CMS Mock Service - Extended Documentation

Customer Management System Mock Service - Now with Drivers, Clients, and Admins
Port: **3001**
Technology: **Python FastAPI**
Storage: **File-based JSON**

📋 Table of Contents

- Overview
- Entity Types
- API Endpoints
- Data Models
- Quick Start Examples
- Advanced Usage
- File Storage

Overview

The CMS Mock Service now manages **four types of entities**:

1. **Customers** - Organizations or individuals using the logistics service
2. **Drivers** - Delivery personnel with vehicle assignments
3. **Clients** - Business clients with membership tiers
4. **Admins** - System administrators with role-based permissions

Key Features

✅ Full CRUD operations for all entity types
✅ Status/role-based filtering
✅ Thread-safe file storage
✅ Automatic data persistence
✅ Input validation with Pydantic
✅ Auto-generated Swagger documentation
✅ RESTful API design

Entity Types

1. Customers

Organizations or individuals who use the logistics service.

Fields:

- `id` - Unique identifier (UUID)

- `name` - Customer name
- `email` - Email address (validated)
- `phone` - Phone number (optional)
- `address` - Physical address (optional)
- `company` - Company name (optional)
- `status` - active | inactive | pending
- `created_at` - Creation timestamp
- `updated_at` - Last update timestamp

2. Drivers

Delivery personnel assigned to vehicles.

Fields:

- `id` - Unique identifier (UUID)
- `name` - Driver name
- `email` - Email address (validated)
- `phone` - Phone number (required)
- `license_number` - Driver's license number (required)
- `vehicle_id` - Assigned vehicle ID (optional)
- `status` - available | on_duty | off_duty | inactive
- `created_at` - Creation timestamp
- `updated_at` - Last update timestamp

3. Clients

Business clients with membership levels.

Fields:

- `id` - Unique identifier (UUID)
- `name` - Client/company name
- `email` - Email address (validated)
- `phone` - Phone number (optional)
- `address` - Physical address (optional)
- `membership_level` - basic | silver | gold | platinum
- `created_at` - Creation timestamp
- `updated_at` - Last update timestamp

4. Admins

System administrators with role-based access.

Fields:

- `id` - Unique identifier (UUID)
- `name` - Admin name
- `email` - Email address (validated)
- `phone` - Phone number (optional)

- `role` - super_admin | admin | moderator | support
- `permissions` - Array of permission strings
- `created_at` - Creation timestamp
- `updated_at` - Last update timestamp

API Endpoints

Health Check

```
GET /health
```

Returns service status and entity counts.

Response:

```json
{
  "status": "healthy",
  "service": "CMS Mock Service",
  "entities": {
    "customers": 3,
    "drivers": 4,
    "clients": 5,
    "admins": 3
  }
}
```

Customers API

Base Path: `/customers`

Method	Endpoint	Description
GET	`/customers/`	Get all customers
GET	`/customers/{id}`	Get customer by ID
POST	`/customers/`	Create new customer
PUT	`/customers/{id}`	Update customer
DELETE	`/customers/{id}`	Delete customer

Drivers API

Base Path: `/drivers`

Method	Endpoint	Description
GET	`/drivers/`	Get all drivers
GET	`/drivers/?status={status}`	Filter drivers by status
GET	`/drivers/{id}`	Get driver by ID
POST	`/drivers/`	Create new driver
PUT	`/drivers/{id}`	Update driver
DELETE	`/drivers/{id}`	Delete driver

Status Filter Values: `available`, `on_duty`, `off_duty`, `inactive`

Clients API

Base Path: `/clients`

Method	Endpoint	Description
GET	`/clients/`	Get all clients
GET	`/clients/?membership_level={level}`	Filter by membership
GET	`/clients/{id}`	Get client by ID
POST	`/clients/`	Create new client
PUT	`/clients/{id}`	Update client
DELETE	`/clients/{id}`	Delete client

Membership Levels: `basic`, `silver`, `gold`, `platinum`

Admins API

Base Path: `/admins`

Method	Endpoint	Description
GET	`/admins/`	Get all admins
GET	`/admins/?role={role}`	Filter admins by role
GET	`/admins/{id}`	Get admin by ID
POST	`/admins/`	Create new admin
PUT	`/admins/{id}`	Update admin
DELETE	`/admins/{id}`	Delete admin

Admin Roles: `super_admin`, `admin`, `moderator`, `support`

Quick Start Examples

1. Get All Drivers

```
curl http://localhost:3001/drivers/
```

2. Create a New Driver

```
curl -X POST http://localhost:3001/drivers/ \
  -H "Content-Type: application/json" \
  -d '{
    "name": "John Driver",
    "email": "john@swiftlogistics.com",
    "phone": "+1-555-0100",
    "license_number": "DL12345678"
  }'
```

3. Filter Drivers by Status

```
# Get all available drivers
curl "http://localhost:3001/drivers/?status=available"

# Get all drivers on duty
curl "http://localhost:3001/drivers/?status=on_duty"
```

4. Create a New Client

```
curl -X POST http://localhost:3001/clients/ \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Tech Corp",
    "email": "contact@techcorp.com",
    "phone": "+1-555-0200",
    "address": "123 Business St, San Francisco, CA",
    "membership_level": "gold"
  }'
```

5. Update a Driver Status

```
curl -X PUT http://localhost:3001/drivers/{driver_id} \
  -H "Content-Type: application/json" \
  -d '{
    "status": "on_duty",
    "vehicle_id": "VH-123"
  }'
```

6. Filter Clients by Membership

```
# Get all platinum clients
curl "http://localhost:3001/clients/?membership_level=platinum"
```

7. Create an Admin User

```
curl -X POST http://localhost:3001/admins/ \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Support Agent",
    "email": "support@swiftlogistics.com",
    "phone": "+1-555-0300",
    "role": "support",
    "permissions": ["tickets.read", "tickets.write", "customers.read"]
  }'
```

Advanced Usage

Python Example

```python
import requests

BASE_URL = "http://localhost:3001"

# Get all available drivers
response = requests.get(f"{BASE_URL}/drivers/", params={"status":
"available"})
available_drivers = response.json()

# Assign a driver to a delivery
if available_drivers:
    driver_id = available_drivers[0]["id"]
    update_data = {
        "status": "on_duty",
        "vehicle_id": "VH-001"
    }
    requests.put(f"{BASE_URL}/drivers/{driver_id}", json=update_data)

# Create a new client
new_client = {
    "name": "Startup Inc",
    "email": "hello@startup.com",
    "membership_level": "silver"
}
response = requests.post(f"{BASE_URL}/clients/", json=new_client)
client = response.json()
print(f"Created client: {client['id']}")
```

JavaScript Example

```
const BASE_URL = 'http://localhost:3001';

// Get all platinum clients
async function getPlatinumClients() {
  const response = await fetch(`${BASE_URL}/clients/?
membership_level=platinum`);
  const clients = await response.json();
  return clients;
}

// Create a new driver
async function createDriver(driverData) {
  const response = await fetch(`${BASE_URL}/drivers/`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(driverData)
  });
  return await response.json();
}

// Usage
const newDriver = {
  name: 'Alice Driver',
  email: 'alice@swiftlogistics.com',
  phone: '+1-555-0150',
  license_number: 'DL99887766'
};

createDriver(newDriver).then(driver => {
  console.log('Created driver:', driver.id);
});
```

File Storage

Each entity type is stored in a separate JSON file:

```
services/mocks/cms-mock/data/
├── customers.json    # Customer records
├── drivers.json      # Driver records
├── clients.json      # Client records
└── admins.json       # Admin records
```

Data Persistence

- All changes are **immediately written** to disk
- **Thread-safe** file operations using locks
- **Atomic writes** prevent data corruption
- **JSON format** for easy inspection and backup

Example File Structure

drivers.json:

```json
{
  "uuid-1": {
    "id": "uuid-1",
    "name": "Mike Wilson",
    "email": "mike@swiftlogistics.com",
    "phone": "+1-555-0201",
    "license_number": "DL123456789",
    "vehicle_id": "VH-001",
    "status": "available",
    "created_at": "2024-01-15T10:00:00",
    "updated_at": "2024-01-15T10:00:00"
  }
}
```

Swagger Documentation

Interactive API documentation is available at:

http://localhost:3001/docs

Features:

- Try out API endpoints directly
- View request/response schemas
- See all available filters and parameters
- Test authentication flows

Error Responses

All endpoints follow consistent error handling:

404 Not Found

```
{
  "detail": "Driver with id {id} not found"
}
```

422 Validation Error

```
{
  "detail": [
    {
      "loc": ["body", "email"],
      "msg": "value is not a valid email address",
      "type": "value_error.email"
    }
  ]
}
```

Architecture

```
cms-mock/
├── app.py                      # FastAPI application
├── data/                       # JSON data files
│   ├── customers.json
│   ├── drivers.json
│   ├── clients.json
│   └── admins.json
└── src/
    ├── config/
    │   └── settings.py         # Configuration
    ├── models/
    │   └── schemas.py          # Pydantic models
    ├── routes/
    │   ├── cms_routes.py       # Customer endpoints
    │   ├── driver_routes.py    # Driver endpoints
    │   ├── client_routes.py    # Client endpoints
    │   └── admin_routes.py     # Admin endpoints
    ├── services/
    │   ├── cms_service.py      # Customer business logic
    │   ├── driver_service.py   # Driver business logic
    │   ├── client_service.py   # Client business logic
    │   └── admin_service.py    # Admin business logic
    └── utils/
        └── file_storage.py     # File storage utility
```

Summary

The **extended CMS Mock Service** now provides comprehensive management for:

✅ **Customers** - End users and organizations
✅ **Drivers** - Delivery personnel with vehicle tracking
✅ **Clients** - Business clients with membership tiers
✅ **Admins** - System administrators with role-based access

All entities support:

- Full CRUD operations
- Filtered queries
- Status/role management
- Thread-safe persistence
- RESTful API design

Next Steps:

- Explore the Swagger UI at http://localhost:3001/docs
- Try the example requests above
- Integrate with your application
- Check the data files in `data/` directory