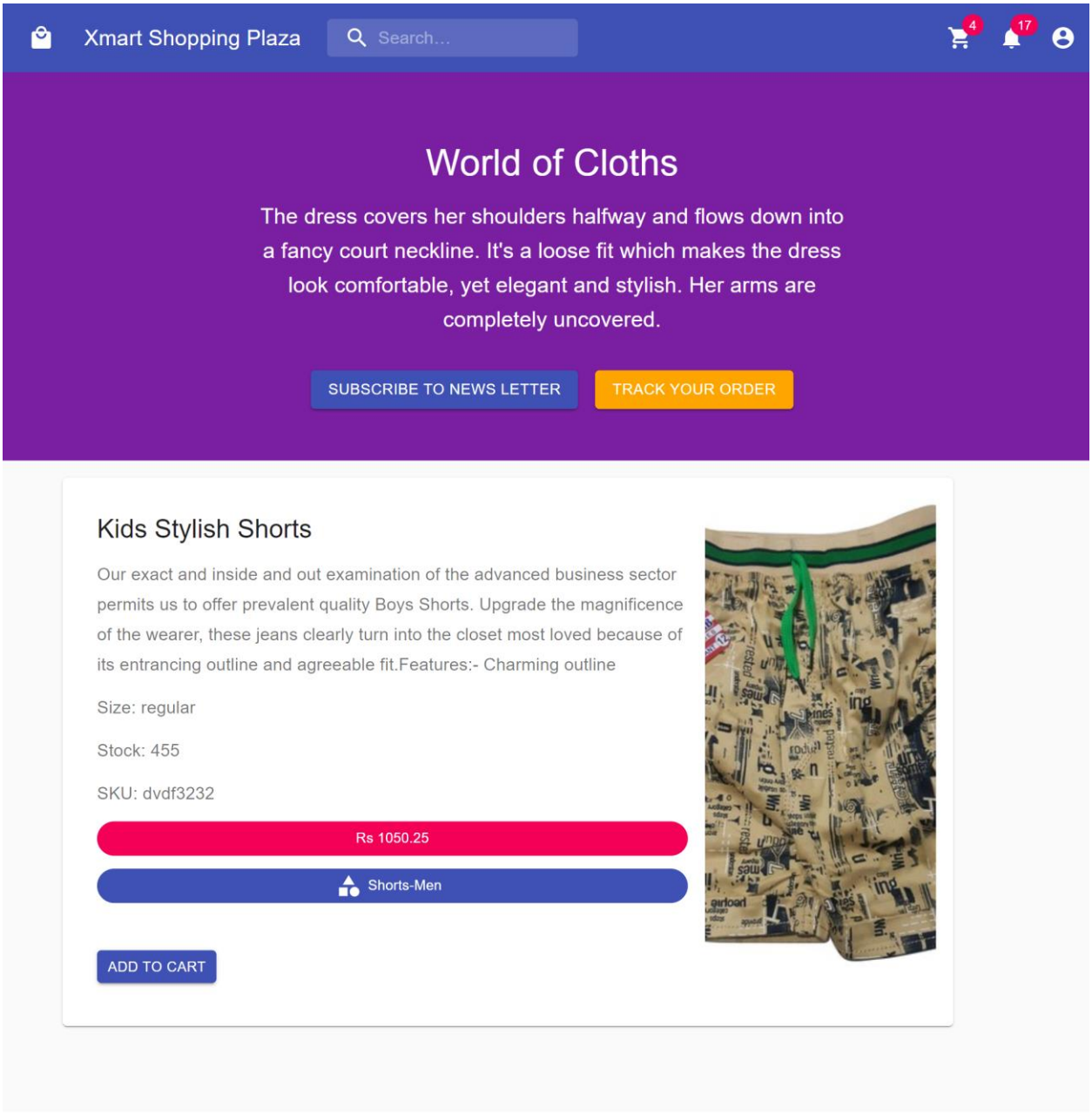



Xmart Shopping Plaza
Search...
World of Cloths
The dress covers her shoulders halfway and flows down into a fancy court neckline. It's a loose fit which makes the dress look comfortable, yet elegant and stylish. Her arms are completely uncovered.
SUBSCRIBE TO NEWS LETTER
TRACK YOUR ORDER
Boys Shorts
Our exact and inside and out examination of the advanced business sector permits us to offer prevalent quality Boys Shorts. Upgrade the magnificence of the wearer, these jeans clearly turn into the closet most loved because of its entrancing outline and agreeable fit.Features:- Charming outline
Rs 852.25
Shorts-Men
VIEW
ADD TO CART
Plain Mens Raody Shorts, Size: 28-36
Our exact and inside and out examination of the advanced business sector permits us to offer prevalent quality Boys Shorts. Upgrade the magnificence of the wearer, these jeans clearly turn into the closet most loved because of its entrancing outline and agreeable fit.Features:- Charming outline
Rs 852.25
Shorts-Men
VIEW
ADD TO CART
Kids Stylish Shorts
Our exact and inside and out examination of the advanced business sector permits us to offer prevalent quality Boys Shorts. Upgrade the magnificence of the wearer, these jeans clearly turn into the closet most loved because of its entrancing outline and agreeable fit.Features:- Charming outline
Rs 1050.25
Shorts-Men
VIEW
ADD TO CART
Plain Cotton Kids Round Neck T-Shirts
Our exact and inside and out examination of the advanced business sector permits us to offer prevalent quality Boys Shorts. Upgrade the magnificence of the wearer, these jeans clearly turn into the closet most loved because of its entrancing outline and agreeable fit.Features:- Charming outline
Rs 1452.25
Tshirt-Men
VIEW
ADD TO CART
Cotton Casual V-Neck Plain T Shirt, Size: Large And Extra Large
Our exact and inside and out examination of the advanced business sector permits us to offer prevalent quality Boys Shorts. Upgrade the magnificence of the wearer, these jeans clearly turn into the closet most loved because of its entrancing outline and agreeable fit.Features:- Charming outline
Rs 1852.25
Tshirt-Men
VIEW
ADD TO CART
Plain Soft Feel Unisex Cotton Corporate Collar T-Shirt Blank Polo
Our exact and inside and out examination of the advanced business sector permits us to offer prevalent quality Boys Shorts. Upgrade the magnificence of the wearer, these jeans clearly turn into the closet most loved because of its entrancing outline and agreeable fit.Features:- Charming outline
Rs 2552.25
Tshirt-Men
VIEW
ADD TO CART
Cotton Ultra Casual Plain T Shirt, Size: S
Our exact and inside and out examination of the advanced business sector permits us to offer prevalent quality Boys Shorts. Upgrade the magnificence of the wearer, these jeans clearly turn into the closet most loved because of its entrancing outline and agreeable fit.Features:- Charming outline
Rs 2552.25
Tshirt-Men
VIEW
ADD TO CART
Unisex Sealed Rose Perfume, For Cosmetic Raw Material
Our exact and inside and out examination of the advanced business sector permits us to offer prevalent quality Boys Shorts. Upgrade the magnificence of the wearer, these jeans clearly turn into the closet most loved because of its entrancing outline and agreeable fit.Features:- Charming outline
Rs 3200
Ladies-Perfume
VIEW
ADD TO CART
Kazima Black Rose Apparel Concentrated Attar Perfume, Packaging Size: 8 mL
Our exact and inside and out examination of the advanced business sector permits us to offer prevalent quality Boys Shorts. Upgrade the magnificence of the wearer, these jeans clearly turn into the closet most loved because of its entrancing outline and agreeable fit.Features:- Charming outline
Rs 2000
Ladies-Perfume
VIEW
ADD TO CART
Norex Rose Fragrance
Our exact and inside and out examination of the advanced business sector permits us to offer prevalent quality Boys Shorts. Upgrade the magnificence of the wearer, these jeans clearly turn into the closet most loved because of its entrancing outline and agreeable fit.Features:- Charming outline
Rs 1500
Ladies-Perfume
VIEW
ADD TO CART
Unisex Spray Pump 100ML Damante Sapphire Rose Body Perfume, For Daily Use
Our exact and inside and out examination of the advanced business sector permits us to offer prevalent quality Boys Shorts. Upgrade the magnificence of the wearer, these jeans clearly turn into the closet most loved because of its entrancing outline and agreeable fit.Features:- Charming outline
Rs 2500
Ladies-Perfume
VIEW
ADD TO CART
Xmart Shopping Plaza
Sri Lanka's largest shopping platform
Copyright © Xmart.com 2021.

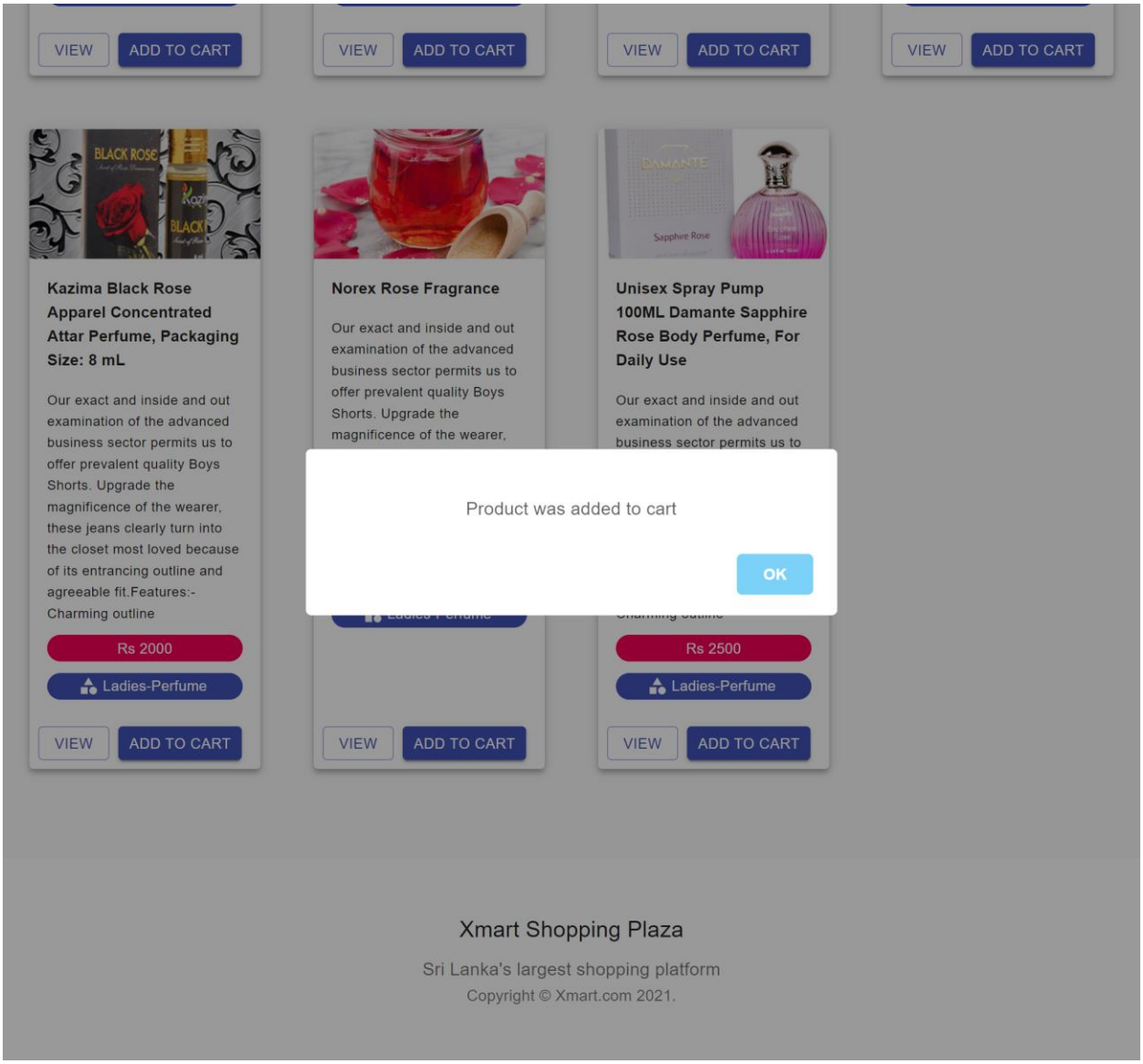
Search products



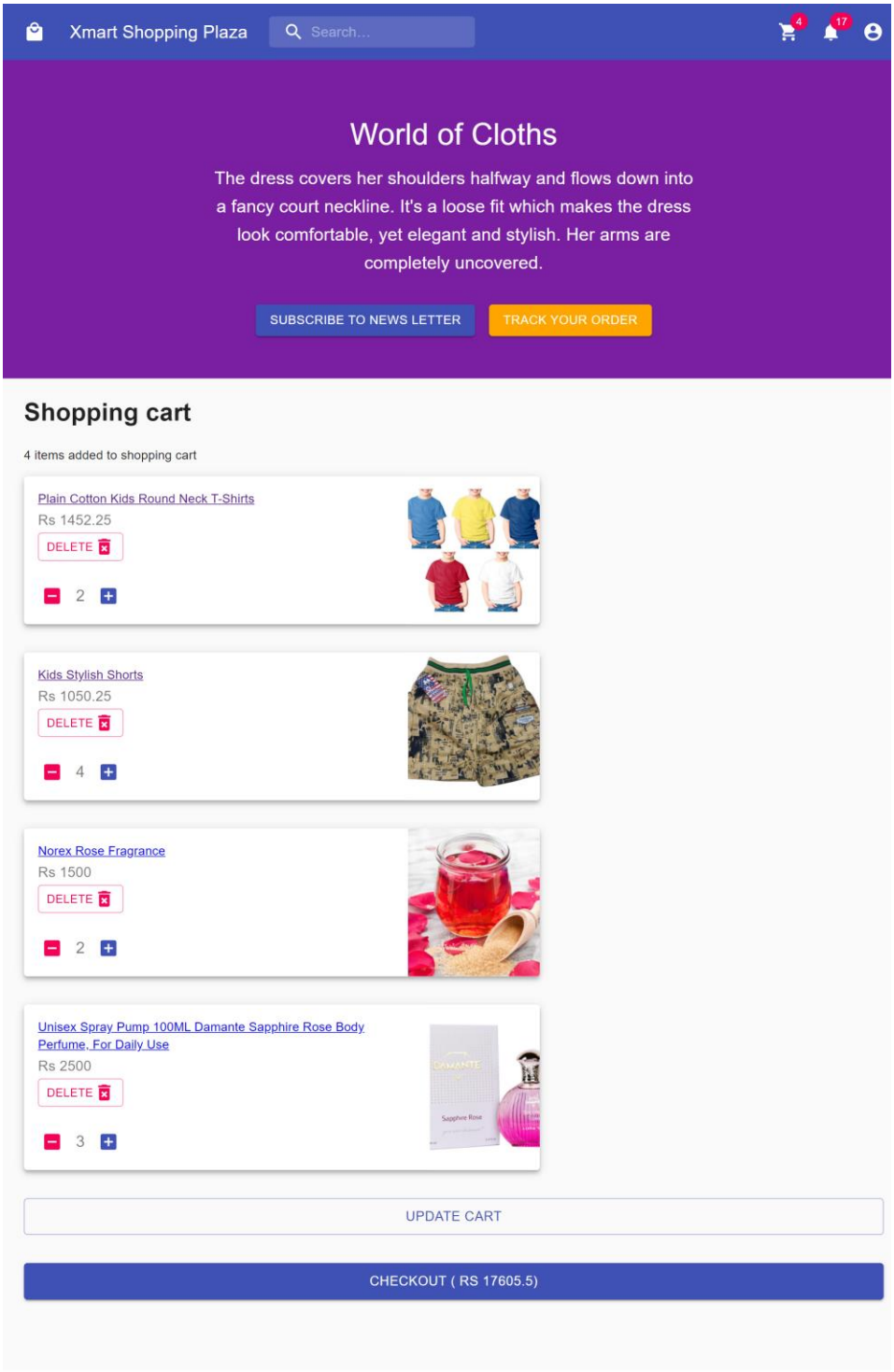
View product



Add to cart


VIEW
ADD TO CART
Kazima Black Rose Apparel Concentrated Attar Perfume, Packaging Size: 8 mL
Our exact and inside and out examination of the advanced business sector permits us to offer prevalent quality Boys Shorts. Upgrade the magnificence of the wearer, these jeans clearly turn into the closet most loved because of its entrancing outline and agreeable fit.Features:- Charming outline
Rs 2000
Ladies-Perfume
Norex Rose Fragrance
Our exact and inside and out examination of the advanced business sector permits us to offer prevalent quality Boys Shorts. Upgrade the magnificence of the wearer,
Unisex Spray Pump 100ML Damante Sapphire Rose Body Perfume, For Daily Use
Our exact and inside and out examination of the advanced business sector permits us to
Rs 2500
Product was added to cart
OK
Xmart Shopping Plaza
Sri Lanka's largest shopping platform
Copyright © Xmart.com 2021.

Shopping cart

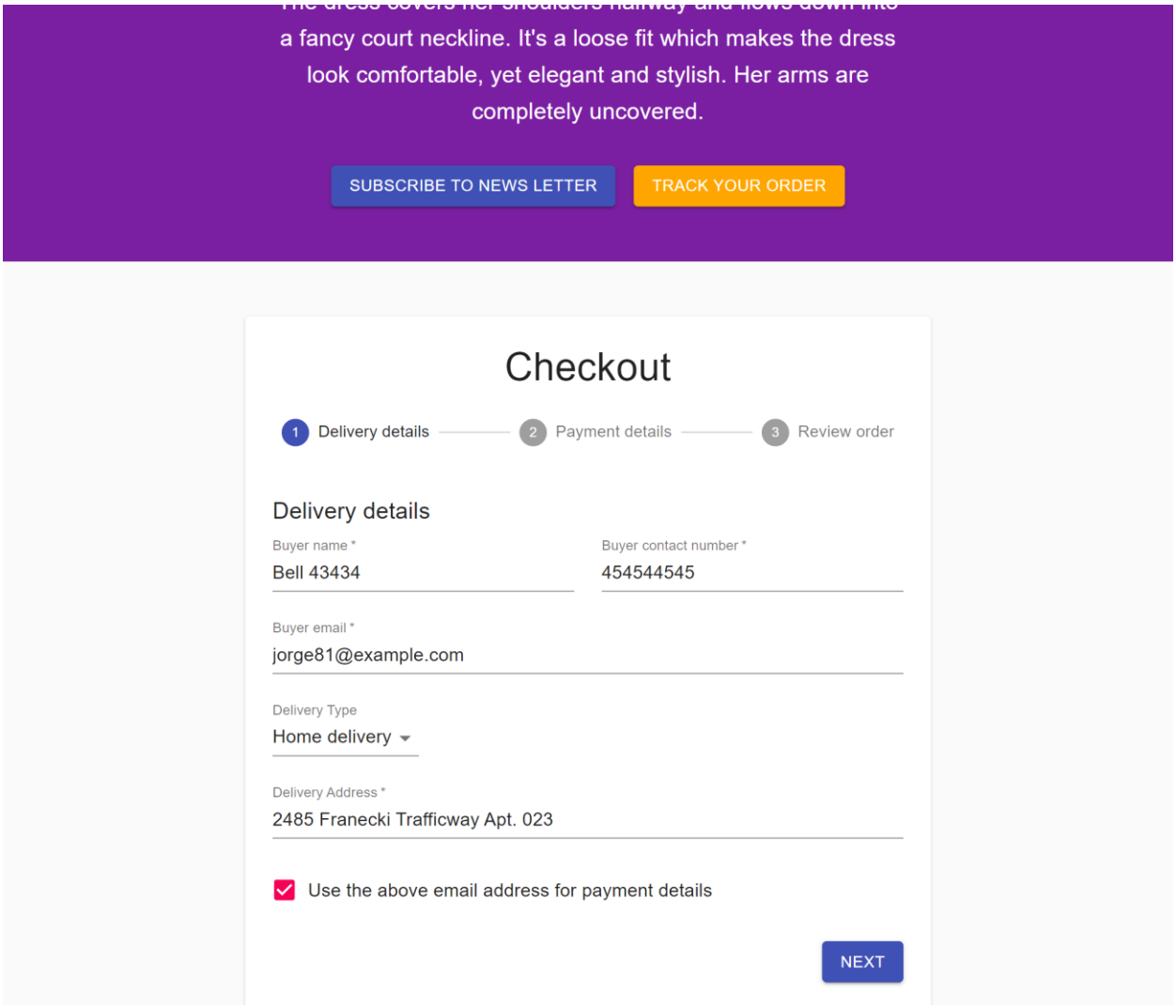


Xmart Shopping Plaza

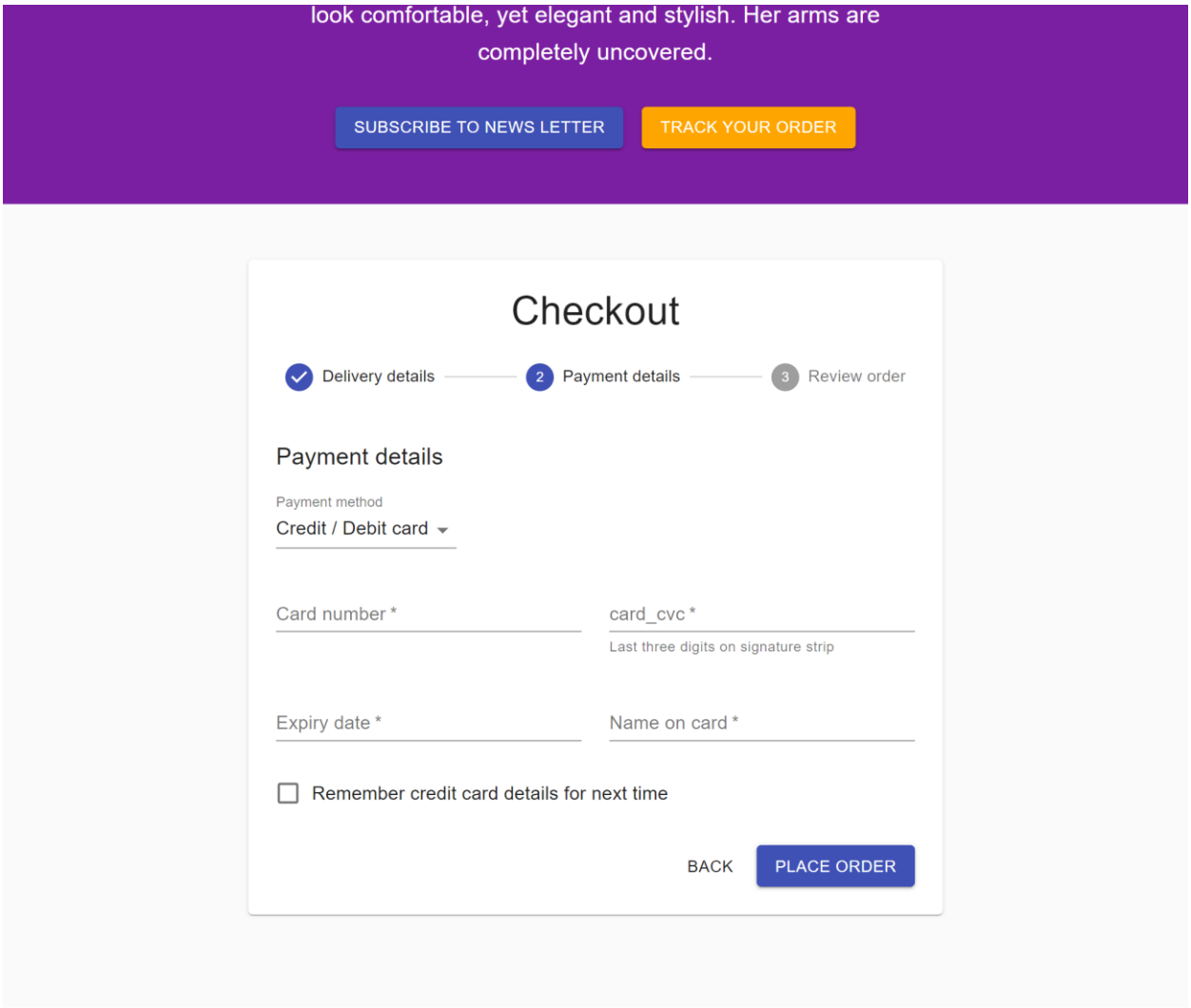
Sri Lanka's largest shopping platform

Copyright © Xmart.com 2021.

Order details

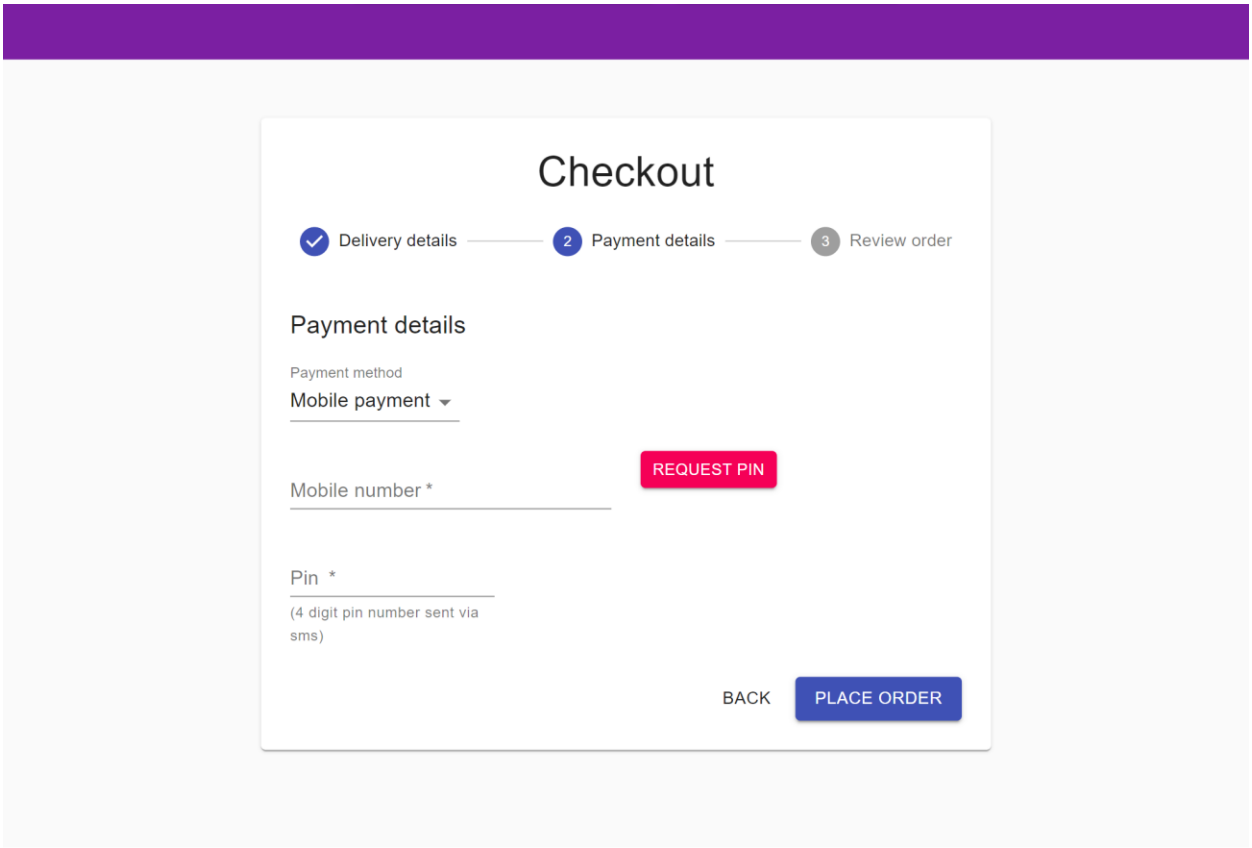


Payment credit card



Xmart Shopping Plaza

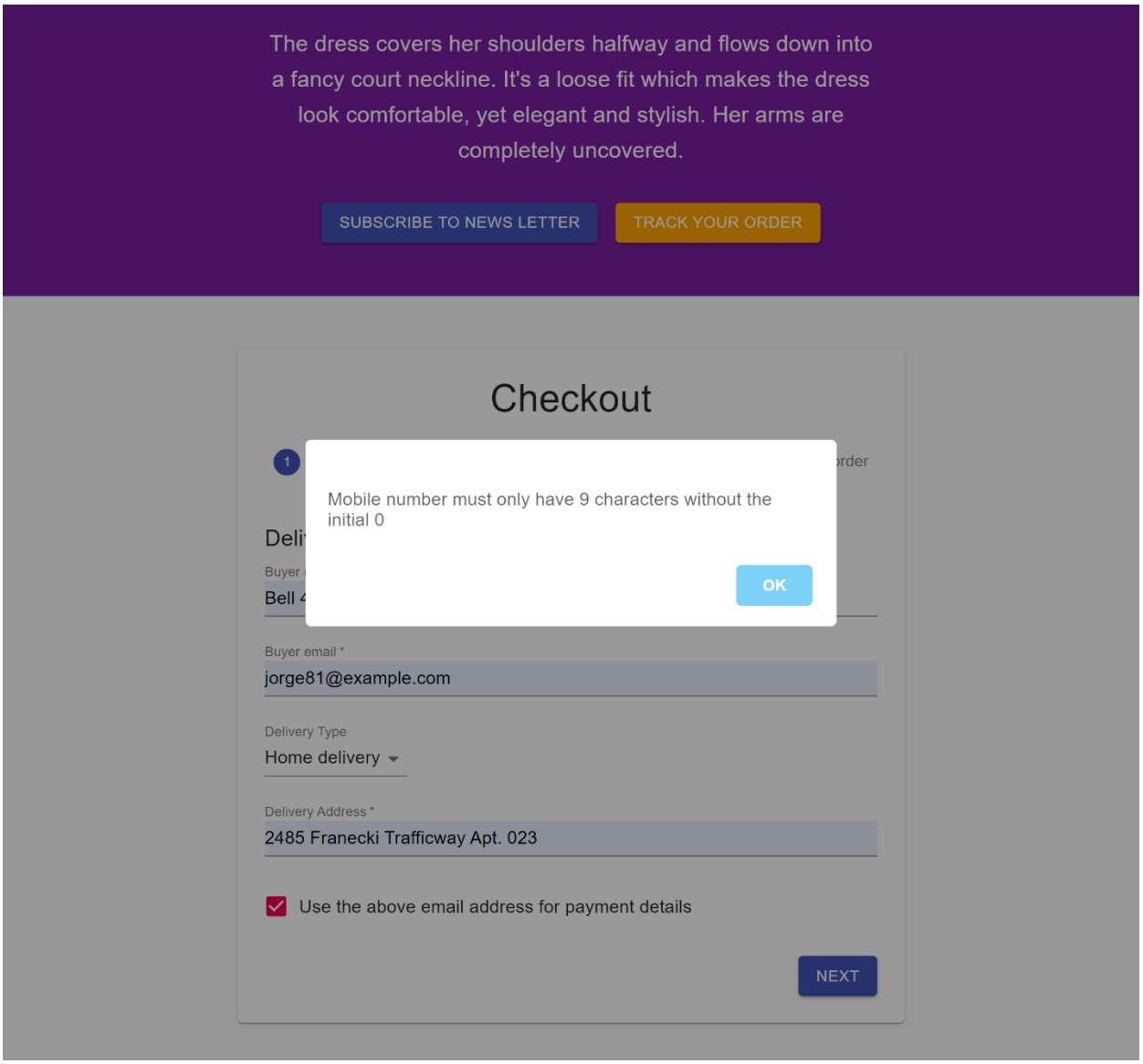
Payment mobile



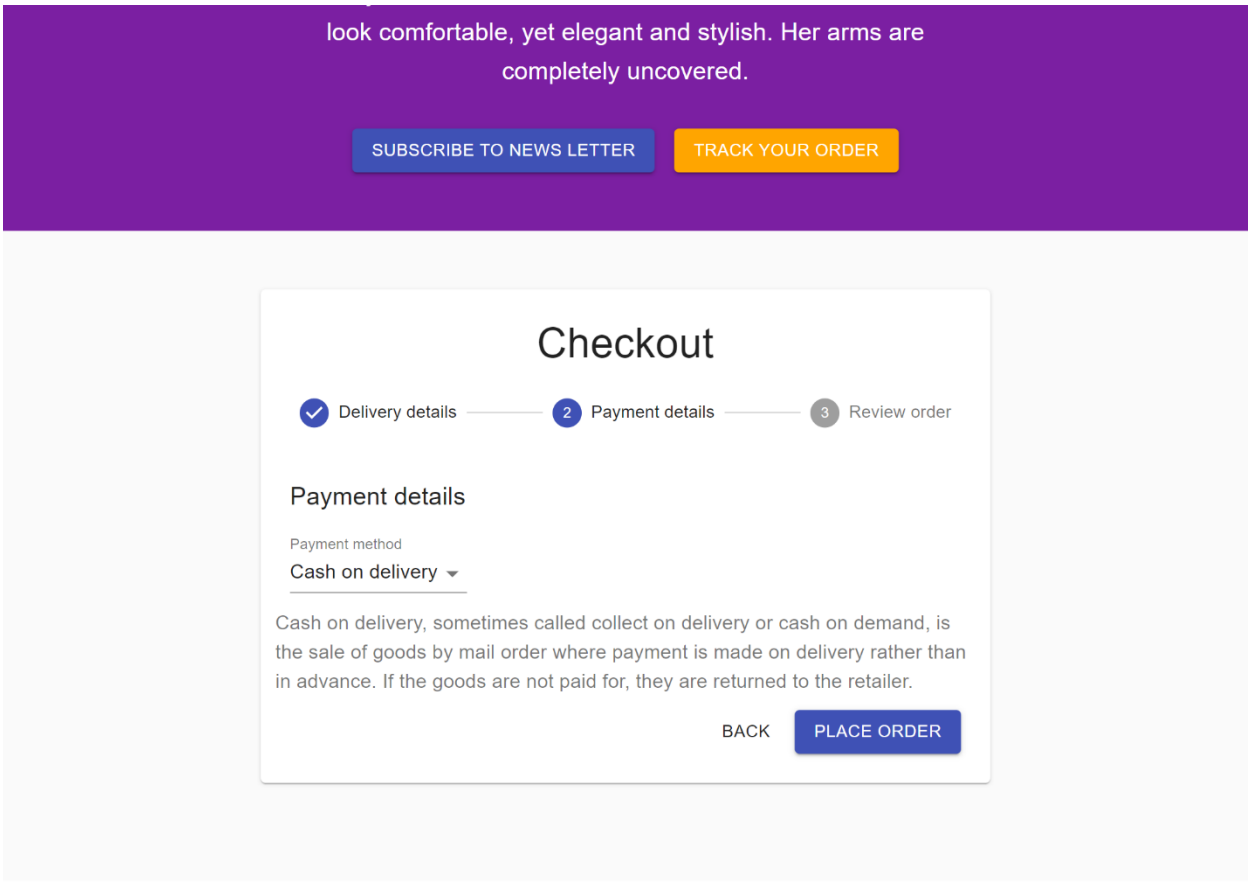
Xmart Shopping Plaza

Sri Lanka's largest shopping platform

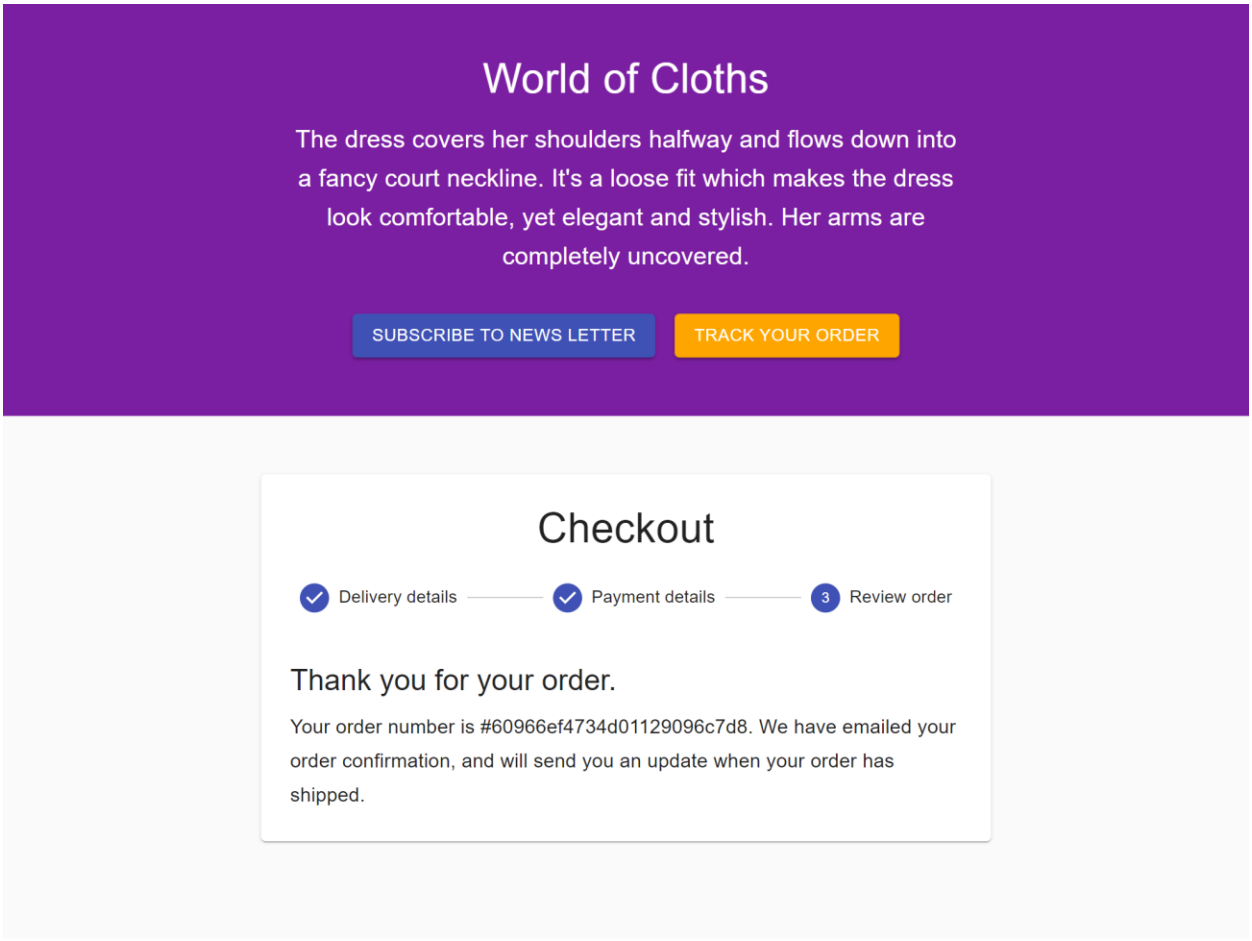
Validators


The dress covers her shoulders halfway and flows down into a fancy court neckline. It's a loose fit which makes the dress look comfortable, yet elegant and stylish. Her arms are completely uncovered.
SUBSCRIBE TO NEWS LETTER
TRACK YOUR ORDER
Checkout
1
Mobile number must only have 9 characters without the initial 0
OK
Buyer email *
jorge81@example.com
Delivery Type
Home delivery
Delivery Address *
2485 Franecki Trafficway Apt. 023
Use the above email address for payment details
NEXT

Payment cash on delivery



Order success



Xmart Shopping Plaza

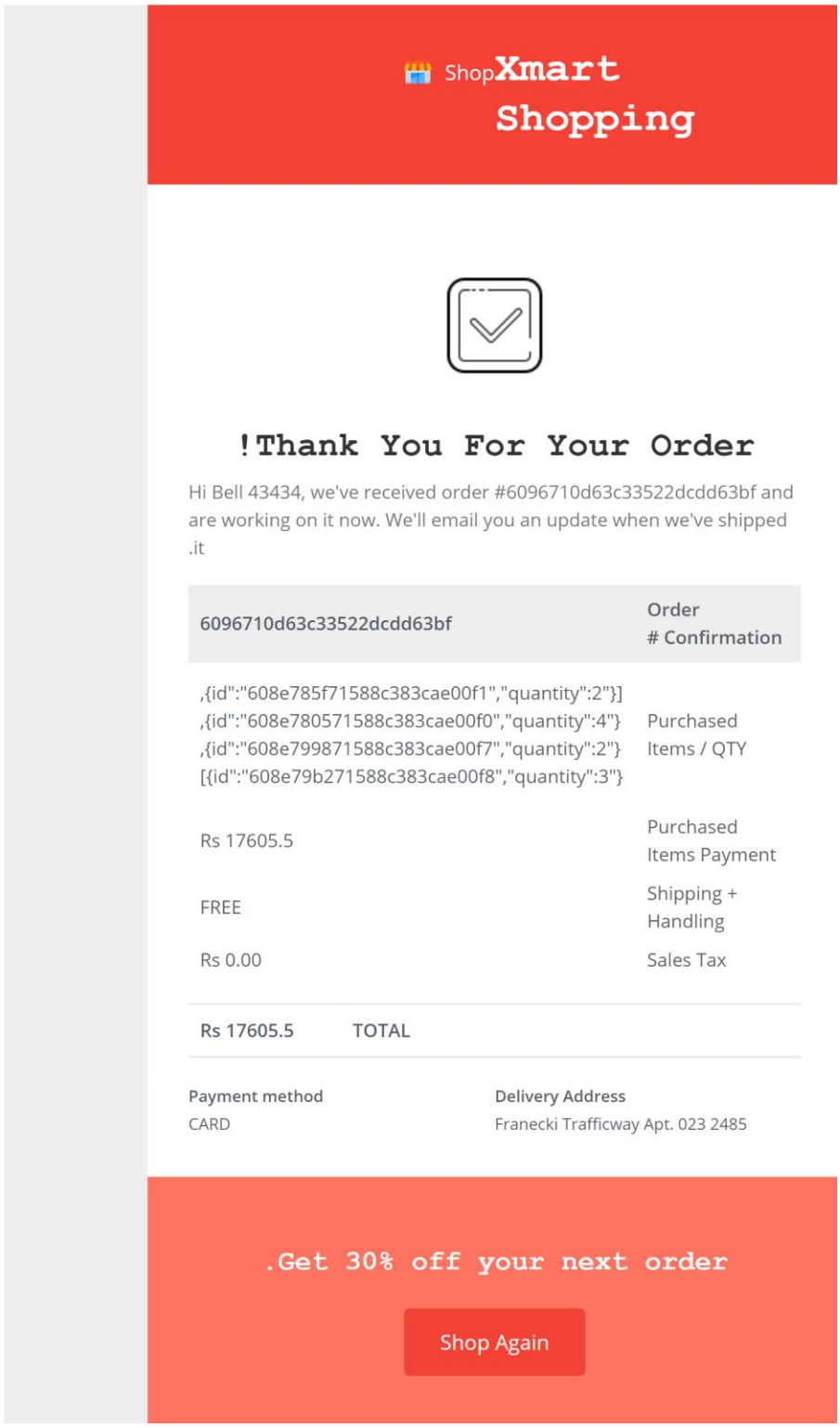
Sri Lanka's largest shopping platform

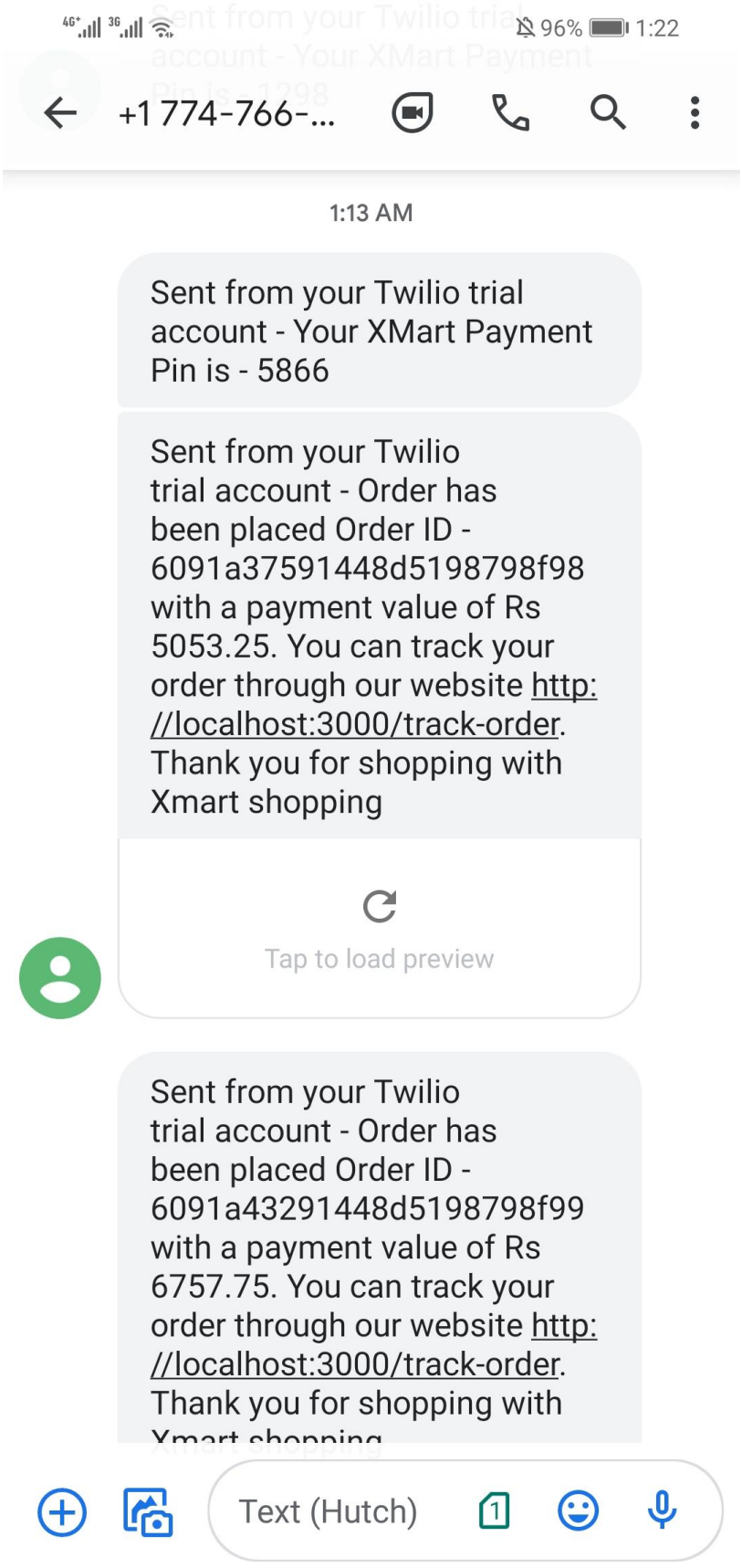
Copyright © Xmart.com 2021.

تاریخ:
16:38:42 08-05-2021

ronicyteam@gmail.com

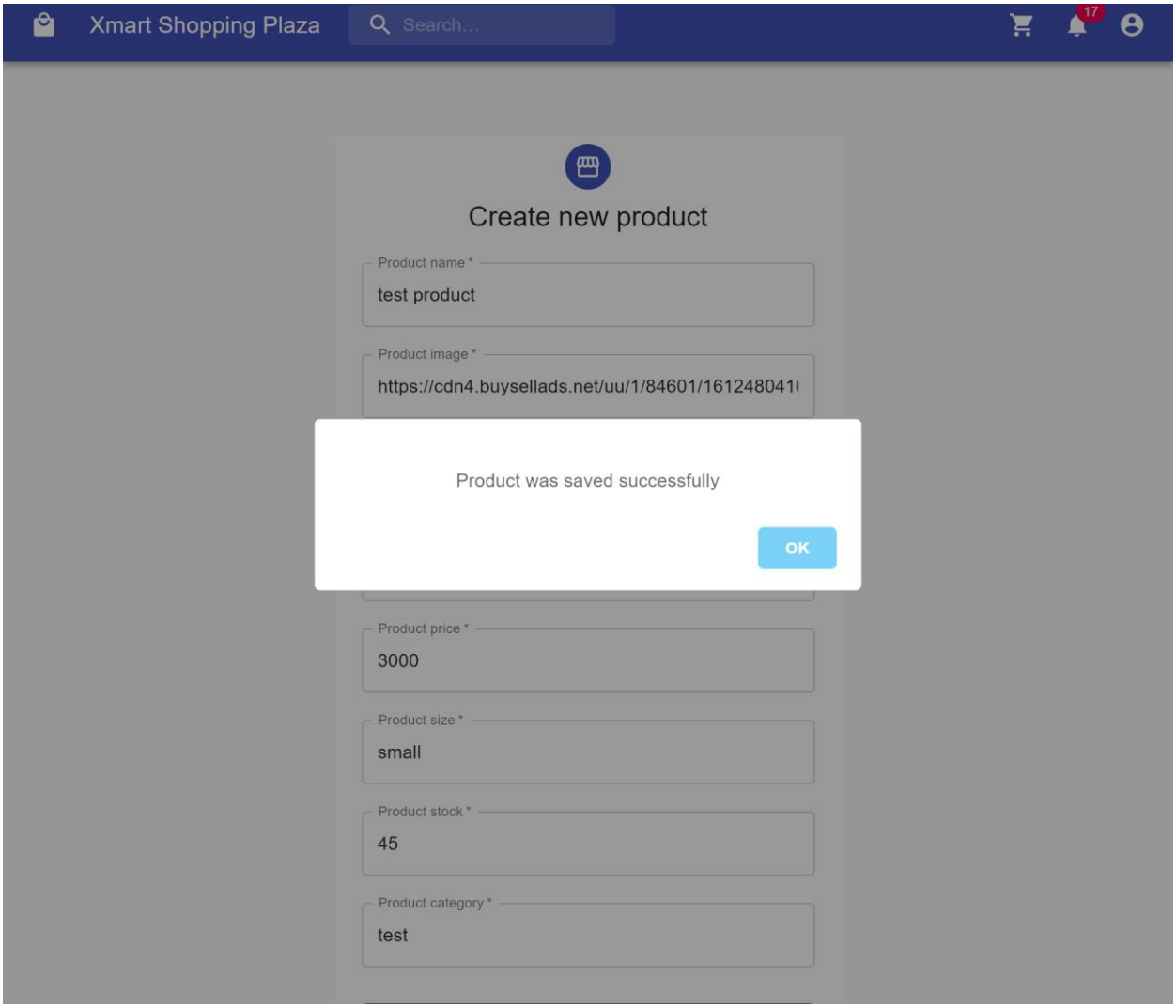
Order has been placed - 6096710d63c33522dcdd63bf



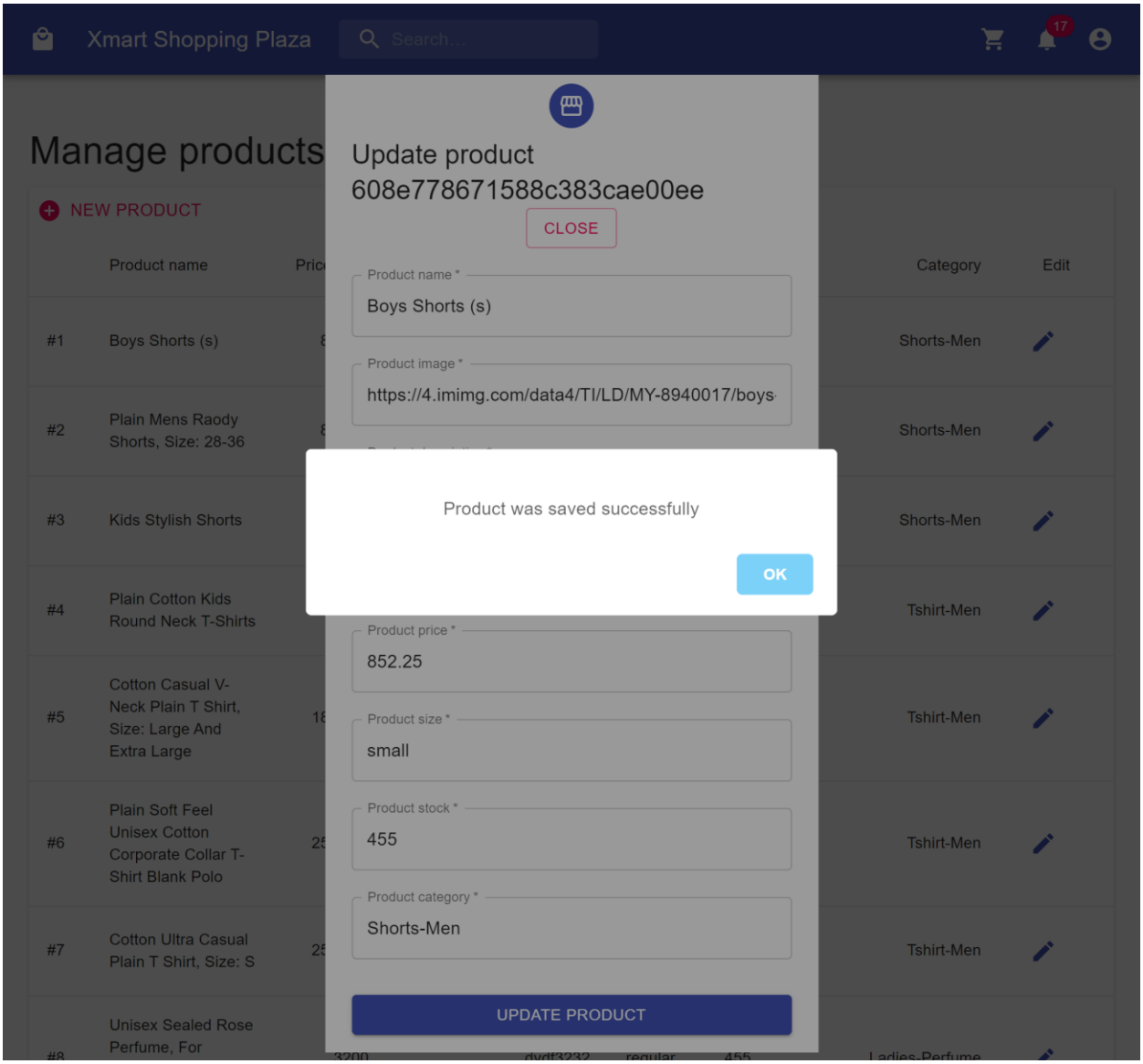

96% 1:22
+1 774-766-...
1:13 AM
Sent from your Twilio trial account - Your XMart Payment Pin is - 5866
Sent from your Twilio trial account - Order has been placed Order ID - 6091a37591448d5198798f98 with a payment value of Rs 5053.25. You can track your order through our website http://localhost:3000/track-order. Thank you for shopping with Xmart shopping
Tap to load preview
Sent from your Twilio trial account - Order has been placed Order ID - 6091a43291448d5198798f99 with a payment value of Rs 6757.75. You can track your order through our website http://localhost:3000/track-order. Thank you for shopping with
Text (Hutch)

Seller dashboard list products

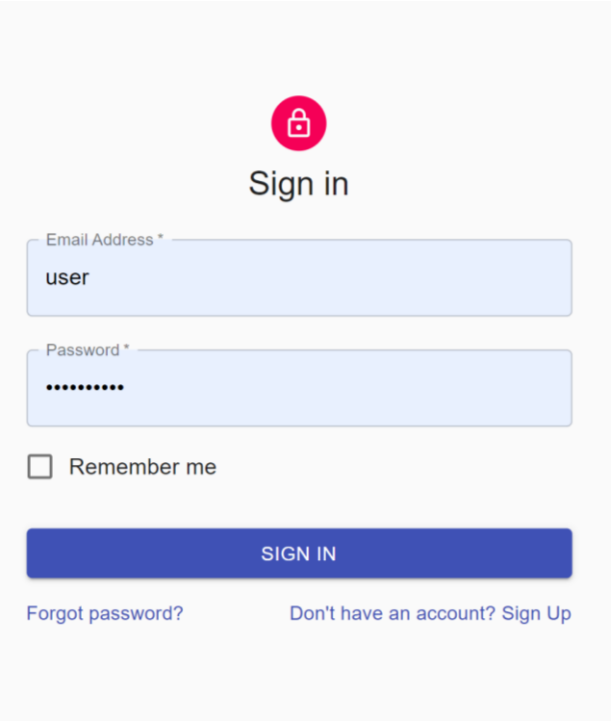
Seller dashboard add products



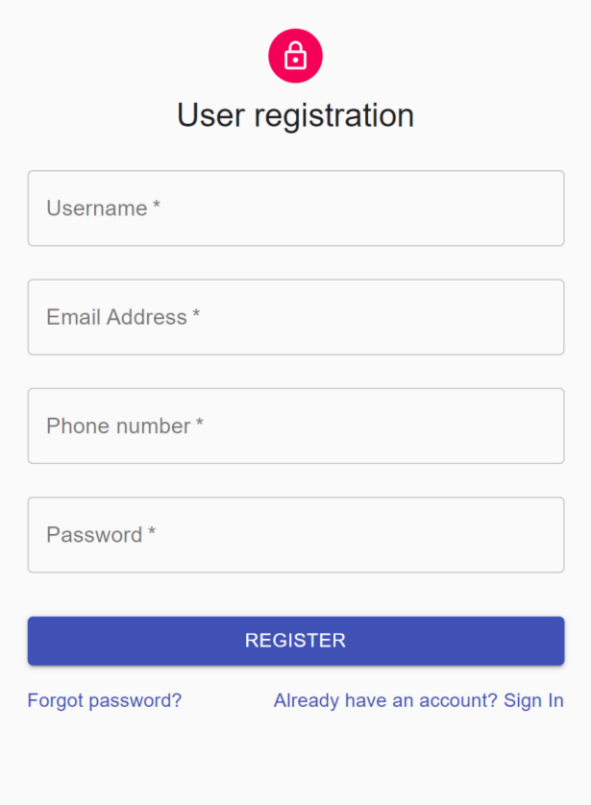
Seller dashboard update products



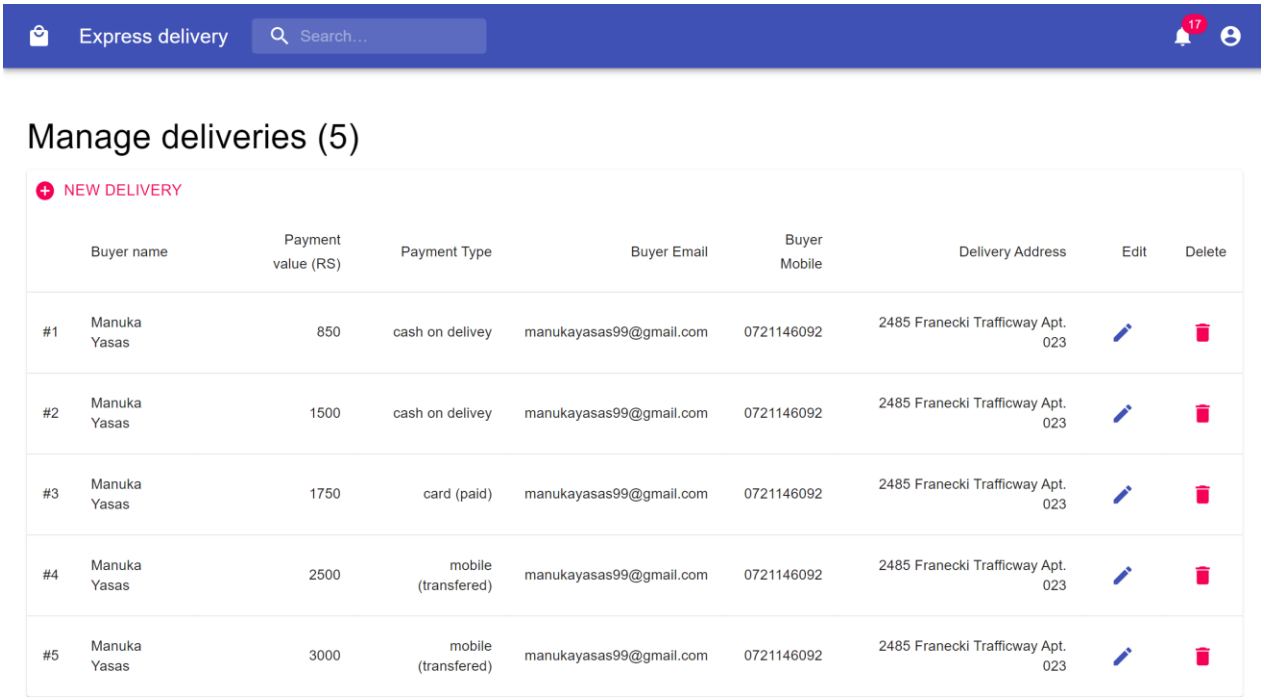
Delivery dashboard login



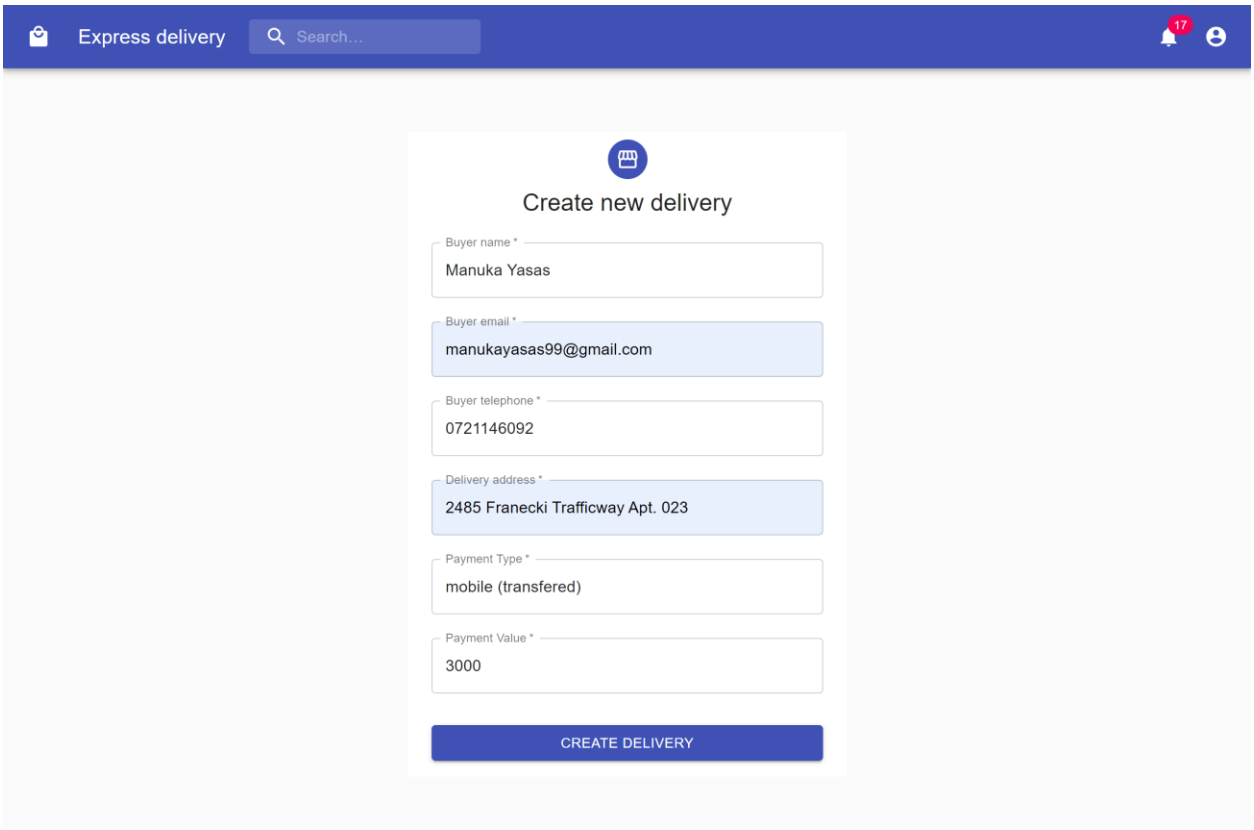
Delivery dashboard register



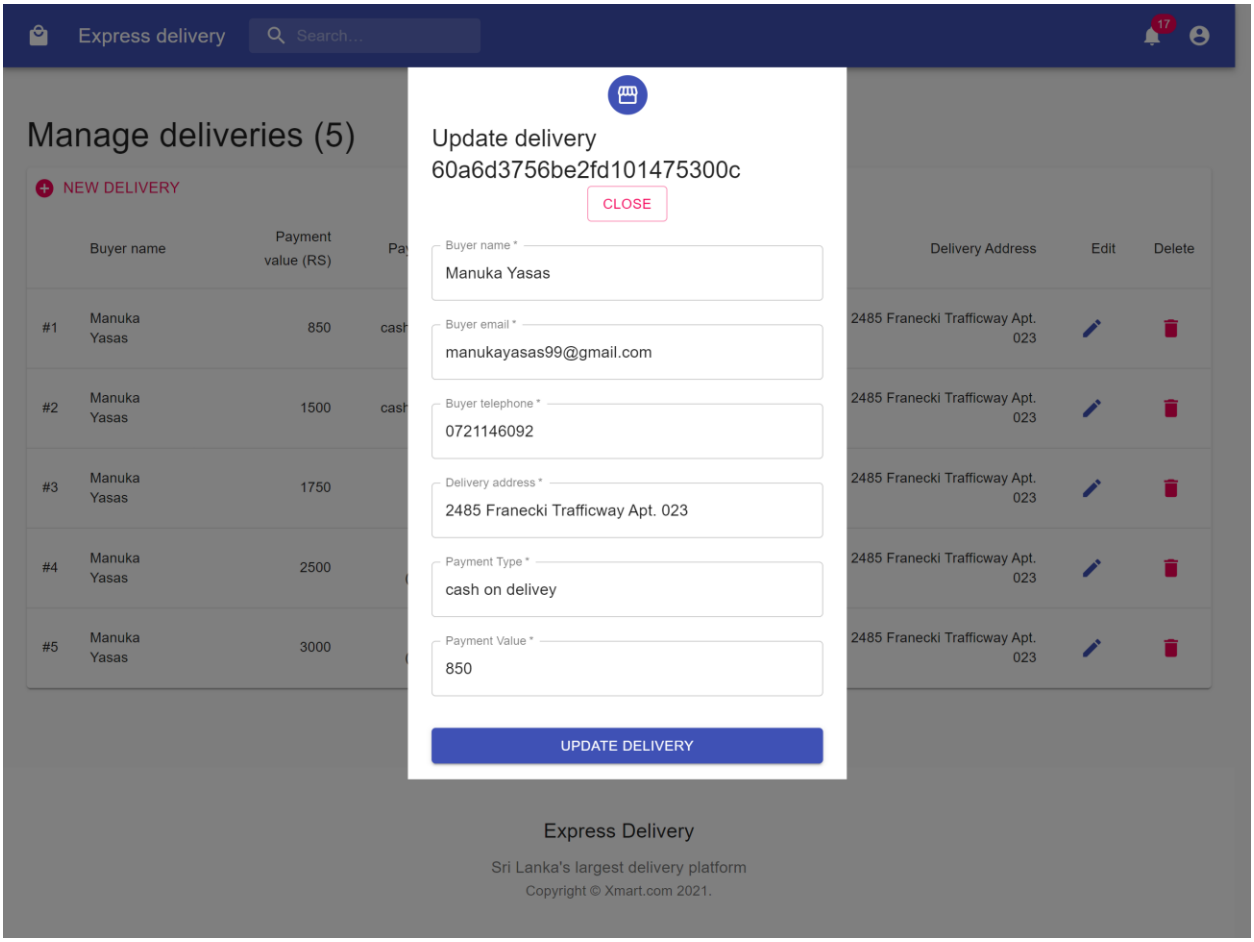
Delivery dashboard list delivery



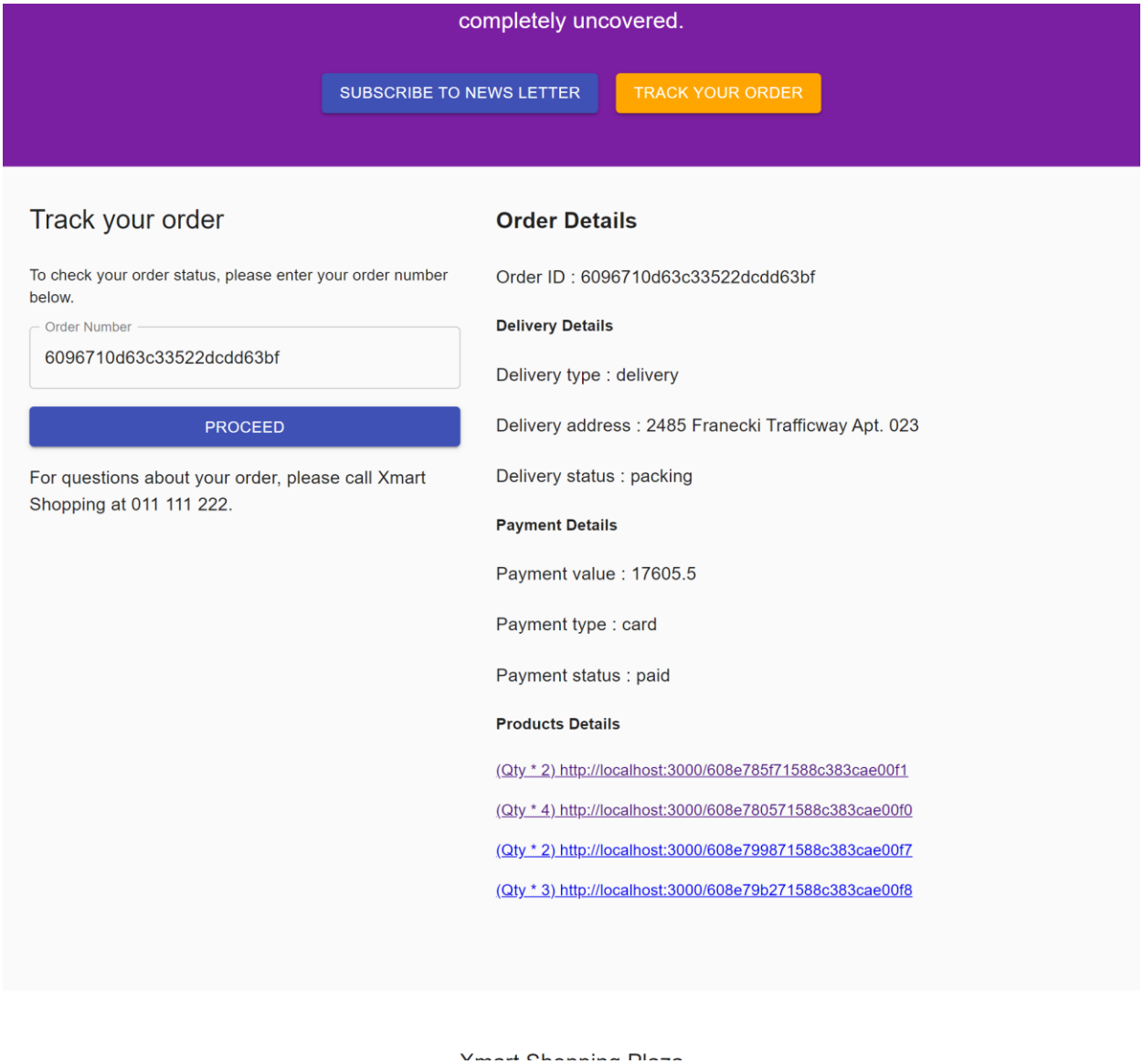
Delivery dashboard add delivery



Delivery dashboard update delivery



Track order


completely uncovered.
SUBSCRIBE TO NEWS LETTER
TRACK YOUR ORDER
Track your order
To check your order status, please enter your order number below.
Order Number
6096710d63c33522dcdd63bf
PROCEED
For questions about your order, please call Xmart Shopping at 011 111 222.
Order Details
Order ID : 6096710d63c33522dcdd63bf
Delivery Details
Delivery type : delivery
Delivery address : 2485 Franecki Trafficway Apt. 023
Delivery status : packing
Payment Details
Payment value : 17605.5
Payment type : card
Payment status : paid
Products Details
(Qty * 2) http://localhost:3000/608e785f71588c383cae00f1
(Qty * 4) http://localhost:3000/608e780571588c383cae00f0
(Qty * 2) http://localhost:3000/608e799871588c383cae00f7
(Qty * 3) http://localhost:3000/608e79b271588c383cae00f8

6. Appendix

6.1.Rest Services

6.1.1. Authentication service

server.js

```
require('dotenv').config({path:"./config.env"});
const express = require ("express");
const connectDB = require ('./config/db');
const errorHandler = require('./middleware/error');

// Connect DB
connectDB();
const app = express();

app.use(express.json());
app.use('/api/auth/',require('./routes/auth_rou'));
app.use('/api/private/',require('./routes/private_rou'));

// Error Handler  middleware

app.use(errorHandler);

const PORT = process.env.PORT || 5000;

const server = app.listen(PORT,() => console.log ('Server running on port '+PORT)
);

process.on("unhandledRejection",(err, promise) => {
    console.log('Logged Error:' +err);
    server.close (() => process.exit (1) );
})
```

Package.json

```
{
  "name": "ds_project",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "server": "nodemon server.js"
  },
  "keywords": [],
  "author": "Harini Gunawardana IT19215884",
  "license": "ISC",
  "dependencies": {
    "axios": "^0.21.1",
    "bcryptjs": "^2.4.3",
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "jsonwebtoken": "^8.5.1",
    "mongoose": "^5.12.6",
    "nodemailer": "^6.6.0",
    "react-router-dom": "^5.2.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.7"
  }
}
```

Config/db.js

```
const mongoose = require ('mongoose');
const connectDB = async() => {
    await mongoose.connect(process.env.MONGO_URI,{
        useNewUrlParser: true,
        useCreateIndex: true,
        useUnifiedTopology: true,
        useFindAndModify: true,

    });

    console.log(" MongoDB Connected");
};

    module.exports = connectDB;
```

Controller/auth.js

```
const crypto = require("crypto");
const User = require("../models/User");
const sendEmail = require("../utils/sendEmail");

exports.register = async (req, res) => {
  // Register
  const {
    firstname,
    lastname,
    username,
    email,
    telephone,
    password,
    userrole,
  } = req.body; //Register
  try {
    const user = await User.findOne({ email }); // check wheather there is a user
 with the requested email
    if (!user) {
      const user = await User.create({
        firstname,
        lastname,
        username,
        email,
        telephone,
        password,
        userrole,
      }); //if not, create the user
      sendToken(user, 201, res);
    } else {
      return res
        .status(400)
        .json({
          success: false,
          error: "Email in use. Please provide new email",
        });
    }
  } catch (error) {
    console.log(error);
  }
};

exports.login = async (req, res) => {
```

```
  //Login
  const { email, password } = req.body; // check that email and password are prov
ided
  if (!email || !password) {
    return res
      .status(422)
      .json({ success: false, error: "Please provide email and password" }); // b
ad request
  }

  try {
    const user = await User.findOne({ email }).select("+password"); // check that
 user is avaliable by email

    if (!user) {
      return res
        .status(400)
        .json({ success: false, error: "Invalid credentials" });
    }

    const isMatch = await user.matchPasswords(password); // check that passwords
match

    if (!isMatch) {
      return res
        .status(404)
        .json({ success: false, error: "Invalid credentials" });
    }

    sendToken(user, 200, res);
  } catch (error) {
    res.status(500).json({ success: false, error: error.messeage });
  }
};
exports.forgotPassword = async (req, res) => {
  // Forgot Password
  const { email } = req.body;
  try {
    const user = await User.findOne({ email }); // checks if email exists or not
    if (!user) {
      return res
        .status(404)
        .json({ success: false, error: "Email could not be sent" });
    }
```

```
    const resetToken = user.getResetPasswordToken(); // Generate reset token, con
vert into hash code and store
    await user.save();
    const resetUrl = `http://localhost:3000/passwordreset/${resetToken}`; // mess
age with url
    const message = `
                        <h1> You have requested a password reset </h1>
                        <p> Please go to this link to reset your password </p>
                        <a href=${resetUrl} clicktracking=off>${resetUrl}</a>
                      `;
    try {
      await sendEmail({
        to: user.email,
        subject: "Password reset request",
        text: message,
      });
      res.status(200).json({ success: true, data: "Email sent" });
    } catch (error) {
      user.resetPasswordToken = undefined;
      user.resetPasswordExpire = undefined;
      await user.save();
      return res
        .status(500)
        .json({ success: false, error: "Email could not be sent" });
    }
  } catch (error) {
    console.log(error);
  }
};

exports.resetPassword = async (req, res) => {
  //Reset Password
  const resetPasswordToken = crypto
    .createHash("sha256")
    .update(req.params.resetToken)
    .digest("hex"); // compare the token in url and hashed version

  try {
    const user = await User.findOne({
      resetPasswordToken,
      resetPasswordExpire: { $gt: Date.now() }, // token expiration
    });

    if (!user) {
      return res
```

```
        .status(400)
        .json({ success: false, error: "Invalid Reset Token" });
    }
    user.password = req.body.password;
    user.resetPasswordToken = undefined;
    user.resetPasswordExpire = undefined;
    await user.save();

    res.status(201).json({
      success: true,
      data: "Password Updated Success",
      token: user.getSignedJwtToken(),
    });
  } catch (error) {
    console.log(err);
  }
};

const sendToken = (user, statusCode, res) => {
  const token = user.getSignedJwtToken();
  res.status(statusCode).json({ success: true, token });
};

exports.ValidateToken = async (req, res) => {
  try {
    // validate token in request
    if (!req.body.token)
      return res.status(422).json({ message: "JWT token is required" });
    // validate JWT token
    var decodedToken = jwt.verify(req.body.token, KEY);
    // fetch user token token user id
    var user = await User.findById(decodedToken.user_id);
    return res.status(200).json(user);
  } catch (error) {
    return res.status(400).json({ message: "JWT token is not valid" });
  }
};
```

Controller/private.js

```
exports.getPrivateData = (req,res,next) => {
    res.status(200).json({
        success:true,
        data:"You got access to the private data in this route",

    });
};
```

Middleware/auth.js

```
// Check for json web tokens in the header

const jwt = require('jsonwebtoken');
const User = require ('../models/User');
const ErrorResponse = require('../utils/errorResponse');

exports.protect = async (req,res,next)=>{
    let token;

    if(req.headers.authorization && req.headers.authorization.startsWith("Bearer"
))
    {
       token = req.headers.authorization.split("")[1]
    }
    if(!token)
    {
      return res.status(401).json ({success:false, error:"Not authorized to acces
s this route"});
    }
    try
    {
        const decoded = jwt.verify(token,process.env.JWT_SECRET);
        const user = await user.findById(decoded.id); // find the user by token
        if(!user)
        {
            return next(new ErrorResponse ("No user found with this id",404)); //
 if the user not found return the error
        }
        req.user= user;
        next();
    }
    catch (error)
    {
```

```
        return next (new ErrorResponse (" Not authorized to access this route",40
1));
    }
};
```

Middleware/error.js

```
const ErrorResponse = require("../utils/errorResponse");

const errorHandler = (err, req, res, next) => {
  let error = { ...err };

  error.message = err.message;

  if (err.code === 11000) {
    const message = `Duplicate Field value entered`;
    error = new ErrorResponse(message, 400);
  }

  if (err.name === "ValidationError") {
    const message = Object.values(err.errors).map((val) => val.message);
    error = new ErrorResponse(message, 400);
  }

  console.log(error.message);

  res.status(error.statusCode || 500).json({
    success: false,
    error: error.message || "Server Error",
  });
};

module.exports = errorHandler;
```

Models/User.js

```
const crypto = require('crypto');
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');



//const { resetPassword } = require('../routes/auth');
const UserSchema = new mongoose.Schema(
    {

        firstname: {
            type: String,
            required:[true, "Please provide first name"]  // validations
        },


       lastname: {
            type: String,
            required:[true, "Please provide last name"]  // validations
        },

        username: {
            type: String,
            required:[true, "Please provide a username"]  // validations
        },

        email: {
            type: String,
            required:[true, "Please provide an email"] ,   // validations
            unique: true,
            match: [
                /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[
[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-
Z]{2,}))$/,
                "Please provide a valid email",
              ],
            },

            telephone: {
                type: String,
```

```
            required:[true, "Please provide a telephone number"] , //validati
ons

        },



        password: {
            type: String,
            required: [true,"Please provide a password"], // validations
            minlength:6,
            select:false   // to avoid passing the password when quering unle
ss specifically asked for it.
        },


        userrole: {
            type: String,
            required:[true, "Please provide the role"]  // validations
        },


        resetPasswordToken: String,
        resetPasswordExpire: Date
    }
);


UserSchema.pre("save", async function(next){  //
    if (!this.isModified("password")){
        next();
    }

const salt = await bcrypt.genSalt(10);    // await bcz it returns a promise
this.password = await bcrypt.hash (this.password,salt); // this refers to passwor
d in controller/auth.js
// Change the password that was sent, save  the new password in password field

next();

});


UserSchema.methods.matchPasswords = async function (password)
{
```

```
    return await bcrypt.compare (password, this.password);    // comparing passwo
rds
};

UserSchema.methods.getSignedJwtToken =  function ()
{
   return jwt.sign({id:this._id}, process.env.JWT_SECRET,{expiresIn: process.env.
JWT_EXPIRE}); //
};

UserSchema.methods.getResetPasswordToken = function()
{
    const resetToken = crypto.randomBytes(20).toString("hex");
    this.resetPasswordToken = crypto.createHash("sha256").update(resetToken).dige
st("hex");
    this.resetPasswordExpire = Date.now() + 10* (60*1000);// 10 mins  //Reset Pas
sword token  expires on 10 mins
    return resetToken;
}

const User = mongoose.model("User",UserSchema);
module. exports = User;
```

Routes/auth_rou.js

```
const express = require("express");
const router = express.Router();

const {
  register,
  login,
  forgotPassword,
  resetPassword,
  ValidateToken,
} = require("../controller/auth");

// register user
router.post("/register", register);

// authenticate
router.post("/login", login);

// recover password
router.post("/forgotPassword", forgotPassword);
```

```
// accepting reset token when resetting the password
router.put("/passwordreset/:resetToken", resetPassword);

// validate token
router.post("/validateToken", ValidateToken);

module.exports = router;
```

Routes/private_rou.js

```
const express = require("express");
const { getPrivateData } = require("../controller/private");
const router = express.Router();
const { protect } = require("../middleware/auth");

router.route("/").get(protect, getPrivateData);

module.exports = router;
```

Utils/errorResponse.js

```
  // Blue print to error response message

  class ErrorResponse extends Error

  {
      constructor (message, statusCode)
      {
          super(message);
          this.statusCode = statusCode;
      }

  }

  module.exports = ErrorResponse;
```

Utils/sendMail.js

```
const nodemailer = require("nodemailer");

const sendEmail = (options) => {
  const transporter = nodemailer.createTransport({
    service: process.env.EMAIL_SERVICE,
    auth: {
      user: process.env.EMAIL_USERNAME,
      pass: process.env.EMAIL_PASSWORD,
    },
  });

  const mailOptions = {
    from: process.env.EMAIL_FROM,
    to: options.to,
    subject: options.subject,
    html: options.text,
  };

  transporter.sendMail(mailOptions, function (err, info) {
    if (err) {
      console.log(err);
    } else {
      console.log(info);
    }
  });
};

module.exports = sendEmail;
```

6.1.2. Seller service

Index.js

```
const bodyParser = require("body-parser");
const cors = require("cors");
const { connect } = require("mongoose");
const express = require("express");
const { DB, PORT } = require("./config");
const Authenticate = require("./middlewares/Authenticate");
const SellerAuth = require("./middlewares/SellerAuth");

const app = express();

app.use(bodyParser.json());
app.use(cors());

app.all("*", Authenticate);
app.use("/api/admin/product", SellerAuth, require("./routes/ProductRoutes"));

const startApp = async () => {
  try {
    await connect(DB, {
      useFindAndModify: true,
      useUnifiedTopology: true,
      useNewUrlParser: true,
    });

    console.log("Connected to database");

    await app.listen(PORT, () =>
      console.log(`App is listening on port ${PORT}`)
    );
  } catch (error) {
    console.error(error);
    startApp();
  }
};

startApp();
```

Package.json

```
{
  "name": "buyer_service",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "Manuka Yasas",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.19.0",
    "cors": "^2.8.5",
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "mongoose": "^5.12.5",
    "nodemon": "^2.0.7",
    "request": "^2.88.2"
  }
}
```

Config/index.js

```
require("dotenv").config();

module.exports = {
  DB: process.env.APP_DB,
  PORT: process.env.APP_PORT,
  AUTH_GATEWAY: process.env.AUTH_GATEWAY,
};
```

.env

```
APP_DB = mongodb://127.0.0.1:27017/ds_buyer_service
APP_PORT = 5006
AUTH_GATEWAY = http://192.168.61.1:8280/auth/validateToken
```

Controller/productController.js

```
const Product = require("../model/Product");

// list all the products
exports.listProducts = async (req, res) => {
  try {
    let products = await Product.find();
    return res.status(200).json(products);
  } catch (error) {
    console.error(error);
    return res.status(400).json({ message: "Error when retrriving products" });
  }
};

// find product based on id
exports.findProduct = async (req, res) => {
  try {
    let product = await Product.findById(req.params.pid);
    if (product) return res.status(200).json(product);
    else return res.status(400).json({ message: "Product not found" });
  } catch (error) {
    console.error(error);
    return res.status(400).json(error);
  }
};

// create / update a product
exports.saveProduct = async (req, res) => {
  try {
    // initialize product to be be created or update
    var product = new Product();

    // if product id is sent then retrive that product and update else create new
 product.
    if (req.params.pid) product = await Product.findById(req.params.pid);

    // set new product data
    product.name = req.body.name;
    product.image = req.body.image;
    product.description = req.body.description;
    product.price = req.body.price;
    product.sku = req.body.sku;
    product.size = req.body.size;
```

```
    product.stock = req.body.stock;
    product.category = req.body.category;
    // save product
    var result = {};
    try {
      result = await product.save();
    } catch (error) {
      return res.status(422).json(error);
    }
    // updating / saving failes
    if (result && result.error)
      return res.status(400).json({
        message: "Error when saving product",
        product: result._doc,
      });
    // saved successfully
    return res.status(200).json({
      message: "Product was saved successfully",
      product: result._doc,
    });
  } catch (error) {
    console.error(error);
    return res.status(400).json({ message: "Product not found" });
  }
};

// delete a product
exports.deleteProduct = async (req, res) => {
  try {
    let result = await Product.findByIdAndDelete(req.params.pid);
    // delete failed
    if (result && result.error)
      return res.status(400).json({ message: "Error when deleting product" });
    // delete successfull
    return res
      .status(200)
      .json({ message: "Product was deleted successfully" });
  } catch (error) {
    console.error(error);
    return res.status(400).json(error);
  }
};
```

Middlewares/Authenticate.js

```
const { getAuthUserFromBearerToken } = require("../util/Auth");

const Authenticate = async (req, res, next) => {
  // get the auth user from token by the auth service
  let user = await getAuthUserFromBearerToken(req.header("authorization"));
  if (user) req.user = user;
  return next();
};

module.exports = Authenticate;
```

Middlewares/SellerAuth.js

```
const SellerAuth = async (req, res, next) => {
  if (req.user && req.user.role == "seller") return next();
  else return res.send(401, "Unauthorized");
};

module.exports = SellerAuth;
```

Model/Product.js

```
const { model, Schema } = require("mongoose");

const productSchema = new Schema(
  {
    name: {
      type: String,
      required: [true, "Please enter a valid product name"],
      minlength: [8, "Product name must have at least 8 characters"],
    },
    image: {
      type: String,
      required: [true, "Please enter a valid product image"],
```

```
      minlength: [8, "Product image must have at least 8 characters"],
    },
    description: {
      type: String,
      required: [true, "Please enter a valid product description"],
      minlength: [20, "Product description must have at least 20 characters"],
    },
    category: {
      type: String,
      required: [true, "Please enter a valid product category"],
      minlength: [4, "Product category must have at least 4 characters"],
    },
    price: {
      type: Number,
      required: [true, "Please enter a valid product price"],
      minlength: [3, "Product price must have at least 3 characters"],
    },
    sku: {
      type: String,
      required: [true, "Please enter a valid product sku"],
      minlength: [8, "Product sku must have at least 8 characters"],
    },
    size: {
      type: String,
      required: [true, "Please enter a valid product size"],
      minlength: [2, "Product size must have at least 2 characters"],
    },
    stock: {
      type: Number,
      required: [true, "Please enter a valid product name"],
      minlength: [1, "Product stock must have at least 1 character"],
    },
  },
  { timestamps: true }
);

productSchema.index({ name: "text", description: "text" });

module.exports = model("product", productSchema);
```

Routes/ProductRoutes.js

```
const router = require("express").Router();
const ProductController = require("../controllers/ProductController");

// save product
router.post("/", ProductController.saveProduct);

// update product
router.put("/:pid", ProductController.saveProduct);

// get all products
router.get("/", ProductController.listProducts);

// get product based on id
router.get("/:pid", ProductController.findProduct);

// delete product
router.delete("/:pid", ProductController.deleteProduct);

module.exports = router;
```

Util/Auth.js

```
const util = require("util");
var request = require("request");
const { AUTH_GATEWAY } = require("../config");

// contact auth service through esb to validate the jwt token and get the user
const getAuthUserFromBearerToken = async (bearer_token) => {
  var token = String(bearer_token).slice(7);
  const options = {
    url: AUTH_GATEWAY,
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    maxRedirects: 20,
    body: JSON.stringify({
      token: `${token}`,
    }),
  };

  const requestPromise = util.promisify(request);
  try {
    const response = await requestPromise(options);
    // console.log(response.body);
    let jsonBody = response.body ? JSON.parse(response.body) : {};
    if (response.statusCode == 200 && jsonBody._id) return jsonBody;
    else return null;
  } catch (error) {
    console.error(error);
  }
};

module.exports = { getAuthUserFromBearerToken };
```

6.1.3. Buyer service

index.js

```
const bodyParser = require("body-parser");
const cors = require("cors");
const { connect } = require("mongoose");
const express = require("express");
const { DB, PORT } = require("./config");
const Authenticate = require("./middlewares/Authenticate");

const app = express();

app.use(bodyParser.json());
app.use(cors());

app.all("*", Authenticate);
app.use("/api/product", require("./routes/ProductRoutes"));
app.use("/api/order", require("./routes/OrderRoutes"));
app.use("/api/cart", require("./routes/CartRoutes"));
app.use("/api/payment", require("./routes/PaymentRoutes.js"));

const startApp = async () => {
  try {
    await connect(DB, {
      useFindAndModify: true,
      useUnifiedTopology: true,
      useNewUrlParser: true,
    });

    console.log("Connected to database");

    await app.listen(PORT, () =>
      console.log(`App is listening on port ${PORT}`)
    );
  } catch (error) {
    console.error(error);
    startApp();
  }
};

startApp();
```

package.json

```
{
    "name": "buyer_service",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1",
        "start": "nodemon index.js"
    },
    "keywords": [],
    "author": "Manuka Yasas",
    "license": "ISC",
    "dependencies": {
        "axios": "^0.21.1",
        "body-parser": "^1.19.0",
        "cors": "^2.8.5",
        "dotenv": "^8.2.0",
        "express": "^4.17.1",
        "fs": "0.0.1-security",
        "handlebars": "^4.7.7",
        "jsonwebtoken": "^8.5.1",
        "md5": "^2.3.0",
        "mongoose": "^5.12.5",
        "nodemailer": "^6.6.0",
        "nodemailer-smtp-transport": "^2.7.4",
        "nodemon": "^2.0.7",
        "passport": "^0.4.1",
        "passport-jwt": "^4.0.0",
        "request": "^2.88.2",
        "twilio": "^3.61.0"
    }
}
```

Config/index.js

```
require("dotenv").config();

module.exports = {
  DB: process.env.APP_DB,
  PORT: process.env.APP_PORT,
  KEY: process.env.APP_KEY,
  AUTH_GATEWAY: process.env.AUTH_GATEWAY,
  MAIL_USER: process.env.MAIL_USER,
  MAIL_PASSWORD: process.env.MAIL_PASSWORD,
  TWILIO_ACCOUNT_SID: process.env.TWILIO_ACCOUNT_SID,
  TWILIO_AUTH_TOKEN: process.env.TWILIO_AUTH_TOKEN,
  TWILIO_NUMBER: process.env.TWILIO_NUMBER,
  PAYMENT_SECRET: process.env.PAYMENT_SECRET,
  ORDER_SECRET: process.env.ORDER_SECRET,
};
```

controller/CartController.js

```
const Cart = require("../model/Cart");
const Product = require("../model/Product");
const OrderValidator = require("../validators/OrderValidator");

exports.addToCart = async (req, res) => {
  try {
    // validate the products in the cart
    //use the validator used to validate order products
    var validatedOrder = await OrderValidator.ValidateOrderProducts(req, res);

    var cart = await Cart.findOne({ user_id: req.user._id });

    if (!cart) {
      // ifthe user does not habve a cart
      cart = new Cart({
        user_id: req.user._id,
        products: validatedOrder.products,
        payment_value: validatedOrder.payment_value,
      });
    } else {
      // if the user already have a cart
      cart.products = [...cart.products, ...validatedOrder.products];
      cart.payment_value =
        validatedOrder.payment_value + validatedOrder.payment_value;
```

```
    }
    var result = await cart.save();

    // if cart save fail
    if (result && result.error) return res.status(400).json(result);

    return res
      .status(200)
      .json({ message: "All changes was saved", cart: result._doc });
  } catch (error) {
    console.error(error);
    return res.status(400).json({ message: "Unexpected error" });
  }
};

// get cart od the logged in user
exports.getCart = async (req, res) => {
  try {
    var cart = await Cart.findOne({ user_id: req.user._id });
    // if the cart exist and if the cart has products, get the products rom the c
art
    if (cart && cart.products.length > 0) {
      for (var index = 0; index < cart.products.length; index++) {
        try {
          // add all the product data
          cart.products[index].data = await Product.findById(
            cart.products[index].id
          );
        } catch (error) {
          console.log(error);
        }
      }
    }
    return res.status(200).json(cart ? cart : {});
  } catch (error) {
    console.error(error);
    return res.status(400).json({ message: "User cart not found" });
  }
};

// add, update or delete a product in cart
exports.storeToCart = async (req, res) => {
  try {
    var validatedCart = {};
    // validate the products in the cart if it has products only
```

```
    if (
      req.body.products &&
      Array.isArray(req.body.products) &&
      req.body.products.length > 0
    )
      validatedCart = await OrderValidator.ValidateOrderProducts(req, res);
    else {
      // no cart products means that user have deleted all the products from the
cart
      // we can allow it because the cart can be empty
      validatedCart.products = [];
      validatedCart.payment_value = 0;
    }
    // get cart details
    var cart = await Cart.findOne({ user_id: req.user._id });
    if (!cart) {
      // ifthe user does not habve a cart
      cart = new Cart({
        user_id: req.user._id,
        products: validatedCart.products,
        payment_value: validatedCart.payment_value,
      });
    } else {
      // if the user already have a cart
      cart.products = validatedCart.products;
      cart.payment_value = validatedCart.payment_value;
    }
    // save cart
    var result = await cart.save();
    if (result && result.error) return res.status(400).json(result);
    return res
      .status(200)
      .json({ message: "All changes was saved", cart: result._doc });
  } catch (error) {
    console.error(error);
    return res.status(400).json({ message: "Invalid cart details" });
  }
};
```

controller/OrderController.js

```
const { ORDER_SECRET } = require("../config");
const Order = require("../model/Order");
var md5 = require("md5");
const OrderValidator = require("../validators/OrderValidator");

// new order
exports.newOrder = async (req, res) => {
  try {
    // validate the order products including product id and product stock
    var validatedOrder = await OrderValidator.ValidateOrderProducts(req, res);

    //save the order
    let order = new Order({
      order_status: "pending",
      user_id: req.user._id,
      products: validatedOrder.products,
      payment_value: validatedOrder.payment_value,
    });

    let result = await order.save();
    if (result && result.error) return res.status(400).json(result);
    else return res.status(200).json(result);
  } catch (error) {
    console.error(error);
    return res.status(400).json(error);
  }
};

// retrive the order tails based on id
exports.getOrderDetails = async (req, res) => {
  try {
    var order = await Order.findById(req.params.order_id);
    return res.status(200).json(order);
  } catch (error) {
    console.error(error);
    return res.status(400).json({ message: "Order not found" });
  }
};

// after the order is created user can save the buyer info and the delivery info
exports.saveOrderDetails = async (req, res) => {
  try {
    // validate the requset
```

```
    var validatedOrder = await OrderValidator.ValidateOrderDetails(req, res);

    // get the order
    var order = await Order.findById(req.params.order_id);

    order.buyer_name = validatedOrder.buyer_name;
    order.buyer_email = validatedOrder.buyer_email;
    order.buyer_phone = validatedOrder.buyer_phone;
    order.delivery_type = validatedOrder.delivery_type;
    order.delivery_address = validatedOrder.delivery_address;

    // save details
    var result = await order.save();
    if (result && result.error) return res.status(400).json(result.error);
    else {
      // hash parameters of the order to confirm payment details from gateways
      var hash_order_code = md5(
        `${ORDER_SECRET}${order._id}${order.payment_value}`
      );
      result = { ...result._doc, hash_order_code };
      return res.status(200).json(result);
    }
  } catch (error) {
    console.error(error);
    return res.status(400).json({ message: "Order not found" });
  }
};
```

controller/PaymentController.js

```
var md5 = require("md5");
const { PAYMENT_SECRET } = require("../config");
const Order = require("../model/Order");
const {
  notifyPaymentSuccessfull,
  notifyPaymentFailed,
} = require("../util/Payment");
const OrderValidator = require("../validators/OrderValidator");

// sent from payment gateways
// payement gateways will send info after the payemnt completed
exports.paymentOrderNotification = async (req, res) => {
  try {
    var order = await Order.findById(req.body.order_id);

    // hash paras and validate if the payment was made
    var hash_pay_code_valid = md5(
      `${PAYMENT_SECRET}${order._id}${order.payment_value}`
    );

    if (hash_pay_code_valid == req.body.hash_pay_code) {
      // complete order if hashed matches
      console.log("Valid payment");
      order.payment_status = "paid";
      order.delivery_status = "packing";
      order.order_status = "validating";
      order.payment_type = req.body.payment_type;
      var result = await order.save();

      // send error if saving failed
      if (result && result.error) {
        notifyPaymentFailed(order);
        return res.status(400).json(result);
      }

      // send email and mobile sms after completing payment
      notifyPaymentSuccessfull(order);
    } else {
      console.error("Failed to match hash in order");
      notifyPaymentFailed(order);
    }
  } catch (error) {
```

```
    console.error(error);
    console.error("Invalid payment");
    notifyPaymentFailed(order);
  }
  return res.status(200).json("ok");
};

// complete the cash on delivery order
exports.codPayment = async (req, res) => {
  try {
    var order = await Order.findById(req.body.order_id);

    // validateOrderDetails function accpts req , res.
    //validateOrderDetails will validate the req.body therefore create a new requ
est object and append order data and user to it other than writing a new function
.
    // append the order to the reuest body and the request user to the user
    var req_data = { body: { ...order._doc }, user: { ...req.user } };

    // validate order details
    var validatedOrder = OrderValidator.ValidateOrderDetails(req_data, res);

    order.payment_status = "pending";
    order.delivery_status = "packing";
    order.order_status = "validating";
    order.payment_type = "COD";
    var result = await order.save();

    if (result && result.error) {
      notifyPaymentFailed(order);
      return res.status(400).json(result);
    }

    // send email and mobile sms after completing payment
    notifyPaymentSuccessfull(order);
    return res
      .status(200)
      .json({ message: "Order was placed successfully", status: 1 });
  } catch (error) {
    console.error(error);
    notifyPaymentFailed(order);
    return res.status(400).json({ errors: { message: "Payment failed" } });
  }
};
```

controller/ProductController.js

```
const Product = require("../model/Product");

// list all the products
exports.listProducts = async (req, res) => {
  try {
    let products = await Product.find();
    return res.status(200).json(products);
  } catch (error) {
    console.error(error);
    return res.status(422).json(error);
  }
};

// find product based on id
exports.findProduct = async (req, res) => {
  try {
    let product = await Product.findById(req.params.pid);
    if (product) return res.status(200).json(product);
    else return res.status(422).json({ message: "Product not found" });
  } catch (error) {
    console.error(error);
    return res.status(422).json(error);
  }
};

// search products , full text search
exports.searchProducts = async (req, res) => {
  try {
    let products = await Product.find({
      $text: { $search: req.params.text },
    });
    if (products.length > 0) return res.status(200).json(products);
    else return res.status(400).json({ message: "Products not found" });
  } catch (error) {
    console.error(error);
    return res.status(400).json(error);
  }
};

// save a new product
exports.storeProduct = async (req, res) => {
  try {
    const newProduct = new Product({
```

```
      name: req.body.name,
      image: req.body.image,
      description: req.body.description,
      price: req.body.price,
      sku: req.body.sku,
      size: req.body.size,
      stock: req.body.stock,
      category: req.body.category,
    });
    const result = await newProduct.save();
    if (result && result.error) return res.status(422).json(result);
    else
      return res
        .status(200)
        .json({ message: "Product was saved successfully" });
  } catch (error) {
    console.error(error);
    return res.status(422).json(error);
  }
};

// update a product
exports.updateProduct = async (req, res) => {
  try {
    let product = await Product.findById(req.params.pid);
    if (product) {
      product.name = req.body.name;
      product.image = req.body.image;
      product.description = req.body.description;
      product.price = req.body.price;
      product.sku = req.body.sku;
      product.size = req.body.size;
      product.stock = req.body.stock;
      const result = await product.save();
      if (result && result.error) return res.status(422).json(result);
      else
        return res
          .status(200)
          .json({ message: "Product was updated successfully" });
    } else return res.status(422).json({ message: "Product not found" });
  } catch (error) {
    console.error(error);
    return res.status(422).json(error);
  }
};
```

```
// delete a product
exports.deleteProduct = async (req, res) => {
  try {
    let result = await Product.findByIdAndDelete(req.params.pid);
    if (result && result.error) return res.status(422).json(result);
    else
      return res
        .status(200)
        .json({ message: "Product was deleted successfully" });
  } catch (error) {
    console.error(error);
    return res.status(422).json(error);
  }
};

// buy a product
exports.buyProduct = async (req, res) => {
  try {
    let result = await Product.findByIdAndDelete(req.params.pid);
    if (result && result.error) return res.status(422).json(result);
    else
      return res
        .status(200)
        .json({ message: "Product was deleted successfully" });
  } catch (error) {
    console.error(error);
    return res.status(422).json(error);
  }
};
```

middlewares/Authenticate.js

```
const { getAuthUserFromBearerToken } = require("../util/Auth");

const Authenticate = async (req, res, next) => {
  // get the auth user from token by the auth service
  let user = await getAuthUserFromBearerToken(req.header("authorization"));
  if (user) req.user = user;
  return next();
};

module.exports = Authenticate;
```

middlewares/UserAuth.js

```
const { getAuthUserFromBearerToken } = require("../util/Auth");

const userAuth = async (req, res, next) => {
  if (req.user && req.user._id.length > 0) return next();
  else return res.send(401, "Unauthorized");
};

module.exports = userAuth;
```

model/Cart.js

```
const { model, Schema } = require("mongoose");

const cartSchema = new Schema({
    user_id: { type: String, required: true },
    products: { type: Array, required: true },
    payment_value: { type: Number, required: false },
}, { timestamps: true });

module.exports = model("cart", cartSchema);
```

model/Order.js

```
const { model, Schema } = require("mongoose");

const orderSchema = new Schema(
  {
    order_status: { type: String, required: true },
    user_id: { type: String, required: true },
    products: { type: Array, required: true },

    buyer_name: { type: String, required: false },
    buyer_email: { type: String, required: false },
    buyer_phone: { type: Number, required: false },

    payment_value: { type: Number, required: false },
    payment_type: {
      type: String,
      default: "card",
      enum: ["COD", "card", "mobile"],
    },
```

```
    payment_status: { type: String, required: false },

    delivery_type: {
      type: String,
      default: "delivery",
      enum: ["pickup", "delivery"],
    },
    delivery_address: { type: String, required: false },
    delivery_status: { type: String, required: false },
  },
  { timestamps: true }
);

module.exports = model("order", orderSchema);
```

model/Product.js

```
const { model, Schema } = require("mongoose");

const productSchema = new Schema({
    name: { type: String, required: true },
    image: { type: String, required: true },
    description: { type: String, required: true },
    category: { type: String, required: true },
    price: { type: Number, required: true },
    sku: { type: String, required: true },
    size: { type: String, required: true },
    stock: { type: Number, required: true },
}, { timestamps: true });

productSchema.index({ name: "text", description: "text" });

module.exports = model("product", productSchema);
```

routes/CartRoutes.js

```
const CartController = require("../controllers/CartController");
const userAuth = require("../middlewares/UserAuth");

const router = require("express").Router();

// add product to the user cart
router.post("/add", userAuth, CartController.addToCart);

// get the user cart
router.get("/", CartController.getCart);

// add, update and delete product , product _quantity from cart
router.post("/", userAuth, CartController.storeToCart);

module.exports = router;
```

routes/OrderRoutes.js

```
const router = require("express").Router();
const OrderController = require("../controllers/OrderController");
const userAuth = require("../middlewares/UserAuth");

// new order
router.post("/", userAuth, OrderController.newOrder);

//get the order by id
router.get("/:order_id", OrderController.getOrderDetails);

// save order details including delivery
router.post("/:order_id", userAuth, OrderController.saveOrderDetails);

module.exports = router;
```

routes/PaymentRoutes.js

```
const router = require("express").Router();
const PaymentController = require("../controllers/PaymentController");
const userAuth = require("../middlewares/UserAuth");

// receive order comple notification from gateways
router.post("/notify", PaymentController.paymentOrderNotification);

// complete cod order
router.post("/pay/cod", userAuth, PaymentController.codPayment);

module.exports = router;
```

routes/ProductRoutes.js

```
const router = require("express").Router();
const ProductController = require("../controllers/ProductController");

// get all products
router.get("/", ProductController.listProducts);

// save product
router.post("/", ProductController.storeProduct);

// get product based on id
router.get("/:pid", ProductController.findProduct);

// search product
router.get("/search/:text", ProductController.searchProducts);

// update product
router.patch("/:pid", ProductController.updateProduct);

// delete product
router.delete("/:pid", ProductController.deleteProduct);

module.exports = router;
```

util/Auth.js

```
const util = require("util");
var request = require("request");
const { AUTH_GATEWAY } = require("../config");

// contact auth service through esb to validate the jwt token and get the user
const getAuthUserFromBearerToken = async (bearer_token) => {
  var token = String(bearer_token).slice(7);
  const options = {
    url: AUTH_GATEWAY,
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    maxRedirects: 20,
    body: JSON.stringify({
      token: `${token}`,
    }),
  };

  const requestPromise = util.promisify(request);
  try {
    const response = await requestPromise(options);
    // console.log(response.body);
    let jsonBody = response.body ? JSON.parse(response.body) : {};
    if (response.statusCode == 200 && jsonBody._id) return jsonBody;
    else return null;
  } catch (error) {
    console.error(error);
  }
};

module.exports = { getAuthUserFromBearerToken };
```

util/MailService.js

```
var nodemailer = require("nodemailer");
var { MAIL_USER, MAIL_PASSWORD } = require("../config");
var fs = require("fs");

// send a mail using gmail
exports.sendMail = (mailOptions) => {
    var transporter = nodemailer.createTransport({
        service: "gmail",
        auth: {
            user: MAIL_USER,
            pass: MAIL_PASSWORD,
        },
    });

    transporter.sendMail(mailOptions, function(error, info) {
        if (error) {
            console.log(error);
        } else {
            console.log("Email sent: " + info.response);
        }
    });
};

// read a .html file and get its data
exports.readHTMLFile = async(path) => {
    try {
        var data = fs.readFileSync(path, { encoding: "utf-8" });
        return data;
    } catch (error) {
        console.error(error);
        return null;
    }
};
```

util/Payment.js

```
const { MAIL_USER } = require("../config");
const { sendMail, readHTMLFile } = require("./MailService");
var handlebars = require("handlebars");
const { sendSms } = require("./SmsService");

// send email and sms notifing payment successfully
exports.notifyPaymentSuccessfull = async (order) => {
  // send email
  var html = await readHTMLFile("./templates/mail/OrderPlaced.html");
  // get the email template
  var template = handlebars.compile(html);

  var replacements = {
    orderID: order._id,
    buyer_name: order.buyer_name,
    delivery_address: order.delivery_address,
    payment_type: order.payment_type,
    payment_value: order.payment_value,
    products: JSON.stringify(order.products),
  };
  var htmlToSend = template(replacements);

  var mailOptions = {
    from: MAIL_USER,
    to: order.buyer_email,
    subject: `Order has been placed - ${order._id}`,
    html: htmlToSend,
  };
  sendMail(mailOptions);

  // send sms
  var smsOptions = {
    to: order.buyer_phone,
    body: `Order has been placed Order ID - ${order._id} with a payment value of
Rs ${order.payment_value}. You can track your order through our website http://lo
calhost:3000/track-order. Thank you for shopping with Xmart shopping`,
  };
  // sendSms(smsOptions);
};

// send email and sms notifing payment successfully
exports.notifyPaymentFailed = async (order) => {
```

```
  var message = `Order has not been placed - ${order._id}, There were errors whil
e placing your order please contact our customer service for assistance`;

  var mailOptions = {
    from: MAIL_USER,
    to: order.buyer_email,
    subject: `Order has not been placed - ${order._id}`,
    text: message,
  };
  sendMail(mailOptions);

  // send sms
  var smsOptions = {
    to: order.buyer_phone,
    body: message,
  };
  // sendSms(smsOptions);
};
```

util/SmsService.js

```
const {
    TWILIO_ACCOUNT_SID,
    TWILIO_AUTH_TOKEN,
    TWILIO_NUMBER,
} = require("../config");

const client = require("twilio")(TWILIO_ACCOUNT_SID, TWILIO_AUTH_TOKEN);

// send sms using twilio
exports.sendSms = (smsOptions) => {
    try {
        client.messages
            .create({
                body: smsOptions.body,
                from: TWILIO_NUMBER,
                to: `+94${smsOptions.to}`,
            })
            .then((message) => console.log(message));
    } catch (error) {
        console.log(error);
    }
};
```

validators/OrderValidator.js

```
const Product = require("../model/Product");

// validate the request and check if there is user_id, buyer_email, buyer_name, b
uyer_phone, delivery_type and address
exports.ValidateOrderDetails = async (req, res) => {
  var mailformat = /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-
9-]+)*$/;
  var order = req.body;
  var err_message = "";

  if (!order) err_message = "Invalid order";
  else if (order.user_id !== req.user._id)
    err_message = "This order does not belongs to you";
  else if (order.order_status !== "pending")
    err_message =
      "User cannot modify order details at this stage please contact our customer
 service";
  else if (!order.buyer_name || order.buyer_name.length == 0)
    err_message = "Buyer name is required";
  else if (!order.buyer_email || order.buyer_email.length == 0)
    err_message = "Buyer email is required";
  else if (!order.buyer_email.match(mailformat))
    err_message = "Email format is invalid";
  else if (!order.buyer_phone) err_message = "Buyer contact number is required";
  else if (isNaN(order.buyer_phone))
    err_message = "Mobile number contains invalid characters";
  else if (order.buyer_phone.toString().length != 9)
    err_message =
      "Mobile number must only have 9 characters without the initial 0";
  else if (!order.delivery_type) err_message = "Select delivery type";
  else if (
    !(order.delivery_type == "pickup" || order.delivery_type == "delivery")
  )
    err_message = "Selected delivery type is not valid";
  else if (order.delivery_type == "delivery" && !order.delivery_address)
    err_message = "Enter delivery address";
  else if (order.delivery_type == "pickup") order.delivery_address = "";

  if (err_message.length > 0)
    return res.status(422).json({ message: err_message });
  else return order;
};
```

```
// validate order products
// validate the request and check if there is product_id, product_quantity,
exports.ValidateOrderProducts = async (req, res) => {
  var order = req.body;
  var err_message = "";

  try {
    // validate req body
    if (!order) err_message = "Invalid order details";
    else if (
      !order.products ||
      !Array.isArray(order.products) ||
      order.products.length == 0
    )
      err_message = "Invalid order products";
    else {
      var productErrors = [];
      var totalValue = 0;
      var validateOrderProducts = [];

      // validate products individually
      for (let index = 0; index < order.products.length; index++) {
        try {
          var orderProduct = order.products[index];
          // check if order product id exist in DB
          var product = await Product.findById(orderProduct.id);
          if (!product)
            productErrors.push(`${orderProduct.id} Invalid product details`);
          else if (!orderProduct.quantity || isNaN(orderProduct.quantity))
            productErrors.push(`${orderProduct.id} Select order quantity`);
          else if (orderProduct.quantity > product.stock)
            productErrors.push(
              `${orderProduct.id} Product does not have enough stock`
            );
          else {
            totalValue += product.price * orderProduct.quantity;
            // to remove unnessory data sent from the frontend( ex description, i
mage)
            validateOrderProducts.push({
              id: orderProduct.id,
              quantity: orderProduct.quantity,
            });
          }
        } catch (error) {
          productErrors.push(`${orderProduct.id} Invalid product details`);
```

```
        }
      }

      if (productErrors.length > 0)
        return res
          .status(422)
          .json({ message: "There were errors in products", productErrors });
      else {
        // set order payment and validated order products
        order.payment_value = totalValue;
        order.products = validateOrderProducts;
      }
    }
    if (err_message.length > 0)
      return res.status(422).json({ message: err_message });
    else return order;
  } catch (error) {
    return res.status(422).json({ message: "There were errors in products" });
  }
};
```

.env

```
APP_KEY = oOTypsdWE4fzzwe
APP_DB = mongodb://127.0.0.1:27017/ds_buyer_service
APP_PORT = 5001
AUTH_GATEWAY = http://192.168.61.1:8280/auth/validateToken
MAIL_USER = ronicyteam@gmail.com
MAIL_PASSWORD = ronicy123
TWILIO_ACCOUNT_SID = AC58e2c8a739370106a9c458b8b56e53bb
TWILIO_AUTH_TOKEN = 852295c50ae6fb3dbbf2a6d9990964c8s
TWILIO_NUMBER = +17747664321
ORDER_SECRET = ssdsd3324v343535
PAYMENT_SECRET = 445433ddss2WFvTY158
```

6.1.4. Credit card payment gateway service

index.js

```
const bodyParser = require("body-parser");
const cors = require("cors");
const { connect } = require("mongoose");
const express = require("express");
const { DB, PORT } = require("./config");

const app = express();

app.use(bodyParser.json());
// app.use(cors());

app.use("/api/gateway/card", require("./routes/GatewayRoutes"));

const startApp = async() => {
    try {
        await connect(DB, {
            useFindAndModify: true,
            useUnifiedTopology: true,
            useNewUrlParser: true,
        });

        console.log("Connected to database");

        await app.listen(PORT, () =>
            console.log(`App is listening on port ${PORT}`)
        );
    } catch (error) {
        console.error(error);
        startApp();
    }
};

startApp();
```

package.json

```
{
    "name": "card_service",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1",
        "start": "nodemon index.js"
    },
    "author": "Manuka Yasas",
    "license": "ISC",
    "dependencies": {
        "axios": "^0.21.1",
        "body-parser": "^1.19.0",
        "cors": "^2.8.5",
        "dotenv": "^8.2.0",
        "express": "^4.17.1",
        "md5": "^2.3.0",
        "mongoose": "^5.12.7",
        "nodemon": "^2.0.7"
    }
}
```

Config/index.js

```
require("dotenv").config();

module.exports = {
  DB: process.env.APP_DB,
  PORT: process.env.APP_PORT,
  PAYMENT_SECRET: process.env.PAYMENT_SECRET,
  ORDER_SECRET: process.env.ORDER_SECRET,
  PAYMENT_NOTIFY_URL: process.env.PAYMENT_NOTIFY_URL,
};
```

Controller/GatewayController.js

```
var md5 = require("md5");
const { default: axios } = require("axios");
const {
  ORDER_SECRET,
  PAYMENT_SECRET,
  PAYMENT_NOTIFY_URL,
} = require("../config");
const Card = require("../models/Card");
const {
  validatePaymentRequest,
  matchCardDetails,
} = require("../util/GatewayValidations");

exports.makePayment = async (req, res) => {
  try {
    // validate request body
    var validatedDetails = validatePaymentRequest(req, res);
    // hash paras and validate if the payment details are valid
    var hash_order_code_valid = md5(
      `${ORDER_SECRET}${req.body.order_id}${req.body.transfer_amount}`
    );
    //check if payment details and hash matches
    if (hash_order_code_valid == req.body.hash_order_code) {
      // get card details
      var card = await Card.findOne({ card_no: req.body.card_no });

      // match credit card details
      var transferInfo = matchCardDetails(card, validatedDetails, res);
      // complete transaction
      card.balance -= transferInfo.transfer_amount;
      var result = await card.save();

      // save failed
      if (result && result.error) return res.status(400).json(result.error);
      //payment complted therefour notify main server payment complted
      notifyServer(transferInfo.order_id, transferInfo.transfer_amount);
      return res.status(200).json({
        payment: "Payment was successfull",
        status: 1,
      });
    }
    // if hashing fails
```

```
    return res
      .status(400)
      .json({ message: "Invalid payment details, refresh and try again" });
  } catch (error) {
    console.log(error);
    return res.status(400).json({ message: "Invalid card number" });
  }
};

const notifyServer = (orderID, transfer_amount) => {
  // hash the payment data to validate at the buyer server(main)
  var hash_pay_code = md5(`${PAYMENT_SECRET}${orderID}${transfer_amount}`);
  axios
    .post(PAYMENT_NOTIFY_URL, {
      order_id: orderID,
      payment_type: "card",
      hash_pay_code,
    })
    .then((res) => console.log(res))
    .catch((err) => console.log(err));
};
```

Model/Card.js

```
const { model, Schema } = require("mongoose");

const cardSchema = new Schema(
  {
    card_no: { type: Number, required: true },
    card_cvc: { type: Number, required: true },
    card_holder_name: { type: String, required: true },
    balance: { type: Number, required: true },
  },
  { timestamps: true }
);

module.exports = model("card", cardSchema);
```

Routes/GatewayRoutes.js

```
const GatewayController = require("../controllers/GatewayController");

const router = require("express").Router();

router.post("/", GatewayController.makePayment);

module.exports = router;
```

util/GatewayValidations.js

```
// validate payment request boduy
exports.validatePaymentRequest = (req, res) => {
  var message = "";
  var details = req.body;

  if (!details && !details.order_id) message = "Invalid payment details";
  else if (!details.hash_order_code) message = "Invalid payment hash";
  else if (!details.card_no) message = "Enter card number";
  else if (isNaN(details.card_no)) message = "Invalid card number";
  else if (!details.card_cvc) message = "Enter card CVC";
  else if (isNaN(details.card_cvc)) message = "Invalid card CVC";
  else if (!details.card_holder_name) message = "Enter card holder name";
  else if (!details.transfer_amount) message = "Enter transfer amount";
  else if (isNaN(details.transfer_amount)) message = "Invalid transfer amount";

  if (message.length > 0) return res.status(422).json({ message });
  else return details;
};

// validate credit card deatils with provided details
exports.matchCardDetails = (card, req_details, res) => {
  var err_message = "";
  if (!card)
    err_message =
      "There is no credit card associated with the card no provided";
  else if (card.card_cvc != req_details.card_cvc)
    err_message = "Card CVC number did not match";
  else if (card.card_holder_name != req_details.card_holder_name)
    err_message = "Card holder name did not match";
  else if (card.balance < req_details.transfer_amount)
    err_message = "Card balance is not suffient";
```

```
  if (err_message.length > 0)
    return res.status(422).json({ message: err_message });
  else return req_details;
};
```

.env

```
APP_DB = mongodb://127.0.0.1:27017/ds_buyer_service
APP_PORT = 5003
ORDER_SECRET = ssdsd3324v343535
PAYMENT_SECRET = 445433ddss2WFvTY158
PAYMENT_NOTIFY_URL = http://192.168.61.1:8280/payment/notify/server
```

6.1.5. Mobile payment gateway service

Index.js

```
const bodyParser = require("body-parser");
const cors = require("cors");
const { connect } = require("mongoose");
const express = require("express");
const { DB, PORT } = require("./config");

const app = express();

app.use(bodyParser.json());
// app.use(cors());

app.use("/api/gateway/mobile", require("./routes/GatewayRoutes"));

const startApp = async() => {
    try {
        await connect(DB, {
            useFindAndModify: true,
            useUnifiedTopology: true,
            useNewUrlParser: true,
        });

        console.log("Connected to database");

        await app.listen(PORT, () =>
            console.log(`App is listening on port ${PORT}`)
        );
    } catch (error) {
        console.error(error);
        startApp();
    }
};

startApp();
```

Package.json

```
{
    "name": "mobile_payment_gateway_service",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1",
        "start": "nodemon index.js"
    },
    "author": "Manuka Yasas",
    "license": "ISC",
    "dependencies": {
        "axios": "^0.21.1",
        "body-parser": "^1.19.0",
        "cors": "^2.8.5",
        "dotenv": "^8.2.0",
        "express": "^4.17.1",
        "md5": "^2.3.0",
        "mongoose": "^5.12.7",
        "nodemon": "^2.0.7",
        "twilio": "^3.61.0"
    }
}
```

Config/index.js

```
require("dotenv").config();

module.exports = {
  DB: process.env.APP_DB,
  PORT: process.env.APP_PORT,
  TWILIO_ACCOUNT_SID: process.env.TWILIO_ACCOUNT_SID,
  TWILIO_AUTH_TOKEN: process.env.TWILIO_AUTH_TOKEN,
  TWILIO_NUMBER: process.env.TWILIO_NUMBER,
  PAYMENT_SECRET: process.env.PAYMENT_SECRET,
  ORDER_SECRET: process.env.ORDER_SECRET,
  PAYMENT_NOTIFY_URL: process.env.PAYMENT_NOTIFY_URL,
};
```

Controllers/GatewayController.js

```
var md5 = require("md5");
const { default: axios } = require("axios");
const {
  ORDER_SECRET,
  PAYMENT_SECRET,
  PAYMENT_NOTIFY_URL,
} = require("../config");
const Mobile = require("../models/Mobile");
const {
  validatePaymentRequest,
  validateMobileNumber,
} = require("../util/GatewayValidations");
const { sendSms } = require("../util/SmsService");

exports.makePayment = async (req, res) => {
  try {
    var validatedPayment = validatePaymentRequest(req, res);

    // hash paras and validate if the payment details are valid
    var hash_order_code_valid = md5(
      `${ORDER_SECRET}${req.body.order_id}${req.body.transfer_amount}`
    );
    //check if payment details and hash matches
    if (hash_order_code_valid == req.body.hash_order_code) {
      // get the mobile object from DB to validate
      var mobile = await Mobile.findOne({
        mobile_no: validatedPayment.mobile_no,
      });

      // match mobile data with request data
      var err_message = "";
      if (!mobile) err_message = "invalid mobile number";
      else if (mobile.pin != req.body.pin)
        err_message = "pin number did not match";
      // iff data is not matched
      if (err_message.length > 0)
        return res.status(400).json({ message: err_message });

      // if data is matched
      // complete transaction
      mobile.balance += Number.parseFloat(req.body.transfer_amount);
      var result = await mobile.save();
```

```
      // save failed
      if (result && result.error) return res.status(400).json(result.error);

      //payment complted therefour notify main server payment complted
      notifyServer(req.body.order_id, req.body.transfer_amount);
      return res.status(200).json({
        payment: "Payment was successfull",
        status: 1,
      });
    }
    // if hashing fails
    return res
      .status(400)
      .json({ message: "Invalid payment details, refresh and try again" });
  } catch (error) {
    console.log(error);
    return res.status(400).json({
      errors: { message: "This mobile number does not accept payments" },
    });
  }
};

exports.requestPin = async (req, res) => {
  try {
    var phone_number = req.params.number;
    console.log(phone_number);

    // validate mobile number
    var error = validateMobileNumber(phone_number);
    if (error.length > 0) return res.status(400).json({ mesaage: error });

    // send pin number
    // generate random 4 digit pin
    var pin = Math.floor(1000 + Math.random() * 9000);
    console.log(pin);

    var mobile = await Mobile.findOne({ mobile_no: phone_number });
    if (!mobile) {
      mobile = new Mobile({
        mobile_no: phone_number,
        pin,
      });
    } else mobile.pin = pin;
```

```
    var result = await mobile.save();

    // save failed
    if (result && result.error)
      return res
        .status(400)
        .json({ message: "Unexpected error please try again" });

    // send the pin via sms
    var smsOptions = {
      to: phone_number,
      body: `Your XMart Payment Pin is - ${pin}`,
    };

    await sendSms(smsOptions);

    return res.status(200).json({
      message: "Pin was sent successfull",
    });
  } catch (error) {
    console.log(error);
    return res
      .status(400)
      .json({ errors: { message: "Invalid mobile number" } });
  }
};

// notify server that the payment was completed
const notifyServer = (orderID, transfer_amount) => {
  // hash the payment data to validate at the buyer server
  var hash_pay_code = md5(`${PAYMENT_SECRET}${orderID}${transfer_amount}`);
  console.log(hash_pay_code);
  axios
    .post(PAYMENT_NOTIFY_URL, {
      order_id: orderID,
      payment_type: "mobile",
      hash_pay_code,
    })
    .then((res) => console.log(res))
    .catch((err) => console.log(err));
};
```

Models/Mobile.js

```
const { model, Schema } = require("mongoose");

const mobileSchema = new Schema({
    mobile_no: { type: Number, required: true },
    pin: { type: Number, required: true },
    balance: { type: Number, default: 0 },
}, { timestamps: true });

module.exports = model("mobile", mobileSchema);
```

Routes/GatewayRoutes.js

```
const GatewayController = require("../controllers/GatewayController");

const router = require("express").Router();

router.post("/", GatewayController.makePayment);

router.post("/request-pin/:number", GatewayController.requestPin);

module.exports = router;
```

util/GatewayValidations.js

```
exports.validatePaymentRequest = (req, res) => {
  var message = "";
  var details = req.body;

  if (!details) message = "Invalid payment details";
  else if (!details.order_id) message = "Invalid order details";
  else if (!details.hash_order_code) message = "Invalid payment hash";
  else if (this.validateMobileNumber(details.mobile_no).length > 0)
    message = "Enter a valid 9 digit mobile number, without the initial 0";
  else if (!details.pin) message = "Enter pin number";
  else if (isNaN(details.pin))
    message = "Pin number contains invalid characters";
  else if (details.pin.length != 4) message = "Pin number should have 4 digits";
  else if (!details.transfer_amount) message = "Enter transfer amount";
  else if (isNaN(details.transfer_amount)) message = "Invalid transfer amount";

  if (message.length > 0) return res.status(422).json({ message });
  else return details;
};

exports.validateMobileNumber = (number) => {
  if (!number || isNaN(number) || number.length != 9)
    return "Enter a valid 9 digit mobile number, without the initial 0";
  return "";
};
```

util/SmsService.js

```
const {
  TWILIO_ACCOUNT_SID,
  TWILIO_AUTH_TOKEN,
  TWILIO_NUMBER,
} = require("../config");

const client = require("twilio")(TWILIO_ACCOUNT_SID, TWILIO_AUTH_TOKEN);

exports.sendSms = async (smsOptions) => {
  try {
    await client.messages
      .create({
        body: smsOptions.body,
        from: TWILIO_NUMBER,
        to: `+94${smsOptions.to}`,
      })
      .then((message) => console.log(message));
  } catch (error) {
    console.log(error);
  }
};
```

.env

```
APP_DB = mongodb://127.0.0.1:27017/ds_buyer_service
APP_PORT = 5004

TWILIO_ACCOUNT_SID = AC58e2c8a739370106a9c458b8b56e53bb
TWILIO_AUTH_TOKEN = 852295c50ae6fb3dbbf2a6d9990964c8s
TWILIO_NUMBER = +17747664321

ORDER_SECRET = ssdsd3324v343535
PAYMENT_SECRET = 445433ddss2WFvTY158
PAYMENT_NOTIFY_URL = http://192.168.61.1:8280/payment/notify/server
```

6.1.6. Delivery service

Server.js

```
require("dotenv").config({ path: "./config.env" });
const express = require("express");
const connectDB = require("./config/db");
const passport = require("passport");
const cors = require("cors");

// Routes
const deliveryRoutes = require("./routes/api/delivery");
const AuthRoutes = require("./routes/api/AuthRoutes");
const { UserAuth } = require("./middlewares/UserAuth");

const app = express();
//Bodyparser middleware
app.use(express.json());
app.use(cors());

// Connect to the DB
connectDB();

app.use(passport.initialize());
require("./middlewares/Passport")(passport);

// auth routes
app.use("/api/auth", AuthRoutes);

// delivery routes
app.use("/api/admin/delivery", UserAuth, deliveryRoutes);

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log("Server run at port " + PORT));

//
```

Package.json

```
{
  "name": "delivery_api",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon server",
    "start": "nodemon server.js"
  },
  "keywords": [],
  "author": "Harini Gunawardana",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.0.1",
    "cors": "^2.8.5",
    "dotenv": "^9.0.2",
    "express": "^4.17.1",
    "jsonwebtoken": "^8.5.1",
    "mongoose": "^5.12.9",
    "nodemon": "^2.0.7",
    "passport": "^0.4.1",
    "passport-jwt": "^4.0.0"
  }
}
```

config/db.js

```
const mongoose = require("mongoose");

const connectDB = async () => {
  await mongoose.connect(process.env.MONGO_URI, {
    useNewUrlParser: true,
    useCreateIndex: true,
    useUnifiedTopology: true,
    useFindAndModify: true,
  });

  console.log(" MongoDB Connected");
};

module.exports = connectDB;
```

controllers/AuthController.js

```
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const User = require("../models/User");
const {
  validateUserInfo,
  validateLoginCredentials,
} = require("../util/AuthValidator");

// register user
exports.Register = async (req, res) => {
  try {
    var validatedUserInfo = await validateUserInfo(req, res);

    // save user
    const hashedPassword = await bcrypt.hash(validatedUserInfo.password, 12);

    const newUser = new User({
      ...validatedUserInfo,
      role: "user",
      password: hashedPassword,
    });

    var result = await newUser.save();

    return res.status(201).json({
      message: "User was registered",
      success: true,
      user: result._doc,
    });
  } catch (error) {
    console.error(error);
    return res.status(400).json({
      message: "User was not registered",
      success: false,
    });
  }
};

// authenticate user
exports.Authenticate = async (req, res) => {
  try {
    let validatedLoginCredentials = validateLoginCredentials(req, res);
    // fetch user with that email
```

```
var user = await User.findOne({
  email: validatedLoginCredentials.email,
});
// response error if user was not found
if (!user) {
  return res.status(400).json({
    message: "Unable match user credentials",
    success: false,
  });
}

// match password
var isMatch = await bcrypt.compare(
  validatedLoginCredentials.password,
  user.password
);
// return error if passwods dopes nbot match
if (!isMatch)
  return res.status(400).json({
    message: "Password does not match",
    success: false,
  });
else {
  // login user with jwt token
  // jwt - create token
  let token = jwt.sign(
    {
      user_id: user._id,
      role: user.role,
      name: user.name,
    },
    process.env.AUTH_KEY,
    { expiresIn: "2 days" }
  );

  let result = {
    user_id: user._id,
    role: user.role,
    name: user.name,
    token,
    expiresIn: 168,
  };

  return res.status(200).json({
    ...result,
```

```
        message: "Success user login",
        success: true,
      });
    }
  } catch (error) {
    console.error(error);
    return res.status(400).json({
      message: "Failed to authenticate",
      success: false,
    });
  }
};

exports.Logout = async (req, res) => {
  req.logOut();
  return res.status(200).json({ message: "You have successfully logout" });
};

exports.Profile = async (req, res) => res.send(req.user);

exports.ValidateToken = async (req, res) => {
  try {
    // validate token in request
    if (!req.body.token)
      return res.status(422).json({ message: "JWT token is required" });
    // validate JWT token
    var decodedToken = jwt.verify(req.body.token, process.env.AUTH_KEY);
    // fetch user token token user id
    var user = await User.findById(decodedToken.user_id);
    return res.status(200).json(user);
  } catch (error) {
    return res.status(400).json({ message: "JWT token is not valid" });
  }
};
```

controllers/DeliveryController.js

```
const Delivery = require("../models/Delivery");

exports.newDelivery = async (req, res) => {
  const newDelivery = new Delivery(req.body);
  try {
    const delivery = await newDelivery.save();
    if (!delivery) throw Error("Something went wrong");
    res.status(200).json(delivery);
  } catch (error) {
    console.log(error);
    res.status(400).json({ message: error });
  }
};

exports.updateDelivery = async (req, res) => {
  try {
    const delivery = await Delivery.findByIdAndUpdate(req.params.id, req.body);
    if (!delivery) throw Error("Error when updating");

    res.status(200).json({ success: true });
  } catch (error) {
    console.log(error);
    res.status(400).json({ message: error });
  }
};

exports.deleteDelivery = async (req, res) => {
  try {
    const delivery = await Delivery.findByIdAndDelete(req.params.id);
    if (!delivery) throw Error("No Delivery found");

    res.status(200).json({ success: true });
  } catch (error) {
    res.status(400).json({ message: error });
  }
};

exports.findDelivery = async (req, res) => {
  try {
    const delivery = await Delivery.findById(req.params.id);
    if (!delivery) throw Error("No such item");
    res.status(200).json(delivery);
  } catch (error) {
```

```
    res.status(400).json({ message: error });
  }
};

exports.listAllDelivery = async (req, res) => {
  try {
    const delivery = await Delivery.find();
    if (!delivery) throw Error("No items");
    res.status(200).json(delivery);
  } catch (error) {
    res.status(400).json({ message: error });
  }
};
```

middlewares/Passport.js

```
const { Strategy, ExtractJwt } = require("passport-jwt");
const User = require("../models/User");

const opts = {
  jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
  secretOrKey: process.env.AUTH_KEY,
};

module.exports = (passport) => {
  passport.use(
    new Strategy(opts, async (payload, done) => {
      await User.findById(payload.user_id)
        .then((user) => (user ? done(null, user) : done(null, false)))
        .catch((error) => done(null, false));
    })
  );
};
```

middlewares/UserAuth,js

```
const passport = require("passport");

// passport middleware
exports.UserAuth = passport.authenticate("jwt", { session: false });
```

models/Delivery.js

```
const mongoose = require("mongoose");

const DeliverySchema = new mongoose.Schema({
  buyername: {
    type: String,
    required: [true, "Please provide buyer name"], // validations
  },

  email: {
    type: String,
    required: [true, "Please provide an email"], // validations
    match: [
      /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-
9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-
Z]{2,}))$/,
      "Please provide a valid email",
    ],
  },

  telephone: {
    type: String,
    required: [true, "Please provide a telephone number"], //validations
  },

  address: {
    type: String,
    required: [true, "Please provide the delivery address"], // validations
  },

  paymentType: {
    type: String,
    required: [true, "Please provide the payment type"], // validations
  },

  paymentValue: {
    type: Number,
    required: [true, "Please provide the payment value"], // validations
  },

  date: {
    type: Date,
    default: Date.now,
  },
```

```
});

const Delivery = mongoose.model("Delivery", DeliverySchema);
module.exports = Delivery;
```

models/User.js

```
const { model, Schema } = require("mongoose");

const userSchema = new Schema(
  {
    name: { type: String, required: true },
    email: { type: String, required: true },
    phone: { type: Number, required: true },
    password: { type: String, required: true },
    role: { type: String, default: "buyer", enum: ["driver", "user"] },
  },
  { timestamps: true }
);

const User = model("user", userSchema);
module.exports = User;
```

routers/api/AuthRoutes.js

```
const router = require("express").Router();
const { UserAuth } = require("../../middlewares/UserAuth");
const AuthController = require("../../controllers/AuthController");

// register user
router.post("/register", AuthController.Register);

// login user
router.post("/login", AuthController.Authenticate);

router.get("/profile", UserAuth, AuthController.Profile);

router.post("/validateToken", AuthController.ValidateToken);

module.exports = router;
```

routers/api/delivery.js

```
const express = require("express");
const router = express.Router();
const DeliveryController = require("../../controllers/DeliveryController");

//Create delivery POST
router.post("/", DeliveryController.newDelivery);

// UPDATE delivery  api/delete/:id
router.patch("/:id", DeliveryController.updateDelivery);

// DELETE delivery  api/delete/:id
router.delete("/:id", DeliveryController.deleteDelivery);

// GET all deliveries
router.get("/", DeliveryController.listAllDelivery);

// GET one delivery
router.get("/:id", DeliveryController.findDelivery);

module.exports = router;
```

util/AuthValidator.js

```
const User = require("../models/User");

// validations
const validateUserInfo = async (req, res) => {
  var mailformat =
    /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/;
  var message = "";
  var user = req.body;

  //  validate request body

  if (!user) message = "Invalid user details";
  else if (!user.name || user.name.length == 0)
    message = "Enter a valid username";
  else if (!user.email || user.email.length == 0)
    message = "Enter a valid email";
  else if (!user.email.match(mailformat)) message = "Email format is invalid";
  else if (!user.phone) message = "Enter user contact number";
  else if (isNaN(user.phone))
    message = "Mobile number contains invalid characters";
  else if (user.phone.toString().length != 9)
    message = "Enter a valid 9 digit mobile number";
  else if (!user.password || user.name.password < 8)
    message = "Password must have at least 8 characters";
  else {
    // validate email
    var isEmailValid = await validateEmail(user.email);
    if (!isEmailValid)
      message = "There is a registered user with the email provided.";
  }
  // handle req errors
  if (message.length > 0)
    return res.status(422).json({
      message,
      success: false,
    });
  else return user;
};

// check if the provided email is saved on db
const validateEmail = async (email) => {
  let user = await User.findOne({ email });
  return user ? false : true;
```

```
};

const validateLoginCredentials = (req, res) => {
  var message = "";
  var credentials = req.body;
  // validate req body
  if (!credentials) message = "Invalid login credentials";
  else if (!credentials.email || credentials.email.length == 0)
    message = "Enter a valid user email";
  else if (!credentials.password || credentials.password.length == 0)
    message = "Enter a valid password";
  // handle req errors
  if (message.length > 0)
    return res.status(422).json({
      message,
    });
  else return credentials;
};

module.exports = {
  validateUserInfo,
  validateLoginCredentials,
};
```

.config.env

AUTH_KEY = qwqwqwqwqw323

PORT = 5010

MONGO_URI = mongodb://127.0.0.1:27017/ds_delivery_service