Python Mock Services Migration

Overview

This document outlines the migration of Node.js mock services to Python using FastAPI.

Services Migrated

1. CMS Mock Service (Customer Management System)

- **Port**: 3001
- **Location**: `services/mocks/cms-mock-python/`
- **Features**:
  - Customer CRUD operations
  - Customer status management (active, inactive, pending)
  - Email validation
  - Mock data initialization

2. ROS Mock Service (Route Optimization System)

- **Port**: 3002
- **Location**: `services/mocks/ros-mock-python/`
- **Features**:
  - Route CRUD operations
  - Route optimization algorithm
  - Distance and duration calculation
  - Route status tracking (planned, in_progress, completed, cancelled)
  - Multiple stop support

3. WMS Mock Service (Warehouse Management System)

- **Port**: 3003
- **Location**: `services/mocks/wms-mock-python/`
- **Features**:
  - Inventory CRUD operations
  - SKU-based lookup
  - Stock level checking
  - Reorder level management
  - Warehouse location tracking
  - Inventory status management (available, reserved, out_of_stock, damaged)

Technology Stack

- **Framework**: FastAPI 0.109.0
- **Server**: Uvicorn with auto-reload
- **Validation**: Pydantic 2.5.3
- **Configuration**: Pydantic Settings

- **CORS**: FastAPI CORS Middleware

Project Structure

Each service follows the same structure:

```
service-name-python/
├── app.py                  # Main application entry point
├── requirements.txt        # Python dependencies
├── Dockerfile             # Docker configuration
├── README.md              # Service documentation
├── .gitignore            # Git ignore patterns
└── src/
    ├── __init__.py
    ├── config/
    │   ├── __init__.py
    │   └── settings.py    # Configuration management
    ├── models/
    │   ├── __init__.py
    │   └── schemas.py     # Pydantic models
    ├── routes/
    │   ├── __init__.py
    │   └── *_routes.py    # API endpoints
    ├── services/          # Business logic (CMS, ROS)
    │   ├── __init__.py
    │   └── *_service.py
    ├── handlers/          # Business logic (WMS)
    │   ├── __init__.py
    │   └── *_handlers.py
    └── utils/             # Helper functions (ROS)
        ├── __init__.py
        └── helpers.py
```

Setup Instructions

Option 1: Using Setup Script (Recommended)

```
# Run the setup script
./scripts/setup-python-mocks.sh

# Start all services
./scripts/start-python-mocks.sh

# Stop all services
./scripts/stop-python-mocks.sh
```

Option 2: Manual Setup

For each service:

```bash
cd services/mocks/<service-name>-python

# Create virtual environment
python3 -m venv venv

# Activate virtual environment
source venv/bin/activate

# Install dependencies
pip install -r requirements.txt

# Run the service
python app.py

# Deactivate when done
deactivate
```

Option 3: Using Docker

For each service:

```bash
cd services/mocks/<service-name>-python

# Build the image
docker build -t <service-name>-python .

# Run the container
docker run -p <port>:<port> <service-name>-python
```

API Documentation

Each service provides automatic API documentation:

- **Swagger UI**: `http://localhost:<port>/docs`
- **ReDoc**: `http://localhost:<port>/redoc`

CMS Mock Service (Port 3001)

- GET `/api/customers` - List all customers
- GET `/api/customers/{id}` - Get customer by ID
- POST `/api/customers` - Create customer
- PUT `/api/customers/{id}` - Update customer
- DELETE `/api/customers/{id}` - Delete customer

ROS Mock Service (Port 3002)

- GET `/api/routes` - List all routes
- GET `/api/routes/{id}` - Get route by ID

- POST `/api/routes` - Create route
- PUT `/api/routes/{id}` - Update route
- DELETE `/api/routes/{id}` - Delete route
- POST `/api/routes/{id}/optimize` - Optimize route

WMS Mock Service (Port 3003)

- GET `/api/inventory` - List all inventory items
- GET `/api/inventory/{id}` - Get item by ID
- GET `/api/inventory/sku/{sku}` - Get item by SKU
- POST `/api/inventory` - Create inventory item
- PUT `/api/inventory/{id}` - Update inventory item
- DELETE `/api/inventory/{id}` - Delete inventory item
- GET `/api/inventory/check-stock/{sku}` - Check stock level

Testing

Health Check

```
# CMS Mock
curl http://localhost:3001/health

# ROS Mock
curl http://localhost:3002/health

# WMS Mock
curl http://localhost:3003/health
```

Sample API Calls

Create Customer (CMS)

```
curl -X POST http://localhost:3001/api/customers \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Test Customer",
    "email": "test@example.com",
    "phone": "+1-555-0100",
    "address": "123 Test St",
    "company": "Test Corp"
  }'
```

Create Route (ROS)

```
curl -X POST http://localhost:3002/api/routes \
  -H "Content-Type: application/json" \
  -d '{
    "origin": "New York, NY",
    "destination": "Boston, MA",
    "vehicle_id": "VEH-001",
    "driver_id": "DRV-001"
  }'
```

Create Inventory Item (WMS)

```
curl -X POST http://localhost:3003/api/inventory \
  -H "Content-Type: application/json" \
  -d '{
    "sku": "PROD-999",
    "name": "Test Product",
    "quantity": 100,
    "unit_price": 49.99,
    "reorder_level": 20
  }'
```

Key Features

1. Type Safety with Pydantic

All data models use Pydantic for automatic validation and serialization.

2. Automatic API Documentation

FastAPI generates interactive API documentation automatically.

3. CORS Support

All services have CORS configured to accept requests from any origin.

4. In-Memory Storage

Data is stored in memory with pre-populated mock data for testing.

5. Error Handling

Proper HTTP status codes and error messages for all operations.

6. Configuration Management

Environment-based configuration using Pydantic Settings.

Migration Notes

Node.js → Python Equivalents

Node.js	Python
Express.js	FastAPI
`package.json`	`requirements.txt`
`index.js`	`app.py`
JavaScript modules	Python packages
npm/yarn	pip
node_modules/	venv/

Advantages of Python Implementation

1. **Type Hints**: Better IDE support and code clarity
2. **Automatic Validation**: Pydantic validates all input/output
3. **Auto-generated Docs**: Swagger UI and ReDoc out of the box
4. **Async Support**: Native async/await with FastAPI
5. **Data Models**: Clean separation with Pydantic models
6. **Less Boilerplate**: More concise code

Troubleshooting

Port Already in Use

```
# Find process using port
lsof -i :<port>

# Kill the process
kill -9 <PID>
```

Virtual Environment Issues

```
# Remove and recreate venv
rm -rf venv
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

Module Import Errors

Make sure you're running from the service root directory and the virtual environment is activated.

Next Steps

1. Update the main `docker-compose.yml` to include Python services
2. Update adapters to connect to Python mock services
3. Create integration tests for Python services
4. Add logging and monitoring
5. Implement persistent storage (optional)

Support

For issues or questions:

- Check service logs
- Review API documentation at `/docs`
- Ensure all dependencies are installed
- Verify ports are not in use