🏗️ Estructura del Proyecto (repositorio GitHub)

```
educa_platform/
├── backend/
│   ├── core/
│   │   ├── models/
│   │   ├── views/
│   │   ├── serializers/
│   │   └── urls.py
│   ├── users/
│   │   ├── models.py
│   │   ├── views.py
│   │   ├── serializers.py
│   │   ├── permissions.py
│   │   └── urls.py
│   ├── contents/
│   │   ├── models.py
│   │   ├── views.py
│   │   ├── serializers.py
│   │   └── urls.py
│   ├── statistics/
│   ├── settings.py
│   └── manage.py
├── frontend/
│   ├── templates/
│   ├── static/
│   │   ├── css/
│   │   ├── js/
│   │   └── icons/
│   └── index.html
├── docs/
│   ├── README.md
│   ├── API_REFERENCE.md
│   └── ACCESSIBILITY_GUIDELINES.md
├── requirements.txt
└── README.md
```

🧠 Modelado de la Base de Datos

👤 `User` (custom `AbstractUser`)

- username (str)
- email (str)
- password (hashed)
- is_premium (bool)

- date_joined (datetime)

📚 `Course`

- title (str)
- description (text)
- image (img path or URL)
- level_required (choices: Free, Premium)
- created_at (datetime)

🧩 `Pictogram` (opcional para integrar ARASAAC)

- pictogram_id (str, desde API ARASAAC)
- course (FK)
- caption (str)

📊 `UsageStats`

```
- user (FK)
- course (FK)
- last_access (datetime)
- times_accessed (int)
```

🔐 Gestión de Usuarios y Accesos

Registro/Login

- Usá `django-allauth` o `dj-rest-auth` si optás por API con tokens.
- Seguridad con `django-axes` o `django-rest-knox`.

Acceso a Contenido

- Middleware o decorador `@premium_required` para vistas restringidas.
- En frontend: deshabilitar o bloquear botón de acceso según `user.is_premium`.

```
# Decorador de ejemplo

from django.http import HttpResponseForbidden
```

```
def premium_required(view_func):

    def _wrapped_view(request, *args, **kwargs):

        if not request.user.is_authenticated or not
request.user.is_premium:

            return HttpResponseForbidden("Contenido exclusivo para
usuarios premium.")

        return view_func(request, *args, **kwargs)

    return _wrapped_view
```

📈 Panel de Administración

- Django admin personalizado (`ModelAdmin`) con filtros por tipo de usuario, cursos más vistos, fechas.
- Estadísticas con gráficos: integrá `django-plotly-dash` o `Chart.js` en el frontend con una API de backend que exponga datos.

🔗 Integraciones externas

- **ARASAAC**: pictogramas desde su API REST pública.
- **Generador de voz**: usá `gTTS` (Google Text-to-Speech) para convertir texto a audio y almacenarlo como `.mp3`, o integrá directamente APIs como `ResponsiveVoice` o `SpeechCloud`.

🎨 Diseño accesible y responsive

Requisito	Sugerencia
Diseño responsive	CSS Grid + Flexbox + media queries
Accesibilidad	Uso semántico del HTML + atributos ARIA
Pictogramas & contraste	Alto contraste, pictogramas con alt text descriptivo
Navegación con teclado	`tabindex`, focus visuales visibles

Librerías útiles	`Bootstrap 5`, `A11y Dialog`, `Accessible Rich Widgets`

Además, para accesibilidad en el backend:

- `django-accessible-admin` para hacer el admin más inclusivo.
- `django-user-agents` para adaptar contenido según dispositivo.

🔧 Recomendaciones Técnicas

- **Entorno virtual**: Python `venv` o `poetry` para dependencias.
- **Documentación continua**: `README.md`, diagramas UML simples, guías de acceso.
- **Pruebas unitarias**: `pytest-django`, cobertura con `coverage.py`.
- **CI/CD básico**: integrá GitHub Actions para test automático.

📦 Librerías Clave

Propósito	Librería Sugerida
Autenticación	`django-allauth`
APIs REST	`Django REST Framework`
Gestión de imágenes/audio	`Pillow`, `gTTS`, `FFmpeg`
Estadísticas/gráficos	`Chart.js`, `plotly`, `matplotlib`
Diseño web	`Bootstrap`, `Tailwind`
Tests	`pytest`, `factory_boy`