

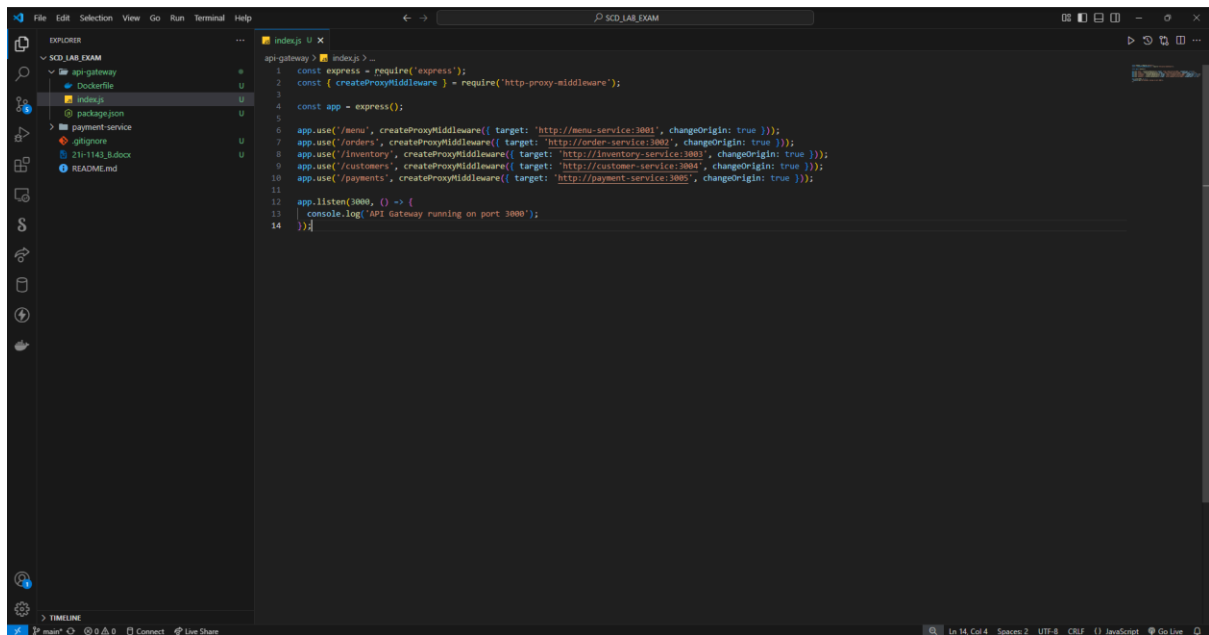
Umar Farooq

21i-1143

SE-B

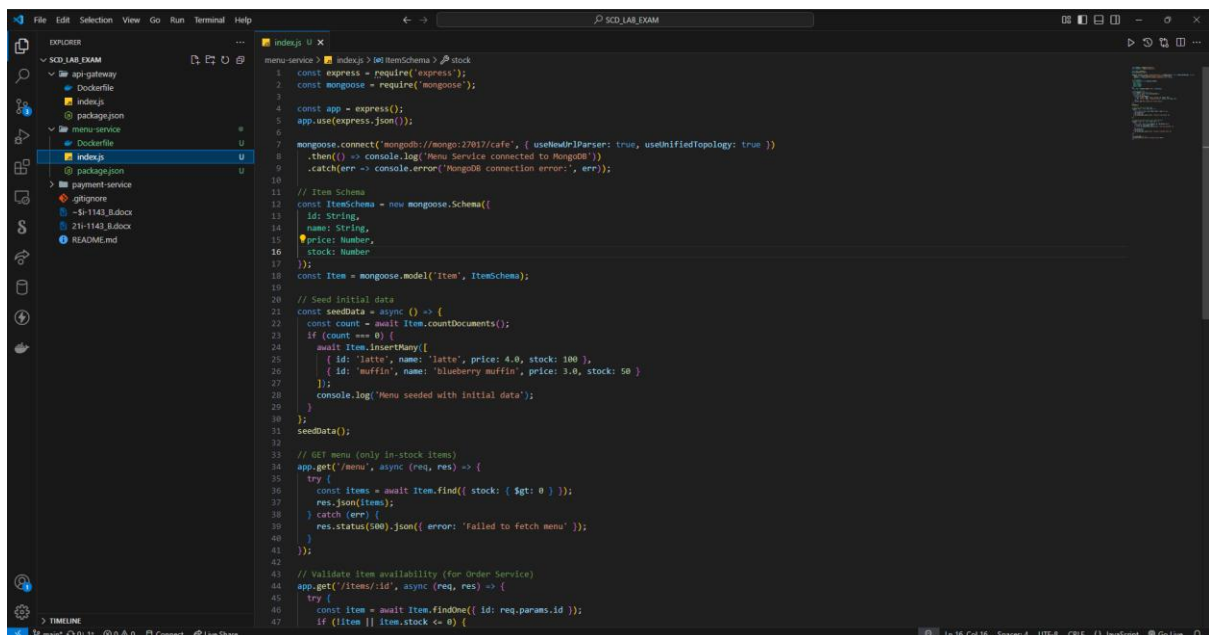
Github repo link: https://github.com/yuri8822/SCD_LAB_EXAM

Api gateway made:



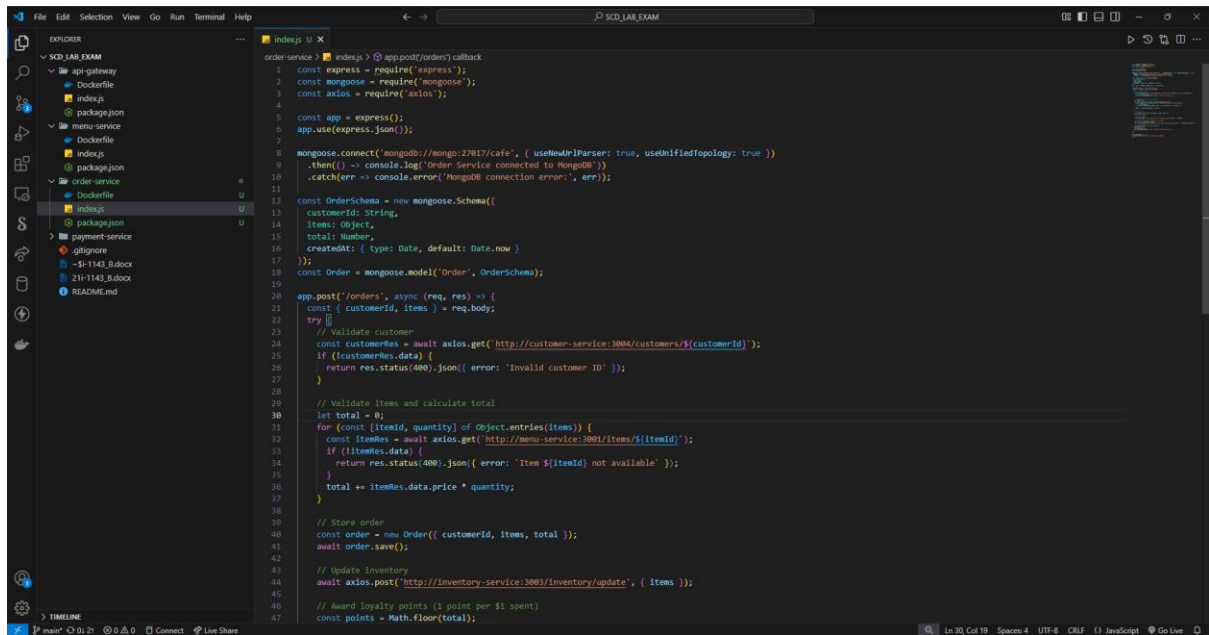
```
api-gateway > index.js > ...
1 const express = require('express');
2 const { createProxyMiddleware } = require('http-proxy-middleware');
3
4 const app = express();
5
6 app.use('/menu', createProxyMiddleware({ target: 'http://menu-service:3001', changeOrigin: true }));
7 app.use('/orders', createProxyMiddleware({ target: 'http://order-service:3002', changeOrigin: true }));
8 app.use('/inventory', createProxyMiddleware({ target: 'http://inventory-service:3003', changeOrigin: true }));
9 app.use('/customers', createProxyMiddleware({ target: 'http://customer-service:3004', changeOrigin: true }));
10 app.use('/payments', createProxyMiddleware({ target: 'http://payment-service:3005', changeOrigin: true }));
11
12 app.listen(3000, () => {
13   console.log('API Gateway running on port 3000');
14 })
```

Menu service done:



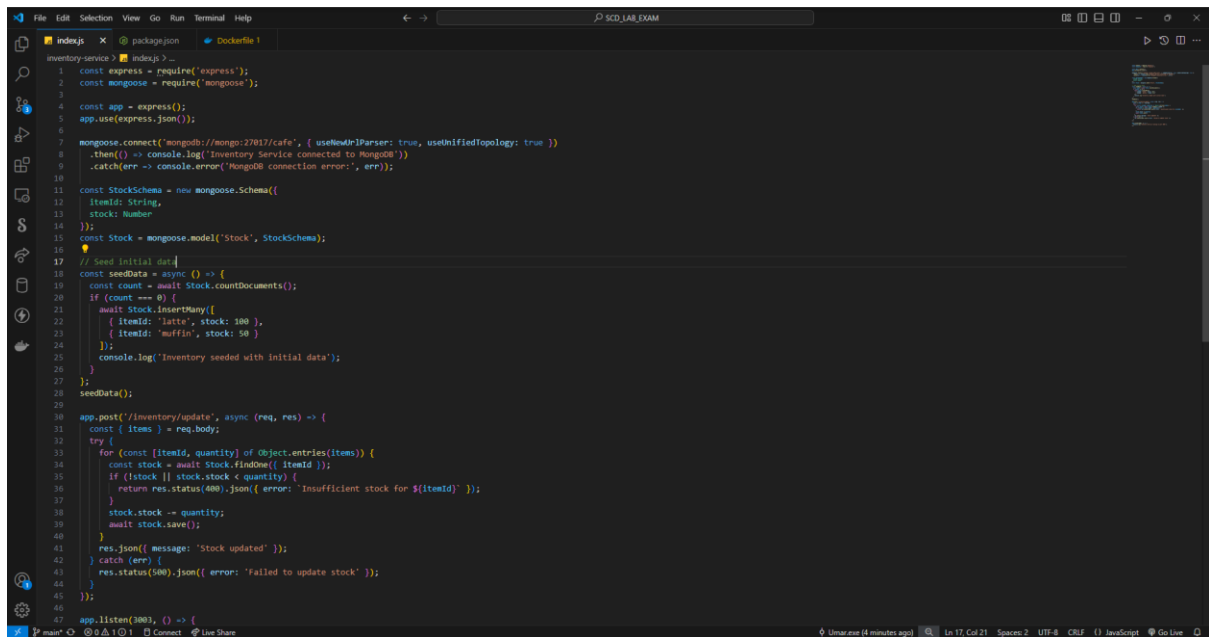
```
menu-service > index.js > @itemSchema > stock
1 const express = require('express');
2 const mongoose = require('mongoose');
3
4 const app = express();
5 app.use(express.json());
6
7 mongoose.connect('mongodb://mongo:27017/cafe', { useNewUrlParser: true, useUnifiedTopology: true })
8 .then(() => console.log('Menu Service connected to MongoDB'))
9 .catch(err => console.error('MongoDB connection error:', err));
10
11 // Item Schema
12 const ItemSchema = new mongoose.Schema({
13   id: String,
14   name: String,
15   price: Number,
16   stock: Number
17 });
18 const Item = mongoose.model('Item', ItemSchema);
19
20 // Seed initial data
21 const seedData = async () => {
22   const count = await Item.countDocuments();
23   if (count === 0) {
24     await Item.insertMany([
25       { id: 'latte', name: 'latte', price: 4.0, stock: 100 },
26       { id: 'muffin', name: 'blueberry muffin', price: 3.0, stock: 50 }
27     ]);
28     console.log('Menu seeded with initial data');
29   }
30 };
31 seedData();
32
33 // GET menu (only in-stock items)
34 app.get('/menu', async (req, res) => {
35   try {
36     const items = await Item.find({ stock: { $gt: 0 } });
37     res.json(items);
38   } catch (err) {
39     res.status(500).json({ error: 'Failed to fetch menu' });
40   }
41 });
42
43 // Validate item availability (for Order Service)
44 app.get('/items/:id', async (req, res) => {
45   try {
46     const item = await Item.findOne({ id: req.params.id });
47     if (!item || item.stock <= 0) {
```

Order service done:



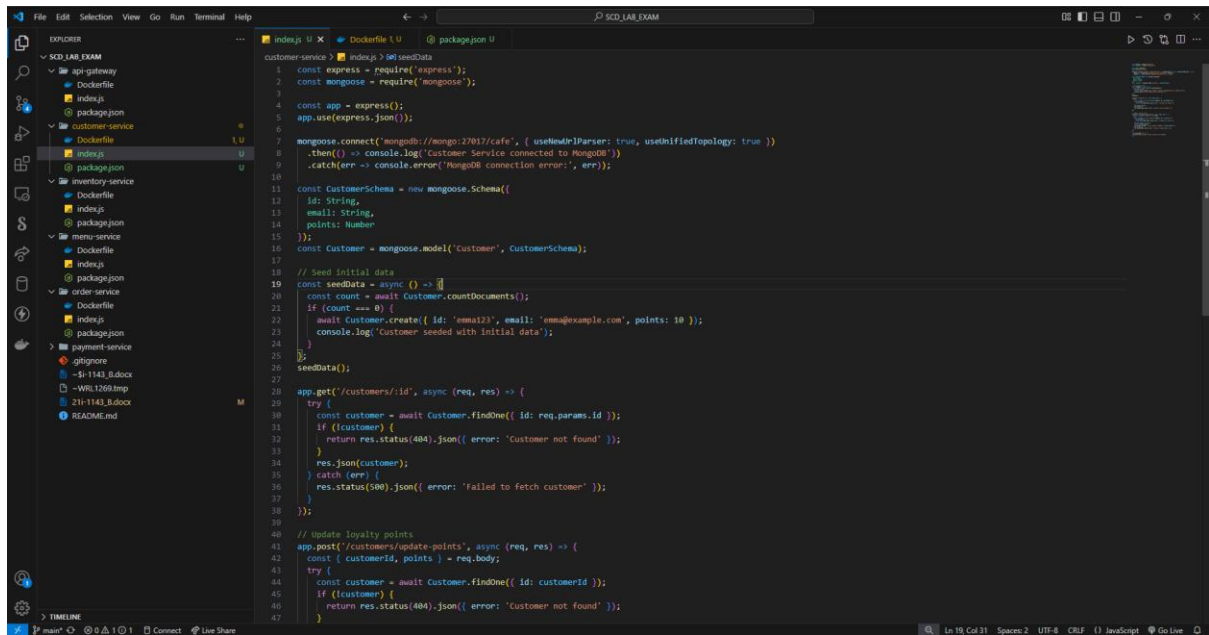
```
order-service > index.js @ app.post('/orders') callback
1 const express = require('express');
2 const mongoose = require('mongoose');
3 const axios = require('axios');
4
5 const app = express();
6 app.use(express.json());
7
8 mongoose.connect('mongodb://mongo:27017/caf ', { useNewUrlParser: true, useUnifiedTopology: true })
9 .then(() => console.log('Order Service connected to MongoDB'))
10 .catch(err => console.error('MongoDB connection error:', err));
11
12 const OrderSchema = new mongoose.Schema({
13   customerId: String,
14   items: Object,
15   total: Number,
16   createdAt: { type: Date, default: Date.now }
17 });
18 const Order = mongoose.model('Order', OrderSchema);
19
20 app.post('/orders', async (req, res) => {
21   const { customerId, items } = req.body;
22   try {
23     // Validate customer
24     const customerRes = await axios.get('http://customer-service:3004/customers/${$customerId}');
25     if (!customerRes.data) {
26       return res.status(400).json({ error: 'Invalid customer ID' });
27     }
28
29     // Validate items and calculate total
30     let total = 0;
31     for (const [itemId, quantity] of Object.entries(items)) {
32       const itemRes = await axios.get('http://menu-service:3001/items/${$itemId}');
33       if (!itemRes.data) {
34         return res.status(400).json({ error: 'Item ${{itemId}} not available' });
35       }
36       total += itemRes.data.price * quantity;
37     }
38
39     // Store order
40     const order = new Order({ customerId, items, total });
41     await order.save();
42
43     // Update inventory
44     await axios.post('http://inventory-service:3003/inventory/update', { items });
45
46     // Award loyalty points (1 point per $1 spent)
47     const points = Math.floor(total);
```

Inventory service done:



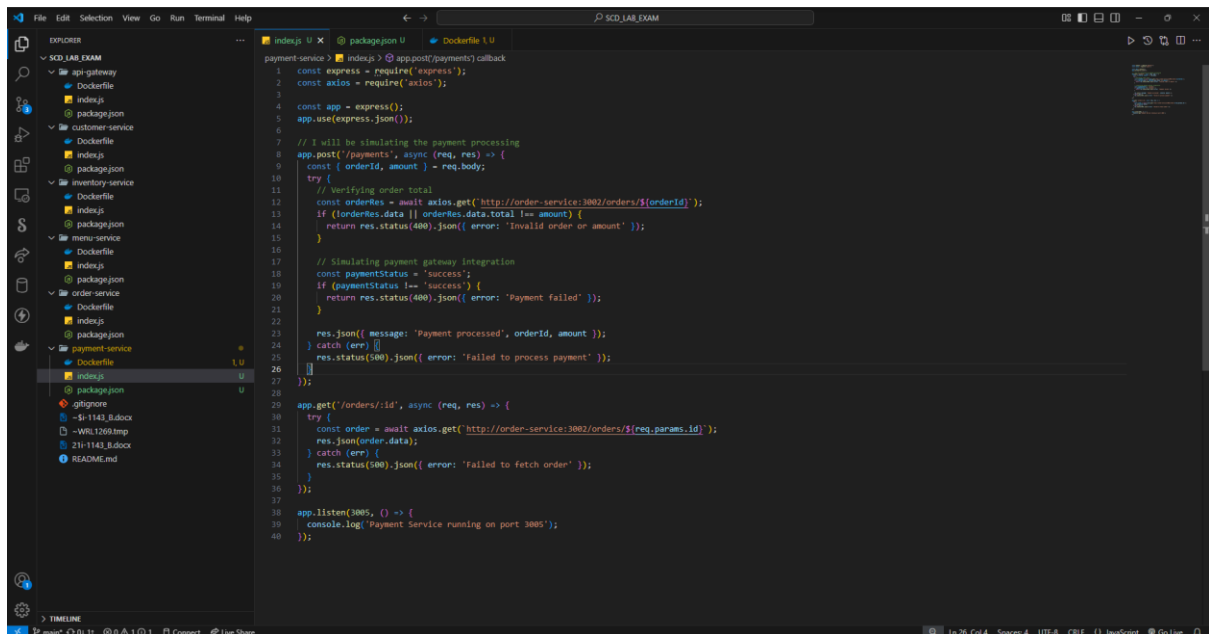
```
inventory-service > index.js ...
1 const express = require('express');
2 const mongoose = require('mongoose');
3
4 const app = express();
5 app.use(express.json());
6
7 mongoose.connect('mongodb://mongo:27017/caf ', { useNewUrlParser: true, useUnifiedTopology: true })
8 .then(() => console.log('Inventory Service connected to MongoDB'))
9 .catch(err => console.error('MongoDB connection error:', err));
10
11 const StockSchema = new mongoose.Schema({
12   itemId: String,
13   stock: Number
14 });
15 const Stock = mongoose.model('Stock', StockSchema);
16
17 // Seed initial data
18 const seedData = async () => {
19   const count = await Stock.countDocuments();
20   if (count === 0) {
21     await Stock.insertMany([
22       { itemId: 'latte', stock: 100 },
23       { itemId: 'muffin', stock: 50 }
24     ]);
25     console.log('Inventory seeded with initial data');
26   }
27 };
28 seedData();
29
30 app.post('/inventory/update', async (req, res) => {
31   const { items } = req.body;
32   try {
33     for (const [itemId, quantity] of Object.entries(items)) {
34       const stock = await Stock.findOne({ itemId });
35       if (!stock || stock.stock < quantity) {
36         return res.status(400).json({ error: 'Insufficient stock for ${{itemId}}' });
37       }
38       stock.stock -= quantity;
39       await stock.save();
40     }
41     res.json({ message: 'Stock updated' });
42   } catch (err) {
43     res.status(500).json({ error: 'Failed to update stock' });
44   }
45 });
46
47 app.listen(3003, () => {
```

Customer service done:



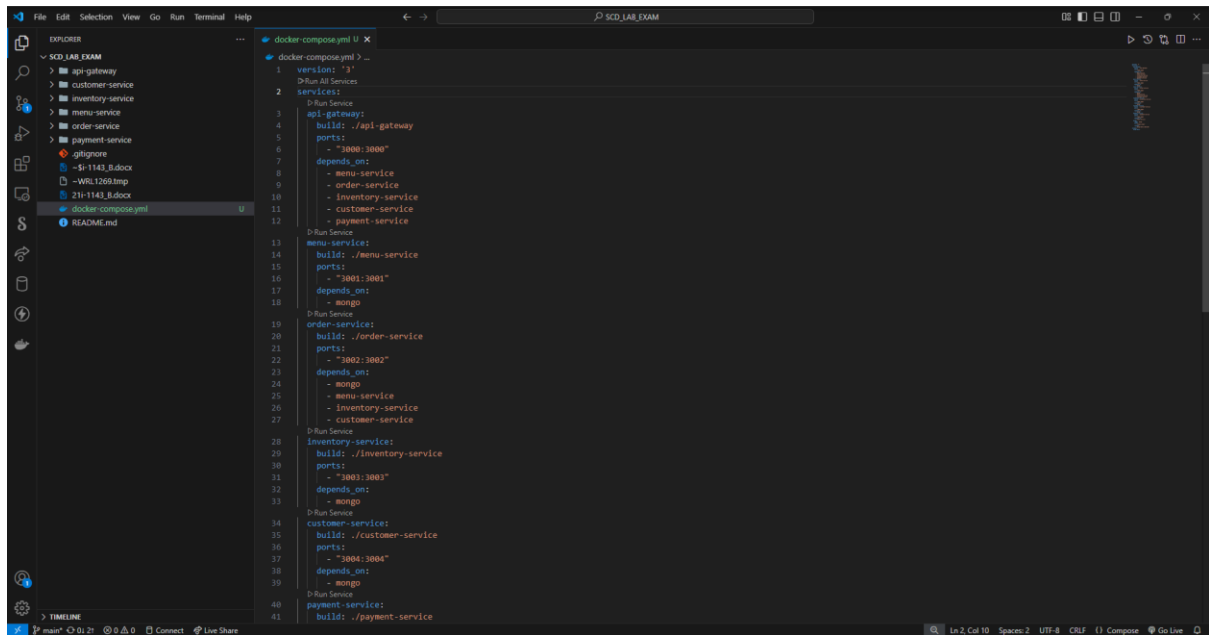
```
customer-service > index.js > @seedData
1 const express = require('express');
2 const mongoose = require('mongoose');
3
4 const app = express();
5 app.use(express.json());
6
7 mongoose.connect('mongodb://mongo:27017/cafe', { useNewUrlParser: true, useUnifiedTopology: true })
8 .then(() => console.log('Customer Service connected to MongoDB'))
9 .catch(err => console.error('MongoDB connection error:', err));
10
11
12 const CustomerSchema = new mongoose.Schema({
13   id: String,
14   email: String,
15   points: Number
16 });
17 const Customer = mongoose.model('Customer', CustomerSchema);
18
19 // Seed initial data
20 const seedData = async () => {
21   const count = await Customer.countDocuments();
22   if (count === 0) {
23     await Customer.create([ { id: 'emma123', email: 'emma@example.com', points: 10 } ]);
24     console.log('Customer seeded with initial data');
25   }
26   seedData();
27 }
28
29 app.get('/customers/:id', async (req, res) => {
30   try {
31     const customer = await Customer.findOne({ id: req.params.id });
32     if (!customer) {
33       return res.status(404).json({ error: 'Customer not found' });
34     }
35     res.json(customer);
36   } catch (err) {
37     res.status(500).json({ error: 'Failed to fetch customer' });
38   }
39 });
40
41 // Update loyalty points
42 app.post('/customers/update-points', async (req, res) => {
43   const { customerId, points } = req.body;
44   try {
45     const customer = await Customer.findOne({ id: customerId });
46     if (!customer) {
47       return res.status(404).json({ error: 'Customer not found' });
48     }
49     customer.points = points;
50     await customer.save();
51     res.json({ message: 'Points updated', points });
52   } catch (err) {
53     res.status(500).json({ error: 'Failed to update points' });
54   }
55 });
56
57 app.listen(3005, () => {
58   console.log('Customer Service running on port 3005');
59 });
```

Done with Payment service:



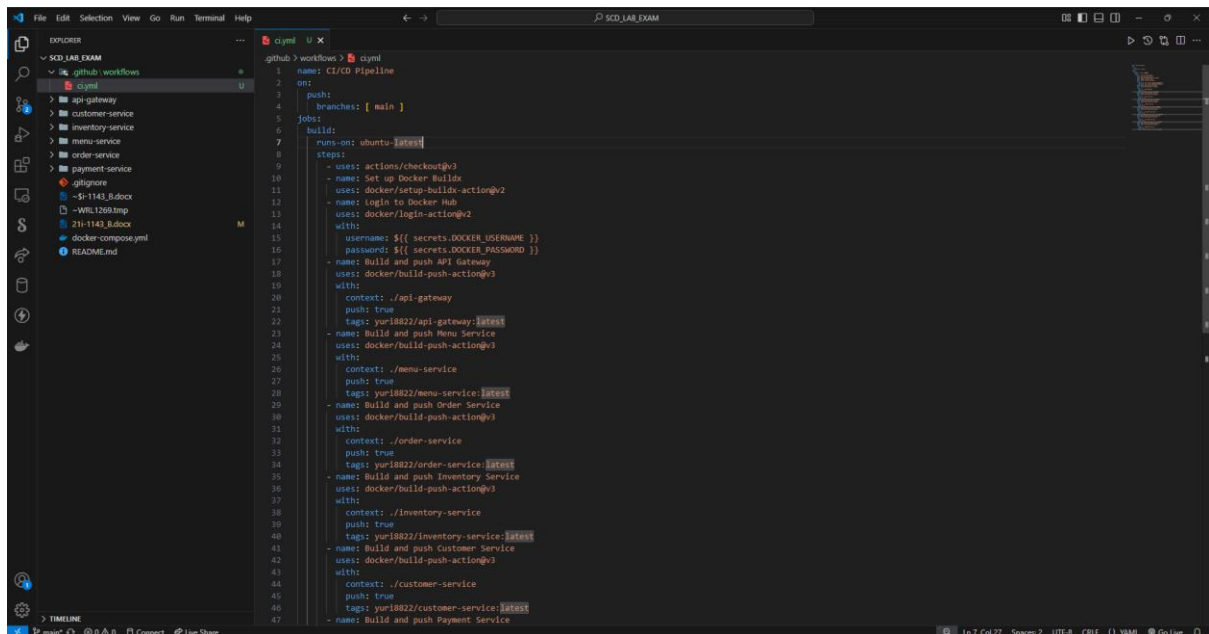
```
payment-service > index.js > @app.post('/payments') callback
1 const express = require('express');
2 const axios = require('axios');
3
4 const app = express();
5 app.use(express.json());
6
7 // I will be simulating the payment processing
8 app.post('/payments', async (req, res) => {
9   const { orderId, amount } = req.body;
10   try {
11     // Verifying order total
12     const orderRes = await axios.get('http://order-service:3002/orders/${orderId}');
13     if (!orderRes.data || orderRes.data.total !== amount) {
14       return res.status(400).json({ error: 'Invalid order or amount' });
15     }
16
17     // Simulating payment gateway integration
18     const paymentStatus = 'success';
19     if (paymentStatus !== 'success') {
20       return res.status(400).json({ error: 'Payment failed' });
21     }
22
23     res.json({ message: 'Payment processed', orderId, amount });
24   } catch (err) {
25     res.status(500).json({ error: 'Failed to process payment' });
26   }
27 });
28
29 app.get('/orders/:id', async (req, res) => {
30   try {
31     const order = await axios.get('http://order-service:3002/orders/${req.params.id}');
32     res.json(order.data);
33   } catch (err) {
34     res.status(500).json({ error: 'Failed to fetch order' });
35   }
36 });
37
38 app.listen(3005, () => {
39   console.log('Payment Service running on port 3005');
40 });
```

Set up the docker-compose file to tie everything together:



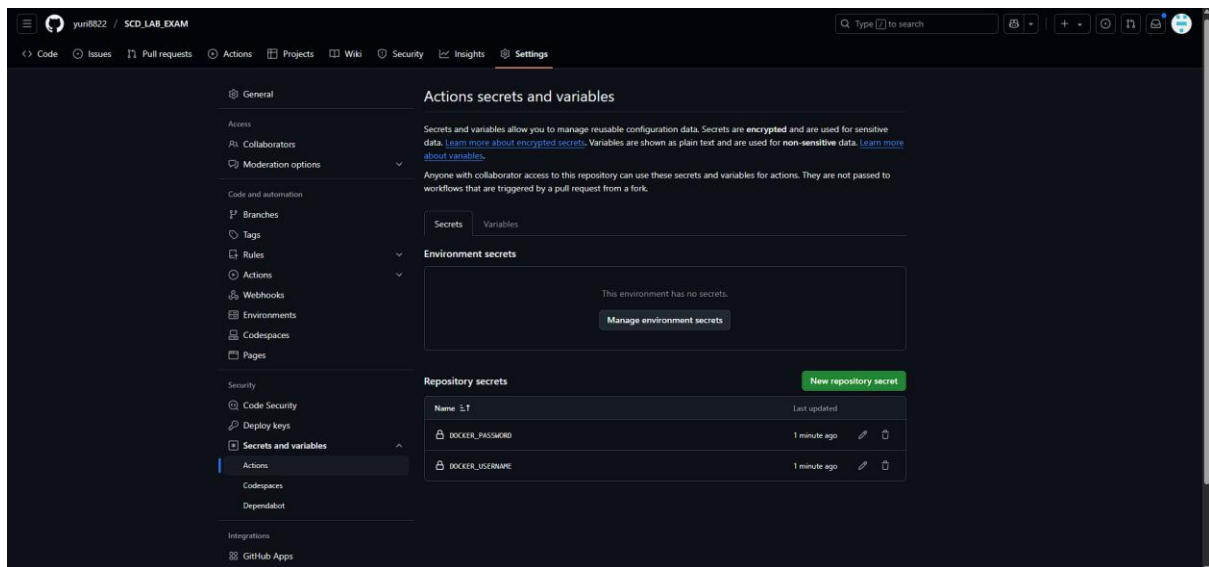
```
1 version: '3'
2
3 services:
4   # Run all Services
5   api-gateway:
6     build: ./api-gateway
7     ports:
8       - "3000:3000"
9     depends_on:
10      - menu-service
11      - order-service
12      - inventory-service
13      - customer-service
14      - payment-service
15
16   # Run Service
17   menu-service:
18     build: ./menu-service
19     ports:
20       - "3001:3001"
21     depends_on:
22      - mongo
23
24   # Run Service
25   order-service:
26     build: ./order-service
27     ports:
28       - "3002:3002"
29     depends_on:
30      - mongo
31      - menu-service
32      - inventory-service
33      - customer-service
34
35   # Run Service
36   inventory-service:
37     build: ./inventory-service
38     ports:
39       - "3003:3003"
40     depends_on:
41      - mongo
42
43   # Run Service
44   customer-service:
45     build: ./customer-service
46     ports:
47       - "3004:3004"
48     depends_on:
49      - mongo
50
51   # Run Service
52   payment-service:
53     build: ./payment-service
```

Set up the ci.yml file for github actions:

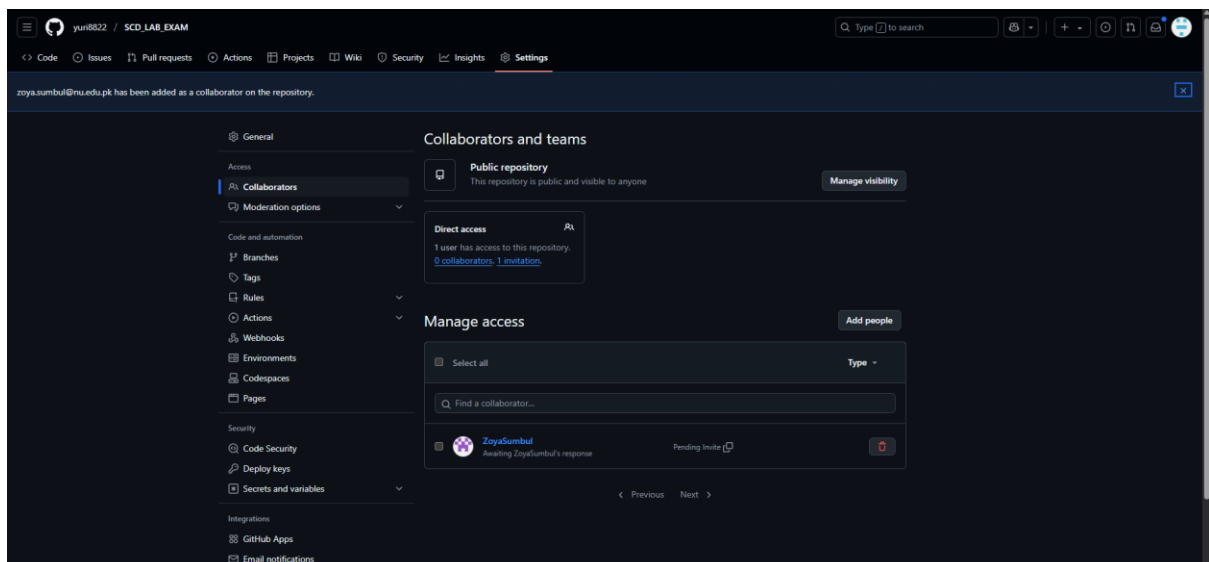


```
1 name: CI/CD Pipeline
2 on:
3   push:
4     branches: [ main ]
5 jobs:
6   build:
7     runs-on: ubuntu-latest
8     steps:
9       - uses: actions/checkout@v3
10       - name: Set up Docker Builds
11         uses: docker/setup-buildx-action@v2
12       - name: Login to Docker Hub
13         uses: docker/login-action@v2
14         with:
15           username: ${{ secrets.DOCKER_USERNAME }}
16           password: ${{ secrets.DOCKER_PASSWORD }}
17       - name: Build and push API Gateway
18         uses: docker/build-push-action@v3
19         with:
20           context: ./api-gateway
21           push: true
22           tags: yur18822/api-gateway:latest
23       - name: Build and push Menu Service
24         uses: docker/build-push-action@v3
25         with:
26           context: ./menu-service
27           push: true
28           tags: yur18822/menu-service:latest
29       - name: Build and push Order Service
30         uses: docker/build-push-action@v3
31         with:
32           context: ./order-service
33           push: true
34           tags: yur18822/order-service:latest
35       - name: Build and push Inventory Service
36         uses: docker/build-push-action@v3
37         with:
38           context: ./inventory-service
39           push: true
40           tags: yur18822/inventory-service:latest
41       - name: Build and push Customer Service
42         uses: docker/build-push-action@v3
43         with:
44           context: ./customer-service
45           push: true
46           tags: yur18822/customer-service:latest
47       - name: Build and push Payment Service
```

Added secrets to github actions:



Added collaborator:



Building:

The screenshot shows a Visual Studio Code editor with a GitHub Actions workflow file named `ci.yml` open. The workflow is configured for a CI/CD pipeline on the `main` branch, running on `ubuntu-latest`. It includes steps for checking out code, setting up Docker Buildx, logging into Docker Hub, and building and pushing Docker images for three services: `api-gateway`, `menu-service`, and `order-service`. The workflow uses `docker/build-push-action@v3` for building and pushing images.

```
github > workflows > ci.yml
1 name: CI/CD Pipeline
2 on:
3   push:
4     branches: [ main ]
5 jobs:
6   build:
7     runs-on: ubuntu-latest
8     steps:
9       - uses: actions/checkout@v3
10      - name: Set up Docker Buildx
11        uses: docker/setup-buildx-action@v2
12      - name: Login to Docker Hub
13        uses: docker/login-action@v2
14        with:
15          username: ${{ secrets.DOCKER_USERNAME }}
16          password: ${{ secrets.DOCKER_PASSWORD }}
17      - name: Build and push API Gateway
18        uses: docker/build-push-action@v3
19        with:
20          context: ./api-gateway
21          push: true
22          tags: yur18822/api-gateway:latest
23      - name: Build and push Menu Service
24        uses: docker/build-push-action@v3
25        with:
26          context: ./menu-service
27          push: true
28          tags: yur18822/menu-service:latest
29      - name: Build and push Order Service
30        uses: docker/build-push-action@v3
31        with:
32          context: ./order-service
33          push: true
```

The left sidebar shows the **GRAPH** view, indicating the workflow steps and their dependencies. The bottom panel shows the **TERMINAL** output, displaying the build progress and image sizes for each service.

Building 17.0s (11/22)

- sha256:513d779256048c96229af5f506803305460b72775217272e19ffac1635e08e 24.12MB / 191.90MB
- extracting sha256:311da6c45ea1576925308ea391bc53dece9be95960a6c0ffcd2576712017 4.7%
- sha256:ca26f68102188b67f57b74753ab8ca5db8d545bbae304d7ce9f42ddccid 1.09MB / 2.27MB
- [inventory-service:internal] load build context
- [menu-service:internal] load build context
- [customer-service:internal] load build context
- transferring context: 2.15s
- transferring context: 2.25s

The status bar at the bottom indicates the file is `ci.yml` in the `main` branch, with 11 lines, 44 columns, 2 spaces, UTF-8 encoding, and CRLF line endings.