Geacex: Gestión de actividades extraescolares

Autor: Miguel Pascual Armero
Grado de ingeniería informática
Desarrollo web

Profesor colaborador: Gregorio Robles Martínez
Profesor: Santi Caballe Llobet

Fecha de entrega: 18/01/2023

FICHA DEL TRABAJO FINAL

Título del trabajo:	Geacex: gestión de actividades extraescolares
Nombre del autor:	Miguel Pascual Armero
Nombre del consultor:	Gregorio Robles Martínez
Fecha de entrega (mm/aaaa):	01/2023
Área del Trabajo Final:	Desarrollo web
Titulación:	*Grado de ingeniería informática*

Resumen del Trabajo:

La gestión de actividades extraescolares es un proceso habitual en los miles de centros educativos de nuestro país y que, además, implica a distintos actores de la comunidad educativa. A día de hoy, es una tarea que principalmente se realiza de forma manual y analógica, y que conlleva muchos quebraderos de cabeza.

Como respuesta a esta problemática, nace Geacex; una aplicación web centrada en la gestión de estas actividades, que conecta a distintos tipos de usuario para lograr sus objetivos.

Por un lado, tenemos a los centros educativos, que ofrecen sus instalaciones para llevas a cabo estas actividades y quieren ofrecérselas a sus alumnos. Por otro, las madres y padres quieren conocer los horarios de las actividades y poder inscribir a sus hijas e hijos. Por último, los proveedores ofrecen sus actividades a los centros para que puedan ponerse en marcha.

A nivel tecnológico, la aplicación se ha desarrollado en Python usando el *framework* Django y el sistema gestor de bases de datos PostgreSQL y, en el *frontend*, junto a HTML y CSS, se ha utilizado Bootstrap y Javascript.

Como resultado, hemos obtenido un producto mínimo viable (MVP) con un conjunto de funcionalidades suficiente para satisfacer los requerimientos establecidos.

Palabras clave:

Python, Django, actividades extraescolares, desarrollo web, aplicaciones web.

Abstract:

The management of extracurricular activities is a common process in the thousands of educational centers in our country and, in addition, involves different actors from the educational community. Today, it is a task that is mainly done manually and analogically, and that entails many headaches.

In response to this problem, Geacex was created; a web application focused on the management of these activities, which connects different types of users to achieve their goals.

On one hand, we have the educational centers, that offer their facilities to carry out these activities and want to offer them to their students. On the other, parents want to know the schedules of these activities and be able to register their children. Finally, the providers offer their activities to the centers so that they can get started with them.

At a technological level, the application has been developed in Python using the Django framework and the PostgreSQL database management system and, for the frontend, along with HTML and CSS, Bootstrap and Javascript have been used.

As a result, we have obtained a minimum viable product (MVP) with a set of functionalities sufficient to satisfy the established requirements.

Keywords:

Python, Django, extracurricular activities, web development, web apps.

Índice

Lista de figuras

Lista de tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

Como padre y miembro de la AMPA (asociación de madres y padres de alumnos) del colegio al que van mis hijas, el inicio del curso escolar trae una serie de lugares comunes a los que todas las familias nos tenemos que enfrentar: compra de material escolar, reuniones de inicio de curso, cambios de profesores, etc.

Entre todos ellos, también aparece la gestión y organización de actividades extraescolares, a las que se pueden inscribir los alumnos del centro. Estas actividades gozan de bastante éxito, ya que, al ceder el centro educativo las instalaciones, suelen tener unos precios más económicos que las que tienen lugar fuera del mismo. Además, sus horarios van acompasados con los del propio centro, lo que permite una mejor conciliación de la vida familiar.

Sin embargo, a pesar de su popularidad, su organización suele ser un quebradero de cabeza para las personas encargadas de ello, ya que hay que poner en contacto a distintos actores (dirección del colegio, proveedores de actividades, padres y madres) para que se pongan de acuerdo en aspectos como los horarios, inscripciones o pagos.

Haciendo un pequeño estudio de mercado a través de Internet, podemos ver que no existen muchas soluciones destinadas a la gestión de extraescolares; y las pocas que podemos encontrar tienen unos precios elevados. Por todo esto, he decidido enfocar mi proyecto en el desarrollo de una aplicación web centrada en la gestión de actividades extraescolares cuyas funciones iremos detallando a continuación.

1.2 Objetivos del Trabajo

El objetivo principal de este proyecto es desarrollar una aplicación web de gestión de actividades extraescolares, poniendo en práctica todos los conocimientos adquiridos (técnicas, metodologías, tecnologías, lenguajes de programación) a lo largo del grado de ingeniería informática.

Además, vamos a especificar una serie de subobjetivos que también queremos alcanzar y que vamos a dividir en dos partes. Primero, definiremos, a alto nivel, que funcionalidades nos va a ofrecer la aplicación que vamos a desarrollar. Em segundo lugar, describiremos, desde el punto de vista académico, cuáles son nuestras metas a la hora de realizar este TFG.

Empezaremos definiendo cuales son las funcionalidades que va a proporcionar la aplicación a los distintos actores implicados:

- **Centro educativo**:
  - Alta y acceso a la web.
  - Gestión de los horarios en su calendario de actividades.
  - Añadir las actividades ofertadas por los proveedores a su calendario.
  - Abrir/cerrar plazo de inscripción a las actividades.
- **Proveedor de actividades:**
  - Alta y acceso a la web.
  - Alta y configuración de actividades (cursos/edades, precio, alumnos mínimos/máximos).
- **Tutores (Padre/madre/familiar):**
  - Alta y acceso a la web.
  - Alta y gestión de alumnos.
  - Inscripción de los alumnos en actividades de su centro educativo.

Además, la aplicación ofrece otras funcionalidades como:

- Control de alumnos máximos y mínimos por grupo. El grupo tiene que llegar al mínimo de alumnos para que se forme y nunca podrá sobrepasar el número de alumnos máximo.
- Control de los plazos de inscripciones. Durante el plazo de inscripciones, los alumnos se podrán inscribir en los grupos de actividades que ofrezca su centro educativo. Cuando el plazo se cierre, solo se podrán realizar consultas.
- Optimizada y adaptada para móviles.

Desde el punto de vista académico, en relación a la elaboración del TFG, queremos conseguir los siguientes objetivos:

- Poner en práctica todos los conocimientos adquiridos durante la realización del grado en ingeniería informática y, especialmente, los obtenidos en el itinerario de ingeniería del software.
- Conocer y experimentar el ciclo de vida de una aplicación web, pasando por todas sus fases: análisis, diseño, implementación, pruebas y despliegue.
- Trabajar y profundizar en las tecnologías, metodologías y lenguajes de programación que vamos a usar durante el proyecto. Para ello, tendremos que realizar una labor de investigación y aprendizaje acorde a su utilización dentro del proyecto.
- Presentar un MVP (*Minimun viable product* o producto mínimo viable) que tenga un conjunto de funcionalidades suficientes para satisfacer las necesidades de nuestros supuestos clientes iniciales.

1.3 Enfoque y método seguido

El primer paso lógico a la hora de abordar un proyecto de este tipo consiste en llevar a cabo un estudio de mercado para conocer el estado del mismo. Una vez hecho, comprobamos que prácticamente no existen soluciones tecnológicas para el problema planteado y que las pocas que existen, aunque rápidas y sencillas como el uso de formularios de Google, presentan muchas limitaciones.

Por lo tanto, decidimos crear una aplicación web nueva con un conjunto inicial de requisitos mínimos que la hagan usable y funcional para los usuarios. Estas funcionalidades se pueden ver mejoradas y ampliadas en futuras iteraciones, las cuales quedan fuera del alcance de este trabajo.

Para adaptar el desarrollo de nuestra aplicación a los distintos hitos en forma de entregas, hemos decidido usar el modelo de ciclo de vida en cascada, en el cual vamos avanzando secuencialmente de una etapa a otra según las vamos terminando, como podemos ver en la ilustración 1. Aun así, ha sido necesario volver atrás en alguna fase para hacer alguna corrección necesaria.

Una vez seleccionadas las tecnologías que se iban a usar en la aplicación, comenzamos la fase de análisis, donde identificamos a los actores de nuestro sistema y detallamos los requisitos, tanto en historias de usuario como en casos de uso.

A continuación, en la fase de diseño, creamos el modelo entidad-relación para nuestra base de datos, definimos la arquitectura de nuestro sistema, tanto en la parte de *backend* como en la de *frontend*, y generamos unos prototipos para nuestros interfaces.

Por último, en la fase de implementación, una vez configurado nuestro entorno de desarrollo, pasamos a codificar todas las funcionalidades necesarias para satisfacer los requisitos definidos. Una vez finalizado, se facilita el código para poder ser ejecutado en local como imagen *Docker* y se despliega en el servicio de alojamiento web Pythonanywhere, para poder ser accedido *online* en cualquier momento.

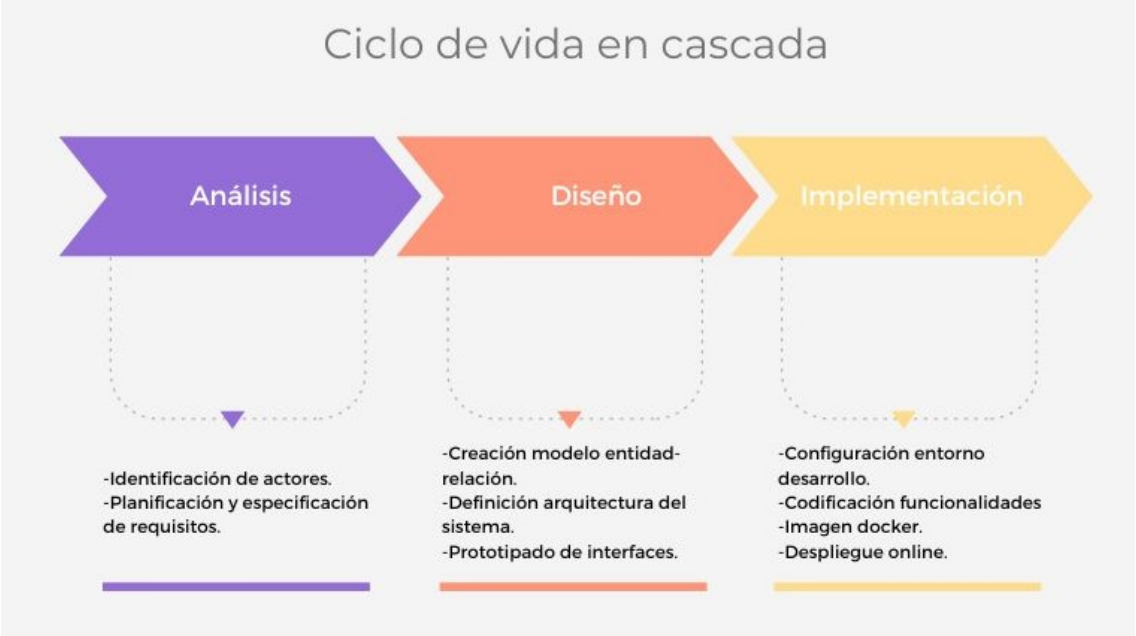

Ilustración 1. Ciclo de vida en cascada.

1.4 Planificación del Trabajo

La planificación del proyecto viene marcada por la entrega de 4 PECs a lo largo de la elaboración del mismo. Las fechas de estas PECs son las siguientes:

	Inicio	Entrega
PEC 1. Plan de trabajo	28/09/2022	11/10/2022
PEC 2. Análisis y diseño	12/10/2022	11/11/2022
PEC 3. Implementación	12/11/2022	02/01/2023
PEC 4. Memoria y presentación	03/01/2023	19/01/2023

Tabla 1. Fechas PECs.

Debido a esto, hemos decidido ajustar nuestras tareas a los plazos de entrega de las PECs. Dentro de cada gran tarea, hemos creado diferentes subtareas que nos irán marcando el ritmo de trabajo adecuado. A continuación, se muestra el desglose de tareas y su duración:

Tarea

Nombre	Fecha de inicio	Fecha de fin	Duración
Plan de trabajo	28/9/22	10/10/22	13
Búsqueda e investigación sobre el proyecto	28/9/22	2/10/22	5
Definición de requisitos y objetivos	3/10/22	6/10/22	4
Definición de tareas y calendario	3/10/22	6/10/22	4
Elaboración y presentación del plan	7/10/22	10/10/22	4
Análisis y diseño	12/10/22	11/11/22	31
Definición de requisitos	12/10/22	15/10/22	4
Definición de actores y casos de uso	16/10/22	18/10/22	3
Elaboración de casos de usos	19/10/22	27/10/22	9
Diseño de base de datos	28/10/22	30/10/22	3
Diseño de prototipo pantallas	31/10/22	8/11/22	9
Diagrama de arquitectura	9/11/22	10/11/22	2
Revisión y entrega del documento	10/11/22	11/11/22	2
Implementación	12/11/22	2/1/23	52
Instalación y configuración del entorno	12/11/22	13/11/22	2
Implementación backend	14/11/22	15/12/22	32
Implementación frontend	21/11/22	20/12/22	30
Elaboración documentación	15/12/22	27/12/22	13
Revisión y entrega	28/12/22	2/1/23	6
Memoria y presentación	3/1/23	19/1/23	17
Elaboración memoria	3/1/23	11/1/23	9
Elaboración video presentación	12/1/23	15/1/23	4
revisión y entrega final	16/1/23	19/1/23	4

Ilustración 2. Planificación de tareas.

También se ha creado un diagrama para mostrar las tareas de forma visual. Se adjunta a este documento en el Anexo I. Diagrama de Gant.

1.4.1. Análisis de riesgos

El análisis de riesgos es una fase muy importante a la hora de planificar un proyecto. El hecho de tener claro a qué riesgos nos podemos enfrentar, puede ayudarnos a prevenirlos o anticiparnos a sus consecuencias. A continuación, vamos a identificar qué riesgos nos vamos a poder encontrar a lo largo de este proyecto:

Riesgo	Descripción	Probabilidad	Gravedad
Alcance del proyecto incierto	Exceder el alcance inicial del proyecto debido a unos objetivos mal definidos.	Baja	Media
Mala planificación temporal	Las tareas pueden requerir más tiempo del planificado y podemos sufrir retrasos en nuestro calendario.	Media	Media
Falta de conocimiento de alguna tecnología o lenguaje	Usar tecnologías o lenguajes de programación de los que no tenemos mucho conocimiento puede hacer que nos retrasemos o que tengamos que realizar cambios operativos.	Baja	Alta
Problemas o compromisos personales	Problemas personales o familiares, enfermedad, compromisos ineludibles, que nos pueden producir retrasos.	Media	Baja
Cambios laborales	Aumento de la carga de trabajo, cambios de horarios, etc.	Baja	Baja
Problemas con los recursos en uso	Daños o desperfecto en el hardware, cortes de suministro, accidentes, etc.	Baja	Baja

Tabla 2. Análisis de riesgos.

De forma más visual, podemos colocar nuestros riesgos en una matriz de riesgos:

		Gravedad		
		Baja	Media	Alta
Probabilidad	Alta			
	Media	**-Problemas o compromisos personales.**	**-Mala planificación temporal.**	
	Baja	**-Cambios laborales. -Problemas con los recursos en uso.**	**-Alcance del proyecto incierto.**	**-Falta de conocimiento de alguna tecnología o lenguaje.**

Tabla 3. Matriz de riesgos.

1.5 Breve sumario de productos obtenidos

Los productos obtenidos a lo largo del proyecto y que serán entregados son los siguientes:

- Código fuente alojado en Github:
  https://github.com/MPascualArm/geacex
- Instrucciones para ejecución en local con Docker.
- Despliegue online en Pythonanywhere:
  https://mpascual.eu.pythonanywhere.com/
- Memorial final.
- Presentación del proyecto. Diapositivas y video.

1.6 Breve descripción de los otros capítulos de la memoria

- **Capítulo 2 "Análisis y diseño".** En este capítulo, abordamos la selección de tecnologías. Luego, la identificación y especificación de actores y requisitos, que plasmamos en historia de usuario y casos de uso. A continuación, definimos nuestro modelo entidad-relación y la arquitectura de nuestro sistema. Por último, presentamos los prototipos de nuestros interfaces.
- **Capítulo 3 "Implementación".** Empezamos este apartado definiendo nuestro entorno de desarrollo y la estructura de nuestro proyecto. También hablamos sobre el funcionamiento de nuestra aplicación. Para terminar, explicamos la ejecución en local del proyecto y su acceso vía web.
- **Capítulo 4 "Conclusiones".** Aquí presentaremos los resultados obtenidos, los cambios a lo largo del trabajo y algunas ideas finales.
- **Capítulo 5 "Glosario".** Catálogo de términos usados a lo largo de la memoria.
- **Capítulo 6 "Bibliografía".** Listado de recursos y materiales empleados para el trabajo.
- **Capítulo 7 "Anexos".** Contenidos referenciados y agregados al final de la memoria.

2. Análisis y diseño

2.1. Requisitos del proyecto

2.1.1. Actores

Antes de empezar a detallar los requisitos, necesitamos identificar a los actores que van a interactuar con el sistema. En nuestro caso, hemos identificado 4 actores principales:

Actor	Descripción
Administrador	La figura encargada de administrar y gestionar la aplicación web.
Centro educativo	Este actor será el representante de un centro educativo. Los centros educativos podrán darse de alta, gestionar sus calendarios y los grupos formados para cada actividad que elijan.
Tutor	Los tutores son los padres, madres o tutores legales de los alumnos. Podrán darlos de alta en la aplicación, consultar los distintos grupos por horario y actividad y realizar inscripciones.
Proveedor de actividades	Este actor será el representante de la empresa que organiza las actividades en los centros educativos. Podrá dar de alta sus actividades para que sean usadas en los grupos de los colegios.

Tabla 4. Actores del sistema.

2.1.2. Historias de usuario

Las historias de usuario son una forma informal de representar los requisitos de un proyecto, de forma breve y usando un lenguaje coloquial. A continuación, detallamos algunas de las historias de usuarios más relevantes para el desarrollo de nuestra aplicación:

Historia de usuario	Como centro educativo quiero poder crear y gestionar un calendario semanal de actividades para que puedan usarlo tanto los proveedores como los alumnos.
Descripción	El centro podrá crear un calendario para las actividades del curso académico actual, definiendo una fecha inicial y final de las mismas. Este se distribuirá en una planificación semanal por franjas de una hora donde se podrán asignar actividades.

Tabla 5. Historia de usuario I.

Historia de usuario	Como centro educativo quiero controlar el plazo de inscripciones a las actividades para poder abrirlo y cerrarlo cuando desee.
Descripción	Antes de inicio de las actividades, el centro podrá abrir el plazo de inscripciones. Durante este plazo, los tutores podrán inscribir a los alumnos en los grupos creados por el centro educativo al que pertenecen. Una vez cerrado, solo podrán consultarlo.

Tabla 6. Historia de usuario II.

Historia de usuario	Como tutor quiero apuntar a mis hijos o hijas a las actividades ofrecidas por mi centro para que puedan realizarlas.
Descripción	El tutor debe poder darse alta en la web a él y a los alumnos correspondientes. Consultar los grupos por actividad creados por su centro y poder inscribirlos si el plazo está abierto.

Tabla 7. Historia de usuario III.

Historia de usuario	Como tutor quiero poder consultar la información referente a los grupos de mi centro para conocer sus detalles.
Descripción	El tutor debe saber si el grupo llega al mínimo de alumnos y se forma o si está cerrado por llegar al máximo. También ver información como el precio o el horario.

Tabla 8. Historia de usuario IV.

Historia de usuario	Como proveedor de actividades quiero poder ofrecer mis actividades a los centros educativos para que puedan ofertarlas.
Descripción	El proveedor de actividades debe poder darse de alta en la web y añadir las actividades que oferta con su información asociada.

Tabla 9. Historia de usuario V.

Historia de usuario	Como proveedor de actividades quiero conocer que grupos se han formado en los distintos centros educativos para poder organizar su realización.
Descripción	El proveedor debe poder consultar la información relativa a los distintos grupos de los centros donde se oferte su actividad.

Tabla 10. Historia de usuario VI.

Historia de usuario	Como administrador quiero notificar a los distintos usuarios de los eventos importantes del sistema para que estén informados de manera rápida.
Descripción	El sistema notificará por correo electrónico a los distintos usuarios de la aplicación de diferentes eventos que ocurran en el sistema (alta/baja de usuarios, grupos completos o no, etc.).

Tabla 11. Historia de usuario VII.

2.1.3. Casos de uso

Un caso de uso es la descripción de una actividad o una acción que tiene que realizar un actor para llevar a cabo un proceso. Los casos de uso nos permiten documentar de forma más formal y detallada el conjunto de requisitos de un proyecto.
A continuación, vamos a detallar los casos de uso generados a partir de las historias de usuario especificadas en el apartado anterior, para así obtener una especificación más concreta y precisa que nos sirva para el desarrollo de nuestra aplicación.

Nombre	CU01 – Registrar nueva cuenta.
Actor principal	Centro educativo, tutor o proveedor.
Precondiciones	La cuenta no existe aún en el sistema.
Postcondiciones	La nueva cuenta queda almacenada en el sistema.
Escenario principal de éxito	1. El usuario accede a la pantalla de registro. 2. El usuario rellena los datos obligatorios según que actor sea: -Centro educativo: Nombre, provincia, localidad y correo electrónico. -Tutor: DNI, nombre, primero apellido, segundo apellido, correo electrónico y código de centro (proporcionado por el propio centro). -Proveedor: NIF, nombre y correo electrónico. 3. El usuario elige una contraseña que cumpla con la complejidad mínima requerida. 4. El usuario pulsa el botón “aceptar”. 5. El sistema muestra la página de perfil del usuario. 6. El sistema notifica de la creación de la nueva cuenta.
Extensiones	2.a. No se rellena algún campo obligatorio. 2.a.1. El sistema avisa al usuario que debe de cumplimentar todos los campos y vuelve al paso 2. 3.a. El usuario no escoge una contraseña que cumpla los requisitos de complejidad. 3.a.1. El sistema avisa al usuario de que debe elegir una contraseña correcta y vuelve al paso 3.

Tabla 12. CU01. Registrar nueva cuenta.

Nombre	CU02 – Identificarse en el sistema.
Actor principal	Centro educativo, tutor o proveedor.
Precondiciones	La cuenta existe en el sistema.
Postcondiciones	El sistema muestra la página de inicio del usuario.
Escenario principal de éxito	1. El usuario accede a la pantalla de identificación. 2. El usuario introduce sus datos de identificación (correo electrónico y contraseña).

	3. El usuario pulsa el botón “Aceptar”. 4. El sistema muestra la página de inicio del usuario.
Extensiones	2.a. Los datos de identificación son incorrectos. 2.a.1. El sistema avisa al usuario de que debe de introducir otra vez los datos y vuelve al paso 2.

Tabla 13. CU02. Identificarse en el sistema.

Nombre	CU03 – Modificar datos de la cuenta.
Actor principal	Centro educativo, tutor o proveedor.
Precondiciones	El usuario está identificado en el sistema.
Postcondiciones	Los datos modificados se almacenan en el sistema.
Escenario principal de éxito	1. El usuario accede a su perfil de usuario. 2. El sistema muestra sus datos actuales. 3. El usuario modifica sus datos. 4. El sistema almacena y muestra sus nuevos datos.
Extensiones	3.a. El usuario introduce un dato erróneo o vacío. 3.a.1. El sistema muestra un error y vuelve al paso 2.

Tabla 14. CU03. Modificar datos de la cuenta.

Nombre	CU04 – Salir del sistema.
Actor principal	Centro educativo, tutor o proveedor.
Precondiciones	El usuario está identificado en el sistema.
Postcondiciones	El sistema muestra la página de inicio de la aplicación.
Escenario principal de éxito	1. El usuario pulsa el botón de “salir”. 2. El sistema cierra la sesión del usuario. 3. El sistema muestra la página de inicio de la aplicación.
Extensiones	-

Tabla 15. CU04. Salir del sistema.

Nombre	CU05 – Crear calendario de actividades.
Actor principal	Centro educativo.
Precondiciones	El centro educativo está identificado en el sistema.
Postcondiciones	El nuevo calendario se almacena en el sistema y es accesible por los tutores del centro.
Escenario principal de éxito	1. El centro accede a su sección de calendarios. 2. El centro selecciona “crear nuevo calendario”. 3. El sistema muestra la pantalla de alta de calendario. 4. El centro introduce los datos del calendario: año, día de inicio, día de fin y franjas horarias para las actividades. 5. El sistema muestra el calendario en la sección de calendarios.
Extensiones	-

Tabla 16. CU05. Crear calendario de actividades.

Nombre	CU06 – Eliminar calendario de actividades.
Actor principal	Centro educativo.
Precondiciones	El centro educativo está identificado en el sistema.
Postcondiciones	El nuevo calendario se elimina del sistema y de los perfiles de los tutores del centro.
Escenario principal de éxito	1. El centro accede a su sección de calendarios. 2. El sistema muestro todos sus calendarios. 3. El centro selecciona un calendario. 4. El sistema muestra el calendario. 5. El centro elige eliminar el calendario. 6. El sistema muestra un mensaje de advertencia. 7. El centro acepta el mensaje de advertencia. 8. El sistema eliminar el calendario y vuelve a la sección de calendarios.
Extensiones	7.a. El centro no acepta el mensaje de advertencia. 7.a.1. Volvemos al paso 4.

Tabla 17. CU06. Eliminar calendario de actividades.

Nombre	CU07 – Modificar calendario de actividades.
Actor principal	Centro educativo.
Precondiciones	El centro educativo está identificado en el sistema.
Postcondiciones	Los datos modificados del calendario se almacenan en el sistema.
Escenario principal de éxito	1. El centro accede a su sección de calendarios. 2. El sistema muestro todos sus calendarios. 3. El centro selecciona un calendario. 4. El sistema muestra el calendario. 5. El centro modifica los datos del calendario. 6. El sistema muestra un mensaje de advertencia. 7. El centro acepta el mensaje de advertencia. 8. El sistema guarda los datos modificados y vuelve a la sección de calendarios.
Extensiones	7.a. El centro no acepta el mensaje de advertencia. 7.a.1. Volvemos al paso 4.

Tabla 18. CU07. Modificar calendario de actividades.

Nombre	CU08 – Asignar actividades a un calendario.
Actor principal	Centro educativo.
Precondiciones	El centro educativo está identificado en el sistema. El calendario existe en el sistema. Las actividades existen en el sistema.
Postcondiciones	Los grupos (actividad + franja horaria) creados quedan almacenados en el sistema.
Escenario principal de	1. El centro educativo accede a un calendario propio.

éxito	2. El centro educativo selecciona una franja horaria libre del calendario. 3. El sistema muestra las actividades disponibles. 4. El centro educativo selecciona una actividad. 5. El sistema crea un grupo con la actividad y la franja horaria. 6. El centro educativo nombra al grupo. 7. El sistema almacena el nuevo grupo.
Extensiones	-

Tabla 19. CU08. Asignar actividades a un calendario.

Nombre	CU09 – Eliminar un grupo del calendario.
Actor principal	Centro educativo.
Precondiciones	El centro educativo está identificado en el sistema. El calendario existe en el sistema. El grupo existe en el sistema.
Postcondiciones	El grupo se elimina del sistema y la franja horaria queda libre. Los alumnos inscritos en el grupo quedan libres.
Escenario principal de éxito	1. El centro educativo accede a un calendario propio. 2. El sistema muestra el calendario. 3. El centro educativo selecciona un grupo del calendario. 4. El centro educativo selecciona eliminar el grupo. 5. El sistema muestra un mensaje de advertencia. 6. El centro acepta el mensaje de advertencia. 7. El sistema elimina el grupo. 8. El sistema muestra el calendario. 9. El sistema notifica a los tutores de los alumnos inscritos.
Extensiones	6.a. El centro no acepta el mensaje de advertencia. 6.a.1. Volvemos al paso 2.

Tabla 20. CU09. Eliminar un grupo del calendario.

Nombre	CU10 – Abrir/cerrar el plazo de inscripciones de un calendario.
Actor principal	Centro educativo.
Precondiciones	El centro educativo está identificado en el sistema. El calendario existe en el sistema.
Postcondiciones	El plazo de inscripciones queda abierto o cerrado para ese calendario.
Escenario principal de éxito	1. El centro educativo accede a un calendario propio. 2. El centro educativo selecciona abrir el plazo de inscripciones. 3. El sistema almacena la selección. 4. El sistema notifica a los tutores adscritos al centro.
Extensiones	2.a. El centro selecciona cerrar el plazo de inscripciones.

	2.a.1. El sistema notifica a los tutores si los grupos de sus alumnos se han completado o no (min. alumnos). 2.a.2. El sistema notifica a los proveedores si los grupos de sus actividades se han completado o no (min. alumnos).

Tabla 21. CU10. Abrir/cerrar el plazo de inscripciones de un calendario.

Nombre	CU11 – Dar de alta un alumno.
Actor principal	Tutor.
Precondiciones	El tutor está identificado en el sistema.
Postcondiciones	Se crea el alumno en el sistema.
Escenario principal de éxito	1. El tutor accede a su sección de alumnos. 2. El tutor selecciona dar de alta un alumno. 3. El sistema muestra la pantalla de alta de alumno. 4. El tutor introduce los datos del alumno: nombre, primer apellido, segundo apellido, edad y curso. 5. El sistema muestra al alumno en la sección de alumnos. 6. El sistema notifica al tutor del alumno.
Extensiones	-

Tabla 22. CU11. Dar de alta un alumno.

Nombre	CU12 – Eliminar un alumno.
Actor principal	Tutor.
Precondiciones	El tutor está identificado en el sistema.
Postcondiciones	Se elimina al alumno en el sistema.
Escenario principal de éxito	1. El tutor accede a su sección de alumnos. 2. El sistema muestra a los alumnos del tutor. 2. El tutor selecciona un alumno. 3. El sistema muestra al alumno. 4. El tutor selecciona eliminar un alumno. 5. El sistema muestra un mensaje de advertencia. 6. El tutor acepta el mensaje de advertencia. 7. El sistema elimina al alumno y muestra la sección de alumnos. 8. El sistema notifica al tutor del alumno.
Extensiones	6.a. El centro no acepta el mensaje de advertencia. 6.a.1. Volvemos al paso 2.

Tabla 23. CU12. Eliminar un alumno.

Nombre	CU13 – Modificar los datos de un alumno.
Actor principal	Tutor.
Precondiciones	El tutor está identificado en el sistema.
Postcondiciones	Se guardan los datos modificados del alumno.

Escenario principal de éxito	1. El tutor accede a su sección de alumnos. 2. El sistema muestra a los alumnos del tutor. 2. El tutor selecciona un alumno. 3. El sistema muestra al alumno. 4. El tutor modifica los datos del alumno. 5. El sistema muestra un mensaje de advertencia. 6. El tutor acepta el mensaje de advertencia. 7. El sistema guarda los datos modificados y muestra la sección de alumnos.
Extensiones	6.a. El centro no acepta el mensaje de advertencia. 6.a.1. Volvemos al paso 2.

Tabla 24. CU13. Modificar los datos de un alumno.

Nombre	CU14 – Inscribir un alumno al grupo de una actividad.
Actor principal	Tutor.
Precondiciones	El tutor está identificado en el sistema. El alumno existe en el sistema. El calendario de su centro educativo existe en el sistema. El grupo existe en el sistema.
Postcondiciones	El alumno queda inscrito en el grupo y el número de alumnos del grupo se aumenta en uno.
Escenario principal de éxito	1. El tutor accede al calendario de su centro educativo. 2. El sistema muestra el calendario y sus grupos disponibles. 3. El tutor selecciona uno de los grupos. 4. El sistema muestra a los alumnos disponibles del tutor. 5. El tutor selecciona a un alumno. 6. El sistema comprueba que el curso del alumno está permitido en la actividad. 7. El sistema asigna al alumno al grupo y lo almacena en el sistema. 8. El sistema notifica al tutor del alumno.
Extensiones	4.a. Si el grupo está completo (max. alumnos), el sistema muestra una advertencia y volvemos al paso 2. 7.a. El tutor quiere eliminar al alumno del grupo. 7.a.1. El tutor sigue los pasos 1, 2 y 3. 7.a.2. El tutor elimina al alumno del grupo. 7.a.3. El sistema actualiza el grupo eliminando al alumno.

Tabla 25. CU14. Inscribir un alumno al grupo de una actividad.

Nombre	CU15 – Consultar los grupos a los que está inscrito un alumno.
Actor principal	Tutor.
Precondiciones	El tutor está identificado en el sistema. El alumno existe en el sistema. El grupo existe en el sistema.
Postcondiciones	Se muestra en pantalla la información solicitada.

Escenario principal de éxito	1. El tutor accede a su sección de alumnos. 2. El sistema muestra los alumnos del tutor. 3. El tutor selecciona uno de los alumnos. 4. El sistema muestra los datos del alumno y los grupos a los que está inscritos.
Extensiones	-

Tabla 26. CU15. Consultar los grupos a los que está inscrito un alumno.

Nombre	CU16 – Dar de alta una actividad.
Actor principal	Proveedor.
Precondiciones	El proveedor está identificado en el sistema.
Postcondiciones	Se crea la actividad en el sistema.
Escenario principal de éxito	1. El proveedor accede a su sección de actividades. 2. El proveedor selecciona dar de alta una actividad. 3. El sistema muestra la pantalla de alta de actividad. 4. El proveedor introduce los datos de la actividad: nombre, mínimo y máximo de alumnos, precio y cursos. 5. El sistema muestra la actividad en la sección de actividades. 6. El sistema notifica al proveedor.
Extensiones	-

Tabla 27. CU16. Dar de alta una actividad.

Nombre	CU17 – Eliminar una actividad.
Actor principal	Proveedor.
Precondiciones	El proveedor está identificado en el sistema.
Postcondiciones	Se elimina la actividad del sistema.
Escenario principal de éxito	1. El proveedor accede a su sección de actividades. 2. El sistema muestra las actividades del proveedor. 2. El proveedor selecciona una actividad. 3. El sistema muestra la actividad. 4. El proveedor selecciona eliminar una actividad. 5. El sistema muestra un mensaje de advertencia. 6. El proveedor acepta el mensaje de advertencia. 7. El sistema elimina la actividad y muestra la sección de actividades. 7. El sistema notifica al proveedor.
Extensiones	6.a. El proveedor no acepta el mensaje de advertencia. 6.a.1. Volvemos al paso 2.

Tabla 28. CU17. Eliminar una actividad.

Nombre	CU18 – Modificar los datos de una actividad.
Actor principal	Proveedor
Precondiciones	El proveedor está identificado en el sistema.
Postcondiciones	Se guardan los datos modificados de la actividad.
Escenario principal de éxito	1. El proveedor accede a su sección de actividades. 2. El sistema muestra las actividades del proveedor. 2. El proveedor selecciona una actividad. 3. El sistema muestra la actividad. 4. El proveedor modifica los datos de la actividad. 5. El sistema muestra un mensaje de advertencia. 6. El proveedor acepta el mensaje de advertencia. 7. El sistema guarda los datos modificados y muestra la sección de actividades.
Extensiones	6.a. El centro no acepta el mensaje de advertencia. 6.a.1. Volvemos al paso 2.

Tabla 29. CU18. Modificar los datos de una actividad.

2.1.4. Diagramas de casos de uso

En la ilustración 3, podemos ver los casos de uso comunes para los tres tipos de usuario: centro educativo, tutor y proveedor. Estos casos están relacionados con la gestión de sus cuentas de usuario; registrar una nueva cuenta, identificarse en el sistema, modificar datos de la cuenta y salir del sistema.

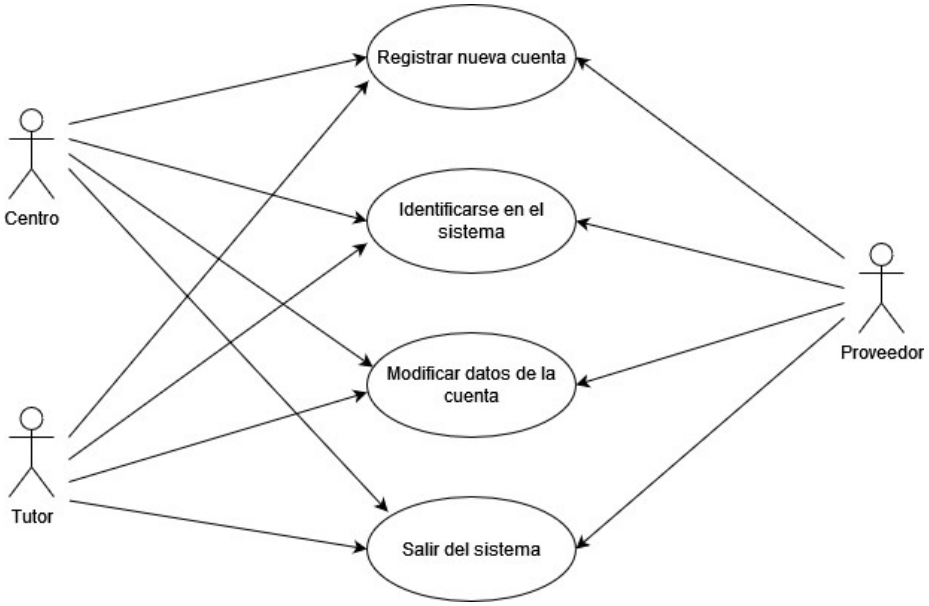

Ilustración 3. Casos de uso "usuarios".

En la ilustración 4 aparecen los casos de uso del usuario centro educativo. Por una parte, tenemos los casos de uso asociados con la gestión de calendarios; crear, modificar y eliminar calendario. Por otra, las relativas a las actividades y los grupos; asignar actividades a calendario, eliminar un grupo y abrir/cerrar el plazo de inscripciones a los grupos.

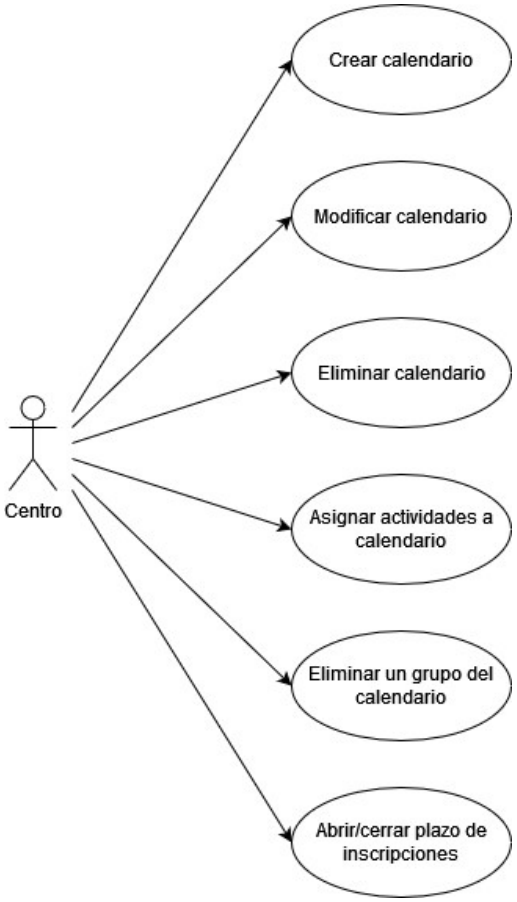

Ilustración 4. Casos de uso "centro educativo".

En la ilustración 5, se encuentran los casos de uso del usuario proveedor, centrados en la gestión de actividades; altas, modificación y eliminación.

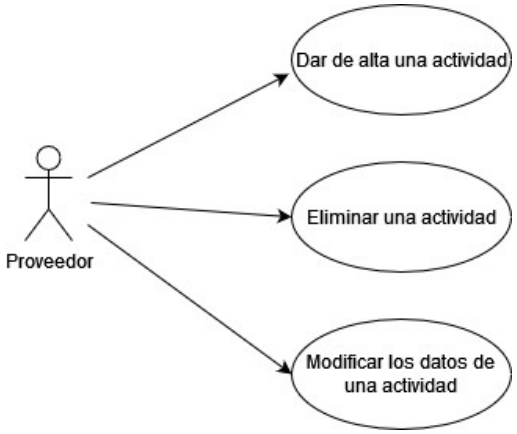

Ilustración 5. Casos de uso "proveedor".

Por último, en la ilustración 6, tenemos los casos de uso del usuario tutor, centrados en la gestión de alumnos y en su inscripción a los grupos de actividades.

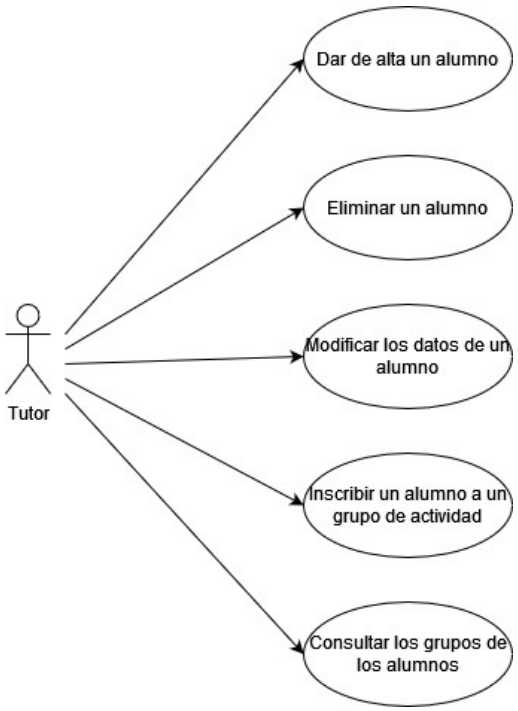

Ilustración 6. Casos de uso "tutor".

2.2. Diseño

2.2.1. Modelo entidad-relación

El diagrama del modelo entidad-relación de nuestro proyecto permite plasmar las entidades y atributos que hemos definido, y las relaciones que existen entre ellas, facilitando así su implementación en nuestra base de datos PostgreSQL. A continuación, se muestra este diagrama:

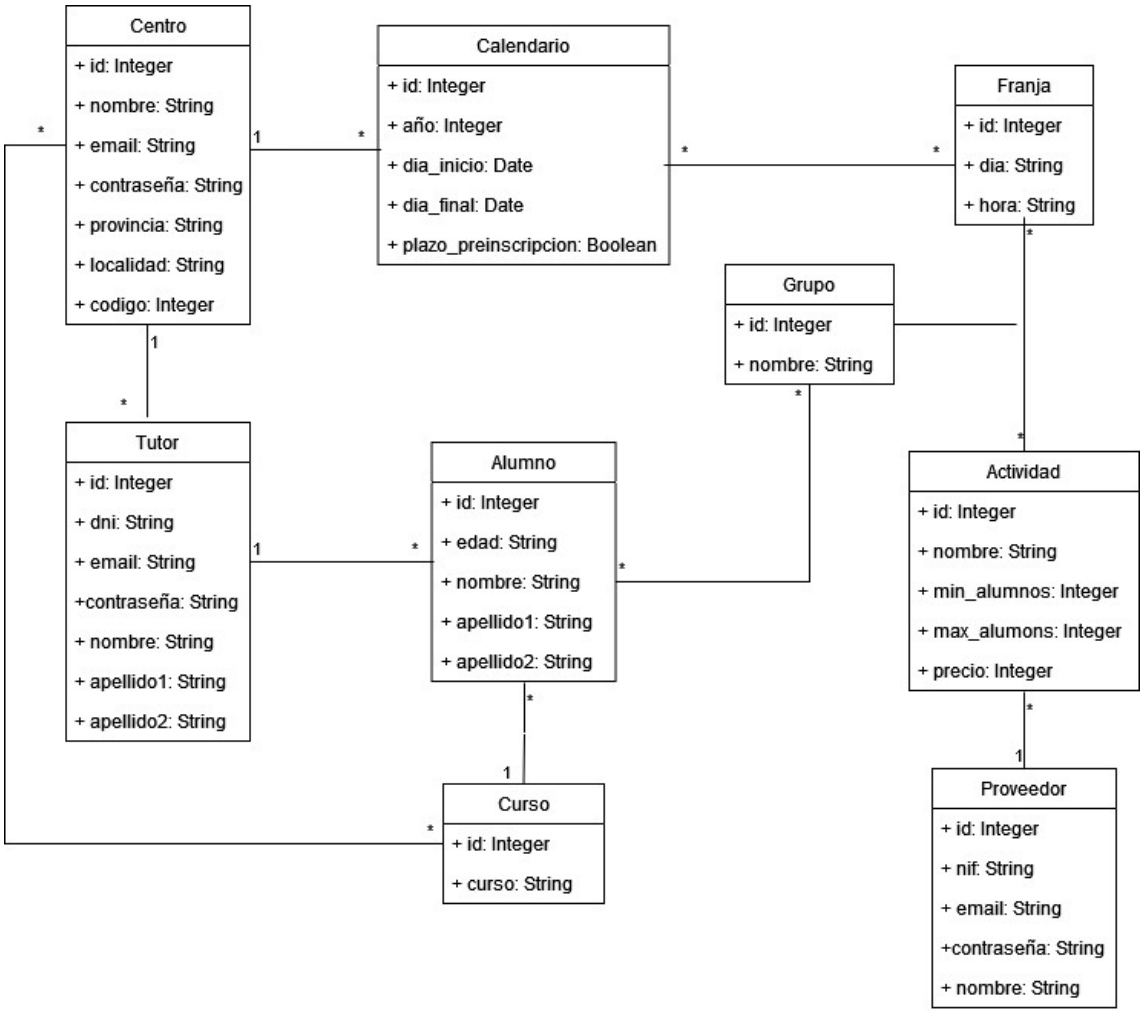

Ilustración 7. Modelo entidad-relación.

2.2.2. Tecnologías y recursos

En este apartado, vamos a presentar las tecnologías que vamos a usar en la implementación de nuestro proyecto, tanto en *backend* como en *frontend,* y en la fase de despliegue.

Django [1] es un *framework* web de alto nivel desarrollado en Python que permite el desarrollo rápido de sitios web seguros y mantenibles . Además, es gratuito y de código abierto. Django nos permite trabajar con cualquier *framework* del lado del cliente y con distintos formatos de contenido (HTML, XML, JSON, …), poniendo el foco en la seguridad y escalabilidad del sitio. Además, está escrito usando principios y patrones de diseño para fomentar la creación de código mantenible y reutilizable.
Django usa una arquitectura muy similar a la conocida MVC (*Model View Controller*), llamada MVT [2] (*Model View Template*).

- *Model*. Esta capa se encarga de toda la lógica referente a la estructura de los datos y su gestión en la base de datos.
- *View.* Las vistas reciben peticiones HTTP y devuelven respuestas HTTP. Acceden a los datos de los modelos (capa *Model*) requeridos por las peticiones y delegan el formateo de la respuesta a la plantilla (capa *Template*).
- *Template*. Las plantillas definen la estructura de la respuesta que se va a mostrar, normalmente en HTML. Parte del contenido se puede generar dinámicamente con los datos obtenidos de los modelos por las vistas.

Por último, Django nos ofrece un mapeador de URLs que nos permite conectar cada dirección web de nuestro sitio con una vista concreta.

PostgreSQL [3] es un potente sistema de gestión de bases de datos relacionales de código abierto con más de 35 años de desarrollo activo y conocido por su fiabilidad, robustez y alto rendimiento. PostgreSQL es uno de los sistemas de bases de datos soportado oficialmente por Django. Trabajar con ambos conlleva varios beneficios. Por ejemplo, Django tiene varios tipos de datos que solo funcionan en PostgreSQL, el cual ofrece un conjunto de características de las que solo se puede aprovechar Django.
Además, Pythoneverywhere, la plataforma donde alojaremos nuestro sitio web, ofrece un servicio de PostgreSQL totalmente gestionado que nos permitirá desplegar nuestra aplicación de la forma más sencilla y segura posible.

HTML [4] (*HyperText Markup Language*) es el lenguaje de marcado estándar que se usa para crear páginas web. Nos permite estructurar nuestra página con etiquetas que luego serán interpretadas por un navegador para su visualización. En este proyecto, usaremos HTML en nuestras plantillas para dar formato visual al contenido dinámico que nos proporcione Django a través de sus vistas.

CSS [5] (*Cascading Style Sheets*) es el lenguaje usado para dar estilo a un documento HTML u otros tipos de lenguaje de marcado. CSS nos permite crear páginas más atractivas y mejores interfaces de usuarios modificando cosas como tipos de fuente, colores, márgenes y un largo etcétera. Al igual que HTML, usaremos CSS en nuestras plantillas para mejorar su aspecto visual.

Bootstrap [6] es un *framework* de código abierto creado por Twitter para el desarrollo *frontend* de aplicaciones web. Contiene una colección de elementos prediseñados en HTML, CSS y Javascript que podemos modificar y adaptar a nuestro proyecto de forma rápida y sencilla. Además, está diseñado para que sus componentes se adapten a cualquier tipo de dispositivo.
Bootstrap nos va a servir para complementar nuestro código HTML y CSS propio.

Javascript [7] es un lenguaje de programación que se usa principalmente en el lado del cliente, aunque cada vez se extiende más uso en el lado del servidor. Normalmente se ejecuta dentro de un navegador web, permitiendo mejoras en la interfaz de usuario y páginas web más dinámicas. Además, nos va a permitir enviar y recibir información del backend de nuestro proyecto.

Pythonanywhere [8] es una solución PaaS (*Platform as a service*) basada en la nube que nos proporciona la infraestructura necesaria para poder desplegar nuestras aplicaciones Python en línea. Además del alojamiento, también dispone de un entorno de desarrollo (IDE) online basado en un editor web.
El uso de Pythonanywhere nos va a posibilitar tener una versión *live* de nuestro proyecto que podremos usar de forma *online.*

2.2.3. Arquitectura

2.2.3.1. Backend

Django basa su arquitectura en el patrón MVT, que es muy similar al conocido patrón MVC. La principal diferencia entre ambos es que el propio Django se encarga del trabajo realizado por el componente *Controller* en MVC, a través de sus *Templates*. Éstas contienen una mezcla de HTML y DTL [9] (*Django Template Language*) que veremos más adelante.

Vamos a ver los tres componentes que forma el patrón MVT:

- Componente Model

Es el encargado de la estructura y comportamiento de los datos de la aplicación además de la conexión con la base de datos, en nuestro caso, nuestro servidor de PostgreSQL. Cada modelo que creemos se convertirá en una única tabla de la base de datos, y los atributos del modelo serán los campos de esa tabla. Una vez conectados a la base de datos, podremos acceder a los registros de las distintas tablas y manipularlos con las diferentes operaciones DML [10] (*Data Manipulation Language*).

```
class Alumno(models.Model):
    nombre = models.CharField(max_length=20)
    apellido1 = models.CharField(max_length=20)
    apellido2 = models.CharField(max_length=20)
    edad = models.PositiveSmallIntegerField
    tutor = models.ForeignKey(Tutor, on_delete=models.CASCADE)
    curso = models.ForeignKey(Curso, on_delete=models.CASCADE)
```

Ilustración 8. Ejemplo de modelo.

Los modelos de datos de Django se almacenan en un fichero llamado **models.py** y cada modelo será una subclase de la clase **Django.db.models.Model**. Para poder usar los modelos en el resto de la aplicación, tenemos que editar el fichero **settings.py** y añadir su nombre en la sección **INSTALLED_APPS**.

En nuestro proyecto, el servidor de bases de datos elegido es PostgreSQL. Este sistema gestor de bases de datos es soportado tanto por Django como por Pythonanywhere.

Por último, cada vez que creamos o modificamos algún modelo, es necesario propagar los cambios de Django a la base de datos. Este proceso, llamado migración [11], consta de dos pasos. Primero, con el comando "**manage.py makemigrations**" creamos un fichero cos los cambios que se han producido desde la última migración. A continuación, con el comando "**manage.py migrate**", hacemos efectiva la modificación en la base de datos.

- Componente View

Este componente va a contener toda la lógica de nuestra aplicación. *View* va a interactuar con el componente *Model* para obtener los datos necesarios que luego va a usar para modificar las *templates* que va a presentar al usuario. Además, va a ser el encargado de recibir las peticiones web y devolver las respuestas correspondientes. Al igual que en *Model,* este componente se almacena en un fichero concreto, **view.py**.

Una vez hayamos creado toda nuestra lógica en **view.py**, necesitamos mapear cada una de las páginas de nuestra aplicación web con su funcionalidad correspondiente. Esto lo haremos en el fichero **urls.py**, donde enlazaremos cada URL de nuestro proyecto con la función a ejecutar.

```
urlpatterns = [
    path('', views.index, name='index'),
    path('admin/', admin.site.urls),
]
```
Ilustración 9. Ejemplo de mapeo de URLs.

- Componente Template

Esta capa es la encargada de gestionar la *interface* de usuario de la aplicación. Cada *template* va a ser un archivo HTML que contendrá contenido tanto estático como dinámico. Este contenido dinámico viene generado por la capa *View* al tratar los datos obtenidos desde la capa *Model*.

Para insertar este contenido entre el HTML, Django usará el lenguaje DTL. Haciendo uso de su sintaxis, no solo podemos mostrar el valor de variables dinámicas, si no crear condiciones o bucles.

```
{% if alumnos %}
    <ul>
    {% for alumno in alumnos %}
        <li><a>{{ alumno.nombre }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No hay alumnos.</p>
{% endif %}
```
Ilustración 10. Ejemplo de lenguaje DTL.

Para que nuestra aplicación encuentre nuestras *templates*, tenemos que guardarlas todas en el mismo directorio y configurarlo en el archivo **settings.py.**

A continuación, se muestra un mapa de la arquitectura:

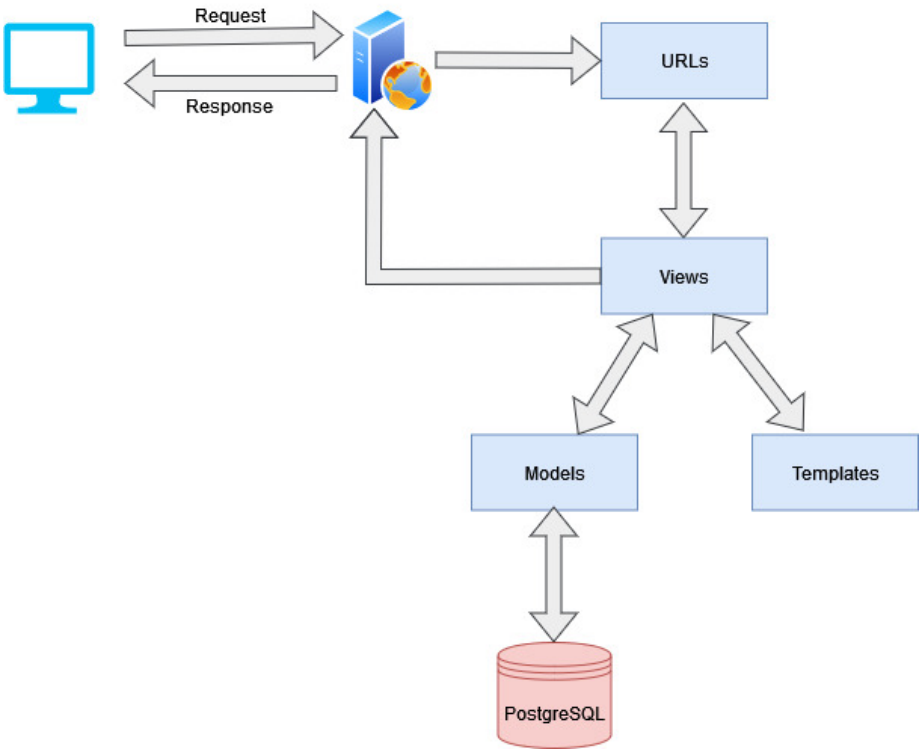

Ilustración 11. Mapa de la arquitectura

2.2.2.2. Frontend

Para el *frontend*, nos basaremos en las *templates* de Django. Estas plantillas son archivos de HTML con los que daremos formato a los datos estáticos y dinámicos que nos proporcione la lógica de la aplicación.
Para enriquecer y mejorar el aspecto visual de nuestras *templates* usaremos CSS y algunos de los componentes prediseñados que nos ofrece Bootstrap en su biblioteca, como, por ejemplo, elementos de formulario o botones.

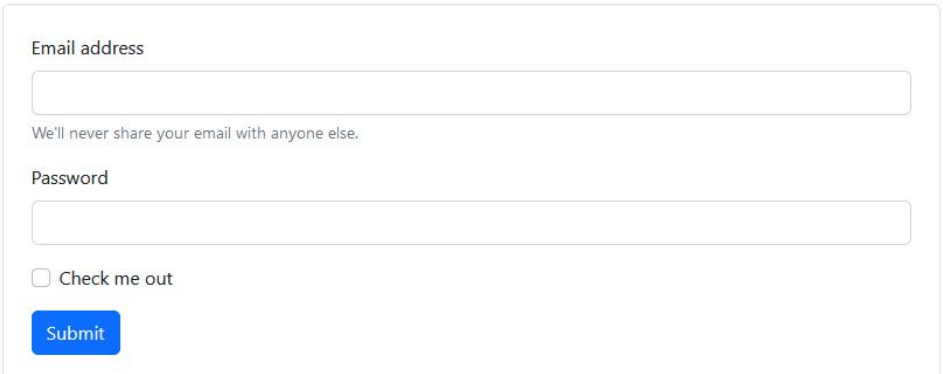

Ilustración 12. Formulario con Bootstrap.

Ilustración 13. Botones con Bootstrap.

Además, usaremos el lenguaje Javascript para añadir interactividad a alguna de nuestras páginas y conseguir una comunicación con nuestro *backend* para enviar y recibir datos relevantes a nuestros modelos.

2.2.3. Prototipos de pantalla

Para representar algunos de los conceptos que queremos plasmar en la interface de nuestra aplicación, hemos optado por elaborar unos *wireframes* de baja fidelidad de las pantallas más importantes de nuestra aplicación web. Para ello, hemos usado la herramienta online Balsamiq Cloud (https://balsamiq.cloud).

2.2.3.1. Página de Inicio

En la página de inicio, tenemos un menú superior donde podemos acceder a los apartados de los distintos usuarios: centros, tutores y proveedores. El resto de la página estará compuesta de imágenes y textos que informarán sobre las funciones y beneficios que tiene nuestra aplicación.

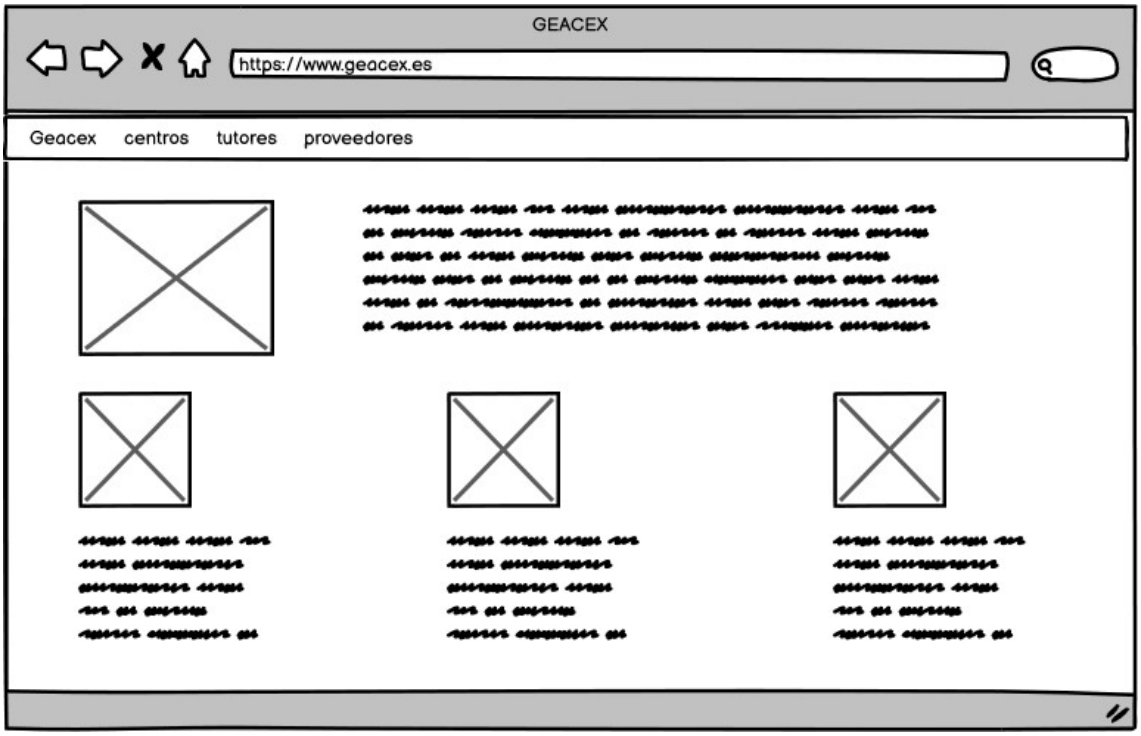

Ilustración 14. Pantalla inicio.

2.2.3.2. Registro

La página de registro contiene un formulario con los campos necesario para que el usuario pueda darse de alta en el sistema. Cada formulario se adaptará al tipo de usuario que quiera registrarse.

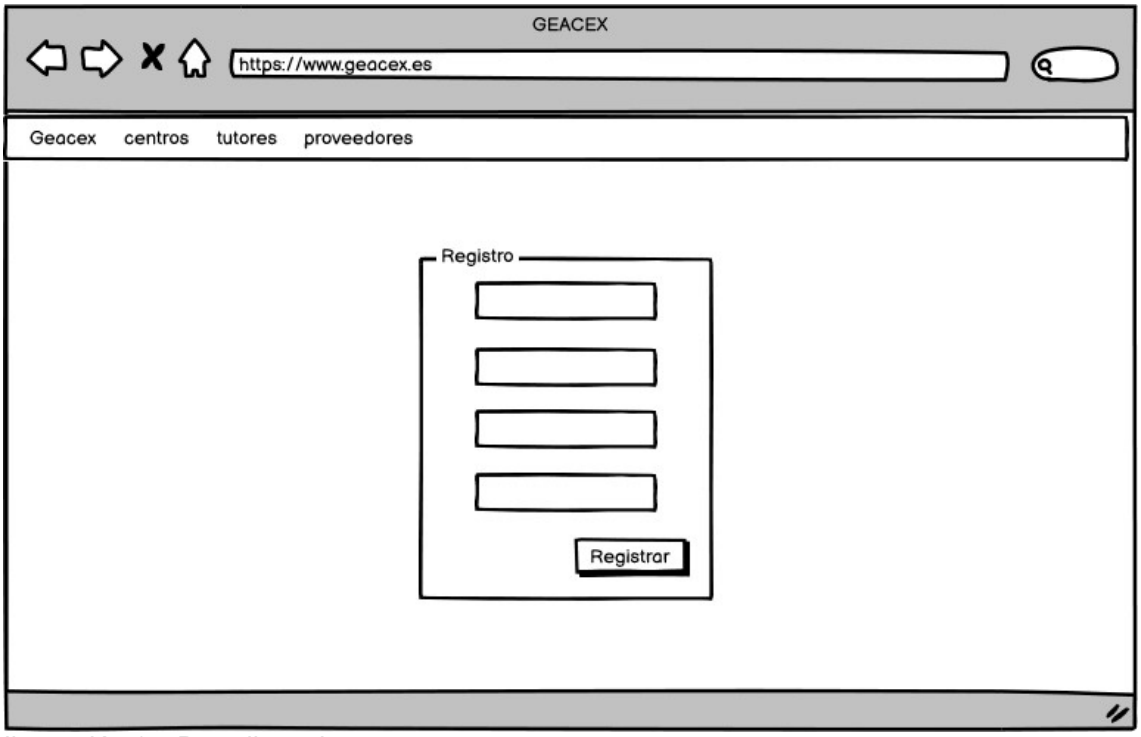

Ilustración 15. Pantalla registro.

2.2.3.3. Login

La página de *login* solo contendrá un formulario donde se pide el usuario y la contraseña.

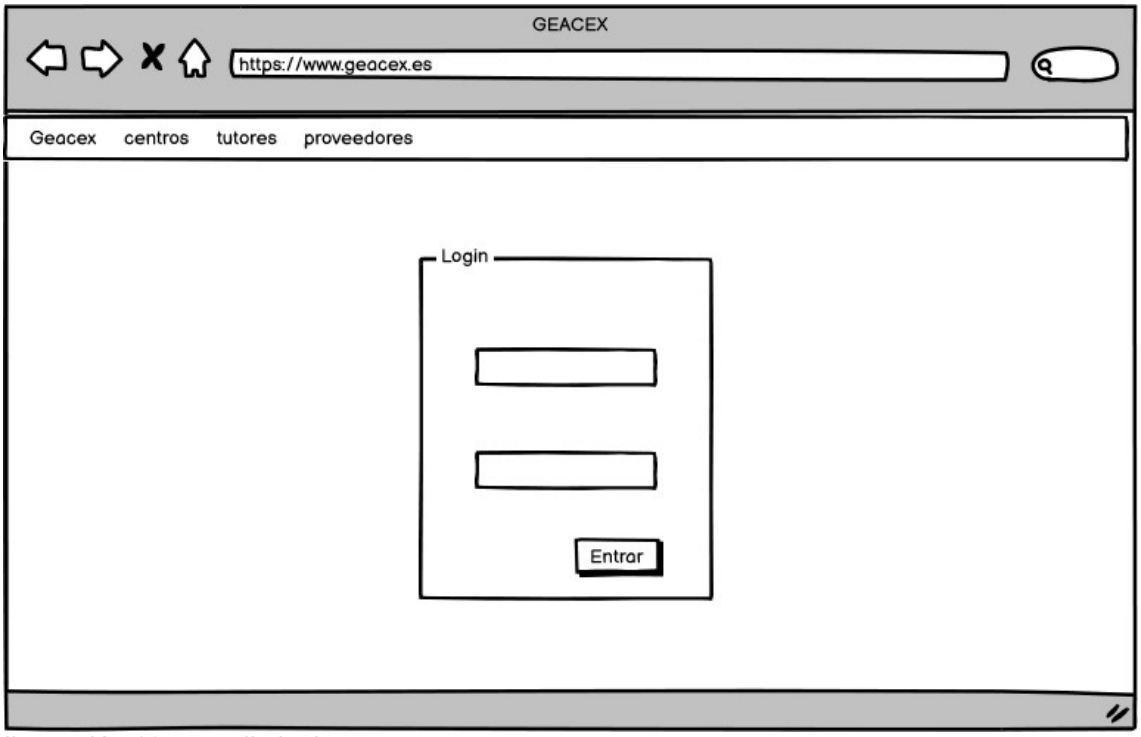

Ilustración 16. pantalla login.

2.2.3.4. Zona privada

La zona privada será similar para los distintos usuarios, con un menú superior donde se podrá acceder a las distintas secciones de su cuenta y una parte central donde se mostrará la información requerida.

Ilustración 17. Pantalla zona privada.

3. Implementación

3.1. Entorno de desarrollo

A continuación, vamos a explicar brevemente cómo hemos configurado el entorno de trabajo para el desarrollo del proyecto.

3.1.1 IDE

El IDE escogido ha sido Visual Studio Code, desarrollado por Microsoft. Es una herramienta gratuita y de código libre que se encuentra entre las más populares del mercado. Permite el desarrollo de código en una amplia variedad de lenguajes, incluidos Python, HTML o CSS. Una de sus principales características es el uso de extensiones para ampliar sus funcionalidades, algunas de las cuales han sido de mucha utilidad para la implementación de este proyecto.

3.1.2. Tecnologías utilizadas

-Python 3.10: Es el lenguaje de programación que principalmente usaremos en nuestra aplicación y la base del *framework* Django. Podemos descargarlo e instalarlo desde su página web oficial https://www.python.org/.

-Django 4.1.3: *Framework* de alto nivel basado en Python para el desarrollo de aplicaciones web. La instalación de Django se realiza desde pip, el instalador de paquetes de Python.

```
pip install django
```

-PostgreSQL 15.0: Sistema de gestión de bases de datos relacionales de código abierto. Es soportado oficialmente por Django y saca provecho de la mayoría de sus funcionalidades.

La instalación se ha hecho en la misma máquina local donde se encuentra el código de la aplicación y se ha usado la herramienta pgAdmin 4 para su gestión. Se puede descargar en https://www.postgresql.org/.

3.1.3 Librerías

Dentro de Django, hemos usado distintas librerías para extender sus funcionalidades. Algunas de las más interesantes son:

-Psycopg2 [12]: Es el adaptador de bases de datos PostgreSQL para Python. Soporta muchos de los tipos de Python y los ajusta para encajar en los tipos de datos de PostgreSQL, permitiendo extenderlos y personalizarlos.
-Django-localflavor [13]: Es una colección de piezas de código útiles en determinados países o culturas. Permite localizar componentes en modelos y

formularios de Django. En nuestro caso, lo hemo usado para trabajar con campos como DNI, NIF o el listado de provincias de España.

-Django-crispy-forms [14] y crispy-bootstrap5: Crispy-forms permite gestionar los formularios en Django, añadiéndoles nuevas funcionalidades como el *layout* y el renderizado de los mismos. Junto a la librería crispy-bootstrap5, nos permite dar estilos Bootstrap a los formularios que creemos.

3.2. Estructura del proyecto

El hecho de trabajar con un proyecto de Django va a hacer que tengamos definida una estructura de directorios y archivos configurada para facilitar su desarrollo y ejecución. Vamos a verlo a continuación:

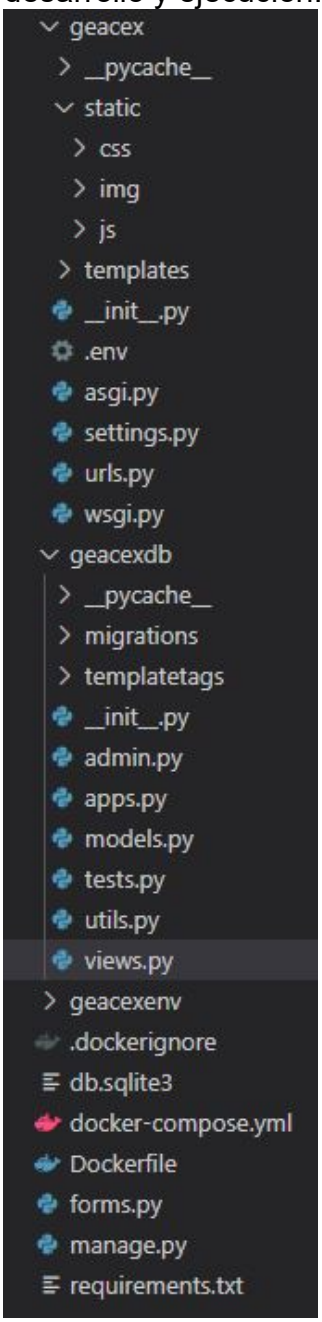

Ilustración 18. Estructura del proyecto.

-**static**: Desde este directorio, Django sirve ficheros estáticos a la aplicación web. En nuestro caso, tenemos la hoja de estilos CSS (\css), imágenes (\img) y código Javascript (\js). Para poder usar estos recursos en nuestras plantillas HTML, tendremos que cargarlos usando la etiqueta {% load static %}.

-**\templates**: Aquí almacenamos las plantillas HTML para nuestra aplicación. Estas plantillas contienen, por un lado, las partes estáticas de código HTML y, por otro; etiquetas DTL que definen como se insertará el contenido dinámico recibido desde el *backend* en las mismas. Además, hemos usado CSS y componentes Bootstrap para mejorar el diseño de la nuestra aplicación. Como parte del modelo MVT, las plantillas definen la capa de presentación.

-**.env**: En este archivo tenemos definidas variables de entorno que usaremos a lo largo de toda la aplicación. Esto nos permite no exponer de forma directa en el código datos como direcciones de servidores, usuarios o contraseñas. Un ejemplo de variable de entorno:

```
DB_HOST=127.0.0.1
```

-**Settings.py**: Aquí se guarda toda la configuración de nuestro proyecto, ya sea, ajustes de la base de datos, directorios de recursos o almacenamiento de logs.
-**Urls.py**: Guarda todas las asociaciones entre las URLs de nuestra aplicación web y la vista correspondiente que debe recoger la petición HTTP y devolver una respuesta, tal y como se define en el modelo MVT. Un ejemplo de estas asociaciones sería:

```
path('cambio_password', views.cambioPassword, name='cambiopassword'),
```

-**\geacexdb\migrations**: En este directorio tenemos las migraciones de Django. Estas migraciones son el mecanismo usado para propagar los cambios que hacemos a nuestros modelos a la base de datos. En Django, las migraciones se ejecutan en dos pasos. Primero, la creamos con el comando “manage.py makemigrations” y, a continuación, la aplicamos sobre la base de datos con “manage.py migrate”.

-**\geacexdb\templatetags**: Esta carpeta contiene etiquetas y filtros para nuestras plantillas que hemos creado específicamente para nuestro proyecto.

-**\geacexdb\models.py**: Aquí vamos a definir nuestros modelos de datos, especificando todos los campos de información que queramos que contengan. Normalmente, cada modelo se convertirá en una tabla de la base de datos. Dentro del patrón MVT, los modelos definen la capa de almacenamiento de datos.
Un ejemplo de modelo sería:

```
class Actividad(models.Model):

    nombre = models.CharField(max_length=20)
```

```
    min_alumnos = models.PositiveSmallIntegerField()
    max_alumnos = models.PositiveSmallIntegerField()
    precio = models.PositiveSmallIntegerField()
    proveedor = models.ForeignKey(Proveedor, on_delete=models.CASCADE)
    cursos = models.ManyToManyField(Curso)

    def __str__(self):
        cursos = " ".join(curso.nombre for curso in self.cursos.all())
            return  self.nombre  +  "  |  "  +  cursos  +  "  |  "  +
self.proveedor.usuario.username
```

-**\geacexdb\utils.py**: Este archivo contiene funciones o constantes definidas por nosotros y que usaremos en distintos puntos del proyecto.

-**\geacexdb\views.py**: Aquí guardamos las vistas de nuestro proyecto. Las vistas son las encargadas de recoger las peticiones web enviadas desde las URLs y devolver una respuesta web. Para construir esta respuesta, la vista puede necesitar acceder a la base de datos a través de nuestros modelos, para obtener cierta información. Dentro del patrón MVT, las vistas definen la capa de lógica de negocio de la aplicación

-**Forms.py**: En este archivo definimos los formularios [15] para nuestros modelos en Django. Los formularios HTML son un grupo de campos y *widgets* que permiten a los usuarios de nuestra aplicación introducir información y que esta sea enviada al servidor web para su procesamiento. Django nos proporciona un sistema propio para definir estos formularios de forma programática y asociarlos a nuestros modelos, además de la presentación y validación de los mismos. Un ejemplo de formulario en Django sería:

```
class CalendarioForm(ModelForm):

    class Meta:
        model = Calendario
                fields    =    ['nombre',    'dia_inicio',    'dia_final',
'plazo_inscripcion']
        widgets = {
            'dia_inicio': TextInput(attrs={'type': 'date'}),
            'dia_final': TextInput(attrs={'type': 'date'}),
        }

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.helper = FormHelper()
        self.helper.add_input(Submit('submit', 'Enviar'))
```

-**Requirements.txt**: Es un archivo de texto que contiene las librerías, módulos y paquetes Python usados en el proyecto. Estas dependencias son necesarias para el funcionamiento de la aplicación. Cuando despleguemos o ejecutemos

nuestra aplicación en un entorno nuevo, Python usará este archivo para instalarlas de forma automática.

-**Dockerfile**: Contiene la secuencia de instrucciones necesarias para la creación de una imagen Docker de la aplicación.

-**Docker-compose.yml**: Define los servicios, redes y volúmenes necesarios para una aplicación Docker.

3.3. Estructura y funcionamiento de la aplicación

En este apartado, vamos a ver algunos de los aspectos más relevantes del uso y estructura de nuestra aplicación web.

3.3.1. Estructura general de la aplicación

En la ilustración 19 se puede ver la estructura general de la aplicación. Desde la página de inicio, cada tipo de usuario (centro educativo, tutor o proveedor) puede acceder a su área privada. Estas áreas están aisladas unas de las otras y permiten que cada usuario gestione sus recursos de forma independiente.

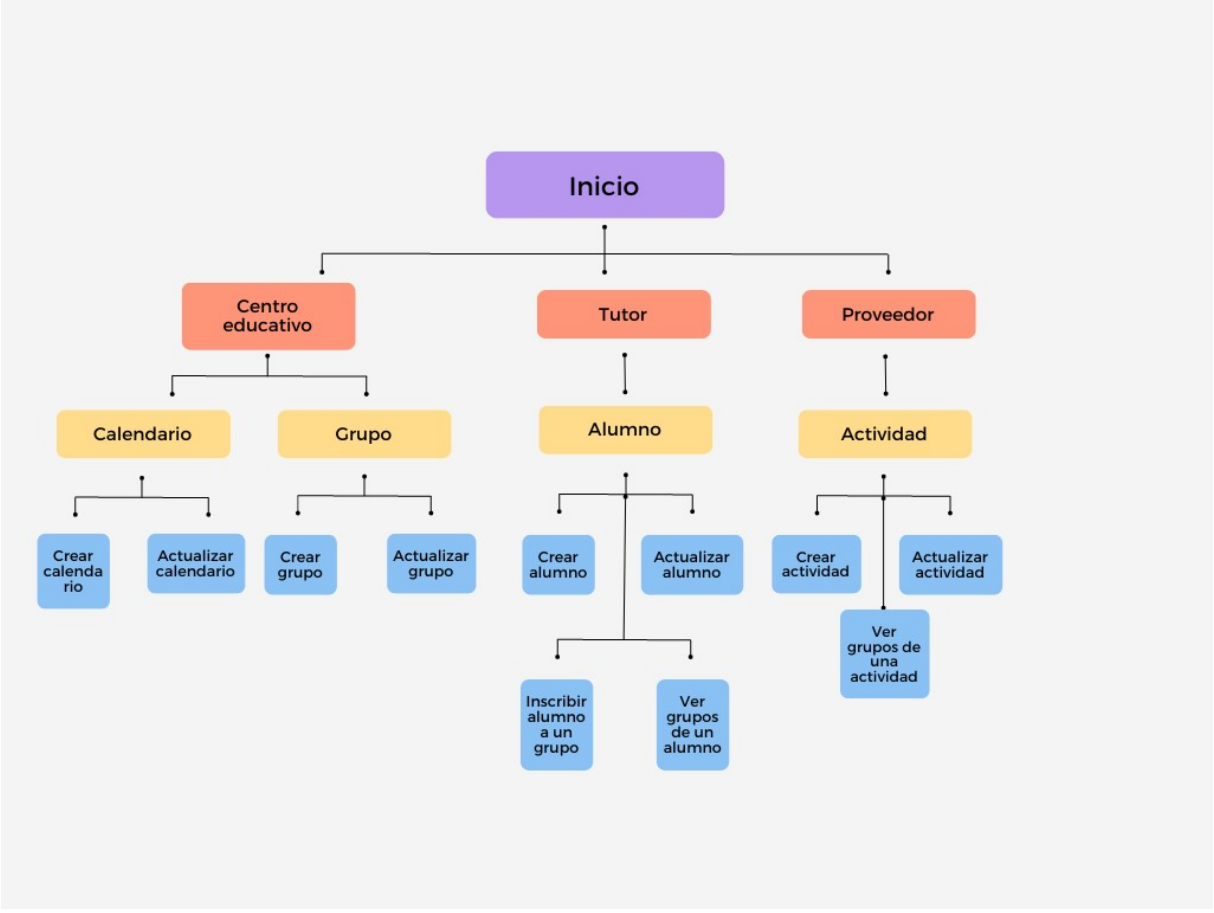

Ilustración 19. Estructura general de la aplicación.

3.3.2. Funcionamiento general de la aplicación

3.3.2.1. Gestión de usuarios

Como hemos comentado, la aplicación web es usada por tres tipos de usuarios: centros educativos, tutores y proveedores de actividades. Los usuarios comparten varias funcionalidades que se han adaptado a las características de cada uno. Desde el menú de la página de inicio, cada uno de ellos puede acceder a su página de acceso.

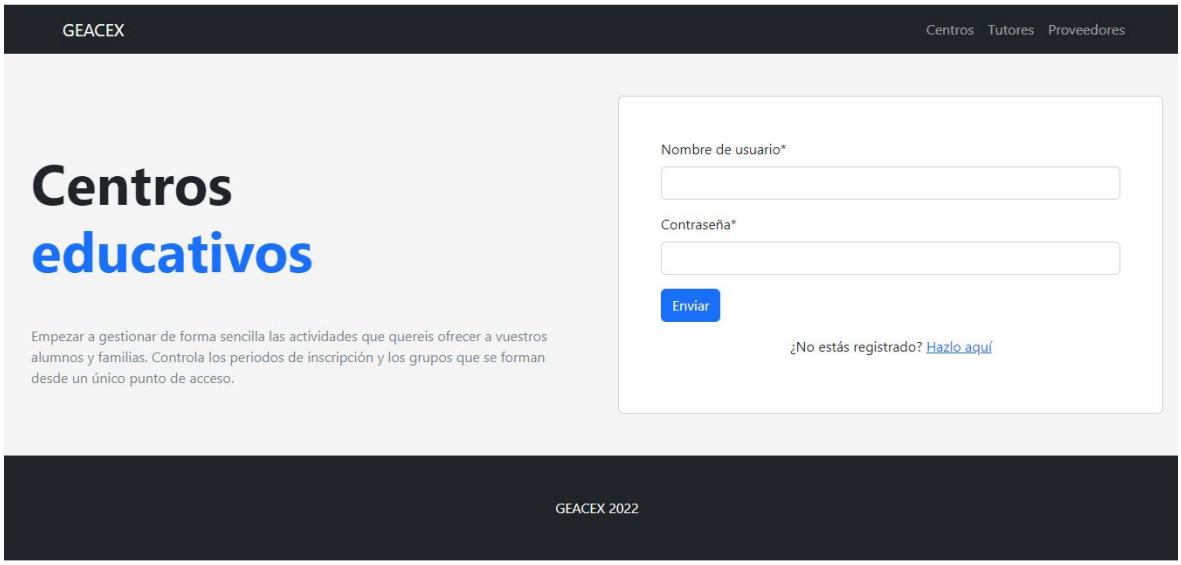

Ilustración 20. Página de login.

Si aún no estuvieran registrados, podrán hacerlo pinchando en el enlace correspondiente, el cual los llevará a su correspondiente formulario de registro.
En el caso concreto de los tutores, estos siempre irán asociados a un centro educativo. Para ello, cuando un centro se registra, se crea un código aleatorio alfanumérico. El centro puede consultar este código en su área y proporcionárselo a los tutores. El código le será requerido al tutor cuando se vaya a registrar y así quedará asociado a su centro educativo.

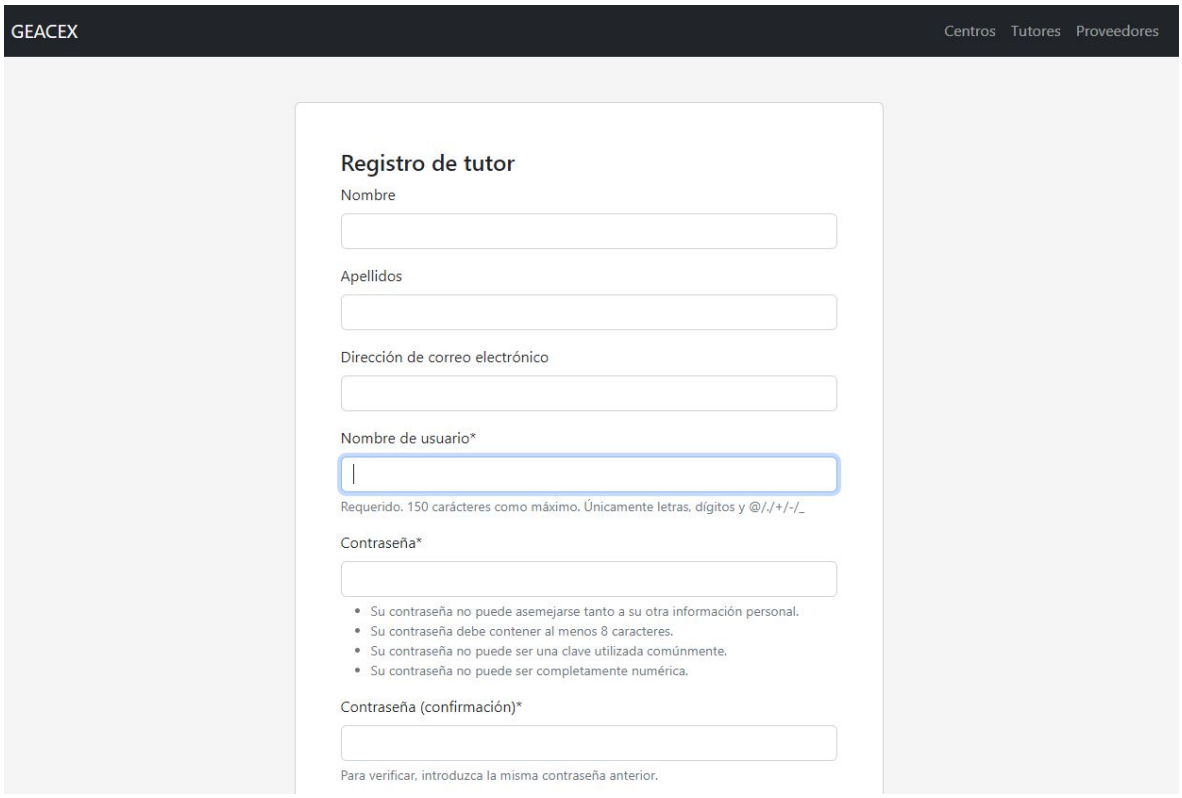

Ilustración 21. Formulario de registro.

Una vez el usuario haya realizado el login, tendrá acceso a su área privada. Dentro de ésta, los usuarios podrán actualizar sus datos siempre que lo deseen y realizar un cambio de contraseña. También podrán cerrar su sesión pinchando en el enlace que se encuentra a la derecha en el menú superior.

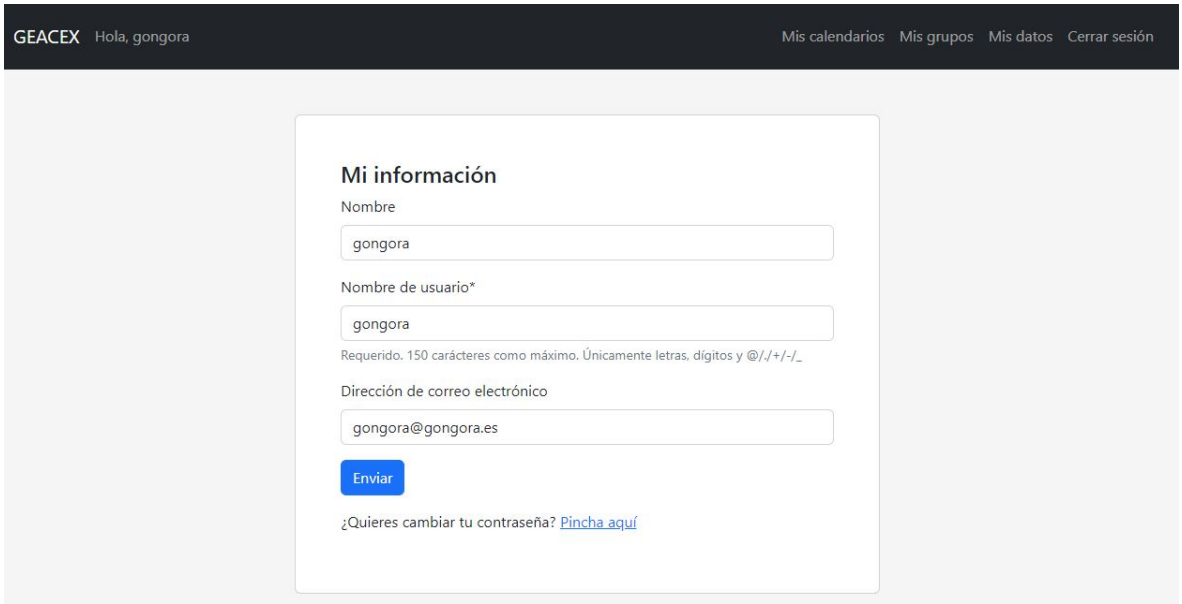

Ilustración 22. Formulario de actualización de datos.

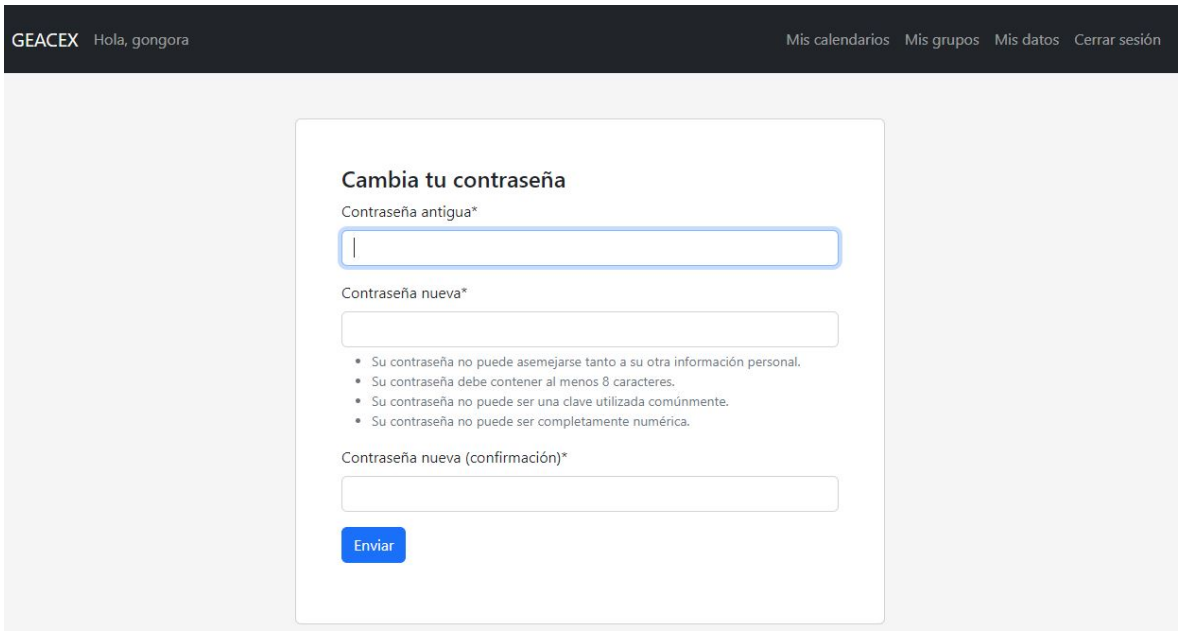

Ilustración 23. Formulario de cambio de contraseña.

En los siguientes subapartados, vamos a ver las funcionalidades propias de cada usuario.

3.3.2.2. Usuario proveedor

El usuario proveedor representa a las empresas, clubes, etc. que quieren dar de alta las actividades que organizan para que puedan se ofertadas en los centros educativos y los alumnos se puedan inscribir en ellas.
Cuando el proveedor accede a su área privada, podrá consultar un listado de las actividades que tiene dadas de alta en el sistema. Además, podrá modificarlas o borrarlas.

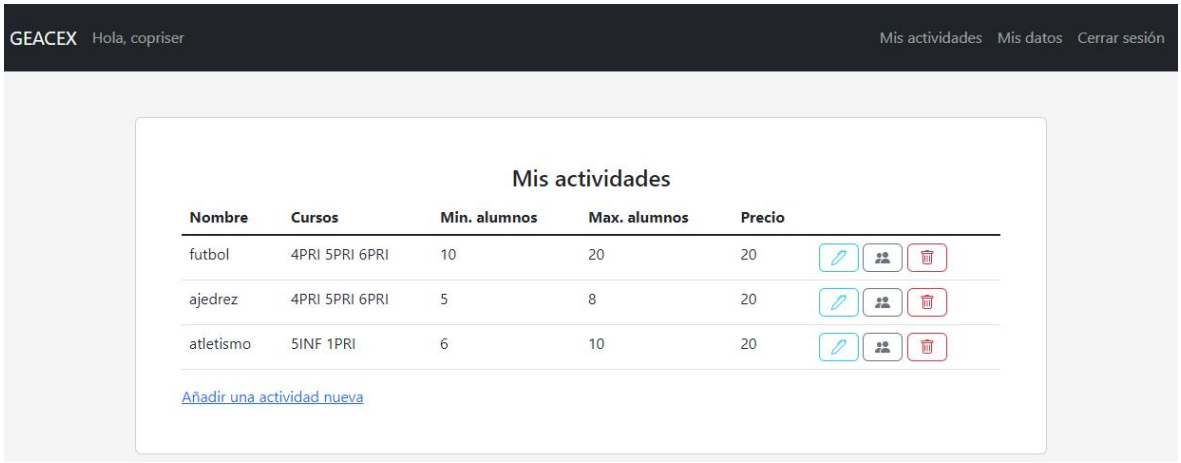

Ilustración 24. Listado de actividades.

Además, podrá consultar en qué centros, horarios y cuántos alumnos hay apuntados a cada una de sus actividades, obteniendo así toda la información relevante necesaria.

Ilustración 25. Grupos de una actividad.

3.3.2.3. Usuario centro educativo

Los centros educativos están interesados en conocer qué actividades les ofrecen los proveedores, para poder crear grupos en diferentes días y horarios y así permitir que los tutores los consulten y puedan apuntar a los alumnos. Además, los centros gestionarán diferentes calendarios para los distintos cursos escolares o periodos de vacaciones, donde podrán definir las fechas en que tienen lugar las actividades y los plazos de inscripción.

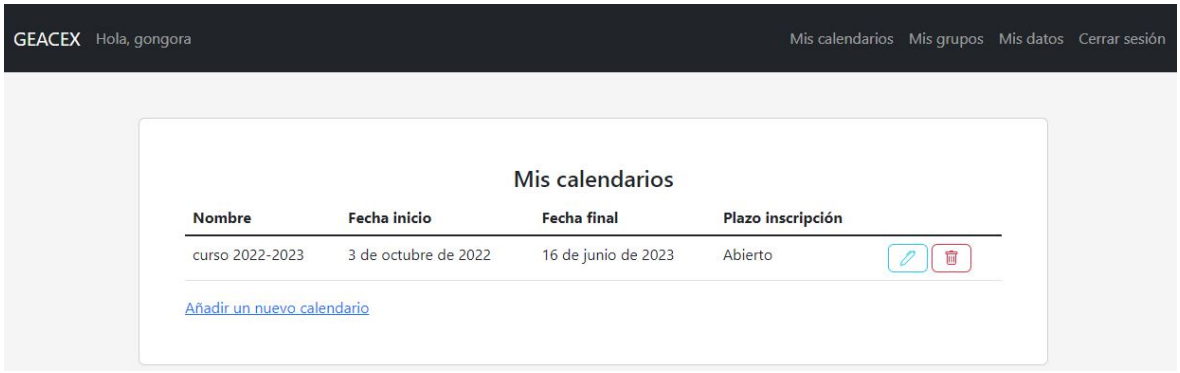

Ilustración 26. Calendarios de un centro educativo.

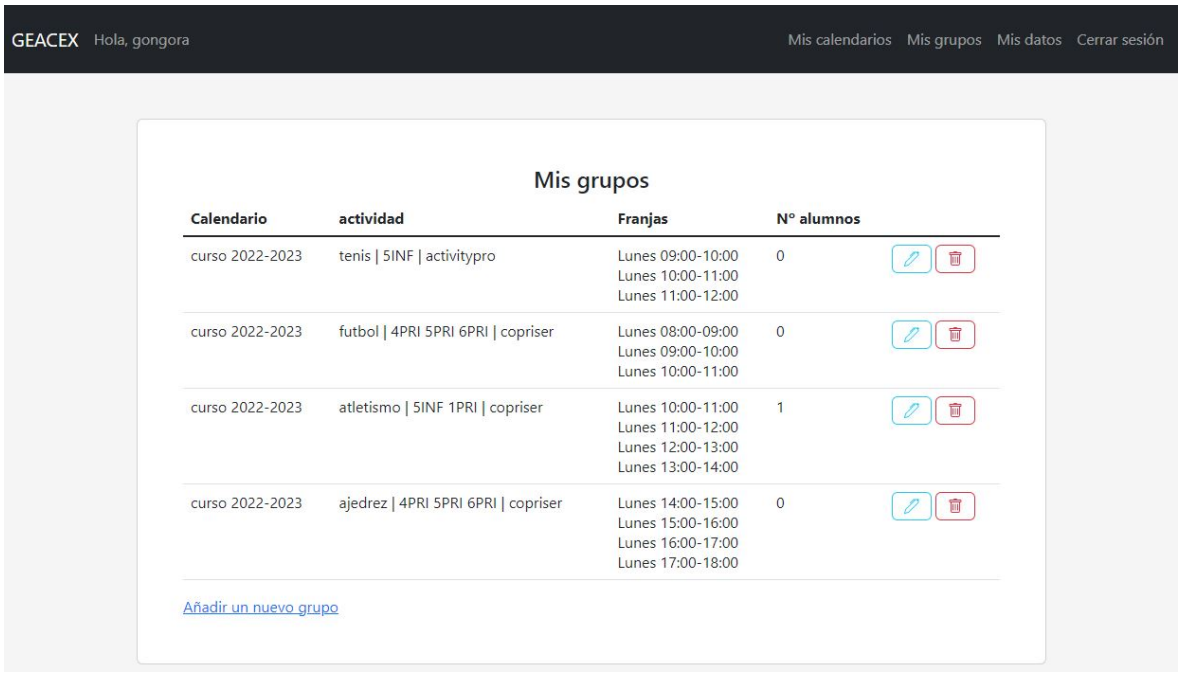

Ilustración 27. Grupos de un centro educativo.

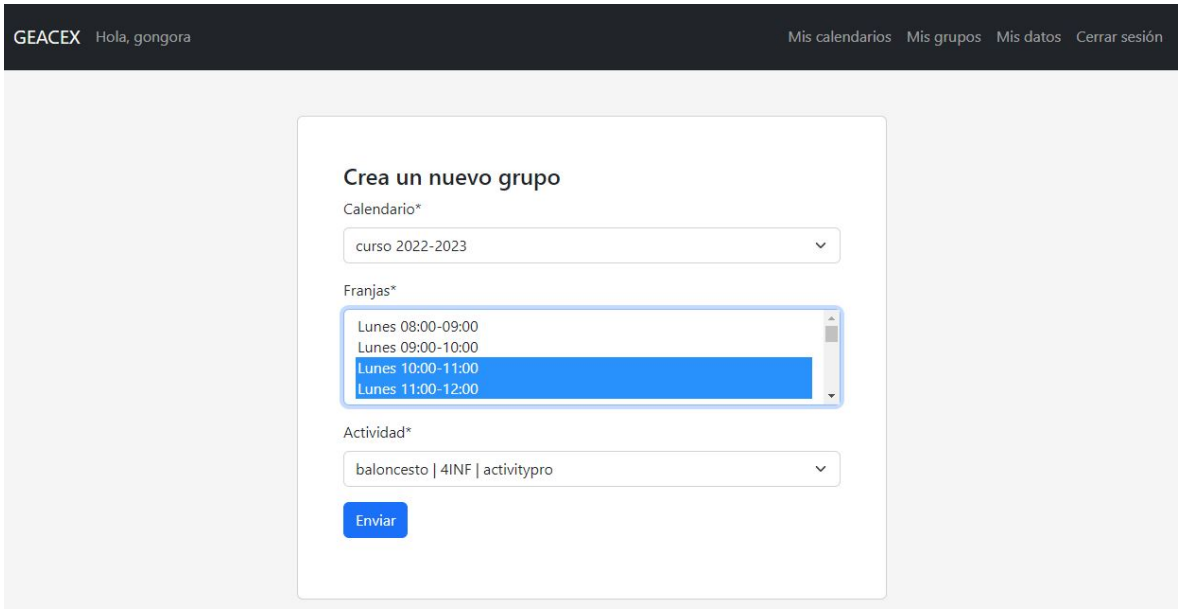

Ilustración 28. Creación de un nuevo grupo.

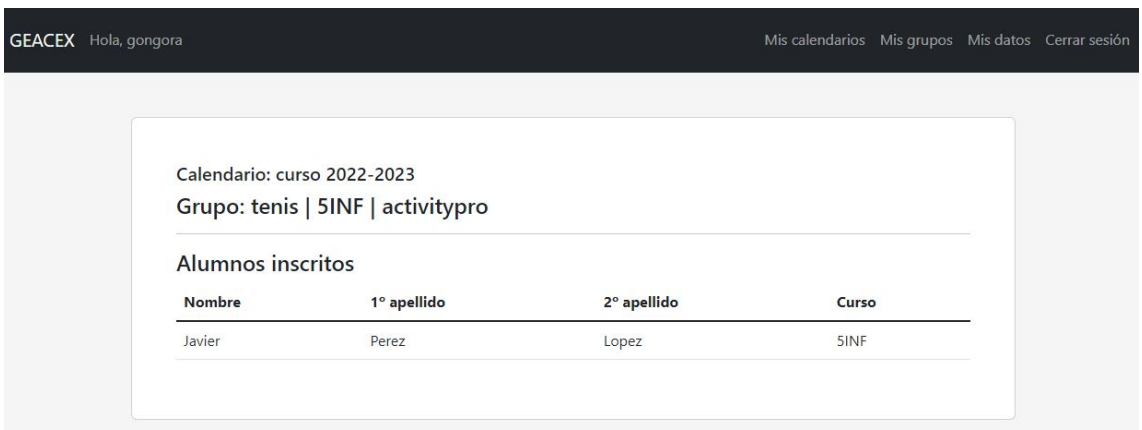

Ilustración 29. Alumnos suscritos a un grupo.

3.3.2.4. Usuario tutor

Por último, los tutores podrán dar de alta a los alumnos en el sistema y, así, poder inscribirlos en las actividades ofrecidas en su centro educativo. A la hora de realizar la inscripción, las actividades aparecerán filtradas por centro y curso del alumno, además de si el plazo de inscripción está abierto o cerrado. Además, los alumnos podrán ser darse de baja del grupo de la actividad en cualquier momento.

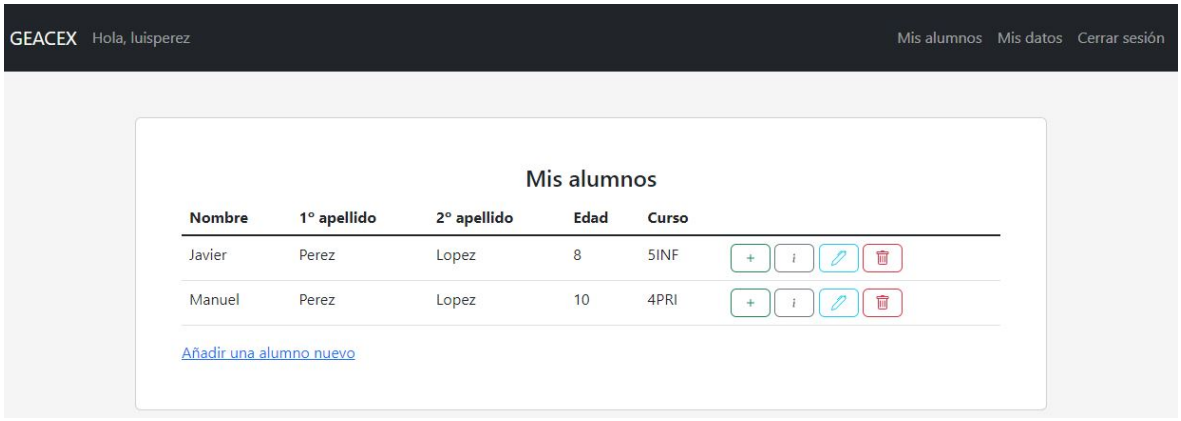

Ilustración 30. Alumnos de un tutor.

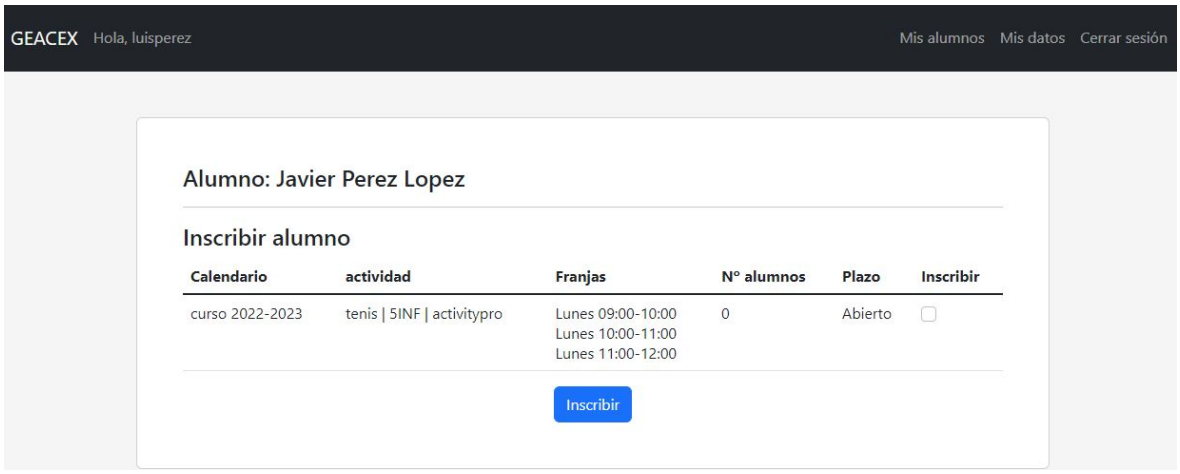

Ilustración 31. Inscripción de alumno en un grupo.

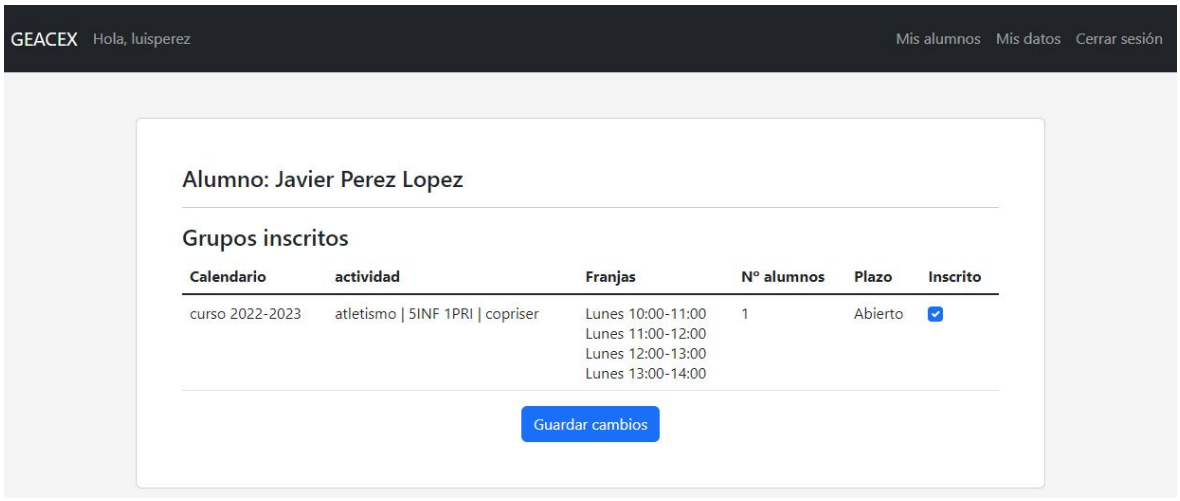

Ilustración 32. Información de grupos inscritos del alumno.

3.4. Ejecución en local con Docker

Para facilitar la distribución, ejecución y despliegue de nuestra aplicación, hemos optado por empaquetarla en una imagen *Docker* [16]. Una imagen *docker* contiene todo lo necesario (bibliotecas, código, herramientas) para que el software que contiene se pueda ejecutar en cualquier entorno.
Para poder ejecutar la imagen *docker* de nuestro proyecto, es necesario tener instalado el software Docker Compose, que se encuentra disponible para Windows, Mac y Linux. Las instrucciones para su instalación se encuentran en la página oficial de Docker https://docs.docker.com/compose/install/

Una vez instalado Docker, tenemos que abrir un terminal y situarnos en el directorio raíz del proyecto, al mismo nivel que el archivo Dockerfile, para poder seguir los siguientes pasos.

Primero, debemos construir la imagen Docker de nuestro proyecto siguiendo la secuencia de instrucciones que se encuentran en el archivo Dockerfile. Para ello, ejecutamos en el terminal el siguiente comando:

docker compose build

A continuación, tenemos que levantar el contenedor con nuestra imagen y sus servicios dependientes, en nuestro caso, la base de datos en PostgreSQL. Lo hacemos con el comando:

docker compose up -d

Por último, es necesario aplicar las migraciones que hayamos hecho, de forma que nuestros modelos de datos se creen PostgreSQL en forma de tablas. Este paso solo es necesario realizarlo una vez, aunque, si los modelos se modificaran, habría que volver a ejecutarlo:

docker compose exec web python manage.py migrate

Una vez hayamos ejecutado estos comandos con éxito, podemos acceder a la aplicación desde cualquier navegador web accediendo a la dirección http://localhost:8000**.**

3.5. Despliegue en línea

Para realizar el despliegue real de la aplicación, de forma que esta pueda ser accedida de forma online, se ha optado por usar la plataforma PythonAnywhere (https://www.pythonanywhere.com/). Este servicio está dedicado específicamente a alojar, ejecutar y desarrollar aplicaciones en Python, siendo compatible con muchos de sus *frameworks*, entre ellos, Django.

La aplicación se encuentra disponible en la siguiente dirección: https://mpascual.eu.pythonanywhere.com/

El registro de nuevo usuarios está abierto para cualquiera de los tres perfiles. Aun así, aquí están los datos para un usuario de cada perfil que ya están creados y se pueden usar para probar la aplicación.

Tutor:
Usuario: luisperez
Contraseña: Tutor_2022
Centro:
Usuario: velazquez
Contraseña: Centro_2022
Proveedor:
Usuario: activityorg
Contraseña: Proveedor_2022

3.6. Tests

Uno de los objetivos que queríamos conseguir en nuestra aplicación web es que fuese fácilmente accesible desde dispositivos móviles. Para comprobar si lo hemos logrado, vamos a pasar el test de optimización para móviles que nos ofrece Google en la web https://search.google.com/test/mobile-friendly.

Como vemos en la siguiente imagen, nuestra aplicación web está preparada y optimizada para ser usada en dispositivos móviles.

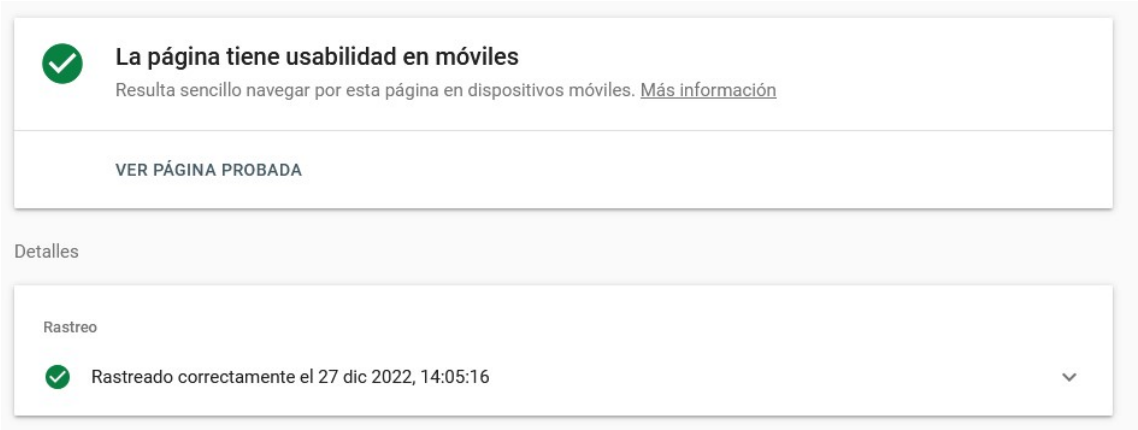

Ilustración 33. Resultado del test de optimización para móviles.

También vamos a realizar un test de carga *online* para ver el rendimiento que tiene nuestro sitio web. Para ello, usaremos la herramienta GTmetrix (https://gtmetrix.com/), que realiza un análisis en profundidad de distintos aspectos de nuestra página para otorgarle una puntuación final.

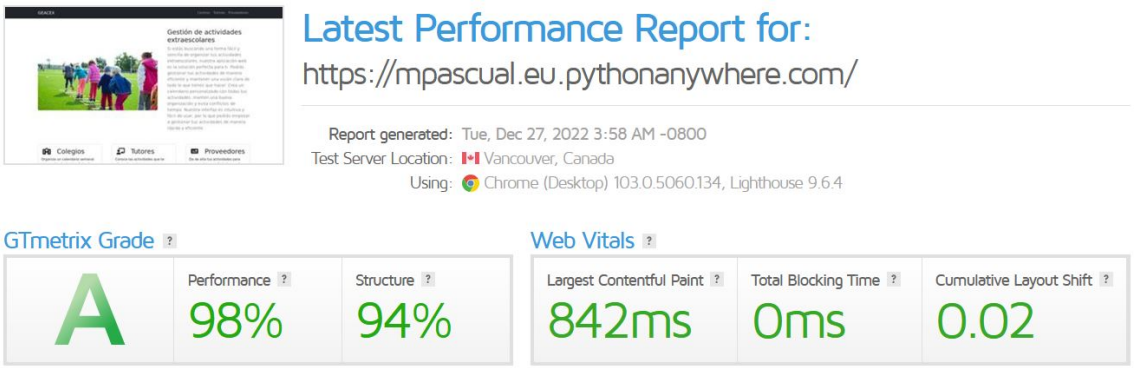

Ilustración 34. Resultado GTmetrix.

Como vemos, nuestra aplicación web ha obtenido un resultado final de A, con un 98% en rendimiento y 94% en estructura.

4. Conclusiones

La realización de este trabajo nos ha permitido conocer de primera mano el ciclo de vida de una aplicación web, desde la idea primigenia hasta su implementación y despliegue en el mundo real. También, hemos podido profundizar en algunas de las tecnologías actuales más usadas como Python, Django o Bootstrap. Debido a la falta de tiempo y conocimiento, no se ha podido usar JavaScript tanto como se había planeado, y queda como asignatura pendiente para futuras revisiones.

En cuanto a los objetivos planteados, la mayoría de ellos se han conseguido con éxito, obteniendo un producto con un conjunto de funcionalidades suficiente para satisfacer las necesidades del conjunto de usuarios inicial, permitiendo su ampliación o mejora de forma sencilla en siguientes iteraciones.

Respecto a la planificación, creemos que ha sido correcta, llegando a cumplir todos los hitos temporales sin problema. En cuanto a las subtareas de cada hito, aunque han servido como guía orientativa, en la práctica, muchas de ellas tenían lugar en paralelo o se han realizado en diferente orden. Si que ha sido necesario ajustar algunas tareas ya realizadas, sobre todo de la fase de análisis y diseño, para poder asegurar el éxito del proyecto.

Por último, el proyecto queda abierto a recibir mejoras en futuras líneas de trabajo. Entre otras, algunas nuevas funcionalidades a implementar serían:

- Sistema de notificaciones a usuarios vía email.
- Posibilidad de realizar los pagos mensuales de las actividades.
- Componente calendario más interactivo y dinámico.

5. Glosario

***Frontend*:** Es la parte de una aplicación web que los usuarios finales pueden ver e interactuar. Se encarga tanto de la interfaz como de la experiencia de usuario. Principalmente, se usan lenguajes como HTML, CSS y Javascript para su desarrollo y se ejecuta en el navegador del cliente.

***Backend**:* Es la parte que contiene toda la lógica de negocio en una aplicación o sitio web, de la cual se alimenta el *frontend*. Se almacena en el lado del servidor y suele estar desarrollada en lenguajes como Python, Java o Ruby.

IDE: Un entorno integrado de desarrollo, o *integrated development environment*, es un *software* para el diseño de aplicaciones que combina distintos tipos de herramientas dentro de un mismo interfaz. Estas herramientas pueden ser editores, depuradores, bibliotecas o *plugins.*

MVP: Producto mínimo viable o *mínimum viable product*, es un producto con las suficientes funcionalidades para satisfacer a un conjunto inicial de clientes, permitiendo obtener un *feedback* rápido para seguir mejorándolo y ampliándolo.

Framework: Es un entorno de trabajo ya desarrollado que proporciona una estructura y funcionalidades bases sobre las que implementar aplicaciones de más alto nivel.

URL: Localizador de recursos uniforme o *uniform resource locator,* es la dirección única que apunta a un recurso en la red. Estos recursos pueden ser una página web, una imagen o un GIF, por ejemplo.

Wireframe: Es una representación visual a bajo nivel de la estructura de una página web.

6. Bibliografía

[1] MDN Web Docs. Introducción a Django. https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction. Consultado en 10/2022.

[2] Ask Python. Django MVT Architecture. https://www.askpython.com/django/django-mvt-architecture. Consultado el 10/2022.

[3] PostgreSQL.org. What is PostgreSQL? https://www.postgresql.org/about/. Consultado el 10/2022.

[4] MDN Web Docs. HTML: Lenguaje de etiquetas de hipertexto. https://developer.mozilla.org/es/docs/Web/HTML. Consultado el 10/2022.

[5] MDN Web Docs. CSS. https://developer.mozilla.org/es/docs/Web/CSS. Consultado el 10/2022.

[6] GetBootstrap. Build fast, responsive sites with Bootstrap. https://getbootstrap.com/. Consultado el 10/2022.

[7] MDN Web Docs. Javascript. https://developer.mozilla.org/es/docs/Web/JavaScript. Consultado el 10/2022.

[8] Pythonanywhere. Host, run, and code Python in the cloud. https://www.pythonanywhere.com/. Consultado el 12/2022.

[9] Django Project. The Django template language. https://docs.djangoproject.com/en/4.1/ref/templates/language/. Consultado el 10/2022.

[10] Techopedia (13-10-2014). Data Manipulation Language (DML). https://www.techopedia.com/definition/1179/data-manipulation-language-dml. Consultado el 10/2022.

[11] Hepper, Daniel (09-01-2019). Django Migrations: A Primer. *Real Python.* https://realpython.com/django-migrations-a-primer/. Consultado el 11/2022.

[12] Psycopg.org. Psycopg – PostgreSQL database adapter for Python. https://www.psycopg.org/docs/. Consultado el 11/2022.

[13] django-localflavor.readthedocs.io. The “local flavor” app. https://django-localflavor.readthedocs.io/en/latest/. Consultado el 11/2022.

[14] Freitas, Vitor (28-11-2018). Advanced Form Rendering with Django Crispy Forms. *Simpleisbetterthancomplex*. https://simpleisbetterthancomplex.com/tutorial/2018/11/28/advanced-form-rendering-with-django-crispy-forms.html. Consultado el 11/2022.

[15] MDN Web Docs. Django Tutorial Part 9: Working with forms. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Forms. Consultado el 11/2022.

[16] Vincent, Will (08-12-2022). Django, Docker, and PostgreSQL Tutorial. *LearnDjango*. https://learndjango.com/tutorials/django-docker-and-postgresql-tutorial. Consultado el 12/2022.

7. Anexos

7.1. Anexo I. Diagrama de Gantt.

Se entrega en un fichero adjunto llamado “Anexo I. Diagrama de Gantt.pdf” para su correcta visualización.