

**UNIVERSIDAD DE COSTA RICA**  
**Facultad de Ingeniería**  
**Escuela de Ingeniería Eléctrica**

**IE0499 – Proyecto Eléctrico**

**Creación de una plataforma de presentación interactiva  
remota con Django y Reveal.js**

por

**Emmanuel Morera Salas**

**Ciudad Universitaria Rodrigo Facio**

Enero de 2021



# **Creación de una plataforma de presentación interactiva remota con Django y Reveal.js**

por

**Emmanuel Morera Salas**

A84376

**IE0499 – Proyecto Eléctrico**

Aprobado por

---

Ing. Fabián Abarca Calderón, M.Sc

*Profesor guía*

---

Ing. Lochi Yu Lo, PhD

*Profesor lector*

---

Ing. Teodoro Willink Castro, MSc

*Profesor lector*

Enero de 2021



## Resumen

# Creación de una plataforma de presentación interactiva remota con Django y Reveal.js

por

**Emmanuel Morera Salas**

Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

Profesor guía: Ing. Fabián Abarca Calderón, M.Sc

Enero de 2021

En este proyecto, se desarrollará una plataforma para generación de presentaciones interactivas en la *web*, basado en el estudio de DJANGO y REVEAL.JS como plataformas (*framework*), para el diseño de aplicaciones web. Donde se estudiará la posibilidad de utilizar a DJANGO como (*framework*) *back end*, que administre bases de datos y controle sesiones de usuarios, con el fin de procesar la información suministrada por ellos, en las interacciones generadas durante las presentaciones *web*. Por otro lado se estudiará la posibilidad de utilizar a REVEAL.JS como plataforma (*framework*) *front end*, para el diseño de presentaciones *web*, creando una plantilla base, que muestre el contenido de las presentaciones *web* al usuario final. A su vez, utilizar diferentes *plugins* de REVEAL.JS que mejoren la experiencia del usuario final, al contar con elementos de interacción, guiados por el expositor, que cuenta con el control de la presentación que observan, de forma remota. Por ultimo se diseñará e implementará una aplicación *web*, que contenga las características principales de estos dos *frameworks* y que pueda ser accedida por medio de un servidor *web* de internet.

**Palabras claves:** *web, framework, plugin, back end, front end, DJANGO, REVEAL.JS.*

---

### Acerca de IE0499 – Proyecto Eléctrico

El Proyecto Eléctrico es un curso semestral bajo la modalidad de trabajo individual supervisado, con el propósito de aplicar estrategias de diseño y análisis a un problema de temática abierta de la ingeniería eléctrica. Es un requisito de graduación para el grado de Bachiller en Ingeniería Eléctrica de la Universidad de Costa Rica.



## Abstract

# Creación de una plataforma de presentación interactiva remota con Django y Reveal.js

Original in Spanish. Translated as: “Creating a remote interactive presentation platform with DJANGO and REVEAL.JS”

by

**Emmanuel Morera Salas**

University of Costa Rica

Department of Electrical Engineering

Tutor: Ing. Fabián Abarca Calderón, M.Sc

January of 2021

In this project, we will develop a platform for the generation of interactive *web* presentations, based on the study of DJANGO and REVEAL.JS as framework, for the design of *web* applications. Where we will study the possibility of using a DJANGO as *back end* framework, which manages databases and controls user sessions, in order to process the information provided by them, in the interactions generated during *web* presentations.

On the other hand, we will study the possibility of using a REVEAL.JS as a *front end* framework, for the design of web presentations, creating a base template that shows the content of the final user web presentations. At the same time, use different plugins of REVEAL.JS that improve the final user experience, by having interaction elements, guided by the exhibitor, who has control of the presentation they observe, remotely.

Finally, we will be desing and implement a web application that contains the main characteristics of these two frameworks and that can be accessed through an the internet web server.

**Keywords:** *web, framework, plugin, back end, front end, DJANGO, REVEAL.JS.*

---

### About IE0499 – Proyecto Eléctrico (“Electrical Project”)

The “Electrical Project” (or “capstone project”) is a course of supervised individual work of one semester, with the purpose of applying design and analysis strategies to a problem in an open topic in electrical engineering. It is a requisite of graduation for the Bachelor of Science in Electrical Engineering, granted by the University of Costa Rica.





*Dedicado a mi familia y amigos.*

## **Agradecimientos**

Quiero agradecer primero a Dios y mi familia por el apoyo que me han brindado a lo largo de mi carrera en la Universidad, por estar siempre presentes en cada uno de los retos que he enfrentado a lo largo de esta experiencia.

Quiero agradecer a mi profesor guía, Ing. Fabián Abarca Calderón, M.Sc por su apoyo durante el desarrollo del proyecto eléctrico, que a pesar de los desafíos de desarrollo de un proyecto eléctrico, en medio de una pandemia, pudimos sacar adelante el mismo. Agradecer a mis profesores lectores Ing. Lochi Yu Lo, PhD e Ing. Teodoro Willink Castro, MSc, por el apoyo brindado durante la realización del proyecto eléctrico Finalmente a esta institución behemente por brindarme la oportunidad de estudiar en ella.



# Índice general

<b>Índice general</b>	<b>xi</b>
<b>Índice de figuras</b>	<b>xiii</b>
<b>Nomenclatura</b>	<b>xv</b>
<b>1 Introducción</b>	<b>1</b>
1.1. Alcances del proyecto . . . . .	1
1.2. Justificación . . . . .	1
1.3. Problema a resolver . . . . .	2
1.4. Objetivo general . . . . .	2
1.5. Objetivos específicos . . . . .	2
1.6. Metodología . . . . .	3
1.7. Consideraciones sobre el consumo de ancho de banda . . . . .	3
<b>2 Presentaciones web para sesiones virtuales sincrónicas</b>	<b>5</b>
2.1. Contexto del curso . . . . .	5
2.1.1. Materiales y evaluación del curso . . . . .	5
2.1.2. Tipos de interacción en sesiones virtuales . . . . .	6
2.1.3. Tipos de contenido en sesiones virtuales sincrónicas . . . . .	6
<b>3 Entorno de desarrollo web Django</b>	<b>7</b>
3.1. ¿Que es Django? . . . . .	7
3.2. Estructura de Django . . . . .	8
3.2.1. Directorio raíz de la aplicación . . . . .	8
3.2.2. Directorio principal . . . . .	9
3.2.3. Directorio de aplicaciones en DJANGO . . . . .	11
3.3. Manejo de la aplicación en DJANGO . . . . .	13
<b>4 Entorno de presentaciones web Reveal.js</b>	<b>15</b>
4.1. Características de REVEAL.JS . . . . .	15
4.2. Estructura de REVEAL.JS . . . . .	16

4.3.	Instalación de REVEAL.JS . . . . .	19
4.4.	Reveal Multiplex . . . . .	19
4.4.1.	Estructura de Multiplex . . . . .	20
4.4.2.	Instalación de Multiplex . . . . .	21
<b>5</b>	<b>Entorno de producción para el sitio <i>web</i> (Heroku)</b>	<b>23</b>
5.1.	Tipos de entorno de producción . . . . .	23
5.2.	Heroku como entorno de producción . . . . .	24
5.2.1.	GitHub como repositorio para Heroku . . . . .	24
5.2.2.	Descripción general de la implementación de una aplicación <i>web</i> en Heroku . .	25
<b>6</b>	<b>Resultados de la implementación <i>web</i></b>	<b>29</b>
6.1.	Implementación del proyecto en DJANGO . . . . .	29
6.1.1.	Dependencias . . . . .	29
6.1.2.	Estructura de Django a implementar . . . . .	29
6.1.3.	Directorio raíz de la aplicación . . . . .	29
6.1.4.	Directorio de aplicación Principal . . . . .	30
6.1.5.	Aplicación para Sesiones de usuario . . . . .	32
6.1.6.	Aplicación para Presentación de diapositivas . . . . .	37
6.1.7.	Directorio Static . . . . .	39
6.2.	Implementación de REVEAL.JS . . . . .	40
6.2.1.	Dependencias . . . . .	40
6.2.2.	Estructura de REVEAL.JS para la aplicación <i>web</i> . . . . .	40
6.3.	Presentaciones remotas con Multiplex . . . . .	43
6.4.	Implementación del servidor web . . . . .	45
6.4.1.	Dependencias para la puesta en marcha en un servidor en linea . . . . .	45
6.4.2.	Visualización de la aplicación <i>web</i> en Heroku . . . . .	47
<b>7</b>	<b>Conclusiones y recomendaciones</b>	<b>51</b>
7.1.	Conclusiones . . . . .	51
7.2.	Recomendaciones . . . . .	52
	<b>Bibliografía</b>	<b>53</b>

# Índice de figuras

2.1. Tipos de interacción según flujos de información (representado con una flecha). . . . .	6
(a). Presentar . . . . .	6
(b). Interactuar . . . . .	6
(c). Demostrar . . . . .	6
(d). Colaborar . . . . .	6
3.1. Elementos que contienen un directorio raíz en Django. . . . .	8
3.2. Elementos del directorio principal en una aplicación en Django. . . . .	9
3.3. Estructura básica de una “aplicación” en Django. . . . .	11
4.1. Logo de REVEAL.JS. . . . .	16
4.2. Diagrama de comunicación de Multiplex. . . . .	20
6.1. Elementos que contienen un directorio raíz del proyecto <i>web</i> . . . . .	30
6.2. Elementos que contienen el directorio principal del proyecto <i>web</i> . . . . .	31
6.3. Elementos que contiene el directorio de la “aplicación” de sesiones de usuarios . . . . .	33
6.4. Elementos que contiene el directorio de la aplicación presentación . . . . .	37
6.5. Elementos que contiene el directorio <i>static</i> . . . . .	40
6.6. Inicio de sesión en la aplicación web de prueba . . . . .	47
6.7. Registro de usuarios en la aplicación web de prueba . . . . .	48
6.8. Página de Inicio para la aplicación web de prueba . . . . .	49
6.9. Ejemplo de Selección única de la aplicación web de prueba . . . . .	49
6.10. Manejo de datos adquiridos por el evento de selección única de la aplicación web de prueba	50
6.11. Pagina de la administración de la aplicación web de prueba . . . . .	50



# Nomenclatura

*.py* Python

*framework* plataforma de trabajo

*HTML* Lenguaje de marcas de hipertexto (del inglés HyperText Markup Language)

*MTV* Modelo plantilla vista (del inglés *Model view controller*)

*MVC* Modelo vista controlador (del inglés *Model view controller*)

*plugin* Complemento de una aplicación

*web* Hace referencia a una página alojada en la red de informacion global (del inglés *World Wide Web*)

*www* Red de informacion global (del inglés *World Wide Web*)

*EIE* Escuela de Ingeniería Eléctrica de la Universidad de Costa Rica

*IEEE* Instituto de Ingenieros Eléctricos y Electrónicos (del inglés *Institute of Electrical and Electronics Engineers*)





# INTRODUCCIÓN

LAS TECNOLOGÍAS WEB han alcanzado gran madurez y versatilidad y los navegadores web son omnipresentes en los dispositivos inteligentes con los que accedemos a sesiones virtuales sincrónicas en la actualidad. Considerando lo anterior, con DJANGO (tecnología *back end*) y REVEAL.JS (tecnología *front end*) es posible crear una plataforma para presentaciones interactivas adaptadas a las necesidades del curso MODELOS PROBABILÍSTICOS DE SEÑALES Y SISTEMAS y que aproveche algunas de estas ventajas.

## 1.1. Alcances del proyecto

En este proyecto se desarrolla el *back end* y *front end* de un sitio web diseñado específicamente para **sesiones virtuales sincrónicas** en donde hay alta presencia de contenido de *matemáticas, multimedia y programación*. El diseño asume lo siguiente:

1. El docente crea el contenido de la presentación directamente en código HTML (es decir, no hay interfaz gráfica para creación de diapositivas).
2. El docente hace una presentación sincrónica a través de una conferencia virtual (por ejemplo, con Zoom), mientras controla el avance de las diapositivas de la presentación en una página web.
3. En un navegador web, los participantes inician sesión con un usuario registrado a la página donde pueden ver la presentación controlada por el docente (a quien escuchan por la conferencia virtual) y participan en las interactividades ofrecidas en la presentación.

## 1.2. Justificación

La reciente necesidad de migrar a las lecciones virtuales ha obligado a utilizar nuevas herramientas tecnológicas para compartir contenido con los estudiantes, tanto de forma sincrónica como asincrónica. Sin embargo, no todas las tecnologías o sistemas disponibles (Google Slides, PowerPoint, Slidebean, Nearpod y L<sup>A</sup>T<sub>E</sub>X Beamer, entre otros) satisfacen las necesidades específicas de nuestros cursos, ya sea porque no permiten multimedia o no permiten interacción o no son de uso libre u otras razones.

En este contexto, es deseable explorar las posibilidades de creación de un sistema para **compartir contenido** multimedia e **interactuar** con los estudiantes que tenga mayor flexibilidad y esté basado en tecnologías abiertas.

### 1.3. Problema a resolver

Para el sitio web previsto es necesario:

- UNA PLATAFORMA *BACK END* que registre a los usuarios participantes y sus actividades en una *sesión*, y que gestione la información almacenada, posiblemente de forma interactiva (por ejemplo, en los resultados de una votación tipo *quiz*).
- UNA TECNOLOGÍA *FRONT END* que permita mostrar contenido multimedia en la forma de presentación de diapositivas, al mismo tiempo que permite utilizar toda la versatilidad de los lenguajes web modernos: HTML, CSS y JavaScript.

**Elección de tecnologías** De las varias posibles soluciones para este tipo de desarrollos, se eligen DJANGO y REVEAL.JS por su popularidad, “extensibilidad”, facilidad de uso y amplia documentación.

**DJANGO** es un sistema en Python para la gestión de sitios web con bases de datos, basado en la arquitectura modelo-plantilla-vista (MTV, del inglés *model-template-views*).

**REVEAL.JS** es un conjunto de librerías de CSS y JavaScript que ofrecen una forma sencilla de crear presentaciones sobrias y elegantes, y que incluye *plug-ins* que le agregan funcionalidad.

Ambos recursos serán presentados y analizados en capítulos posteriores.

### 1.4. Objetivo general

Crear un sitio web que permita la presentación remota y sincrónica de materiales de clase y que permita el registro e interacción de usuarios.

### 1.5. Objetivos específicos

1. Investigar las posibilidades de DJANGO como plataforma (*framework*) *back end* para un sitio con bases de datos, usuarios registrados y sesiones.
2. Investigar las posibilidades de REVEAL.JS como tecnología *front end* para creación de presentaciones web e incluir varias formas de contenido multimedia.
3. Desarrollar un sitio web capaz de registrar las respuestas de usuarios registrados en presentaciones remotas, para luego reportar estos datos al administrador.
4. Crear una presentación modelo para mostrar todos los alcances del proyecto.

## 1.6. Metodología

El proyecto se divide en las siguientes actividades:

1. Estudio de DJANGO como *framework* de desarrollo web en python.
2. Estudio de bases de datos SQLite 3 para el manejo de las cuentas de datos de usuarios y respuestas de preguntas.
3. Diseño de la base de datos para el manejo de usuarios y respuestas de preguntas.
4. Estudio de REVEAL.JS como *framework* para creación de presentaciones interactivas.
5. Estudio del *plugin* Multiplex para el control remoto de presentaciones creadas con el *framework* REVEAL.JS.
6. Estudio de GitHub como alojamiento de la página web
7. Diseño de una estructura en DJANGO para el desarrollo de la página web.
8. Diseño de las plantillas a utilizar como base para las presentaciones interactivas.
9. Diseño del alojamiento (*hosting*) web con GitHub adaptado a las necesidades de presentación web.

## 1.7. Consideraciones sobre el consumo de ancho de banda

Una motivación adicional para el proyecto es el hecho (sin demostración aún) de que las videoconferencias donde se transmite voz + video + pantalla compartida consumen un ancho de banda que, en ocasiones y según la experiencia reciente de profesores y estudiantes, resulta prohibitivo para los estudiantes, y por tanto no tienen un *streaming* fluido.

La modalidad propuesta de voz + presentación en navegador web (que se carga previamente) promete ser menos demandante de recursos de conectividad. Esta es una hipótesis también por confirmar.



# PRESENTACIONES WEB PARA SESIONES VIRTUALES SINCRÓNICAS

LA RECIENTE NECESIDAD de migrar a las lecciones virtuales ha obligado a utilizar nuevas herramientas tecnológicas para compartir contenido con los estudiantes, tanto de forma sincrónica como asincrónica.

## 2.1. Contexto del curso

Este Proyecto Eléctrico se desarrolló considerando las necesidades del curso MODELOS PROBABILÍSTICOS DE SEÑALES Y SISTEMAS, y por tanto las características de su contenido y de su evaluación van a guiar las decisiones de diseño.

### 2.1.1. Materiales y evaluación del curso

Consideraciones generales:

- Los contenidos del curso se dividen en cinco temas.
- Estos temas están subdivididos en 21 presentaciones que desarrollan la teoría y muestran algunos ejemplos.
- Cada presentación tiene una práctica con ejercicios resueltos.
- Cada presentación tiene un video explicativo.

Por tanto, el contenido de las presentaciones no es una exposición de la teoría (ya cubierta en las presentaciones) sino que es, en general, un repaso breve de teoría, ejercicios para que resuelvan los estudiantes y prácticas de programación, entre otras (como se amplía en la sección 2.1.2).

Estas tres actividades pueden incluir distintos tipos e interacción, más allá de la presentación magistral.

### 2.1.2. Tipos de interacción en sesiones virtuales

El profesor Fabián Abarca Calderón, guía de este Proyecto Eléctrico, ha dividido los tipos de interacción en sesiones virtuales de la siguiente forma, en función del “flujo de información”:

- Presentar
- Interactuar
- Demostrar
- Colaborar

Esta división corresponde a la interacción entre docentes, estudiantes y “el sistema”, concebido como una *instancia* o *simulación* del objeto de estudio (pero que *no* es la teoría). La figura 2.1 ilustra esta clasificación.

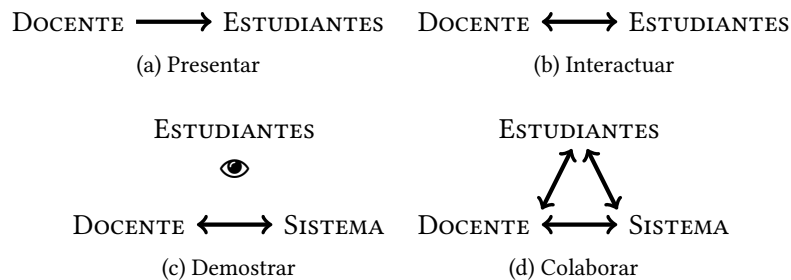


Figura 2.1: Tipos de interacción según flujos de información (representado con una flecha).

En el curso MODELOS PROBABILÍSTICOS DE SEÑALES Y SISTEMAS, a menudo “el sistema” será una simulación o análisis computacional de datos con conceptos de estadística y probabilidad.

### 2.1.3. Tipos de contenido en sesiones virtuales sincrónicas

Estos son algunos de los contenidos de interés para el curso MODELOS PROBABILÍSTICOS DE SEÑALES Y SISTEMAS:

- Información estática
- Gráficas estáticas
- Gráficas interactivas
- Animaciones
- Multimedia
- Navegación de páginas web
- Código estático
- Código interactivo

Todas estas categorías pueden ser incorporadas utilizando REVEAL.JS y DJANGO, como se demostrará en próximos capítulos.

## ENTORNO DE DESARROLLO WEB DJANGO

DJANGO ES UN *FRAMEWORK* DE DESARROLLO WEB DE *CÓDIGO ABIERTO* escrito en *Python* que fomenta un desarrollo rápido, limpio y pragmático de aplicaciones web [2].

### 3.1. ¿Que es Django?

Django es un entorno de trabajo de alto nivel para el desarrollo de aplicaciones web rápido y limpio basado en Python, que responde a problemas comunes utilizando patrones de diseño [1]. Fue creado en el 2005 por Adrian Holovaty y Simon Willison. Actualmente es desarrollado por Django Software Foundation.

Django se basa en una filosofía de “no te repitas” o DRY (*don't repeat yourself*, por sus siglas en inglés), utilizando una serie de capas que mantiene el mínimo numero de dependencias entre sí, permitiendo una sencilla reutilización de los contenidos y simplificando futuros cambios, sin impacto para los demás módulos [2].

El modelo de trabajo es una variación del modelo MVC (*Model View Controller*), llamado **MTV** (*Model Template View*) definidas como:

**Models** los modelos conforman los **datos** de la aplicación e interactúa directamente con la base de datos (creando tablas, etc.).

**Templates** son **plantillas** HTML y CSS para estandarizar la forma como se presentan los datos en el navegador.

**Views** recurre al uso de “vistas” para determinar cuáles son los datos (de los modelos) que se desean **presentar** o “inyectar” en las plantillas.

**“Aplicaciones” de DJANGO** Las aplicaciones en DJANGO son una unidad que contiene las bases de datos (modelos) y vistas específicas a una sección del sitio completo. Por ejemplo, la sección de **blog** podría ser una aplicación, la sección de ventas otra, y la sección de personal otra aplicación.

DJANGO provee la posibilidad de crear aplicaciones web de forma dinámica haciendo un uso eficiente de los recursos del servidor web.

## 3.2. Estructura de Django

La estructura que implementa Django para administrar una aplicación web es de forma jerárquica, basada en directorios y subdirectorios, que contienen los elementos que conforman la aplicación.

Para crear un proyecto en DJANGO se utiliza el comando descrito en el código 1:

```
1 $ django-admin startproject aplicacion_web
```

Código 1: Iniciar un proyecto en DJANGO. Tomado de la documentación de DJANGO [1]

Este a su vez crea un directorio raíz con el nombre de la aplicación web y provee la estructura jerárquica inicial del proyecto.

### 3.2.1. Directorio raíz de la aplicación

Al iniciar un proyecto en DJANGO se crea un directorio raíz para la aplicación web junto con los elementos indicados en la figura 3.1, exceptuando los directorios para aplicaciones específicas en DJANGO y la base de datos, ya que estos últimos, se crean según la necesidad del proyecto.

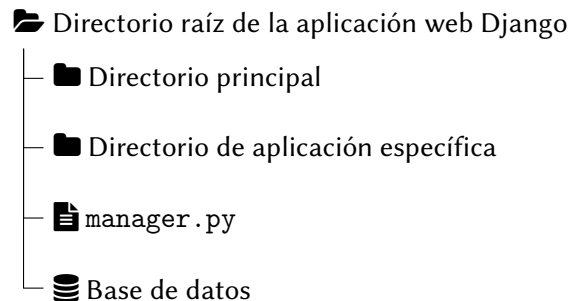


Figura 3.1: Elementos que contienen un directorio raíz en Django.

donde

**Directorio principal** contiene los archivos necesarios para la configuración de la aplicación web y su interacción con el servidor web.

**Directorio de aplicación específica** contiene los modelos de la aplicación a ejecutar y la forma en la que se presentan los datos al usuario final

**manager.py** es la utilidad de *línea de comandos* que maneja los comandos de Python para interactuar con DJANGO.

**Base de datos** la base de datos utilizada por defecto para DJANGO es SQLite, aunque puede soportar otros formatos como: PostgreSQL, Oracle, MySQL.



### 3.2.2. Directorio principal

Como se observa en la figura 3.2 el directorio principal está conformado por diferentes archivos de Python, que proporcionan la configuración general de la aplicación, la tabla de direccionamiento para las diferentes vistas de la aplicación, junto con la configuración para las comunicaciones de la aplicación con el servidor web, entre otros.

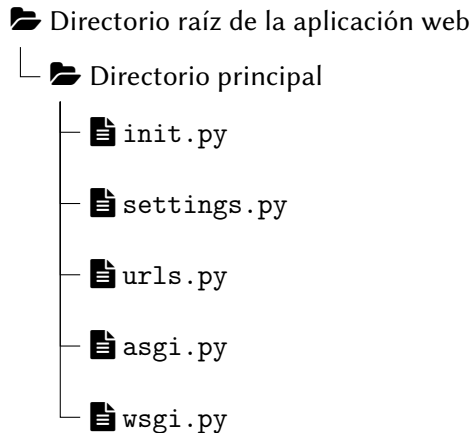


Figura 3.2: Elementos del directorio principal en una aplicación en Django.

donde

**El archivo de inicialización (`init.py`)** Es un archivo en blanco que le indica a DJANGO que la carpeta como tal es un paquete de Python a procesar.

**El archivo de direccionamiento interno (`urls.py`)** Cuenta con una dirección por defecto a la “vista” del administrador de la página web, como se muestra en la línea 5 del código 2, en este archivo se deben de colocar las direcciones de los directorios que contienen las diferentes “vistas” de las aplicaciones que se van a mostrar al usuario final.

```
1 from django.contrib import admin
2 from django.urls import path
3
4 urlpatterns = [
5     path('admin/', admin.site.urls),
6 ]
```

Código 2: Configuración inicial del archivo `urls.py` en DJANGO. Tomado de la documentación de DJANGO [1]

**El archivo de configuración (settings.py)** Se encarga de la configuración general de la aplicación web. DJANGO provee una configuración inicial básica para el entorno de desarrollo del proyecto que comprende la habilitación del proyecto DJANGO en modo depuración como se observa en la línea 2 del código 3, la instalación de las aplicaciones básicas requeridas por DJANGO para su funcionamiento, como se observa en la línea 7 del código 3, la base de datos a utilizar, como se observa en la línea 19 del código 3, por defecto SQLite3 y el directorio donde se alojan los archivos estáticos, como se observa en la línea 28 del código 3. Todo esto entre otros parámetros.

```
1  # SECURITY WARNING: don't run with debug turned on in production!
2  DEBUG = True
3  ALLOWED_HOSTS = []
4
5  # Application definition
6
7  INSTALLED_APPS = [
8      'django.contrib.admin',
9      'django.contrib.auth',
10     'django.contrib.contenttypes',
11     'django.contrib.sessions',
12     'django.contrib.messages',
13     'django.contrib.staticfiles',
14     'application.apps.ApplicationConfig',
15 ]
16
17 # Database
18 # https://docs.djangoproject.com/en/3.1/ref/settings/#databases
19
20 DATABASES = {
21     'default': {
22         'ENGINE': 'django.db.backends.sqlite3',
23         'NAME': BASE_DIR / 'db.sqlite3',
24     }
25 }
26 # Static files (CSS, JavaScript, Images)
27 # https://docs.djangoproject.com/en/3.1/howto/static-files/
28
29 STATIC_URL = '/static/'
```

Código 3: Parte del archivo settings.py en DJANGO. Tomado de la documentación de DJANGO [1]

Cabe resaltar que la línea 14 del código 3 muestra como instalar las “aplicaciones” creadas por el

desarrollador en DJANGO, las cuales se discutirán mas adelante.

**Comunicación entre la aplicación y el servidor web** Se crean dos archivos para ello, uno basado en comunicación sincrónica entre el servidor y la aplicación `wsgi.py` (por defecto) y otro para comunicaciones asíncronas `asgi.py`, dependiendo de los requerimientos del desarrollador de la aplicación.

### 3.2.3. Directorio de aplicaciones en DJANGO

Se debe crear un directorio para cada “aplicación” que se implemente en DJANGO, por medio de la utilidad de linea de comandos de DJANGO, utilizando el comando en el código 4.

```
1 $ python manage.py startapp application
```

Código 4: Crear una aplicación en DJANGO. Tomado de la documentación de DJANGO [1]

Dicho comando crea una carpeta para la “aplicación”, dentro de la cual crea por defecto la estructura básica de archivos con la cual DJANGO puede interactuar con la “aplicación”, como se observa en la figura 3.3, exceptuando los directorios `templates` y `migrations`, ya que estos se crean según la necesidad del proyecto.

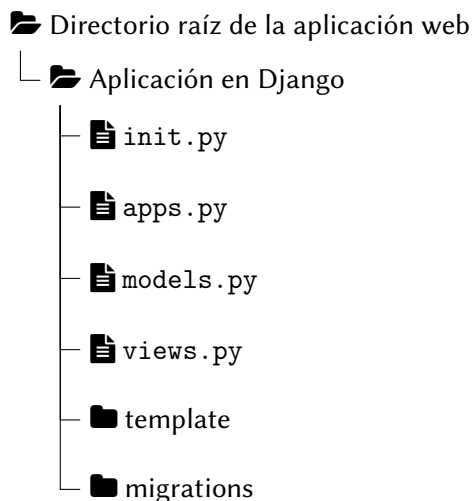


Figura 3.3: Estructura básica de una “aplicación” en Django.

**Archivo de configuración de la “aplicación” (`apps.py`)** Este define el nombre de la “aplicación” y la configuración inicial con el cual puede ser instalado en la sección de aplicaciones del archivo `settings.py` en el directorio principal, como se observa en el código 5

```
1 from django.apps import AppConfig
2
3 class ApplicationConfig(AppConfig):
4     name = 'application'
```

Código 5: Configuración básica del archivo `apps.py` en DJANGO. Tomado de la documentación de DJANGO [1]

**Archivo de modelado de la “aplicación” (`models.py`)** Se encarga de definir la estructura interna de la aplicación, junto con sus variables, por ejemplo en el código 6 se define que el modelo para la aplicación “ApplicationModel”, cuenta con dos variables, una para números enteros y otra para cadenas de caracteres, las cuales son retornadas por la “aplicación” al momento de requerirlas.

```
1 from django.db import models
2
3 # Create your models here.
4 class ApplicationModel(models.Model):
5     numero_entero = models.IntegerField(null=True)
6     cadena_caracteres = models.CharField(max_length=50)
7     def __str__(self): # Returns the object tag
8         return '{}{}'.format(self.numero_entero, self.cadena_caracteres)
```

Código 6: Configuración básica del archivo `models.py` en DJANGO. Tomado de la documentación de DJANGO [1]

**Archivo de direccionamiento (`views.py`)** Se encarga de direccionar las respuesta a las solicitudes de la aplicación, a las “vistas” que se le mostrará al usuario final, la cual esta en formato HTML, como se muestra en la línea 5 del código 7, donde se retorna el contenido de `ApplicationView.html` como respuesta de la “aplicación” solicitada.

```
1 from django.shortcuts import render
2
3 # Create your views here.
4 def ApplicationView(request):
5     return render(request, 'ApplicationView.html')
```

Código 7: Configuración básica del archivo `views.py` en DJANGO. Tomado de la documentación de DJANGO [1]

En este archivo es donde se puede “inyectar” la información en las plantillas, utilizando toda la funcionalidad de Python para manipular los datos “llamados” desde la base de datos (según los modelos especificados).

**Directorio (template)** Comprende los archivos HTML correspondientes a las “aplicaciones” creadas por el desarrollar.

Las plantillas son documentos HTML que pueden estilizarse de cualquier forma con los archivos CSS correspondientes. Para incorporar información específica de la aplicación, DJANGO ofrece una serie de herramientas útiles que modifican el código HTML, como controles de flujo, en este caso un if:

```
1 {% if lista_estudiantes %}
2     <p>Número de estudiantes: {{ lista_estudiantes|length }}</p>
3 {% elif len(lista_estudiantes) == 1 %}
4     <h2>¡Solo queda un estudiante!</h2>
5 {% else %}
6     <h3>Ya no quedan estudiantes.</h3>
7 {% endif %}
```

Código 8: Ejemplo de control de flujo en una plantilla HTML con comandos de DJANGO.

Puede observarse que en general los comandos de DJANGO dentro de la plantilla están identificados con {% comando %} y las variables con {{ variable }}.

Hay una amplia lista de utilidades disponible en [la página oficial](#).

**Directorio migrations** Comprende las migraciones de las “aplicaciones” creadas por el desarrollar, las cuales se mencionarán mas adelante.

### 3.3. Manejo de la aplicación en DJANGO

Una vez definido la estructura para la aplicación web con DJANGO y creadas las “aplicaciones” internas del proyecto, se debe de proceder a realizar una migración de la tabla de “aplicaciones”, contenida en el archivo de configuración `settings.py`, hacia la base de datos a utilizar, utilizando el comando descrito en el código 9 para generar las migraciones de las “aplicaciones” internas del proyecto y utilizar el comando descrito en el código 10 para crear la tabla de las “aplicaciones” en la base de datos.

```
1 $ python manage.py makemigrations application
```

Código 9: Crear las migraciones para las nuevas aplicaciones. Tomado de la documentación de DJANGO [1]

```
1 $ python manage.py migrate
```

Código 10: Hace la migración a la base de datos a utilizar. Tomado de la documentación de DJANGO [1]

Seguido a esto se debe de crear un super usuario para administrar los diferentes componentes de la aplicación web, utilizando el comando descrito en el código 11.

```
1 $ python manage.py createsuperuser
```

Código 11: Crear un super usuario en DJANGO. Tomado de la documentación de DJANGO [1]

DJANGO cuenta con un la posibilidad de interactuar directamente con la linea de comandos de python por medio del comando descrito en el código 12, con el cual se tiene acceso a todas las variables de entorno y estructura de Python que utiliza DJANGO.

```
1 $ python manage.py shell
```

Código 12: Interactuar con la línea de comandos de Python desde DJANGO. Tomado de la documentación de DJANGO [1]

Finalmente se cuenta con comandos para desplegar la aplicación web, utilizando un servidor local en el navegador web del sistema operativo a utilizar, como por ejemplo el comando descrito en el código 13, el cual despliega la aplicación web en un *Local host* utilizando la dirección <http://127.0.0.1:8000>

```
1 $ python manage.py runserver
```

Código 13: Desplegar aplicación web de DJANGO en un navegador web de forma local. Tomado de la documentación de DJANGO [1]

A su vez DJANGO cuenta con una extensa documentación sobre los diferentes módulos, atributos y opciones que ofrece para el desarrollo de una aplicación web, disponible en [la pagina oficial](#)

## ENTORNO DE PRESENTACIONES WEB

# REVEAL.JS

LAS ÚLTIMAS VERSIONES DE HTML, CSS Y JAVASCRIPT OFRECEN UNA GRAN VERSATILIDAD para la presentación de información en un navegador *web*. Esto ya no solo como información estática, sino que también de forma interactiva y con apariencia adaptable de forma dinámica a pantallas grandes y pequeñas, lo que representa una ventaja sobre otros formatos.

REVEAL.JS es un *framework* popular para crear presentaciones con características típicas de un software de presentación, creado por Hakim El Hattab y una comunidad creciente. Permite [5]:

- Personalización de diapositivas.
- Animación de diapositivas.
- Transición de diapositivas.
- Uso de multimedia (imágenes y videos).

La ventaja, claro está, es que permite utilizar todo el poder del desarrollo *web* y ser combinado con otras tecnologías, como DJANGO en el *back end*, o librerías de visualización de gráficas o nuevas herramientas de programación (de particular interés en la carrera) en el *front end*, entre otros.

### 4.1. Características de REVEAL.JS

El entorno de trabajo, o *framework*, para presentaciones REVEAL.JS, ofrece la posibilidad de crear presentaciones *web*, de forma gratuita y sencilla utilizando las ventajas de los estilos **CSS** y con la personalización deseada por medio de **JavaScript** [5].

REVEAL.JS puede albergar diferentes tipos de contenidos dentro de las presentaciones, como:

- Contenido *markup* (HTML)
- Contenido [Markdown](#)
- Contenido multimedia
- Código de lenguajes de programación
- Ecuaciones matemáticas
- Numeración de transiciones
- Temporización de transiciones
- Enlaces a páginas *web*

El contenido de una presentación de REVEAL.JS por defecto es editado de la forma WYSIWYM (*what you see is what you mean*), por ser un archivo HTML *per se*. [Slides.com](#) es una alternativa WYSIWYG.

A su vez, REVEAL.JS cuenta con características especiales que lo diferencian de una presentación convencional, dentro de las cuales destaca lo siguiente [5]:

- Transiciones verticales.
- Transiciones auto-animadas.
- Modo de vista general de la diapositiva.
- Adición y reproducción de notas.
- Descarga de la presentación en formato PDF.
- Control remoto de la presentación.

Tiene también una serie de *plugins*, escritos en JavaScript, dentro de su repositorio con el cual se agregan funcionalidades adicionales como las siguientes [5]:

**Reveal Highlight:** Proporciona la posibilidad de remarcar secciones de texto en la presentación.

**Reveal Markdown:** Proporciona la posibilidad de escribir contenido Markdown en la presentación.

**Reveal Notes:** Proporciona la posibilidad de agregar notas a la presentación.

**Reveal Math:** Proporciona la posibilidad de incluir ecuaciones matemáticas en la presentación.

**Reveal Zoom:** Proporciona la posibilidad de acercarse a elementos específicos de la presentación.

A su vez cuenta con la posibilidad de instalar *plugins* diseñados para REVEAL.JS que se encuentran fuera de su repositorio principal, como por ejemplo uno que será fundamental para el desarrollo del proyecto:

**Reveal Multiplex:** Proporciona la posibilidad de control remoto de la presentación.



Figura 4.1: Logo de REVEAL.JS.

## 4.2. Estructura de REVEAL.JS

La estructura general de REVEAL.JS está dividida en tres secciones: La cabecera del archivo HTML, el cuerpo del archivo HTML y la secuencia de inicialización del *framework*.

**La cabecera del archivo HTML** Está conformado por los archivos de configuración básica de REVEAL.JS, los cuales son `reveal.ccs` y `reset.css` y los archivos de configuración de los temas para la página *web*, como por ejemplo el tema `black.css` para la página *web* y el tema `monokai.css` para el resaltado de código, observados en las líneas 9 y 10 del código 14.



```
1  <head>
2    <meta charset="utf-8">
3    <meta name="viewport" content="width=device-width, initial-scale=1.0,
      ↪ maximum-scale=1.0, user-scalable=no">
4
5    <title>Aplicación Web</title>
6
7    <link rel="stylesheet" href="dist/reset.css">
8    <link rel="stylesheet" href="dist/reveal.css">
9    <link rel="stylesheet" href="dist/theme/black.css" id="theme">
10   <!-- Theme used for syntax highlighted code -->
11   <link rel="stylesheet" href="plugin/highlight/monokai.css"
      ↪ id="highlight-theme">
12 </head>
13
```

Código 14: Estructura REVEAL.JS para la cabecera del archivos HTML. Tomado de la documentación de REVEAL.JS [5]

**El cuerpo del archivo HTML** Está definido por una clase `reveal`, la cual se encarga de establecer las características del *framework* REVEAL.JS. Dentro de dicha clase se define una clase `slides`, la cual contiene la estructura general de la presentación y por último cada diapositivas es separada por medio de un elemento de sección o `<section>` en código HTML, proporcionando en un solo archivo HTML la posibilidad de crear una presentación completa, como se observa en el código 15.

```
1 <div class="reveal">
2   <div class="slides">
3     <section>
4       <h2>Primera diapositiva</h2>
5       <p>Contenido de la primera diapositiva.</p>
6     </section>
7
8     <section>
9       <h2>Segunda diapositiva</h2>
10      <p>Contenido de la segunda diapositiva.</p>
11    </section>
12  </div>
13 </div>
```

Código 15: Ejemplo de estructura de REVEAL.JS para el cuerpo del archivo HTML. Tomado de la documentación de REVEAL.JS [5]

**Secuencia de inicialización de REVEAL.JS** Se realiza dentro del cuerpo del código HTML, utilizando los elementos `<script>` del código HTML, para definir los directorios donde se localizan los *plugins* de REVEAL.JS que se van a utilizar, como se observa en las líneas 2 a 5 del código 16, seguido por el *script* de inicialización de REVEAL.JS con las dependencias de los *plugins* antes mencionados, como se observa en la línea 14 del código 16.

```
1 <body>
2   <script src="dist/reveal.js"></script>
3   <script src="plugin/notes/notes.js"></script>
4   <script src="plugin/markdown/markdown.js"></script>
5   <script src="plugin/highlight/highlight.js"></script>
6   <script>
7     // More info about initialization & config:
8     // - https://revealjs.com/initialization/
9     // - https://revealjs.com/config/
10    Reveal.initialize({
11      hash: true,
12    },
13    // Learn about plugins: https://revealjs.com/plugins/
14    plugins: [ RevealMarkdown, RevealHighlight, RevealNotes ]
15  );
16 </script>
17 </body>
```

Código 16: Secuencia de inicialización de REVEAL.JS. Tomado de la documentación de REVEAL.JS [5]

### 4.3. Instalación de REVEAL.JS

Para la instalación básica se puede descargar la última versión de [REVEAL.JS](#) con el cual se puede almacenar los archivos de configuración CSS y JavaScript necesarios para el funcionamiento de REVEAL.JS de forma local o en el servidor *web* a utilizar.

También en REVEAL.JS se puede hacer uso de redes de entrega de contenido, o **CDN** (*Content Delivery Network*), con el cual se puede hacer uso de archivos de configuración de REVEAL.JS sin tener que alojarlos localmente o en el servidor remoto, sino que se acceden en otro servidor. Sin embargo, esto no permite la modificación de los archivos, que sería necesario para adaptar a necesidades específicas. Dicho contenido de CDN puede ser consultado en [la página oficial](#) de alojamiento CDN de REVEAL.JS.

A su vez, REVEAL.JS cuenta con una extensa documentación sobre sus diferentes contenidos, características, personalización y aplicaciones que ofrece para el desarrollo de una presentación *web*, la cual puede ser consultada en [la página oficial de REVEAL.JS](#)

### 4.4. Reveal Multiplex

Una característica necesaria para la aplicación *web* prevista es la de tener control de las transiciones de diapositivas de forma remota. Para esto, el *plugin* **Multiplex** facilita la implementación del control remoto de las diapositivas *web* creadas con REVEAL.JS. Su estructura general está en la figura 4.2.



Figura 4.2: Diagrama de comunicación de Multiplex.

Para utilizar el *plugin* [Multiplex](#) se requiere lo siguiente:

**Una presentación maestra** Que es la que controla las transiciones entre las diapositivas de la presentación en los navegadores de los usuarios.

**Una presentación cliente** Que es la que va a seguir el flujo de las transiciones ejecutadas por la presentación maestra. También tiene la posibilidad de navegación autónoma, que representa una característica deseable, según la retroalimentación de estudiantes.

**Un servidor** El cual se encarga de transmitir los **eventos** (transiciones de diapositivas, en este caso) desde la presentación maestra a la presentación cliente.

#### 4.4.1. Estructura de Multiplex

En cuanto a su estructura, al ser un *plugin* de REVEAL.JS se debe inicializar como un elemento del *script* de REVEAL.JS, como se indicó en la sección anterior e inicializar los siguientes elementos:

**Un elemento *secret*** que almacena un identificador para la presentación maestra, con la cual el servidor *Socket.io* se asegura de establecer el control solo de la Presentación Maestra, este dato es suministrado por el servidor *Socket.io*; en el caso de la presentación maestra, es un número entero y en el caso de la Presentación Cliente, es un valor nulo, como se observa en la línea 4 del código 17.

**Un elemento *id*** que almacena el identificador de la comunicación, el cual sincroniza la presentación maestra con la presentación cliente, dicho identificador es proporcionado por el servidor *Socket.io* y debe ser igual en ambas presentaciones, como se observa en la línea 4 del código 17.

**Un elemento *url*** que almacena la dirección del servidor *Socket.io*, como se observa en la línea 6 del código 17.

A su vez se debe indicar las dependencias que requerirá cada presentación como aparece a partir de la línea 8 del código 17, donde para cada presentación se debe agregar la dirección del *plugin* de *Socket.io* y la dirección correspondiente del *plugin* de la presentación maestra o cliente según corresponda.

```

1 <script>
2   Reveal.initialize({
3     multiplex: {
4       secret: null, // null so the clients do not have control of the
                     ↪ master presentation
5       id: '1ea875674b17ca76', // id, obtained from socket.io server
6       url: 'https://reveal-multiplex.glitch.me/' // Location of socket.io
                     ↪ server
7     },
8     dependencies: [
9       { src: 'https://reveal-multiplex.glitch.me/socket.io/socket.io.js',
10         ↪ async: true },
11       { src: 'https://reveal-multiplex.glitch.me/master.js', async: true }
12       { src: 'https://reveal-multiplex.glitch.me/client.js', async: true }
13     ]
14   });
15 </script>

```

Código 17: Ejemplo de la estructura del *plugin* Multiplex. Tomado de la documentación de Reveal-Multiplex. [6]

#### 4.4.2. Instalación de Multiplex

El *plugin* Multiplex se debe instalar en la carpeta correspondiente REVEAL.JS, utilizando el comando descrito en el código 18 y luego activar el módulo de nodos para el Multiplex con el comando descrito en el código 19. Dicha instalación se puede realizar de forma local o en el servidor a utilizar.

```
1 $ npm install reveal-multiplex
```

Código 18: Instalación del *plugin* Multiplex de REVEAL.JS. Tomado de la documentación de Reveal-Multiplex. [6]

```
1 $ node node_modules/reveal-multiplex
```

Código 19: Activación del módulo de nodos para el *plugin* Multiplex de REVEAL.JS. Tomado de la documentación de Reveal-Multiplex. [6]

Durante el desarrollo de la aplicación *web*, Multiplex brinda la opción de utilizar un nodo estático para poder acceder a la presentación maestra de forma local, utilizando el comando de instalación descrito en el código 20 para la instalación y la activación del mismo, por medio del código 21.

```
1 $ npm install node-static
```

Código 20: Instalación del módulo de nodos estático para el *plugin* Multiplex de REVEAL.JS. Tomado de la documentación de Reveal-Multiplex. [6]

```
1 $ static
```

Código 21: Activación del módulo de nodos estático para el *plugin* Multiplex de REVEAL.JS. Tomado de la documentación de Reveal-Multiplex. [6]

Cabe resaltar que Reveal Multiplex pone a disposición el servidor <https://reveal-multiplex.glitch.me/> como servidor *Socket.io* de pruebas para *reveal-multiplex*, contando con los elementos de instalación del *plugin* antes descritos, pero no garantiza su estabilidad ni disponibilidad, debido a la utilización del mismo. A su vez, dicho servidor proporciona la opción de generar los elementos **secret** y **id** necesarios para establecer la comunicación entre la Presentación Maestra y la Cliente, por medio de un servicio de [Token](#)

Para una mayor información sobre el *plugin* Reveal-Multiplex acceder a [la página oficial de Reveal-Multiplex](#).

## ENTORNO DE PRODUCCIÓN PARA EL SITIO *WEB* (HEROKU)

UN ENTORNO DE PRODUCCIÓN proporciona el servidor en el que corre el sitio *web* para uso externo, en “la nube”. Un entorno de producción típico incluye lo siguiente [8]:

**Hardware** en el que la aplicación *web* va a ser ejecutado.

**Sistema operativo** en el que está basado la aplicación *web* (por ejemplo, Linux o Windows).

**Lenguaje de programación y librerías** sobre los cuales está escrita la aplicación *web*.

**Servidor *web*** que se encarga de suplir páginas *web* y contenido *web* (por ejemplo, Apache)

**Servidor de aplicaciones** que transmite las peticiones “dinámicas” entre la aplicación *web* (por ejemplo, DJANGO) y el servidor *web* utilizado.

**Base de datos** que utiliza la aplicación *web*, y donde se almacena la información del sitio.

### 5.1. Tipos de entorno de producción

Actualmente existen dos tipos de entornos de producción para aplicaciones *web*, los cuales son entornos de *infraestructura* como servicio y entornos de *plataforma* como servicio [8].

**Entorno de infraestructura como servicio** Conocidos como entornos **IaaS** (*Infrastructure as a Service*), el entorno ofrece un equipo de cómputo “virtual” con restricciones de procesamiento, memoria y almacenamiento, dependiendo del contrato del servicio, pero donde el desarrollador tiene libertad de uso de dicho hardware, por lo que debe encargarse de la instalación y mantenimiento del software base para la aplicación *web*, muy útil para aplicaciones específicas. [8]

**Entorno de plataforma como servicio** Conocidos como entornos **PaaS** (*Platform as a Service*), los cuales brindan una plataforma computacional “virtual”, la cual cuenta con una serie de restricciones de hardware y software específicos, dependiendo del nivel de contrato del servicio, liberando al desarrollador de encargarse de la instalación y mantenimiento de la plataforma de software de la aplicación *web*. [8]

Actualmente muchos entornos PaaS brindan soporte de Python como lenguaje de desarrollo para aplicaciones *web*, lo que beneficia el uso de DJANGO como *Framework* de desarrollo de aplicaciones *web*.

## 5.2. Heroku como entorno de producción

Heroku es un servicio de plataforma en la nube (*PaaS*) basado en *contenedores*, que brinda soporte para aplicaciones DJANGO. Es utilizado por desarrolladores para implementar, administrar y escalar aplicaciones *web* modernas, por medio de una plataforma elegante, flexible y fácil de usar [7].

**Ventajas de utilizar Heroku** Dentro de las principales ventajas se destaca lo siguiente [8]:

- Heroku cuenta con una licencia gratuita para la producción de aplicaciones pequeñas, lo cual facilita el desarrollo e implementación de las mismas.
- Cuenta con el apoyo del repositorio GitHub para el manejo de archivos estáticos de la aplicación *web* y el control de versiones, que facilita la trazabilidad del desarrollo del proyecto.
- Heroku proporciona una interfaz de usuario intuitiva que facilita su uso y el escalado de aplicaciones en su plataforma, conforme se desarrolla el proyecto.

**Desventajas de utilizar Heroku** Heroku cuenta con algunas desventajas para su uso, como las siguientes [8]:

- Cuenta con almacenamiento limitado, de ahí la importancia del uso de repositorios como GitHub.
- La licencia gratuita mantiene “dormida” la aplicación después de media hora de inactividad de la misma, provocando retardos durante su “despertar”.
- La licencia gratuita cuenta con un máximo de 550 horas mensuales de uso continuo (aproximadamente 22 días continuos).

Las demás limitantes se pueden conocer en [la página oficial de Heroku](#).

### 5.2.1. GitHub como repositorio para Heroku

GitHub es un repositorio gratuito para alojar proyectos en línea, utilizando un control de versiones, que facilita el avance y trazabilidad del grupo de desarrollo de la aplicación *web*. Ofrece un servicio de alojamiento para sitios *web* estáticos, que toma los archivos HTML, CSS y JavaScript directamente desde el repositorio en GitHub, ejecutando los archivos a través de un proceso de compilación y publicación del sitio *web*, como es el caso de Heroku. [3]



### 5.2.2. Descripción general de la implementación de una aplicación web en Heroku

Para implementar una aplicación web en la plataforma Heroku, primero se debe realizar ajustes en la configuración general de la aplicación web, en el caso de DJANGO se modifica el archivo `settings.py` para agregar a las variables `SECRET_KEY` y `DEBUG`, la habilitación de una llave secreta de seguridad y deshabilitar la opción de desarrollador de DJANGO que esta habilitada por defecto, como se observa en las líneas de la 5 a la 10 del código 22. A su vez, se debe agregar la versión del servidor de archivos estático recomendado por Heroku para DJANGO e incluir los directorios estáticos que lo manejarán, como se observa en el código 22 de la línea 12 a la 28. Por último se debe agregar la variable de ambiente para la base de datos utilizada en Heroku, como se muestra en el código 22 a partir de la línea 31.

Luego se debe crear una serie de archivos en la carpeta raíz de la aplicación web que contienen los elementos necesarios para la configuración del entorno y las dependencias del mismo, los archivos requeridos son [8]:

**runtime.txt** Contiene el lenguaje de programación y la versión a utilizar, en este caso Python.

**requirements.txt** Contiene las dependencias de los componentes de Python, incluyendo a DJANGO.

**procfile** Lista de procesos que han de ejecutarse para desplegar la aplicación web. En el caso de DJANGO, esto será normalmente el servidor de aplicaciones web *Gunicorn* (con un script `.wsgi`).

**wsgi.py** Configuración *WSGI* para desplegar la aplicación DJANGO en el entorno Heroku

Las dependencias requeridas dentro del archivo **requirements.txt** son:

- La versión de DJANGO que se utiliza para la aplicación web.
- La versión de la variable de entorno `dj-database-url` que maneja la base de datos de DJANGO.
- El servidor HTTP *gunicorn* para aplicaciones DJANGO.
- La versión de la base de datos que soporta Heroku, la cual es *psycpg2-binary*.
- La versión del servidor de ficheros estáticos recomendado para DJANGO, el cual es *whitenoise*.

A su vez se debe crear un repositorio de la aplicación web en GitHub, donde se almacenen los archivos del directorio raíz de la aplicación, exceptuando la base de datos, ya que consume mucho espacio de almacenamiento en Github. Para mayor información sobre cuentas y manejo de Github acceder a [la página oficial de Github](#).

```

1  import os
2  from pathlib import Path
3
4  # SECURITY WARNING: keep the secret key used in production secret!
5  SECRET_KEY =
    ↪ os.environ.get('DJANGO_SECRET_KEY', '7+u$bo_@%~7ho*96~f#3iy*bhos4351t-0qa-2ix2sq855pub=')
6  # SECURITY WARNING: don't run with debug turned on in production!
7  # DEBUG = True
8  DEBUG = bool( os.environ.get('DJANGO_DEBUG', True) )
9  DJANGO_DEBUG=''
10 ALLOWED_HOSTS = ['*']
11
12 # Application definition
13 MIDDLEWARE = [
14     ...
15     'whitenoise.middleware.WhiteNoiseMiddleware',
16     ...
17 ]
18
19 # Simplified static file serving.
20 # https://warehouse.python.org/project/whitenoise/
21 STATICFILES_STORAGE = 'whitenoise.storage.CompressedManifestStaticFilesStorage'
22
23 # Static files (CSS, JavaScript, Images)
24 # The absolute path to the directory where collectstatic will collect static
    ↪ files for deployment.
25 STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
26
27 # The URL to use when referring to static files (where they will be served from)
28 STATIC_URL = '/static/'
29
30 # Heroku: Update database configuration from $DATABASE_URL.
31 import dj_database_url
32 db_from_env = dj_database_url.config(conn_max_age=500)
33 DATABASES['default'].update(db_from_env)

```

Código 22: Ejemplo de modificaciones para la puesta en marcha de la aplicación *web* en Heroku. Tomado del artículo “Desplegando DJANGO a Producción” [8]

Por ultimo se debe crear el enlace entre la aplicación web en Heroku el directorio local de la aplicación web, como se observa en el código 24, seguido por el comando “push” para enviar nuestra apli-

cación a los contenedores de Heroku y proceder al despliegue de la misma en la *web*.

```
1 $ heroku create
```

Código 23: Comando para enlazar la aplicación en Heroku con el directorio local de la aplicación. Tomado del artículo “Desplegando DJANGO a Producción” [8]

```
1 $ heroku create
```

Código 24: Comando para enlazar la aplicación en Heroku con el directorio local de la aplicación. Tomado del artículo “Desplegando DJANGO a Producción” [8]

Heroku cuenta con la posibilidad de ejecutar líneas de comando en linux de forma remota, por medio del usuario de Heroku para la administración de la aplicación web, como por ejemplo en comando 25 se observa como se Heroku ejecuta el programa *manage.py* con el cual puede actualizar la tabla de migraciones de la base de datos en el servidor de Heroku.

```
1 $ heroku run python manage.py migrate
```

Código 25: Ejemplo de uso del comando run en Heroku. [8]

A su vez Heroku cuenta con una extensa documentación sobre su arquitectura, líneas de comandos y soporte para diferentes lenguajes de programación con que cuenta, disponible en [la pagina oficial](#)



## RESULTADOS DE LA IMPLEMENTACIÓN WEB

EL OBJETIVO DE ESTE PROYECTO ELÉCTRICO es la implementación de un sitio *web* funcional de ejemplo, utilizando DJANGO, REVEAL.JS y Heroku. Algunos de los detalles más importantes de la puesta en marcha se muestran a continuación.

### 6.1. Implementación del proyecto en DJANGO

#### 6.1.1. Dependencias

Se realiza una instalación de las diferentes dependencias tanto para el desarrollo de la página web de forma local, como para la puesta en marcha en un servidor web.

**Dependencias para desarrollo local** El sistema operativo utilizado para el desarrollo de la aplicación web en forma local es un Ubuntu 16.04.7 LTS y se procede a instalar las siguientes dependencias:

**Python:** versión 3.6.12

**Django:** versión 3.1.1

**SQLite:** version 3

#### 6.1.2. Estructura de Django a implementar

La estructura general de la aplicación web a desarrollar consta de un directorio general de la aplicación *web*, dos directorios para aplicaciones internas de DJANGO y un directorio para los archivos estáticos.

#### 6.1.3. Directorio raíz de la aplicación

El directorio raíz para la aplicación web se definió con el nombre de `electric_project`, dentro del cual se encuentra el directorio principal de la aplicación, llamado con el mismo nombre, el directorio para el manejo de las aplicaciones de usuarios, *users*, el directorio para el manejo de las aplicaciones de presentaciones, *presentation* y el directorio para el manejo de archivos estáticos, *static*.

A su vez, se encuentra la base de datos SQLite para el proceso de desarrollo local, la utilidad de línea de comandos de DJANGO y los archivos de configuración general para la puesta en marcha de la aplicación en la plataforma Heroku, indicados en los capítulos anteriores, como se muestra en la figura 6.1.

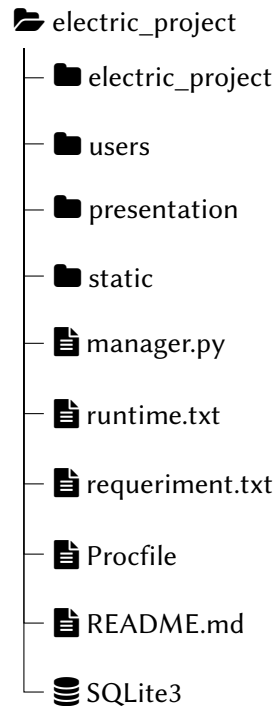


Figura 6.1: Elementos que contienen un directorio raíz del proyecto *web*

#### 6.1.4. Directorio de aplicación Principal

La estructura del directorio de la aplicación principal en el proyecto está conformada por los mismos archivos descritos en la sección Directorio principal del capítulo 3, antes mencionado, como se observa en la figura 6.2:

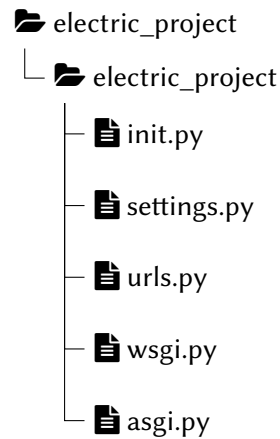


Figura 6.2: Elementos que contienen el directorio principal del proyecto *web*

**En el archivo de configuración (*settings.py*)** se instalan las aplicaciones de presentación de diapositivas y de manejo de usuarios, como se indica en las líneas 10 y 11 de la sección de código 26 del archivo de configuración.

```
1  # Application definition
2
3  INSTALLED_APPS = [
4      'django.contrib.admin',
5      'django.contrib.auth',
6      'django.contrib.contenttypes',
7      'django.contrib.sessions',
8      'django.contrib.messages',
9      'django.contrib.staticfiles',
10     'presentation.apps.PresentationConfig',
11     'users.apps.UsersConfig',
12 ]
```

Código 26: Configuración de archivo `settings.py` para nuestra aplicación *web*.

**En el archivo de direccionamiento interno (*urls.py*)** Se establece que la ruta por defecto de la aplicación *web* apunta hacia la “vista” de la página de inicio (`home_page`) y a su vez se indica la ruta de la página de inicio (`home_page/`) que apunta a su propia “vista”, como se observa en la línea 2 y 3 del código 27.

```
1 urlpatterns = [  
2     path('', views.home_page),  
3     path('home_page/', views.home_page),  
4     path('login/', views.login),  
5     path('logout/', views.logout),  
6     path('register/', views.register),  
7     path('admin/', admin.site.urls),  
8     path('presentation/', include('presentation.urls'))  
9 ] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Código 27: Configuración de archivo `urls.py` para nuestra aplicación *web*.

A su vez se indican las direcciones de inicio de sesión (*login/*), fin de sesión (*logout/*), registro (*register/*) apuntando a sus respectivas “vistas”, que se encuentran dentro de la “aplicación” *users*, como se observa entre las líneas 4 y 6 del código 27.

También se puede observar el direccionamiento de la ruta *presentation/* hacia su respectiva “vista”, que se encuentra dentro del directorio de la aplicación *presentation*, como se observa en la línea 8 del código 27.

Cabe resaltar que se define una ruta para acceder al administrador de aplicación *web*, que ofrece DJANGO por defecto y la definición de la ruta de los documentos estáticos indicada en el archivo de configuración *settings.py*, como se observa en la línea 7 y 9 del fragmento de código 27, respectivamente.

**En cuanto a los demás archivos (*init.py* , *wsgi.py* y *asgi.py*)** Se utiliza la configuración básica indicada en el capítulo 3

### 6.1.5. Aplicación para Sesiones de usuario

En cuanto a la estructura de directorio para la aplicación de sesiones de usuario, es la misma descrita en la sección Directorio de aplicaciones en DJANGO del capítulo 3 antes mencionado, como se observa en la figura 6.3.



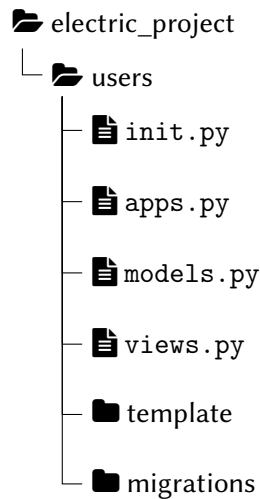


Figura 6.3: Elementos que contiene el directorio de la “aplicación” de sesiones de usuarios

**En el archivo de aplicación (*apps.py*)** Se define la “aplicación” de sesiones de usuarios con el nombre “users” como se observa en el código 28.

```
1 from django.apps import AppConfig
2 class UsersConfig(AppConfig):
3     name = 'users'
```

Código 28: Configuración del archivo `app.py` para nuestra “aplicación” *users*.

**En cuanto al archivo del modelado de la “aplicación” de sesiones de usuarios** No se modifica debido a que se importa el modelado de los formularios para creación y autenticación de usuarios que brinda DJANGO por defecto.

**En el archivo de “vistas” para la sesiones de usuarios (*views.py*)** se importan los modelos de autenticación, inicio y fin de sesión, como se observa en las líneas 3, 2 y 5 del código 29 y los formularios para la administración de usuarios, que pone a disposición DJANGO en su repositorio, indicados en la línea 4 del código 29.

```
1 from django.shortcuts import render, redirect
2 from django.contrib.auth import logout as do_logout
3 from django.contrib.auth import authenticate
4 from django.contrib.auth.forms import AuthenticationForm, UserCreationForm
5 from django.contrib.auth import login as do_login
```

Código 29: Importación de modelos para administración de usuarios por parte de DJANGO. Tomado de Implementar un sistema clásico de registro, login y logout [4]

A su vez, se definen las siguientes “vistas” como respuestas a las solicitudes de páginas HTML, realizadas por la aplicación:

**Página de Inicio de la aplicación:** Para la “vista” de la página de inicio de la aplicación de sesiones de usuario, se realiza una consulta para determinar si el usuario, que está realizando la solicitud, se encuentra autenticado, si es así, se direcciona al archivo *home\_page.html* para mostrar el contenido de la página de inicio, pero en caso contrario se direcciona la respuesta a la página de inicio de sesión, como se observa en el código 30

```
1 # Create your views here.
2 def home_page(request):
3     # si estamos identificados devolvemos al home
4     if request.user.is_authenticated:
5         return render(request, "home_page.html")
6     # En caso contrario redireccionamos a login
7     return redirect('/login')
```

Código 30: Definición de la “vista” para la página de inicio de la aplicación *users*. Tomado de Implementar un sistema clásico de registro, login y logout [4]

**Página de inicio de sesión:** Para la “vista” de la página de inicio de sesión, se utiliza el formulario de autenticación, importado anteriormente y se despliega por medio del archivo *login.html* como se observa en la línea 21 del código 31. En caso de que el usuario actualice los credenciales de autenticación, su nombre y contraseña, se procede a direccionar la “vista” actual de la página de inicio de sesión a la “vista” de la página de inicio de la aplicación, como se observa la línea 19 del código 31 .

```
1 def login(request):
2     # Creamos el formulario de autenticación vacío
3     form = AuthenticationForm()
4     if request.method == "POST":
5         # Añadimos los datos recibidos al formulario
6         form = AuthenticationForm(data=request.POST)
7         # Si el formulario es válido...
8         if form.is_valid():
9             # Recuperamos las credenciales validadas
10            username = form.cleaned_data['username']
11            password = form.cleaned_data['password']
12            # Verificamos las credenciales del usuario
13            user = authenticate(username=username, password=password)
14            # Si existe un usuario con ese nombre y contraseña
15            if user is not None:
16                # Hacemos el login manualmente
17                do_login(request, user)
18                # Y le redireccionamos a la portada
19                return redirect('/')
20            # Si llegamos al final renderizamos el formulario
21            return render(request, "login.html", {'form': form})
```

Código 31: Definición de la “vista” para el inicio de sesión de la aplicación *users*. Tomado de Implementar un sistema clásico de registro, login y logout [4]

**Página de fin de sesión:** Como respuesta a la solicitud de una “vista” de fin de sesión se direcciona a la página de inicio de la aplicación *home\_page*, pues no se desarrolla una “vista” para indicar el fin de sesión, como se observa en la línea 5 del código 32

```
1 def logout(request):
2     # Finalizamos la sesión
3     do_logout(request)
4     # Redireccionamos a la portada
5     return redirect('/')
```

Código 32: Definición de la “vista” para el fin de sesión de la aplicación *users*. Tomado de Implementar un sistema clásico de registro, login y logout [4]

**Página de registro:** Como respuesta a la solicitud de una “vista” para el registro de usuarios, se hace uso del formulario creación de usuario proporcionado por DJANGO, como se observa en la línea 3 del

código 33 y se muestra por medio de la “vista” de la página *register.html* . Una vez creado el usuario, el mismo se autentica e inicia sesión, para direccionar la “vista” actual a la “vista” de la página de inicio de la aplicación *home\_page*, como se indica en la línea 16 del código 33 . Cabe resaltar que las líneas 18, 19 y 20 del código 33 esconden el uso de texto explicativo que provee DJANGO para los formularios de registro de usuario, con el cual se presenta la información de una forma clara.

```

1 def register(request):
2     # Creamos el formulario de autenticación vacío
3     form = UserCreationForm()
4     if request.method == "POST":
5         # Añadimos los datos recibidos al formulario
6         form = UserCreationForm(data = request.POST)
7         # Si el formulario es válido...
8         if form.is_valid():
9             # Creamos la nueva cuenta de usuario
10            user = form.save()
11            # Si el usuario se crea correctamente
12            if user is not None:
13                # Hacemos el login manualmente
14                do_login(request, user)
15                # Y le redireccionamos a la portada
16                return redirect('/')
17            # Si queremos borramos los campos de ayuda
18            form.fields['username'].help_text = None
19            form.fields['password1'].help_text = None
20            form.fields['password2'].help_text = None
21            # Si llegamos al final renderizamos el formulario
22            return render(request, "register.html", {'form': form})

```

Código 33: Definición de la “vista” para el registro de usuarios en la aplicación *users*. Tomado de Implementar un sistema clásico de registro, login y logout [4]

**Dentro de la carpeta templates que contiene la aplicación users** se encuentran los archivos HTML correspondientes a las “vista” de la página de inicio, página de inicio de sesión y página de registro.

**Los demás archivos y carpetas que contiene la aplicación** Se utiliza la configuración básica indicada en la sección Directorio de aplicaciones en DJANGO del capítulo 33

### 6.1.6. Aplicación para Presentación de diapositivas

En cuanto a la estructura de directorio para la aplicación de Presentación de diapositivas, es la misma descrita en la sección Directorio de aplicaciones en DJANGO del capítulo 3 antes mencionado y se agrega un archivo *forms.py* que contiene la estructura del formulario que almacena la información de las variables de la presentación de diapositivas, como se observa en la figura 6.4.

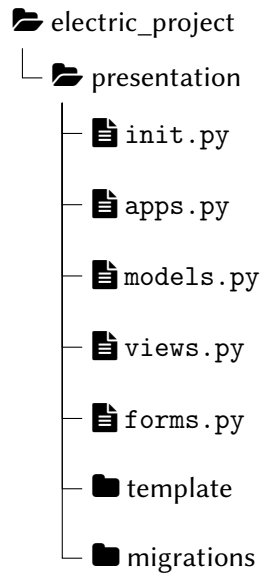


Figura 6.4: Elementos que contiene el directorio de la aplicación presentación

**En el archivo de aplicación (*apps.py*)** Se define la “aplicación” presentación de diapositiva con el nombre “presentation” como se observa en el código 34.

```
1 from django.apps import AppConfig
2 class PresentationConfig(AppConfig):
3     name = 'presentation'
```

Código 34: Configuración del archivo `app.py` para nuestra “aplicación” *presentation*.

**En cuanto al archivo del modelado de la “aplicación” presentación de diapositivas** Se crea una clase llamada selección única para definir un modelo de datos que contienen las respuestas de la interacción de usuarios, para nuestra aplicación de ejemplo se almacenan en las variables `user`, `color` y `vol`, como se observa en el código 35.

```

1 from django.db import models
2 # Create your models here.
3 class SeleccionUnica(models.Model):
4     user = models.CharField(max_length=50)
5     color = models.CharField(max_length=50)
6     vol = models.IntegerField(null=True)
7     def __str__(self): #Devuelve etiqueta del objeto
8         return '{}-{}-{}'.format(self.user, self.color, self.vol)

```

Código 35: Configuración del archivo `models.py` para nuestra “aplicación” *presentation*.

**En cuanto al archivo de “vista” para la presentación de diapositivas** Se utiliza el formulario Selección Única, que almacena las variables definidas por el modelo de la “aplicación” presentación de diapositivas, para determinar si el usuario que hace la solicitud de la página web de la presentación es administrador o no, con el fin de direccionar la “vista” a la página HTML de la presentación Maestras o Cliente, con el cual trabaja el *plugin* Multiplex de REVEAL.JS, como se observa en el código 36

```

1 from presentation.forms import SeleccionUnicaForm
2 # Create your views here.
3 def presentation(request):
4     if request.method == "POST":
5         form = SeleccionUnicaForm(request.POST)
6         if form.is_valid():
7             form.save()
8     else:
9         form = SeleccionUnicaForm()
10    if request.user.username == "admin":
11        return render(request, 'presentation_master.html', {'respuesta':form})
12    else:
13        return render(request, 'presentation.html', {'respuesta':form})

```

Código 36: Configuración del archivo `views.py` para nuestra “aplicación” *presentation*.

**En cuanto al formulario de la presentación de diapositivas *forms.py*** se define una clase Formulario de Selección Única que almacena las variables adquiridas por el modelo Selección Única. En el formulario se define los campos, las etiquetas y atributos de las variables que se van a almacenar provenientes del modelo utilizado, como se observa en el código 37.

```
1 from django import forms
2 from presentation.models import SeleccionUnica
3 class SeleccionUnicaForm(forms.ModelForm):
4
5     class Meta:
6         model = SeleccionUnica
7
8         fields = [
9             'user',
10            'color',
11            'vol',
12        ]
13        labels = {
14            'user': 'Usuario',
15            'color': 'Color',
16            'vol': 'Volumen',
17        }
18        widgets = {
19            'user': forms.TextInput(attrs={'class': 'form-control'}),
20            'color': forms.TextInput(attrs={'class': 'form-control'}),
21            'vol': forms.TextInput(attrs={'class': 'form-control'}),
22        }
```

Código 37: Configuración del archivo `forms.py` para nuestra “aplicación” *presentation*.

**Dentro de la carpeta templates que contiene la aplicación *presentation*** se encuentran los archivos HTML correspondientes a las “vista” de la página de la presentación Maestra y la página de la presentación Cliente.

**Los demás archivos y carpetas que contiene la aplicación** Se utiliza la configuración básica indicada en la sección Directorio de aplicaciones en DJANGO del capítulo 33

### 6.1.7. Directorio Static

En dicho directorio se encuentran los archivos estáticos que se utilizan para desplegar contenido informativo al usuario final, como por ejemplo imágenes, videos, código, entre otro. A su vez esta carpeta aloja la plantilla base del código HTML que se utiliza tanto para la Presentación Maestra como para la Presentación Cliente, los cuales contienen la estructura del ?? REVEAL.JS, como se observa en la figura 6.5.

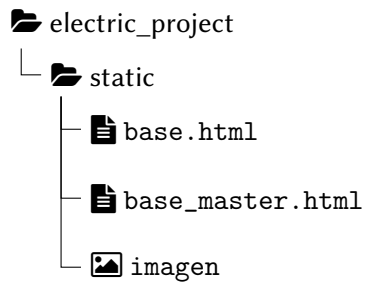


Figura 6.5: Elementos que contiene el directorio *static*

## 6.2. Implementación de REVEAL.JS

### 6.2.1. Dependencias

Para utilizar el *plugin* REVEAL.JS de forma local, se instaló las siguientes dependencias:

- Reveal.js version v4.1.0
- Node.js: versión v10.22.1

### 6.2.2. Estructura de REVEAL.JS para la aplicación web

Se utiliza la Estructura de REVEAL.JS descrita en el capítulo 4

**En cuanto a la cabecera de los archivos HTML** Se utiliza el tema *blood* y el estilo *monokai*, como se observa en el código 38. A su vez, se utiliza un repositorio de *cdnjs* para acceder a los archivos de configuración de REVEAL.JS de manera remota. Esta estructura se establece en la cabecera de los archivos *base.html* y *base\_master.html* que contienen la plantilla de REVEAL.JS a mostrar.



```

1  {% load static %}
2  ...
3  <head>
4  ...
5      <title>Prueba de reveal.js</title>
6      <link rel="stylesheet" type="text/css"
7          ↪ href="https://cdnjs.cloudflare.com/.../reveal.js/4.1.0/theme/blood.min.css"
8          ↪ id="theme">
9      <link rel="stylesheet"
10         ↪ href="https://cdnjs.cloudflare.com/.../4.1.0/plugin/highlight/monokai.min.css"
11         ↪ id="highlight-theme">-
12  ...
13 </head>
14 ...

```

Código 38: Estructura REVEAL.JS de la cabecera para lo archivos HTML a utilizar en nuestra “aplicación”. Tomado de la documentación de REVEAL.JS [5]

**En cuanto al cuerpo del archivo HTML** Se utiliza la clase reveal, junto con un indicador de contenido, para unir el cuerpo de los archivos *base.html* y *base\_master.html* con el cuerpo de los demás archivos HTML y así estandarizar la plantilla REVEAL.JS para todas las páginas *web* a mostrar, como se observa en el código 39.

```

1  <html>
2      <body>
3          <div class="reveal">
4              <div class="slides">
5                  <div class="usuario">Bienvenido </b>{{request.user.username}}</b></div>
6                  {% block content %}
7                      <!-- Aquí el contenido de cada presentación -->
8                  {% endblock %}
9              </div>
10         </div>
11         ...
12     </body>
13 </html>

```

Código 39: Estructura REVEAL.JS para el cuerpo de los archivos HTML a utilizar en nuestra “aplicación”. Tomado de la documentación de REVEAL.JS [5]

Para los demás archivos HTML se utiliza la estructura para el cuerpo de la aplicación REVEAL.JS, utilizando una extensión del archivo base que se desea utilizar y agregando los elementos del cuerpo HTML que se desea mostrar, como se indica en la figura 40, donde se utiliza la extensión del archivo *base.html* para definir lo que se va a mostrar en la página de inicio de la aplicación.

```
1 {% extends "base.html" %}
2 {% block title %} Welcome {% endblock %}
3 {% block content %}
4 <section id="home_page">
5     <h2>Proyecto Eléctrico</h2>
6     <p>
7         Bienvenido <b>{{request.user.username}}</b>.
8         Puede acceder a nuestra presentación interactiva a continuación:
9         <a href="/presentation">Primera presentacion</a>
10    </p>
11    <hr />
12    <a href="/logout">Cerrar sesión</a>
13 </section>
14 {% endblock %}
```

Código 40: Estructura REVEAL.JS para el cuerpo de los archivos HTML a utilizar en nuestra “aplicación”. Tomado de la documentación de REVEAL.JS [5]

**En cuanto a la secuencia de inicialización de REVEAL.JS** Se sigue el estándar establecido en la sección de la Estructura de REVEAL.JS indicada en el capítulo 4, agregando los *plugins* necesarios para manejo de contenido matemático, como se observa en las líneas del código 41

```

1 <html>
2   <body>
3     <script
4       ↪ src="https://cdnjs.cloudflare.com/.../4.1.0/plugin/math/math.min.js"></script>
5     <script>
6       Reveal.initialize({
7         hash: true,
8         math: {
9           mathjax:
10            ↪ 'https://cdn.jsdelivr.net/gh/mathjax/mathjax@2.7.8/MathJax.js',
11            config: 'TeX-AMS_HTML-full',
12          },
13          // Learn about plugins: https://revealjs.com/plugins/
14          plugins: [ RevealMarkdown, RevealHighlight, RevealNotes, RevealMath ]
15        });
16      </script>
17    </body>
18  </html>

```

Código 41: Estructura REVEAL.JS para la inicialización de los archivos HTML a utilizar en nuestra “aplicación”. Tomado de la documentación de REVEAL.JS [5]

## 6.3. Presentaciones remotas con Multiplex

### Estructura del documento HTML en Multiplex

Se implementa la estructura básica de Multiplex indicada en el capítulo 4, utilizando el servidor de pruebas que brinda Multiplex para nuestro proyecto. Modificando nuestros archivos base con la inicialización del *plugin* para la Presentación Maestra en el archivo *base\_master.html* y para la Presentación Cliente en el archivo *base.html* como se observa en el código 42 y 43 respectivamente.

```
1 <html>
2   <body>
3     ...
4     multiplex: {
5       // Example values. To generate your own, see the socket.io server
6       ↪ instructions.
7       secret: '16079428255931371693', // Obtained from the socket.io server. Gives
8       ↪ this (the master) control of the presentation
9       id: 'e4f58391ab15991a', // Obtained from socket.io server
10      url: 'https://reveal-multiplex.glitch.me/' // Location of socket.io server
11    },
12    // Don't forget to add the dependencies
13    dependencies: [
14      { src: 'https://reveal-multiplex.glitch.me/socket.io/socket.io.js',
15        ↪ async: true },
16      { src: 'https://reveal-multiplex.glitch.me/master.js', async: true
17        ↪ },
18    ],
19  }
20  ...
21 </body>
22 </html>
```

Código 42: Inicialización del *plugin* Multiplex de REVEAL.JS para la Presentación Maestra. Tomado de la documentación de Reveal-Multiplex. [6]

```
1 <html>
2   <body>
3     ...
4     multiplex: {
5       // Example values. To generate your own, see the socket.io server
6       ↪ instructions.
7       secret: null, // null so the clients do not have control of the master
8       ↪ presentation
9       id: 'e4f58391ab15991a', // id, obtained from socket.io server
10      url: 'https://reveal-multiplex.glitch.me/' // Location of socket.io server
11    },
12    // Don't forget to add the dependencies
13    dependencies: [
14      { src: 'https://reveal-multiplex.glitch.me/socket.io/socket.io.js',
15        ↪ async: true },
16      { src: 'https://reveal-multiplex.glitch.me/client.js', async: true },
17    ],
18  }
19  ...
20 </body>
21 </html>
```

Código 43: Inicialización del *plugin* Multiplex de REVEALJS para la Presentación Cliente. Tomado de la documentación de Reveal-Multiplex. [6]

Cabe destacar que para el desarrollo y prueba de la “aplicación” *web* de forma local se realizó la instalación de las dependencias del *plugin* Multiplex, pero para su implementación en el servidor web de Heroku, se eliminó las carpetas que contenían dichas dependencias, por un factor de espacio de almacenamiento, para los archivos estáticos en el repositorio Github, por lo que se recurre al uso del servidor *socket.io* antes mencionado.

## 6.4. Implementación del servidor web

### 6.4.1. Dependencias para la puesta en marcha en un servidor en línea

Para el uso del servidor *web* Heroku, se procede a instalar las siguientes dependencias:

- heroku: versión v7.47.6

**Dependencias para la puesta en marcha de la aplicación *web* en Heroku** En el directorio raíz se crea una serie de archivos que contienen las especificaciones del lenguaje de programación, junto

con las dependencias de DJANGO para poder desplegar la aplicación web a producción en el servidor Heroku, los archivos necesarios son los siguientes:

1. El archivo `runtime.txt` que contiene el lenguaje de programación a utilizar.

**python** version 3.7.3

2. El archivo `requirements.txt` que contiene las dependencias de Python para ejecutar la aplicación web en DJANGO:

**Django** version 3.1.1

**dj-database-url** version 0.5.0

**gunicorn** version 20.0.4

**psycpg2-binary** version 2.8.3

**whitenoise** version 5.2.0

3. El archivo `Procfile` que contiene el proceso que se van a ejecutar en el servidor HTTP *gunicorn* para la interacción con la aplicación *web*, como se observa en el código 44

```
1 $ web: gunicorn electric_project.wsgi --log-file -
```

Código 44: Contenido del archivo `Procfile`. [8]

Se realiza la configuración del archivo `settings.py` para poder desplegar a producción la aplicación y se siguen los pasos de creación de la aplicación web descritos en el capítulo 5.

### 6.4.2. Visualización de la aplicación web en Heroku

Al realizar el despliegue de la aplicación web en Heroku podemos ingresar al servidor web por medio de un explorador de internet con conectividad al mismo, por medio del siguiente enlace <https://pure-tundra-87753.herokuapp.com/> el cual es el servidor suministrado por la aplicación web para poder acceder a la misma.

Al ingresar a la aplicación web se despliega una página de inicio de sesión para la autenticación del usuario, como se observa en la figura 6.6

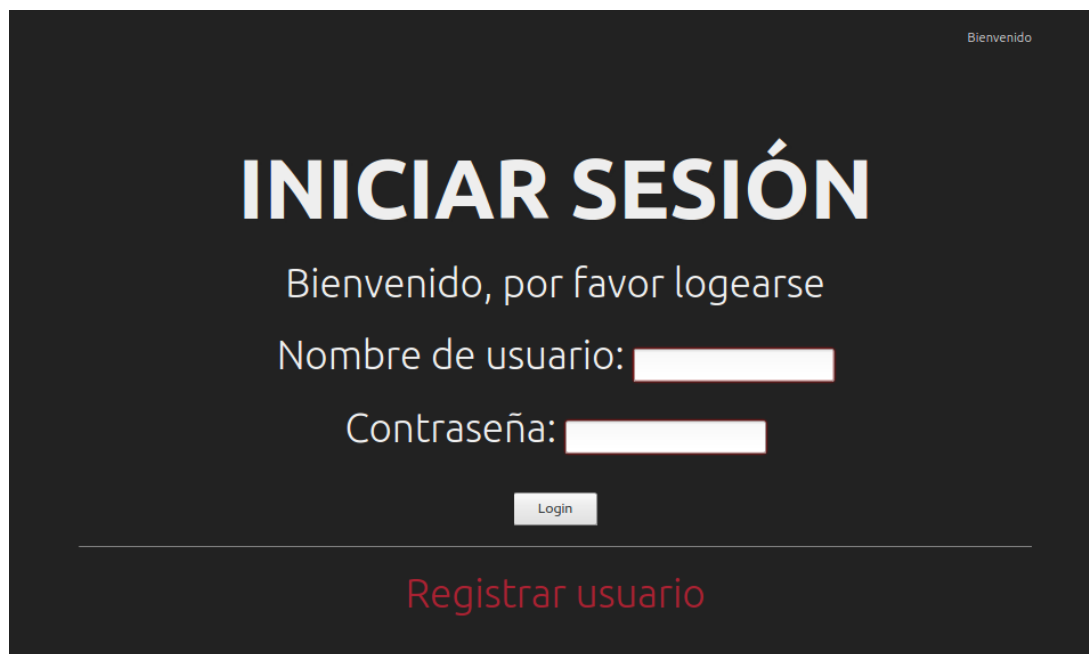


Figura 6.6: Inicio de sesión en la aplicación web de prueba

Se puede acceder al formulario de registro de usuario, como se muestra en la figura 6.7



Figura 6.7: Registro de usuarios en la aplicación web de prueba

Previamente se crearon 3 usuarios para pruebas, los cuales son:

- Usuario **admin** con la contraseña **1dm2n345678**
- Usuario **user1** con la contraseña **1s2r345678**
- Usuario **user2** con la contraseña **1s2r345678**

El usuario *admin* es el encargado de controlar la Presentación Maestra; los usuarios *user1* y *user2* solo pueden acceder a la Presentación como Clientes, con el fin de identificar el control del Multiplex para el control remoto de la presentación.

Una vez autenticado se re-direcciona a la página de inicio, dando la bienvenida al usuario y brindando la opción de acceder a la presentación de prueba, como se observa en la figura 6.8.





Figura 6.8: Página de Inicio para la aplicación web de prueba

Al acceder a la presentación interactiva podemos observar diferentes tipos de contenido relacionados al uso del *plugin* REVEAL.JS para la aplicación web de prueba, como por ejemplo en la figura 6.9 se observa un ejemplo de selección única del color favorito y en la figura 6.10 se observa el formulario que guarda la selección del color favorito por el usuario admin, para su análisis posterior.



Figura 6.9: Ejemplo de Selección única de la aplicación web de prueba



Figura 6.10: Manejo de datos adquiridos por el evento de selección única de la aplicación web de prueba

Se puede acceder al administrador de Django, como se observa en la figura 6.11, mediante el siguiente usuario y contraseña:

- Usuario **electric\_project** con la contraseña **electric\_project\_2020**



Figura 6.11: Pagina de la administración de la aplicación web de prueba

Todos los archivos que involucran el diseño y configuración de la aplicación *web* se encuentran en un repositorio de Github denominado [electric\\_project](#), en el cual pueden acceder para referencia.

## CONCLUSIONES Y RECOMENDACIONES

FINALMENTE, con el desarrollo del proyecto y los diferentes retos que se afrontaron a lo largo del mismo, se ha llegado a las siguientes conclusiones:

### 7.1. Conclusiones

- Al estudiar la estructura general de DJANGO como plataforma de (*framework*) *back end*, se diseñó una aplicación *web* capaz de manejar base de datos y controlar el flujo de información de usuarios de la aplicación, hacia ella, por medio de un administrador *web*, provisto de forma local o remota, por DJANGO.
- Al analizar DJANGO como (*framework*) manejador de sesiones de usuarios, se utilizaron los modelos y formularios provistos por DJANGO, en su repositorio general, para la creación y autenticación de los usuarios que acceden a la aplicación, de forma sencilla.
- Al estudiar a REVEAL.JS como (*framework*) *front end* para la creación de presentaciones *web*, se definió una plantilla para el diseño de presentaciones *web* que puede contener elementos multimedia, según los requerimientos del expositor.
- Al estudiar el *plugin* Multiplex de REVEAL.JS, como elemento de control para presentaciones remotas, se creó una estructura de control para presentaciones en la aplicación *web*, basada en un control maestro, sobre sesiones clientes, en tiempo real.
- Se desarrolló una aplicación *web* capaz de utilizar la versatilidad de DJANGO como (*framework*) *back end* y las bondades de REVEAL.JS como (*framework*) *front end*, en una sola aplicación *web*, mejorando el rendimiento de la página *web*, por la modularidad que ofrece DJANGO y estableciendo la estructura base para creación de Presentaciones *web* con REVEAL.JS.
- Se desarrolló una aplicación *web* capaz de contener las respuestas de usuarios, ante preguntas elaboradas dentro de la presentación *web*, por medio de una tabla de datos, administrada por DJANGO, que interactúa con la base de datos de la aplicación.

## 7.2. Recomendaciones

En caso de requerir continuar desarrollando la aplicación *web* o se desea tomar como base para un nuevo proyecto, se recomienda tomar en cuenta los siguientes aspectos:

1. Explorar la posibilidad de implementar la aplicación *web* de DJANGO, utilizando otros sistemas operativos, especializados para servidores *web*, como el caso de *Windows Server*, *Ubuntu Server*, *Red Hat*, entre otros.
2. Explorar la posibilidad de implementar la aplicación *web* de DJANGO, por medio de infraestructura de servicios *web* (*IaaS*), como el caso de Amazon Web Services *IaaS*.
3. Explorar la posibilidad de hacer una conferencia de audio en el sitio mismo de la presentación, quizá con la incorporación de plataformas como Jitsi o BigBlueButton, o incluso WebRTC, que permite crear una aplicación propia.
4. Explorar la posibilidad de incluir en la página *web* un chat grupal en el sitio mismo de la presentación, quizá con la incorporación de plataformas como JivoChat o Smartsupp, o incluso aplicaciones como Telegram o WhatsApp.

# Bibliografía

- [1] Django. The web framework for perfectionists with deadlines.|django, 2021. <https://www.djangoproject.com>.
- [2] Código Facilito. Curso django, 2021. [https://codigofacilito.com/videos/curso\\_django\\_introduccion\\_1](https://codigofacilito.com/videos/curso_django_introduccion_1).
- [3] GitHub. Trabajar con páginas de github, 2020. <https://docs.github.com/es/free-pro-team@latest/github/working-with-github-pages/>.
- [4] Héctor Costa Guzmán. Implementar un sistema clásico de registro, login y logout, 2021. <https://www.hektorprofe.net/tutorial/django-sistema-registro-login-logout>.
- [5] Hakim El Hattab. The html presentation framework.|reveal.js, 2020. <https://revealjs.com/>.
- [6] Hakim El Hattab. Multiplex plugin, 2021. <https://github.com/reveal/multiplex>.
- [7] salesforce company. Heroku, 2021. <https://www.heroku.com>.
- [8] MDN web docs. Tutorial de django parte 11: Desplegando django a producción, 2021. <https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Deployment>.