## ТЗ: Дизайн-Система Веб-Форм

## 1. Цель проекта

Разработать модульную систему на Python для **генерации и рендеринга веб-форм** в двух разных стилях (**Bootstrap** и **Material Design**).

## 2. Требуемые Паттерны и их Роль

| Паттерн | Назначение в проекте |
|---|---|
| **Abstract Factory (Абстрактная фабрика)** | Создание **семейства связанных UI-компонентов** (Кнопка, ПолеВвода) для конкретной темы. Отвечает за генерацию **разных словарей данных** (с разными CSS-классами) для Bootstrap и Material. |
| **Builder (Строитель)** | **Пошаговое конструирование сложного объекта — Веб-Формы**. Он собирает компоненты (словарь данных), полученные от Фабрики, в нужную структуру (список компонентов). |
| **Singleton (Одиночка)** | **Централизованное управление конфигурацией** (DesignSystemConfig). Хранит ссылку на текущую активную ThemeFactory и обеспечивает единственную точку доступа к настройкам системы. |

## 3. Детали реализации

### 3.1. Abstract Factory (Генерация Компонентов)

- **Абстрактные Продукты (Интерфейсы UI):** Button, Input.
- **Конкретные Продукты:**
    - **Bootstrap:** BootstrapButton, BootstrapInput.
    - **Material:** MaterialButton, MaterialInput.
- **Метод Рендеринга:** Метод render() каждого конкретного продукта должен возвращать **словарь Python** с метаданными, необходимыми для Jinja2-шаблона. Словарь должен включать как минимум:

    - 'type': (e.g., 'button', 'input')
    - 'label': Текст для отображения.
    - 'css_class': Строка с CSS-классами, специфичными для темы (например, 'btn btn-primary' или 'material-raised-button').
- **Абстрактная Фабрика:** ThemeFactory с методами create_button() и create_input().
- **Конкретные Фабрики:** BootstrapFactory, MaterialFactory.

### 3.2. Builder (Сборка Формы)

- **Продукт:** Класс WebForm, содержащий **список** словарей-компонентов.
- **Строитель:** FormBuilder с методами, использующими компоненты от Фабрики и добавляющими их в список WebForm:
    - add_title(text)
    - add_input(label)
    - add_submit_button(label)
- **Директор:** FormDirector, который знает последовательность сборки типовых форм. Должен иметь метод build_login_form(factory), который принимает **конкретную фабрику** и использует ее для создания компонентов через Строителя.

### 3.3. Singleton (Конфигурация)

- **Одиночка:** Класс DesignSystemConfig.
- **Функциональность:** Должен предоставлять метод get_instance() и хранить ссылку на **текущую активную ThemeFactory**.

### 3.4. Визуализация с Jinja2 (Ключевой элемент)

1. **Шаблон:** Создать один **универсальный HTML-шаблон** (form.html), который умеет принимать список компонентов (WebForm.components) и рендерить их, используя логику {% if item.type == '...' %} и вставляя предоставленные CSS-классы.
2. **Финальный Рендер:** Создать функцию, которая принимает объект WebForm, загружает шаблон form.html и возвращает **финальную HTML-строку**, готовую для отображения в браузере.

## 4. Итоговый результат

Программа должна продемонстрировать:

1. Создание Login-формы с использованием BootstrapFactory (через Директора).
2. **Рендеринг** этой формы в HTML с Bootstrap-классами.
3. Переключение активной темы в DesignSystemConfig на MaterialFactory.
4. Создание второй формы (например, Registration-формы) с использованием MaterialFactory.
5. **Рендеринг** второй формы в HTML с Material Design-классами.