06

CHAPTER

チャット UI を実装する

Django Templates × htmx × Tailwind — JavaScript ほぼゼロで動的 UI

本書のサンプル章

AGENDA

この章で学ぶこと

01 **Django Templates の基本構造**
ベーステンプレート + 継承

02 **htmx による非同期通信**
JavaScript なしでチャット部分だけ更新

03 **Tailwind CSS でスタイリング**
クラス名だけで見た目を整える

04 **Django ビューでフォーム処理**
POST → DB 保存 → 部分 HTML を返す

05 **静的ファイルの配置と管理**
STATICFILES_DIRS / STATIC_ROOT

06 **つまずきポイント集**
テンプレート未読込・ htmx 不動作・スタイル未反映

TECH STACK

本章で使う 3 つの技術と完成イメージ

サーバー側

Django Templates

テンプレート継承で
共通枠を再利用

非同期通信

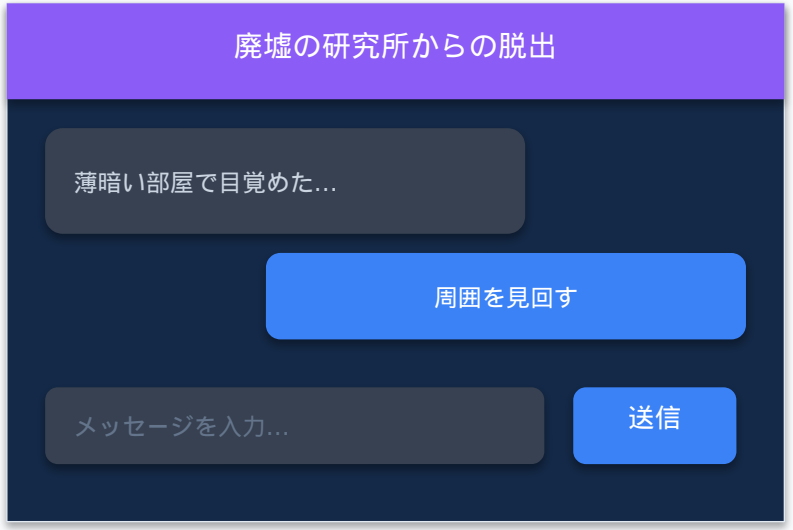
htmx

HTML 属性だけで
Ajax を実現

スタイリング

Tailwind CSS

クラス名で見た目を
サクッと整える



なぜ htmx ？

- ✓ **学習コストが低い** — *HTML 属性だけ*
- ✓ **JS をほぼ書かない** — *コードがシンプル*
- ✓ **Django との相性◎** — *サーバー側で完結*
- ✓ **デバッグしやすい** — *Network タブで確認*

→ React/Vue より遥かに早く動くものができる。複雑な SPA 以外なら htmx で十分。

STRUCTURE

ファイル構成と役割

adventure_game/

```
├── game/
│   ├── templates/
│   │   ├── game/
│   │   │   ├── base.html          #  共通枠
│   │   │   └── game.html          #  ゲーム画面
│   │   └── partials/
│   │       └── chat_messages.html #  チャット部分
│   ├── views.py                   #  ロジック
│   └── urls.py
├── server/
│   └── settings.py
└── manage.py
```

テンプレートの 3 つの役割

base.html — 全ページ共通の枠組み

ヘッダー・フッター・CDN 読込

game.html — ゲーム画面全体

{% extends 'base.html' %}

partials/chat_messages.html — チャット部分のみ

htmx で更新する対象

BASE TEMPLATE

base.html — テンプレート継承の仕組み

game/templates/game/base.html

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <title>{% block title %}AI Adventure{% endblock %}</title>
  <!-- Tailwind CSS CDN -->
  <script src="https://cdn.tailwindcss.com">
  </script>
  <!-- htmx CDN -->
  <script src="https://unpkg.com/htmx.org@1.9.10">
  </script>
</head>
<body class="bg-gray-900 text-white">
  <header>...</header>
  <main>
    {% block content %}{% endblock %}
  </main>
  <footer>...</footer>
</body>
</html>
```

3 つのキー要素

{% block %}

子で上書き可能

title / content / extra_scripts

CDN 読み込み

Tailwind + htmx

ビルドツール不要

ダークテーマ

bg-gray-900

Tailwind で一発適用

クラス名の読み方

数字

大きいほど効果が強い

```
p-2 < p-4 < p-8
```

色 - 数字

数字が大きいほど濃い

```
gray-100 < gray-500 < gray-900
```

サイズ略語

sm < md < lg < xl < 2xl

```
text-sm < text-lg < text-2xl
```

本章で使う主要クラス

bg-gray-900
背景色（濃いグレー）

text-white
文字色（白）

min-h-screen
最小高さを画面全体に

flex
Flexbox レイアウト

px-4 / py-2
左右/上下パディング

rounded-lg
角を丸く（大）

shadow-2xl
影（最大）

animate-spin
回転アニメ

PART 2

htmx で動的 UI

HTML 属性だけで Ajax — JS をほぼ書かない

htmx の主要 3 属性

01

hx-post

POST 送信先の URL

```
"{% url 'game:game_action' %}"
```

送信ボタンを押した時の宛先

02

hx-target

更新対象の要素

```
"#chat-container"
```

CSS セレクタで指定

03

hx-swap

更新方法

```
"innerHTML"
```

innerHTML / outerHTML / beforeend など

実際のフォーム

```
<form
    hx-post="{% url 'game:game_action' %}"
    hx-target="#chat-container"
    hx-swap="innerHTML">
    {% csrf_token %}      # Django 必須のセキュリティトークン
    <input name="message"> <button>送信</button>
</form>
```

ゲーム画面の 3 つのエリア

3 つの主要エリア

ヘッダー　*ゲームタイトル*

`gradient-to-r from-blue-600 to-purple-600`

チャットコンテナ　*#chat-container*

`h-96 overflow-y-auto`
スピナー重ねる

入力フォーム　*hx-post で送信*

`innerHTML` を入れ替え

PARTIAL

chat_messages.html — 部分テンプレート

game/templates/partials/chat_messages.html

```
{% for message in messages %}
<div class="{% if message.role == 'user' %}
              text-right{% endif %}">
    <div class="...rounded-lg
        {% if message.role == 'user' %}
            bg-blue-600 text-white
        {% else %}
            bg-gray-700 text-gray-100
        {% endif %}">
        <p>{{ message.role|upper }}</p>
        <p>{{ message.content }}</p>
        <p>{{ message.timestamp|date:"H:i" }}</p>
    </div>
</div>
{% empty %}
  <p>まだメッセージがありません</p>
{% endfor %}
```

なぜ partials/ ？

1. **更新範囲を明確化**

   htmx で差し替える部分だけを別ファイルにする

2. **再利用しやすい**

   他の場所からも include できる

3. **責任の分離**

   全体（game.html）と部分（partial）を切り分け

→ *ページ全体ではなく、一部分だけを返すテンプレート*

EVENTS

htmx の主要イベント

`htmx:beforeRequest`

WHEN

リクエスト送信直前

USE FOR

ローディング表示
ボタン無効化

`htmx:afterRequest`

WHEN

リクエスト完了後

USE FOR

ローディング非表示

`htmx:afterSwap`

WHEN

DOM 更新直後

USE FOR

スクロール
追加処理

`htmx:afterSettle`

WHEN

アニメ完了後

USE FOR

最終調整
レンダリング待ち

`htmx:responseError`

WHEN

エラー発生時

USE FOR

エラー表示
UI 復旧

本章の活用例: beforeRequest でスピナー表示 → afterSwap でスクロール → responseError で UI 復旧

PART 3

Django ビューと静的ファイル

POST → DB 保存 → 部分 HTML を返すだけ

views.py

home & game_action

def home

初回アクセス時

```
def home(request):
    sid = request.session\
        .session_key
    if not sid:
        request.session.create()

    gs, created = GameSession\
        .objects.get_or_create(
            session_id=sid)

    #  初回なら開始メッセージ
    if created:
        ai = generate_game_response(
            "", gs)
        ChatMessage.objects.create(
            session=gs,
            role='assistant',
            content=ai)

    return render(request,
        'game/game.html',
        {'messages': messages})
```

def game_action

送信時 (htmx が呼ぶ)

```
def game_action(request):
    msg = request.POST.get(
        'message', '').strip()

    #  ユーザーメッセージを保存
    ChatMessage.objects.create(
        session=gs,
        role='user',
        content=msg)

    # AI 応答を生成 &  保存
    ai = generate_game_response(
        msg, gs)
    ChatMessage.objects.create(
        session=gs,
        role='assistant',
        content=ai)

    #  部分テンプレを返す
    return render(request,
      'partials/chat_messages.html',
      {'messages': messages})
```

@rate_limit デコレーターで 4 秒間隔を保証

rate_limit デコレーター

```
last_request_time = 0
min_interval = 4   # 15 RPM = 4 秒

def rate_limit(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        global last_request_time
        elapsed = time.time() \
            - last_request_time
        if elapsed < min_interval:
            time.sleep(
              min_interval - elapsed)
        result = func(*args, **kwargs)
        last_request_time = time.time()
        return result
    return wrapper

@rate_limit
def generate_game_response(...):
    ...
```

ポイント

4 秒間隔を強制

前回呼び出しから 4 秒未満なら待機

@ デコレーターで適用

関数定義の上に @rate_limit

429 エラーを未然に防ぐ

Gemini の 15 RPM に収まる

第 5 章のリトライと併用

それでも失敗時は指数バックオフ

STATIC FILES

静的ファイル — 開発と本番の使い分け

STATIC_URL

URL プレフィックス

```
/static/
```

ブラウザからの参照パス

STATICFILES_DIRS

開発時の検索パス

```
[BASE_DIR / 'static']
```

プロジェクトルートの static/

STATIC_ROOT

本番用の収集先

```
BASE_DIR / 'staticfiles'
```

collectstatic でここに集約

本番用 : collectstatic コマンド

```
$ uv run manage.py collectstatic
```

→ 各アプリ + STATICFILES_DIRS の静的ファイルを STATIC_ROOT に集約。開発環境（DEBUG=True）では不要。

PART 4

つまずきと比較

よくある詰まりポイントと、 htmx vs React/Vue

TROUBLESHOOTING

つまずきポイント TOP 4

01

テンプレートが読み込まれない

```
TemplateDoesNotExist
```

→ INSTALLED_APPS に 'game'
→ 配置パスを確認

02

htmx が動かない

```
ページ全体がリロードされる
```

→ CDN 読込確認
→ hx-post のタイポ
→ キャッシュクリア

03

スタイルが反映されない

```
Tailwind クラスが効かない
```

→ Tailwind CDN 確認
→ クラス名のタイポ
→ Ctrl+Shift+R

04

CSRF トークンエラー

```
Forbidden (403)
```

→ {% csrf_token %} 必須
→ POST フォームに追加

DONE

チャット UI 完成。動的なゲーム体験へ。

htmx vs React/Vue

用途	推奨
初心者の学習	htmx
管理画面・社内ツール	htmx
CRUD 中心の Web アプリ	htmx
複雑な SPA	React / Vue
リアルタイムチャット	WebSocket + JS

NEXT

第 7 章

ゲームロジックの改良

- 会話履歴の永続化
- セッション管理の改善
- ゲームの完成度を高める

→ ロジックを磨いて完成へ

本書では学習コストを最小化する htmx を採用。複雑な SPA は React/Vue を選ぶ。