Zoho ServiceDesk API - Usage Guide

Quick Start

1. Installation

```
# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements_updated.txt
```

2. Configuration

Update the configuration in the `Config` class with your Zoho credentials:

```
self.ZOHO_CONFIG = {
    "CLIENT_ID": "your_client_id",
    "CLIENT_SECRET": "your_client_secret",
    "REFRESH_TOKEN": "your_refresh_token",
    "ACCOUNTS_URL": "https://accounts.zoho.com",
    "API_BASE_URL": "your_servicedesk_url"
}
```

3. Running the Application

```
python refactored_zoho_ticket_api.py
```

API Endpoints

Create Ticket

POST `/create_ticket`

Headers:

```
Content-Type: application/json
```

Request Body:

```
{
  "subject": "Ticket Subject",
  "description": "Detailed description of the issue",
  "requester_email": "user@example.com",
  "template": {
    "name": "Template Name"
  },
  "urgency": {
    "name": "High"
  },
  "category": {
    "name": "Category Name"
  },
  "subcategory": {
    "name": "Subcategory Name"
  },
  "item": {
    "name": "Item Name"
  },
  "udf_fields": {
    "udf_char1": "Custom Field Value"
  }
}
```

Response (Success):

```
{
  "message": "Ticket created successfully in Zoho Service Desk",
  "zoho_ticket_id": "123456789"
}
```

Response (Error):

```
{
  "error": "Error message description"
}
```

Health Check

GET `/health`

Response:

```
{
  "status": "healthy",
  "timestamp": "2024-01-01T12:00:00.000Z"
}
```

Design Patterns Used

1. Singleton Pattern

- **Purpose**: Ensure only one configuration instance
- **Implementation**: `Config` class
- **Benefits**: Consistent configuration across the application

2. Strategy Pattern

- **Purpose**: Allow different implementations of token management and API clients
- **Implementation**: `TokenManager` and `ApiClient` abstract base classes
- **Benefits**: Easy to swap implementations and add new providers

3. Factory Pattern

- **Purpose**: Create configured service instances
- **Implementation**: `ServiceFactory` and `FlaskAppFactory`
- **Benefits**: Centralized object creation and configuration

4. Dependency Injection

- **Purpose**: Reduce coupling between components
- **Implementation**: Services receive dependencies through constructors
- **Benefits**: Easier testing and better maintainability

Architecture Layers

1. Controller Layer (`TicketController`)

- Handles HTTP requests and responses
- Validates input data format
- Returns appropriate HTTP status codes

2. Service Layer (`TicketService`)

- Contains business logic
- Validates business rules
- Orchestrates operations between different components

3. API Client Layer (`ZohoServiceDeskClient`)

- Handles external API communication
- Manages request/response formatting
- Implements retry logic for failed requests

4. Token Management Layer (`ZohoTokenManager`)

- Manages OAuth2 token lifecycle
- Handles token refresh automatically
- Provides token expiry tracking

5. Configuration Layer (`Config`)

- Centralizes application configuration
- Provides singleton access to settings
- Easy to modify for different environments

Error Handling

The application implements comprehensive error handling:

1. **Validation Errors**: Missing or invalid request data
2. **Authentication Errors**: Token expiry or invalid credentials
3. **API Errors**: External service failures
4. **Network Errors**: Connection issues or timeouts

Logging

The application uses Python's built-in logging module:

```
import logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
```

Testing

Unit Testing Example

```
import unittest
from unittest.mock import Mock, patch
from refactored_zoho_ticket_api import TicketService, TicketRequest

class TestTicketService(unittest.TestCase):
```

```
    def setUp(self):
        self.mock_api_client = Mock()
        self.ticket_service = TicketService(self.mock_api_client)

    def test_create_ticket_success(self):
        # Test implementation
        pass
```

Integration Testing

```
# Run with pytest
pip install pytest
pytest tests/
```

Production Deployment

1. Environment Variables

Create a `.env` file:

```
ZOHO_CLIENT_ID=your_client_id
ZOHO_CLIENT_SECRET=your_client_secret
ZOHO_REFRESH_TOKEN=your_refresh_token
ZOHO_API_BASE_URL=your_api_url
FLASK_ENV=production
```

2. Using Gunicorn

```
gunicorn -w 4 -b 0.0.0.0:5000 "refactored_zoho_ticket_api:create_app()"
```

3. Docker Deployment

```
FROM python:3.9-slim

WORKDIR /app
COPY requirements_updated.txt .
RUN pip install -r requirements_updated.txt

COPY . .
EXPOSE 5000

CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:5000", "refactored_zoho_ticket_api:create_app(
```

Security Considerations

1. **Environment Variables**: Store sensitive data in environment variables
2. **HTTPS**: Always use HTTPS in production
3. **Rate Limiting**: Implement rate limiting to prevent abuse
4. **Input Validation**: Validate all input data
5. **Logging**: Don't log sensitive information

Performance Optimization

1. **Connection Pooling**: Use requests session for connection reuse
2. **Caching**: Cache frequently accessed data
3. **Async Operations**: Consider async operations for I/O bound tasks
4. **Monitoring**: Implement application performance monitoring

Troubleshooting

Common Issues:

1. **Token Refresh Fails**
   - Check client credentials
   - Verify refresh token validity
   - Check network connectivity
2. **API Requests Fail**
   - Verify API URL
   - Check request format
   - Review API documentation
3. **Internal Server Errors**
   - Check application logs
   - Verify configuration
   - Test with minimal request data

Debug Mode

```
app.run(debug=True)  # Only for development
```

Contributing

1. Follow PEP 8 style guidelines
2. Add type hints for all functions
3. Write comprehensive tests
4. Update documentation
5. Use meaningful commit messages

Support

For issues and questions:

1. Check the error logs
2. Review the API documentation
3. Test with the provided examples
4. Check network connectivity and credentials