

# cafeMuji 開発環境セットアップガイド

## 概要

cafeMuji注文管理システムの開発環境を構築するための詳細なガイドです。このガイドに従うことで、誰でも同じ開発環境を再現できます。

## 前提条件

### 必要なソフトウェア

- **Python:** 3.8以上（推奨: 3.11）
- **Git:** バージョン管理
- **テキストエディタ:** VSCode、PyCharm等
- **ブラウザ:** Chrome、Firefox等

### 推奨OS

- **macOS:** 10.15以上
- **Windows:** 10以上
- **Linux:** Ubuntu 20.04以上

## 環境構築手順

### 1. Python環境の準備

#### 1.1 Pythonのインストール確認

```
# Pythonバージョンの確認
python3 --version
# 期待される出力: Python 3.11.x

# pipの確認
pip3 --version
# 期待される出力: pip 23.x.x
```

#### 1.2 Pythonのインストール（必要な場合）

```
# macOS (Homebrew使用)
brew install python@3.11

# Ubuntu/Debian
sudo apt update
sudo apt install python3.11 python3.11-venv python3.11-pip

# Windows
# https://www.python.org/downloads/ からダウンロード
```

## 2. プロジェクトのクローン

### 2.1 リポジトリのクローン

```
# リポジトリのクローン
git clone https://github.com/username/cafeMuji.git
cd cafeMuji
```

```
# 現在のブランチ確認
git branch
```

### 2.2 プロジェクト構造の確認

```
# プロジェクト構造の表示
ls -la
```

```
# 期待される出力
# cafeMuji/
# |
# |--- config/          # Django設定
# |--- food/            # フード注文管理
# |--- ice/             # アイスクリーム注文管理
# |--- shavedice/       # かき氷注文管理
# |--- mobile/         # モバイル対応
# |--- common/         # 共通機能
# |--- templates/      # HTMLテンプレート
# |--- static/         # 静的ファイル
# |--- manage.py       # Django管理コマンド
# |--- requirements.txt # 依存関係
```

## 3. 仮想環境の構築

### 3.1 仮想環境の作成

```
# 仮想環境の作成
python3 -m venv venv
```

```
# 仮想環境の有効化
# macOS/Linux
source venv/bin/activate
```

```
# Windows
venv\Scripts\activate
```

```
# 仮想環境の確認
which python
# 期待される出力: /path/to/cafeMuji/venv/bin/python
```

### 3.2 依存関係のインストール

```
# pipのアップグレード
pip install --upgrade pip
```

```
# 依存関係のインストール
pip install -r requirements.txt
```

```
# インストール確認
pip list
```

## 4. データベースの設定

### 4.1 データベースの初期化

```
# マイグレーションの実行
python manage.py migrate

# マイグレーション状況の確認
python manage.py showmigrations
```

## 4.2 スーパーユーザーの作成

```
# 管理画面用ユーザーの作成
python manage.py createsuperuser

# ユーザー名: admin
# メールアドレス: admin@example.com
# パスワード: [安全なパスワード]
```

## 5. 開発サーバーの起動

### 5.1 サーバーの起動

```
# 開発サーバーの起動
python manage.py runserver

# 期待される出力
# Watching for file changes with StatReloader
# Performing system checks...
# System check identified no issues (0 silenced).
# Django version 5.2.1, using settings 'config.settings'
# Starting development server at http://127.0.0.1:8000/
# Quit the server with CONTROL-C.
```

### 5.2 アクセステスト

```
# ブラウザで以下にアクセス
# http://127.0.0.1:8000/admin/      # 管理画面
# http://127.0.0.1:8000/food/      # フード注文
# http://127.0.0.1:8000/ice/       # アイスクリーム注文
# http://127.0.0.1:8000/shavedice/ # かき氷注文
```

## 開発ツールの設定

### 1. VSCode設定

#### 1.1 推奨拡張機能

```
// .vscode/extensions.json
{
  "recommendations": [
    "ms-python.python",
    "ms-python.black-formatter",
    "ms-python.flake8",
    "ms-python.isort",
    "ms-python.pylint",
    "ms-python.python-docstring",
    "ms-python.python-type-stub",
    "ms-python.pylance"
  ]
}
```

#### 1.2 ワークスペース設定

```
// .vscode/settings.json
{
  "python.defaultInterpreterPath": "./venv/bin/python",
  "python.linting.enabled": true,
  "python.linting.pylintEnabled": true,
  "python.formatting.provider": "black",
  "python.sortImports.args": ["--profile", "black"],
  "editor.formatOnSave": true,
  "editor.codeActionsOnSave": {
    "source.organizeImports": true
  }
}
```

## 2. コード品質ツール

### 2.1 Black (コードフォーマッター)

```
# Blackのインストール
pip install black

# コードの自動フォーマット
black .

# 特定ファイルのフォーマット
black food/views.py
```

### 2.2 Flake8 (リンター)

```
# Flake8のインストール
pip install flake8

# コードの品質チェック
flake8 .

# 特定ディレクトリのチェック
flake8 food/
```

### 2.3 isort (インポートソーター)

```
# isortのインストール
pip install isort

# インポートの自動ソート
isort .

# 特定ファイルのソート
isort food/views.py
```



## テスト環境の構築

### 1. テストの実行

```
# 全テストの実行
python manage.py test

# 特定アプリのテスト
python manage.py test food
python manage.py test ice
python manage.py test shavedice

# カバレッジ付きテスト
pip install coverage
coverage run --source='.' manage.py test
```

```
coverage report
coverage html
```

## 2. テストデータの準備

```
# テスト用データの作成
```

```
python manage.py shell
```

```
# サンプルデータの作成例
```

```
from food.models import FoodOrder
from ice.models import Order
from shavedice.models import ShavedIceOrder
```

```
# フード注文のサンプル
```

```
FoodOrder.objects.create(
    menu='からあげ丼',
    quantity=2,
    eat_in=True,
    clip_color='yellow',
    clip_number=1,
    group_id='test_group_001'
)
```

```
# アイスクリーム注文のサンプル
```

```
Order.objects.create(
    group_id='test_group_001',
    size='S',
    container='cup',
    flavor1='jersey',
    clip_color='white',
    clip_number=2
)
```

```
# かき氷注文のサンプル
```

```
ShavedIceOrder.objects.create(
    flavor='抹茶',
    group_id='test_group_001',
    clip_color='yellow',
    clip_number=3
)
```

## デバッグ環境の構築

### 1. Django Debug Toolbar

```
# Django Debug Toolbarのインストール
```

```
pip install django-debug-toolbar
```

```
# settings.pyに追加
```

```
INSTALLED_APPS = [
    # ... 既存のアプリ
    'debug_toolbar',
]
```

```
MIDDLEWARE = [
    'debug_toolbar.middleware.DebugToolbarMiddleware',
    # ... 既存のミドルウェア
]
```

```
INTERNAL_IPS = [
    '127.0.0.1',
]
```

### 2. ログ設定

```
# settings.py
LOGGING = {
```

```

'version': 1,
'disable_existing_loggers': False,
'formatters': {
    'verbose': {
        'format': '{levelname} {asctime} {module} {process:d} {thread:d} {message}',
        'style': '{',
    },
},
'handlers': {
    'console': {
        'class': 'logging.StreamHandler',
        'formatter': 'verbose',
    },
    'file': {
        'class': 'logging.FileHandler',
        'filename': 'debug.log',
        'formatter': 'verbose',
    },
},
'root': {
    'handlers': ['console', 'file'],
    'level': 'INFO',
},
'loggers': {
    'django': {
        'handlers': ['console', 'file'],
        'level': 'INFO',
        'propagate': False,
    },
},
}

```



## 本番環境への準備

### 1. 環境変数の設定

```

# .envファイルの作成
cat > .env << EOF
DEBUG=False
SECRET_KEY=your-secret-key-here
ALLOWED_HOSTS=.onrender.com,localhost,127.0.0.1
DATABASE_URL=sqlite:///db.sqlite3
EOF

```

```

# .envファイルの読み込み
pip install python-dotenv

```

### 2. 静的ファイルの収集

```

# 静的ファイルの収集
python manage.py collectstatic --noinput

# 収集されたファイルの確認
ls -la staticfiles/

```

### 3. セキュリティ設定

```

# settings.py
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
SECURE_BROWSER_XSS_FILTER = True
SECURE_CONTENT_TYPE_NOSNIFF = True

```

# 開発リソース

## 1. 参考ドキュメント

- Django公式ドキュメント: <https://docs.djangoproject.com/>
- Python公式ドキュメント: <https://docs.python.org/>
- Django REST framework: <https://www.django-rest-framework.org/>

## 2. 開発コミュニティ

- Django公式フォーラム: <https://forum.djangoproject.com/>
- Stack Overflow: <https://stackoverflow.com/questions/tagged/django>
- GitHub: <https://github.com/django/django>

## 3. 学習リソース

- Django Tutorial: <https://docs.djangoproject.com/en/stable/intro/tutorial01/>
- Real Python: <https://realpython.com/tutorials/django/>
- Django Girls: <https://tutorial.djangogirls.org/>

# よくある問題と解決方法

## 1. 仮想環境の問題

```
# 仮想環境が有効にならない
deactivate # 既存の仮想環境を無効化
source venv/bin/activate # 再度有効化
```

```
# パッケージが見つからない
pip install --upgrade pip
pip install -r requirements.txt
```

## 2. データベースの問題

```
# マイグレーションエラー
python manage.py makemigrations
python manage.py migrate
```

```
# データベースのリセット
rm db.sqlite3
python manage.py migrate
python manage.py createsuperuser
```

## 3. 静的ファイルの問題

```
# 静的ファイルが表示されない
python manage.py collectstatic --noinput
python manage.py runserver --insecure
```



# 開発のベストプラクティス

## 1. コード品質

- **命名規則:** PEP 8に従った命名
- **ドキュメント:** 関数・クラスにdocstringを記述
- **テスト:** 新機能には必ずテストを作成
- **レビュー:** コードレビューを実施

## 2. バージョン管理

- **コミット:** 小さく、意味のある単位でコミット
- **ブランチ:** 機能開発は専用ブランチで実施
- **メッセージ:** 明確で説明的なコミットメッセージ

## 3. セキュリティ

- **環境変数:** 機密情報は環境変数で管理
- **入力検証:** ユーザー入力の適切な検証
- **権限管理:** 最小権限の原則を適用

---

作成日: 2025年8月 作成者: 村岡 優次郎 バージョン: 1.0