

cafeMuji 注文管理システム 技術仕様書

1. 技術スタック

1.1 バックエンド

- 言語: Python 3.x
- フレームワーク: Django 5.2.1
- Webサーバー: gunicorn 23.0.0
- WSGI: Django標準WSGI

1.2 フロントエンド

- テンプレートエンジン: Django Templates
- CSS: カスタムCSS（レスポンス対応）
- JavaScript: 最小限（Djangoの機能を活用）

1.3 データベース

- 開発環境: SQLite 3.x
- 本番環境: SQLite（PostgreSQL推奨）
- ORM: Django ORM

1.4 インフラ・デプロイ

- ホスティング: Render.com
- 静的ファイル: Render CDN
- バージョン管理: Git

2. プロジェクト構造

2.1 ディレクトリ構成

```

cafeMuji/
├── config/
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   ├── asgi.py
│   └── wsgi.py
└── food/
    ├── models.py
    ├── views.py
    ├── urls.py
    ├── admin.py
    └── templates/

```

プロジェクト設定

Django設定

メインURL設定

ASGI設定

WSGI設定

フード注文管理アプリ

データベースモデル

ビュー（処理ロジック）

URLルーティング

管理画面設定

テンプレート

ice/	# アイスクリーム注文管理アプリ
shavedice/	# かき氷注文管理アプリ
mobile/	# モバイル対応
common/	# 共通機能
static/	# 静的ファイル
templates/	# 共通テンプレート
manage.py	# Django管理コマンド
requirements.txt	# 依存関係
Procfile	# Render用設定

2.2 アプリケーション設計

各アプリは独立した機能を持ち、DjangoのMVT（Model-View-Template）パターンに従って設計されています。

3. データベース設計

3.1 フード注文テーブル (FoodOrder)

```
class FoodOrder(models.Model):
    # 注文内容
    menu = models.CharField(max_length=20) # メニュー名
    quantity = models.PositiveIntegerField(default=1) # 数量
    eat_in = models.BooleanField(default=True) # 店内/テイクアウト

    # クリップ情報
    clip_color = models.CharField(max_length=10) # クリップ色
    clip_number = models.IntegerField() # クリップ番号

    # 注文管理
    group_id = models.CharField(max_length=50) # グループID
    status = models.CharField(max_length=10, default='ok') # 注文状態
    is_completed = models.BooleanField(default=False) # 完了フラグ

    # 時刻管理
    timestamp = models.DateTimeField(auto_now_add=True) # 受注時刻
    completed_at = models.DateTimeField(null=True, blank=True) # 完了時刻

    # その他
    note = models.TextField(blank=True, null=True) # 備考
```

3.2 アイスクリーム注文テーブル (Order)

```
class Order(models.Model):
    group_id = models.CharField(max_length=20) # グループID
    size = models.CharField(max_length=2) # サイズ (S/W)
    container = models.CharField(max_length=10) # 容器 (cup/cone)
    flavor1 = models.CharField(max_length=50) # フレーバー1
    flavor2 = models.CharField(max_length=50, blank=True, null=True) # フレーバー2
    is_completed = models.BooleanField(default=False) # 完了フラグ
    timestamp = models.DateTimeField(auto_now_add=True) # 受注時刻
    status = models.CharField(max_length=10, default='ok') # 注文状態
    clip_color = models.CharField(max_length=10) # クリップ色
    clip_number = models.IntegerField() # クリップ番号
    completed_at = models.DateTimeField(null=True, blank=True) # 完了時刻
    is_auto_stopped = models.BooleanField(default=False) # 自動STOP
    note = models.TextField(blank=True, null=True) # 備考
    is_pudding = models.BooleanField(default=False) # アフォガードプリン
```

3.3 かき氷注文テーブル (ShavedIceOrder)

```
class ShavedIceOrder(models.Model):
    flavor = models.CharField(max_length=50) # フレーバー
```

```

group_id = models.CharField(max_length=50, default='') # グループID
is_completed = models.BooleanField(default=False) # 完了フラグ
clip_color = models.CharField(max_length=10, default='white') # クリップ色
clip_number = models.IntegerField(default=0) # クリップ番号
timestamp = models.DateTimeField(auto_now_add=True) # 受注時刻
completed_at = models.DateTimeField(null=True, blank=True) # 完了時刻
status = models.CharField(max_length=10, default='ok') # 注文状態
is_auto_stopped = models.BooleanField(default=False) # 自動STOP
note = models.TextField(blank=True, null=True) # 備考

```

4. ビュー設計

4.1 フード注文管理

- **food_register**: 注文登録画面
- **add_temp_food**: 仮注文追加
- **remove_temp_food**: 仮注文削除
- **food_kitchen**: キッチン画面
- **complete_food**: 注文完了処理
- **confirm_food**: 本注文確定

4.2 アイスcream注文管理

- **ice**: 注文画面
- **register**: 注文登録
- **detail**: 注文詳細
- **deshap**: 注文削除

4.3 かき氷注文管理

- **shavedice**: 注文画面
- **shavedice_register**: 注文登録
- **shavedice_kitchen**: キッチン画面
- **wait_time**: 待ち時間表示

5. URL設計

5.1 メインURL設定

```

# config/urls.py
urlpatterns = [
    path('admin/', admin.site.urls),
    path('food/', include('food.urls')),
    path('ice/', include('ice.urls')),
    path('shavedice/', include('shavedice.urls')),
    path('mobile/', include('mobile.urls')),
]

```

5.2 各アプリのURL

各アプリケーションで独立したURLルーティングを実装し、名前空間を明確に分離しています。

6. セッション管理

6.1 仮注文の管理

```
# セッションから仮注文リストを取得
temp_food = request.session.get('temp_food', [])

# 仮注文をセッションに保存
request.session['temp_food'] = temp_food
```

6.2 セッション設定

```
# settings.py
SESSION_ENGINE = 'django.contrib.sessions.backends.db'
SESSION_COOKIE_AGE = 86400 # 24時間
```

7. セキュリティ実装

7.1 CSRF保護

```
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def add_temp_food(request):
    # 仮注文追加処理
```

7.2 入力値検証

```
# 入力値の妥当性チェック
if not menu or eat_in_str not in ['0', '1']:
    return redirect('food_register')
```

8. パフォーマンス最適化

8.1 データベースクエリ最適化

- 適切なインデックス設計
- 必要最小限のフィールド取得
- クエリの効率化

8.2 静的ファイル最適化

- CSS/JSの最小化
- 画像の最適化
- CDNの活用

9. エラーハンドリング

9.1 例外処理

```
try:
    # データベース操作
    order.save()
except Exception as e:
    # エラーログ記録
    logger.error(f"注文保存エラー: {e}")
```

9.2 ログ設定

Django標準のログ機能を活用し、適切なログレベルで情報を記録

10. テスト戦略

10.1 単体テスト

- モデルのテスト
- ビューのテスト
- フォームのテスト

10.2 統合テスト

- エンドツーエンドのテスト
- データベース操作のテスト

作成日: 2025年8月 作成者: 村岡 優次郎 バージョン: 1.0