

```
<!DOCTYPE html>
<html>
<head>
<title>API_SPECIFICATION.md</title>
<meta http-equiv="Content-type" content="text/html; charset=UTF-8">
</head>
<!-- ここから追記 1.(数式に対応させる)-->
<script type="text/javascript" src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML"></script>
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({
    tex2jax:{inlineMath: [['$', '$']]},
    messageStyle: "none"
  });
</script>
<!--追記 1.ここまで-->
<body>
<script>
  mermaid.initialize({
    startOnLoad: true,
    theme: document.body.classList.contains('vscode-dark') || document.body.classList.contains('vscode-high-contrast')
      ? 'dark'
      : 'default'
  });
</script>
<!-- ここから追記 2.(--- で改ページ)-->
<style>
  hr {
    opacity: 0;
    break-after: page;
  }
</style>
<!--追記 2.ここまで-->
```

FastAPI アプリケーション 機能仕様書

目次

- 1. はじめに
 - 2. アーキテクチャ概要
 - 3. データベースモデル
 - 4. 認証・セキュリティ
 - 5. FastAPIエンドポイント
 - 6. MCP Tools（在庫最適化ツール）
 - 7. 環境変数
-

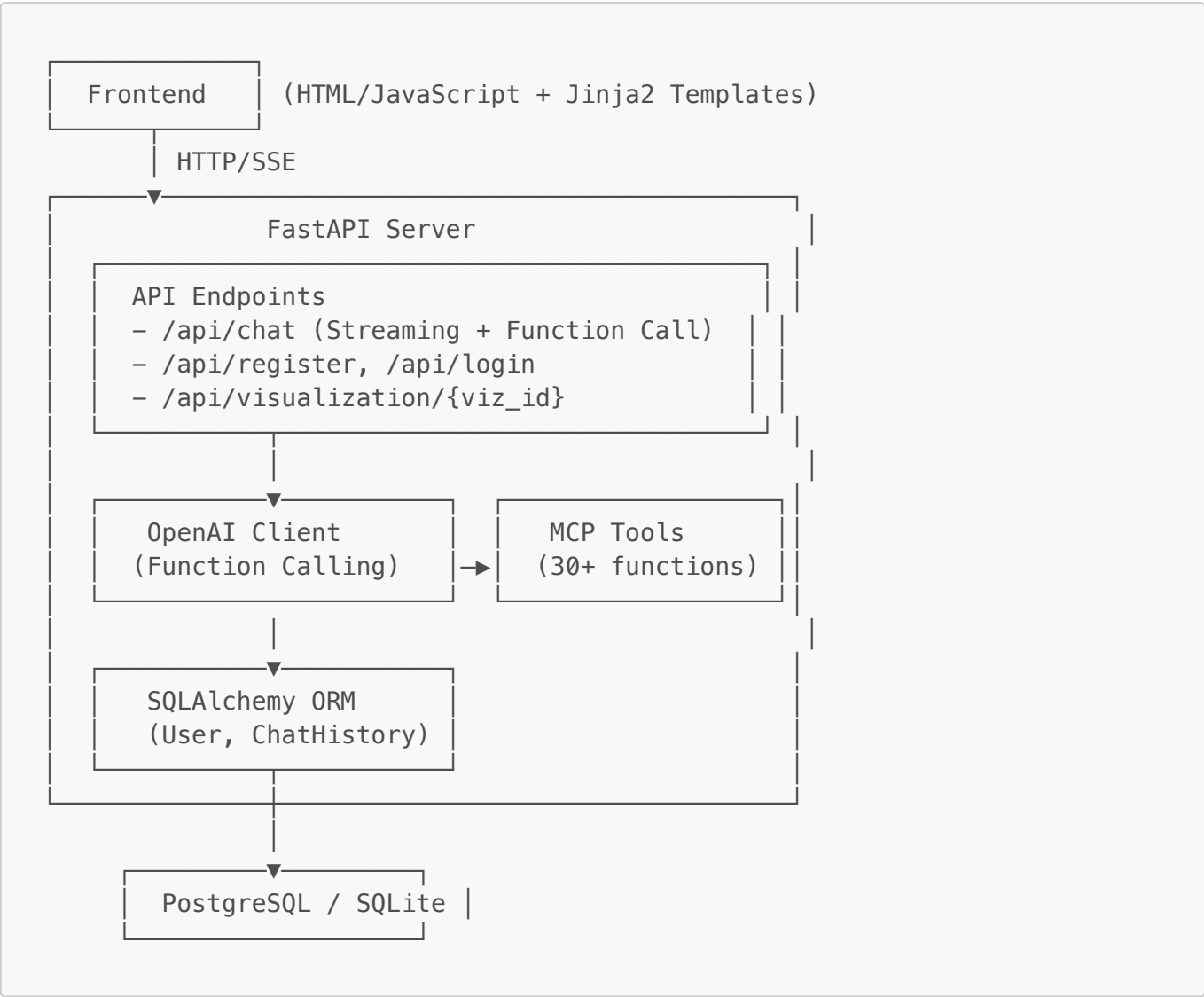
はじめに

本アプリケーションは、**在庫最適化専門のAIチャットボット**です。FastAPI + OpenAI Function Calling + MCP（Model Context Protocol）Toolsを組み合わせ、ユーザーと対話しながら在庫最適化計算を実行します。

主な機能

- ユーザー認証（JWT）
- チャット履歴の保存
- OpenAI Function Callingによる自動ツール呼び出し
- 30種類以上の在庫最適化機能（EOQ、安全在庫、定期発注、需要予測など）
- インタラクティブな可視化（Plotly）

アーキテクチャ概要



データベースモデル

User モデル

ファイル: `database.py:17-26`

フィールド	型	説明
id	Integer	プライマリキー
email	String	メールアドレス (ユニーク・インデックス)
username	String	ユーザー名 (ユニーク・インデックス)
hashed_password	String	bcryptハッシュ化パスワード
created_at	DateTime	作成日時 (UTC)
chat_histories	Relationship	ChatHistoryテーブルとのリレーション

リレーション: 1対多 (User → ChatHistory)

ChatHistory モデル

ファイル: database.py:28-37

フィールド	型	説明
id	Integer	プライマリキー
user_id	Integer	外部キー (users.id)
role	String	メッセージの役割 ("user" または "assistant")
content	Text	メッセージ本文
created_at	DateTime	作成日時 (UTC)
user	Relationship	Userテーブルとのリレーション

データベース関数

init_db()

ファイル: database.py:40-41

機能: データベーステーブルを作成 (マイグレーション)

入力: なし

出力: なし

使用例:

```
init_db() # アプリ起動時に実行
```

get_db()

ファイル: database.py:44-49

機能: データベースセッションを生成 (FastAPI Dependency)

入力: なし

出力: Generator[Session] - SQLAlchemyセッション

使用例:

```
@app.post("/api/example")
async def example(db: Session = Depends(get_db)):
    users = db.query(User).all()
    return {"users": users}
```

認証・セキュリティ

認証方式

- **JWT (JSON Web Token)** - HS256アルゴリズム
- **トークン有効期限:** 7日間
- **パスワードハッシュ:** bcrypt

認証関数

get_password_hash(password: str) -> str

ファイル: auth.py:22-23

機能: パスワードをbcryptでハッシュ化

入力:

- **password** (str): 平文パスワード

出力:

- **str:** bcryptハッシュ

使用例:

```
hashed = get_password_hash("mypassword123")
# "$2b$12$..."
```

verify_password(plain_password: str, hashed_password: str) -> bool

ファイル: auth.py:19-20

機能: パスワードを検証

入力:

- **plain_password** (str): 平文パスワード
- **hashed_password** (str): ハッシュ化パスワード

出力:

- **bool**: 一致したらTrue

使用例:

```
is_valid = verify_password("mypassword123", hashed)
# True
```

create_access_token(data: dict, expires_delta: Optional[timedelta] = None) -> str

ファイル: **auth.py**:25-33

機能: JWTアクセストークンを生成

入力:

- **data** (dict): ペイロードデータ (例: {"sub": "user_id"})
- **expires_delta** (Optional[timedelta]): 有効期限 (デフォルト: 7日間)

出力:

- **str**: JWT文字列

使用例:

```
token = create_access_token(data={"sub": "123"})
# "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
```

get_current_user(credentials: HTTPAuthorizationCredentials, db: Session) -> User

ファイル: **auth.py**:35-56

機能: JWTトークンからユーザーを取得 (認証必須)

入力:

- **credentials** (HTTPAuthorizationCredentials): Bearer トークン
- **db** (Session): データベースセッション

出力:

- **User**: 認証済みユーザー

例外:

- **HTTPException(401)**: トークン無効またはユーザーが存在しない

使用例:

```
@app.get("/api/protected")
async def protected(current_user: User = Depends(get_current_user)):
    return {"username": current_user.username}
```

```
get_current_user_optional(credentials: Optional[HTTPAuthorizationCredentials],
db: Session) -> Optional[User]
```

ファイル: auth.py:58-79

機能: JWTトークンからユーザーを取得（認証オプション）

入力:

- `credentials` (Optional[HTTPAuthorizationCredentials]): Bearerトークン（なくてもOK）
- `db` (Session): データベースセッション

出力:

- `Optional[User]`: 認証済みユーザー（トークンがない場合はNone）

使用例:

```
@app.post("/api/chat")
async def chat(current_user: Optional[User] =
Depends(get_current_user_optional)):
    if current_user:
        # ログイン済みユーザー: チャット履歴を保存
        pass
    else:
        # 未ログインユーザー: チャット履歴を保存しない
        pass
```

FastAPIエンドポイント

1. HTMLページ

GET /

ファイル: main.py:70-78

機能: ホームページを表示

認証: 不要

レスポンス: HTMLResponse

動作:

- ローカル環境 (`ENVIRONMENT=local`): チャット画面 (`index.html`)
- 本番環境 (`ENVIRONMENT=production`): ログイン画面 (`login.html`)

使用例:

```
curl http://localhost:8000/
```

GET /chat

ファイル: `main.py:80-83`

機能: チャットページを表示

認証: 不要

レスポンス: `HTMLResponse` (`index.html`)

使用例:

```
curl http://localhost:8000/chat
```

2. 設定API

GET /api/config

ファイル: `main.py:85-92`

機能: フロントエンド用の設定情報を取得

認証: 不要

入力: なし

出力:

```
{
  "model": "gpt-4o-mini",
  "environment": "local",
  "skip_auth": true
}
```

使用例:

```
curl http://localhost:8000/api/config
```

3. 認証API

POST /api/register

ファイル: `main.py`:94–118

機能: 新規ユーザーを登録

認証: 不要

入力:

```
{
  "email": "user@example.com",
  "username": "myusername",
  "password": "securepassword123"
}
```

出力:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "bearer"
}
```

エラー:

- 400: Email already registered
- 400: Username already taken

使用例:

```
curl -X POST http://localhost:8000/api/register \
  -H "Content-Type: application/json" \
  -d
'{"email":"test@example.com","username":"testuser","password":"test123"}'
```

POST /api/login

ファイル: `main.py`:120–131

機能: ユーザーログイン

認証: 不要

入力:

```
{
  "email": "user@example.com",
  "password": "securepassword123"
}
```

出力:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "bearer"
}
```

エラー:

- **401**: Incorrect email or password

使用例:

```
curl -X POST http://localhost:8000/api/login \
-H "Content-Type: application/json" \
-d '{"email":"test@example.com","password":"test123"}'
```

4. チャットAPI**POST /api/chat****ファイル:** main.py:133-310**機能:** AIチャット（ストリーミング + Function Calling対応）**認証:** オプション（ログイン済みの場合はチャット履歴を保存）**入力:**

```
{
  "messages": [
    {
      "role": "user",
      "content": "年間需要15000個、発注コスト500円、保管費率25%、  
単価12円でEOQを計算してください"
    }
  ],
  "model": "gpt-4o-mini"
}
```

出力: Server-Sent Events (text/event-stream)

```
data: {"function_call": {"name": "calculate_eoq_raw", "result": {...}}}  
  
data: {"content": "計算結果"}  
  
data: {"content": "は以下の通りです..."}  
  
data: [DONE]
```

処理フロー:

1. ユーザーメッセージをデータベースに保存（ログイン時のみ）
2. システムプロンプト（ツール使用指示）を追加
3. OpenAI API呼び出し（Function Calling有効）
4. Function callがある場合:
 - MCP関数を実行（**execute_mcp_function**）
 - 結果をストリーミング送信
 - 最終応答を生成（ストリーミング）
5. アシスタントメッセージをデータベースに保存（ログイン時のみ）

特徴:

- **ストリーミングレスポンス:** リアルタイムでテキストを表示
- **Function Calling:** LLMが自動的にツールを呼び出し
- **日本語対応:** システムプロンプトで日本語出力を強制
- **可視化対応:** グラフや図を自動生成

使用例:

```
curl -X POST http://localhost:8000/api/chat \  
-H "Content-Type: application/json" \  
-H "Authorization: Bearer YOUR_TOKEN" \  
-d '{"messages": [{"role": "user", "content": "EQQを計算してください"}]}'
```

5. 可視化API**GET** /api/visualization/{viz_id}**ファイル:** main.py:312–356**機能:** 可視化HTMLを取得**認証:** 不要（viz_idはUUIDで推測困難）**入力:**

- パスパラメータ: **viz_id** (str) - 可視化ID (UUID)

出力: HTMLResponse (Plotlyインタラクティブグラフ)

エラー:

- 404:** Visualization not found

処理フロー:

- ファイルシステム (**/tmp/visualizations/{viz_id}.html**) から検索
- ファイルが見つからない場合、メモリキャッシュから検索
- どちらにもない場合は404エラー

使用例:

```
curl http://localhost:8000/api/visualization/550e8400-e29b-41d4-a716-446655440000
```

6. ヘルスチェックAPI

GET /health

ファイル: **main.py:358-361**

機能: アプリケーションの稼働状態を確認

認証: 不要

入力: なし

出力:

```
{  
  "status": "ok"  
}
```

使用例:

```
curl http://localhost:8000/health
```

MCP Tools (在庫最適化ツール)

MCP Toolsは、OpenAI Function Callingを通じて自動的に呼び出される在庫最適化関数群です。

ツール一覧（32種類）

カテゴリ別分類

1. EOQ（経済発注量）計算

ツール名	機能	ファイル参照
calculate_eoq_raw	基本EOQ計算	mcp_tools.py:262
calculate_eoq_incremental_discount_raw	増分数量割引EOQ	mcp_tools.py:216
calculate_eoq_all_units_discount_raw	全単位数量割引EOQ	mcp_tools.py:170
visualize_eoq	EOQ可視化	mcp_tools.py:1153

2. 安全在庫計算

ツール名	機能	ファイル参照
calculate_safety_stock	単一品目安全在庫計算	mcp_tools.py:300
optimize_safety_stock_allocation	マルチエシュロン安全在庫最適化 (MESSA)	mcp_tools.py:338
visualize_safety_stock_network	安全在庫ネットワーク可視化	mcp_tools.py:1136

3. (Q,R)方策（定量発注方式）

ツール名	機能	ファイル参照
optimize_qr_policy	(Q,R)方策の最適化	mcp_tools.py:463
simulate_qr_policy	(Q,R)方策のシミュレーション	mcp_tools.py:410

4. (s,S)方策

ツール名	機能	ファイル参照
optimize_ss_policy	(s,S)方策の最適化	mcp_tools.py:561
simulate_ss_policy	(s,S)方策のシミュレーション	mcp_tools.py:508

5. 基在庫方策

ツール名	機能	ファイル参照
simulate_base_stock_policy	基在庫シミュレーション（需要配列指定）	mcp_tools.py:879

ツール名	機能	ファイル参照
base_stock_simulation_using_dist	基在庫シミュレーション（分布ベース）	mcp_tools.py:1276
calculate_base_stock_levels	基在庫レベル計算	mcp_tools.py:921
simulate_network_base_stock	ネットワーク基在庫シミュレーション	mcp_tools.py:1241

6. 定期発注方式

ツール名	機能	ファイル参照
optimize_periodic_inventory	定期発注最適化（Adam/Momentum/SGD）	mcp_tools.py:1031
optimize_periodic_with_one_cycle	Fit One Cycle学習率スケジューラ	mcp_tools.py:1196
find_optimal_learning_rate_periodic	最適学習率探索	mcp_tools.py:1165
visualize_periodic_optimization	定期発注最適化結果可視化	mcp_tools.py:1124

7. 需要分析・予測

ツール名	機能	ファイル参照
forecast_demand	需要予測（Holt-Winters）	mcp_tools.py:957
visualize_forecast	需要予測結果可視化	mcp_tools.py:1000
analyze_demand_pattern	需要パターン分析	mcp_tools.py:636
find_best_distribution	最適確率分布フィッティング	mcp_tools.py:794
visualize_demand_histogram	需要ヒストグラム可視化	mcp_tools.py:812

8. その他の在庫計算

ツール名	機能	ファイル参照
calculate_wagner_whitin	Wagner-Whitinアルゴリズム	mcp_tools.py:606
compare_inventory_policies	在庫方策比較	mcp_tools.py:654
analyze_inventory_network	在庫ネットワーク分析	mcp_tools.py:359

9. 可視化

ツール名	機能	ファイル参照
visualize_last_optimization	直前の最適化結果可視化	mcp_tools.py:380

ツール名	機能	ファイル参照
visualize_inventory_simulation	在庫シミュレーション可視化	mcp_tools.py:732
visualize_simulation_trajectories	シミュレーション軌道可視化	mcp_tools.py:1337
visualize_supply_chain_network	サプライチェーンネットワーク可視化	mcp_tools.py:1380
compare_inventory_costs_visual	在庫コスト比較可視化	mcp_tools.py:834

10. ユーティリティ

ツール名	機能	ファイル参照
generate_sample_data	サンプルデータ生成	mcp_tools.py:392

ツール詳細仕様

以下、主要なツールの詳細仕様の記載します。

calculate_eoq_raw

機能: 基本的な経済発注量（EOQ）を計算

入力パラメータ:

```
{
  "annual_demand": 15000,
  "order_cost": 500.0,
  "holding_cost_rate": 0.25,
  "unit_price": 12.0,
  "backorder_cost": 0.0,
  "visualize": false
}
```

パラメータ	型	必須	説明
annual_demand	integer	✓	年間需要量（units/年）
order_cost	number	✓	発注固定費用（円/回）
holding_cost_rate	number	✓	在庫保管費率（0.25 = 25%）
unit_price	number	✓	単価（円/unit）
backorder_cost	number	-	バックオーダーコスト（円/unit/日）
visualize	boolean	-	可視化するか（デフォルト: false）

出力:

```
{
  "success": true,
  "eoq_units": 1000,
  "eoq_days": 24.33,
  "total_cost": 3000.0,
  "order_cost_total": 1500.0,
  "holding_cost_total": 1500.0,
  "backorder_cost_total": 0.0,
  "annual_orders": 15,
  "parameters": {
    "d": 41.1,
    "K": 500.0,
    "h": 0.00822,
    "unit_price": 12.0
  },
  "visualization_id": "550e8400-e29b-41d4-a716-446655440000"
}
```

処理フロー:

- 1. 生パラメータを計算用パラメータに変換 (convert_eoq_params_from_raw)
- 2. EOQ計算 (calc_eoq_basic)
- 3. 可視化リクエストがある場合、グラフを生成
- 4. 結果を返す

実装: mcp_tools.py:1401 (execute_mcp_function内)

calculate_eoq_all_units_discount_raw

機能: 全単位数量割引を考慮したEOQ計算

入力パラメータ:

```
{
  "annual_demand": 15000,
  "order_cost": 500.0,
  "holding_cost_rate": 0.25,
  "price_table": [
    {"quantity": 0, "price": 15.0},
    {"quantity": 1000, "price": 12.0},
    {"quantity": 2000, "price": 10.0}
  ],
  "backorder_cost": 0.0,
  "visualize": false
}
```

パラメータ	型	必須	説明
-------	---	----	----

パラメータ	型	必須	説明
annual_demand	integer	✓	年間需要量 (units/年)
order_cost	number	✓	発注固定費用 (円/回)
holding_cost_rate	number	✓	在庫保管費率 (0.25 = 25%)
price_table	array	✓	単価テーブル [{quantity, price}, ...]
backorder_cost	number	-	バックオーダーコスト (円/unit/日)
visualize	boolean	-	可視化するか (デフォルト: false)

出力:

```
{
  "success": true,
  "eoq_units": 2000,
  "selected_price": 10.0,
  "total_cost": 152500.0,
  "procurement_cost": 150000.0,
  "order_cost_total": 1875.0,
  "holding_cost_total": 625.0,
  "discount_info": {
    "breakpoints": [0, 1000, 2000],
    "prices": [15.0, 12.0, 10.0]
  },
  "visualization_id": "550e8400-e29b-41d4-a716-446655440001"
}
```

実装: `mcp_tools.py:1401` (execute_mcp_function内)

calculate_safety_stock

機能: 単一品目の安全在庫レベルを計算

入力パラメータ:

```
{
  "mu": 100.0,
  "sigma": 20.0,
  "LT": 7,
  "b": 50.0,
  "h": 0.5,
  "fc": 10000.0
}
```

パラメータ	型	必須	説明
-------	---	----	----

パラメータ	型	必須	説明
mu	number	✓	平均需要量 (units/日)
sigma	number	✓	需要の標準偏差
LT	integer	✓	リードタイム (日)
b	number	✓	品切れ費用 (円/unit/日)
h	number	✓	在庫保管費用 (円/unit/日)
fc	number	-	発注固定費用 (円/回) デフォルト: 10000

出力:

```
{
  "success": true,
  "safety_stock": 52.96,
  "reorder_point": 752.96,
  "service_level": 0.95,
  "expected_demand_during_lt": 700.0,
  "std_demand_during_lt": 52.92,
  "parameters": {
    "mu": 100.0,
    "sigma": 20.0,
    "LT": 7,
    "b": 50.0,
    "h": 0.5
  }
}
```

実装: `mcp_tools.py:1401` (execute_mcp_function内)

optimize_safety_stock_allocation

機能: マルチエシュロン在庫ネットワーク全体での安全在庫配置を最適化 (MESSA: MEta Safety Stock Allocation)

入力パラメータ:

```
{
  "items_data": "[{\"name\":\"製品A\",\"process_time\":1,\"max_service_time\":2,\"avg_demand\":100,\"demand_std\":20,\"holding_cost\":0.5,\"stockout_cost\":50,\"fixed_cost\":1000}]",
  "bom_data": "[{\"child\":\"部品B\",\"parent\":\"製品A\",\"quantity\":2}]"
}
```

パラメータ	型	必須	説明
-------	---	----	----

パラメータ	型	必須	説明
items_data	string	✓	品目データのJSON配列文字列
bom_data	string	✓	BOM（部品表）データのJSON配列文字列

items_data構造:

```
[
  {
    "name": "製品A",
    "process_time": 1,
    "max_service_time": 2,
    "avg_demand": 100,
    "demand_std": 20,
    "holding_cost": 0.5,
    "stockout_cost": 50,
    "fixed_cost": 1000
  }
]
```

bom_data構造:

```
[
  {
    "child": "部品B",
    "parent": "製品A",
    "quantity": 2
  }
]
```

出力:

```
{
  "success": true,
  "optimal_safety_stocks": {
    "製品A": 45.2,
    "部品B": 90.4
  },
  "total_holding_cost": 67.8,
  "service_levels": {
    "製品A": 0.95,
    "部品B": 0.98
  },
  "network_info": {
    "num_nodes": 2,
    "num_edges": 1,
    "total_items": 2
  }
}
```

```
    }  
}
```

実装: `mcp_tools.py:1401` (`execute_mcp_function`内)

`optimize_qr_policy`

機能: (Q,R)方策（連続監視型・定量発注方式）の最適パラメータを計算

入力パラメータ:

```
{  
  "mu": 100.0,  
  "sigma": 20.0,  
  "lead_time": 7,  
  "holding_cost": 0.5,  
  "stockout_cost": 50.0,  
  "fixed_cost": 1000.0,  
  "n_samples": 10,  
  "n_periods": 100  
}
```

パラメータ	型	必須	説明
mu	number	✓	1日あたりの平均需要量（units/日）
sigma	number	✓	需要の標準偏差
lead_time	integer	✓	リードタイム（日）
holding_cost	number	✓	在庫保管費用（円/unit/日）
stockout_cost	number	✓	品切れ費用（円/unit）
fixed_cost	number	✓	固定発注費用（円/回）
n_samples	integer	-	シミュレーションサンプル数（デフォルト: 10）
n_periods	integer	-	シミュレーション期間（日）（デフォルト: 100）

出力:

```
{  
  "success": true,  
  "optimal_Q": 1000,  
  "optimal_R": 750,  
  "expected_cost": 15000,  
  "holding_cost": 7500,  
  "stockout_cost": 2500,  
  "order_cost": 5000,  
}
```

```
    "service_level": 0.95,
    "optimization_details": {
      "iterations": 50,
      "convergence": true
    }
  }
```

実装: `mcp_tools.py:1401` (`execute_mcp_function`内)

optimize_periodic_inventory

機能: 定期発注方式の最適化 (Adam/Momentum/SGD)

入力パラメータ:

```
{
  "items_data": "[{\n\"name\": \"Stage0\", \"demand_mean\": 10, \"demand_std\": 2, \"net_replenishment_time\": 3, \"h\": 0.5}]",
  "bom_data": "[]",
  "algorithm": "adam",
  "learning_rate": 0.01,
  "max_iterations": 100,
  "tolerance": 0.001,
  "beta1": 0.9,
  "beta2": 0.999,
  "backorder_cost": 100.0,
  "visualize": false
}
```

パラメータ	型	必須	説明
items_data	string	✓	段階データのJSON配列文字列
bom_data	string	✓	BOMデータのJSON配列文字列
algorithm	string	✓	"adam" / "momentum" / "sgd"
learning_rate	number	✓	学習率 (例: 0.01)
max_iterations	integer	-	最大反復回数 (デフォルト: 100)
tolerance	number	-	収束判定閾値 (デフォルト: 0.001)
beta1	number	-	Adam beta1 (デフォルト: 0.9)
beta2	number	-	Adam beta2 (デフォルト: 0.999)
backorder_cost	number	-	バックオーダーコスト (円/unit)
visualize	boolean	-	可視化するか (デフォルト: false)

出力:

```
{
  "success": true,
  "optimal_base_stock_levels": {
    "Stage0": 45.2,
    "Stage1": 60.8
  },
  "total_cost": 1250.0,
  "holding_cost": 800.0,
  "backorder_cost": 450.0,
  "convergence_info": {
    "iterations": 78,
    "converged": true,
    "final_gradient_norm": 0.0008
  },
  "visualization_id": "550e8400-e29b-41d4-a716-446655440002"
}
```

実装: `mcp_tools.py:1401` (`execute_mcp_function`内)

forecast_demand

機能: Holt-Winters法による需要予測

入力パラメータ:

```
{
  "historical_demand": [100, 105, 110, 108, 112, 115, 120],
  "forecast_periods": 7,
  "trend": "additive",
  "seasonal": "additive",
  "seasonal_periods": 7,
  "visualize": false
}
```

パラメータ	型	必須	説明
historical_demand	array	✓	過去需要データ [100, 105, ...]
forecast_periods	integer	✓	予測期間数
trend	string	-	"additive" / "multiplicative" / null
seasonal	string	-	"additive" / "multiplicative" / null
seasonal_periods	integer	-	季節周期（デフォルト: 7）
visualize	boolean	-	可視化するか（デフォルト: false）

出力:

```
{
  "success": true,
  "forecast": [122, 125, 128, 130, 133, 135, 138],
  "confidence_intervals": {
    "lower": [115, 117, 119, 121, 123, 125, 127],
    "upper": [129, 133, 137, 139, 143, 145, 149]
  },
  "trend_component": [0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8],
  "seasonal_component": [1.0, 1.05, 1.1, 1.0, 1.15, 1.2, 1.25],
  "model_params": {
    "alpha": 0.2,
    "beta": 0.1,
    "gamma": 0.05
  },
  "visualization_id": "550e8400-e29b-41d4-a716-446655440003"
}
```

実装: `mcp_tools.py:1401` (execute_mcp_function内)

visualize_last_optimization

機能: 直前に実行した安全在庫最適化結果を可視化

入力パラメータ: なし

出力:

```
{
  "success": true,
  "visualization_id": "550e8400-e29b-41d4-a716-446655440004",
  "url": "http://localhost:8000/api/visualization/550e8400-e29b-41d4-a716-446655440004"
}
```

注意:

- `optimize_safety_stock_allocation`を事前に実行している必要がある
- ユーザーごとにキャッシュが管理される

実装: `mcp_tools.py:1401` (execute_mcp_function内)

generate_sample_data

機能: サプライチェーンネットワーク最適化用のサンプルデータを生成

入力パラメータ:

```
{
  "complexity": "standard"
}
```

パラメータ	型	必須	説明
complexity	string	✓	"simple" (3品目) / "standard" (5品目) / "complex" (8品目)

出力:

```
{
  "success": true,
  "items_data": "[{\"name\":\"製品A\",...}]",
  "bom_data": "[{\"child\":\"部品B\", \"parent\":\"製品A\", \"quantity\":2}]",
  "description": "5品目の標準的なサプライチェーンネットワーク",
  "usage_example": "optimize_safety_stock_allocationに渡してください"
}
```

実装: `mcp_tools.py:1401` (execute_mcp_function内)

MCP Tools実装関数

`execute_mcp_function(function_name: str, arguments: dict, user_id: int = None) -> dict`

ファイル: `mcp_tools.py:1401`

機能: MCPツール関数を実行するディスパッチャー

入力:

- `function_name` (str): 関数名 (例: "calculate_eoq_raw")
- `arguments` (dict): 関数の引数
- `user_id` (int, optional): ユーザーID (キャッシュ管理用)

出力:

- `dict`: 実行結果

処理フロー:

- `function_name`に応じて対応する実装関数を呼び出し
- エラーハンドリング
- 結果をdict形式で返す
- 可視化結果はキャッシュに保存

使用例:

```
result = execute_mcp_function(
    "calculate_eoq_raw",
    {
        "annual_demand": 15000,
        "order_cost": 500.0,
        "holding_cost_rate": 0.25,
        "unit_price": 12.0
    },
    user_id=123
)
```

`get_visualization_html(user_id: int, viz_id: str = None) -> str`

ファイル: `mcp_tools.py`:144-161

機能: ユーザーの可視化HTMLを取得

入力:

- `user_id` (int): ユーザーID
- `viz_id` (str, optional): 可視化ID (UUID)

出力:

- `str`: HTML文字列 (Plotlyグラフ)

例外:

- `KeyError`: キャッシュが見つからない

使用例:

```
html = get_visualization_html(user_id=123, viz_id="550e8400-...")
```

環境変数

変数名	デフォルト値	説明
ENVIRONMENT	"production"	環境設定 ("local" / "production")
DATABASE_URL	"sqlite:///./chat_app.db"	データベース接続URL
SECRET_KEY	"your-secret-key-change-this-in-production"	JWT署名用秘密鍵

変数名	デフォルト値	説明
OPENAI_API_KEY	"not-needed"	OpenAI APIキー
OPENAI_BASE_URL	"http://localhost:1234/v1"	OpenAI API URL（ローカルLLM対応）
OPENAI_MODEL_NAME	"gpt-4o-mini"	使用モデル名
VISUALIZATION_OUTPUT_DIR	"/tmp/visualizations"	可視化ファイル保存先

環境変数設定例

ローカル開発環境

```
ENVIRONMENT=local
DATABASE_URL=sqlite:///./chat_app.db
OPENAI_BASE_URL=http://localhost:1234/v1
OPENAI_MODEL_NAME=llama-3.1-8b-instruct
```

本番環境 (Railway)

```
ENVIRONMENT=production
DATABASE_URL=postgresql://user:password@host:5432/dbname
OPENAI_API_KEY=sk-...
OPENAI_BASE_URL=https://api.openai.com/v1
OPENAI_MODEL_NAME=gpt-4o-mini
SECRET_KEY=random-secure-key-here
VISUALIZATION_OUTPUT_DIR=/tmp/visualizations
```

デプロイメント

Railwayへのデプロイ

設定ファイル: `railway.json`, `nixpacks.toml`

ビルドコマンド:

```
pip install -r requirements.txt
```

起動コマンド:

```
uvicorn main:app --host 0.0.0.0 --port $PORT
```







必須環境変数:

- **DATABASE_URL** - PostgreSQL接続URL (Railwayが自動設定)
- **SECRET_KEY** - JWT署名用秘密鍵 (手動設定)
- **OPENAI_API_KEY** - OpenAI APIキー (手動設定)

まとめ

本アプリケーションは、FastAPI + OpenAI Function Calling + MCP Toolsを組み合わせた**在庫最適化専門AIチャットボット**です。

主要機能

-  ユーザー認証 (JWT)
-  チャット履歴保存
-  ストリーミングレスポンス
-  30種類以上の在庫最適化ツール
-  インタラクティブな可視化 (Plotly)
-  Railway対応

アーキテクチャの特徴

- **Function Calling:** LLMが自動的にツールを選択・実行
- **Two-Step Processing:** パラメータ変換を自動化
- **可視化キャッシュ:** ユーザーごとの可視化結果を管理
- **認証オプション:** ローカル環境では認証不要

技術スタック

- **バックエンド:** FastAPI + Uvicorn
- **データベース:** SQLAlchemy + PostgreSQL/SQLite
- **認証:** JWT (python-jose) + bcrypt
- **AI:** OpenAI API (Function Calling)
- **可視化:** Plotly
- **最適化:** PuLP, SciPy, NumPy

ドキュメント作成日: 2025-10-14

バージョン: 1.0

</body>
</html>