cafeMuji 注文管理システム 運用マニュアル

1. システム概要

1.1 システムの目的

cafe&meal MUJIの業務効率化を目的とした注文管理用Webアプリケーション

1.2 主要機能

- **フード注文管理**: からあげ丼、ルーロー飯の注文受付・管理
- **アイスクリーム注文管理**: 12種類のフレーバー、サイズ・容器選択
- **かき氷注文管理**: 3種類のフレーバー、注文状態管理
- **役割別UI**: レジ、キッチン、各担当者用の専用インターフェース

1.3 システム構成

- **フレームワーク**: Django 5.2.1
- **データベース**: SQLite（開発）/ PostgreSQL（本番推奨）
- **Webサーバー**: gunicorn
- **ホスティング**: Render.com

2. 日常運用

2.1 システム起動確認

```
# システムの状態確認
curl https://your-app.onrender.com/health/

# 期待される応答
OK
```

2.2 ログの確認

```
# Render Dashboardでのログ確認
# Logsタブ → リアルタイムログ

# エラーログの確認
# エラーレベルのログを重点的に確認
```

2.3 データベースの状態確認

```
# 注文数の確認
python manage.py shell
>>> from food.models import FoodOrder
>>> FoodOrder.objects.count()
>>> FoodOrder.objects.filter(is_completed=False).count()
```

```
# 完了済み注文の確認
>>> FoodOrder.objects.filter(is_completed=True).count()
```

3. ユーザー管理

3.1 ユーザーアカウントの作成

```
# スーパーユーザーの作成
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]
```

3.2 権限管理

```
# ユーザー権限の確認
python manage.py shell
>>> from django.contrib.auth.models import User
>>> user = User.objects.get(username='username')
>>> user.user_permissions.all()
>>> user.groups.all()
```

3.3 パスワード管理

```
# パスワードの変更
python manage.py changepassword username

# パスワードリセット
python manage.py shell
>>> from django.contrib.auth.models import User
>>> user = User.objects.get(username='username')
>>> user.set_password('new_password')
>>> user.save()
```

4. データ管理

4.1 注文データの確認

```
# フード注文の確認
python manage.py shell
>>> from food.models import FoodOrder
>>> orders = FoodOrder.objects.all().order_by('-timestamp')
>>> for order in orders[:5]:
...     print(f"{order.menu} ×{order.quantity} - {order.status}")

# アイスクリーム注文の確認
>>> from ice.models import Order
>>> ice_orders = Order.objects.filter(is_completed=False)
>>> for order in ice_orders:
...     print(f"{order.flavor1} {order.size} - {order.status}")
```

4.2 データのバックアップ

```
# 全データのバックアップ
python manage.py dumpdata > backup_$(date +%Y%m%d_%H%M%S).json

# アプリ別バックアップ
python manage.py dumpdata food > food_backup_$(date +%Y%m%d).json
```

```
python manage.py dumpdata ice > ice_backup_$(date +%Y%m%d).json
python manage.py dumpdata shavedice > shavedice_backup_$(date +%Y%m%d).json

# 特定期間のデータバックアップ
python manage.py dumpdata food --indent 2 > food_backup_$(date +%Y%m%d).json
```

4.3 データの復旧

```
# 全データの復旧
python manage.py loaddata backup_20250818_143000.json

# アプリ別復旧
python manage.py loaddata food_backup_20250818.json

# 復旧前の確認
python manage.py shell
>>> from food.models import FoodOrder
>>> FoodOrder.objects.count()
```

4.4 データのクリーンアップ

```
# 古い完了済み注文の削除（30日以上前）
python manage.py shell
>>> from datetime import datetime, timedelta
>>> from django.utils import timezone
>>> from food.models import FoodOrder
>>> cutoff_date = timezone.now() - timedelta(days=30)
>>> old_orders = FoodOrder.objects.filter(
...     is_completed=True,
...     completed_at__lt=cutoff_date
... )
>>> print(f"削除対象: {old_orders.count()}件")
>>> # old_orders.delete()  # 実際の削除は慎重に実行
```

5. パフォーマンス監視

5.1 システムリソースの確認

```
# CPU使用率の確認
top -p $(pgrep gunicorn)

# メモリ使用量の確認
ps aux | grep gunicorn

# ディスク使用量の確認
df -h
du -sh /path/to/cafeMuji
```

5.2 データベースパフォーマンス

```
# クエリの実行時間確認
python manage.py shell
>>> from django.db import connection
>>> from food.models import FoodOrder
>>> connection.queries = []
>>> orders = FoodOrder.objects.filter(is_completed=False)
>>> print(f"クエリ数: {len(connection.queries)}")
>>> for query in connection.queries:
...     print(f"時間: {query['time']}秒 - {query['sql'][:100]}...")
```

5.3 応答時間の監視

```
# 各エンドポイントの応答時間確認
time curl -s https://your-app.onrender.com/food/register/
time curl -s https://your-app.onrender.com/ice/
time curl -s https://your-app.onrender.com/shavedice/

# 負荷テスト（簡単な方法）
ab -n 100 -c 10 https://your-app.onrender.com/food/register/
```

6. セキュリティ管理

6.1 アクセスログの確認

```
# アクセスログの確認
tail -f logs/access.log

# エラーアクセスの確認
grep "ERROR" logs/access.log | tail -20

# 不正アクセスの検出
grep "404\|403\|500" logs/access.log | wc -l
```

6.2 セキュリティ設定の確認

```
# セキュリティ設定の確認
python manage.py shell
>>> from django.conf import settings
>>> print(f"DEBUG: {settings.DEBUG}")
>>> print(f"SECURE_SSL_REDIRECT: {getattr(settings, 'SECURE_SSL_REDIRECT', False)}")
>>> print(f"SESSION_COOKIE_SECURE: {getattr(settings, 'SESSION_COOKIE_SECURE', False)}")
```

6.3 脆弱性チェック

```
# 依存関係の脆弱性チェック
pip install safety
safety check

# Djangoのセキュリティアップデート確認
pip list --outdated | grep Django
```

7. 定期メンテナンス

7.1 日次メンテナンス

```
# 1. システム状態の確認
curl -s https://your-app.onrender.com/health/

# 2. ログファイルの確認
# Render Dashboardでエラーログを確認

# 3. データベースの状態確認
python manage.py shell
>>> from food.models import FoodOrder
>>> print(f"未完了注文: {FoodOrder.objects.filter(is_completed=False).count()}")
```

7.2 週次メンテナンス

```
# 1. データベースのバックアップ
python manage.py dumpdata > weekly_backup_$(date +%Y%m%d).json

# 2. ログファイルのローテーション
```

```
# 古いログファイルの圧縮・削除

# 3. パフォーマンスの確認
# 応答時間、エラー率の確認
```

7.3 月次メンテナンス

```
# 1. 依存関係の更新確認
pip list --outdated

# 2. セキュリティパッチの適用
pip install --upgrade Django

# 3. データベースの最適化
python manage.py shell
>>> from django.db import connection
>>> cursor = connection.cursor()
>>> cursor.execute("VACUUM;")  # SQLiteの場合
```

8. トラブルシューティング

8.1 よくある問題と解決方法

8.1.1 システムが起動しない

症状: アプリケーションが起動しない、エラーページが表示される

確認項目:

```
# 1. ログの確認
# Render Dashboard → Logsタブ

# 2. 環境変数の確認
# Settings → Environment Variables

# 3. 依存関係の確認
pip install -r requirements.txt
```

解決方法: - 環境変数の設定を確認 - requirements.txtの依存関係を確認 - データベースの接続を確認

8.1.2 データベースエラー

症状: データベース接続エラー、クエリエラー

確認項目:

```
# 1. データベースファイルの確認
ls -la db.sqlite3

# 2. マイグレーションの確認
python manage.py showmigrations

# 3. データベースの整合性確認
python manage.py check
```

解決方法: - マイグレーションの実行: `python manage.py migrate` - データベースファイルの修復 - バックアップからの復旧

8.1.3 静的ファイルが表示されない

症状: CSS、JavaScript、画像が表示されない

確認項目:

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

# 2. 静的ファイルの設定確認
python manage.py shell
>>> from django.conf import settings
>>> print(f"STATIC_ROOT: {settings.STATIC_ROOT}")
>>> print(f"STATIC_URL: {settings.STATIC_URL}")
```

解決方法: - 静的ファイルの再収集 - 設定ファイルの確認 - ファイルパーミッションの確認

8.1.4 セッションが保持されない

症状: ログイン状態が保持されない、仮注文が消える

確認項目:

```
# セッション設定の確認
python manage.py shell
>>> from django.conf import settings
>>> print(f"SESSION_ENGINE: {settings.SESSION_ENGINE}")
>>> print(f"SESSION_COOKIE_AGE: {settings.SESSION_COOKIE_AGE}")
```

解決方法: - セッション設定の確認 - クッキーの設定確認 - セキュリティ設定の調整

8.2 エラーログの分析方法

8.2.1 ログレベルの理解

```
DEBUG: デバッグ情報（開発時のみ）
INFO: 一般的な情報
WARNING: 警告（注意が必要）
ERROR: エラー（機能に影響）
CRITICAL: 致命的なエラー（システム停止の可能性）
```

8.2.2 エラーログの解析

```
# エラーログの抽出
grep "ERROR" logs/django.log | tail -20

# 特定のエラーの検索
grep "DatabaseError" logs/django.log

# エラーの発生頻度確認
grep "ERROR" logs/django.log | cut -d' ' -f1 | sort | uniq -c
```

9. バックアップ・復旧手順

9.1 自動バックアップの設定

```
# cronジョブでの自動バックアップ
# 毎日午前2時にバックアップを実行
0 2 * * * cd /path/to/cafeMuji && python manage.py dumpdata > backup_$(date +\%Y\%m\%d).json

# 週次バックアップ（日曜日午前3時）
0 3 * * 0 cd /path/to/cafeMuji && python manage.py dumpdata > weekly_backup_$(date +\%Y\%m\%d).json
```

9.2 復旧手順

```
# 1. システムの停止
# Render Dashboard → Manual Deploy → Suspend

# 2. データベースの復旧
python manage.py loaddata backup_20250818.json

# 3. システムの再開
# Render Dashboard → Manual Deploy → Resume
```

9.3 バックアップの検証

```
# バックアップファイルの整合性確認
python manage.py shell
>>> from food.models import FoodOrder
>>> original_count = FoodOrder.objects.count()
>>> print(f"元のデータ数: {original_count}")

# バックアップファイルの確認
python -m json.tool backup_20250818.json | head -20
```

10. パフォーマンス最適化

10.1 データベース最適化

```
# インデックスの追加
class FoodOrder(models.Model):
    class Meta:
        indexes = [
            models.Index(fields=['status', 'is_completed']),
            models.Index(fields=['clip_color', 'clip_number']),
            models.Index(fields=['timestamp']),
        ]
```

10.2 キャッシュ設定

```
# キャッシュの設定
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'unique-snowflake',
        'TIMEOUT': 300,  # 5分
    }
}

# ビューでのキャッシュ活用
from django.views.decorators.cache import cache_page

@cache_page(60 * 15)  # 15分間キャッシュ
def food_kitchen(request):
    # キッチン画面の表示
    pass
```

10.3 クエリ最適化

```
# 効率的なクエリ例
def get_active_orders():
    return FoodOrder.objects.filter(
        is_completed=False
    ).select_related().order_by('timestamp')

# バッチ処理での最適化
def bulk_update_orders(orders, status):
    for order in orders:
        order.status = status
    FoodOrder.objects.bulk_update(orders, ['status'])
```

11. 監視・アラート

11.1 監視項目

- **システム可用性**: ヘルスチェックエンドポイント
- **応答時間**: 各ページの表示時間
- **エラー率**: エラーログの発生頻度
- **データベース性能**: クエリ実行時間
- **リソース使用量**: CPU、メモリ、ディスク

11.2 アラート設定

```
# エラー率の監視
import logging
from django.core.mail import send_mail

class ErrorAlertHandler(logging.Handler):
    def emit(self, record):
        if record.levelno >= logging.ERROR:
            send_mail(
                'cafeMuji エラーアラート',
                f'エラーが発生しました: {record.getMessage()}',
                'from@example.com',
                ['admin@example.com'],
                fail_silently=True,
            )

# ログ設定に追加
LOGGING = {
    'handlers': {
        'alert': {
            'class': 'path.to.ErrorAlertHandler',
        },
    },
    'loggers': {
        'django': {
            'handlers': ['alert'],
            'level': 'ERROR',
        },
    },
}
```

12. 更新・リリース管理

12.1 リリース手順

```
# 1. 開発ブランチでのテスト
git checkout develop
```

```
python manage.py test

# 2. メインブランチへのマージ
git checkout main
git merge develop

# 3. タグの作成
git tag -a v1.1.0 -m "Version 1.1.0 release"

# 4. リモートへのプッシュ
git push origin main
git push origin v1.1.0

# 5. Render.comでの自動デプロイ
# プッシュ後、自動的にデプロイが開始される
```

12.2 ロールバック手順

```
# 1. 前回のデプロイへのロールバック
# Render Dashboard → Deploys → 前回のデプロイ → Rollback

# 2. データベースの復旧（必要に応じて）
python manage.py loaddata backup_before_update.json

# 3. システムの動作確認
curl -s https://your-app.onrender.com/health/
```

12.3 リリースノート

```
# リリースノート v1.1.0

## 新機能
- 新しいメニュー「ルーロー飯」の追加
- 注文状態の詳細管理機能

## 改善点
- レスポンシブデザインの改善
- パフォーマンスの最適化

## 修正点
- セッション管理のバグ修正
- データベースクエリの最適化

## 既知の問題
- なし

## 更新手順
1. コードの更新
2. データベースマイグレーション
3. 静的ファイルの更新
```

作成日: 2025年8月 **作成者**: 村岡 優次郎 **バージョン**: 1.0