SCS 3203 - Middleware Architecture Report

Visal Jayathilaka - 19000669
Thisari Gunawardena - 19000537
Venushka Chandrarathna - 19000162
Maleesha Gunarathna - 19000464

Table of Contents

Preface **2**

Introduction **2**

Architecture Diagram **2**

Workflow Diagrams **3**

ESB-Services Path Mapping **4**

Interface Definitions **5**

1. Authentication Service 5
2. Customer Service 6
3. Delivery Service 6
4. Inventory Service 7
5. Notification Service 9
6. Payment Service 10

Appendix **12**

main.ts -auth 12
auth.controller.ts 15
token.services.ts 17
Main.ts -customer 18
Customer.controller.ts 21
Main.ts - delivery 23
Delivery.controller.ts 27
Main.ts - inventory 27
Product.controller.ts 34
Main.ts - notification 37
Notification.controller.ts 40
Main.ts - payment 41
Payment.contoller.ts 45

Preface

The purpose of this report is to deliver the results of the REST API Assignment of SCS 3203 Middleware Architecture which was to develop a collaborative shopping platform.

Introduction

According to the requirements gathered there are two main actors in the system, namely, buyer and seller. The main functionalities identified are,

1. A seller may add, update and delete items
2. A buyer may search and buy items uploaded by the sellers
3. A buyer may buy one or more items
4. A buyer may request to deliver the items bought through the platform
5. A buyer may pay for the items bought via credit card or a mobile service provider's service to credit the mobile bill and receive confirmation via email and SMS

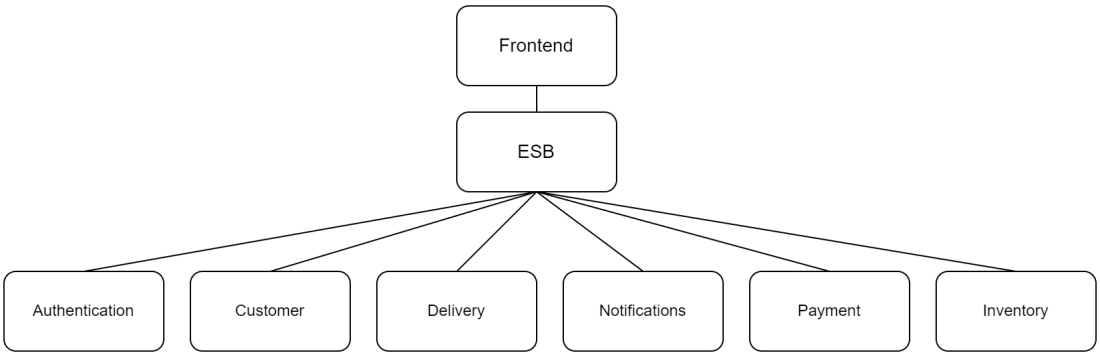
For the payment gateway, delivery service, SMS service, email service and mobile payment dummy services were implemented.

Six main services were identified from the given requirements.

1. **Authorization Service:** Responsible for providing authentication and authorization, login and signup functionalities
2. **Customer Service:** Responsible for handling buyers purchasing products
3. **Delivery Service:** Responsible for providing delivery price when the delivery address is given
4. **Inventory Service:** Responsible for handling adding, updating and removing products from the platform
5. **Notification Service:** Responsible for sending mobile and email confirmations
6. **Payment Service:** Responsible for handling card and mobile payments

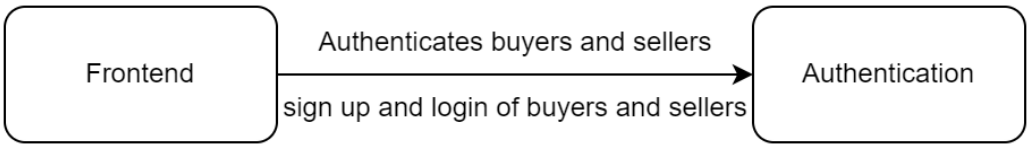
Architecture Diagram

The below diagram shows the high-level architecture of the system developed. The front end is connected to the backend via the WSO2 Enterprise Integrator.

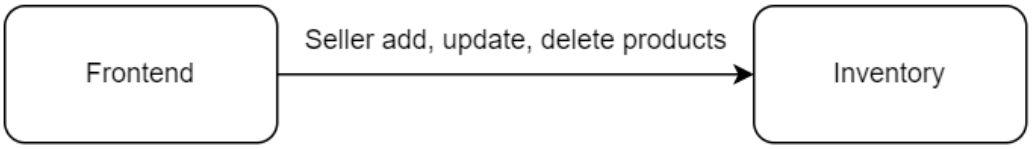


Workflow Diagrams

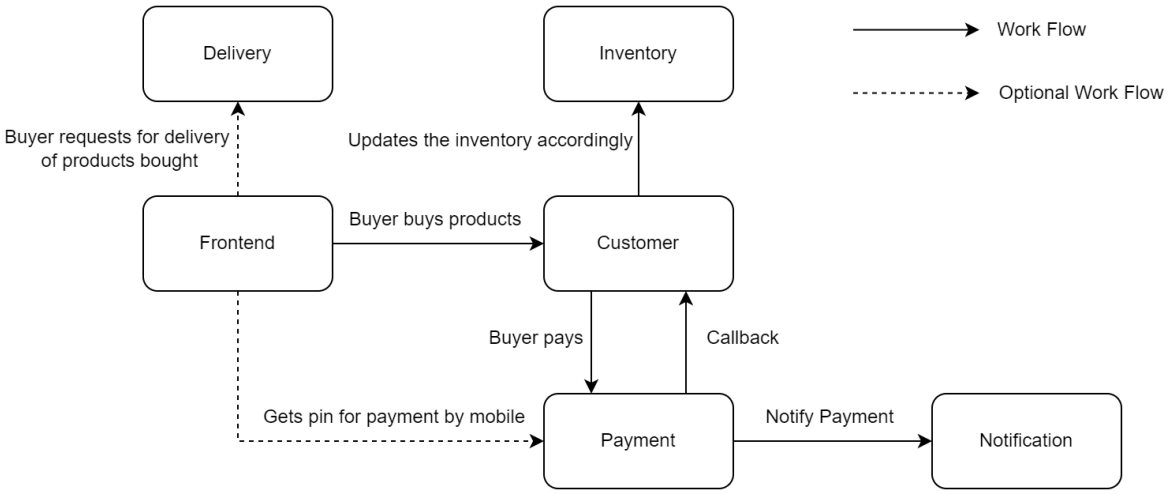
The authentication service authenticates buyers and sellers. It also provides login and signup functionality for the users.



The inventory service updates inventory service accordingly when a seller adds/ updates/ deletes items.



The customer service is called when a buyer buys products. If the buyer requests the delivery service offered in the platform, then the delivery service is called to get the delivery fee and adds it to the total price. It then calls the payment service. If the customer picks the card payment option, then the payment is made accordingly. If the buyer picks the mobile payment option to make the payment, the payment service will send a pin for authenticating the mobile payment. Once the payment is successful, the notification service will send an SMS/ email notification to the buyer. Finally, the inventory service is called to update the inventory accordingly.



ESB-Services Path Mapping

- customer | POST http://localhost:8290/customer/buy -> http://localhost:3888/api/customer/buy
- customer | GET http://localhost:8290/customer/callback/{data}/{token} -> http://localhost:3888/api/customer/callback/:data/:token
- products | DELETE http://localhost:8290/products/{id} -> http://localhost:4111/api/products/:id
- products | GET http://localhost:8290/products -> http://localhost:4111/api/products
- delivery | GET http://localhost:8290/delivery/{address} -> http://localhost:3555/api/delivery/:address
- payment | GET http://localhost:8290/payment/getMobilePin/{mobile} -> http://localhost:3666/api/payment/getMobilePin/:mobile
- products | GET http://localhost:8290/products/{name} -> http://localhost:4111/api/products/:name
- delivery | POST http://localhost:8290/delivery/ -> http://localhost:3555/api/delivery
- products | POST http://localhost:8290/products/ -> http://localhost:4111/api/products/
- notification | POST http://localhost:8290/notification/mail -> http://localhost:4222/api/notification/mail
- notification | POST http://localhost:8290/notification/sms -> http://localhost:4222/api/notification/sms
- payment | POST http://localhost:8290/payment/card -> http://localhost:3666/api/payment/card
- payment | POST http://localhost:8290/payment/mobile -> http://localhost:3666/api/payment/mobile
- products | POST http://localhost:8290/products/buy -> http://localhost:4111/api/products/buy
- products | PUT http://localhost:8290/products/{id} -> http://localhost:4111/api/products/{uri.var.id}
- authentication | GET http://localhost:8290/authentication/ -> http://localhost:3444/api/authentication
- authentication | POST http://localhost:8290/authentication/login -> http://localhost:3444/api/authentication/login
- authentication | POST http://localhost:8290/authentication/signup -> http://localhost:3444/api/authentication/signup

Interface Definitions

1. Authentication Service

- authentication | POST http://localhost:8290/authentication/signup -> http://localhost:3444/api/authentication/signup

```
summary : enter email and password to login to the system
requestBody schema:
          type: object
          properties:
            username: string
            email:string
            password: string
            role: string
  responses:
    UnauthorizedError:
     description: Access token is missing or invalid
    200:
     description: Signed up
```

- authentication | GET http://localhost:8290/authentication/ -> http://localhost:3444/api/authentication

```
summary : Verification of the tokens
requestBody schema:
          type: object
          properties:
            username: string
            email: string
            password: string
            role: string
  responses:
    200:
      description: OK (The token successfully verified)
    UnauthorizedError:
      description: Invalid Token
```

- authentication | GET http://localhost:8290/authentication/ -> http://localhost:3444/api/authentication

```
summary : enter email and password to login to the system
requestBody schema:
          type: object
          properties:
            email: string
            password: string
responses:
    200:
      description: OK (successfully authenticated)
    401:
      description: Incorrect Credentials
    401:
      description: User does not exist
```

2. Customer Service

- customer | GET http://localhost:8290/customer/callback/{data}/{token} ->

  http://localhost:3888/api/customer/callback/:data/:token

```
summary: Callback confirming the details processed sucessfully
parameters:
  - in: path
    name: data
    required: true
    schema:
      type: object
  - in: path
    name: token
    required: true
    schema:
      type: string
        responses:
          '200':
            description: Successfully Completed
              schema:
                Type:string
```

- customer | POST http://localhost:8290/customer/buy ->

  http://localhost:3888/api/customer/buy

```
summary: Sends payment details,delivery details and product details.
requestBody:products,payment details,delivery details
  required: true
  content:
    application/json:
      schema:
 requestBody:
  required: true
   content:
    application/x-www-form-urlencoded:
      schema:
        type: object
        properties:
          products: object
          paymentDetails: object
          deliveryDetails: object
          token: string
          email: string
responses:
  '200':
     description: Sucessfully Completed
       schema:
         type:string
```

3. Delivery Service

- delivery | GET http://localhost:8290/delivery/{address} ->

  http://localhost:3555/api/delivery/:address

```
    summary: request the delivery rate.
    parameters:
        - in: query
          name: address
          required: true
```

```
            Schema:string
      responses:
            '200':
                description: Returns delivery rates
            content:
              application/json:
                schema:
                  type: object
                  data: string
```

- delivery | POST http://localhost:8290/delivery/ ->

  http://localhost:3555/api/delivery

```
summary: request the delivery rate.
requestBody:
  required: true
  content:
    application/x-www-form-urlencoded:
      schema:
        type: object
        properties:
          username: string
          address: string
          products: array
          deliveryPrice: number
  '200':
    description: Delivery Successful
    schema:
      content:
       type:string
```

4. Inventory Service

- products | GET http://localhost:8290/products/{name} ->

  http://localhost:4111/api/products/:name

```
summary: Returns the product with the searched name.
responses:
   '200':
     description: A JSON array with the object with all the details of the product
     content:
       application/json:
         schema:object
           items: string
```

- products | GET http://localhost:8290/products ->

  http://localhost:4111/api/products

```
summary: Returns the list of product list.
      responses:
        '200':
          description: A JSON array with the list of products
          content:
            application/json:
              schema: array
                items: object
```

- products | POST http://localhost:8290/products/ ->

  http://localhost:4111/api/products/

```
paths:
```

```
  /products:
    post:
      summary: add a product.
      requestBody:
        required: true
        content:
          application/x-www-form-urlencoded:
            schema:
              type: object
              properties:
                name:string
                price:string
                amount:string
                image_url:string
      responses:
        '200':
          description: The product has been successfully added
        '500':
          description: Error in adding product
```

- products | PUT http://localhost:8290/products/ -> http://localhost:4111/api/products/:id

```
paths:
  /products/{productId}:
    put:
      summary: Updates a product.
       parameters:
        - in: query
          name: id
          required: true
          schema:
            type: string
      requestBody:
        required: true
        content:
          application/x-www-form-urlencoded:
            schema:
              type: object
              properties:
                name: string
                price: string
                amount: string
                image_url: string
      responses:
        '200':
          description: Product updated
          content:
            application/json:
              schema:
                type: object
                items:
                  type: string
        '404':
          description: Product not found
```

- products | POST http://localhost:8290/products/ -> http://localhost:4111/api/products/buy

```
paths:
```

```
  /products/buy:
    post:
      summary: Updates the amount of a product.
       parameters:
        - in: query
          name: id
          required: true
          schema:
            type: string
      requestBody:
        required: true
        content:
          application/x-www-form-urlencoded:
            schema:
              type: object
              properties:
                amount: number
      responses:
        '200':
          description: The product updated
        '404':
          description: Product not Found
```

- products | DELETE http://localhost:8290/products/{id} -> http://localhost:4111/api/products/:id

```
paths:
  /products/{productId}:
    delete:
      summary: Deletes the  product.
       parameters:
        - in: query
          name: id
          required: true
          schema:
            type: string
      responses:
        '200':
          description: The product amount has been successfully deleted
          content:
            application/json:
              schema:
                type: object
                items:
                  type: string
```

5. Notification Service

- notifications| POST http://localhost:8290/notification/ -> http://localhost:4222/api/notification/mail

```
paths:
  /notification/mail:
    post:
      summary: send the emails.
      requestBody:
        required: true
        content:
          application/x-www-form-urlencoded:
```

```
            schema:
              type: object
              properties:
                email: string
                subject:string
                message: string
      responses:
        '200':
          description: The email has been sent successfully
```

- notifications| POST http://localhost:8290/notification/ -> http://localhost:4222/api/notification/sms

```
paths:
  /notification/sms:
    post:
      summary: send the sms messages.
      requestBody:
        required: true
        content:
          application/x-www-form-urlencoded:
            schema:
              type: object
              properties:
                phone: string
                message string
      responses:
        '200':
          description: The sms message has been sent successfully
          content:
            application/json:
              schema:
                type: object
                items:
                  type: string
```

6. Payment Service

- payment | POST http://localhost:8290/payment/card -> http://localhost:3666/api/payment/card

```
paths:
  /payment/card:
    post:
      summary: Insert the card details.
      requestBody:
        required: true
        content:
          application/x-www-form-urlencoded:
            schema:
              type: object
              properties:
                cardNumber:string
                cvc:string
                cardHolder:string
                amount:string
                callback:string
                email string
      responses:
        '200':
          description: The card details has been sent successfully
```

```
          content:
            application/json:
              schema:
                type: object
                items:
                  type: string
```

- payment | POST http://localhost:8290/payment/mobile -> http://localhost:3666/api/payment/mobile

```
paths:
  /payment/mobile:
    post:
      summary: Insert the mobile payment details.
      requestBody:
        required: true
        content:
          application/x-www-form-urlencoded:
            schema:
              type: object
              properties:
                mobileNumber:string
                amount:string
                pin: string
                callback:string
      responses:
        '200':
          description: The mobile payment details has been sent successfully
          content:
             schema:
              type : string
```

- payment | GET http://localhost:8290/payment/getMobilePin/{mobile} -> http://localhost:3666/api/payment/getMobilePin/:mobile

```
paths:
  /payment/getMobilePin/{mobileNumber}:
    get:
      summary: Insert the mobile number and get the mobile pin.
    parameters:
        - in: query
          name: mobileNumber
          required: true
          schema:
            type: string
      responses:
        '200':
          description: The mobile pin number has been recived
           content:
              schema:
               type : string
```

Appendix

main.ts -auth

```
import { db } from '@middleware-scs3203/db';
import { createResponse } from '@middleware-scs3203/utilities';
import * as express from 'express';
import { json, urlencoded } from 'express';
import * as morgan from 'morgan';
import AuthController from './app/controllers/auth.cotroller';
import errorHandler from './app/middleware/error-handler';
import * as cors from 'cors';

const database = new db(
 process.env.DB_HOST,
 process.env.DB_USER,
 process.env.DB_PASSWORD,
 'login'
);

const authController = new AuthController(database);
const app = express();

app.use(json());
app.use(urlencoded({ extended: true }));
app.use(
 morgan(
   '[REQUEST] :method :url :status :response-time ms - :res[content-length]',
   {}
 )
);

//options for cors midddleware
const options: cors.CorsOptions = {
 allowedHeaders: [
   'Origin',
   'X-Requested-With',
   'Content-Type',
   'Accept',
   'X-Access-Token',
   'Authorization',
 ],
```

```
 credentials: true,
 methods: 'GET,HEAD,OPTIONS,PUT,PATCH,POST,DELETE',
 origin: 'http://localhost:3000',
 preflightContinue: false,
};

//use cors middleware
app.use(cors(options));

//add your routes

//enable pre-flight
app.options('*', cors(options));

// paths:
//   /authentication/login:
//     post:
//     summary : enter email and password to login to the system
//      requestBody:
//         required: true
//         content:
//           application/x-www-form-urlencoded:
//             schema:
//               type: object
//               properties:
//                 email:
//                   type: string
//                 password:
//                   type: string
//       responses:
//         200:
//           description: OK (successfully authenticated)
//         401:
//           description: Incorrect Credentials
//         1001:
//           description: User does not exist

app.post('/api/authentication/login', (req, res, next) => {
 authController
   .login(req.body.email, req.body.password)
   .then((result) => {
     res.json(createResponse(result));
```

```
   })
   .catch(next);
});

// paths:
//   /authentication/signup:
//     post:
//     summary : enter email and password to login to the system
//      requestBody:
//         required: true
//         content:
//           application/x-www-form-urlencoded:
//             schema:
//               type: object
//               properties:
//                 username:
//                   type: string
//                 email:
//                   type: string
//                 password:
//                   type: string
//                 role:
//                   type: string
//       responses:
//         UnauthorizedError:
//          description: Access token is missing or invalid
//         200:
//          description: Verfied Token

app.post('/api/authentication/signup', (req, res, next) => {
 authController
   .signup(req.body.username, req.body.email, req.body.password, req.body.role)
   .then((result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

// paths:
//   /authentication:
//     get:
//     summary : Verification of the tokens
```

```
//      requestBody:
//        required: true
//        content:
//          application/x-www-form-urlencoded:
//            schema:
//              type: object
//              properties:
//                username:
//                  type: string
//                email:
//                  type: string
//                password:
//                  type: string
//                role:
//                  type: string
//       responses:
//         200:
//           description: OK (The token successfully verified)
//         UnauthorizedError:
//           description: Access token is missing or invalid

app.get('/api/authentication', (req, res, next) => {
 authController
   .authenticate(req.header('authorization'))
   .then((result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

app.use(errorHandler);

const port = process.env.PORT_AUTH || 3444;
const server = app.listen(port, () => {
 console.log(`Listening at http://localhost:${port}/api`);
});
server.on('error', console.error);
```

auth.controller.ts

```
import { UnauthorizedException } from '@middleware-scs3203/utilities';
```

```
import { compare, hash } from 'bcrypt';
import { signToken, verifyToken } from '../services/token.service';

export default class AuthController {
 constructor(private db: any) {}

 private async getUserByEmail(email) {
   const user = await this.db.query('SELECT * FROM users WHERE email = ?', [
     email,
   ]); //get user by email

   if (user.length == 0) return false; //if user does not exist
   else return user[0]; //return user
 }

 //user login
 async login(email, password) {
   const user = await this.getUserByEmail(email); //get user by email
   if (!user) throw new UnauthorizedException('User does not exist'); //if user does
not exist

   const isPasswordCorrect = await compare(password, user.password); //compare
password
   if (!isPasswordCorrect)
     throw new UnauthorizedException('Incorrect Credentials'); //if password is
incorrect

   const token = signToken(user.uid, user.username, user.email, user.role); //signed
token
   const decoded = verifyToken(token); //verify token
   return token;
 }

 //user signup
 async signup(username, email, password, role) {
   const saltRounds = 10; //salt rounds
   const hashedPassword = await hash(password, saltRounds); //hashed password
   const { insertId } = await this.db.query(
     'INSERT INTO users (username, email, password ,role) VALUES (?, ?, ?,?)',
//insert user into database
     [username, email, hashedPassword, role]
   );
```

```
   const token = signToken(insertId, username, email, role); //signed token
   const decoded = verifyToken(token); //verify token
   return token;
 }

 async authenticate(token) {
   const decoded = verifyToken(token); //verify token
   if (!decoded) throw new UnauthorizedException('Invalid Token'); //if token is
invalid
   return decoded;
 }
}
```

token.services.ts

```
import { sign, verify } from 'jsonwebtoken';

function signToken(uid, username, email, role) {
 const token = sign(
   {
     id: uid,
     username: username,
     email: email,
     role: role,
   },
   process.env.JWT_SECRET
 );
 return token;
}

function verifyToken(token) {
 try {
   const decoded = verify(token, process.env.JWT_SECRET);
   return decoded;
 } catch (error) {
   return false;
 }
}

export { signToken, verifyToken };
```

Main.ts -customer

```
import * as express from 'express';
import { json } from 'express';
import { urlencoded } from 'express';
import * as morgan from 'morgan';
import { db } from '@middleware-scs3203/db';
import CustomerController from './app/controllers/customer.controller';
import errorHandler from './app/middleware/error-handler';
import { createResponse } from '@middleware-scs3203/utilities';
import authentication from './app/middleware/authentication';
import * as cors from 'cors';


const customerController = new CustomerController();
const app = express();

app.use(json());
app.use(urlencoded({ extended: true }));

app.use(
 morgan(
   '[REQUEST] :method :url :status :response-time ms - :res[content-length]',
   {}
 )
);

//options for cors midddleware
const options: cors.CorsOptions = {
 allowedHeaders: [
   'Origin',
   'X-Requested-With',
   'Content-Type',
   'Accept',
   'X-Access-Token',
   'Authorization'
 ],
 credentials: true,
 methods: 'GET,HEAD,OPTIONS,PUT,PATCH,POST,DELETE',
 origin: "http://localhost:3000",
 preflightContinue: false,
};
```

```
//use cors middleware
app.use(cors(options));

//add your routes

//enable pre-flight
app.options('*', cors(options));

declare global {
 // eslint-disable-next-line @typescript-eslint/no-namespace
 namespace Express {
   interface Request {
     user: {
       email: string;
       id: string;
       username: string;
       role: string;
     };
   }
 }
}

app.use(authentication);

// /customer/buy:
//     post:
//       summary: Sends payment details,delivery details and product details.
//       requestBody:products,payment details,delivery details
//         required: true
//         content:
//           application/json:
//             schema:
//        requestBody:
//         required: true
//          content:
//           application/x-www-form-urlencoded:
//             schema:
//               type: object
//               properties:
//                 products:
//                   type: object
//                 paymentDetails:
```

```
//                   type: object
//                 deliveryDetails:
//                   type: object
//                 token:
//                   type: string
//                 email:
//                   type: string
//       responses:
//         '200':
//           description:

app.post('/api/customer/buy', (req, res, next) => {
 const { products, paymentDetails, deliveryDetails } = req.body;
 const token = req.header('authorization').split(' ').pop();
 console.log(req.user);
 customerController
   .buyItems(products, paymentDetails, deliveryDetails, token ,req.user.email)
   .then((result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

// paths:
//   /customer/callback{data,token}:
//     get:
//       summary: Callback confirming the details processed sucessfully
//        parameters:
//         - in: path
//           name: data
//           required: true
//           schema:
//             type: object
//         - in: path
//           name: token
//           required: true
//           schema:
//             type: string
//               responses:
//                 '200':
//                   description:
```

```
app.get('/api/customer/callback/:data/:token', async (req, res, next) => {
 const { data, token } = req.params;
 customerController
   .completePayment(data, token)
   .then((result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

app.use(errorHandler);

const port = process.env.CUSTOMER || 3888;
const server = app.listen(port, () => {
 console.log(`Listening at http://localhost:${port}/api`);
});
server.on('error', console.error);
```

Customer.controller.ts

```
import axios from 'axios';
import { sign, verify } from 'jsonwebtoken';
import { userInfo } from 'os';

export default class CustomerController {
 // constructor(private db: any) {}

 async buyItems(products, paymentDetails, deliveryDetails, token, email) {
   console.log(email)
   const data = sign(
     {
       products: products,
       deliveryDetails: deliveryDetails,
     },
     process.env.JWT_SECRET
   );

   if (paymentDetails.paymentMethod == 'card')
     try {
       const payment = await axios.post('http://127.0.0.1:8290/payment/card', {
         cardNumber: paymentDetails.cardNumber,
         cvc: paymentDetails.cvc,
```

```
        cardHolder: paymentDetails.cardHolder,
        amount: paymentDetails.amount,
        callback: `http://127.0.0.1:8290/customer/callback/${data}/${token}`,
        email: email,
      });
      return payment.data.data;
    } catch (e) {
      throw new Error(e);
    }
  else if (paymentDetails.paymentMethod == 'mobile')
    try {
      const payment = await axios.post(
        'http://127.0.0.1:8290/payment/mobile',
        {
          mobileNumber: paymentDetails.mobileNumber,
          amount: paymentDetails.amount,
          pin: paymentDetails.pin,
          callback: `http://127.0.0.1:8290/customer/callback/${data}/${token}`,
        }
      );
      return payment.data.data;
    } catch (e) {
      throw new Error(e);
    }
  else throw new Error('Invalid payment method');
}

async completePayment(data, token) {
  const { products, deliveryDetails } = verify(data, process.env.JWT_SECRET);
  try {
    await axios.post(
      'http://127.0.0.1:8290/products/buy',
      { products: products },
      {
        headers: { authorization: token },
      }
    );
  } catch (e) {
    throw new Error(e);
  }

  if (deliveryDetails.deliveryStatus) {
```

```
    try {
      await axios.post(
        'http://127.0.0.1:8290/delivery/',
        {
          products: products,
          address: deliveryDetails.address,
          deliveryPrice: deliveryDetails.deliveryPrice,
        },
        {
          headers: { authorization: token },
        }
      );
    } catch (e) {
      throw new Error(e);
    }
  }

  return 'Payment process completed';
 }
}
```

Main.ts - delivery

```
import { db } from '@middleware-scs3203/db';
import { createResponse } from '@middleware-scs3203/utilities';
import * as express from 'express';
import { json, urlencoded } from 'express';
import * as morgan from 'morgan';
import DeliveryConstroller from './app/controllers/delivery.controller';
import errorHandler from './app/middleware/error-handler';
import authentication from './app/middleware/authentication';
import * as cors from 'cors';

const database = new db(
 process.env.DB_HOST,
 process.env.DB_USER,
 process.env.DB_PASSWORD,
 'delivery'
)

const deliveryController = new DeliveryConstroller(database);
const app = express();
```

```
app.use(json());
app.use(urlencoded({ extended: true }));
app.use(
 morgan(
   '[REQUEST] :method :url :status :response-time ms - :res[content-length]',
   {}
 )
);

//options for cors midddleware
const options: cors.CorsOptions = {
 allowedHeaders: [
   'Origin',
   'X-Requested-With',
   'Content-Type',
   'Accept',
   'X-Access-Token',
   'Authorization'
 ],
 credentials: true,
 methods: 'GET,HEAD,OPTIONS,PUT,PATCH,POST,DELETE',
 origin: "http://localhost:3000",
 preflightContinue: false,
};

//use cors middleware
app.use(cors(options));

//add your routes

//enable pre-flight
app.options('*', cors(options));

declare global {
 // eslint-disable-next-line @typescript-eslint/no-namespace
 namespace Express {
   interface Request {
     user: {
       email: string;
       id: string;
       username: string;
```

```
       role: string;
     };
   }
 }
}

app.use(authentication);

// paths:
//   /delivery/{address}:
//     get:
//       summary: request the delivery rate.
//     parameters:
//         - in: query
//           name: address
//           required: true
//           schema:
//             type: string
//         '200':
//           description: Returns delivery rates
//           content:
//             application/json:
//               schema:
//                 type: object
//                 items:
//                   type: string

app.get('/api/delivery/:address', (req, res ,next) => {
 const address = req.params.address;
 deliveryController
   .getQuote(address)
   .then((result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

// paths:
//   /delivery:
//     post:
//       summary: request the delivery rate.
//       requestBody:
```

```
//          required: true
//          content:
//            application/x-www-form-urlencoded:
//              schema:
//                type: object
//                properties:
//                  username:
//                    type: string
//                  address:
//                    type: string
//                  products:
//                    type: object
//                  deliveryPrice:
//                    type: number
//          '200':
//            description: Delivery Successful
//            content:
//              application/json:
//                schema:
//                  type: object
//                  items:
//                    type: string

app.post('/api/delivery', (req, res ,next) => {
 deliveryController
   .addDelivery(req.user.username, req.body.address, req.body.products,
req.body.deliveryPrice)
   .then((result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

app.use(errorHandler);

const port = process.env.DELIVERY||3555;
const server = app.listen(port, () => {
 console.log(`Listening at http://localhost:${port}/api`);
});
server.on('error', console.error);
```

Delivery.controller.ts

```
import { ServerException } from '@middleware-scs3203/utilities';

export default class DeliveryConstroller {
 constructor(private db: any) {}

 //get delivery rate
 async getQuote(address) {
   return {
     price:
       'colombo' in address.toLocaleLowerCase
         ? Math.floor(Math.random() * 500)
         : Math.floor(Math.random() * 2000),
   };
 }

 //add delivery
 async addDelivery(
   customer: string,
   address: string,
   items: object,
   deliveryPrice: number
 ) {
   try {
     await this.db.query(
       'INSERT INTO delivery (customer, address, items, delivery_price) VALUES
(?,?,?,?)', //insert delivery into database
       [customer, address, JSON.stringify(items), deliveryPrice]
     );
     return 'Delivery Successful'; //return success message
   } catch (error) {
     throw new Error(error) //throw error
   }
 }
}
```

Main.ts - inventory

```
import { db } from '@middleware-scs3203/db';
import { createResponse } from '@middleware-scs3203/utilities';
import * as express from 'express';
```

```
import * as morgan from 'morgan';
import { json, urlencoded } from 'express';
import ProductController from './app/controllers/product.controller';
import errorHandler from './app/middleware/error-handler';
import authentication from './app/middleware/authentication';
import * as cors from 'cors';

const database = new db(
 process.env.DB_HOST,
 process.env.DB_USER,
 process.env.DB_PASSWORD,
 'inventory'
);

const productController = new ProductController(database);
const app = express();

app.use(json());
app.use(urlencoded({ extended: true }));
app.use(
 morgan(
   '[REQUEST] :method :url :status :response-time ms - :res[content-length]',
   {}
 )
);

//options for cors midddleware
const options: cors.CorsOptions = {
 allowedHeaders: [
   'Origin',
   'X-Requested-With',
   'Content-Type',
   'Accept',
   'X-Access-Token',
   'Authorization',
 ],
 credentials: true,
 methods: 'GET,HEAD,OPTIONS,PUT,PATCH,POST,DELETE',
 origin: 'http://localhost:3000',
 preflightContinue: false,
};
```

```
//use cors middleware
app.use(cors(options));

//add your routes

//enable pre-flight
app.options('*', cors(options));

declare global {
 // eslint-disable-next-line @typescript-eslint/no-namespace
 namespace Express {
   interface Request {
     user: {
       email: string;
       id: string;
       username: string;
       role: string;
     };
   }
 }
}

app.use(authentication);

// paths:
//   /products/:name:
//     get:
//       summary: Returns the product with the searched name.
//       description:
//       responses:
//         '200':
//           description: A JSON array with the object containing all the details of
the product
//           content:
//             application/json:
//               schema:
//                 type: array
//                 items:
//                   type: string

// endpoint to get  products
app.get('/api/products/:name', (req, res, next) => {
```

```
 const name = req.params.name;
 productController
   .getProducts(name)
   .then((result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

// paths:
//   /products:
//     get:
//       summary: Returns the list of product list.
//       description:
//       responses:
//         '200':
//           description: A JSON array with the list of products
//           content:
//             application/json:
//               schema:
//                 type: array
//                 items:
//                   type: string

app.get('/api/products', (req, res, next) => {
 productController
   .getAllProducts(req.user)
   .then((result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

// paths:
//   /products:
//     post:
//       summary: update a product.
//       requestBody:
//         required: true
//         content:
//           application/x-www-form-urlencoded:
//             schema:
```

```
//               type: object
//               properties:
//                 name:
//                   type: string
//                 price:
//                   type: string
//                 amount:
//                   type: string
//                 image_url:
//                   type: string
//       responses:
//         '200':
//           description: The product has been successfully added
//           content:
//             application/json:
//               schema:
//                 type: object
//                 items:
//                   type: string

//endpoint to add products
app.post('/api/products', (req, res, next) => {
 const { name, price, amount, img_url } = req.body;
 productController
   .addProduct(name, price, amount, img_url, req.user)
   .then((result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

// paths:
//   /products/{productId}:
//     put:
//       summary: Updates a product.
//        parameters:
//         - in: query
//           name: id
//           required: true
//           schema:
//             type: string
//       requestBody:
```

```
//         required: true
//         content:
//           application/x-www-form-urlencoded:
//             schema:
//               type: object
//               properties:
//                 name:
//                   type: string
//                 price:
//                   type: string
//                 amount:
//                   type: string
//                 image_url:
//                   type: string
//       responses:
//         '200':
//           description: The product has been successfully updated
//           content:
//             application/json:
//               schema:
//                 type: object
//                 items:
//                   type: string

app.put('/api/products/:id', (req, res, next) => {
 const { name, price, amount, img_url } = req.body;
 const id = parseInt(req.params.id);

 productController
   .updateProduct(id, name, price, amount, img_url)
   .then((result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

// paths:
//   /products/buy:
//     post:
//       summary: Updates the amount of a product.
//        parameters:
//         - in: query
```

```
//           name: id
//           required: true
//           schema:
//             type: string
//       requestBody:
//         required: true
//         content:
//           application/x-www-form-urlencoded:
//             schema:
//               type: object
//               properties:
//                 amount:
//                   type: string

//       responses:
//         '200':
//           description: The product amount has been successfully updated
//           content:
//             application/json:
//               schema:
//                 type: object
//                 items:
//                   type: string

app.post('/api/products/buy', (req, res, next) => {
 const { products } = req.body;
 productController
   .updateAmount(products)
   .then((result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

// paths:
//   /products/{productId}:
//     delete:
//       summary: Deletes the  product.
//        parameters:
//         - in: query
//           name: id
//           required: true
```

```
//           schema:
//             type: string
//       responses:
//         '200':
//           description: The product amount has been successfully deleted
//           content:
//             application/json:
//               schema:
//                 type: object
//                 items:
//                   type: string

app.delete('/api/products/:id', (req, res, next) => {
 const id = parseInt(req.params.id);
 productController
   .deleteProduct(id)
   .then((result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

app.use(errorHandler);

const port = 4111;
const server = app.listen(port, () => {
 console.log(`Listening at http://localhost:${port}/api`);
});
server.on('error', console.error);
```

Product.controller.ts

```
import {
 NotFoundException,
 ServerException,
} from '@middleware-scs3203/utilities';

export default class ProductController {
 constructor(private db: any) {}

 private async getProductById(id: number) {
   const product = await this.db.query('SELECT * FROM products WHERE id = ?', [
```

```
    id,
  ]);
  if (product.length == 0) throw new NotFoundException('Product not found');
  return product[0];
}

async getProducts(name) {
  const query = `SELECT * FROM products WHERE name LIKE '%${name}%'`;
  const products = await this.db.query(query);
  if (products.length == 0) return [];
  return products;
}

async getAllProducts(user) {
  let products;
  if (user.role == 'buyer') {
     products = await this.db.query('SELECT * FROM products');
  } else {
     products = await this.db.query('SELECT * FROM products where uid = ?',
[user.id]);
  }
  if (products.length == 0) return [];
  return products;
}

async addProduct(
  name: string,
  price: number,
  amount: number,
  img_url: string,
  user : any,
): Promise<void> {
  try {
    //TODO: add validation for price and amount

    await this.db.query(
      'INSERT INTO products (name, price, amount, img_url,uid) VALUES (?, ?, ?, ? ,
?)',
      [name, price, amount, img_url ,user.id]
    );
  } catch (error) {
    throw new ServerException('Error in adding product');
```

```
    }
  }

  async updateProduct(
    id: number,
    name: string,
    price: number,
    amount: number,
    img_url: string
  ) {
    const productInfo = await this.getProductById(id);
    //TODO: add validation for price and amount
    console.log(typeof name, name);
    name = name ? name : productInfo.name;
    price = price ? price : productInfo.price;
    amount = amount ? amount : productInfo.amount;
    img_url = img_url ? img_url : productInfo.img_url;

    try {
      await this.db.query(
        'UPDATE products SET name = ?, price = ?, amount = ?, img_url = ? WHERE id =
?',
        [name, price, amount, img_url, id]
      );
    } catch (error) {
      throw new ServerException('Product update failed');
    }

    return 'Product updated';
  }

  async updateAmount(productArray: any) {
    for (const product of productArray) {

      const productInfo = await this.getProductById(product.id);
      if (productInfo.amount < product.amount) {
        throw new ServerException('Not enough product');
      }

      const newAmount = productInfo.amount - product.amount;
      try {
        await this.db.query('UPDATE products SET amount = ? WHERE id = ?', [
```

```
        newAmount,
        product.id,
      ]);
    } catch (error) {
      throw new ServerException('Product update failed');
    }
  };

  return 'Inventory updated';
 }
 async deleteProduct(id: number) {
  try {
    await this.db.query('DELETE FROM products WHERE id = ?', [id]);
  } catch (error) {
    throw new ServerException('Product delete failed');
  }
 }
}
```

Main.ts - notification

```
import { db } from '@middleware-scs3203/db';
import { createResponse } from '@middleware-scs3203/utilities';
import axios from 'axios';
import * as express from 'express';
import { json, urlencoded } from 'express';
import * as morgan from 'morgan';
import * as cors from 'cors';
import NotificationController from './app/controllers/notification.controller';

const app = express();

const notificationController = new NotificationController();

declare global {
 // eslint-disable-next-line @typescript-eslint/no-namespace
 namespace Express {
  interface Request {
    user: {
      email: string;
      id: string;
      username: string;
```

```
      role: string;
    };
  }
 }
}

//options for cors midddleware
const options: cors.CorsOptions = {
 allowedHeaders: [
   'Origin',
   'X-Requested-With',
   'Content-Type',
   'Accept',
   'X-Access-Token',
   'Authorization'
 ],
 credentials: true,
 methods: 'GET,HEAD,OPTIONS,PUT,PATCH,POST,DELETE',
 origin: "http://localhost:3000",
 preflightContinue: false,
};

//use cors middleware
app.use(cors(options));

//add your routes

//enable pre-flight
app.options('*', cors(options));

app.use(json());
app.use(urlencoded({ extended: true }));
app.use(
 morgan(
   '[REQUEST] :method :url :status :response-time ms - :res[content-length]',
   {}
 )
);

// paths:
//   /notification/mail:
//     post:
```

```
//       summary: send the emails.
//       requestBody:
//         required: true
//         content:
//           application/x-www-form-urlencoded:
//             schema:
//               type: object
//               properties:
//                 email:
//                   type: string
//                 subject:
//                   type: string
//                 message:
//                   type: string
//       responses:
//         '200':
//           description: The email has been sent successfully
//           content:
//             application/json:
//               schema:
//                 type: object
//                 items:
//                   type: string

app.post('/api/notification/mail', async (req, res) => {
   const { email, subject, message } = req.body;
   notificationController.emailNotification(email, subject, message).then((result) =>
{
       res.send(createResponse(result));
   }
   ).catch((err) => {
       res.send(createResponse(err));
   });
})


// paths:
//   /notification/sms:
//     post:
//       summary: send the sms messages.
//       requestBody:
//         required: true
```

```
//          content:
//            application/x-www-form-urlencoded:
//              schema:
//                type: object
//                properties:
//                  phone:
//                    type: string
//                  message:
//                    type: string
//        responses:
//          '200':
//            description: The sms message has been sent successfully
//            content:
//              application/json:
//                schema:
//                  type: object
//                  items:
//                    type: string


app.post('/api/notification/sms', async (req, res) =>
{
   const { phone, message } = req.body;
   notificationController.smsNotification(phone, message).then((result) => {
       res.send(createResponse(result));
   }
   ).catch((err) => {
       res.send(createResponse(err));
   });
}
)

const port = process.env.NOTIFICATION || 4222;
const server = app.listen(port, () => {
 console.log(`Listening at http://localhost:${port}/api`);
});
server.on('error', console.error);
```

Notification.controller.ts

```
export default class NotificationController {
```

```
    async emailNotification(email: string, subject: string, message: string) {
        return `Email sent to ${email} with subject ${subject} and message ${message}`;
    }

    async smsNotification(phone: string, message: string) {
        return `SMS sent to ${phone} with message ${message}`;
    }

}
```

Main.ts - payment

```
import { db } from '@middleware-scs3203/db';
import { createResponse } from '@middleware-scs3203/utilities';
import axios from 'axios';
import * as express from 'express';
import { json, urlencoded } from 'express';
import * as morgan from 'morgan';
import PaymentController from './app/controllers/payment.controller';
import errorHandler from './app/middleware/error-handler';
import * as cors from 'cors';
import authentication from './app/middleware/authentication';

const database = new db(
 process.env.DB_HOST,
 process.env.DB_USER,
 process.env.DB_PASSWORD,
 'payment'
)

const paymentController = new PaymentController(database);
const app = express();


declare global {
 // eslint-disable-next-line @typescript-eslint/no-namespace
 namespace Express {
   interface Request {
     user: {
       email: string;
       id: string;
       username: string;
       role: string;
```

```
    };
  }
 }
}

//options for cors midddleware
const options: cors.CorsOptions = {
 allowedHeaders: [
   'Origin',
   'X-Requested-With',
   'Content-Type',
   'Accept',
   'X-Access-Token',
   'Authorization'
 ],
 credentials: true,
 methods: 'GET,HEAD,OPTIONS,PUT,PATCH,POST,DELETE',
 origin: "http://localhost:3000",
 preflightContinue: false,
};

//use cors middleware
app.use(cors(options));

//add your routes

//enable pre-flight
app.options('*', cors(options));

app.use(json());
app.use(urlencoded({ extended: true }));
app.use(
 morgan(
   '[REQUEST] :method :url :status :response-time ms - :res[content-length]',
   {}
 )
);

// paths:
//   /payment/card:
//     post:
//       summary: Insert the card details.
```

```
//       requestBody:
//         required: true
//         content:
//           application/x-www-form-urlencoded:
//             schema:
//               type: object
//               properties:
//                 cardNumber:
//                   type: string
//                 cvc:
//                   type: string
//                 cardHolder:
//                   type: string
//                 amount:
//                   type: string
//                 callback:
//                   type: string
//                 email:
//                   type: string
//       responses:
//         '200':
//           description: The card details has been sent successfully
//           content:
//             application/json:
//               schema:
//                 type: object
//                 items:
//                   type: string


app.post('/api/payment/card', (req, res ,next) => {
 paymentController
   .cardPayment(req.body.cardNumber, req.body.cvc, req.body.cardHolder,
req.body.amount,req.body.callback,req.body.email)
   .then(async (result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

// paths:
//   /payment/mobile:
```

```
//     post:
//       summary: Insert the mobile payment details.
//       requestBody:
//         required: true
//         content:
//           application/x-www-form-urlencoded:
//             schema:
//               type: object
//               properties:
//                 mobileNumber:
//                   type: string
//                 amount:
//                   type: string
//                 pin:
//                   type: string
//                 callback:
//                   type: string
//       responses:
//         '200':
//           description: The mobile payment details has been sent successfully
//           content:
//             application/json:
//               schema:
//                 type: object
//                 items:
//                   type: string

app.post('/api/payment/mobile', (req, res ,next) => {
 paymentController
   .mobilePayment(req.body.mobileNumber, req.body.amount ,req.body.pin
,req.body.callback)
   .then(async (result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

// paths:
//   /payment/getMobilePin/{mobileNumber}:
//     get:
//       summary: Insert the mobile number and get the mobile pin.
//     parameters:
```

```
//         - in: query
//           name: mobileNumber
//           required: true
//           schema:
//             type: string
//       responses:
//         '200':
//           description: The mobile pin number has been recived
//           content:
//             application/json:
//               schema:
//                 type: object
//                 items:
//                   type: string

app.get('/api/payment/getMobilePin/:mobileNumber', (req, res ,next) => {
 paymentController
   .getMobilePaymentPin(req.params.mobileNumber)
   .then((result) => {
     res.json(createResponse(result));
   })
   .catch(next);
});

app.use(errorHandler);

const port = process.env.PAYMENT || 3666;
const server = app.listen(port, () => {
 console.log(`Listening at http://localhost:${port}/api`);
});
server.on('error', console.error);
```

Payment.contoller.ts

```
import {
 ServerException,
 UnauthorizedException,
} from '@middleware-scs3203/utilities';
import axios from 'axios';

export default class PaymentController {
 constructor(private db: any) {}
```

```
private async paymentSuccessMessage(type: string, body: any) {
  let message;
  try {
    if (type == 'mobile')
      message = await axios.post('http://127.0.0.1:8290/notification/sms', {
        phone: body.phone,
        message: body.message + ' of' + body.amount,
        subject : body.subject
      });
    else if (type == 'email')
      message = await axios.post('http://127.0.0.1:8290/notification/mail', {
        email: body.email,
        subject: body.subject,
        message: body.message + ' of' + body.amount,
      });
    else throw new ServerException('Invalid type');
  } catch (e) {
    throw new ServerException('Payment success message failed' + e);
  }
  console.log(message.data.data);
  return message.data.data;
}

async cardPayment(
  cardNumber: string,
  cvc: string,
  cardHolder: string,
  amount: number,
  callback: string,
  email: string
) {
  try {
    console.log(cardNumber, cvc, cardHolder, amount, callback, email);
    await this.db.query(
      'INSERT INTO card_payment (cardNumber, cvc, cardHolder, amount) VALUES
(?,?,?,?)',
      [cardNumber, cvc, cardHolder, amount]
    );
    await axios.get(callback);
    const message = await this.paymentSuccessMessage('email', {
      email: email,
      subject: 'Payment Gateway',
```

```
      message: 'Your payment was successful',
      amount: amount,
    });
    return message.data.data;
  } catch (e) {
    throw new ServerException('Card payment failed' + e);
  }

}

async mobilePayment(
  mobileNumber: string,
  amount: number,
  pin: string,
  callback: string
) {
  try {
    const result = await this.db.query(
      'SELECT * FROM mobile_payment WHERE pin = ? AND mobileNumber = ? ',
      [pin, mobileNumber]
    );
    if (result.length == 0)
      throw new UnauthorizedException('Invalid details');
    await this.db.query(
      'UPDATE mobile_payment SET mobileNumber = ?, amount = ? WHERE pin = ?',
      [mobileNumber, amount, pin]
    );

    axios.get(callback);
    return await this.paymentSuccessMessage('mobile', {
      phone: mobileNumber,
      message: 'Your payment was successful',
      subject : 'Payment Gateway',
      amount: amount,
    });
  } catch (e) {
    throw new ServerException('Mobile payment failed' + e);
  }
}

async getMobilePaymentPin(mobileNumber: string) {
  const pin = Math.floor(Math.random() * 10000);
```

```
  try {
    await this.db.query(
      'INSERT INTO mobile_payment (mobileNumber, pin) VALUES (?,?)',
      [mobileNumber, pin]
    );
    const message = await axios.post(
      'http://127.0.0.1:8290/notification/sms',
      {
        phone: mobileNumber,
        message: `Your pin is ${pin}`,
      }
    );
    return message.data.data;
  } catch (e) {
    throw new ServerException('Mobile payment failed' + e);
  }
 }
}
```