ODOO_HACK Backend - Master Data API

Overview

This is a FastAPI-based backend implementation for master data management in an accounting system. It provides REST APIs for managing Contacts, Products, Taxes, Chart of Accounts, and HSN code lookup.

Features

- **Async SQLAlchemy 2.0** with PostgreSQL
- **Tenant-aware** multi-tenant architecture
- **JWT Authentication** integration
- **HSN API Proxy** with fallback to local data
- **Comprehensive Testing** with pytest
- **Type Hints** and validation with Pydantic

Setup Instructions

1. Install Dependencies

```
cd backend
uv sync
```

2. Environment Configuration

Copy the example environment file and configure it:

```
cp .env.example .env
# Edit .env with your database and API credentials
```

3. Database Setup

Ensure PostgreSQL is running (via Docker Compose):

```
# From project root
docker-compose up -d db
```

4. Run the Application

```
uv run uvicorn main:app --reload --host 0.0.0.0 --port 8000
```

5. Database Migrations

After adding models, run Alembic revision:

```
# Initialize Alembic (first time only)
uv run alembic init alembic

# Generate migration
uv run alembic revision --autogenerate -m "Add master data models"

# Apply migration
uv run alembic upgrade head
```

API Endpoints

Authentication

- `POST /api/v1/register` - User registration
- `POST /api/v1/login` - User authentication

Master Data

- **Contacts**: `/api/v1/masters/contacts`
- **Products**: `/api/v1/masters/products`
- **Taxes**: `/api/v1/masters/taxes`
- **Chart of Accounts**: `/api/v1/masters/accounts`
- **HSN Search**: `/api/v1/masters/hsn`

Testing

Run Unit Tests

```
uv run pytest -q
```

Run Tests with Coverage

```
uv run pytest --cov=. --cov-report=html
```

Run Specific Test File

```
uv run pytest tests/test_masters_api.py -v
```

Manual Testing with curl

1. Authentication (Setup)

If your authentication system is already configured, you'll need a JWT token. For testing, the API currently uses a dummy authentication that works without a real token.

2. Create a Contact

```
curl -X POST "http://localhost:8000/api/v1/masters/contacts" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "ABC Corporation",
    "email": "contact@abc-corp.com",
    "phone": "+91-9876543210",
    "address": "123 Business Park, Mumbai",
    "gstin": "27AABCU9603R1ZM",
    "contact_type": "customer"
  }'
```

3. List Contacts

```
curl -X GET "http://localhost:8000/api/v1/masters/contacts?
page=1&per_page=10"
```

4. Create a Product

```
curl -X POST "http://localhost:8000/api/v1/masters/products" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Office Chair Executive",
    "sku": "CHAIR-EXE-001",
    "description": "Ergonomic executive office chair with lumbar
support",
    "unit_price": 15999.00,
    "hsn_code": "94036000",
    "unit_of_measurement": "pcs"
  }'
```

5. Search Products

```
# Search by name
curl -X GET "http://localhost:8000/api/v1/masters/products?
search=chair&page=1&per_page=10"

# Search by HSN code
curl -X GET "http://localhost:8000/api/v1/masters/products?
search=94036000"

# Search by SKU
curl -X GET "http://localhost:8000/api/v1/masters/products?search=CHAIR-
EXE-001"
```

6. Create a Tax

```
curl -X POST "http://localhost:8000/api/v1/masters/taxes" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "CGST 9%",
    "tax_type": "CGST",
    "rate": 9.0000,
    "description": "Central GST 9% rate"
  }'
```

7. HSN Code Search

```
# Search for furniture HSN codes
curl -X GET "http://localhost:8000/api/v1/masters/hsn?q=furniture"

# Search for table HSN codes
curl -X GET "http://localhost:8000/api/v1/masters/hsn?q=table"

# Search by HSN code
curl -X GET "http://localhost:8000/api/v1/masters/hsn?q=94036000"
```

8. Update Operations

```
# Update a contact (replace {contact_id} with actual ID)
curl -X PUT "http://localhost:8000/api/v1/masters/contacts/{contact_id}"
\
  -H "Content-Type: application/json" \
  -d '{
    "phone": "+91-9876543211",
    "address": "456 New Business Park, Mumbai"
  }'
```

```
# Update a product (replace {product_id} with actual ID)
curl -X PUT "http://localhost:8000/api/v1/masters/products/{product_id}"
\
  -H "Content-Type: application/json" \
  -d '{
    "unit_price": 16999.00,
    "description": "Updated description with new features"
  }'
```

HTTPie Commands (Alternative)

If you prefer HTTPie over curl:

```
# Install HTTPie
pip install httpx[cli]

# Create contact
http POST localhost:8000/api/v1/masters/contacts \
  name="XYZ Ltd" \
  email="info@xyz.com" \
  contact_type="vendor"

# Search products
http GET localhost:8000/api/v1/masters/products search=="office chair"

# HSN search
http GET localhost:8000/api/v1/masters/hsn q=="computer"
```

Authentication Integration

The current implementation includes a temporary authentication system for testing. To integrate with your actual authentication:

1. **Replace the `get_current_user` dependency** in `api/masters.py`
2. **Update the `CurrentUser` class** to match your user model
3. **Ensure your user object has a `tenant_id` attribute**

Example integration:

```
from your_auth_module import get_current_user as actual_get_current_user

# Replace the dummy dependency
async def get_current_user() -> YourUserModel:
    return await actual_get_current_user()
```

Tenant ID Usage

All database operations are scoped to the current user's tenant using `current_user.tenant_id`. This ensures data isolation between different tenants in a multi-tenant environment.

HSN API Configuration

The HSN search endpoint supports external API integration:

1. **Set `HSN_API_URL`** in your environment
2. **Optionally set `HSN_API_KEY`** for authentication
3. **If external API fails**, the system automatically falls back to built-in HSN data

Error Handling

The API returns standard HTTP status codes:

- `200` - Success
- `201` - Created
- `400` - Bad Request (validation errors)
- `404` - Resource not found
- `422` - Unprocessable Entity (Pydantic validation)
- `500` - Internal Server Error

Development Notes

File Structure

```
backend/
├── models/masters.py       # SQLAlchemy models
├── schemas/masters.py      # Pydantic schemas
├── crud/masters.py         # Database operations
├── api/masters.py          # FastAPI routes
├── services/hsn_api_proxy.py # HSN API integration
├── tests/test_masters_api.py # Test suite
├── database.py             # Database setup
├── config.py               # Configuration
└── main.py                 # Application entry point
```

Key Technologies

- **FastAPI** - Modern async web framework
- **SQLAlchemy 2.0** - Async ORM
- **Pydantic** - Data validation and serialization
- **asyncpg** - Async PostgreSQL adapter
- **httpx** - Async HTTP client
- **pytest-asyncio** - Async testing

Performance Considerations

- Uses connection pooling for database
- Async operations throughout
- Pagination for large result sets
- Efficient database queries with proper indexing

This completes the Master Data Backend implementation for your ODOO_HACK project!