Propuesta de Constructor de Landings

Arquitectura de Carpetas y Archivos (estilo GitHub)

```
landing-builder/
│
├── app/
│   ├── __init__.py
│   ├── routes/
│   │   ├── builder.py        # Lógica del constructor
│   │   ├── metrics.py        # Registro de analíticas
│   │   └── domains.py        # Gestión de dominios/subdominios
│   ├── models.py             # Modelos de SQLAlchemy
│   ├── templates/
│   │   ├── base.html
│   │   ├── builder/          # Interfaz del editor de landings
│   │   ├── themes/           # Carpeta con plantillas disponibles
│   │   └── render/           # Landings renderizadas por usuarios
│   ├── static/
│   │   ├── css/, js/, img/
│   │   └── user_generated/   # Archivos de landings personalizadas
│   └── forms.py              # Formularios de edición
│
├── database/
│   └── migrations/
│
├── scripts/
│   └── analytics_worker.py   # Recolector de métricas asincrónicas
│
├── config.py
├── run.py
├── requirements.txt
└── README.md
```

⚙️ Base de Datos (SQLAlchemy)

Modelo básico:

```
class Landing(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, nullable=False)
    theme = db.Column(db.String(50))
```

```
    custom_fields = db.Column(db.JSON)  # Título, CTA, etc.
    slug = db.Column(db.String(100), unique=True)
    domain = db.Column(db.String(255))  # Subdominio o dominio personalizado
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

class Metric(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    landing_id = db.Column(db.Integer, db.ForeignKey('landing.id'))
    visit_time = db.Column(db.DateTime, default=datetime.utcnow)
    action_type = db.Column(db.String(20))  # 'visit', 'form_submit'
```

🧩 Templates Personalizables

- [Jinja2 Blocks] para cada plantilla base en `templates/themes/`.
- La landing del usuario se genera combinando la plantilla elegida con los `custom_fields` almacenados.
- Un sistema básico con `render_template("themes/modern.html", **custom_fields)` te permite mostrar un HTML con contenido exclusivo por landing.

🔌 Librerías y Herramientas Clave

Propósito	Herramienta Recomendada
Backend	Flask, SQLAlchemy, Flask-WTF
Auth (opcional)	Flask-Login, OAuth2
Analíticas	Redis + Celery o Flask Background Tasks
Dominios personalizados	Flask-Subdomain + configuración DNS
Formularios dinámicos	WTForms + JS dinámico

Frontend builder (visual)	GrapesJS (opcional, avanzado)
Hosting	Replit, Render o Vercel (con workaround)
Uptime y pings	UptimeRobot + cron + health route

🔧 Paso a Paso para Implementar

1. **Inicializar proyecto Flask**: estructura de carpetas + configuración básica + entorno virtual.
2. **Definir modelos y migraciones**: `Landing` y `Metric`.
3. **Implementar el sistema de creación de landing**:
    - Selector de plantilla.
    - Formulario de campos personalizados.
    - Guardado en BD.
    - Renderizado en `themes/plantilla.html`.
4. **Agregar analíticas**:
    - Middleware o decorador que registre cada visita o envío.
    - Script que los procese en batch (ej. `analytics_worker.py`).
5. **Subdominios (opcional)**:
    - Activar soporte en Flask (`app.config['SERVER_NAME'] = "tudominio.com"`).
    - Gestionar DNS dinámico o usar un dominio por default tipo `usuario.landingbuilder.com`.
6. **Panel del usuario**:
    - Dashboard con métricas.
    - Edición y administración de landings.
7. **Testing + Deploy**:
    - Verificar que cada ruta funcione con datos personalizados.
    - Hospedar y usar UptimeRobot o Pingdom para monitoreo.

🎁 Extra: Tips de Diseño Accesible

Ya que importa la accesibilidad:

- Usar etiquetas `<label>` siempre unidas a inputs.
- Validación clara y mensajes con ARIA live regions.
- Templates con contraste alto, navegación por teclado y fuentes escalables.