Escuela Técnica Superior de Ingeniería Informática

Grado en Ingeniería Informática

Desarrollo de una aplicación web para la gestión de los clientes de un profesional de la Podología

Development of a web application for the customers management of a Podiatry professional

Realizado por

Félix Durán Domínguez

Tutorizado por

Eduardo Guzmán de los Riscos

Departamento

Lenguaje y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, Noviembre, 2016

Fecha defensa:

El secretario del tribunal

Resumen

El principal objetivo que tiene la realización de este trabajo de fin de grado es, por un lado, plasmar los conocimientos adquiridos durante la estancia en la facultad de Informática, pero sobretodo, su finalidad es completar esos conocimientos añadiendo otros que son de gran utilidad para el ámbito profesional en la rama de las aplicaciones Web. Las tecnologías utilizadas en este proyecto han sido HTML5, CSS3, Bootstrap JavaScript, JQuery, Python (en concreto el framework Django), Angular y MySQL.

Esta aplicación tiene dos metas fundamentales. La primera es el de dar a conocer a un profesional de la Podología, el cual expondrá a sus clientes y posibles clientes su nueva aplicación para llegar a más gente. Su segunda meta es la de ofrecer una aplicación web al podólogo donde gestionar todo lo referente a sus clientes, como son su información, sus citas, calendario, mensajes y su blog de noticias.

Para el desarrollo de esta aplicación se ha utilizado una arquitectura de modelo-vista-controlador, para así separar los datos y la lógica de negocio de la interfaz del usuario teniendo siempre en mente la idea de reutilización de código y de separación de conceptos para facilitar el desarrollo de la aplicación.

Palabras Clave

- Aplicación Web
- Podología
- Profesional
- Gestión
- Cliente
- Servicios
- Salud
- Proyecto

Sumary

The main objective of the realization of this end of degree work is, on one hand, shows the acquired knowledge during the stay at the Computer Science University, but especially, it's purpose has been to complete this knowledge by adding other that are very useful in the professional field in the branch of web applications. The used technologies in this project have been HTML5, CSS3, Bootstrap, JavaScript, JQuery, Python (specifically Django framework), Angular and MySQL.

This application has two main goals. The first one is to make known to a podiatry professional, which will expose to his clients and potential clients his new application in order to reach more people. The second goal is to offer a web application to the chiropodist where he can manage everything about his clients, like their information, his events, calendar, messages, and his news blog.

For the development of this application a model-view-controller architecture has been used, in order to separate the data and the bussines logic from the user interface, always having in mind the idea of code reuse, and separation of concepts in order to ease the development of this application

Key words

- Web Application
- Podiatry
- Professional
- Management
- Client
- Service
- Health
- Project

Índice

1. Introducción

Objetivos y motivación

Este proyecto se basa fundamentalmente en los siguientes aspectos:

1. Realizar un ejercicio real de contacto con el Cliente. En este proyecto la interacción con el Cliente, ha sido continua, teniendo reuniones periódicas para discutir todos los aspectos de la aplicación, ya sean de diseño, de implementación, de datos, etc.

2. Poner en práctica lo aprendido durante el grado en Ingeniería Informática. Se han pretendido plasmar los conocimientos adquiridos durante el grado a la hora de realizar este proyecto. Metodologías, lenguajes, estructura de código y bases de datos entre otros han sido los conocimientos que me han ayudado a completar el trabajo.

3. Adquirir conocimientos nuevos. Además de utilizar como base los conocimientos adquiridos durante el grado, se ha puesto especial dedicación en aprender nuevas tecnologías y lenguajes que resultan muy útiles para el ámbito profesional. Adquirir nuevos conocimientos es de vital importancia para nuestra profesión. Estos nuevos conocimientos son: Lenguaje Python y su framework Django, HTML5, CSS3 y Bootstrap, JavaScrip, JQuery y MySQL. Estos conocimientos se han ido adquiriendo mediante la realización de cursos, la consulta en foros, la lectura de APIs, y la búsqueda de información en internet.

4. Ofrecer una herramienta útil y completa a un Cliente. Otra de las motivaciones fundamentales ha sido la de poner a disposición de un gran conocido y amigo los conocimientos y aptitudes que se han ido ganando con el tiempo y ofrecerle una herramienta útil y potente para su actividad profesional.

Métodos

Para el desarrollo de la aplicación se ha llevado una metodología incremental iterativa con unos pasos muy definidos. Lo primero ha sido estudiar los requisitos que proponía la aplicación. Una vez establecidos dichos requisitos y siempre manteniendo el contacto con el cliente se ha ido construyendo la aplicación desde sus cimientos, empezando por el diseño de las diferentes funcionalidades, pasando por la implementación de las mismas y acabando en las pruebas necesarias para asegurar el correcto funcionamiento de cada módulo. Todo esto ha sido posible gracias a una obtención previa de los conocimientos necesarios para llevar a cabo el proyecto.

Estudio de las tecnologías utilizadas

A continuación se detallarán las tecnologías que se han utilizado a la hora de desarrollar esta aplicación.

1. Lenguajes y Frameworks

Python:

Python es un lenguaje de programación interpretado y multiparadigma. Es de tipado débil, "no tipado" o de tipado dinámico ya que permite que una misma variable pueda tomar valor de distinto tipo. Se trata de un lenguaje identado y su característica más importante son la facilidad y simpleza de su uso.

Django:

Django es uno de los framework de Python por excelencia, su meta es facilitar la creación de sitios web complejos y pone especial énfasis en la re-utilización del código, el desarrollo rápido y el principio de "no te repitas". Es de código abierto y está escrito en su totalidad en Python.

HTML:

Lenguaje de marcado para el diseño del contenido de las páginas web.

CSS:

Lenguaje de diseño gráfico para crear la presentación de un documento estructurado en un lenguaje de marcado.

JavaScript:

Lenguaje de programación interpretado, orientado a objetos, basado en prototipos, imperativo y débilmente tipado. Permite mejoras en la interfaz del usuario y da dinamismo a las páginas web.

JQuery:

Biblioteca de JavaScript que añade una gran cantidad de funcionalidades a este para así mejorar su funcionalidad. Es de código abierto y cuenta con una grandísima comunidad.

Bootstrap:

Framework de código abierto desarrollado por Twitter. Contiene plantillas CSS y JavaScript para facilitar y mejorar el diseño de las páginas web. Gracias a bootstrap podemos crear interfaces de usuario que se adaptan tanto a los distintos navegadores como a las diversas plataformas.

MySQL:

Sistema de gestión de bases de datos relacionales de Oracle. Es de código abierto y es una de las más populares del mundo para el desarrollo web. Entre sus ventajas cabe destacar su baja necesidad de requerimientos, lo que la permite ser ejecutada en máquinas con pocos recursos, que es de código abierto, fácilmente configurable, y su velocidad al realizar operaciones.

2. Arquitectura M-V-C

El patrón Modelo-Vista-Controlador es una arquitectura software cuya finalidad es separar los datos y la lógica de negocio de la interfaz de usuario a la hora de crear una aplicación. Para ello define la construcción de tres componentes, los modelos (o datos), las vistas (interfaz de usuario), y los controladores (lógica de negocio).

Django permite diseñar la aplicación con una estructura Modelo-Vista-Controlador de un modo eficaz y sencillo lo que hace el desarrollo de la aplicación mucho más ameno y, por consiguiente, se obtiene una mayor satisfacción a la hora de trabajar con este entorno.

3. Hosting PythonAnyWhere

Se ha pretendido hacer un proyecto lo más parecido posible a un trabajo real, con lo cual, se decidió crear un entorno de producción para la aplicación. El problema que surgió a la hora de desplegar la aplicación a producción es que no todos los hosting web soportan Django y que, además, la forma de desplegar Django es un poco diferente, e incluso, más complicada que la que

habíamos utilizado hasta ahora. Finalmente, después de mucha búsqueda se encontró la plataforma PythonAnyWhere, una web que sirve como hosting y como repositorio de datos. Lo primero fue configurar todo el entorno, instalar Python, Django, MySQL y demás tecnologías. Lo siguiente fue alojar todo el trabajo en GitHub, ya que esta plataforma se apoya en GitHub para la gestión de los ficheros. Después, desde la consola de PythonAnyWhere se traen todos los ficheros de GitHub, se migra la base de datos y se recolectan los ficheros estáticos para que funcione todo correctamente.

Una vez hecho esta ya se dispone de la aplicación alojada a un hosting gratuito. Con esto se puede comprobar el funcionamiento de la aplicación en cualquier ordenador o dispositivo desde un entorno de producción real.

2. Formación

La formación ha desempeñado un papel fundamental a la hora de desarrollar este proyecto. Muchos de los lenguajes y tecnologías utilizadas en esta aplicación han sido aprendidas desde cero durante la primera etapa del proyecto.

Aproximadamente un 30% del tiempo empleado en la aplicación ha sido para el aprendizaje de los lenguajes y tecnologías que he empleado en la aplicación.

Las aptitudes adquiridas para la realización del proyecto son las siguientes:

- Lenguaje Python y su framework Django.
- Lenguajes JavaScrip y JQuery, así como el funcionamiento del framework Angular.
- Mejora en los conocimientos de HTML y CSS y el aprendizaje del framework Bootstrap para mejorar el diseño de las vistas.
- Base de datos MySQL en un entorno de producción real.

Además de estos conocimientos se han mejorado notablemente las aptitudes a la hora de enfrentarme a un ejercicio profesional real.

Para lograr aprender dichos conocimientos se han realizado los siguientes estudios:

- Curso de Django por la empresa SeisCocos
- Curso de Angular en la plataforma web CodeSchool
- Estudio de la API de Google
- Estudio de la documentación de DjangoProject
- Estudio de la API de JQuery
- Estudio de la documentación de HTML, CSS y JavaScript en la web w3schools
- Estudio del framework bootstrap en la plataforma librosWeb
- Estudio del despliegue de Django en la plataforma DjangoGirls

3. Análisis de la aplicación

Identificación de Usuarios

- Cliente:

  Estará representado por todas las personas que entren en la web. Constituye el grueso del negocio, los usuarios con este rol son los más importantes para el éxito de la aplicación. Ellos accederán a la aplicación en busca de información y ayuda en todo lo referente a la podología y por esto mismo la aplicación debe ser atractiva y fácil de utilizar. No es tanto un usuario con funciones en la web sino, más bien, un usuario al que se le ofrecen estas funciones, tales como consultar los servicios disponibles, comprobar la formación de nuestro podólogo o contactar con el mismo.

- Administrador/Podólogo

  Es el encargado de gestionar toda la parte administrativa de la aplicación. Se encargará de manejar toda la información de clientes, visitas, mensajes, blog y otros usuarios. En este caso estará representado por nuestro profesional en la podología, el cual será el dueño de la aplicación.

Obtención de requisitos

Para esta fase se llevaron a cabo varias reuniones con el cliente, en las cuales se definieron y analizaron los distintos requisitos que este necesitaba.

Rol Cliente

Requisito funcional 1: Acceso a la web y navegación

Este requisito es el más básico de todos, y está presente en toda la interacción con los clientes. Se trata de la carga de la web al ingresar la URL y la correcta navegación por la misma.

Requisito funcional 2: Descarga de CV

Con este requisito se cubre la necesidad de que los clientes puedan disponer del currículum del podólogo ya que esta aplicación no solo está

enfocada a usuarios que necesiten de los cuidados de un podólogo, sino, también, a otras clínicas interesadas en contratarlo.

Requisito funcional 3: Contacto con el podólogo

Los clientes podrán acceder a la web para escribir un mensaje al podólogo con cualquier contenido que se les antoje, desde pedir una cita, preguntar por un problema, etc.

Rol Administrador

Requisito funcional 4: CRUD usuarios

El administrador puede crear, consultar, modificar y eliminar usuarios del sistema desde el panel de administración.

Requisito funcional 5: CRUD clientes

El administrador puede dar de alta, consultar, modificar y eliminar clientes del sistema desde el panel de administración.

Requisito funcional 6: CRUD visitas

El administrador puede vincular visitas a sus clientes desde el panel de administración. Además también puede modificar la información de estas visitas, consultarlas y eliminarlas cuando lo necesite.

Requisito funcional 7: CRUD mensajes

Cuando un cliente escriba un mensaje al podólogo, este, además de recibir un email con la información de dicho mensaje, podrá verlo en el panel de administración, modificarlo, o eliminarlo.

Requisito funcional 8: CRUD blog

- RF-8.1 Categorías

  El administrador puede crear categorías para clasificar sus posts.

- RF-8.2 Posts

  Dentro de estas categorías el administrador creara posts que tengan que ver con ellas para mantener el orden en su blog.

Requisito funcional 9: Gestión calendario

- RF-9.1 Consultar eventos

  La aplicación permite que el administrador se autentique con su cuenta de Google para consultar sus próximos eventos directamente desde la el panel de administración.

- RF-9.2 Crear eventos

  Además, desde la aplicación el cliente puede crear nuevos eventos y añadirlos a su calendario Google

Casos de Uso: Cliente

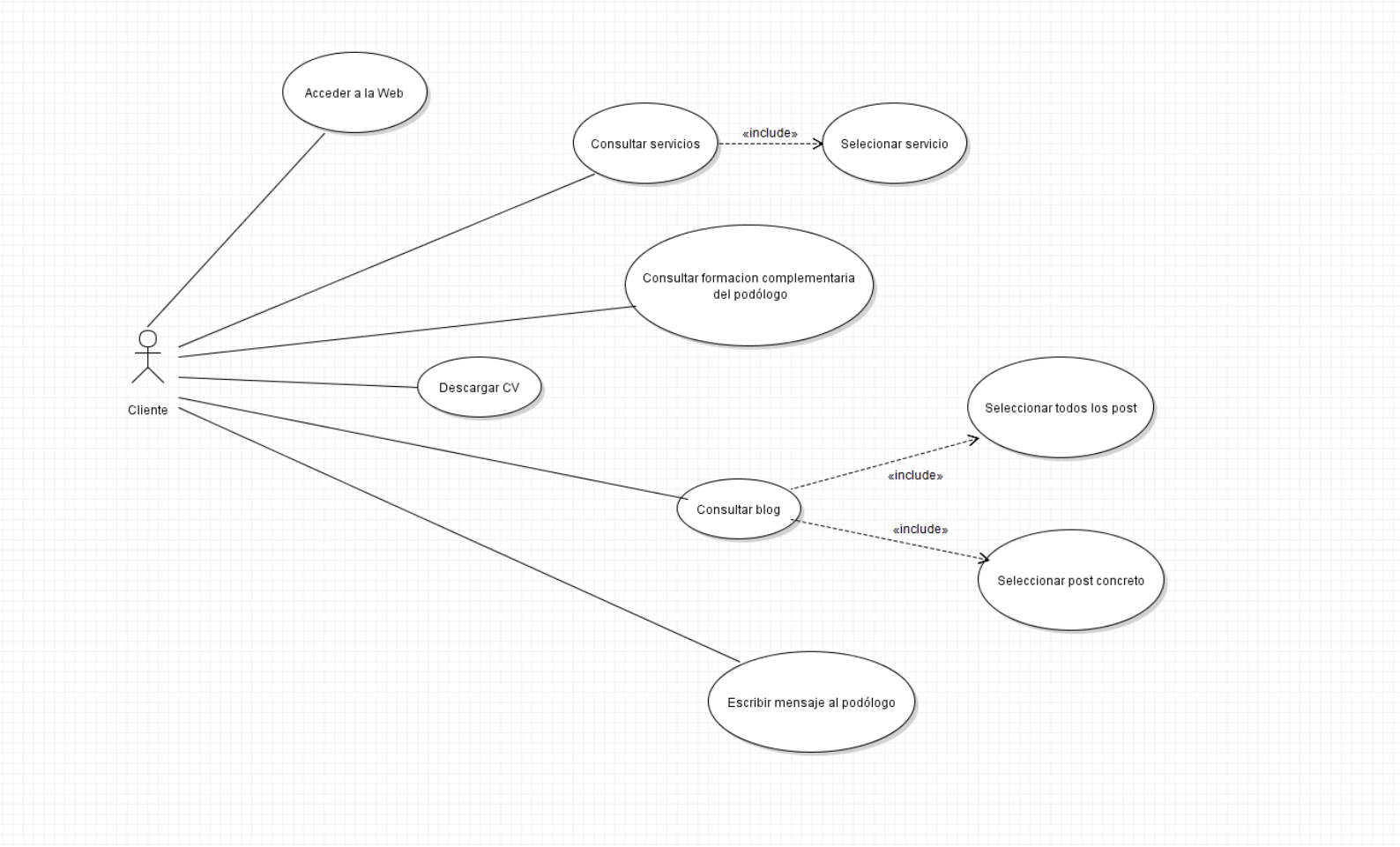


Figura 1. Casos de uso cliente

Descripción del caso de uso del cliente

En el anterior diagrama se muestran las funcionalidades que poseen los usuarios con rol cliente. El cliente accede a la web en busca de información sobre la actividad de nuestro podólogo. Puede acceder a toda la información que el podólogo le proporciona en su web, desde su formación académica, formación complementaria, servicios disponibles, curriculum, y blog de noticias. Los clientes pueden interactuar con la aplicación para ponerse en contacto con el podólogo mediante un formulario web alojado en la página.

No necesita ninguna condición previa para realizar ninguna de estas funcionalidades.

Casos de Uso: Administrador/Podólogo.

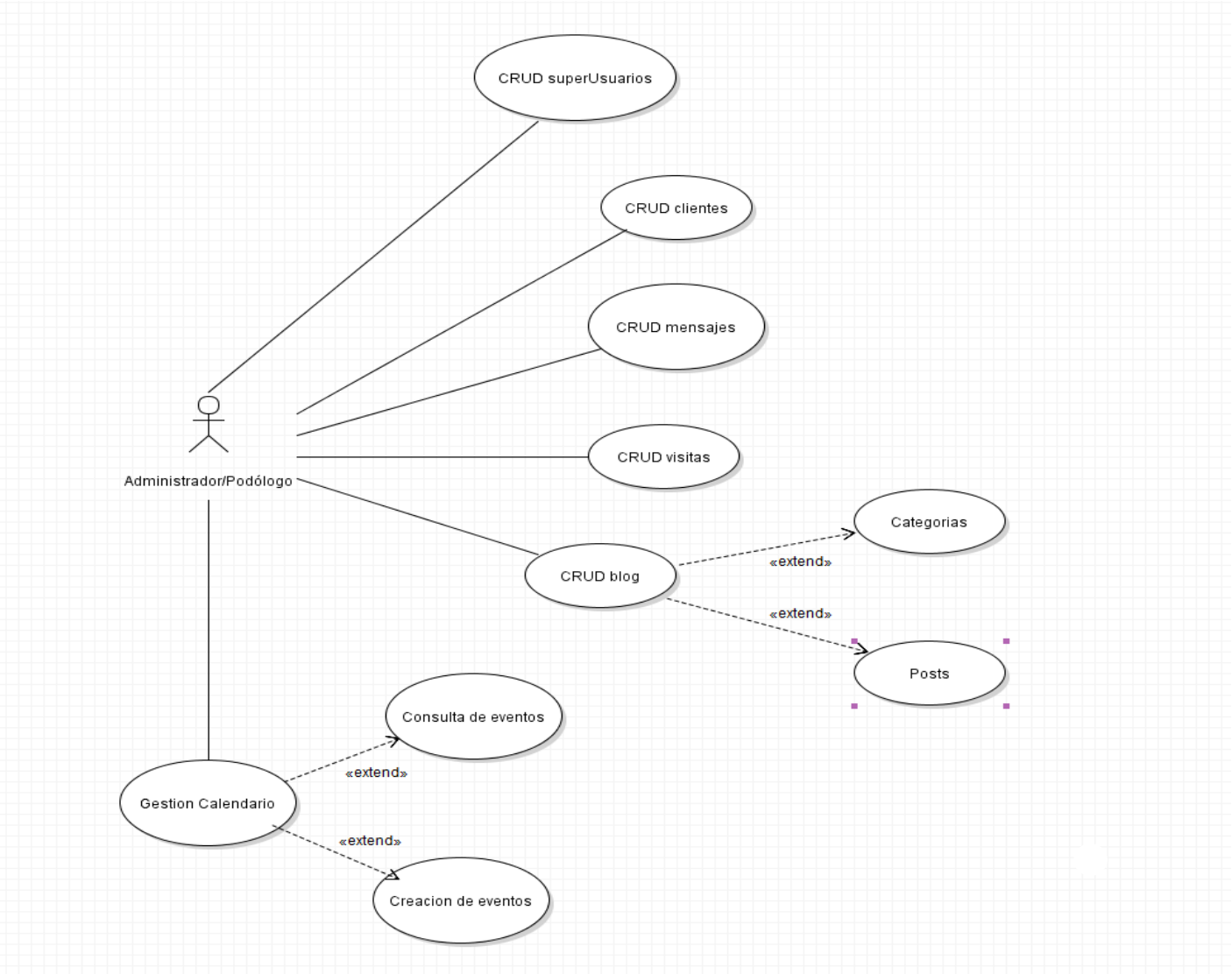


Figura 2. Casos de uso administrador

Descripción del caso de uso del administrador

En el diagrama de la figura 2 podemos comprobar las funcionalidades que posee el administrador del sistema, representado en este caso por el podólogo.

Para acceder a estas funcionalidades hay que estar previamente registrado en la aplicación. En una primera instancia solo existe un administrador del sistema, el podólogo. Este puede elegir si crear otros administradores o simplemente otros usuarios con menos permisos que puedan acceder a la aplicación.

Para los CRUD de clientes, mensajes, visitas y blog, el usuario en cuestión tiene que tener los permisos necesarios para ello. Si dicho usuario tiene los permisos, podrá realizar las tareas de creación, modificación, consulta o borrado de los datos almacenados en cada uno de estos campos.

Para la gestión del calendario personal se necesita tener una cuenta de Google, ya que el calendario está ligado a dicha cuenta.

4. Diseño del sistema

Teniendo bien definidos los requisitos, usuarios y funcionalidades a desarrollar se comienza con el diseño de la aplicación.

Lo primero es hacer un esquema de la base de datos para su posterior análisis e implementación en la aplicación.

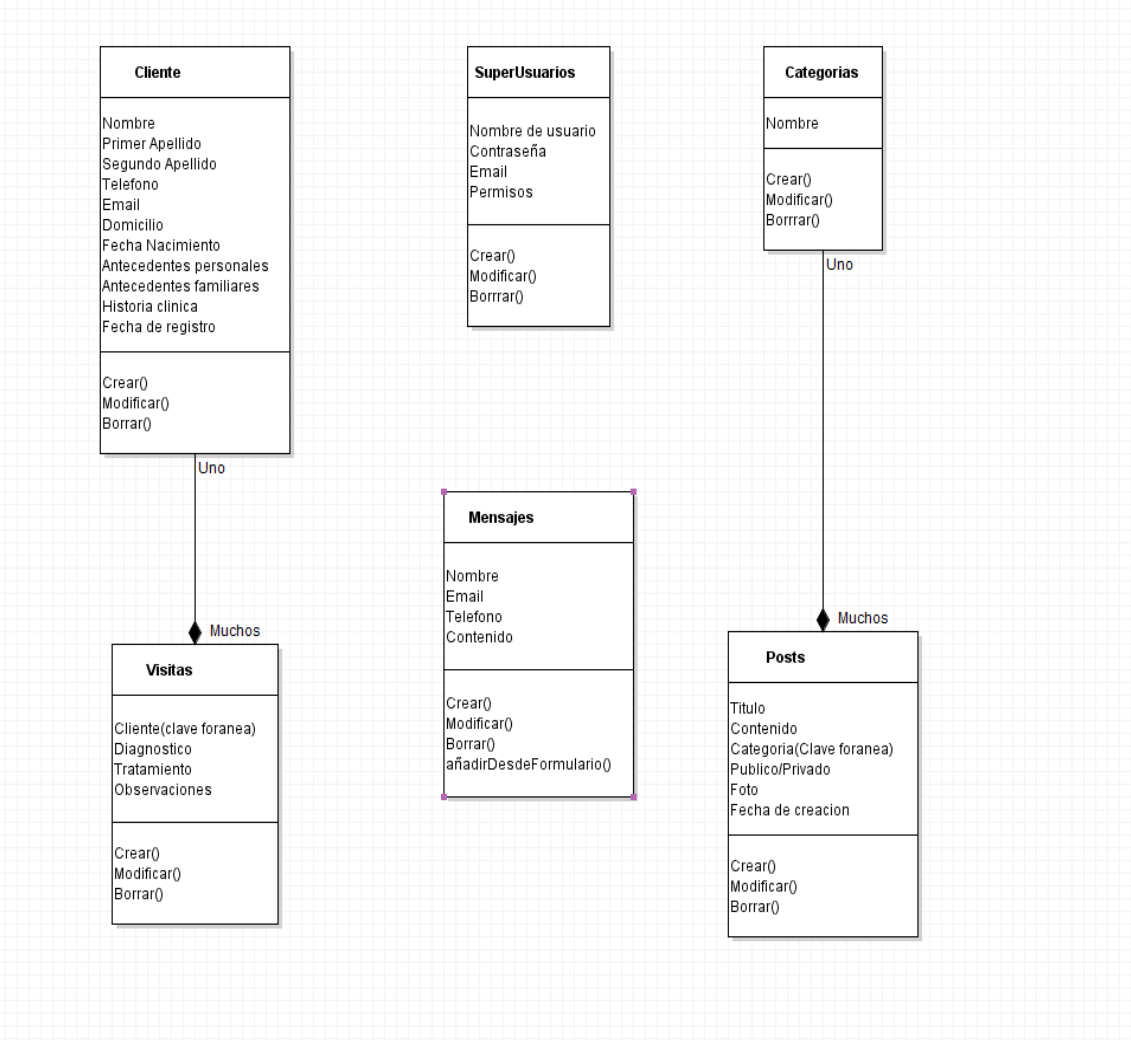


Figura 3. Diagrama de clases

Teniendo este diagrama, la creación de los modelos en Django para representar esta estructura fue lo siguiente a realizar, a continuación se muestran los módulos de código que representan cada una de estas entidades. Señalar que este código está almacenado en el fichero “models.py” el cual es el encargado de recoger las entidades en Django.

```python
class Cliente(models.Model):
    Nombre = models.CharField(max_length=200)
    Primer_Apellido = models.CharField(max_length=200)
    Segundo_Apellido = models.CharField(max_length=200)
    Telefono = models.CharField(max_length=200)
    Email = models.EmailField(max_length=200)
    Domicilio = models.CharField(max_length=200)
    Fecha_Nacimiento = models.DateField()
    Antecedentes_Personales = models.TextField()
    Antecedentes_Familiares = models.TextField()
    Historia_Clinica = models.TextField('Historia Clinica')
    Dia_registro = models.DateTimeField('Fecha')
    def __str__(self):
        return self.Nombre + " " + self.Primer_Apellido + " " + self.Segundo_Apellido
```

Figura 4. Modelo cliente

```python
class Visita(models.Model):
    Cliente = models.ForeignKey(Cliente)
    Diagnostico = models.TextField()
    Tratamiento = models.TextField()
    Observaciones = models.TextField()
    Dia = models.DateField("Dia de la visita")
```

Figura 5. Modelo visita

```python
class Mensaje(models.Model):
    Nombre = models.CharField(max_length=200)
    Email = models.CharField(max_length=200)
    Telefono = models.CharField(max_length=200)
    Mensaje = models.TextField();
```

Figura 6. Modelo mensaje

```python
class Categoria(models.Model):
    Nombre = models.CharField(max_length=200)

    def __str__(self):
        return self.Nombre
```

Figura 7. Modelo categoría

```python
class Post(models.Model):
    Titulo = models.CharField(max_length=200)
    Contenido = models.TextField()
    Categoria = models.ForeignKey(Categoria)
    Fecha_Creacion = models.DateTimeField('Fecha de creacion')
    PUBLICO = 'pub'
    PRIVADO = 'pri'
    pORp_choices = (
        (PUBLICO, 'publico'),
        (PRIVADO, 'privado'),
    )
    Publico_Privado = models.CharField(
        max_length=3,
        choices=pORp_choices,
        default=PUBLICO,
    )
    Foto = models.ImageField(upload_to='postPhotos', default = 'SOMETHING')
    def __str__(self):
        return self.Titulo
```

Figura 8. Modelo post

Esta es la representación en código de cada una de las entidades de nuestra base de datos.

En Django, los controladores están almacenados en un fichero denominado “views.py” donde se guarda la lógica de cada vista HTML. En cambio, los controladores del panel de administración son un caso especial. Django proporciona una forma de comunicación entre la vista de administración y los modelos diferente al de otros lenguajes. Existe un fichero llamado “admin.py” en el que nos encargaremos de plantear cómo deseamos que se muestren los datos en administración y Django lo hará automáticamente. Veremos a continuación los distintos campos de este archivo “admin.py” para ver cómo se estructura la información de cada entidad.

```python
class ClientesAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,              {'fields': ['Nombre','Primer_Apellido','Segundo_Apellido','Telefono','Email',
            'Domicilio','Fecha_Nacimiento','Antecedentes_Personales','Antecedentes_Familiares','Historia_Clinica']}),
        ('Fecha de Creacion', {'fields': ['Dia_registro']}),
    ]
    list_display = ('Nombre','Primer_Apellido','Segundo_Apellido','Telefono','Email',
        'Domicilio','Fecha_Nacimiento','Antecedentes_Personales','Antecedentes_Familiares','Historia_Clinica', 'Dia_registro')
    list_filter = ['Dia_registro']
    search_fields = ['Nombre']


admin.site.register(Cliente, ClientesAdmin)
```

Figura 9. Admin.py - Clientes

```
class VisitasAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,              {'fields': ['Cliente','Diagnostico','Tratamiento','Observaciones']}),
        ('Fecha', {'fields': ['Dia']}),
    ]
    list_display = ('Cliente','Diagnostico','Tratamiento', 'Observaciones', 'Dia')
    list_filter = ['Dia']
    search_fields = ['Cliente']


admin.site.register(Visita, VisitasAdmin)
```

Figura 10. Admin.py - Visitas

```
class MensajesAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,{'fields': ['Nombre','Email','Telefono','Mensaje']}),

    ]
    list_display = ('Nombre', 'Email', 'Telefono' , 'Mensaje')

    search_fields = ['Nombre']


admin.site.register(Mensaje, MensajesAdmin)
```

Figura 11. Admin.py - Mensajes

```
class PostsAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,              {'fields': ['Titulo','Contenido','Categoria','Publico_Privado','Foto']}),
        ('Date information', {'fields': ['Fecha_Creacion']}),
    ]
    list_display = ('Titulo', 'Contenido', 'Fecha_Creacion' , 'Categoria','Foto')
    list_filter = ['Fecha_Creacion']
    search_fields = ['Titulo']

admin.site.register(Post, PostsAdmin)
```

Figura 12. Admin.py - Posts

```
class CategoryAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,                {'fields': ['Nombre']})
    ]
    search_fields = ['Nombre']

admin.site.register(Categoria, CategoryAdmin)
```

Figura 13. Admin.py - Categorías

Como se observa en estas figuras, podemos elegir los campos que se mostrarán en el panel de administración, así como los filtros o campos de búsqueda para facilitar la gestión de los mismos.

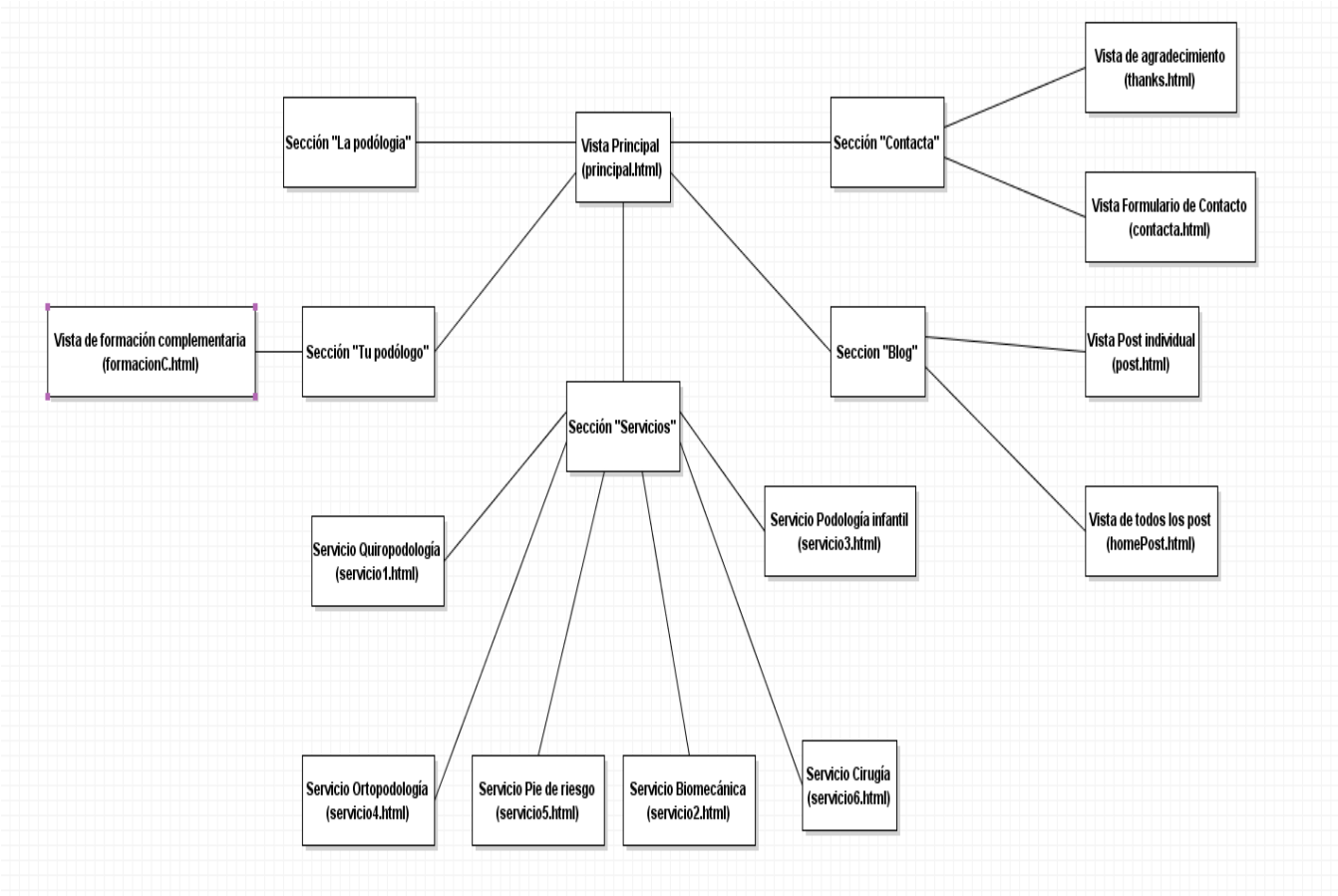


Figura 14. Diagrama de Vistas

En la figura anterior, podemos observar el diseño que se ha llevado para la estructura de las vistas de la web. Desde las distintas secciones de la vista principal se puede navegar a todas las demás. Se ha elegido este diseño ya que permite mantener toda la información básica agrupada en la misma vista y la información más especifica cómo puede ser la de los distintos servicios, o el formulario de contacto, separado de esta para un diseño más limpio y agradable.

5. Procesos de negocio

Se ha dividido el proceso de negocio en dos roles fundamentales:

Rol Usuario/Cliente

Este rol, por un lado, es el que menos participación en la aplicación tiene, pero por otro lado es el más importante, ya que es del que dependerá el éxito de la aplicación, y por consiguiente, los beneficios de nuestro podólogo.

- ### Acceso a la web:

Los clientes accederán a la web para conocer todo lo necesario acerca de nuestro podólogo. Podrán consultar su información, ver su formación, descargar su currículum, entrar en el blog, y ponerse en contacto con nuestro él.

En este caso toma especial importancia el diseño de la web, es necesario que sea una web sencilla de explorar, agradable a la vista y atractiva. Estas son las pautas con las que se ha llevado a cabo el diseño de la web. Los clientes y potenciales clientes disponen de una web con un diseño agradable, fácil y sencillo donde disponen de toda la información que necesitan con tan solo unos pocos clics.

A los clientes se les presenta la información de una manera que cualquier persona pueda acceder a ella, sin complicaciones, cualquier persona tenga o no experiencia navegando en internet encontrará la información que necesita sin ningún problema.

Rol Administrador

Este rol es el que realmente va a utilizar todo el potencial de la aplicación. Este papel lo tomará nuestro podólogo, él dispondrá de una potente herramienta donde gestionar todo lo referente a su actividad profesional.

- ### Identificación y acceso a la aplicación:

El podólogo tendrá un usuario y contraseña con los cuales podrá identificarse para acceder a su aplicación privada. Una vez dentro podrá realizar las tareas posteriormente mencionadas.

- ### Creación y gestión de usuarios:

El podólogo tomará el papel de Administrador en la aplicación, lo que le da control total, y con esto puede crear otros usuarios si lo desea para darle también control total sobre la aplicación o simplemente permitirle el acceso para la visualización de los datos.

- ### Gestión de Clientes:

La aplicación cuenta con un apartado donde crear, modificar o eliminar clientes. Se podrá almacenar toda la información necesaria de cada paciente como son Nombre, Apellidos, Teléfono, Email, Domicilio, Fecha de Nacimiento, Antecedentes Personales, Antecedentes Familiares, Historia Clínica y la fecha de registro.

También se puede consultar y actualizar la información de cualquier cliente fácilmente.

- ### Gestión de Visitas:

A los clientes anteriormente creados, se les puede asignar una serie de visitas, cada vez que un cliente acuda a la clínica o reciba un tratamiento, el podólogo creará una visita para, con esto, llevar un seguimiento de todas las visitas de cada cliente y llevar un estudio de su evolución temporal.

En este campo se guardaran los datos siguientes: Cliente, el cual se seleccionara de la lista de clientes, Diagnostico de la visita, Tratamiento y Observaciones.

- ### Consulta de Mensajes**:**

Como hemos visto en el rol cliente, estos pueden ponerse en contacto con el podólogo por medio de un mensaje. Estos mensajes llegarán al correo personal del podólogo, pero además, se irán almacenando en la aplicación para su gestión y consulta. De esta manera se consigue facilitar al podólogo la tarea de contestar las peticiones de sus clientes.

- ### Gestión de Blog:

Desde la aplicación, el podólogo podrá administrar su blog de manera sencilla. Todo los cambios que haga en esta sección se verán reflejados en la web, así, si por ejemplo decide crear un nuevo post sobre un tema que le parece interesante para sus clientes, solo tendrá que entrar en la sección de post y rellenar los campos. También podrá crear categorías para clasificar dichos posts.

- ### Gestión de su calendario personal:

En esta sección el podólogo accede a una ventana donde lo primero que deberá hacer es autenticarse con su cuenta de Google. Esto es especialmente eficiente ya que este podólogo ya utiliza su calendario de Google. Todos los cambios que el haga en su cuenta de Google se verán reflejados aquí. En esta ventana el podólogo puede consultar su calendario y sus próximos eventos, y también puede crear eventos nuevos si lo necesita. Estos eventos se añadirán a su calendario personal.

6. Casos de prueba

En este apartado comprobaremos el correcto funcionamiento de algunas funcionalidades de la aplicación desde el punto de vista de los distintos roles.

Caso 1: Acceso a la web y Navegación

Comprobar que la web carga correctamente en los distintos navegadores y la navegación es correcta.

Accedemos a la web mediante este enlace: felixdd666.pythonanywhere.com

Figura 15. Portada Web

Nos movemos con el menú de navegación, por ejemplo a servicios

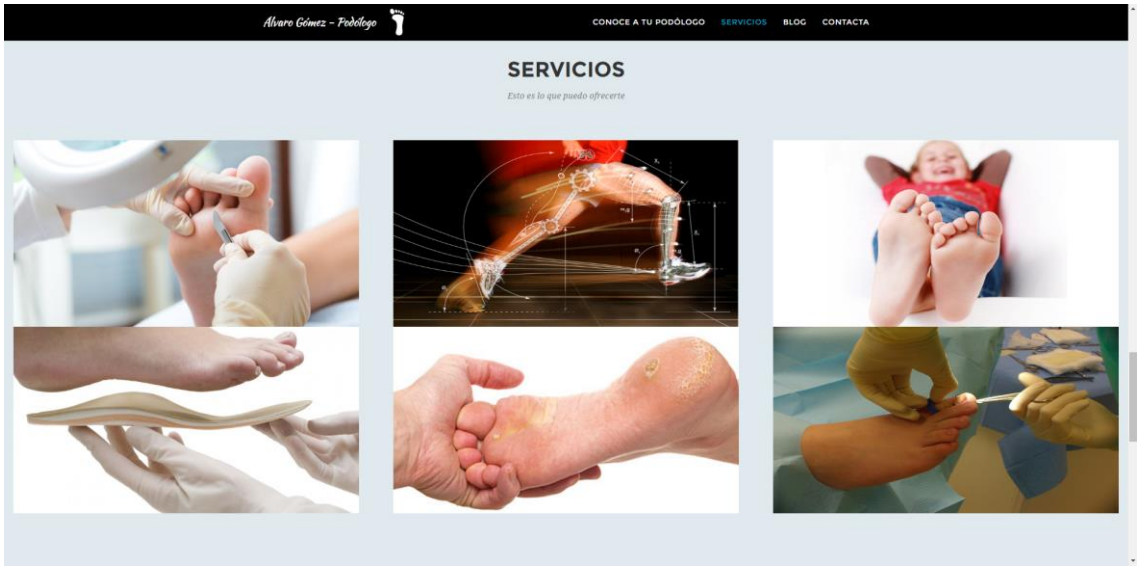


Figura 16. Sección servicios

Pinchamos en el servicio deseado, en este caso podología infantil

Figura 17. Vista podología infantil

Dentro de cada servicio podemos ver una descripción del mismo.

Caso 2: Contacto con el podólogo

Comprobar que cuando un cliente envía un mensaje al podólogo, este lo recibe en su correo y el mensaje se guarda en la aplicación.

Primero nos dirigimos al apartado de contacta de la web.

Figura 18. Sección de contacto

Pulsamos en el botón con forma de sobre y escribimos nuestro mensaje

Figura 19. Formulario de contacto

Datos introducidos:

Nombre -> Félix Durán

Email -> felixdurandom@gmail.com

Teléfono -> 635 012 355

Mensaje -> Mensaje para caso de prueba 2

Al hacer clic en enviar, deberá redirigirnos a la página de agradecimiento, enviar el email al podólogo y guardar el mensaje en la aplicación.

Mensaje de agradecimiento y botón para volver a la página principal:

Figura 20. Vista de agradecimiento

Email correspondiente al mensaje anteriormente creado:

Figura 21. E-mail a podólogo

Mensaje guardado en la base de datos:

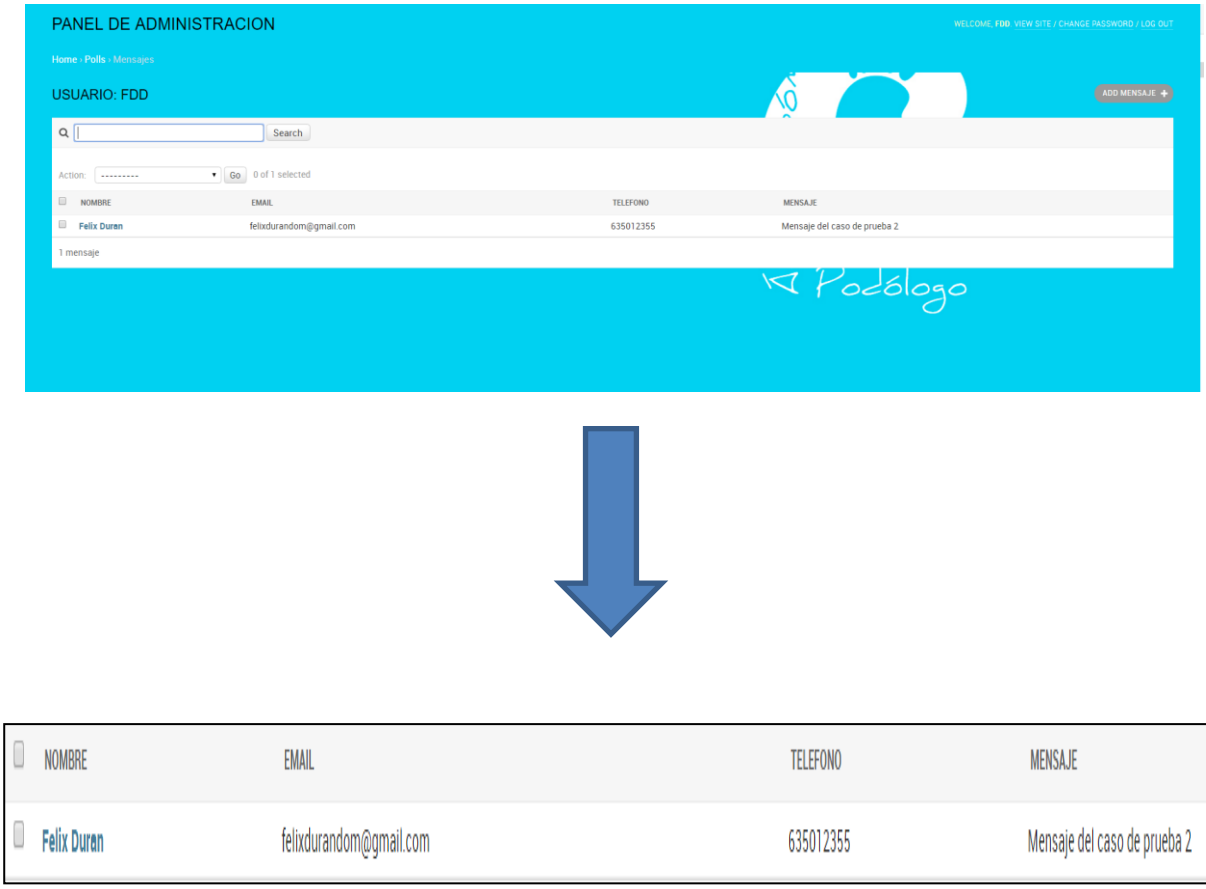


Figura 22. Mensaje almacenado

Caso 3: Acceso al panel de administración

Haciendo uso de los datos de acceso que se le proporcionan al administrador, debemos ser capaces de autenticarnos en la aplicación y entrar en nuestro panel de administración.

Accedemos a la dirección http://felixdd666.pythonanywhere.com/admin y rellenamos los campos necesarios.

Figura 23. Log-In de la aplicación

Cuando aceptamos, la aplicación debe redirigirnos a nuestro panel de administración.

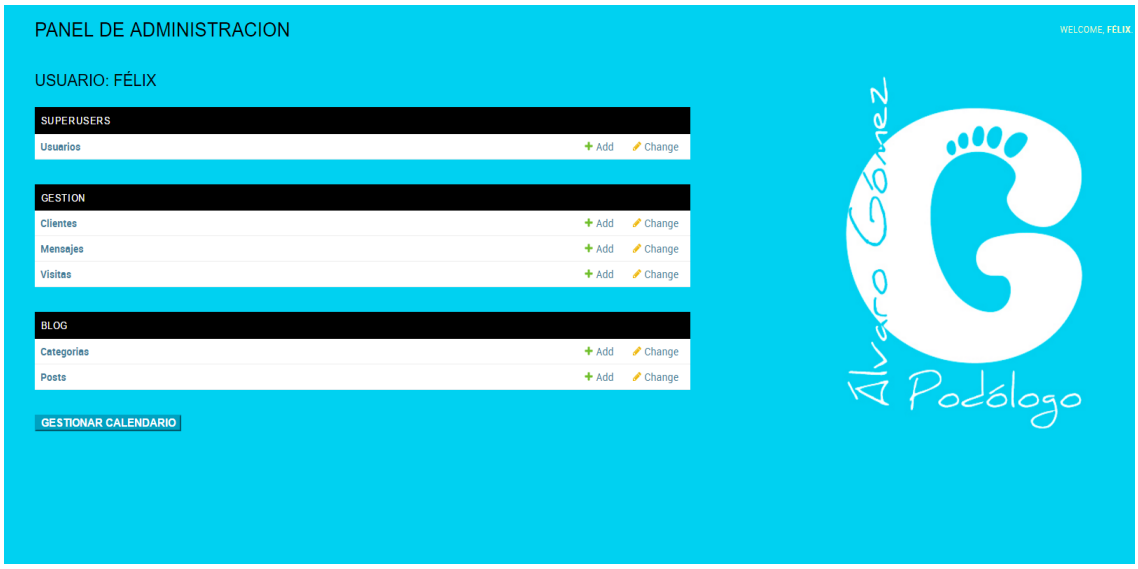


Figura 24. Panel principal de administración

Desde aquí ya podemos hacer las gestiones que pertinentes.

Caso 4: Crear un Cliente

Una vez autenticados y dentro de nuestra aplicación, una de las cosas que podemos hacer es registrar un cliente en nuestra base de datos.

Pinchamos en la opción “ + Add” de los clientes y nos redirigirá al formulario de creación de clientes.

Rellenamos los datos y guardamos.

Figura 25. Creación de un cliente-1

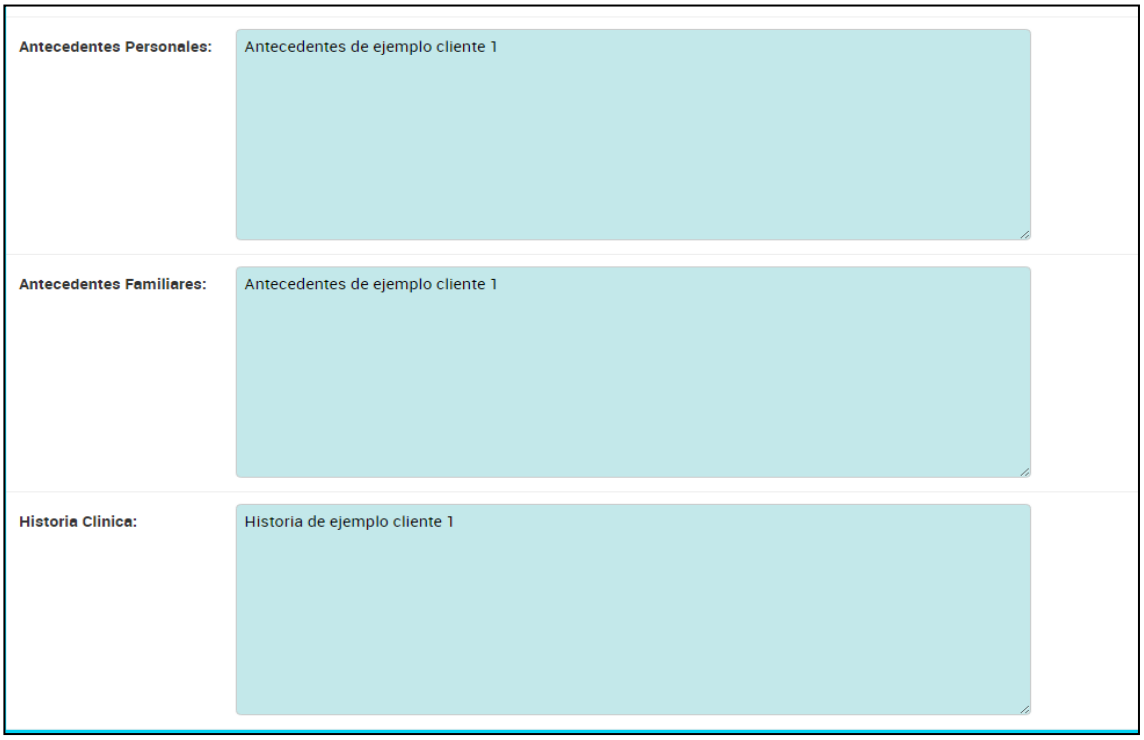


Figura 26. Creación de un cliente-2

A la hora de guardar podemos elegir si seguir editando, guardar y salir o guardar y crear otro nuevo, en este caso elegiremos guardar y salir y nos redirigirá a la lista de todos los clientes.

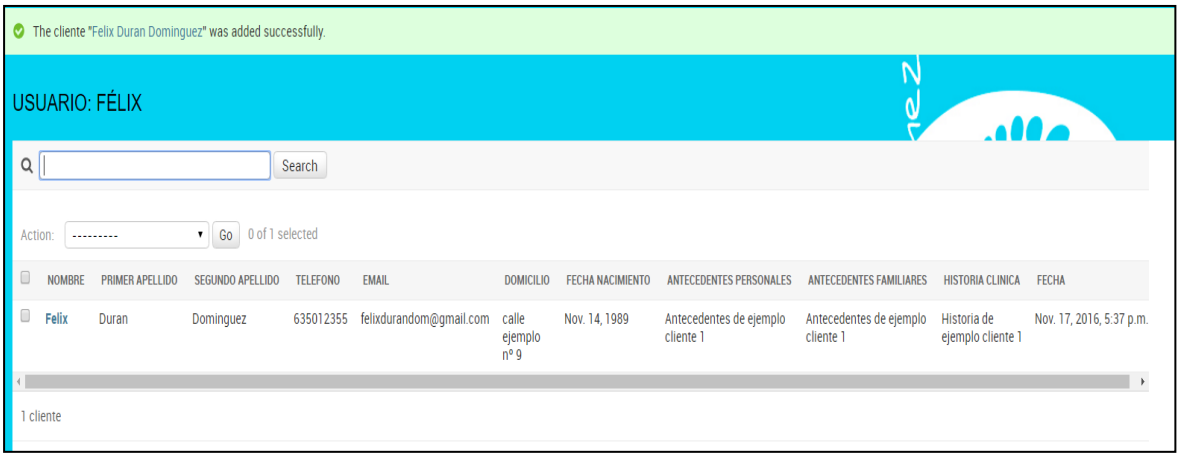


Figura 27. Listado de clientes

Caso 5: Creación de una visita

Desde el panel de administración y con clientes ya creados, cada vez que dicho cliente acuda a una consulta, se creará una visita para registrar lo que se realice ese día.

Desde el panel principal seleccionamos añadir una visita. La aplicación nos redirige al formulario de creación de visitas. Utilizando el desplegable elegimos un cliente de los ya creados y rellenamos la información de ese día para ese cliente.

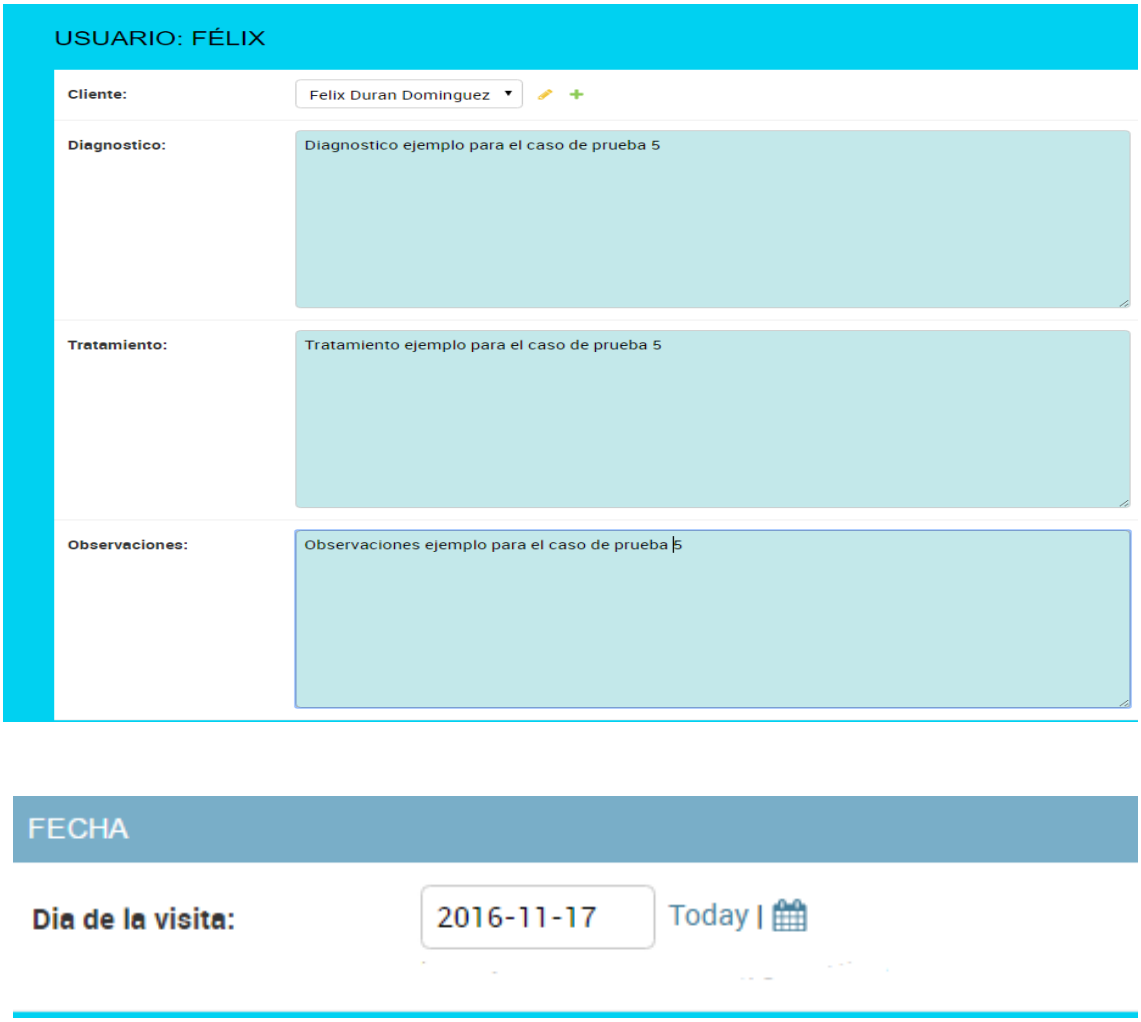


Figura 28. Creación de una visita

Caso 6: Gestión de calendarios

Desde el panel principal podemos acceder a la vista de gestión del calendario para ver nuestro calendario y añadir eventos si se desea.

Pinchamos en el botón de gestión del calendario y este nos redirige a una vista en la que primero tendremos que autenticarnos en Google para acceder a nuestros eventos.

Figura 29. Autorización Google

Pinchamos en “Autorizar” y nos autenticamos con Google.

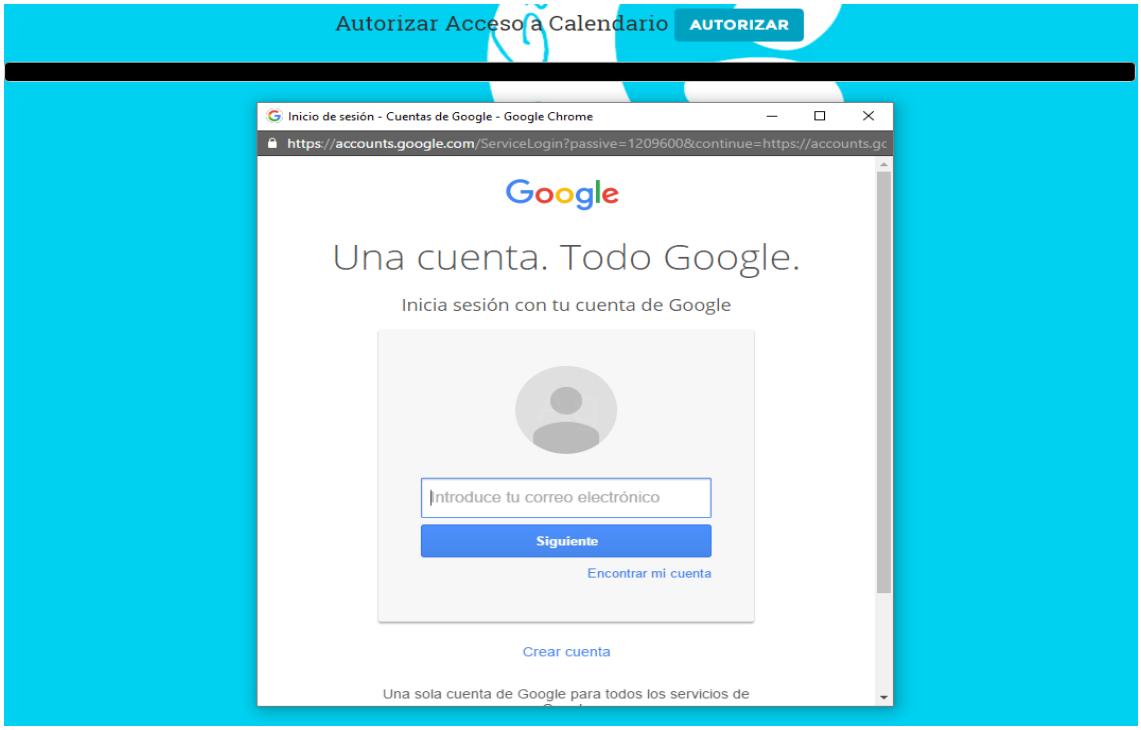


Figura 30. Datos Google

Rellenamos los datos y a continuación se generará la vista del calendario.

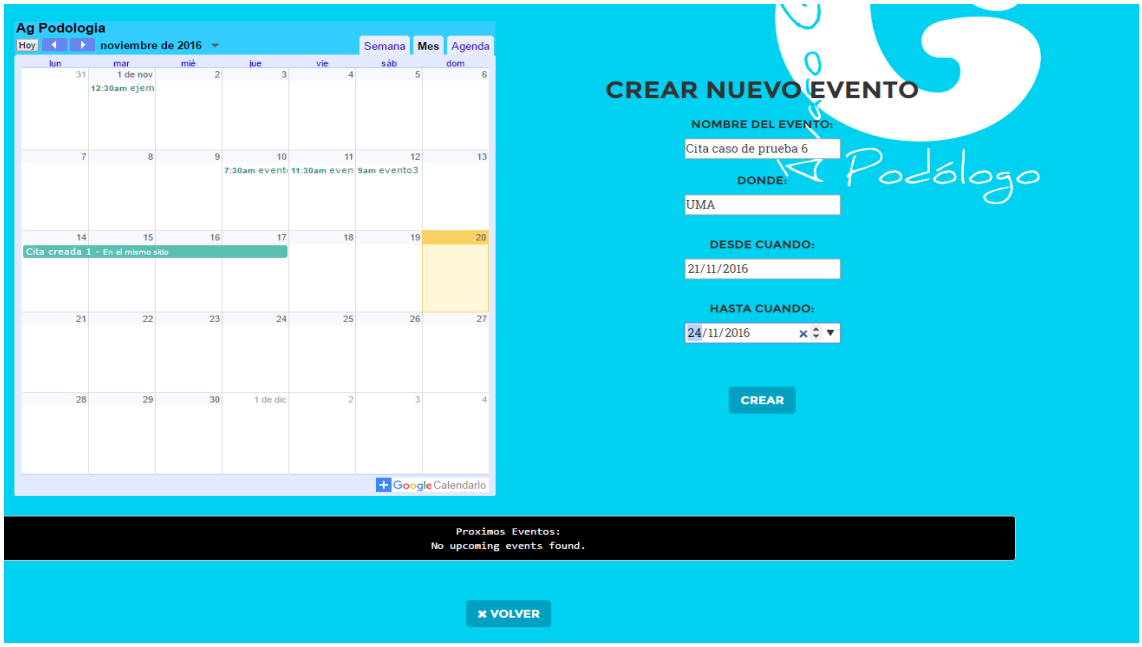


Figura 31. Vista de calendario y creación de evento

Si lo deseamos, podemos rellenar los datos de un nuevo evento y añadirlo a nuestro calendario.

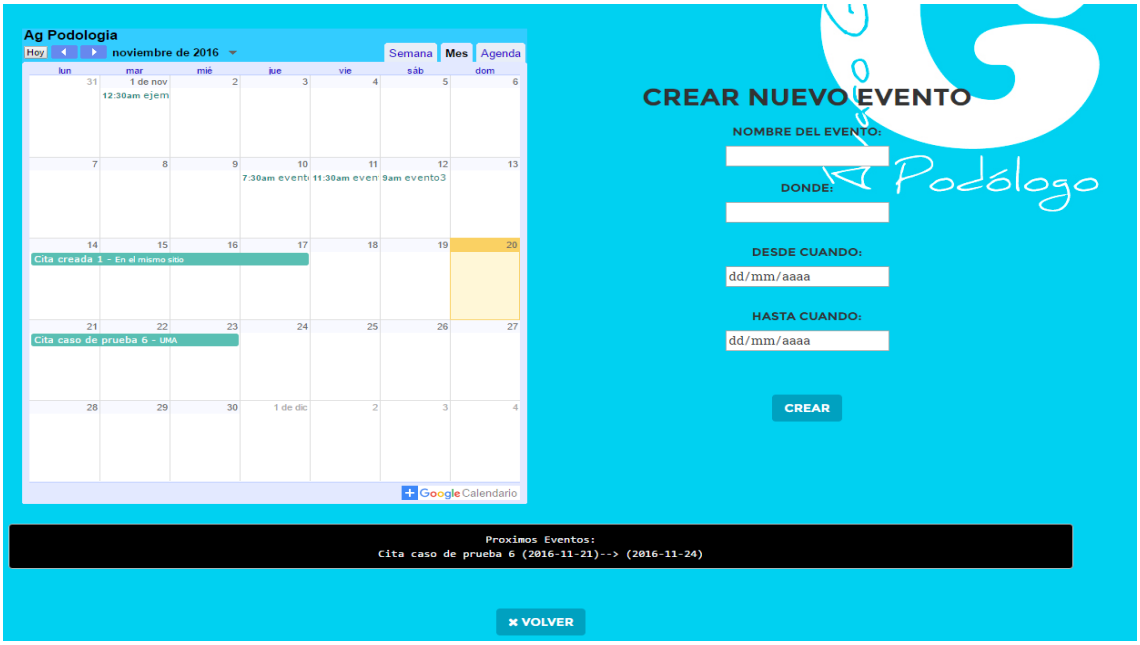


Figura 32. Evento Creado

7. Detalles de la Implementación

En este apartado se pretende mostrar alguna de las funcionalidades de la aplicación desde un punto de vista más técnico, centrándonos en la parte de implementación del código y en la estructura del mismo. Se mostraran varias capturas con el código utilizado en cada momento.

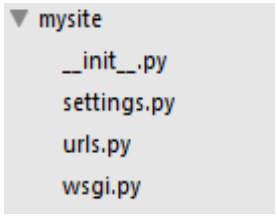
Estructura de ficheros en Django

Como hemos señalado anteriormente, Django es un framework de Python que facilita la tarea de crear aplicaciones web, y como tal, tiene su propia estructura de ficheros.

A continuación se mostrará cómo está estructurado el código en Django:

Esta es la estructura básica de un proyecto Django.

La carpeta contenedora de todo el proyecto es la carpeta llamada “producción”, dentro se crea la carpeta mysite para contener los archivos del proyecto. Tenemos tres subcarpetas en el interior de esta, una con el mismo nombre, otra con el nombre de la aplicación, en este caso polls, y otra con el nombre templates. A continuación veremos en profundidad el contenido de estas carpetas.



La carpeta mysite contiene 4 archivos en lenguaje Python.

El primero es un archivo propio de Python para el trato correcto de los directorios.

Settings.py es el archivo de configuración de Django, en él se tienen que configurar varios aspectos como la base de datos, las rutas de los ficheros estáticos, y varias cosas más que veremos en el siguiente apartado.

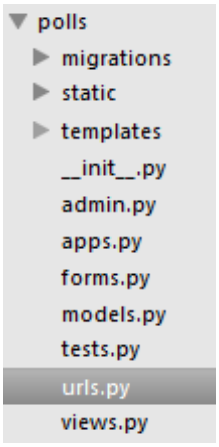
Urls.py es donde se especifican las direcciones de nuestras vistas. Su contenido es el siguiente:

```
urlpatterns = [
    url(r'^', include('polls.urls')),
    url(r'^admin/', admin.site.urls),
]
```

Figura 33. Urls.py

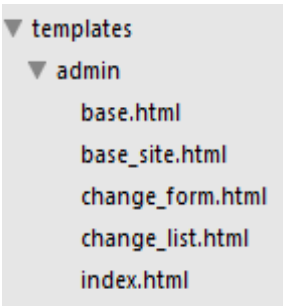
Esto indica que las urls de nuestro proyecto van a ser las que estén en el archivo “polls.urls”, el cual veremos más adelante, y las urls del panel de administración que Django proporciona.

El último archivo contiene información necesaria para el despliegue en producción.



Polls es el fichero principal de nuestra aplicación. Su contenido es el siguiente:

- Migrations: contiene los cambios realizados en la base de datos.
- Static: contiene todos los ficheros estáticos, es decir, archivos CSS, JavaScript, imágenes, fuentes, etc.
- Templates: Aquí se almacenan todas las vistas HTML de la aplicación.
- Admin.py: archivo de configuración para las vistas del panel de administración.
- Apps.py: archivo donde se especifica el nombre de la aplicación.
- Forms.py: archivo que proporciona Django para la creación de formularios web. Contiene los datos de cada formulario.
- Models.py: archivo donde se almacena la información de la base de datos en Django. Contiene la estructura de cada entidad.
- Urls.py: contiene las rutas de todas las vistas HTML del proyecto.
- Views.py: contiene la lógica de las vistas del proyecto.



La carpeta templates del proyecto contiene una subcarpeta llamada admin; esto se hace para poder modificar las vistas del panel de administración de Django sin tener que modificar las plantillas originales.

Se copian en este directorio las plantillas que queramos cambiar y a partir de ese momento Django utilizará estas vistas y no las originales.

Preparando el entorno de trabajo

Lo primero al comenzar la aplicación es configurar el entorno de trabajo a usar. En este caso se han utilizado las siguientes versiones de las distintas tecnologías empleadas:

- Python versión 2.7
- Django versión 1.10
- JQuery versión 3.1.0
- MySql versión 5.6.27

Una vez instalados estos paquetes, los cuales deben instalarse tanto en la máquina local como en la máquina del hosting que se ha utilizado, se procede a la configuración de Django. Cabe señalar que se mantienen dos versiones del “settings.py”; una para las pruebas en local y otra para el despliegue en producción. A continuación se muestra la diferencia que existe entre estas dos versiones.

Para local:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'myDataB',
        'USER': 'root',
        'PASSWORD': [redacted],
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Figura 34. Configuración base de datos en local

Para producción:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'felixDD666$myDataB2',
        'USER': 'felixDD666',
        'PASSWORD': [redacted],
        'HOST': 'felixDD666.mysql.pythonanywhere-services.com',
        'PORT': '3306',
    }
}
```

Figura 35. Configuración base de datos en producción

Como se puede observar la diferencia radica en la base de datos. En local tenemos la base de datos configurada para funcionar con un host local mientras que en el entorno de producción tenemos todos los datos necesarios para trabajar con la base de datos creada en el hosting “pythonAnyWhere”.

Continuamos con la configuración, a continuación se muestran otras partes de este archivo “settings.py” las cuales son necesarias para la correcta configuración de Django, tanto para local como para producción.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Figura 36. Configuración de las plantillas de admin

En esta figura se puede ver la configuración de las plantillas de administración de Django.

```
#Email Data
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = 'podoAlvaro@gmail.com'
EMAIL_HOST_PASSWORD = [redacted]
EMAIL_PORT = 587
```

Figura 37. Configuración para la emisión de emails

Aquí podemos observar la configuración necesaria para la funcionalidad del envío de emails. Cuando un cliente escribe un mensaje para el podólogo en la web, automáticamente se crea un email y se envía desde la cuenta de correo que se observa en la figura anterior, la cual se ha creado específicamente para este fin.

```
# Static files (CSS, JavaScript, Images)

STATIC_URL = '/static/'


STATIC_ROOT = "/var/www/statics"

STATIC_URL = '/static/'

STATICFILES_DIRS = (
    os.path.join(BASE_DIR,'polls/static'),
)
# Media files<

MEDIA_ROOT = os.path.join(BASE_DIR, 'polls/static/media')

MEDIA_URL = '/media/'
```

Figura 38. Rutas para ficheros estáticos y media

En esta figura se observa la configuración necesaria para gestionar los ficheros estáticos y los archivos subidos desde la web, como pueden ser las fotos de los Posts.

```
INSTALLED_APPS = [
    'polls.apps.PollsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Figura 39. Aplicaciones Django

Y por último, lo más básico de toda la configuración, añadir nuestra aplicación a la lista de aplicaciones de Django. La primera aplicación que se observa en la figura anterior es la nuestra, las demás son las predefinidas por Django para cumplir todas sus funciones.

Una vez configurado este fichero ya está listo Django para su uso. A continuación mostraré algunas de las partes más importantes del código final.

Modelos, bases de datos y vistas

```
from __future__ import unicode_literals

from django.db import models
from django.utils import timezone


class Cliente(models.Model):
    Nombre = models.CharField(max_length=200)
    Primer_Apellido = models.CharField(max_length=200)
    Segundo_Apellido = models.CharField(max_length=200)
    Telefono = models.CharField(max_length=200)
    Email = models.EmailField(max_length=200)
    Domicilio = models.CharField(max_length=200)
    Fecha_Nacimiento = models.DateField()
    Antecedentes_Personales = models.TextField()
    Antecedentes_Familiares = models.TextField()
    Historia_Clinica = models.TextField('Historia Clinica')
    Dia_registro = models.DateTimeField('Fecha')
    def __str__(self):
        return self.Nombre + " " + self.Primer_Apellido + " " + self.Segundo_Apellido
```

Figura 40. Modelo cliente

En esta figura se puede observar uno de los campos más importantes de nuestra base de datos, los clientes. Este código se guarda en el fichero “models.py” el cual, como ya hemos visto, es el encargado de tener todas las entidades de la base de datos. Este fichero “models.py” guarda simplemente la estructura de cada una de las tablas, para diseñar de qué modo se van a

mostrar estos datos tenemos que irnos a otro fichero de Django, el fichero “admin.py”

```python
from django.contrib import admin

from .models import *
from django.utils import timezone



class ClientesAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,              {'fields': ['Nombre','Primer_Apellido','Segundo_Apellido','Telefono','Email',
            'Domicilio','Fecha_Nacimiento','Antecedentes_Personales','Antecedentes_Familiares','Historia_Clinica']}),
        ('Fecha de Creacion', {'fields': ['Dia_registro']}),
    ]
    list_display = ('Nombre','Primer_Apellido','Segundo_Apellido','Telefono','Email',
        'Domicilio','Fecha_Nacimiento','Antecedentes_Personales','Antecedentes_Familiares','Historia_Clinica', 'Dia_registro')
    list_filter = ['Dia_registro']
    search_fields = ['Nombre']


admin.site.register(Cliente, ClientesAdmin)
```

Figura 41. Configuración de los campos de cliente

En este fichero crearemos la forma en la que se verán los distintos campos de nuestro panel de administración de Django. Podemos configurar que se va a ver, cómo se va a ver, el orden, los filtros, los campos de búsqueda, etc.

Hasta ahora solo hemos visto cómo se comporta Django dentro del panel de administración, pero no hemos visto como se hace la comunicación entre los modelos y las vistas.

A continuación mostraremos primero, los modelos de nuestro blog, después cómo se muestran en el panel de administración y, por último, cómo mostrarlos en la web principal.

```python
class Categoria(models.Model):
    Nombre = models.CharField(max_length=200)

    def __str__(self):
        return self.Nombre

class Post(models.Model):
    Titulo = models.CharField(max_length=200)
    Contenido = models.TextField()
    Categoria = models.ForeignKey(Categoria)
    Fecha_Creacion = models.DateTimeField('Fecha de creacion')
    PUBLICO = 'pub'
    PRIVADO = 'pri'
    pORp_choices = (
        (PUBLICO, 'publico'),
        (PRIVADO, 'privado'),
    )
    Publico_Privado = models.CharField(
        max_length=3,
        choices=pORp_choices,
        default=PUBLICO,
    )
    Foto = models.ImageField(upload_to='postPhotos', default = 'SOMETHING')
    def __str__(self):
        return self.Titulo
```

Figura 42. Modelos categoría y post

Cabe señalar algunas diferencias entre estos modelos y el modelo anteriormente descrito. Se puede observar que estos dos modelos están conectados, concretamente tienen una relación de clave foránea. Una categoría está compuesta de muchos posts. También podemos ver un campo un tanto especial, el último de ellos, “imageField”, se trata de un campo especial para almacenar imágenes donde tenemos que definir una carpeta en la cual guardaremos dichas imágenes. Si el podólogo desea acompañar algún post de una imagen, puede hacerlo sin necesidad de cambiar el código ni tener conocimientos de informática.

```python
class CategoryAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['Nombre']})
    ]
    search_fields = ['Nombre']

admin.site.register(Categoria, CategoryAdmin)


class PostsAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['Titulo','Contenido','Categoria','Publico_Privado','Foto']}),
        ('Date information', {'fields': ['Fecha_Creacion']}),
    ]
    list_display = ('Titulo', 'Contenido', 'Fecha_Creacion' , 'Categoria','Foto')
    list_filter = ['Fecha_Creacion']
    search_fields = ['Titulo']

admin.site.register(Post, PostsAdmin)
```

Figura 43. Configuración de los campos de categoría y post

Hasta aquí, sería el mismo recorrido que en el ejemplo anterior de los clientes. Ahora veremos cómo hacer que se vean estos post en la web principal, es decir, veremos la comunicación entre los datos y las vistas.

```
def principal(request):
    posts = Post.objects.order_by("-Fecha_Creacion")
    numPost = Post.objects.count()-3;
    return render(request,'polls/principal.html',{
        "posts":posts,
        "numPost": numPost
        })
```

Figura 44. Lógica de la vista principal.html

Esta figura es un trozo del fichero “views.py”, este fichero contiene la lógica de las vistas en Django y permite que este las muestre correctamente. Como se puede ver en esta vista, hemos definido unas variables especiales para poder mostrar los posts que creemos en el panel de administración. La variable “posts” lleva todos los objetos, ordenados por fecha de creación, que hay en el modelo “Post”. La otra variable es simplemente una variable auxiliar para poder mostrar un número definido de posts y no más de ese número.

En la siguiente figura observamos la forma en la que Django se inserta en HTML para formar esta comunicación entre modelos y vistas. En este caso lo que hemos hecho es utilizar un bucle “for” para recorrer todos los posts, y dentro de este hemos utilizado unas sentencias “if” para mostrar solo el contenido que sea público y no se superen más de tres posts mostrados. De estos posts se muestra, por este orden, los siguientes datos:

- Título: {{ post.Titulo }}
- Categoría: {{ post.Categoria }}
- Contenido del post: {{ post.Contenido}}

```
<section id ="blog">
    <div class= "container">
        <div class="row">
            <div class="col-lg-12 col-md-12 col-sm-12 text-center">
                <h2 class="section-heading">BLOG</h2>
                <h3 class="section-subheading text-muted">Entérate de lo último</h3>
            </div>

            {% for post in posts %}
                {% if post.id > numPost%}
                    {%if post.Publico_Privado == "pub"%}
                    <div class = "row">
                        <div class = "col-lg-8 col-md-10 col-sm-10">
                           <a href="{{post.id}}"><h2>{{ post.Titulo }}</h2></a>
                            <h4>Categoria: {{post.Categoria}}</h4>
                            <p>{{ post.Contenido }}</p>
                            <hr>
                        </div>
                        <div class= "col-lg-2 col-md-1 col-sm-1">
                            <a href="{{post.id}}">
                                <span id="iconButtonBlog" class="fa-stack fa-4x">
                                    <i id="icons" class="fa fa-circle fa-stack-2x text-primary
                                    "></i>
                                    <i id="iconsShape" class="fa fa-share fa-stack-1x fa-
                                    inverse"></i>
                                </span>
                            </a>
                        </div>
                    </div>
                    <br><br><div class="lineaDiv">-</div><br><br>
                    {%endif%}
                {%endif%}
            {% endfor %}
        </div>
        <div class= "row text-center">
            <div class="col-lg-8 col-lg-offset-2 col-md-10 col-md-offset-1 col-sm-10 col-sm-
            offset-1">
                <a href="allPosts"><button type="button" class="btn btn-primary">Ver todos
                los posts</button></a>
            </div>

        </div>
    </div>
</section>
```

Figura 45. Sección del blog de la vista principal.html

Funcionalidad de contacto con el podólogo

En este apartado vamos a ver el código que existe detrás de una de las funcionalidades más importantes de este proyecto, el formulario de contacto de los clientes con el podólogo.

Esta funcionalidad toca absolutamente todas las ramas de las que se forma este proyecto. Lo iremos explicando poco a poco. La parte de configuración del fichero “settings.py” ya está explicada anteriormente así que pasaré directamente a la explicación de las otras partes involucradas.

Lo primero que debemos hacer es crear nuestro formulario Django. En el fichero llamado “forms.py” creamos la clase del formulario que necesitamos.

```python
from django import forms

class ContactForm(forms.Form):
    nombre = forms.CharField(required=True,widget=forms.Textarea)
    email = forms.EmailField(required=True)
    telefono = forms.CharField(required=True,widget=forms.Textarea)
    mensaje = forms.CharField(required=True,widget=forms.Textarea)
```

Figura 46. Campos del formulario

Una vez creado, tenemos que hacer que la vista HTML rellene los datos desde el formulario de contacto, y después, configurar lo que se debe hacer cuando el mensaje es enviado, esto se hace usando los métodos apropiados en el fichero "views.py".

```python
def contacta(request):
    if request.method == 'GET':
        form = ContactForm()
    else:
        form = ContactForm(request.POST)
        if form.is_valid():

            nombre = form.cleaned_data['nombre']
            email = form.cleaned_data['email']
            telefono = form.cleaned_data['telefono']
            mensaje = form.cleaned_data['mensaje']
            newMensaje = Mensaje(Nombre = nombre, Email = email,
                Telefono = telefono, Mensaje = mensaje)
            try:
                send_mail("Nuevo mensaje de " + nombre, "Email: " + email + "\n\nTelefono: "
                    + telefono + "\n\nContenido: " + mensaje ,
                     'podoAlvaro@gmail.com', ['agpodologia@gmail.com'])
                newMensaje.save()
            except BadHeaderError:
                return HttpResponse('Invalid header found.')
            return HttpResponseRedirect("thanks")
    return render(request, "polls/contacta.html", {'form': form})

def thanks(request):
    return render(request,'polls/thanks.html',{})
```

Figura 47. Lógicas de las vistas contacta y thanks

Hay mucho que explicar de esta figura, vayamos por partes. Lo primero es crear una variable que contenga el formulario que hemos creado anteriormente, la variable "form". Seguidamente crearemos las variables para cada uno de esos campos que se utilizarán para almacenar el mensaje en la base de datos y para construir el email. Acto seguido creamos una variable para almacenar nuestro nuevo mensaje en la base de datos y le añadimos los campos. Si el formulario es válido y no hay ningún fallo, haremos lo siguiente:

- Llamaremos a la funcion send_mail(), que cogerá dichos datos, creará un email y lo mandará desde la cuenta que hemos creado para este propósito, hasta el email personal de nuestro podólogo.
- Además, se guardará, con la función .save(), el nuevo mensaje en la base de datos.
- Nos redirigirá a la vista “thanks”, cuya lógica se puede observar debajo de la de contacta.

Por supuesto, esta figura no tiene ningún sentido si no añadimos estas dos imágenes:

```
class Mensaje(models.Model):
    Nombre = models.CharField(max_length=200)
    Email = models.CharField(max_length=200)
    Telefono = models.CharField(max_length=200)
    Mensaje = models.TextField();
```

Figura 48. Modelo mensaje

La figura 48 es el modelo de los mensajes que se guardan en la base de datos, cogidos directamente de los “inputs” que los clientes rellenan en la vista HTML de contacto. La figura 49 muestra la vista HTML de contacta y cómo se crea un formulario con Django utilizando la variable “form”.

```
<div id="contactContainer" class="container" ng-controller="Controlador1">
        <div class="row" id="contacHead">
            <div class="col-lg-12 text-center">
                <h2 id="contacHeadText" class="section-heading">Rellena este sencillo
                formulario y te contestaré cuanto antes</h2>
                <br><br><br>
            </div>
        </div>
            <form method="post" name="sentMessage" id="contactForm">
                    {% csrf_token %}
                    <div class="row">
                        <div class="col-md-4 text-center">
                            <div class="form-group" >
                                <h3 class="textContact">Nombre:</h3>{{form.nombre}}
                            </div>
                        </div>
                        <div class="col-md-4 text-center">
                            <div class="form-group">
                                <h3 class="textContact">Email:</h3> {{form.email}}
                            </div>
                        </div>
                        <div class="col-md-4 text-center">
                            <div class="form-group">
                                <h3 class="textContact">Teléfono:</h3> {{form.telefono}}
                            </div>
                        </div>
                    </div>
                    <br><br><br>
                    <div class="row">
                    <div class="col-md-12 text-center">
                        <div class="form-group">
                            <h3 class="textContact">Mensaje:</h3> {{form.mensaje}}
                        </div>
                    </div>
                    </div>
                    <div class="row text-center">
                        <div class="form-actions">
                          <button id="contactSubmit" class="btn btn-primary" type="submit"
                          disabled="disabled">Enviar</button>
                        </div>
                    </div>
            </form>
    </div>
```

Figura 49. Vista del formulario de contacto

Configuración para la autenticación de Google

Para la utilización del calendario de Google y para la creación de eventos desde la aplicación, se necesitan unos pasos previos muy concretos para poder utilizar las credenciales de Google.

Lo primero es crear un proyecto en la Api de Google. Una vez creada debemos configurar la pantalla de autorización de OAuth:

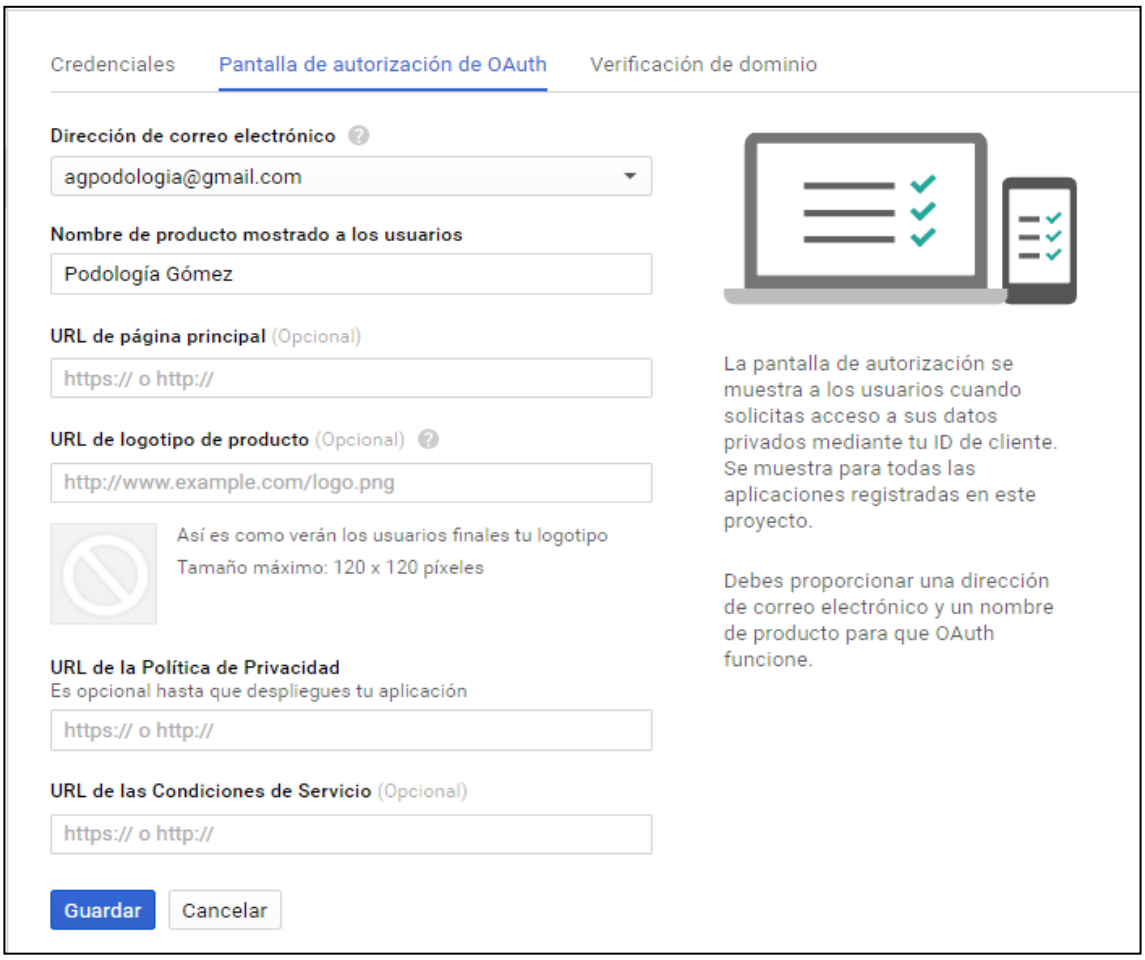


Figura 50. Pantalla de autorización de OAuth en Google API

Seguidamente necesitamos crear un ID de cliente de OAuth para autorizar a nuestro sitio web la utilización de estas API. Como observamos en la figura siguiente, tenemos que configurar los orígenes de JavaScript para que funcione tanto en local como en nuestra aplicación en producción.

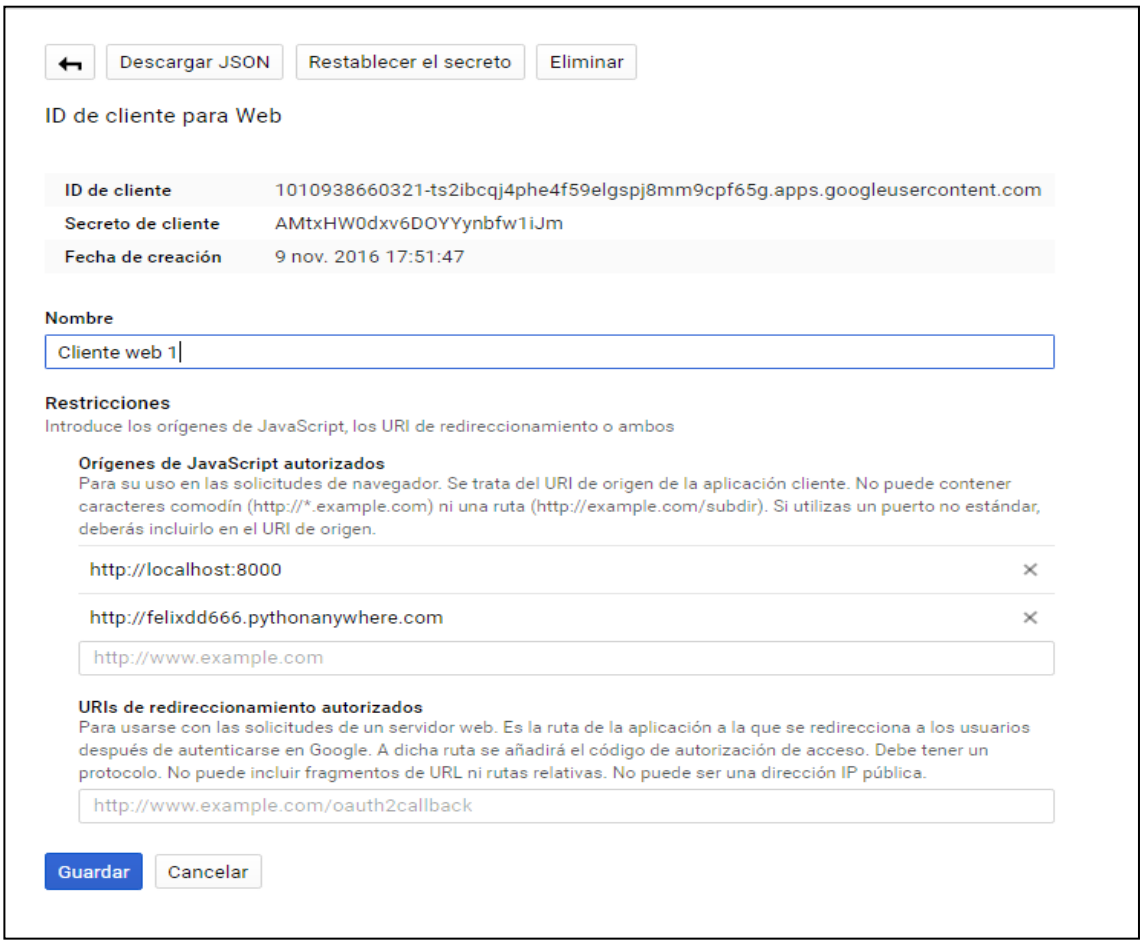


Figura 51. Creación de ID de OAuth

Una vez hecho esto anotamos el ID del cliente que se genera y lo utilizamos en nuestro archivo JavaScript que será el que se encargue de gestionar el contacto de nuestra aplicación con Google. Mostraremos las líneas de código más importantes a continuación.

```
// Developer Console, https://console.developers.google.com
var CLIENT_ID = '1010938660321-ts2ibcqj4phe4f59elgspj8mm9cpf65g.apps.googleusercontent.com';

var SCOPES = ["https://www.googleapis.com/auth/calendar"];
```

Figura 52. Uso del ID OAuth para autenticación JavaScript

La siguiente función comprueba la autorización cuando se carga la vista.

```
window.onload = function(){
  gapi.auth.authorize(
    {
      'client_id': CLIENT_ID,
      'scope': SCOPES.join(' '),
      'immediate': true
    }, handleAuthResult);
}
```

Figura 53. Función de autorización Google

La función anterior llama a la función de la figura siguiente y dependiendo de si esta autenticado o no, muestra, o bien, el “div” con el botón para comenzar la autenticación, el cual podemos ver en la página 40 de esta misma memoria, o bien lo oculta y carga el calendario.

```
function handleAuthResult(authResult){
  var authorizeDiv = document.getElementById('authorize-div');
  if (authResult && !authResult.error) {
    // Hide auth UI, then load client library.
    authorizeDiv.style.display = 'none';
    loadCalendarApi();
  } else {
    // Show auth UI, allowing the user to initiate authorization by
    // clicking authorize button.
    authorizeDiv.style.display = 'inline';
  }
```

Figura 54. Función handleAuthResult de Google API

```
function loadCalendarApi() {
  $('#cuerpoCitas').css('display','block');
  gapi.client.load('calendar', 'v3', listUpcomingEvents);
}

/**
 * Print the summary and start datetime/date of the next ten events in
 * the authorized user's calendar. If no events are found an
 * appropriate message is printed.
 */
function listUpcomingEvents() {
  var request = gapi.client.calendar.events.list({
    'calendarId': 'primary',
    'timeMin': (new Date()).toISOString(),
    'showDeleted': false,
    'singleEvents': true,
    'maxResults': 10,
    'orderBy': 'startTime'
  });
```

Figura 55. Función de carga de calendario y lista de eventos

Cargamos la función de los eventos que nos proporciona Google y además añadimos la funcionalidad de mostrar nuestro “cuerpoCitas” el cual podemos ver en la figura de la página 41.

Si el podólogo decide crear un nuevo evento, rellena los campos y pulsa crear, las siguientes líneas de código serán las que se ejecutaran.

```
function clickCrearCita(){

  var event = {
    'summary': $('#summary').val(),
    'location': $('#where').val(),
    'start': {
      'date': $('#when1').val(),
    },
    'end': {
      'date': $('#when2').val(),
    }
  };
  var request = gapi.client.calendar.events.insert({
    'calendarId': 'primary',
    'resource': event
  });
  console.log("Request");

  request.execute(function(event) {
    appendPre('Event created: ' + event.htmlLink);
    console.log("Execute");
  });
  setTimeout(function(){ location.reload(); }, 1500);
}
```

Figura 56. Función para crear citas

Como se puede ver, cogemos los valores necesarios mediante JQuery y los insertamos en el calendario “primary” que en este caso es el calendario principal de nuestro podólogo. Después de esto y para que los cambios se vean en la aplicación, recargamos la vista, tras esperar un tiempo prudencial, para que se efectúen todos los cambios correctamente.

8. Pruebas

Las pruebas que se han llevado a cabo han sido realizadas en varios ordenadores y dispositivos móviles, así como en varios navegadores.

El periodo de pruebas no ha sido exactamente un periodo concreto, sino todo lo contrario, cada vez que se añadía una nueva característica a la aplicación, esta sometía a pruebas de integración y funcionalidad para comprobar que todo funcionaba bien junto y cumplía su función sin ningún fallo.

Las pruebas de integración han tomado una importancia altísima en este proyecto ya que conforme iban añadiéndose características, se completaban otras y debía comprobarse que todo funcionaba a la perfección y nada se veía alterado por la incorporación de nuevas características.

Se ha comprobado que el diseño y las funcionalidades son correctas tanto en dispositivos móviles como en ordenadores, y en los navegadores Chrome, Firefox, Safari y Edge.

Como añadido se ha realizado también las pruebas en un entorno de producción real. Esto ha hecho que las pruebas deban repetirse tanto en el entorno local como en producción, ya que existen varias diferencias entre ellos.

El entorno de producción es más delicado y debe comprobarse que todo funciona correctamente con más constancia.

9. Conclusiones y trabajo futuro

En la época que vivimos el diseño de aplicaciones web es un negocio que tiene una extensión prácticamente mundial y con una importancia enorme. Todo negocio necesita estar conectado, necesita darse a conocer de la forma más eficiente que permitan las tecnologías presentes. Sin duda alguna la presencia en internet es esencial para este motivo. Por este motivo la finalidad principal que se persigue con este trabajo es la de ejercer un ejemplo de lo que sería un trabajo real y profesional. No fue solo esto la motivación para elegir este proyecto, también ha servido para crecer personalmente, por este motivo se va a centrar este último capítulo en esas dos ideas, el crecimiento tanto profesional como personal que se ha producido desarrollando este proyecto.

En este caso, la creación de esta aplicación, desde su base, viene fundamentada en ayudar a un profesional de la podología, el cual, cabe destacar, que también es un gran amigo mío. Este proyecto pretende ayudar a que este podólogo tenga presencia en internet, dándose a conocer y dándole la posibilidad de crecer como profesional. Desde un punto de vista profesional hace que quede un sentimiento de satisfacción, ya que ayudamos a otras personas con el trabajo y el esfuerzo.

Este sentimiento anteriormente mencionado se completa con la sensación de un trabajo en el cual se han puesto ganas y esfuerzo a partes iguales. Todo lo aprendido mientras se desarrollaba la aplicación será de gran ayuda a la hora de buscar trabajo.

La elección de las tecnologías usadas tampoco ha sido al azar. Se han elegido unas tecnologías que con unas características que las hacen muy útiles de aprender y conocer. Python es un lenguaje muy utilizado y extendido, y con su framework Django hacen de ellos unos conocimientos que vale la pena aprender. Sumándole a esto al aprendizaje sobre el framework bootstrap, el cual hace que el diseño de la aplicación quede más profesional y atractivo, y añadiéndole la mejoría que experimentada con los lenguajes HTML, CSS, JQuery y JavaScript, completan las nuevas aptitudes que se han adquirido realizando este proyecto, aptitudes que sin lugar a muy útiles para la salida al mercado laboral.

Por otro lado, el seguimiento de unas metodologías tan extendidas como la incremental iterativa y el uso de una estructura Modelo-Vista-Controlador hacen, sin lugar a dudas, que aumenten aún más las capacidades adquiridas.

Por último, como ya hemos visto se ha utilizado un entorno de producción real. Esto ha servido para acercarnos a lo que sería un trabajo real en un ámbito profesional real.

Como trabajo futuro existen multitud de funcionalidades que se le podrían añadir a esta aplicación. En concreto, en un futuro próximo se abordarán las siguientes:

- Despliegue de la aplicación con un dominio privado para poder disponer de una dirección URL propia y no una gratuita. Esto hará que el dominio no caduque y tenga una capacidad de cómputo y de almacenaje mayor.

- La posibilidad de añadir grupos de usuarios en el panel de administración. Si el podólogo decide montar una clínica podológica, la aplicación debe ser capaz de dar de alta a otros usuarios para que tengan acceso a la información de los clientes y demás.

- Personalización completa del panel de administración. En este caso he seguido un diseño fiel a la imagen corporativa de la web, pero existe la posibilidad de que el podólogo quiera otro diseño distinto para su panel, de administración.

Gracias a la estructura y tecnologías utilizadas en este proyecto estas mejoras y otras muchas más podrán abordarse con relativa facilidad.

Referencias bibliográficas

- Documentación de Django: https://www.djangoproject.com/
- Despliegue Django: https://tutorial.djangogirls.org/es/deploy/
- Curso de Django: https://www.seiscocos.com/shop6cocos/programacion/24-avanzadoapps.html
- Curso de Angular: https://www.codeschool.com/courses/shaping-up-with-angular-js
- Documentación MySQL: http://www.mysql.com/
- Documentación de Angular: https://docs.angularjs.org/api
- Documentación de la API de Google: https://developers.google.com/
- Documentación de JQuery: http://api.jquery.com/
- Documentación HTML: http://www.w3schools.com/html/
- Documentación CSS: http://www.w3schools.com/css/
- Documentación JavaScript: http://www.w3schools.com/js/
- Documentación Bootstrap: http://librosweb.es/libro/bootstrap_3/
- Alojamiento de producción: https://www.pythonanywhere.com/
- Editor de código: https://www.sublimetext.com/3

Anexo – Manual de Usuario

Este anexo ofrece una guía de uso para facilitar el aprendizaje de la aplicación a usuarios finales. Abordaremos esta guía desde los dos roles que existen en la aplicación.

Cliente

Acceso a la web

Para acceder a la web necesitamos buscar la dirección de la web en el buscador que utilicemos. Ingresamos el siguiente enlace en la barra de búsqueda: felixdd666.pythonanywhere.com

Figura 57. Anexo - Portada Web

Navegación

Una vez dentro de la web podemos comenzar con la navegación. La página principal de la web se divide en:

- **La podología**

Esta es la sección introductoria a la página, se accede haciendo scroll o bien pulsando sobre el botón “Quiero saber más” de la portada. Contiene información básica sobre qué es la podología y a quién va dirigida.

Figura 58. Anexo - Sección “La podología”

- **Tu podólogo**

En esta sección se puede ver la evolución profesional del podólogo, desde sus comienzos en la Universidad de Málaga hasta su actividad actual.

En la figura 47 tenemos dos opciones de intervención. Podemos pulsar sobre la foto de “Mejorando para ti” para ver la formación complementaria del podólogo o también podemos pulsar sobre el botón “Descargar CV” para conseguir una copia del currículum de nuestro podólogo.

Figura 59. Anexo - Sección "Tu podólogo" - 1

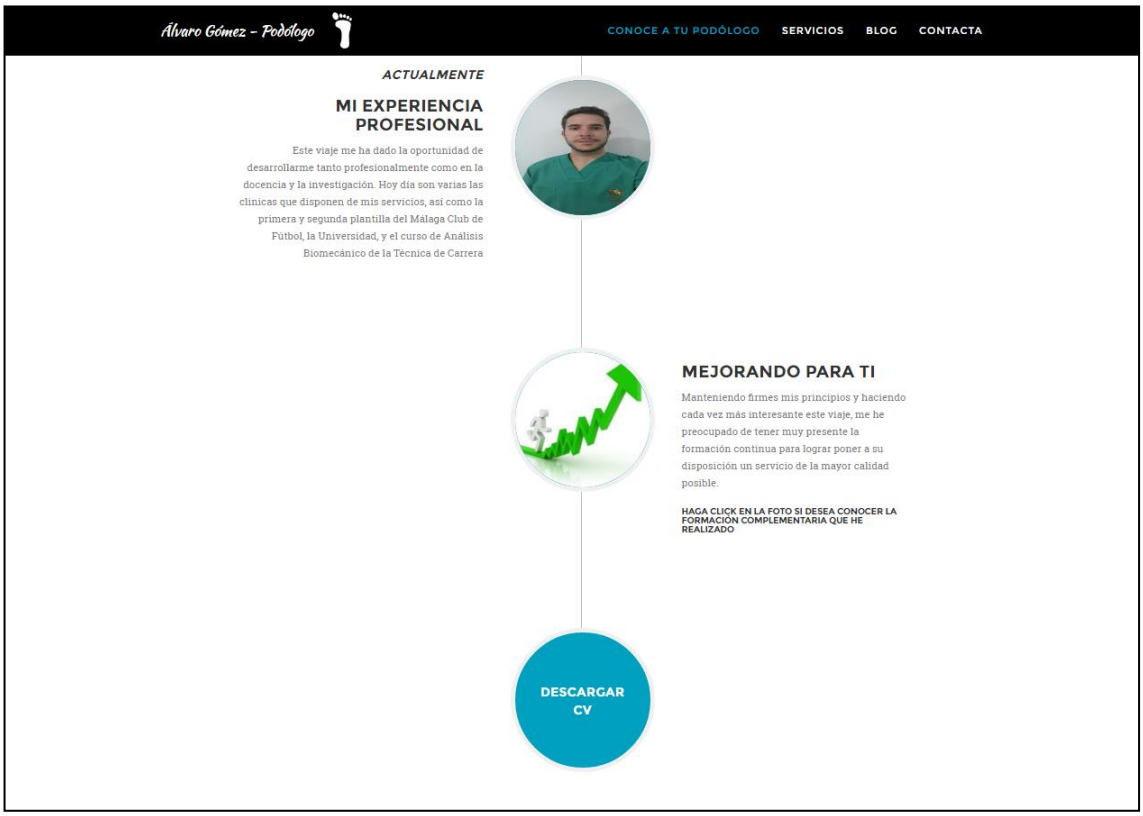
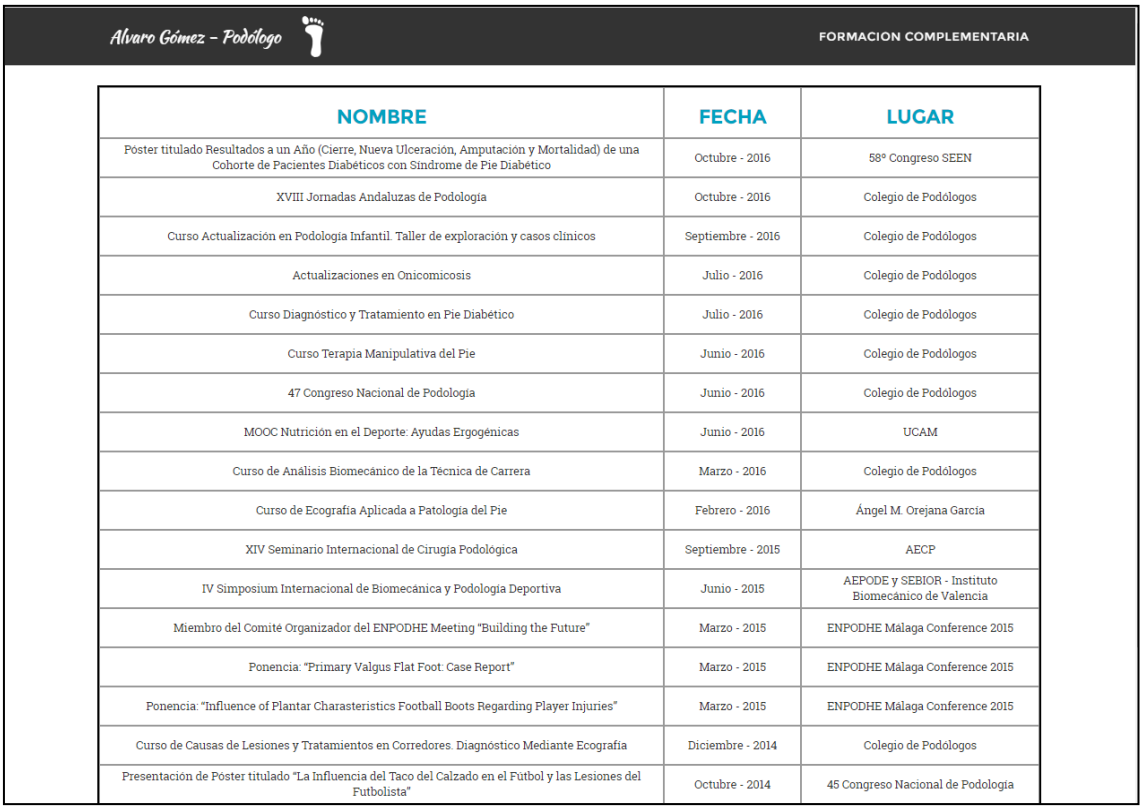


Figura 60. Anexo - Sección "Tu podólogo" - 2



NOMBRE	FECHA	LUGAR
Póster titulado Resultados a un Año (Cierre, Nueva Ulceración, Amputación y Mortalidad) de una Cohorte de Pacientes Diabéticos con Síndrome de Pie Diabético	Octubre - 2016	58º Congreso SEEN
XVIII Jornadas Andaluzas de Podología	Octubre - 2016	Colegio de Podólogos
Curso Actualización en Podología Infantil. Taller de exploración y casos clínicos	Septiembre - 2016	Colegio de Podólogos
Actualizaciones en Onicomicosis	Julio - 2016	Colegio de Podólogos
Curso Diagnóstico y Tratamiento en Pie Diabético	Julio - 2016	Colegio de Podólogos
Curso Terapia Manipulativa del Pie	Junio - 2016	Colegio de Podólogos
47 Congreso Nacional de Podología	Junio - 2016	Colegio de Podólogos
MOOC Nutrición en el Deporte: Ayudas Ergogénicas	Junio - 2016	UCAM
Curso de Análisis Biomecánico de la Técnica de Carrera	Marzo - 2016	Colegio de Podólogos
Curso de Ecografía Aplicada a Patología del Pie	Febrero - 2016	Ángel M. Orejana García
XIV Seminario Internacional de Cirugía Podológica	Septiembre - 2015	AECP
IV Simposium Internacional de Biomecánica y Podología Deportiva	Junio - 2015	AEPODE y SEBIOR - Instituto Biomecánico de Valencia
Miembro del Comité Organizador del ENPODHE Meeting "Building the Future"	Marzo - 2015	ENPODHE Málaga Conference 2015
Ponencia: "Primary Valgus Flat Foot: Case Report"	Marzo - 2015	ENPODHE Málaga Conference 2015
Ponencia: "Influence of Plantar Charasteristics Football Boots Regarding Player Injuries"	Marzo - 2015	ENPODHE Málaga Conference 2015
Curso de Causas de Lesiones y Tratamientos en Corredores. Diagnóstico Mediante Ecografía	Diciembre - 2014	Colegio de Podólogos
Presentación de Póster titulado "La Influencia del Taco del Calzado en el Fútbol y las Lesiones del Futbolista"	Octubre - 2014	45 Congreso Nacional de Podología

Figura 61. Anexo – Vista formación complementaria

Figura 62. Anexo – Descarga de curriculum

- **Servicios**

En esta sección se cuenta con un escaparate donde están los distintos servicios que podemos contratar. Si pasamos el ratón por encima de cada uno se iluminará y verá el nombre de cada uno. Si pulsamos iremos a la vista de cada uno de ellos con una explicación.

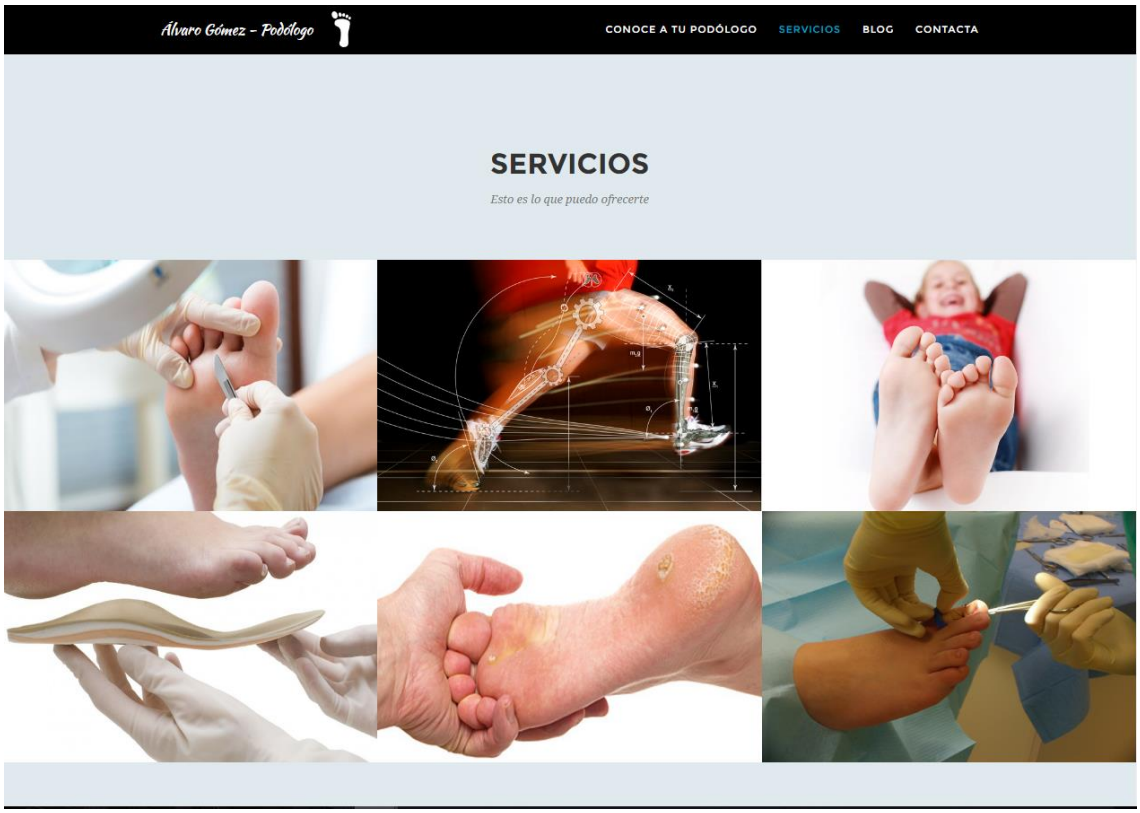


Figura 63. Anexo – Sección Servicios

A continuación se muestra una figura de cada una de las vistas de los servicios.

Figura 64. Anexo – Vista del servicio de quiropodología

Figura 65. Anexo – Vista del servicio de biomecánica

Figura 66. Anexo – Vista del servicio de podología infantil

Figura 67. Anexo – Vista del servicio de ortopodología

Figura 68. Anexo – Vista del servicio de pie de riesgo

Figura 69. Anexo – Vista del servicio de cirugía podológica

- **Blog**

En esta sección los clientes podrán ver los posts que el podólogo haya creado en su panel de administración para compartir con ellos. El podólogo podrá compartir artículos, noticias, y todo lo que le parezca útil y beneficioso para sus clientes.

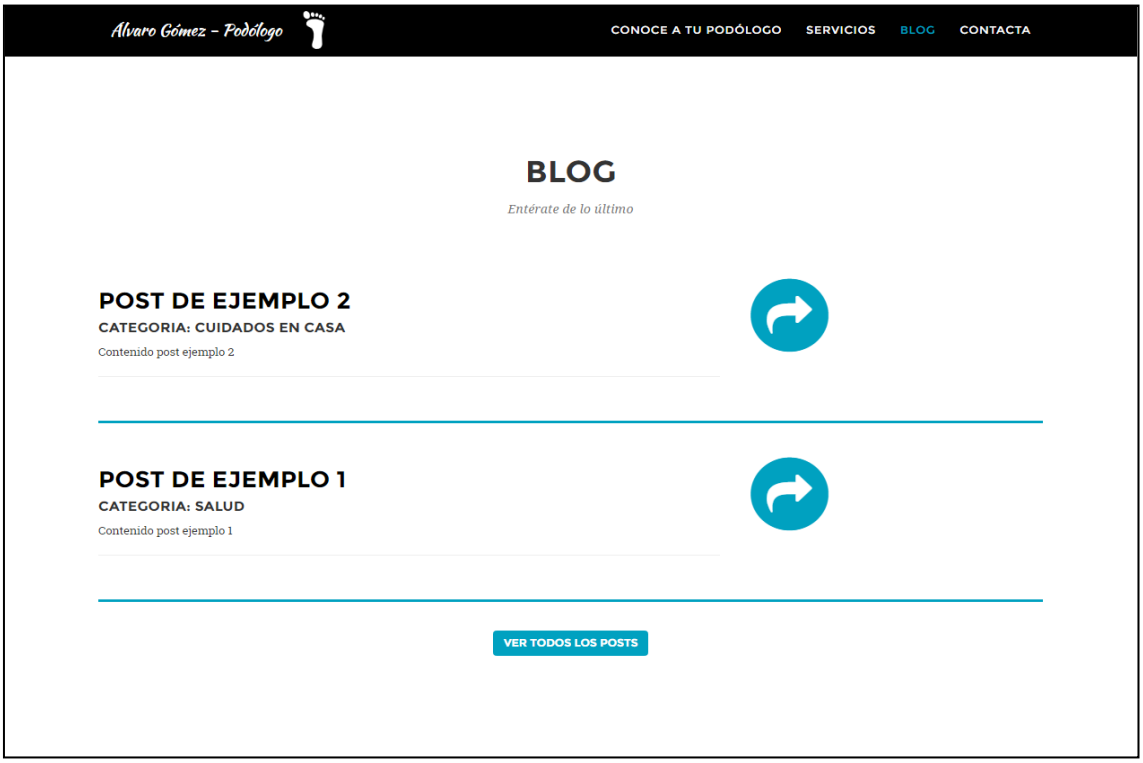


Figura 70. Anexo – Sección del Blog

En la página principal solo se podrán ver un número definido de posts, para no sobrecargarla. Podemos elegir entre ver todos los posts clasificados en categorías, o bien, ver cada post en particular, pulsando sobre su nombre o sobre el botón.

A continuación se muestran las figuras con estas dos posibilidades.

Figura 71. Anexo – Vista de post en solitario

Figura 72. Anexo – Vista de todos los post

- **Entidades en las que colaboro**

Esta sección está formada por una serie de imágenes de cada una de las entidades en las que colabora nuestro podólogo. Estas imágenes son enlaces hacia las páginas de dichas entidades.

Figura 73. Anexo – Sección Entidades

- **Contacta**

Estamos ante la última sección de la página principal de esta aplicación. Se trata de la sección donde el cliente puede mandarle un mensaje al podólogo directamente desde la web.

Figura 74. Anexo – Sección Contacta

Si pulsamos en el sobre se abrirá la vista del formulario para enviar el mensaje.

Figura 75. Anexo – Formulario de contacto

Una vez rellenemos todos los campos correctamente y pulsemos en enviar la aplicación nos redirigirá a una vista de agradecimiento desde la cual podremos volver a la página principal para seguir navegando.

Figura 76. Anexo – Vista de agradecimiento

Administrador

Identificación y Acceso al panel de administración.

Lo primero que tiene que hacer un administrador en la aplicación es autenticarse para acceder a su panel de administración. En este caso, se le proporciona un usuario y una contraseña al podólogo para que acceda a su panel.

Figura 77. Anexo – Panel de identificación

Una vez identificado, accederá a su panel de administración, donde dispondrá de las herramientas necesarias para la gestión de su aplicación

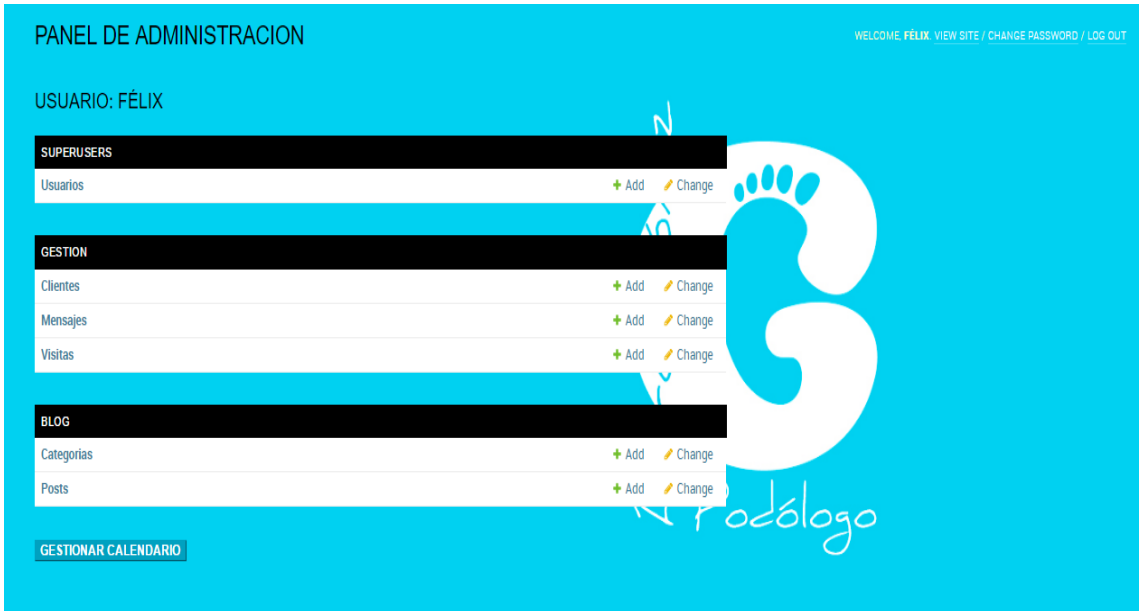


Figura 78. Anexo – Panel de administración

El panel de administración se divide en las siguientes funcionalidades.

- **Gestión de usuarios**

Esta funcionalidad ofrece la posibilidad de crear, modificar, consultar y borrar otros usuarios con rol administrador en la aplicación. En primera instancia solo existirá el podólogo con este rol, pero está en su mano crear otros usuarios con este rol si lo desea, o crear usuarios con menos privilegios para que puedan acceder a la web, como por ejemplo empleados. Para hacerlo, pulsamos sobre el enlace “usuarios”. Lo que nos llevará a la siguiente pantalla:

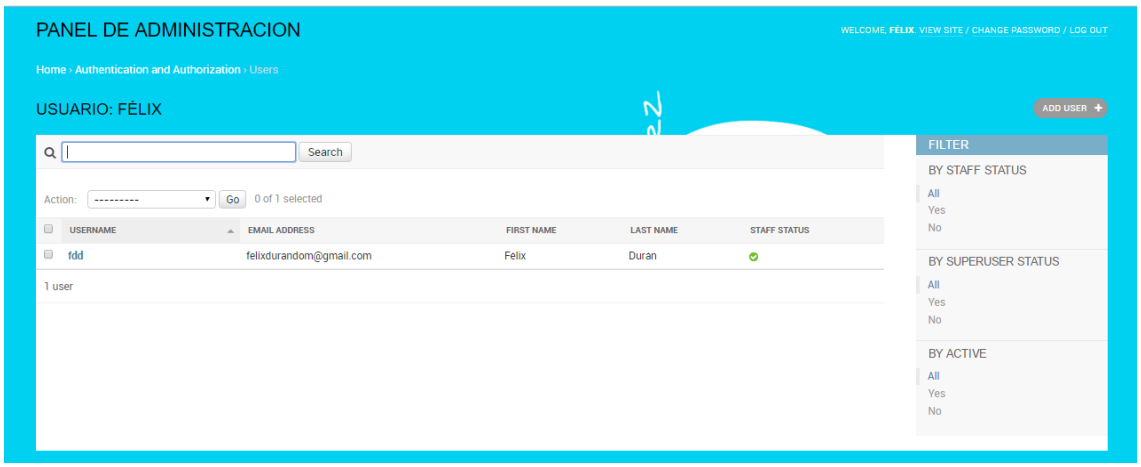


Figura 79. Anexo – Listado de Súper-usuarios

Como se puede comprobar, solo existe un administrador en la aplicación, en este caso es la cuenta la cual se ha utilizado para hacer todas las pruebas. En la práctica este usuario será el del podólogo.

Si queremos crear otro usuario solo tenemos que pulsar en el botón “Add”. Si lo hacemos nos llevará al formulario de creación de usuarios. Una vez dentro podremos rellenar los campos para crear nuevos usuarios a la aplicación.

Figura 80. Anexo – Formulario de creación de usuarios

Una vez rellenados estos campos ya podremos modificar los demás que tiene un usuario. Podemos elegir los permisos que tendrá cada usuario, por ejemplo, si queremos que un empleado pueda acceder a la aplicación pero solo pueda consultar los clientes, entraremos en su perfil y seleccionaremos los permisos necesarios para esto. También se pueden crear otros súper-usuarios con control total en la aplicación.

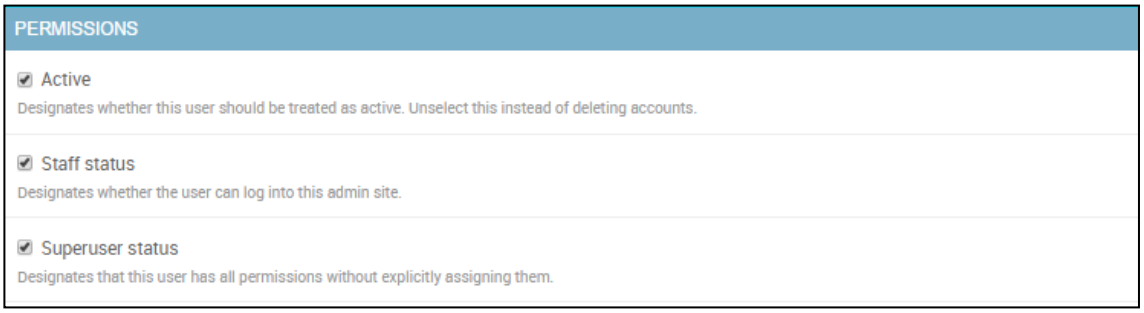


Figura 81. Anexo – Tipos de usuario

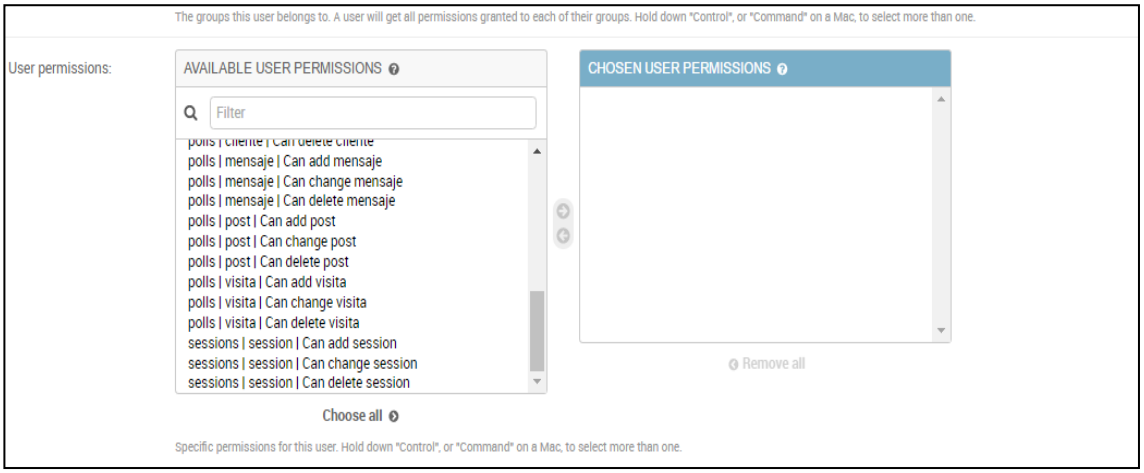


Figura 82. Anexo – Lista de permisos

- ## Gestión de clientes

En este apartado se aborda la funcionalidad de la gestión de los clientes del podólogo. Con esta herramienta podremos consultar nuestra lista de clientes, crear clientes nuevos, modificar los clientes ya existentes o borrar los que necesitemos.

Accedemos a la lista de cliente pulsando sobre el enlace de “clientes”

Figura 83. Anexo – Lista de clientes

Actualmente solo hay un cliente en la aplicación, vamos a crear otro. Pulsamos en el botón “Add” y nos llevará al formulario de creación de clientes.

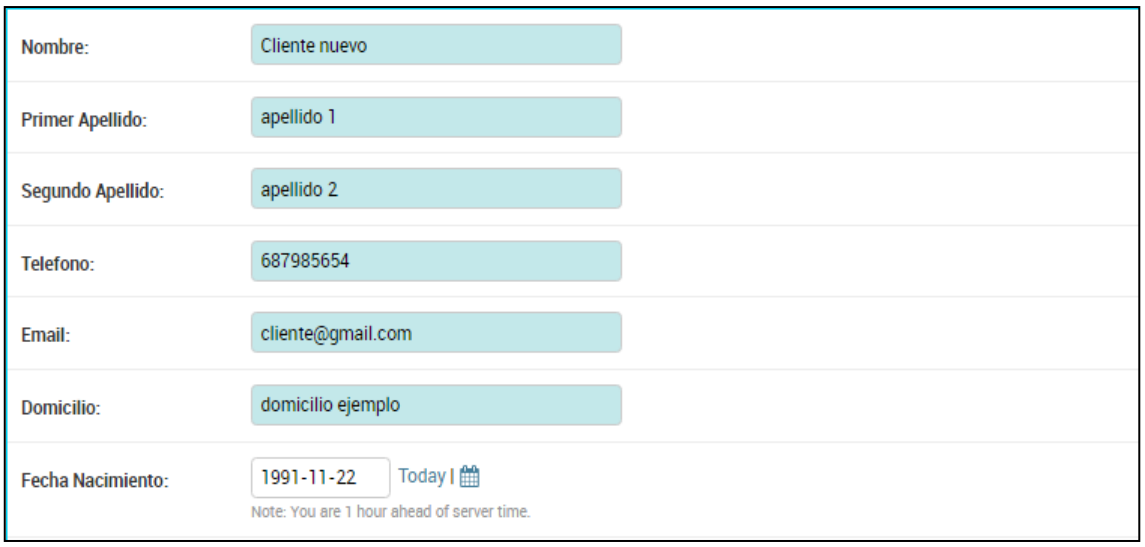


Figura 84. Anexo – Formulario de creación de clientes - 1

Figura 85. Anexo – Formulario de creación de clientes - 2

Una vez rellenados todos los campos pulsaremos en “save” y entonces veremos la lista de clientes con nuestro nuevo cliente ya creado.

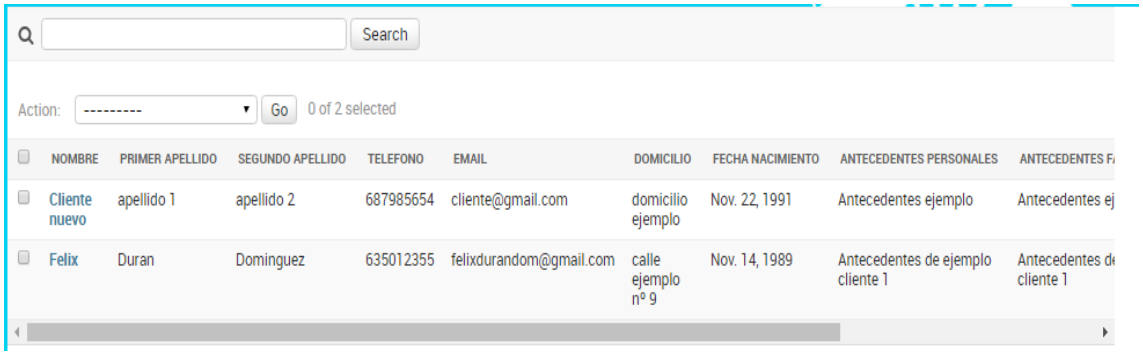


Figura 86. Anexo – Formulario de creación de clientes - 2

Si queremos modificar la información de cualquiera de los clientes no tenemos más que pulsar sobre uno de ellos y nos aparecerá de nuevo el formulario con los campos listos para ser editados.

- ## Seguimiento de las visitas

Esta funcionalidad engloba todo lo referente al seguimiento de los pacientes mediante la creación de visitas. El podólogo será el encargado de crear una visita cada vez que un cliente acuda a él. Si pulsamos sobre el enlace “visitas” se nos abrirá la ventana que contiene la lista de todas las visitas, ordenadas por fecha y filtradas por nombre o apellidos de los clientes.

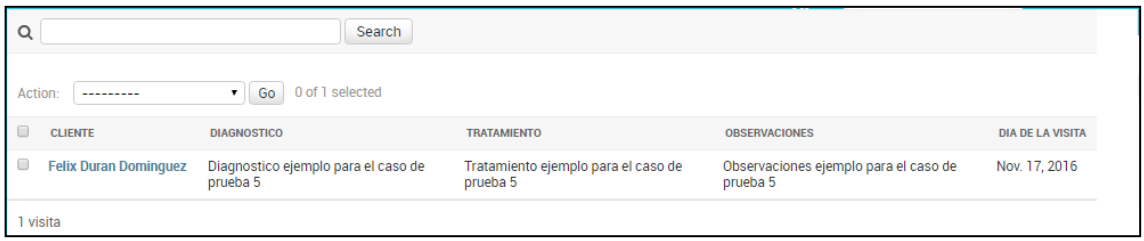


Figura 87. Anexo – Listado de visitas

En este ejemplo solo tenemos una visita creada para el cliente Félix Durán, creemos otra para este cliente. Pulsamos en “add” y rellenamos los campos.

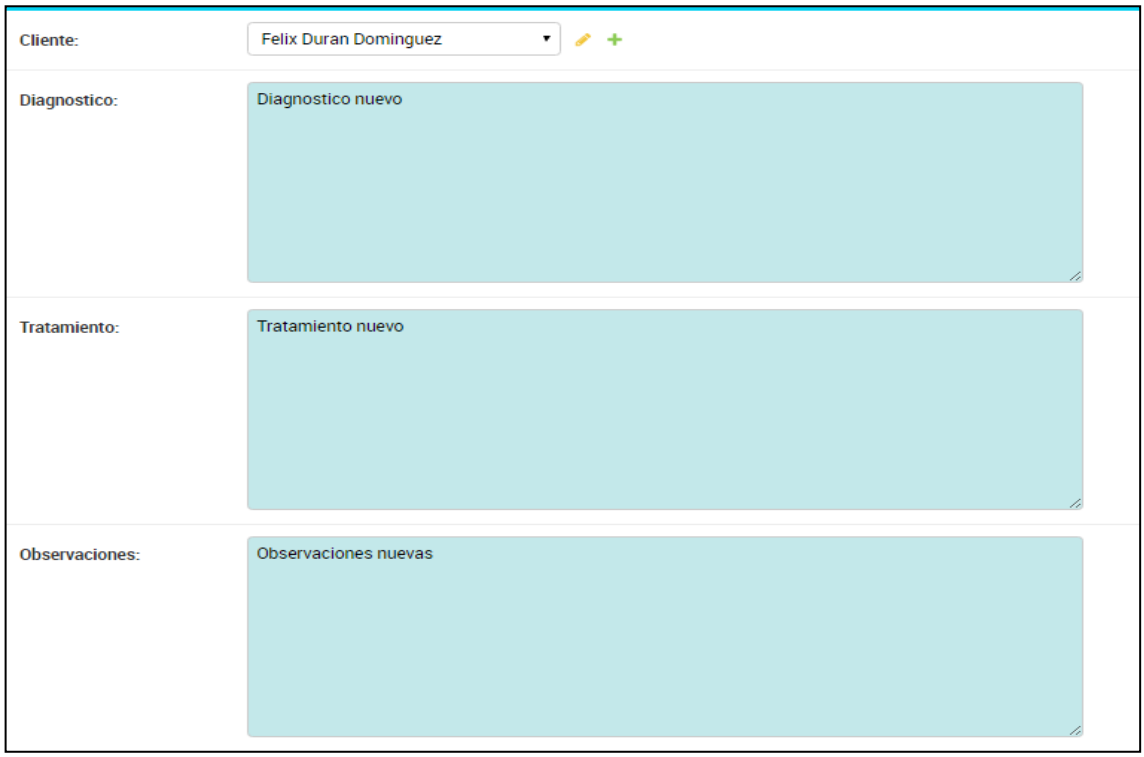


Figura 88. Anexo – Formulario de creación de visitas

Una vez rellenados los campos, pulsamos en el botón “save” para guardar los cambios y observamos en la lista de visitas que se ha creado correctamente.

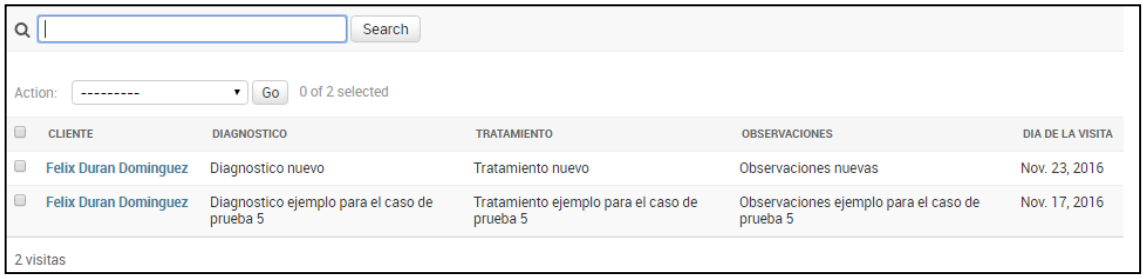


Figura 89. Anexo – Visita creada

Desde esta lista podemos también seleccionar vistas para borrarlas o entrar en ellas para modificar o consultar sus datos si lo necesitamos.

- ## Recepción y consulta de mensajes

Cuando un cliente accede a la web, tiene la posibilidad de mandarle un mensaje al podólogo mediante el formulario de contacto de la web. Este mensaje se le mandará automáticamente al email personal del podólogo, pero además, se guardará en la aplicación para tener todos los mensajes ordenador y almacenados para su posterior gestión y consulta.

Ya hemos visto en la primera parte de este anexo, dedicada al rol cliente, cmo estos pueden acceder al formulario para escribir mensajes. Veamos ahora cómo se ven estos mensajes en el panel de administración.

Figura 90. Anexo – Listado de mensajes

Como se puede observar, tenemos actualmente tres mensajes en la aplicación, entre ellos, el que se escribió en el apartado anterior en la especificación del uso del formulario de contacto.

Además de consultarlos podemos modificarlos o eliminarlos si lo necesitamos.

- ## Gestión del blog

Desde este panel de administración podemos generar contenido que será publicado en la web en la sección de blog. Desde la sección del Blog en el panel principal, podemos acceder tanto a las categorías como a los post. Los post están clasificados en categorías, con lo cual necesitaremos añadir dichos post a una categoría concreta. Veamos cómo hacerlo.

Primero entramos en “categorías” y creamos una categoría nueva.

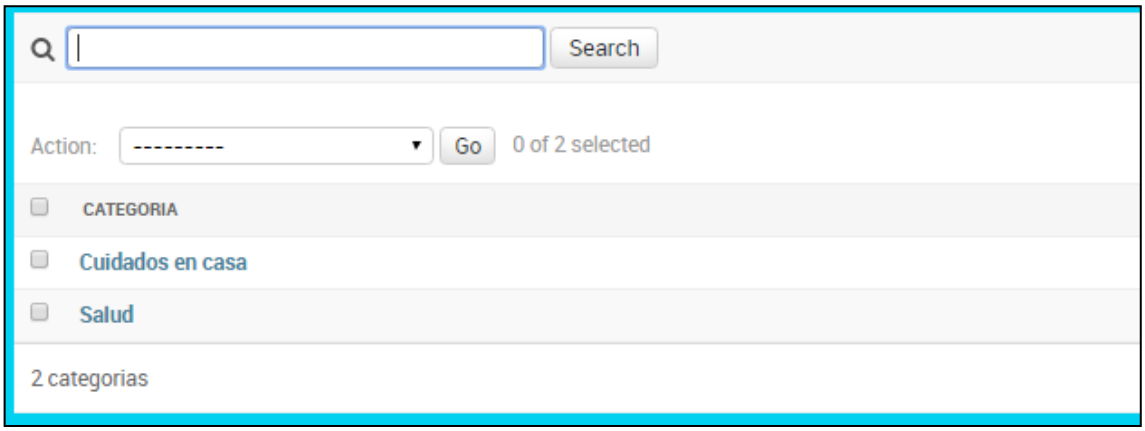


Figura 91. Anexo – Listado de categorías

Podríamos usar las categorías ya creadas, pero para ver cómo se hace vamos a crear una nueva. Accedemos al panel de creación de categorías e ingresamos el nombre de la categoría a crear.

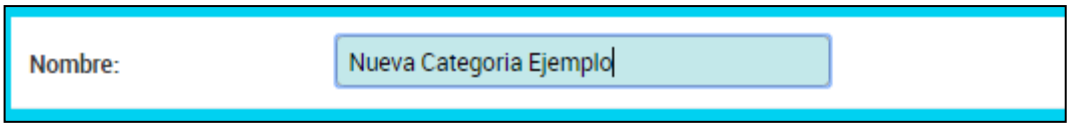


Figura 92. Anexo – Creación de una categoría

Una vez creada nuestra nueva categoría, nos dirigimos al panel de post de la aplicación.

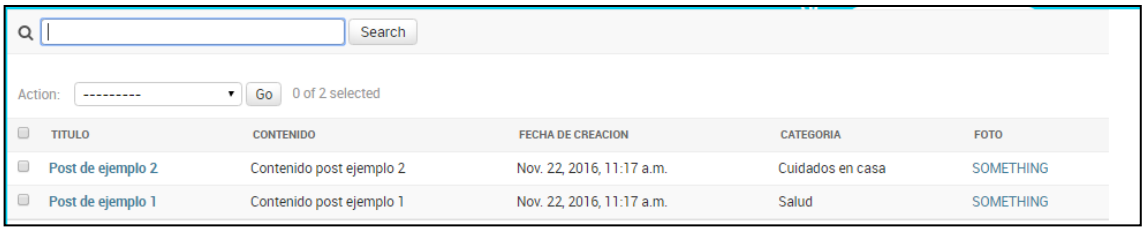


Figura 93. Anexo – Listado de posts

Entramos en el formulario de creación de posts pulsando sobre el botón “add” y rellenamos los campos.

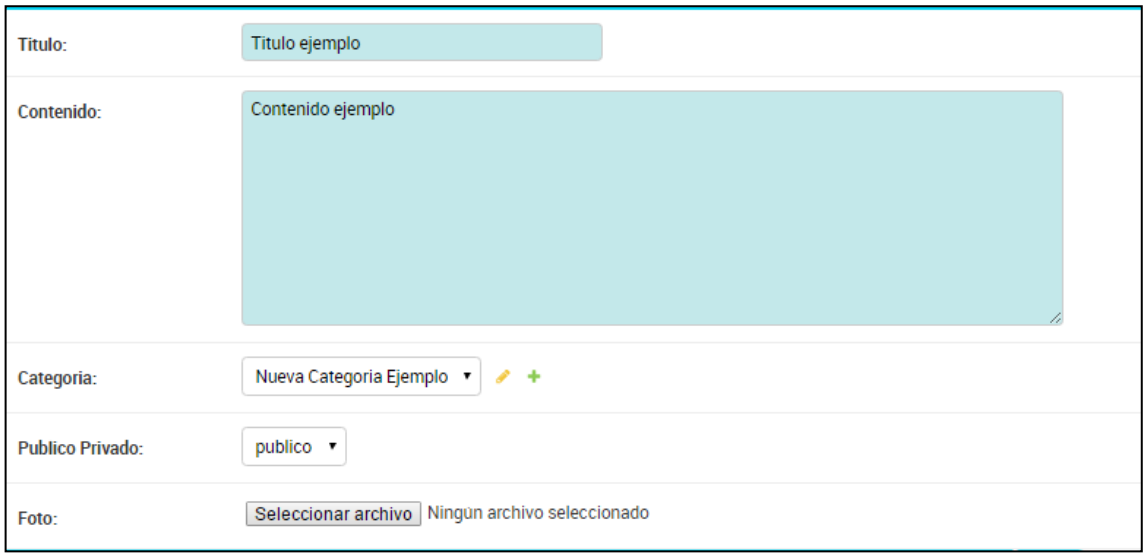


Figura 94. Anexo – Creación de un post

Pulsamos guardar y observamos que el post se guarda en la lista de post y además se publica en la web.

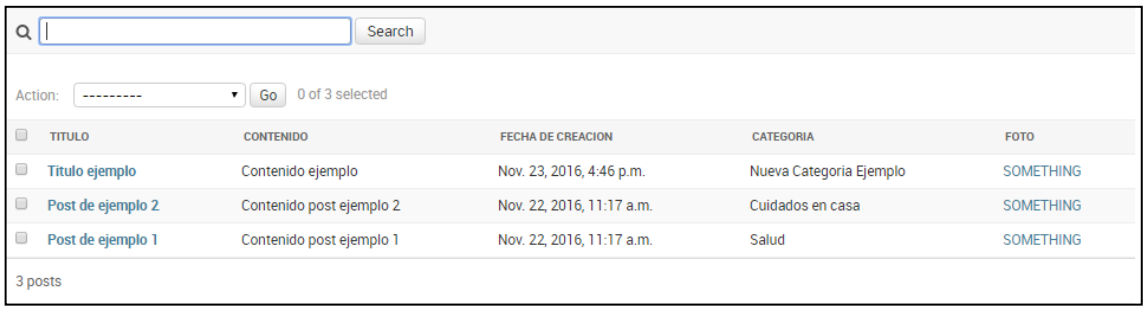


Figura 95. Anexo – Post almacenado

Figura 96. Anexo – Post publicado

- ## Gestión del calendario

Para acceder a la vista de gestión de calendario simplemente debemos pulsar sobre el botón que dice “gestionar calendario” en el panel principal.

Lo primero que nos aparecerá será un botón de autorización para que la aplicación pueda acceder al calendario de Google y muestre los eventos.

Figura 96. Anexo – Autorización de Google

Pulsamos sobre el botón y nos aparecerá la ventana de identificación de Google.

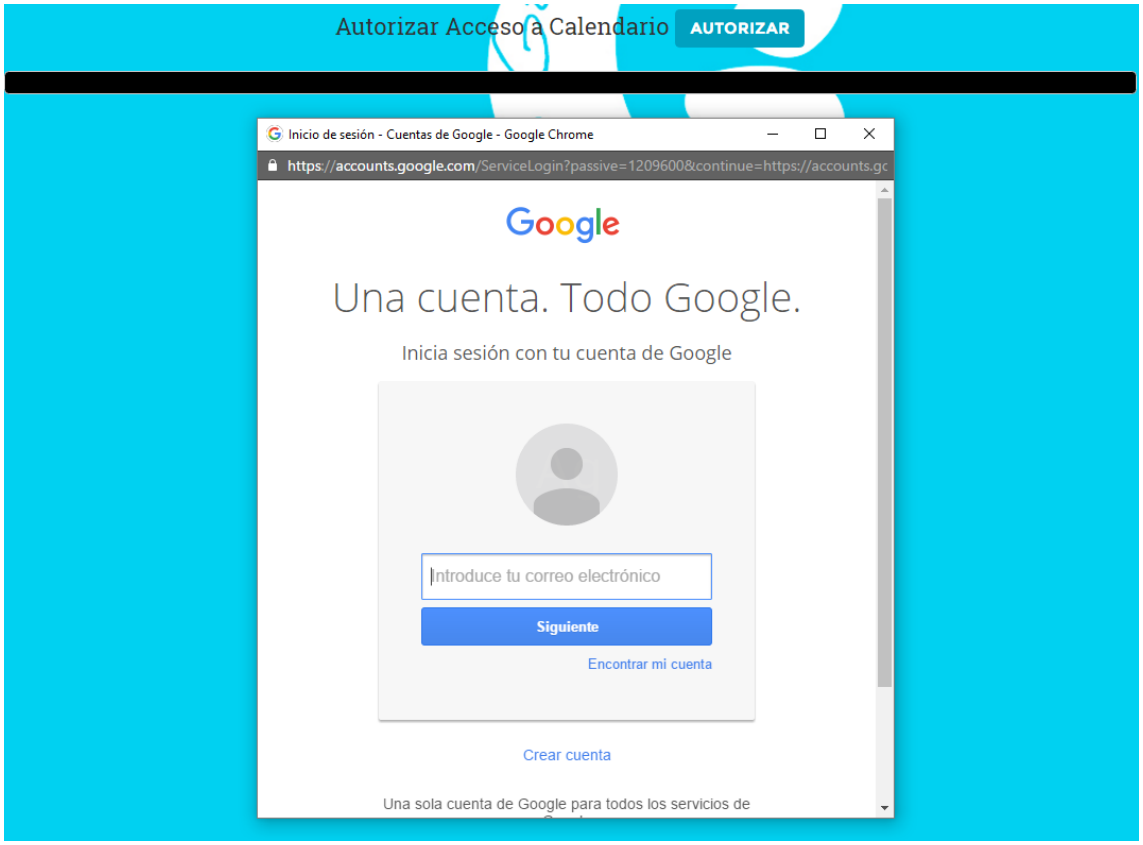


Figura 98. Anexo – Identificación en Google

Una vez nos identifiquemos accederemos a la siguiente vista.

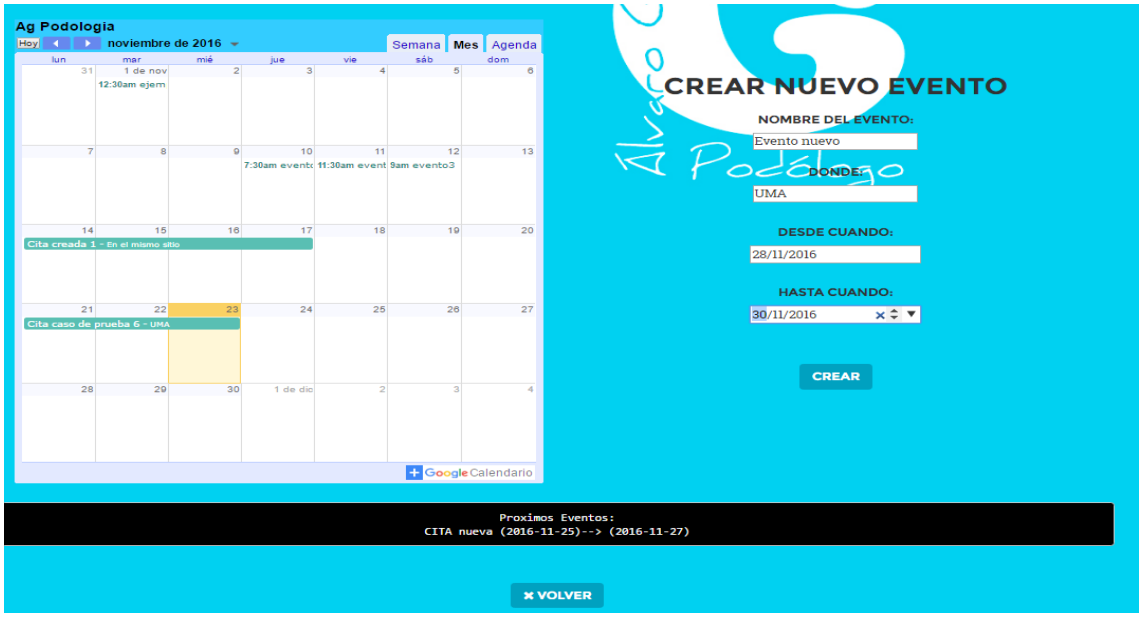


Figura 99. Anexo – Vista del calendario

Si queremos crear un nuevo evento simplemente tenemos que completar los datos del formulario y pulsar el botón crear. Esto recargará la página e insertará el evento en el calendario de Google para poder verlo desde cualquier dispositivo que tenga vinculado dicho calendario.

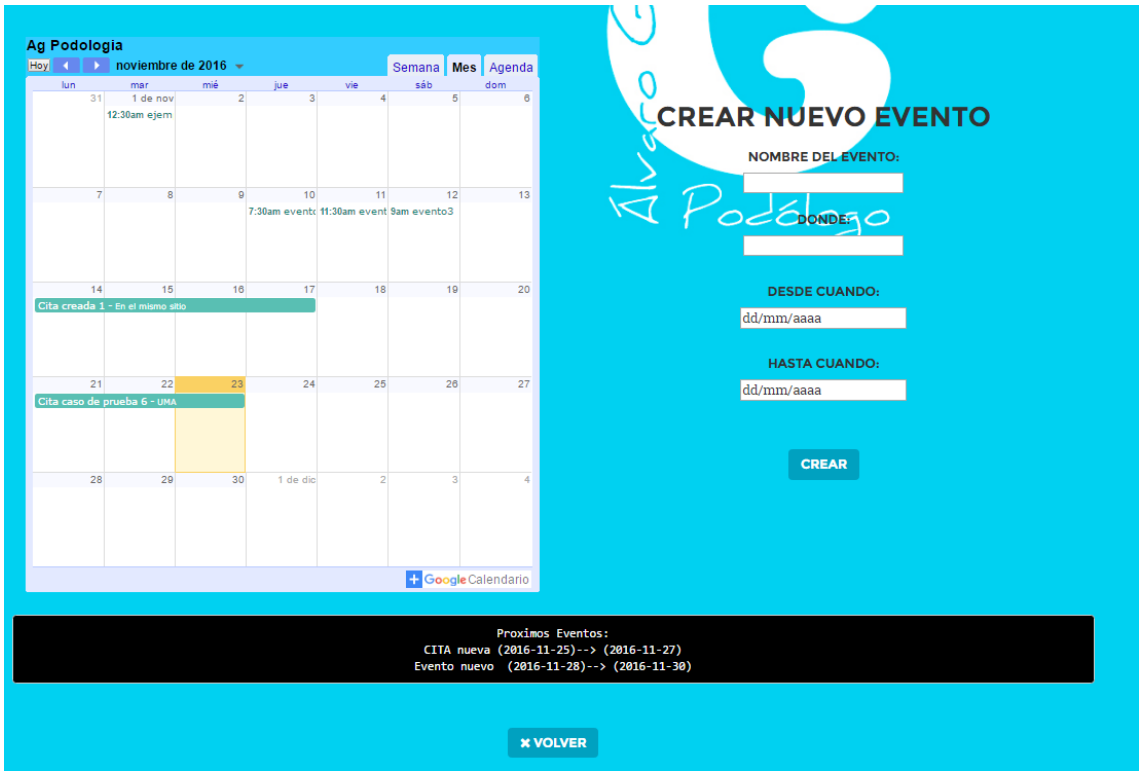


Figura 100. Anexo – Evento añadido