🌟 FastAPI Metadata

🎯 Overview

FastAPI provides powerful built-in documentation features that automatically generate interactive API documentation. This comprehensive guide covers how to enhance your API documentation with metadata, custom descriptions, status codes, tags, and advanced features that make your API professional and developer-friendly.

📚 Part 1: Understanding FastAPI Documentation System

🔧 How FastAPI Documentation Works

```
from fastapi import FastAPI, status, HTTPException, Query, Path, Body
from fastapi.responses import JSONResponse
from typing import Optional, List, Dict, Any
from pydantic import BaseModel, Field
from enum import Enum
import uvicorn

# 🎨 Create FastAPI app with comprehensive metadata
app = FastAPI(
    title="🚀 Advanced API Documentation Demo",
    description="""
    ## 🌟 Welcome to Advanced FastAPI Documentation!

    This API demonstrates comprehensive documentation features including:

    * 📝 **Rich Descriptions**: Markdown-formatted endpoint descriptions
    * 🏷️ **Organized Tags**: Grouped endpoints for better navigation
    * ✅ **Status Codes**: Proper HTTP status code handling
    * 📊 **Response Models**: Structured response documentation
    * 🛡️ **Error Handling**: Comprehensive error response documentation

    ### 🔗 Quick Links
    - 📚 [API Documentation](/docs)
    - 📋 [ReDoc Documentation](/redoc)
    - 🔧 [OpenAPI Schema](/openapi.json)
    """,
    version="2.0.0",
    terms_of_service="https://example.com/terms/",
    contact={
        "name": "API Support Team",
        "url": "https://example.com/contact/",
        "email": "support@example.com",
    },
    license_info={
```

```
        "name": "MIT License",
        "url": "https://opensource.org/licenses/MIT",
    },
    # 🏷️ Global tag definitions for better organization
    openapi_tags=[
        {
            "name": "👤 Users",
            "description": "**User management operations**. Handle user creation,
authentication, and profile management.",
            "externalDocs": {
                "description": "User Management Guide",
                "url": "https://example.com/docs/users/",
            },
        },
        {
            "name": "📦 Products",
            "description": "**Product catalog operations**. Manage products,
categories, and inventory.",
        },
        {
            "name": "🛒 Orders",
            "description": "**Order processing**. Handle order creation, payment,
and fulfillment.",
        },
        {
            "name": "🔧 System",
            "description": "**System utilities**. Health checks, monitoring, and
administrative functions.",
        },
    ]
)
```

📝 Part 2: Operation Descriptions & Metadata

🎯 Basic Metadata Enhancement

```
# 📦 Pydantic models for structured responses
class User(BaseModel):
    """👤 User model with comprehensive validation"""
    id: int = Field(..., description="🆔 Unique user identifier", example=123)
    username: str = Field(..., min_length=3, max_length=50, description="👤 Unique
username", example="john_doe")
    email: str = Field(..., description="📧 User email address",
example="john@example.com")
    full_name: Optional[str] = Field(None, description="🧑 User's full name",
example="John Doe")
    is_active: bool = Field(True, description="✅ Whether the user account is
active")
    created_at: str = Field(..., description="📅 Account creation timestamp",
```

```python
example="2024-01-01T00:00:00Z")

class UserCreate(BaseModel):
    """📝 User creation model"""
    username: str = Field(..., min_length=3, max_length=50, example="new_user")
    email: str = Field(..., example="newuser@example.com")
    password: str = Field(..., min_length=8, example="secure_password123")
    full_name: Optional[str] = Field(None, example="New User")

class APIResponse(BaseModel):
    """📊 Standardized API response wrapper"""
    status: str = Field(..., description="🔄 Response status", example="success")
    message: str = Field(..., description="📝 Human-readable message",
example="Operation completed successfully")
    data: Optional[Any] = Field(None, description="📊 Response data")
    meta: Optional[Dict[str, Any]] = Field(None, description="🔍 Additional
metadata")

# ✅ Basic endpoint with enhanced documentation
@app.get(
    "/users/{user_id}",
    summary="🔍 Retrieve User by ID",
    description="""
    ## 📖 Get User Information

    Retrieves detailed information about a specific user by their unique identifier.

    ### 🎯 Use Cases:
    - 👤 **Profile Display**: Show user profile information
    - 🔍 **User Lookup**: Administrative user search
    - 📊 **Data Validation**: Verify user existence

    ### ⚡ Performance Notes:
    - Response time: ~50ms
    - Cached for 5 minutes
    - Rate limit: 100 requests/minute

    ### 🛡️ Security:
    - Requires valid authentication token
    - Users can only access their own data (unless admin)
    - Sensitive fields are automatically filtered
    """,
    response_description="✅ User information retrieved successfully",
    response_model=APIResponse,
    tags=["👤 Users"]
)
async def get_user(
    user_id: int = Path(
        ...,
        ge=1,
        le=999999,
```

```
        title="User ID",
        description="🆔 Unique identifier for the user (1-999999)",
        example=123
    )
):
    """
    🔍 Retrieve user by ID with comprehensive error handling

    Args:
        user_id (int): User identifier within valid range

    Returns:
        APIResponse: Standardized response with user data

    Raises:
        HTTPException: 404 if user not found, 403 if unauthorized
    """

    # 🎲 Mock user data (replace with database query)
    if user_id == 404:  # Simulate not found
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="👤 User not found. Please check the user ID and try again."
        )

    mock_user = User(
        id=user_id,
        username=f"user_{user_id}",
        email=f"user{user_id}@example.com",
        full_name=f"User {user_id}",
        is_active=True,
        created_at="2024-01-01T00:00:00Z"
    )

    return APIResponse(
        status="success",
        message="User information retrieved successfully",
        data=mock_user.dict(),
        meta={
            "request_id": f"req_{user_id}",
            "api_version": "2.0.0",
            "cached": True,
            "cache_expires": "2024-01-01T00:05:00Z"
        }
    )
```

🎨 Advanced Description with Markdown

```
@app.post(
    "/users/",
    summary="👤 Create New User Account",
    description="""
    ## 🚀 User Registration Endpoint

    Creates a new user account with comprehensive validation and security features.

    ### 📋 Required Information:
    | Field | Type | Validation | Description |
    |-------|------|------------|-------------|
    | `username` | string | 3-50 chars, alphanumeric | 👤 Unique username |
    | `email` | string | Valid email format | 📧 User email address |
    | `password` | string | Min 8 chars, mixed case | 🔒 Secure password |
    | `full_name` | string | Optional | 🧑‍💼 Display name |

    ### ✅ Validation Rules:
    - **Username**: Must be unique, alphanumeric with underscores
    - **Email**: Must be valid format and not already registered
    - **Password**: Minimum 8 characters with mixed case and numbers
    - **Rate Limiting**: Max 3 registration attempts per IP per hour

    ### 🔄 Process Flow:
    1. **Validate Input** → Check all required fields and formats
    2. **Check Uniqueness** → Verify username/email not already taken
    3. **Hash Password** → Secure password storage with bcrypt
    4. **Create Account** → Insert user record into database
    5. **Send Welcome Email** → Automated welcome message
    6. **Return Response** → User data with generated ID

    ### 🛡️ Security Features:
    - 🔒 Password hashing with salt
    - 📧 Email verification required
    - 🚫 Automatic spam detection
    - 📊 Account creation monitoring

    ### 📈 Success Response:
    Returns the created user object with a unique ID and account status.

    ### ⚠️ Possible Errors:
    - `400`: Invalid input data or validation errors
    - `409`: Username or email already exists
    - `429`: Too many registration attempts
    - `500`: Server error during account creation
    """,
    response_description="🎉 User account created successfully with generated ID and
welcome email sent",
    status_code=status.HTTP_201_CREATED,
    response_model=APIResponse,
    tags=["👤 Users"],
```

```
    # 📊 Additional response documentation
    responses={
        201: {
            "description": "✅ User created successfully",
            "content": {
                "application/json": {
                    "example": {
                        "status": "success",
                        "message": "User account created successfully",
                        "data": {
                            "id": 123,
                            "username": "new_user",
                            "email": "newuser@example.com",
                            "full_name": "New User",
                            "is_active": True,
                            "created_at": "2024-01-01T00:00:00Z"
                        },
                        "meta": {
                            "welcome_email_sent": True,
                            "verification_required": True
                        }
                    }
                }
            }
        },
        400: {
            "description": "❌ Validation error",
            "content": {
                "application/json": {
                    "example": {
                        "detail": [
                            {
                                "loc": ["body", "email"],
                                "msg": "Invalid email format",
                                "type": "value_error.email"
                            }
                        ]
                    }
                }
            }
        },
        409: {
            "description": "⚠️ Conflict - Username or email already exists",
            "content": {
                "application/json": {
                    "example": {
                        "status": "error",
                        "message": "Username 'john_doe' is already taken",
                        "error_code": "USERNAME_EXISTS"
                    }
                }
```

```
            }
        }
    }
)
async def create_user(user_data: UserCreate = Body(..., example={
    "username": "awesome_user",
    "email": "awesome@example.com",
    "password": "SecurePass123!",
    "full_name": "Awesome User"
})):
    """
    👤 Create new user with comprehensive validation

    Args:
        user_data (UserCreate): User registration information

    Returns:
        APIResponse: Created user data with metadata

    Raises:
        HTTPException: Various errors based on validation and conflicts
    """

    # 🛡️ Simulate validation checks
    if user_data.username == "admin":
        raise HTTPException(
            status_code=status.HTTP_409_CONFLICT,
            detail="⚠️ Username 'admin' is reserved and cannot be used"
        )

    if "@test.com" in user_data.email:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="❌ Test email domains are not allowed for registration"
        )

    # 🎲 Create mock user (replace with database operation)
    new_user = User(
        id=12345,  # Generated ID
        username=user_data.username,
        email=user_data.email,
        full_name=user_data.full_name,
        is_active=True,
        created_at="2024-01-01T00:00:00Z"
    )

    return APIResponse(
        status="success",
        message="🎉 User account created successfully! Welcome email has been
sent.",
        data=new_user.dict(),
```

```
        meta={
            "welcome_email_sent": True,
            "verification_required": True,
            "account_type": "standard",
            "creation_ip": "127.0.0.1"
        }
    )
```

🔢 Part 3: Status Codes & Error Handling

✅ Comprehensive Status Code Implementation

```
from fastapi import status

# 📊 Product models
class Product(BaseModel):
    """📦 Product model"""
    id: int = Field(..., description="🆔 Product ID", example=1)
    name: str = Field(..., description="📦 Product name", example="Wireless
Headphones")
    description: str = Field(..., description="📝 Product description",
example="High-quality wireless headphones")
    price: float = Field(..., ge=0, description="💰 Product price", example=99.99)
    category: str = Field(..., description="🏷️ Product category",
example="Electronics")
    in_stock: bool = Field(..., description="📦 Stock availability", example=True)
    stock_quantity: int = Field(..., ge=0, description="📊 Available quantity",
example=50)

# ✅ GET with 200 OK (default)
@app.get(
    "/products/",
    summary="📦 List All Products",
    description="""
    ## 🛍️ Product Catalog

    Retrieves a paginated list of all available products with filtering options.

    ### 🔍 Features:
    - 📄 **Pagination**: Control page size and offset
    - 🔍 **Search**: Text search across product names and descriptions
    - 🏷️ **Category Filter**: Filter by product category
    - 💰 **Price Range**: Filter by minimum and maximum price
    - 📦 **Stock Filter**: Show only in-stock items

    ### 📊 Response Format:
    Returns a standardized response with product array and pagination metadata.
    """,
    response_description="📦 Product list retrieved successfully with pagination
```

```
info",
    response_model=APIResponse,
    tags=["📦 Products"]
)
async def list_products(
    skip: int = Query(0, ge=0, description="📄 Number of products to skip"),
    limit: int = Query(10, ge=1, le=100, description="🔢 Maximum products to
return"),
    search: Optional[str] = Query(None, min_length=2, description="🔍 Search term"),
    category: Optional[str] = Query(None, description="🏷️ Filter by category")
):
    # Mock products data
    mock_products = [
        Product(id=1, name="📱 iPhone 15", description="Latest smartphone",
price=999.99, category="Electronics", in_stock=True, stock_quantity=25),
        Product(id=2, name="👕 Cotton T-Shirt", description="Comfortable cotton
tee", price=29.99, category="Clothing", in_stock=True, stock_quantity=100),
    ]

    return APIResponse(
        status="success",
        message=f"Retrieved {len(mock_products)} products",
        data={
            "products": [p.dict() for p in mock_products],
            "pagination": {
                "skip": skip,
                "limit": limit,
                "total": len(mock_products),
                "has_more": False
            }
        }
    )

# ✅ POST with 201 CREATED
@app.post(
    "/products/",
    summary="➕ Create New Product",
    description="""
    ## 🆕 Add Product to Catalog

    Creates a new product in the catalog with full validation and inventory
tracking.

    ### 📋 Validation Rules:
    - Product name must be unique
    - Price must be positive
    - Category must be from predefined list
    - Stock quantity cannot be negative
    """,
    response_description="🆕 Product created successfully and added to catalog",
    status_code=status.HTTP_201_CREATED,
```

```
    response_model=APIResponse,
    tags=["📦 Products"]
)
async def create_product(product: Product):
    """➕ Create new product with validation"""

    # Simulate product creation
    return APIResponse(
        status="success",
        message="🆕 Product created successfully!",
        data=product.dict(),
        meta={
            "created_at": "2024-01-01T00:00:00Z",
            "created_by": "admin",
            "inventory_updated": True
        }
    )

# ✅ PUT with 200 OK
@app.put(
    "/products/{product_id}",
    summary="🖍️ Update Product",
    description="""
    ## 🔄 Update Existing Product

    Updates an existing product with new information. All fields are optional - only
provided fields will be updated.

    ### 🎯 Update Capabilities:
    - 📝 **Product Details**: Name, description, category
    - 💰 **Pricing**: Update product price
    - 📦 **Inventory**: Modify stock quantities
    - 🔄 **Status**: Enable/disable product availability
    """,
    response_description="✅ Product updated successfully",
    response_model=APIResponse,
    tags=["📦 Products"]
)
async def update_product(
    product_id: int = Path(..., ge=1, description="🆔 Product ID to update"),
    product: Product = Body(..., description="📝 Updated product information")
):
    """🖍️ Update existing product"""

    if product_id == 999:  # Simulate not found
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail=f"📦 Product with ID {product_id} not found"
        )

    return APIResponse(
```

```
        status="success",
        message="✅ Product updated successfully",
        data=product.dict(),
        meta={
            "updated_at": "2024-01-01T00:00:00Z",
            "version": 2
        }
    )

# ✅ DELETE with 204 NO CONTENT
@app.delete(
    "/products/{product_id}",
    summary="🗑️ Delete Product",
    description="""
    ## 🗑️ Remove Product from Catalog

    Permanently removes a product from the catalog. This action cannot be undone.

    ### ⚠️ Important Notes:
    - Product must exist in the catalog
    - Cannot delete products with pending orders
    - Inventory will be automatically adjusted
    - All product reviews will be archived

    ### 🔄 Soft Delete Option:
    Consider using PUT to set `is_active: false` instead of permanent deletion.
    """,
    response_description="🗑️ Product deleted successfully from catalog",
    status_code=status.HTTP_204_NO_CONTENT,
    tags=["📦 Products"]
)
async def delete_product(
    product_id: int = Path(..., ge=1, description="🆔 Product ID to delete")
):
    """🗑️ Delete product from catalog"""

    if product_id == 999:  # Simulate not found
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail=f"📦 Product with ID {product_id} not found"
        )

    # Return None for 204 status
    return None
```

📊 Status Code Reference Table

```
# 📋 Create comprehensive status code examples
@app.get("/status-codes/examples", include_in_schema=False)  # Hidden endpoint for
```

```
reference
async def status_code_examples():
    """📚 Comprehensive status code reference"""
    return {
        "success_codes": {
            "200": "✅ OK - Standard successful response",
            "201": "🆕 Created - Resource successfully created",
            "204": "🗑️ No Content - Successful deletion/update with no response
body"
        },
        "client_error_codes": {
            "400": "❌ Bad Request - Invalid input or malformed request",
            "401": "🔐 Unauthorized - Authentication required or invalid",
            "403": "🚫 Forbidden - Valid auth but insufficient permissions",
            "404": "🔍 Not Found - Requested resource does not exist",
            "409": "⚠️ Conflict - Resource conflict (duplicate, etc.)",
            "422": "📝 Unprocessable Entity - Validation errors",
            "429": "⏱️ Too Many Requests - Rate limit exceeded"
        },
        "server_error_codes": {
            "500": "💥 Internal Server Error - Unexpected server issue",
            "502": "🌐 Bad Gateway - Upstream server error",
            "503": "🛠️ Service Unavailable - Server temporarily unavailable"
        }
    }
```

🏷️ Part 4: Tags & Organization

📂 Advanced Tag Management

```
# 🎨 Define comprehensive tag system
class TagCategories:
    """🏷️ Centralized tag definitions for API organization"""

    # 👤 User Management
    USERS = "👤 Users"
    AUTHENTICATION = "🔐 Authentication"
    PROFILES = "👨‍💼 User Profiles"

    # 📦 Product Management
    PRODUCTS = "📦 Products"
    CATEGORIES = "🏷️ Categories"
    INVENTORY = "📊 Inventory"

    # 🛒 Order Processing
    ORDERS = "🛒 Orders"
    PAYMENTS = "💳 Payments"
    SHIPPING = "📦 Shipping"
```

```
    # 🔧 System & Admin
    SYSTEM = "🔧 System"
    ADMIN = "👑 Admin"
    MONITORING = "📊 Monitoring"

# 🛒 Order-related endpoints with consistent tagging
@app.get(
    "/orders/",
    summary="📋 List User Orders",
    description="""
    ## 🛒 Order History

    Retrieves a paginated list of orders for the authenticated user.
    Includes comprehensive order information and status tracking.
    """,
    response_description="📋 Order list retrieved successfully",
    tags=[TagCategories.ORDERS]
)
async def list_orders():
    return APIResponse(
        status="success",
        message="Orders retrieved successfully",
        data={"orders": [], "total": 0}
    )

@app.post(
    "/orders/",
    summary="🛒 Create New Order",
    description="""
    ## 🆕 Place New Order

    Creates a new order with selected products and shipping information.
    Handles inventory validation and payment processing.
    """,
    response_description="🛒 Order created and payment initiated",
    status_code=status.HTTP_201_CREATED,
    tags=[TagCategories.ORDERS, TagCategories.PAYMENTS]
)
async def create_order():
    return APIResponse(
        status="success",
        message="Order created successfully"
    )

# 💳 Payment processing with multiple tags
@app.post(
    "/payments/process/",
    summary="💳 Process Payment",
    description="""
    ## 💰 Payment Processing
```

```
    Handles secure payment processing for orders using various payment methods.
    Supports credit cards, digital wallets, and bank transfers.
    """,
    response_description="💳 Payment processed successfully",
    tags=[TagCategories.PAYMENTS, TagCategories.ORDERS]
)
async def process_payment():
    return APIResponse(
        status="success",
        message="Payment processed successfully"
    )

# 🔧 System monitoring endpoints
@app.get(
    "/health/",
    summary="🏥 Health Check",
    description="""
    ## 🏥 System Health Status

    Provides comprehensive system health information including:
    - Database connectivity
    - External service status
    - System resource usage
    - Cache status
    """,
    response_description="🏥 System health status retrieved",
    tags=[TagCategories.SYSTEM, TagCategories.MONITORING]
)
async def health_check():
    return {
        "status": "✅ healthy",
        "timestamp": "2024-01-01T00:00:00Z",
        "services": {
            "database": "✅ connected",
            "cache": "✅ operational",
            "external_api": "✅ responsive"
        }
    }
```

🔄 Part 5: Response Models & Documentation Flow

📊 Advanced Response Documentation

```
from typing import Union

# 📊 Advanced response models with examples
class ErrorResponse(BaseModel):
    """❌ Error response model"""
    status: str = Field("error", description="🔄 Response status")
```

```
    message: str = Field(..., description="📝 Error message")
    error_code: Optional[str] = Field(None, description="🔍 Machine-readable error
code")
    details: Optional[Dict[str, Any]] = Field(None, description="📋 Additional error
details")
    timestamp: str = Field(..., description="⏰ Error timestamp")
    request_id: str = Field(..., description="🆔 Unique request identifier")

class PaginationMeta(BaseModel):
    """📄 Pagination metadata"""
    page: int = Field(..., description="📄 Current page number", example=1)
    per_page: int = Field(..., description="🔢 Items per page", example=10)
    total_items: int = Field(..., description="📊 Total available items",
example=250)
    total_pages: int = Field(..., description="📄 Total number of pages",
example=25)
    has_next: bool = Field(..., description="➡️ Whether next page exists",
example=True)
    has_previous: bool = Field(..., description="⬅️ Whether previous page exists",
example=False)

class ProductListResponse(BaseModel):
    """📦 Product list response with pagination"""
    status: str = Field("success", description="✅ Response status")
    message: str = Field(..., description="📝 Response message")
    data: Dict[str, Any] = Field(..., description="📊 Response data")
    pagination: PaginationMeta = Field(..., description="📄 Pagination information")

# 🎯 Comprehensive endpoint with multiple response models
@app.get(
    "/products/search/",
    summary="🔍 Advanced Product Search",
    description="""
    ## 🔍 Intelligent Product Search

    Advanced product search with multiple filters, sorting options, and intelligent
ranking.

    ### 🚀 Search Features:

    #### 🔍 **Search Capabilities**:
    - **Full-text search** across product names and descriptions
    - **Fuzzy matching** for typo tolerance
    - **Synonym support** for better results
    - **Auto-complete suggestions** for search terms

    #### 🏷️ **Filtering Options**:
    | Filter | Type | Description |
    |--------|------|-------------|
    | `category` | string | 🏷️ Product category filter |
    | `min_price` | number | 💰 Minimum price threshold |
```

```
    | `max_price` | number | 💰 Maximum price threshold |
    | `in_stock` | boolean | 📦 Stock availability filter |
    | `brand` | string | 🏢 Brand name filter |
    | `rating` | number | ⭐ Minimum rating filter |

    #### 📊 **Sorting Options**:
    - `relevance` - 🎯 Search relevance (default)
    - `price_asc` - 💰 Price low to high
    - `price_desc` - 💰 Price high to low
    - `rating` - ⭐ Customer rating
    - `newest` - 🆕 Recently added
    - `popular` - 🔥 Most popular

    ### 📈 **Performance Metrics**:
    - Average response time: ~120ms
    - Search index updated every 5 minutes
    - Results cached for 1 minute
    - Maximum 1000 results per query

    ### 🔄 **Response Format**:
    Returns paginated results with comprehensive product information and search
metadata.
    """,
    response_description="🔍 Search results with products and metadata",
    response_model=Union[ProductListResponse, ErrorResponse],
    responses={
        200: {
            "description": "✅ Search completed successfully",
            "model": ProductListResponse,
            "content": {
                "application/json": {
                    "example": {
                        "status": "success",
                        "message": "Found 25 products matching your search",
                        "data": {
                            "products": [
                                {
                                    "id": 1,
                                    "name": "📱 iPhone 15 Pro",
                                    "description": "Latest flagship smartphone",
                                    "price": 999.99,
                                    "category": "Electronics",
                                    "rating": 4.8,
                                    "reviews_count": 2847,
                                    "in_stock": True,
                                    "stock_quantity": 15
                                }
                            ],
                            "search_metadata": {
                                "query": "iphone",
                                "results_count": 25,
```

```
                                "search_time_ms": 87,
                                "suggestions": ["iphone case", "iphone charger"]
                            }
                        },
                        "pagination": {
                            "page": 1,
                            "per_page": 10,
                            "total_items": 25,
                            "total_pages": 3,
                            "has_next": True,
                            "has_previous": False
                        }
                    }
                }
            }
        },
        400: {
            "description": "❌ Invalid search parameters",
            "model": ErrorResponse,
            "content": {
                "application/json": {
                    "example": {
                        "status": "error",
                        "message": "Invalid search parameters provided",
                        "error_code": "INVALID_SEARCH_PARAMS",
                        "details": {
                            "min_price": "Must be greater than 0",
                            "max_price": "Must be greater than min_price"
                        },
                        "timestamp": "2024-01-01T00:00:00Z",
                        "request_id": "req_12345"
                    }
                }
            }
        },
        429: {
            "description": "⏱️ Too many search requests",
            "model": ErrorResponse
        }
    },
    tags=[TagCategories.PRODUCTS, "🔍 Search"]
)
async def advanced_product_search(
    q: str = Query(..., min_length=2, description="🔍 Search query", 
example="wireless headphones"),
    category: Optional[str] = Query(None, description="🏷️ Category filter", 
example="Electronics"),
    min_price: Optional[float] = Query(None, ge=0, description="💰 Minimum price", 
example=50.0),
    max_price: Optional[float] = Query(None, ge=0, description="💰 Maximum price", 
example=500.0),
```

```
    sort_by: str = Query("relevance", description="📊 Sort order",
example="price_asc"),
    page: int = Query(1, ge=1, description="📄 Page number", example=1),
    per_page: int = Query(10, ge=1, le=50, description="🔢 Items per page",
example=10)
):
    """🔍 Execute advanced product search with comprehensive filtering"""

    # Validate price range
    if min_price and max_price and min_price > max_price:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="❌ Minimum price cannot be greater than maximum price"
        )

    # Mock search results
    return ProductListResponse(
        status="success",
        message=f"🔍 Found products matching '{q}'",
        data={
            "products": [
                {
                    "id": 1,
                    "name": "🎧 Wireless Headphones Pro",
                    "description": "Premium noise-canceling headphones",
                    "price": 299.99,
                    "category": "Electronics",
                    "rating": 4.7,
                    "in_stock": True
                }
            ],
            "search_metadata": {
                "query": q,
                "results_count": 1,
                "search_time_ms": 95,
                "filters_applied": {
                    "category": category,
                    "price_range": f"{min_price}-{max_price}" if min_price or
max_price else None
                }
            }
        },
        pagination=PaginationMeta(
            page=page,
            per_page=per_page,
            total_items=1,
            total_pages=1,
            has_next=False,
            has_previous=False
```

```
        )
    )
```

🎯 Part 6: Testing Documentation Features

🧪 Interactive Documentation Testing

```
# 🧪 Test endpoint specifically for documentation features
@app.get(
    "/docs-demo/",
    summary="📚 Documentation Feature Demo",
    description="""
    ## 🎯 Documentation Features Showcase

    This endpoint demonstrates all FastAPI documentation features in action:

    ### 📝 **Supported Features**:

    #### ✅ **Basic Features**:
    - [x] Custom summary and description
    - [x] Response descriptions
    - [x] Status code documentation
    - [x] Tag organization
    - [x] Parameter validation

    #### 🚀 **Advanced Features**:
    - [x] Multiple response models
    - [x] Custom examples
    - [x] Markdown formatting
    - [x] External documentation links
    - [x] Deprecation warnings

    ### 🔗 **External Resources**:
    - [FastAPI Documentation](https://fastapi.tiangolo.com/)
    - [OpenAPI Specification](https://swagger.io/specification/)
    - [JSON Schema](https://json-schema.org/)

    ### ⚠️ **Important Notes**:
    > This is a demo endpoint showing documentation capabilities.
    > In production, ensure all endpoints have proper documentation.

    ### 📊 **Code Example**:
    ```
    @app.get("/example", summary="Example", description="Demo endpoint")
    async def example():
        return {"message": "Hello World"}
    ```
    """,
    response_description="📚 Documentation demo response with examples",
```

```
    response_model=APIResponse,
    tags=["📚 Documentation", TagCategories.SYSTEM],
    # 🔗 External documentation
    openapi_extra={
        "externalDocs": {
            "description": "FastAPI Documentation Guide",
            "url": "https://fastapi.tiangolo.com/tutorial/metadata/"
        }
    }
)
async def documentation_demo(
    demo_param: str = Query("example", description="📝 Demo parameter with example",
example="demo_value"),
    optional_param: Optional[int] = Query(None, ge=1, le=100, description="🔢
Optional number parameter")
):
    """📚 Demonstrate documentation features"""

    return APIResponse(
        status="success",
        message="📚 Documentation demo executed successfully",
        data={
            "demo_param": demo_param,
            "optional_param": optional_param,
            "features_demonstrated": [
                "✅ Rich markdown descriptions",
                "🏷️ Tag organization",
                "📊 Response models",
                "🔍 Parameter validation",
                "📝 Custom examples",
                "🔗 External documentation links"
            ]
        },
        meta={
            "documentation_version": "2.0.0",
            "openapi_version": "3.0.2"
        }
    )

# 🚀 Run the application
if __name__ == "__main__":
    uvicorn.run(
        "main:app",
        host="0.0.0.0",
        port=8000,
        reload=True,
        log_level="info"
    )
```

📊 Complete Features Summary

🎯 Documentation Enhancement Checklist

Feature	Usage	Benefits	Example
📝 **Summary**	`summary="Brief title"`	Concise endpoint titles	"Create User Account"
📖 **Description**	`description="""Markdown text"""`	Detailed explanations	Rich formatted docs
✅ **Response Description**	`response_description="Success message"`	Clear success outcomes	"User created successfully"
🔢 **Status Codes**	`status_code=status.HTTP_201_CREATED`	Proper HTTP semantics	201 for creation
🏷️ **Tags**	`tags=["Users", "Admin"]`	API organization	Grouped endpoints
📊 **Response Models**	`response_model=UserResponse`	Structured responses	Type safety
🔍 **Examples**	`example={"key": "value"}`	Clear usage patterns	Interactive docs
⚠️ **Error Responses**	`responses={400: {...}}`	Error documentation	Comprehensive error info

🚀 Best Practices Quick Reference

```
# ✅ GOOD: Comprehensive documentation
@app.post(
    "/users/",
    summary="👤 Create User",  # Clear, concise title
    description="""
    ## 🚀 User Registration

    Creates a new user account with validation.

    ### Requirements:
    - Unique username
    - Valid email format
    - Strong password (8+ chars)
    """,  # Rich markdown description
    response_description="✅ User created with ID",  # Clear success message
    status_code=status.HTTP_201_CREATED,  # Appropriate status
    response_model=APIResponse,  # Structured response
    tags=["👤 Users"]  # Logical grouping
)
async def create_user(user: UserCreate):
    """Docstring for IDE support"""
```

```
    pass

# ❌ BAD: Minimal documentation
@app.post("/users/")
async def create_user(user: UserCreate):
    pass
```

This comprehensive guide provides everything you need to create professional, well-documented FastAPI applications that are developer-friendly and easy to understand! 🎉